HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Bearbeitungsbeginn: 01.09.2018

Vorgelegt am: 31.05.2019

Thesis

zur Erlangung des Grades

Master of Science

im Studiengang Medieninformatik

an der Fakultät Digitale Medien

Lukas Scheuerle

Matrikelnummer: 257396

Entwurf und Entwicklung eines Vektorgrafik- und Animationseditors auf Basis von Standard-Webtechnologien zur Einbindung in den Furtwangen University Didactic Game Editor (FUDGE)

Erstbetreuer: *Prof. Jirka Dell'Oro-Friedl*

Zweitbetreuer: *Prof. Thomas Krach*

Abstract

Es gibt viele verschiedene Spiele-Engines auf dem Markt, die meisten davon sind kommerziell. Viele Engines machen dabei einige Dinge sehr ähnlich, in anderen Dingen unterscheiden sie sich gravierend. Oftmals muss die Engine verlassen werden, um Assets zu erstellen. Auch sind die wenigsten dieser Engines auf die Lehre zugeschnitten, sondern fokussieren sich auf die professionelle und kommerzielle Anwendung. Aufgrund all dieser Faktoren ist es oftmals umständlich oder in der Lehre ungeeignet, diese Engines zu nutzen.

Bei all diesen Problemen soll FUDGE (Furtwangen University Didactic Game Editor) Abhilfe schaffen. FUDGE ist ein speziell auf die Lehre zugeschnittener, auf Webtechnologien basierender und damit Cross-Plattform kompatibler Spiele-Editor. Er soll die grundlegendsten Funktionen einer Spiele-Engine, wie Unity oder Adobe Animate, abbilden und damit den Einstieg erleichtern.

In dieser Masterarbeit werden zwei Teilaspekte von FUDGE geplant, konzipiert und implementiert: ein Vektoreditor zum Erstellen von Texturen sowie ein Animationseditor zum Animieren beliebiger Objekteigenschaften.

There are many different game engines on the market, most of them are commercial. Many engines do some things very similar, in other things they are very different. Often, the engine must be left to create assets. Very few of these engines are tailored to teaching but focus on the professional and commercial application. Because of all these factors, it is often awkward or unsuitable to use these engines in teaching.

For all these problems, FUDGE (Furtwangen University Didactic Game Editor) should help. FUDGE is a web-based, cross-platform game editor that is specifically tailored towards teaching. It is intended to provide the most basic functions of a game engine, such as Unity or Adobe Animate, and thus make it easier to get started.

In this master thesis, two aspects of FUDGE are planned, designed and implemented: a vector editor for creating textures and an animation editor for animating arbitrary object properties.

Inhaltsverzeichnis

Abbildungsverzeichnis

Codeverzeichnis

Abkürzungsverzeichnis

FUDGE – Furtwangen University Didactic Game Editor

HTML – Hypertext Markup Language

CSS – Cascading Style Sheets

TS – TypeScript

JS – JavaScript

UML – Unified Modeling Language

DOM – Document Object Model

OOP – Objektorientierte Programmierung

UI – User Interface

1 Einleitung

1.1 Ausgangssituation

Die Spielebranche ist bereits seit mehreren Jahren eine der umsatzstärksten Unterhaltungsbranchen weltweit. In Deutschland war der Jahresumsatz der Branche im Jahr 2018 bei über 3,3 Milliarden Euro [2], damit liegt Deutschland weltweit auf dem fünften Platz [3] und die Spielebranche in Deutschland vor der Film- und Musikindustrie [4].

Computerspiele werden zu großen Teilen mithilfe von größeren oder kleineren, teilweise kommerziellen, Spiele-Engines entwickelt, welche den Entwicklern eine große Bandbreite an Unterstützung bieten und so den Entwicklungsprozess beschleunigen und vereinfachen. Namhafte Vertreter sind Unity, Unreal Engine, Source Engine und Frostbite Engine. Auch mit anderen Plattformen, welche nicht speziell auf Spiele ausgelegt sind, können Spiele entwickelt werden, wie z. B. Adobe Animate (ehem. Flash), welches seit Jahren im Besonderen zur Entwicklung von Browser-basierten Spielen genutzt wird.

Es ist nicht verwunderlich, dass die Spieleindustrie bereits in den Hochschulen und Universitäten Einzug gehalten hat. So auch an der Hochschule Furtwangen University, an der Jirka Dell'Oro-Friedl als Professor für Game Design unterrichtet und seit 2010 Vorlesungen in den Bereichen Spiele-Design und -Entwicklung hält. Die dabei meistverwendeten Tools waren und sind Unity und Adobe Animate, unter anderem aufgrund ihrer Einsteigerfreundlichkeit durch visuelle Editoren für den inhaltlichen Aufbau. [1]

Doch diese Programme sind nicht für die Lehre konzipiert, wodurch zusätzlicher Aufwand und viele Unannehmlichkeiten entstehen. Vor allem unter dem Aspekt der dezentralen Entwicklung, da die Studierenden größtenteils außerhalb der Vorlesungen und Praktika eigenständig an ihren Übungen weiterentwickeln. Diese Probleme treten unter anderem im Bereich der Codeeinsicht, Problemhilfe, Austausch von Verbesserungsvorschlägen und vielem mehr auf. Aus der Motivation heraus dies zu verbessern, ist die Idee für FUDGE (Furtwangen University Didactic Game Editor) entstanden. [1]

FUDGE soll also vor allem ein didaktischer Game Editor für die Lehre werden, welcher außerdem alle nötigen Werkzeuge mitliefert, um rudimentäre und einfache

Spiele zu entwickeln. Der Fokus und das darüberliegende Paradigma liegen auf der Lehre und alle anderen Entscheidungen haben sich diesem unterzuordnen. Daraus folgt unter anderem, dass sämtliche Daten in menschenlesbarem JSON abgespeichert werden sollen und selbst für FUDGE-interne Strukturen und Algorithmen sind gut verständliche anstatt hyper-performanter Lösungen zu verwenden. Um plattformübergreifende Anwendungen zu ermöglichen und Standardtechnologien mit all ihren bereits anderweitig integrierten Unterstützungen zu nutzen, wird FUDGE auf der Basis der Webtechnologien entwickelt. Mehr zu FUDGE in Kapitel 2.4. [1]

FUDGE soll auch sämtliche zur Designzeit benötigte Editoren beinhalten. Dazu gehören unter anderem auch ein Animationseditor, um Animationen zu erstellen und zu bearbeiten, sowie ein rudimentärer Grafik- bzw. Vektoreditor, um Texturen schnell und simpel erstellen zu können.

Diese beiden Editoren werden im Rahmen dieser Masterarbeit geplant und implementiert.

1.2 Zielsetzung

Ziel dieser Arbeit ist es zwei Editoren zu entwickeln:
Der Animationseditor soll innerhalb von FUDGE eingebunden werden, um Objekte innerhalb der Anwendungen animieren zu können. Dafür müssen sowohl eine Benutzeroberfläche, eine interne Darstellung der Animationen sowie Verbindungen zwischen den Objekten und ihren Animationen hergestellt werden.
Der Vektoreditor ist etwas außerhalb von FUDGE gehalten, soll sich aber an der Bedienung und dem Aussehen des Haupteditors orientieren. Auch hier müssen Benutzeroberfläche und interne Darstellung sowie Export/Import in Texturen ausgearbeitet werden.

Diese beiden Editoren sollen unter der strikten Beachtung der FUDGE Grundsätze entwickelt werden, so dass auch diese von Studenten eingesehen und verstanden werden können.

Die wohlüberlegte und vollständige Planung steht hierbei im Vordergrund, damit wenige Einschränkungen vorhanden und so viele Erweiterbarkeiten wie möglich ohne erheblichen Refaktorisierungsaufwand realisierbar sind.

1.3 Problemdarstellung

FUDGE steckt zum Endzeitpunkt dieser Arbeit noch stark in den Kinderschuhen. Zu Beginn dieser Arbeit war noch keine Zeile Code vorhanden und auch die Planung gerade erst angelaufen. Darum musste an vielen Stellen dieser Arbeit Pionier- und Planungsarbeit geleistet werden, um die spätere Integration in FUDGE so reibungslos wie möglich zu gestalten.

2 Grundlagen

2.1 Web

FUDGE basiert auf den neusten Webtechnologien. Dies hat den Vorteil, dass Webtechnologien zu den heute am weitesten verbreiteten Standards gehören und darum bereits viele Technologien und Techniken zur Verfügung stehen und übernommen werden können, statt auch diese komplett neu entwickeln zu müssen. Neben den grundsätzlichen Möglichkeiten durch HTML und CSS existieren viele weitere Standards, welche für die interaktiven Anwendungen im Web genutzt werden können.

Somit stellt FUDGE eher einen Überbau dar, bei der die bereits existierenden Möglichkeiten vereint und zentralisiert werden, um so die Erstellung solcher Anwendungen zu vereinfachen und übersichtlicher zu gestalten. Darum bezeichnet sich FUDGE auch nicht als Engine sondern als Editor, da er keine eigenen grundlegenden Technologien entwickelt oder bereitstellt, auf denen diese Anwendungen laufen.

2.1.1 HTML

HTML (Hypertext Markup Language, Englisch für Hypertext Auszeichnungssprache) stellt zusammen mit dem in den Browsern erzeugten DOM (Document Object Model, Englisch für Dokumenten-Objekt-Modell) die Grundlage für die Webseiten, wie sie heute bekannt sind, dar. Mithilfe von HTML kann der Inhalt sowie der inhaltliche Aufbau einer Seite definiert werden.

Der aktuelle Standard ist HTML5, welcher viele Funktionalitäten vorsieht. Da diese sehr umfangreich sind, hier nur die für die vorliegende Arbeit wichtigsten Elemente:

- Canvas
  Ein Canvas (dt. Leinwand/Zeichenfläche) Element erlaubt, über die Scripting API auf der definierten Fläche zu malen. Dabei können nicht nur primitive, sondern auch komplexere Objekte gezeichnet werden. Außerdem stellt ein Canvas auch eine Schnittstelle zu WebGL bereit.
- Button (dt. Knopf/Schaltfläche)
- Input (dt. Eingabe)
  Ein Input Element erlaubt dem Nutzer die Eingabe oder Manipulation von

Daten. Dabei ist es egal ob es simple Checkboxen, Radiobuttons, Zahleneingaben oder Textfelder sind, sie alle sind verschiedene Ausprägungen des Input Elements.

- Scripting API
  Die Scripting API erlaubt es einem Entwickler mithilfe von JavaScript das HTML-Dokument und alle seine Eigenschaften und Objekte zu manipulieren und zu verändern.

[5, 6]

2.1.2 CSS

CSS (Cascading Style Sheet, Englisch für Kaskadenstilentwurf) erlaubt die Beeinflussung der Darstellungsform und Formatierung der zuvor im HTML festgelegten Inhalte. So können Farbe, Größe, Ausrichtung und vieles mehr definiert werden.

Der aktuelle Standard ist CSS3.

2.1.3 JavaScript

JavaScript (JS) ist eine Skriptsprache die Webseiten um weitere Funktionalität, die nicht in HTML oder CSS abgebildet werden kann, erweiterbar macht. Ursprünglich speziell für die Anwendung in Browsern entwickelt, wird sie heute auch in anderen Bereichen eingesetzt, wie etwa zur Serverprogrammierung.

Der JS Standard heißt ECMAScript (ES), die aktuelle Version ist 2018 bzw. 6. JavaScript ist zwar eine objektorientierte, aber auch eine klassenlose sowie dynamisch typisierte Programmiersprache. [8]

2.1.4 TypeScript

TypeScript (TS) ist ein von Microsoft entwickeltes Superset für JavaScript. Es erlaubt die Arbeit in JS mit zusätzlichen Features wie der namensgebenden expliziten Typisierung und der damit verbundenen Klassifizierung. Auch Konstrukte wie Interfaces sind in TypeScript möglich. Viele der Standards aus ES6 wurden von TypeScript übernommen. Durch diese Erweiterung ist jeglicher JS Code auch gültiger TypeScript Code, was die Nutzung von bereits existentem JS Code auch in Form von Bibliotheken in TypeScript ermöglicht.

TypeScript wird zur Verwendung nach JS ES6 bzw. ES5 kompiliert, es existiert demzufolge nur zur Designzeit. [7]

2.1.5 WebGL

Die Web Graphics Library (WebGL) ist eine von der Khronos Group - ein Industriekonsortium, das sich für die Entwicklung von Standards im Multimediabereich einsetzt - entwickelte Schnittstelle, mit der 3D Grafiken im Webbrowser hardwarebeschleunigt dargestellt werden können. Das HTML Canvas Element erlaubt die Nutzung von WebGL zur Darstellung von komplexen 3D Grafiken und Szenen.

2.2 Vektorgrafiken

Vektorgrafiken sind im Gegensatz zu Pixelgrafiken nicht durch die Beschreibung der einzelnen Bildpunkte definiert, sondern mathematisch über Formeln beschrieben. Dadurch sind Vektorgrafiken theoretisch unendlich skalierbar, ohne dass Artefakte entstehen oder Pixel sichtbar werden.

Beide Arten der Grafikdarstellung haben ihre Legitimation, da die Anwendungsfälle unterschiedlich sind. Normale Fotos zum Beispiel sind als Vektorgrafik gar nicht umsetzbar aufgrund der Detailtiefe und Farbvielfalt. Für Logos oder andere Objekte mit klarer Linienführung und Farbgebung eignen sich Vektorgrafiken jedoch deutlich besser.

Durch die mathematische Beschreibung ist auch die Dateigröße im Allgemeinen kleiner.

Zur Erstellung dieser verschiedenen Arten bedarf es unterschiedlicher Werkzeuge. Für FUDGE wurde deshalb entschieden, eine interne eigene Darstellung von Vektorgrafiken zu nutzen und dafür einen Editor anzubieten. Dieser wird sich auf die wesentlichen Dinge konzentrieren, um den Nutzer nicht mit seiner Nutzungsvielfalt zu erschlagen. Außerdem können so die Vektordateien einheitlich mit dem Rest der Daten als JSON Datei abgespeichert werden und müssen nicht aus oder in weitere Formate ex- bzw. importiert werden.

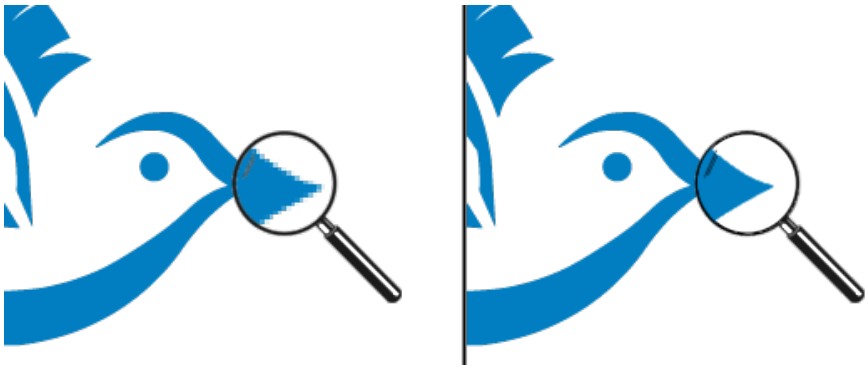

Abbildung 2-1 Symbolbild Pixelgrafik (links) und Vektorgrafik (rechts) im Vergleich.[1]

Auch für Vektorgrafiken gibt es bereits Standards wie z. B. SVG. Dieses Format wurde aus mehreren Gründen nicht in FUDGE angewendet:
Zum einen wird SVG in XML abgespeichert und nicht in JSON wie alles andere in FUDGE und zum anderen kann so die Erweiterung auf 3D für einen Modeleditor direkt mitbedacht werden. SVG hat außerdem sehr viele Fähigkeiten und Eigenschaften, welche in FUDGE nicht benötigt werden und so dem Nutzer den Einstieg erschweren würden.

2.3 Animationen in digitalen Spielen

Ohne Animationen wären viele Spiele heutzutage nicht mehr vorstellbar. Sie bilden einen integralen Teil der heutigen Spielewelt. Es wird unterschieden zwischen zwei verschiedenen Arten der Animation: zeitbasierte und bildbasierte Animationen.

Zeitbasierte Animationen sind nichts anderes als die Veränderung einer Eigenschaft, meist der Transformation (welche aus Position, Rotation und Skalierung besteht), abhängig von der vergangenen Zeit, sie lassen sich also rein mathematisch als Veränderungsfunktionen über Zeit beschreiben. Da solche Formeln aber nicht intuitiv sind und keine einfachen Änderungen zulassen, werden in Animationstools generell die Funktionen automatisch im Hintergrund erzeugt und können dabei von den Animatoren über Schlüsselbilder festgelegt werden, zwischen denen dann interpoliert wird. Diese Schlüsselbilder werden in sogenannten Dopesheets und Curve Views angezeigt.

[1] https://de.serlo.org/informatik/darstellung-informationen/vektor-pixelgrafik

Dabei bieten Dopesheets eine bessere Übersicht wann welcher Schlüsselwert erreicht wird, während die Curve Views die zeitliche Veränderung besser visualisieren und leichter manipulieren lassen.

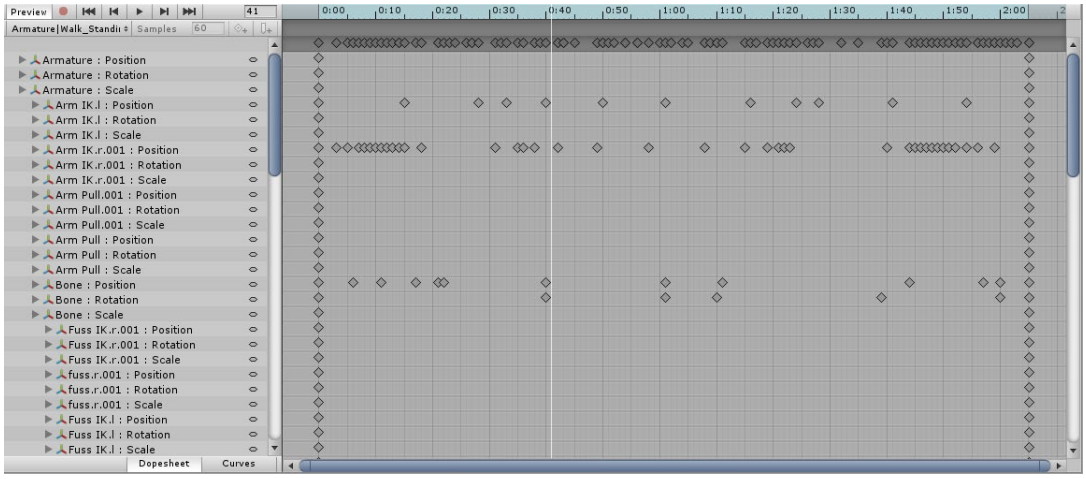

Abbildung 2-2 Dopesheeet für eine Laufanimation in Unity (Version 2018.1)

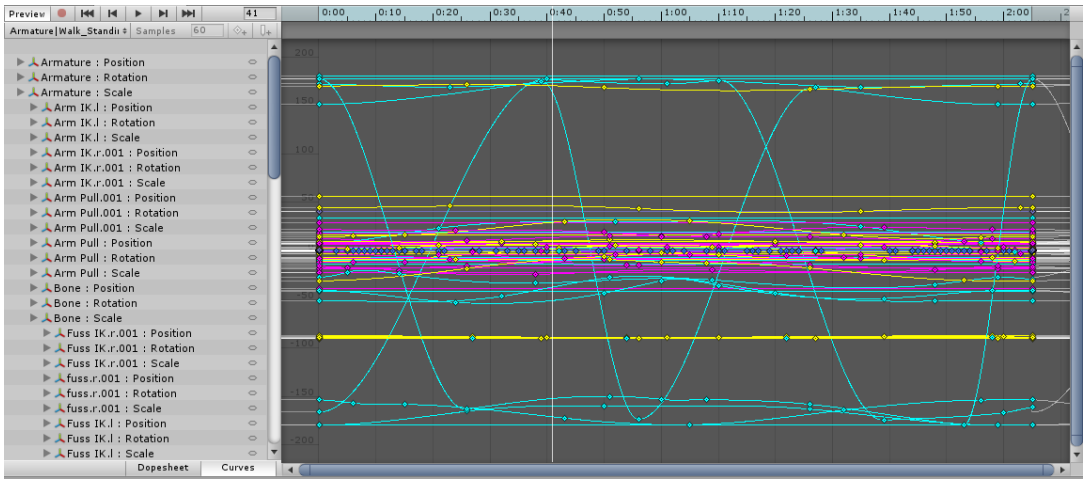

Abbildung 2-3 Curve View für eine Laufanimation in Unity (Version 2018.1)

Bildbasierte Animationen sind im Gegensatz dazu nicht an die Zeit gekoppelt sondern an ein bestimmtes Bild bzw. eine Bildfolge. Auch hier werden Schlüsselbilder erstellt, zwischen denen Veränderungen interpoliert werden. Der Unterschied zur zeitbasierten Animation ist auch der, dass wenn sich die Bildwiederholungsrate (Bilder pro Sekunde, FPS) ändert, ändern die Animationen dementsprechend ihre Geschwindigkeit, während bei der zeitbasierten Animation größere Schritte zwischen den Bildern gemacht werden.

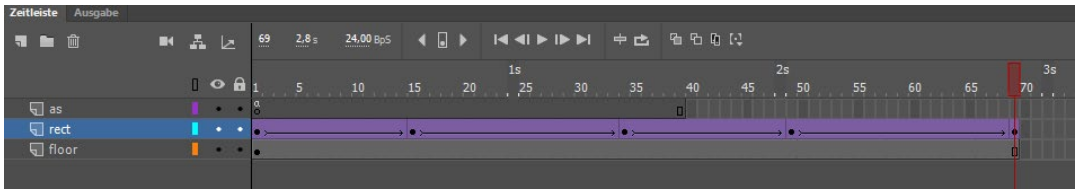

Abbildung 2-4 Einfache Bildbasierte Animation in Adobe Animate (Version 19.1)

Es gibt noch einen dritten, den bildbasierten Animationen zugehörigen, aber selteneren weil aufwändigeren Ansatz, bei dem jedes einzelne Bild neu gezeichnet wird und so nicht eine Veränderung von gleichen Objekten über Zeit oder Bilderfolge abgebildet, sondern ein komplett neues Bild pro angezeigtem Bild erstellt wird. Dieser Ansatz wird nicht in FUDGE abgebildet sein, da dies ein sehr spezieller Anwendungsfall ist, welcher für die Umsetzungen mit prototypischem Charakter in FUDGE nicht relevant ist.

2.4 FUDGE

FUDGE steht, wie anfangs erwähnt, für den Furtwangen University Didactic Game Editor, ein Projekt von Professor Jirka Dell'Oro-Friedl, Professor für Anwendungskonzeption unter besonderer Berücksichtigung von Game Design an der Hochschule Furtwangen. Die Entwicklung von FUDGE ist dem Umstand geschuldet, dass die gängigen Spiele-Engines nicht für die Lehre ausgelegt sind und somit darin einige Probleme mit sich bringen, welche FUDGE versucht zu lösen. Dabei erhebt FUDGE ganz klar nicht den Anspruch, ein Konkurrent für etablierte Systeme zu werden oder die beste Lösung für anspruchsvolle Spiele zu sein.

Als oberstes Paradigma hat sich FUDGE die Lehre zum Ziel gesetzt:

> *Das Hauptziel ist es, die Strukturen und Prozesse von Spielen sowie deren Entwicklung zu verdeutlichen und verständlich zu machen, dabei die Flexibilität und Anwendungsmöglichkeiten hoch und die Bedienbarkeit einfach zu halten. Alle Entscheidungen müssen sich dem unterwerfen. Andere Dinge wie Performanz oder visuelle Effekte haben eine geringere Priorität. [9]*

FUDGE basiert aus verschiedenen Gründen auf den in diesem Kapitel bereits erläuterten Webtechnologien:

- Browser sind plattformübergreifend.
  Durch die Popularität des Internets ist heutzutage jeder Computer, jedes Tablet und jedes Smartphone in der Lage Webseiten darzustellen. Mit FUDGE erstellte und in Webseiten eingebundene Anwendungen können so ohne zusätzlichen Aufwand standardmäßig auf den gängigsten Plattformen ausgeführt werden.
  Zwar ist der Editor auf die Bedienung mit Maus und Tastatur an einem Desktop PC ausgelegt, aber die Anwendung selbst ist nicht darauf beschränkt.
- Standards
  Das Web hat bereits wohletablierte Standards (HTML, CSS, WebGL, etc.) auf denen aufgebaut werden kann. So muss z. B. keine eigene grafische Ausgabe entwickelt werden, da Technologien wie Canvas und WebGL bereits zur Verfügung stehen. Diese Standards werden ständig weiterentwickelt und erweitert, was ihnen eine hohe Akzeptanz und Verbreitung zusichert.
- JavaScript/TypeScript
  Die gängigste Programmiersprache im Web ist JavaScript. Bis auf die Kompilierung von TypeScript nach JavaScript zur Designzeit muss nichts kompiliert werden und somit kann das Spiel direkt und praktisch ohne Wartezeit getestet werden.
  Außerdem gibt es für JavaScript Objekte die JSON, eine menschenlesbare Möglichkeit diese zu serialisieren und abzuspeichern.
- Debugger
  Da sämtliche Anwendungen letztendlich nur Webseiten sind, können die im Browser für die Webentwicklung bereits bereitgestellten und integrierten Debugger genutzt werden, um aus dem Stegreif heraus das Spiel nach Fehlern zu durchsuchen. In anderen Spiele-Engines müssen, um den im Browser ausführbaren Code nach Fehlern durchsuchen zu können, zunächst die Quelldateien herunterladen werden, diese dann innerhalb der Engine geöffnet werden, um dort dann zu versuchen, den Fehler zu reproduzieren.

Es gibt bereits andere webbasierte (Spiele-)Engines wie Babylon.js oder Three.js, diese haben aber alle einen entscheidenden Nachteil: Sie haben keinen visuellen Editor, somit muss alles über Code gemacht werden, was eine größere Einstiegshürde bedeutet. Selbst wenn sie einen integrierten Editor haben, so ist dieser nur online verfügbar und man kann seine Projekte nur (kostenpflichtig) beim Anbieter und nicht lokal speichern, was zur Durchführung einer eventuellen Prüfungsleistung ein Ausschlusskriterium ist.

Darüber hinaus folgen diese Engines oft dem Ansatz einer tiefen Klassenhierarchie anstelle eines komponentenbasierten Modells. Dies sorgt zusammen mit den nicht vorhandenen visuellen Editoren dafür, dass die Lernkurve noch steiler und damit schwerer zu erklimmen ist.

Den oben erwähnten Paradigmen folgen noch weitere angestrebte Vorteile von FUDGE gegenüber anderen Engines, wie etwa dass alle Daten menschenlesbar sein sollten, um nicht nur den Lernenden und Lehrenden die Möglichkeit der direkten Editierung und Einblick in die internen Strukturen zu gewähren, sondern auch die Nutzbarkeit mit Versionsverwaltungssystemen zu verbessern.

Die erzeugten Datenmengen sind außerdem sehr gering in Größe und Umfang, was schnellere Hoch- und Herunterladezeiten zur Folge hat. Auch wird die Lagerung von Abgaben im Rahmen von Veranstaltungen an der Hochschule so deutlich vereinfacht. Dies kommt zum Teil auch durch den geringeren Funktionsumfang und teilweise dem damit verbundenen geringeren Mehr- bzw. Grundaufwand zustande, so dass ein Projekt nur aus den Teilen und Dateien besteht, welche vom Nutzer angelegt wurden. In anderen Engines wie z. B. Unity besteht ein Projekt zusätzlich aus zahllosen weiteren Dateien, welche die Projektgrößen selbst für sehr simple Anwendungen schnell in mehrere hundert MB wachsen lassen.

Mithilfe von Programmen wie Electron können Webanwendungen in native Anwendungen konvertiert werden. So soll auch FUDGE als nativ ausführbare Anwendung zur Verfügung stehen.

FUDGE soll später alle Werkzeuge mitliefern, welche zur Erstellung von interaktiven Anwendungen oder digitalen Spielen benötigt werden, so dass theoretisch keine externen Tools genutzt werden müssen. Darum soll in FUDGE jeweils ein eigener Editor für 2D Grafiken, 3D Modelle, Animationen sowie für den

Szenenaufbau integriert werden. Letzterer bildet die oberste Ebene und ist damit der mächtigste Editor, in dem alles zusammenkommt, vergleichbar mit dem Unity Editor. [1]

3 Konzeption Vektoreditor

3.1 Anforderungen

Die grundlegenden, nichtfunktionalen Anforderungen an den Vektoreditor sind im Allgemeinen durch die Anforderungen von FUDGE definiert, denn dabei gelten als oberste Prioritäten die einfache Nutzbarkeit für Anfänger sowie die klare Verständlichkeit der unterliegenden Strukturen. Es soll aber auch für fortgeschrittene Nutzer Abkürzungen enthalten, welche die Bedienung erleichtern. Des Weiteren sollen die geschaffenen Bedienungskonzepte einheitlich mit den anderen Editoren zusammenpassen. Da diese anderen Editoren zum Zeitpunkt dieser Arbeit noch nicht existieren, müssen diese hier neu entwickelt werden. Letztendlich müssen die unterliegenden Strukturen selbstverständlich nicht nur gut strukturiert und übersichtlich sein, um die Wartbarkeit zu gewährleisten, sondern auch erweiterbar und übertragbar auf die anderen Editoren.

Die funktionalen Anforderungen des Nutzers lassen sich dem Anwendungsfalldiagramm in Abbildung 3-1 entnehmen. Es sollen die wichtigsten Möglichkeiten eines Vektorgrafikeditors abgebildet werden, also die Erstellung und Manipulation von 2D Objekten mit besonderem Fokus auf Position, Form, Größe und Farbe.

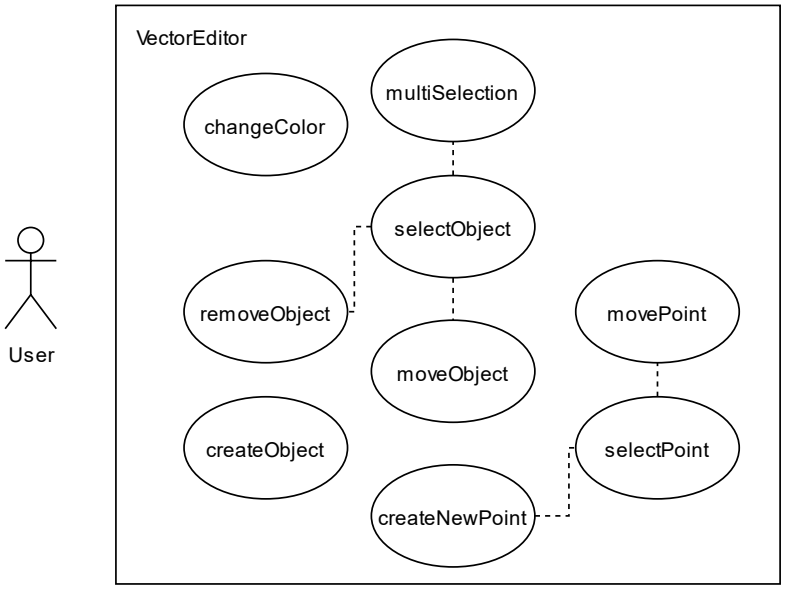

Abbildung 3-1 Anwendungsfalldiagramm Vektoreditor

3.2 Klassenstruktur Grafikobjekte

Aufgrund dieser Anforderungen kann eine Klassenstruktur für die neu zu entwickelnde 2D-Grafik-Darstellung entwickelt werden. Der Name *Sketch* wurde als übergreifende Namensgebung gewählt, um die Natur der dadurch erstellten Objekte klar zu kommunizieren. Zur Abgrenzung wird der Namensraum SketchTypes genutzt.

Mutable
Eine Klasse aus dem FUDGE Kern, welche die Veränderbarkeit eines Objektes steuert. Klassen, welche von Mutable abgeleitet sind, können damit ihre zur Laufzeit offenen Bearbeitungsmöglichkeiten steuern sowie spezielle Schnittstellen zu UI und Animationsmöglichkeiten integrieren. Alle weiteren Klassen in dieser Struktur leiten sich (in)direkt von dieser Klasse ab, um eine direkte Manipulation durch UI und Animator zu ermöglichen.

Vector2
Diese Klasse beinhaltet die Attribute x und y und beschreibt damit einen Punkt im zweidimensionalen Raum. Außerdem bietet sie einige Rechenoperatoren mit Vektoren an, um die Nutzung zu vereinfachen.

SketchPoint
SketchPoint bildet das primitivste Objekt innerhalb eines Sketeches: einen einfachen Punkt welcher primitive Funktionen zur Manipulation wie `move()` zur Verfügung stellt. Da er im Editor klickbar und damit veränderbar sein soll, hat er außerdem eine `Path2D` Komponente.

SketchObject
SketchObject bildet das primitivste Sketch Objekt ab, indem es die grundlegendsten Eigenschaften zur Verfügung stellt, die ein solches Objekt braucht, um richtig angezeigt werden zu können. Dazu gehören besonders die Farbe und die Reihenfolge auf der Zeichenfläche sowie ein Pfadobjekt.
Diese Klasse könnte nach dem Klassendiagramm auch weggelassen und mit SketchPath zusammengeführt werden. Allerdings ist so die Erweiterbarkeit auch direkt gewährleistet, damit in der Zukunft noch weitere, spezialisierte Arten der Objekte (denkbar wäre z. B. Text) hinzugefügt werden können.

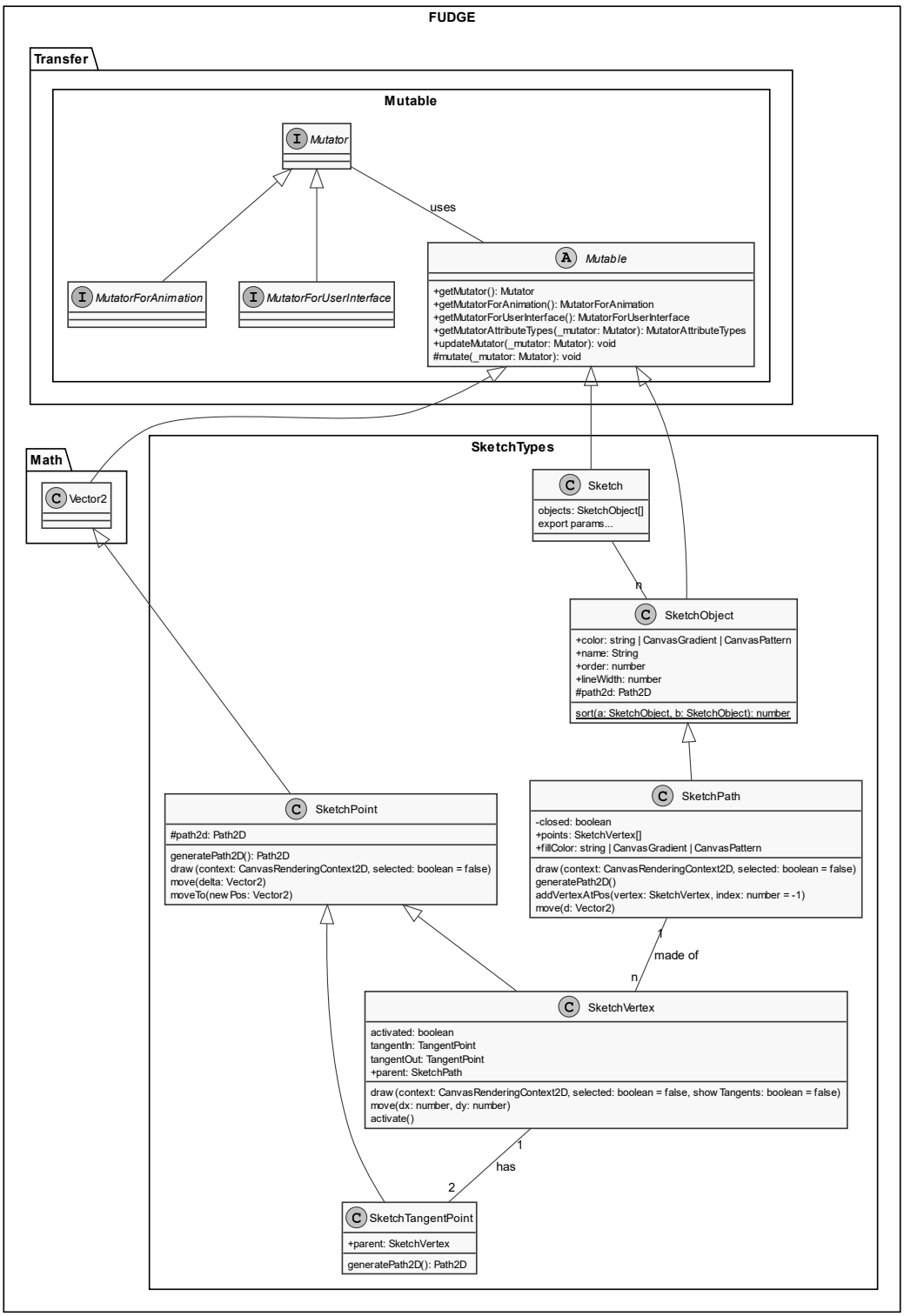

Abbildung 3-2 Klassendiagramm SketchTypes

SketchPath

Ein SketchPath stellt die tatsächliche Klasse zur Beschreibung, der nach diesem

Plan entwickelten Sketch Objekte, dar. Er besteht aus einer geordneten Sammlung von Vertices, welche durch Linien miteinander verbunden werden, um so ein beliebig komplexes Objekt abzubilden.

SketchVertex

Auf der Basis des SketchPoint bildet der SketchVertex einen zentralen Knotenpunkt, über den jedes SketchPath Objekt aufgebaut wird und stellt somit die Eckpunkte der Objekte dar. Dabei kann der Vertex aktiviert werden, um über die dadurch eingeblendeten Tangentenpunkte die Verbindungslinien zu den vorherigen und nächsten Vertices zu verformen.

SketchTangentPoint

Jeweils zwei dieser Tangentenpunkte hängen einem Vertex an und erlauben nach Aktivierung desselben, die Verbindungslinien zu beeinflussen.

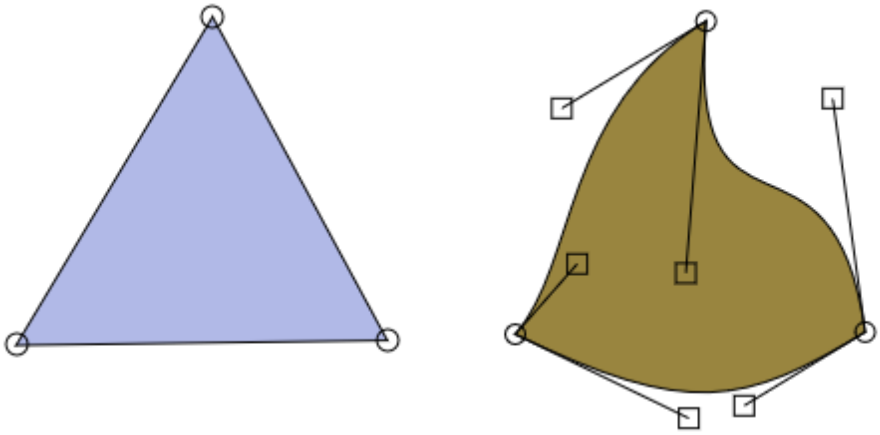

Abbildung 3-3 Zwei Dreiecke, links ohne aktivierte Vertices, rechts mit aktivierten Vertices und verschobenen Tangenten

○ = Vertex, □ = Tangente

Sketch

Ein Sketch bildet die Obermenge zu mehreren SketchObjekten und fasst diese zu einem weiterverwendbaren Objekt zusammen. In dieser Form werden auch alle Sketche zusammengefasst, als Datei gespeichert sowie an weitere Systeme (wie den Renderer für die Textur oder den Animator) übergeben. Darum werden in diesem auch die Export Parameter festgelegt, welche dann diese weiteren Einstellungen speichern sollen. Der Übergang von Sketch auf Textur kann aufgrund des Nichtvorhandenseins der Textur in FUDGE zum Endzeitpunkt dieser Thesis nicht festgeschrieben werden.

Mit Hilfe dieser Objektstruktur lassen sich nicht nur grundlegende, sondern auch beliebig komplexe Formen darstellen. Die einzige Einschränkung hier besteht darin, dass ein Objekt nur eine Farbe, einen Farbverlauf, oder ein Muster haben kann.

3.3 Aufbau Vektoreditor

Die unterliegende Hierarchie ist das eine, die Schnittstelle zum Nutzer das andere. Hier müssen zwei Fronten bedient werden: Einerseits muss ein guter Aufbau geschaffen werden, der die möglichst intuitive Nutzung durch den unerfahrenen Endnutzer zulässt, dabei aber auch dem erfahrenen Nutzer Möglichkeiten der effizienteren Nutzung bietet. Andererseits müssen auch hier die Paradigmen der Erweiterbarkeit, Wartbarkeit und Übertragbarkeit bei der Implementierung beachtet werden.

Zunächst wurde ein fast UI-loser Ansatz verfolgt. Hierbei war es vorgesehen, die häufigsten Aktionen lediglich über Tastenkombinationen zu aktivieren und zu deaktivieren, um so ein schnelles und effizientes Arbeiten zu ermöglichen. Die Interaktionsmöglichkeiten dieses Ansatzes können in Anhang A eingesehen werden. Dieser Ansatz wurde allerdings aufgrund von zwei Hauptgründen verworfen:
1. Die Bearbeitung war zwar schnell, allerdings mussten die Tastenkombinationen irgendwie dargestellt werden.
2. Es konnte keine programmatische Abbildung gefunden werden, welche sich an die Paradigmen hält.

Durch Rücksprache mit Lea Stegk, die zum Abgabezeitpunkt dieser Thesis noch an ihrer eigenen Thesis (UX-Design und Entwicklung der UI für einen Spieleeditor und Optimierung der Userexperience) arbeitet, wurde ein Tool-basiertes Konzept entwickelt. Darin sind verschiedene Tools zur Aktivierung der unterschiedlichen Manipulationsmöglichkeiten vorgesehen. Diese lassen auch eine bessere Erweiterbarkeit zu (s. Kapitel 3.3.1 & 3.3.3).

Zur Akzentuierung von angewählten Elementen wird folgende Darstellung gewählt:

Objekte
Ausgewählte Objekte werden dadurch hervorgehoben, dass ihre Eckpunkte

(Vertices) sichtbar gekennzeichnet werden. Dies hat den Nebeneffekt, dass Vertices erst bewegt werden können, nachdem die zugehörigen Objekte ausgewählt wurden.

Vertices
Anwählbare Vertices werden durch einen kleinen, unausgefüllten Kreis dargestellt. Sobald ein Vertex ausgewählt ist, wird dieser durch einen gefüllten Kreis hervorgehoben.

Tangentenpunkte
Die Tangentenpunkte werden ähnlich wie die Vertices dargestellt, mit dem Unterschied, dass sie statt durch einen Kreis durch ein Quadrat visualisiert werden. Zudem werden sie mit dem zugehörigen Vertex durch eine einfache Linie verbunden. Die Tangentenpunkte werden nur angezeigt, wenn der zugehörige Vertex aktiviert wurde.

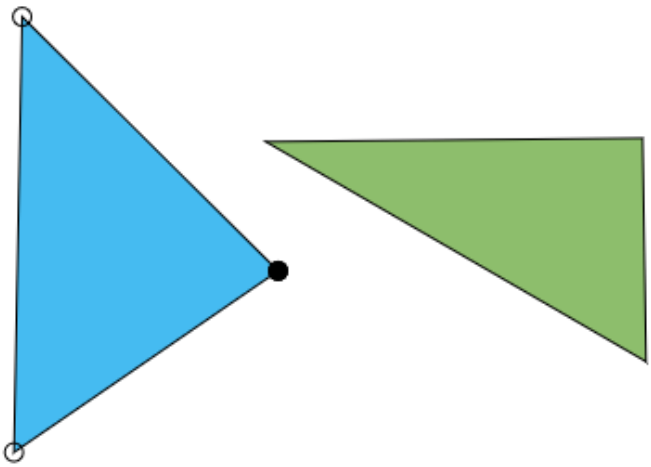

Abbildung 3-4 Vergleich zwischen gewählten und nicht gewählten Objekten

3.3.1 Tools

Die folgenden Tools sollen zur Verfügung stehen, da sie die elementaren Bearbeitungsmöglichkeiten abbilden.

Select
Dies ist das Standardtool. Es erlaubt dem Nutzer auf verschiedene Arten Objekte und Punkte auszuwählen. Durch ein Klicken auf ein Objekt oder einen Punkt kann

es ausgewählt, durch Klicken bei gehaltener STRG-Taste können mehrere Objekte ausgewählt werden. Durch das Aufziehen eines Rechtecks können alle Objekte sowie auswählbare Punkte, in dem davon abgedeckten Bereich, ausgewählt werden.

Transform
Erlaubt es dem Nutzer, die ausgewählten Objekte und Punkte zu rotieren, zu skalieren oder zu verschieben.

Create Shape
Erzeugt eine neue Form unter einem aufgezogenen Rechteck. Dabei stehen verschiedene Vorlagen zur Verfügung, wie z.B. ein Rechteck/Quadrat, eine Ellipse/Kreis, ein Stern, etc.

Add Vertex
Ein Tool mit zwei Anwendungsfällen: entweder kann einem vorhandenen Pfad ein weiterer Vertex hinzugefügt oder ein neuer Pfad aus Vertices erstellt werden. Indem auf eine vorhandene Linie eines ausgewählten Objektes geklickt wird, wird dort ein neues Objekt erstellt. Wenn der erste Klick nicht auf ein ausgewähltes Objekt geht, dann wird der zweite Modus genutzt. Dieser wird abgeschlossen, indem entweder auf den als erstes erstellten Vertex geklickt und so ein geschlossenes Objekt erstellt wird oder indem der Bearbeitungsvorgang abgebrochen und so ein offenes Objekt erstellt wird. Der Vorgang kann durch Tastendruck auf Escape oder ein beliebiges anderes Tool abgebrochen werden.

Weiteres
Weitere Tools können einfach eingebaut werden. Hilfreich könnten Tools zur Kombinierung von mehreren Objekten oder zur Anordnung/gleichmäßigen Verteilung von Objekten sein.

3.3.2 User Interface

Das UI (User Interface, engl. Benutzeroberfläche) orientiert sich hierbei an anderen Grafik- und Vektorprogrammen wie Sketch, Inkscape, PaintDotNet oder Adobe Illustrator. Eine konzeptionelle Darstellung findet sich in Abbildung 3-5.

Am linken Bildschirmrand werden die Haupttools als Icons angezeigt und können durch Klicken ausgewählt werden. Am oberen Bildschirmrand befindet sich unter

der Menüleiste eine Sub-Tool Leiste, welche Untertools auflistet und (sub-)toolspezifische Einstellungen anzeigt. Auf der rechten Seite befinden sich die Einstellungen des aktuell ausgewählten Objektes. Viele Editoren nehmen dafür die gesamte Bildschirmhöhe ein, was aber in diesem Fall nicht notwendig ist. Das Eigenschaftsfenster ändert seine Größe dynamisch mit dem angezeigten Inhalt und befindet sich am oberen Bildschirmrand, um so die Mauswege zu verkürzen. Ist kein Objekt ausgewählt, so werden die allgemeinen Sketch Einstellungen angezeigt.

Alle drei UI Elemente sollten im Optimalfall frei beweglich und andockbar sein, so dass die Nutzer die Möglichkeit haben, ihre Nutzeroberfläche zu individualisieren und an ihren persönlichen Arbeitsfluss anzupassen.

Alle Aspekte des UI, die sich über HTML und CSS lösen lassen, soll auch darüber gelöst werden, damit so wenig wie möglich über eine berechnete Grafikausgabe gesteuert wird. Die Menüleiste wird durch Electron nativ realisiert und die eigentliche Zeichenfläche ist ein HTML-Canvas-Element.

Das tatsächliche Design des UI sollte an das von FUDGE angepasst werden, sobald es dieses gibt (bzgl. der Farben, Schriftarten und Größen).

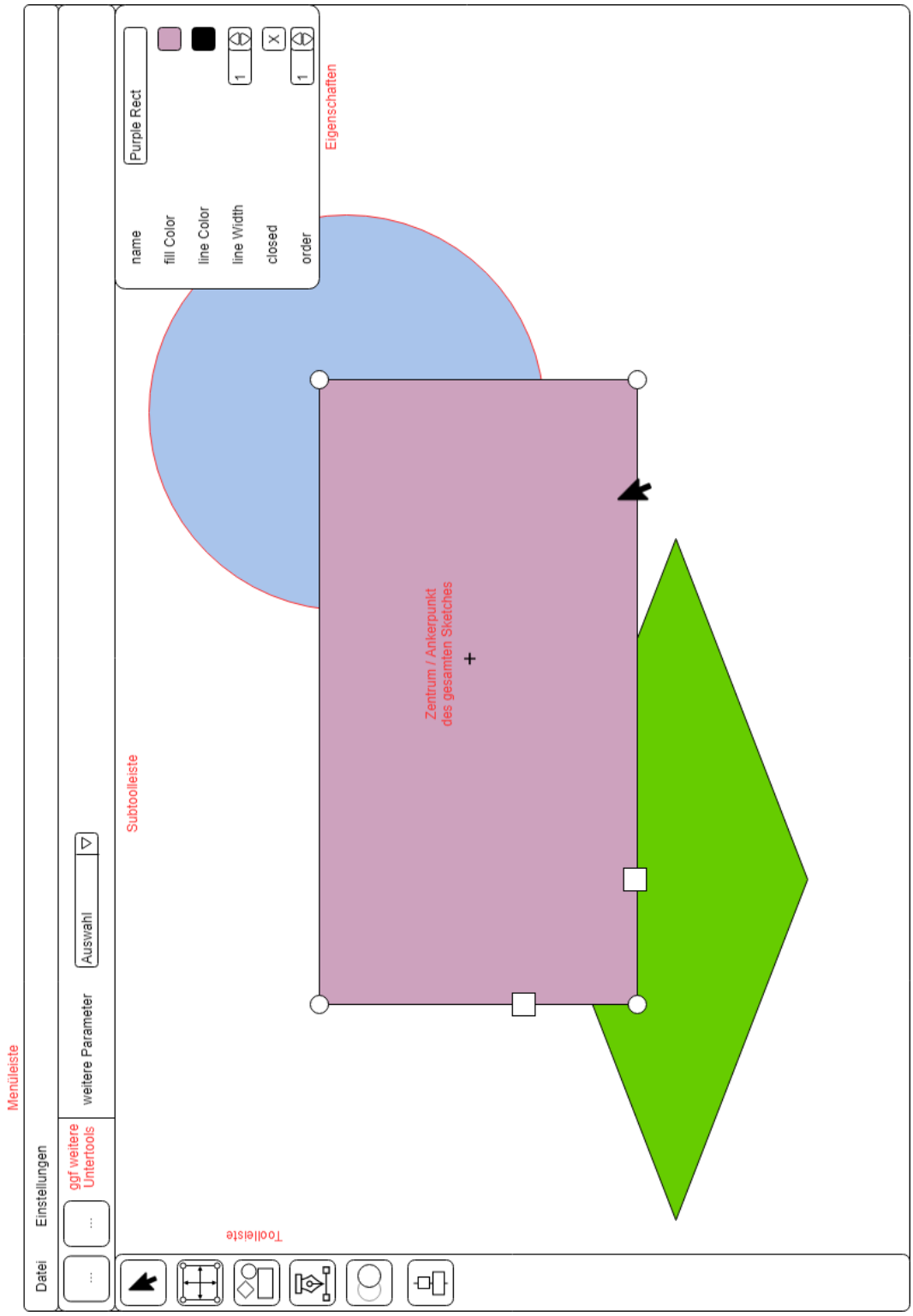

Abbildung 3-5 Konzept Vektoreditor UI

Die Texte in Rot stellen zusätzliche Anmerkungen dar und sind nicht Teil des UI.

3.3.3 Klassenstruktur Vektoreditor

Um den eigentlichen Vektoreditor zu implementieren, müssen UI Elemente genutzt werden. Da alle FUDGE Editoren auf derselben UI Grundlage entwickelt werden sollen, welche dann später auch in den entwickelten Anwendungen zum Tragen kommt, muss das zentrale UI verwendet werden. Da dieses aber zum Endzeitpunkt dieser Thesis noch nicht implementiert ist, musste ein kleiner Vorentwurf dafür entwickelt werden um den Umstieg so einfach wie möglich zu gestalten (s. Abbildung 3-6).

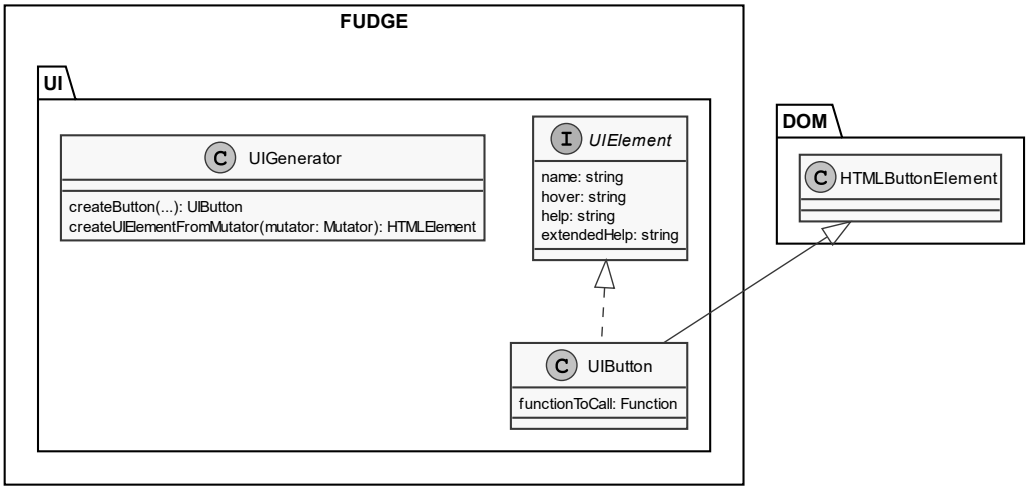

Abbildung 3-6 Klassendiagramm temporäres UI Modul

Dabei kommt das UIElement Interface zum Einsatz, welches definiert, welche Eigenschaften ein UIElement mindestens haben muss. Es geht vor allem darum, ein einheitliches Unterstützungskonzept für den Endnutzer zu entwickeln und so diverse Abstufungen an Hilfestellungen anzubieten. Für den aktuellen Zeitpunkt sollen `name`, `hover`, `help` und `extendedHelp` lediglich den Klartext speichern, später jedoch eine Referenz auf den übersetzbaren Text, der in Sprachdateien abgelegt ist, die eine einfache Lokalisierung ermöglichen sollen. Auch wenn der FUDGE Editor wahrscheinlich nicht in andere Sprachen übersetzt wird, so soll doch die Möglichkeit einer einfachen Integration in die generellen UI Komponenten ermöglicht werden.

Ein UIButton erweitert einen HTMLButton um die Eigenschaften, die im UIElement definiert wurden, sowie um ein Feld für die auszuführende Funktion.

Ein späteres Herzstück des UI stellt der UIGenerator dar. Dieser wird in der Lage sein, aus einem Mutator (s. Kapitel 3.2) ein UI zu erstellen, über die das Element, das von Mutable geerbt hat, direkt verändert werden kann. Dazu werden die entsprechenden HTMLInputElemente automatisch erstellt: für Zeichenketten ein Textfeld, für Zahlen ein Zahlenfeld und für Wahrheitswerte eine Checkbox. Diese Funktionalität soll genutzt werden, um das Eigenschaftsfeld des Vektoreditors dynamisch aufzubauen.

Der eigentliche Vektoreditor nutzt diese UI Klassen an verschiedenen Stellen, im Besonderen im UIHandler.

UIHandler
Der UIHandler bildet die Schnittstelle zwischen dem Editor und den HTML UI Elementen. Dieser kümmert sich um die Erstellung, Aktualisierung und Verwendbarkeit der HTML UI Elemente.

Editor
Ist die große Überklasse, bei der alle Stränge zusammenlaufen und die Befehle weitergegeben werden. Sie verwaltet neben dem aktuell zu bearbeitenden Sketch auch die ausgewählten Objekte, die grafische Ausgabe auf dem Canvas, das gewählte Tool und kümmert sich außerdem um alle Nutzereingaben die nicht mit den HTML UI Elementen zusammenhängen, also sowohl den Tastatureingaben wie auch den Mauseingaben und -bewegungen über die Zeichenfläche.

Shortcut
Ein Shortcut ist lediglich ein Platzhalter, in welchem eine Tastenkombination abgelegt werden kann. So können diese einheitlich verwaltet und einfacher abgefragt werden.

ToolManager
Der ToolManager verwaltet alle, im VektorEditor Namensraum, bekannten Tools. Dazu hat er ein statisches Array, in dem sich alle Tools eintragen können. Dies hat den praktischen Nebeneffekt, dass allein die Existenz der Klassen deren Registrierung bewirkt, so dass keine Instanzen zur Registrierung erschaffen werden müssen, sondern erst dann, wenn sie auch wirklich benötigt werden.

```
class ToolManager {
    static tools: typeof Tool[] = [];

    static registerTool (_tool: typeof Tool): number {
        return ToolManager.tools.push(_tool);
    }
}

class Tool1 extends Tool {
    public static iRegister: number =
ToolManager.registerTool(Tool1);

    constructor() {
        super();
        console.log("Instance of Tool1");
    }
}
```

Code 3-1 Statische Toolverwaltung des ToolManagers[2]

Tool

Die Tool Klasse bildet das Grundgerüst für jede Interaktionsmöglichkeit innerhalb des Vektoreditors. Da jedes Tool eine sichtbare UI Komponente im Editor besitzt, implementiert es das UIElement Interface. Außerdem kann jedes Tool beliebig viele Subtools besitzen, mit denen die Bedienungsmöglichkeiten sowohl algorithmisch als auch visuell für den Nutzer differenzierter aufgeschlüsselt werden können. Des Weiteren schleift das Tool standardmäßig sämtliche Eingabe-Events und seine Sub-Tools durch. Außerdem beinhaltet es Funktionen für zusätzliche Anzeigen auf dem Canvas sowie Subtoolleistenoptionen.
Besonders hervorzuheben ist außerdem die `prequisitesFulfilled()` Funktion, welche als Wahrheitswert zurückliefert, ob die Vorbedingungen für die Anwendbarkeit dieses Tools erfüllt sind, z. B. kann das Transformationstool nur aufgerufen werden, wenn mindestens zwei Punkte oder ein Pfadobjekt ausgewählt sind.

Verschiedene Tools

Auf die verschiedenen Tools wird an dieser Stelle nur kurz eingegangen, da sie bereits in Kapitel 3.3.1 ausführlicher erläutert wurden. Sie implementieren jeweils einige Dinge, die sie brauchen, um ihre Aufgabe zu erfüllen. Besondere Aufmerksamkeit sind in diesem Zusammenhang folgenden zwei Besonderheiten zu schenken:

[2] Gemeinsam erarbeitet mit Jirka Dell'Oro-Friedl. s. https://github.com/JirkaDellOro/FUDGE/issues/27

ToolMove ist sowohl ein Sub-Tool von ToolSelect als auch von ToolTransform. In Ersterem wird es als Komfortfunktion eingebaut, um für simple Bewegungsaktionen nicht das Werkzeug wechseln zu müssen, in Letzterem ist es inhaltlich zugehörig sinnvoll eingebaut. Die modulare Bauweise der Tools erlaubt dies.

Die zweite Besonderheit ist in ToolCreateShapes zu finden, in der ein ähnliches System zur Registrierung der Shapes verwendet wird wie es auch schon für die Tools im Allgemeinen zur Anwendung kommt. So können ganz einfach neue Shapes zur Verwendung hinzugefügt werden.

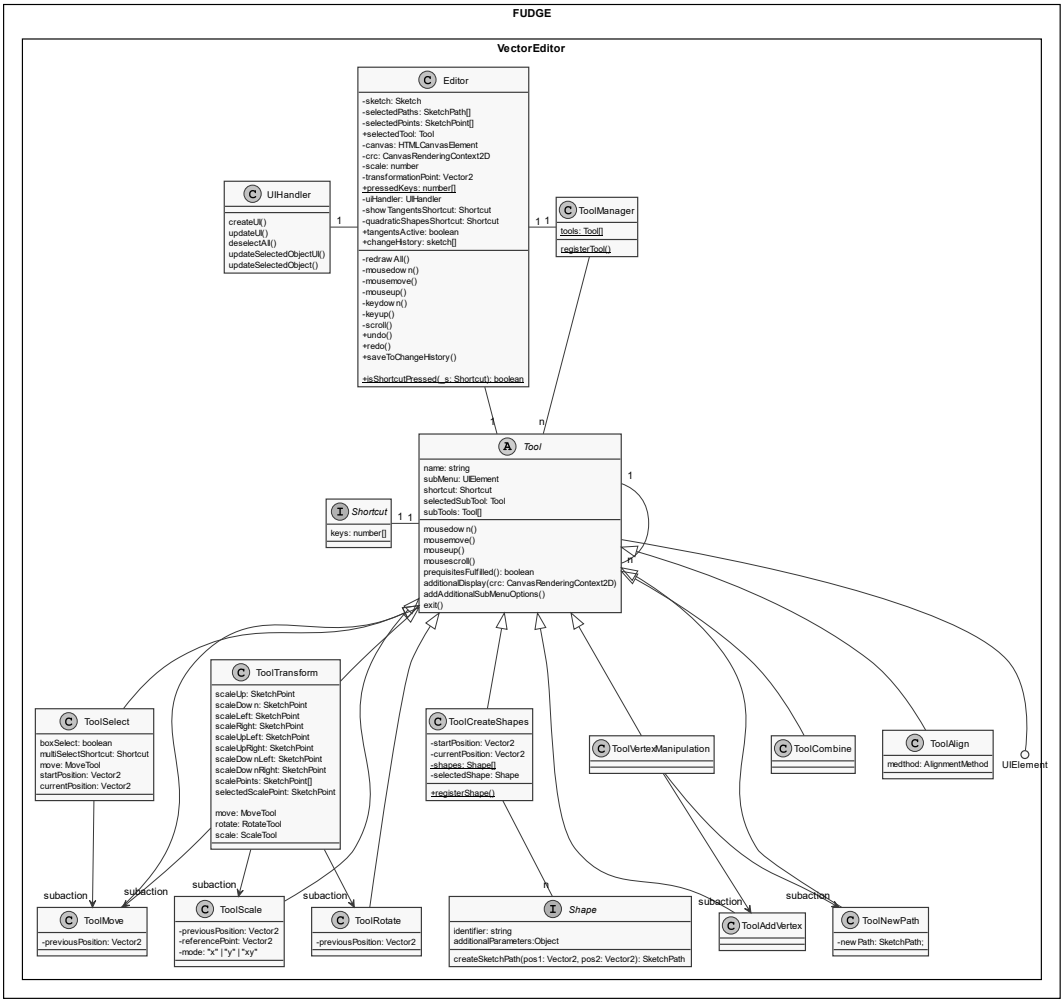

Abbildung 3-7 Klassendiagramm VectorEditor

3.4 Weitere Überlegungen

3.4.1 Verhalten der Tangenten beim Verschieben des Vertex

Während den Testläufen kam die Frage auf, ob und wenn ja, wie sich die Tangentenpunkte im Verhältnis zu ihrem aktivierten Vertex mitbewegen sollen, wenn die Tangentenpunkte nicht explizit eingeblendet sind. Dabei wurden drei verschiedene Ansätze betrachtet: keine Bewegung (1), eine absolute Kopie der Vertex Bewegung durch die Tangentenpunkte, so dass der Positionsunterschied konstant bleibt (2) sowie eine relative Bewegung in Abhängigkeit zum eigenen Vertex sowie zum zugewandten Vertex, so dass der Positionsunterschied zu den beiden Vertices relativ gesehen gleich bleibt (3).

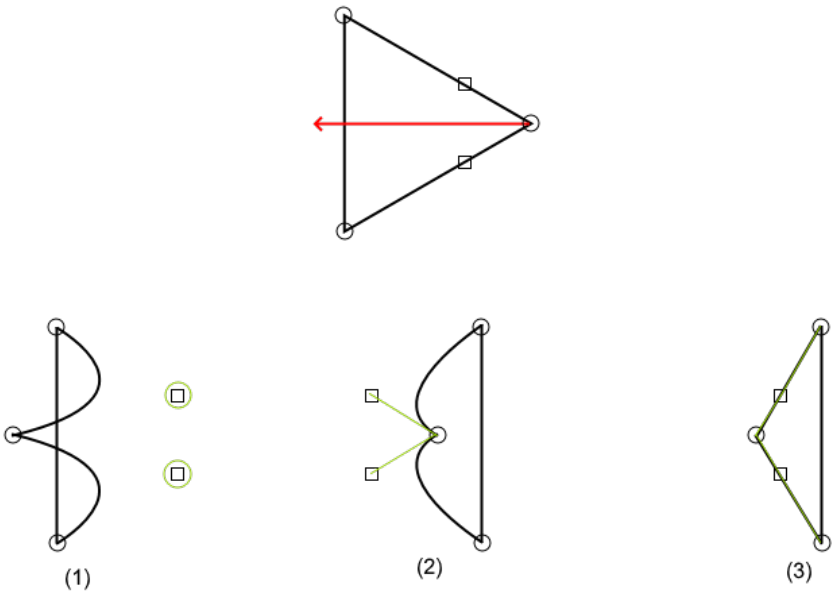

Abbildung 3-8 Tangentenbewegung am Beispiel eines Dreiecks

In Rot die durchgeführte Bewegung des Vertex, in Grün die Referenzen nach denen sich die Tangentenpunkte ausrichten.

Wie in Abbildung 3-8 zu sehen ist, produzieren diese drei Methoden sehr unterschiedliche Ergebnisse. Methode 1 wird für die Bewegung genommen, bei der die Tangentenpunkte explizit eingeblendet sind, da hier die Intention einer unabhängigen Bearbeitung angenommen werden kann. Was die anderen beiden Methoden angeht, so scheint für ein Dreieck zunächst Methode 3 am besten zu sein, da so die Dreiecksform erhalten bleibt. Bei dieser Vorgehensweise gibt es dann jedoch weitere Probleme, wenn die Tangentenpunkte nicht genau auf der

Verbindungslinie zwischen den beiden Vertices liegen. In der Anwendung ist dieser Fall eher selten, da ohne die Aktivierung der Vertices die Tangentenpunkte ignoriert werden und so gerade Verbindungslinien entstehen. Wenn lediglich gerade Linien gemalt werden sollen, müssten die Vertices also gar nicht aktiviert werden.

Bei genauerer Betrachtung von Methode 3, bieten sich zwei weitere Vorgehensweisen an: Die individuelle Struktur der Linie in Abhängigkeit zwischen den beiden Punkten wird beibehalten, wodurch aber die vorher beibehaltene grundsätzliche Form des Objektes wieder verändert wird, so dass der eigentliche Grund für die Weiterführung der Methode 3 nicht mehr zum Tragen kommt (3.1). Oder die für das Dreiecksbeispiel intuitivere Lösung, bei der der Wert an der mittleren Achse gespiegelt wird (3.2). Dies hat aber dann auch wieder den Nachteil, dass an einem eigentlich arbiträr gewählten Punkt der Abstand umgekehrt werden muss.

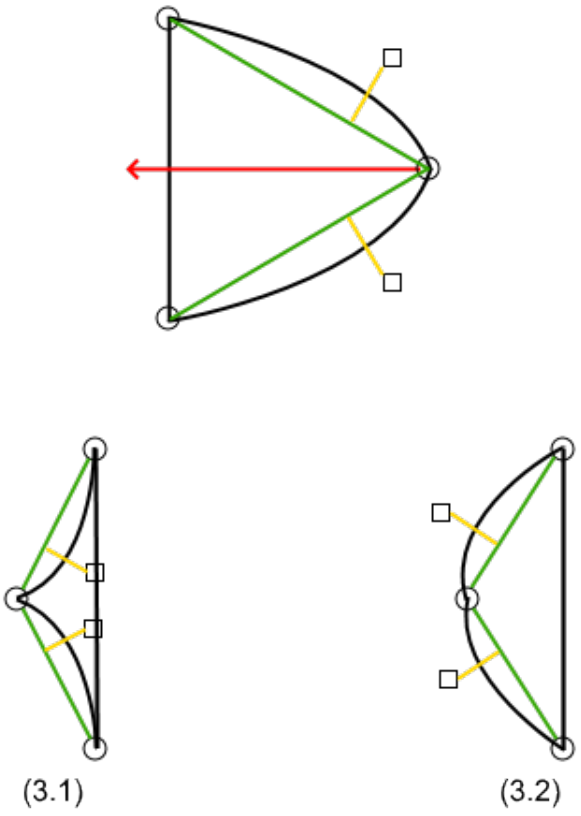

Abbildung 3-9 Tangentenbewegung am Beispiel eines Dreiecks, Fortführung von Methode 3

Eigentlich ist Methode 3.1 die mathematisch korrekte Methode zur Verschiebung der Vektoren bei Betrachtung einzelner Linien und 3.2 verliert, sobald kein Dreieck mehr verwendet wird, den Anhaltspunkt zur Spiegelachse.

Somit stellt sich Methode 3 im Ganzen als unbrauchbar heraus, Methode 1 wird in einem anderen Zusammenhang genutzt und damit wird Methode 2 in diesem Fall verwendet. Auch die Untersuchung von vergleichbaren Programmen (Adobe Illustrator, Adobe Animate) ergibt, dass diese nach der zweiten Methode handeln.

3.4.2 Text

Frühere Entwürfe des Vektoreditors sahen unter anderem noch vor, ein Textelement einbinden zu können. Dieses wurde allerdings im Laufe der Zeit verworfen, da Text nicht Teil der Textur, sondern Teil des UI ist. Dieser UI-Text soll auch in der Szene und damit auch auf Objekten platzierbar sein, damit wäre eine Textintegration eine überflüssige Dopplung.

3.4.3 Ebenen oder Objekte

Viele Grafikeditoren, besonders im Pixelgrafikbereich aber auch im Vektorgrafikbereich, erlauben es dem Nutzer mehrere Ebenen anzulegen sowie auf diesen unabhängig voneinander zu arbeiten, innerhalb derer allerdings alles zusammenhängt. Dies hat den Vorteil, dass so sich überschneidende Objekte auf der gleichen Ebene programmatisch als ein Objekt behandelt und diese somit kombiniert werden können.

Da im Fall von FUDGE jedoch ein Sketch als eine Textur genutzt wird, gehen die einzelnen Ebenen dadurch verloren und um sich unterschiedlich verhaltende Ebenen nutzen zu können, müssen sowieso mehrere FUDGE Objekte genutzt werden. Aufgrund dessen kann die Komplexität der Ebenen auch entfernt werden. Stattdessen wird die Nutzung einer objektbasierten Struktur bevorzugt, bei der alle Zeichenobjekte immer unabhängig voneinander sind und auch unabhängig voneinander bewegt werden können. Es gibt lediglich eine interne Reihenfolge, die festlegt, ob ein Objekt vor einem anderen Objekt im Sketch gezeichnet wird oder dahinter. Dieser Wert ist durch den Nutzer anpassbar.

3.4.4 Verbesserte Möglichkeiten für erfahrene Nutzer

Wie bereits in den Tools (s. Kapitel 3.3.1) angesprochen, sollen sowohl anfängliche als auch fortgeschrittene Nutzerbedürfnisse abgedeckt werden. Dafür wurden Konzepte der Tastatursteuerung, gemeinsam mit Lea Stegk, entwickelt.

Neben den üblichen Hotkeys/Shortcuts zur Auswahl einzelner Funktionen oder Tools, sollen manche Funktionen auch eingeschränkt werden können. Die, von den so für den FUDGE Editor entwickelten Konzepte, für den Vektoreditor relevanten Anforderungen sind hierbei im Besonderen die Einrastfunktion. Diese erlaubt es dem Nutzer die Interaktion bei manchen Aktionen auf bestimme Achsen zu beschränken. So könnte z. B. die Bewegung eines Objektes auf die horizontale Achse beschränkt werden und so versehentliche Verschiebungen auf der vertikalen Achse vermieden werden.

3.4.5 Positionen und Größe

Die Positionen der verschiedenen Sketch-Objekte orientieren sich an den Pixeln bei einem Zoom von eins. Dabei bildet der Ankerpunkt den Ursprung bzw. den Nullpunkt, von dem aus sämtliche Positionen berechnet werden. Dieser Ankerpunkt befindet sich zu Beginn in der Mitte der Bearbeitungsfläche und die Nutzer können daran ihre Objekte ausrichten. Um mit den Standards des Canvas sowie der Pixelbezifferung bei Bildschirmen nicht zu brechen, ist die positive x-Achse nach rechts und die positive y-Achse nach unten ausgerichtet.

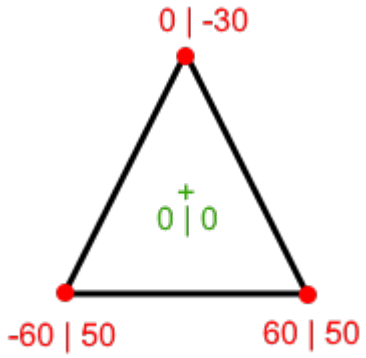

Abbildung 3-10 Beispielkoordinaten für ein Dreieck um den Ursprung herum

3.4.6 Undo

Eine wichtige Fähigkeit eines jeden Systems, besonders bei solchen mit Nutzerinteraktionen, ist es, geschehene Aktionen ungeschehen machen zu können. Dafür ist zu diesem Zeitpunkt kein ausgeklügeltes System vorhanden, stattdessen wird nach der „Holzhammermethode“ nach jeder nicht trivialen Änderung im System eine Momentaufnahme des gesamten Sketches in Form eines JSON Strings im Zwischenspeicher gespeichert, welche bei Bedarf wiederhergestellt werden kann. Dies ist noch ausbaufähig und kann in Zukunft

durch ein ausgeklügeltes System, das lediglich die Änderungen speichert, ersetzt werden, um Speicher- und Rechenleistung zu verringern.

3.4.7 Scrollverhalten

Die Zeichenfläche soll frei bewegbar und (fast) frei skalierbar sein. Die Skalierung wird durch die Betätigung des Scrollrades durchgeführt. Dabei wird um die aktuelle Mausposition herum der sichtbare Ausschnitt des Sketches vergrößert bzw. verkleinert. Dies ist nicht unbegrenzt möglich, sondern es gibt Skalierungsgrenzen, die nicht über- bzw. unterschritten werden können. Die Anfasser der Objekte verändern ihre Größe dabei nicht, so dass die Änderungen stets mit dem gleichen Komfort durchgeführt werden können.

Bewegung kann neben der rechten Maustaste auch durch das Scrollrad erreicht werden. Durch das Halten von Aktionstasten wie STRG oder SHIFT sowie durch das einrasten (s. Kapitel 3.4.4) von Achsen, kann die Zeichenfläche auch horizontal und vertikal durch scrollen verschoben werden.

4 Konzeption Animationseditor

4.1 Anforderungen

Über den Animationseditor soll es dem Nutzer ermöglicht werden, alle animierbaren Eigenschaften eines FUDGE Objektes animieren zu können. Dazu zählen insbesondere die Komponenten, z. B. die Transform Komponente, welche die Position, Rotation und Skalierung des Objektes angibt. Animierbare Eigenschaften sind standardmäßig erstmal nur Zahlen, da diese über einen Zeitraum mit Funktionen verändert werden können. In diesem speziellen Fall sollen aber auch Wahrheitswerte sowie Enumeratoren über Zeit verändert werden können, um die Möglichkeiten durch eine Animation offen zu halten. Dabei soll der Animationseditor es so einfach wie möglich machen, solche Animationen anzulegen und zu bearbeiten.

Die Mutable Klasse beinhaltet eine Schnittstelle für einen speziellen Animations-Mutator (s. Kapitel 3.2). Jede Klasse in FUDGE, die sich von der Mutable Klasse ableitet, stellt sich oder einen Teil ihrer selbst damit als animierbar dar. Das bedeutet im Umkehrschluss, dass der Animationseditor so aufgebaut werden muss, dass er mit jedem möglichen Mutator Objekt auskommen muss.

Zum Animationseditor und der damit verbundenen Animationsstruktur gehört auch eine Möglichkeit diese Animation einem Objekt zuzuordnen, zur Designzeit zu verknüpfen und zur Laufzeit zu verwenden. Obwohl die Animator-Komponente nicht direkt Teil dieser Thesis ist, soll auch sie in der Planung berücksichtigt werden, um später einen reibungslosen Ablauf zu gewährleisten.

Außerdem soll es innerhalb der Animationen möglich sein, sowohl Labels (Beschriftungen) als auch Animationsevents zu erstellen. Ein Animationsevent stellt dabei die Möglichkeit zur Verfügung, zu einem bestimmten Zeitpunkt in der Animation ein individuelles Event auszulösen, auf welches dann innerhalb von verknüpften Skripten reagiert werden kann. So kann z. B. beim Erreichen des höchsten Punktes in einer Sprunganimation eine Funktion zur Erschaffung von Partikeln aufgerufen werden.
Ein Label stellt eine Art Sprungmarke dar, durch die innerhalb von Animationen andere Animationen angesteuert werden können. Dadurch sind direkte

Steuerungen von oben in der Hierarchie möglich, ohne dass zusätzliche Skripte verwendet werden müssen.

4.2 Klassenstruktur Animation

Die Animation grenzt sich gegenüber anderen Klassen durch ihren eigenen Namensraum ab, auch um Zugehörigkeiten zu bündeln.

Animation
Das Kernstück der Animation bildet die in der gleichnamigen Klasse realisierte Animation. Sie vereint alle für die Animation relevanten Daten und Objekte unter sich. So speichert Sie neben dem MutatorForAnimation auch die zugehörigen Sequenzen in einem assoziativen Array, um diese so den richtigen Attributen wieder zuordnen zu können. Neben einigen weiteren Zahlen zur Berechnung des aktuellen Zustandes sowie Listen von Labels und Events, findet sich hier auch eine Einstellung zum Abspielmodus der Animation (s. Kapitel 4.2). Die Dauer der gesamten Animation ist über den Key mit dem höchsten Zeitwert festgelegt.

PLAYMODE
Diese Enumeration gibt an, in welcher Form die Animation abgespielt werden soll. Mögliche Werte und ihre Bedeutung sind:
INHERIT: Die Animation nimmt den gleichen Abspielmodus an wie das Elternobjekt.
LOOP: Die Animation wird in Dauerschleife wiederholt (Standardverhalten).
PINGPONG: Am Ende der Animation wird die Animation rückwärts abgespielt bis sie wieder am Anfang ankommt, wo es dann wieder vorwärts geht.
PLAYONCE: Die Animation wird einmal durchlaufen und bleibt am Ende auf dem letzten Key stehen.
PLAYONCESTOPAFTER: Die Animation wird einmal durchlaufen und bleibt am Ende auf dem ersten Key der Animation stehen, geht also einen Schritt weiter als PLAYONCE.
REVERSELOOP: Wie LOOP, mit dem Unterschied, dass die Animation rückwärts abgespielt wird.
STOP: Bleibt immer auf dem gesetzten Wert stehen. Dies ist sehr nützlich, wenn die über die Labels verknüpften Einzelbilder angesteuert werden sollen.

AnimationEvent
Das AnimationEvent wird durch seinen Namen identifiziert und kann dadurch

abgefangen werden. Außerdem speichert es den Zeitpunkt innerhalb der Animation zu dem es ausgelöst werden soll. In dem Moment, wo dieser Zeitpunkt überschritten wird, wird das Event auf dem Animationsobjekt selbst ausgelöst.

Label

Ähnlich wie das AnimationsEvent speichert auch das Label einen Namen zur Identifikation sowie einen Zeitpunkt an dem es angeheftet ist. Es wird zur Verwendung zusammen mit allen registrierten Labels in eine Enumeration umgewandelt.

Sequence

Die Sequence dient der Verwaltung der Keys, welche einer Eigenschaft zugeordnet werden.

Key

Ein Key beschreibt den Zusammenhang zwischen einem Wert und einem Zeitpunkt, also einen festgelegten Punkt in der Animation eines Attributes. Außerdem speichert er eine Referenz auf die ein- und ausgehenden Funktionen, welche die Veränderung zwischen diesem und dem folgenden bzw. vorherigen Punkt beschreibt. Zudem werden die Steigungen, mit denen die Funktionen den Punkt betreten oder verlassen, gespeichert. Diese sind private Attribute, da eine Änderung dieser immer auch eine Neuberechnung der betroffenen Funktionen mit sich bringen muss.

Function

Durch diese werden die Werte der Eigenschaft zwischen zwei festgelegten Punkten berechnet. Die Funktionen beschreiben eine Formel 3. Grades welche zu jeder Zeit einen Wert zurückliefert. Wie diese Funktion zustande kommt, ist in Kapitel 4.3 genauer erläutert.
Die Funktionen können vollständig durch die zugehörigen Keys berechnet werden, darum werden diese nicht abgespeichert, sondern beim Start der Anwendung vorberechnet.

ComponentAnimator

Die Komponente, welche die Animationen eines Objektes verwaltet und ansteuert. Sie erteilt der Animation den Befehl zum Update und stellt auch Möglichkeiten zu Geschwindigkeitssteuerung und Animationsübergängen bereit.

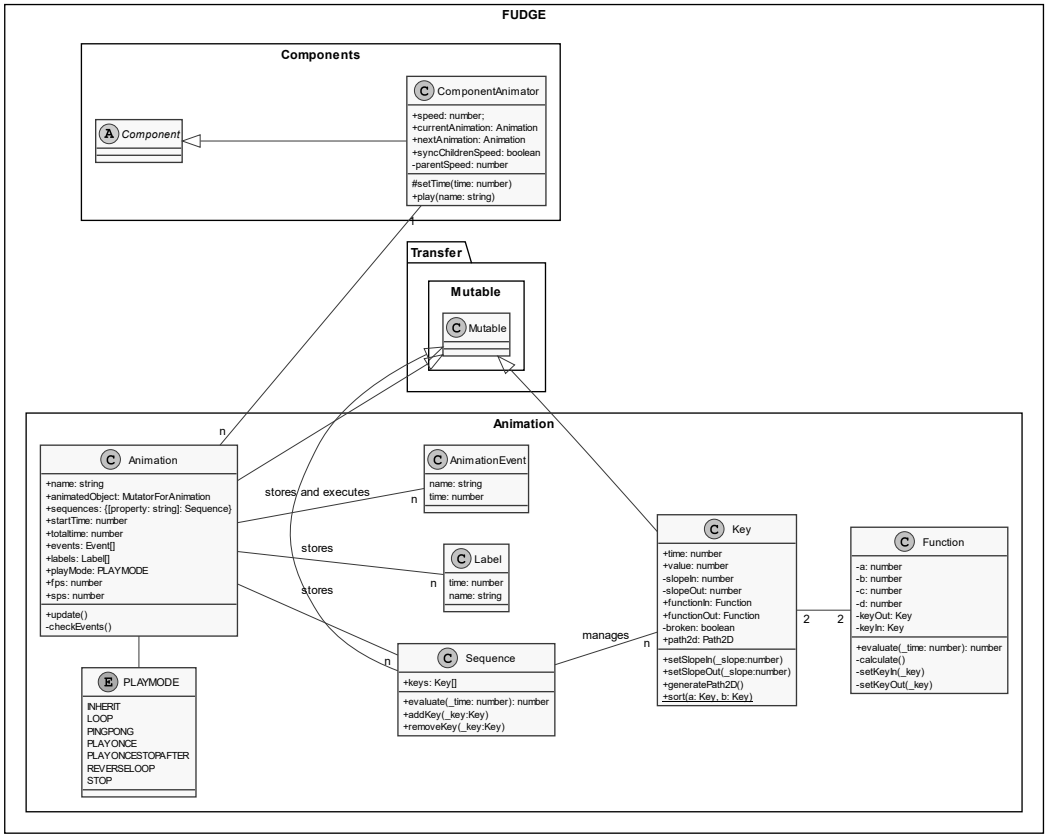

Abbildung 4-1 Klassendiagramm AnimationMathematische Grundlagen der Übergangsformeln

4.3 Mathematische Grundlagen der Übergangsformeln

Um die Übergänge zwischen den verschiedenen festgelegten Punkten so individuell anpassbar wie möglich zu machen, muss die eingehende sowie die ausgehende Steigung frei festlegbar sein. Dies wiederum führt zu dem Problem der Berechnung der dazwischen liegenden Werte.

Da jedem Zeitwert allerdings nur ein Wert zugeordnet werden können muss, können wir von einer rationalen mathematischen Funktion ausgehen. Um die Funktion durch zwei voneinander unabhängige Punkte mit voneinander unabhängigen Steigungen, zu berechnen, kann eine Funktion dritten Grades herangezogen werden.

$$f(x) = ax^3 + bx^2 + cx + d$$

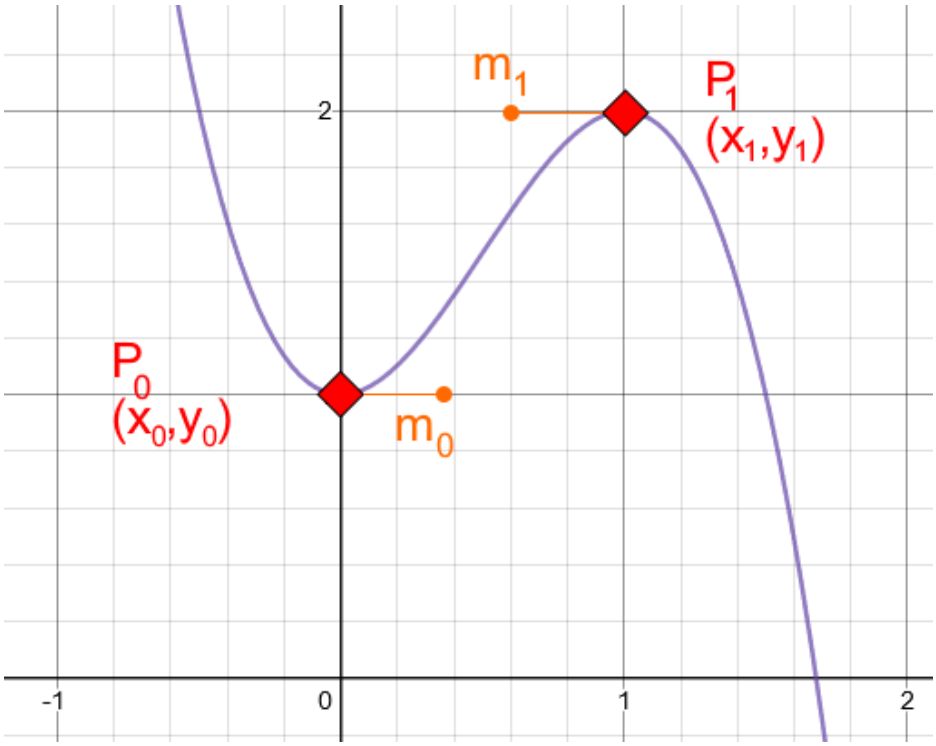

Abbildung 4-2 Grafische Darstellung der zu findenden Funktion

Im vorliegenden Anwendungsfall ist x die Zeit und y der Wert. Dabei ist x gegeben und y ist gesucht. Um y berechnen zu können, müssen also a, b, c und d gefunden werden.

Um die in Abbildung 4-2 dargestellte Funktion allgemein zu lösen, muss ein Gleichungssystem aufgebaut werden. Um vier Variablen berechnen zu können, muss das Gleichungssystem aus vier Gleichungen bestehen. Die ersten beiden können durch Einsetzen der Punkte P_0 und P_1 in die Gleichung dritten Grades herausgefunden werden.

$$y_0 = \mathrm{a}x_0^3 + bx_0^2 + cx_0 + d$$
$$y_1 = \mathrm{a}x_1^3 + bx_1^2 + cx_1 + d$$

Die weiteren Punkte können über die Steigung m_0 und m_1 berechnet werden. Dazu wird von der allgemeinen Funktion dritten Grades die Ableitung gebildet, da diese die Steigung angibt und somit an der Stelle x_0 den Wert m_0 besitzt.

$$f'(x) = 3ax^2 + 2bx + c$$

Durch Einsetzen von m und x können die fehlenden Gleichungen berechnet und das vollständige Gleichungssystem aufgestellt werden.

$m_0 = 3\mathrm{a}x_0^2 + 2\mathrm{b}x_0 + c$
$m_1 = 3\mathrm{a}x_1^2 + 2\mathrm{b}x_1 + c$

Das vollständige Gleichungssystem lautet also:

$$\begin{cases} y_0 = ax_0^3 + bx_0^2 + cx_0 + d \\ y_1 = ax_1^3 + bx_1^2 + cx_1 + d \\ m_0 = 3ax_0^2 + 2bx_0 + c \\ m_1 = 3ax_1^2 + 2bx_1 + c \end{cases}$$

Die Lösung dieses Systems nach a, b, c und d ist:

$$\begin{cases} a = \frac{(m_0+m_1)(x_0-x_1)-2y_0+2y_1}{(x_0-x_1)^3} \\ b = \frac{-m_0(x_0-x_1)(x_0+2x_1)+m_1\left(-2x_0^2+x_1x_0+x_1^2\right)+3(x_0+x_1)(y_0-y_1)}{(x_0-x_1)^3} \\ c = \frac{m_1x_0(x_0-x_1)(x_0+2x_1)-x_1\left(m_0\left(-2x_0^2+x_1x_0+x_1^2\right)+6x_0(y_0-y_1)\right)}{(x_0-x_1)^3} \\ d = \frac{(x_0-3x_1)y_1x_0^2+x_1(x_0(x_1-x_0)(m_1x_0+m_0x_1)-x_1(x_1-3x_0)y_0)}{(x_0-x_1)^3} \end{cases}$$

Durch Verschiebung auf der x-Achse kann $x_0 = 0$ gesetzt werden. Durch Einsetzen dieser Verschiebung in die obere Formel lässt sich diese deutlich vereinfachen:

$$\begin{cases} a = \frac{(-x_1)(m_0+m_1)-2y_0+2y_1}{(-x_1)^3} \\ b = \frac{m_1-m_0-3ax_1^2}{2x_1} \\ c = m_0 \\ d = y_0 \end{cases}$$

Damit lassen sich nun sehr schnell und mit geringem Rechenaufwand a, b, c und d aus den zwei Punkten mit ihren zugehörigen Ein- und Ausgangssteigungen berechnen. Das bedeutet, dass die Formeln nicht gespeichert werden müssen, sondern beim Laden der Animation zur Laufzeit berechnet werden können. Außerdem wird, durch die vorsorgliche Berechnung der Parameter, der laufzeitkritische Teil der Berechnungen noch schneller ausgeführt.

4.4 Aufbau Animationseditor

4.4.1 User Interface

Der Editor selbst soll die Bearbeitung der Keys so einfach und unkompliziert wie möglich gestalten. Dafür sollen die Erstellung und Manipulation der Keys, als auch die Einstellungen der Steigungen, so benutzerfreundlich wie möglich gestaltet werden.

Zur Visualisierung der Keys wird ein ähnlicher Ansatz gewählt wie bei den Vertices im Vektoreditor, so dass die Steigungen ähnlich den Tangentenpunkten angezeigt werden. Allerdings nur in der Kurvenansicht, welche die Übergänge zwischen den einzelnen Punkten visualisiert und nicht in der reinen Key-Ansicht.

Um die Bearbeitung nicht unnötig zu verkomplizieren, werden zwei verschiedene Ansichten eingeführt: das Dopesheet sowie die Curve View. Diese sind bis auf die tatsächliche Darstellung der Keys und einer Schaltfläche in der Eigenschaftsliste gleich.

Beide Ansichten beinhalten oben rechts einen Bereich für grundlegende Einstellungen sowie Abspiel- und damit Vorschaumöglichkeiten. Dort können neben dem Namen der Animation auch die Steps Per Second (SPS), also die Anzahl der Abschnitte pro Sekunde in denen die Keys platziert werden können, eingestellt werden. Diese Abschnitte sollen als Einrastfunktion dienen, um es dem Nutzer einfacher zu machen seine Keys zum gleichen Zeitpunkt wie andere Keys zu platzieren. Wird dieser Wert umgestellt, so verrutschen die bereits platzierten Keys auf den am nächsten gelegenen Abschnitt. Die SPS stellt also nicht die FPS (Frames Per Second, Bilder pro Sekunde) ein, mit denen die Animation abgespielt werden soll, sondern lediglich die Einrasthilfe. Das Feld für die FPS befindet sich darunter, so wie das Feld für die Einstellung des Abspielmodus. Sind die an dieser Stelle eingestellten FPS höher als die globale Einstellung der Anwendung, so wird die FPS zur Laufzeit auf die Anwendungs-FPS limitiert. In diesem Feld befinden sich außerdem die Schaltflächen für Play, Pause, zurück an den Anfang, vor bis zum Ende, einen Abschnitt vor bzw. zurück sowie eine Sprungmarke, welche den Abspielkopf auf den gewählten Abschnitt setzt. Auch befinden sich in diesem Bereich die Knöpfe zum Hinzufügen von Keys, Labels und Events.

Rechts daneben befindet sich die Zeitleiste, in der die SPS, zusammen mit den dazugehörigen Zeitwerten, angezeigt werden. Auf dieser Zeitleiste kann der Abspielkopf verschoben werden. Darunter ist eine Leiste für die Labels und Events.

Unter dem Haupteinstellungs- und Abspielbereich befindet sich die Liste der animierten Eigenschaften sowie ein Button zum Hinzufügen weiterer Eigenschaften. Beim Klicken auf diesen Button kann über ein Pop-up die zu animierende Eigenschaft ausgewählt werden. Die Liste ist nach den Hierarchien im Mutator aufgebaut und einklappbar, so dass Unterpunkte ausgeblendet werden können, um einen besseren Überblick zu gewährleisten. Neben den Eigenschaften befinden sich die Wertefelder, in denen deren Wert, zum Zeitpunkt an dem sich der Abspielkopf befindet, angezeigt wird. Bei Wahrheitswerten ist dies eine Checkbox, bei Enumeratoren ein Dropdown-Menü. Außerdem ist an dieser Stelle eine Schaltfläche, um dieses Element und alle zugehörigen Keys aus der Animation zu entfernen.

In der rechten unteren Ecke befindet sich ein Knopf zum Umschalten zwischen den beiden Ansichten.

Der Aufbau des UI ist stark an andere Animationseditoren wie Unity oder Blender angelehnt.

4.4.1.1 Dopesheet

Als Dopesheet wird die in Abbildung 4-3 dargestellte Ansicht bezeichnet, welche lediglich die Keys zugehörig zu ihren Werten in Abhängigkeit von der Zeit darstellt. Sie dient dabei vor allem der Darstellung des Gesamtzusammenhangs aller Keys im zeitlichen Ablauf. Dabei stehen die einem zu animierenden Element zugehörigen Keys auf einer horizontalen Ebene mit dem Element selbst. Gruppierungen haben im ausgeklappten Zustand keine Keys, im zusammengeklappten Zustand zeigen sie einen Key pro Abschnitt an, in dem einer der Unterelemente der Gruppierung einen Key hat.

In der Ansicht eines einzelnen Elementes wird außerdem eine Schaltfläche mit einem Schloss hinzugefügt, durch das die Bearbeitung der Werte dieses Elementes ein- und ausgeschaltet werden kann.

4.4.1.2 Curve View

Im Curve View wird, wie in Abbildung 4-4 gezeigt, neben den Keys noch eine Legende eingebaut. Die Keys sind in dieser Ansicht nicht auf einer Ebene mit den zugehörigen Eigenschaften, sondern vertikal passend zur Legende angeordnet. Die visuelle Zuordnung zu den Eigenschaften erfolgt über eine Farbcodierung. Die Keys sind durch die visualisierten Übergangsfunktionen verbunden. Wird ein Key angewählt, so werden sowohl die zugehörigen Steigungen daneben eingeblendet als auch das Eigenschaftsfenster in der rechten oberen Ecke, in welchem genauere Einstellungen zu den Steigungen vorgenommen werden können. Unter anderem kann hier das Verhalten der Steigungen in Abhängigkeit von der gegenüberliegenden Steigung festgelegt werden, um einen weicheren Übergang zu gewährleisten. Als zusätzliche Komfortfunktion werden Schaltflächen zur Ausrichtung der Steigung auf den benachbarten Key hinzugefügt.

Wahrheitswerte werden als 1 oder 0 auf der Legende angezeigt, Enumeratoren werden nicht angezeigt.

Einzelne Kurven können aus oder eingeblendet werden, indem auf den farbigen Kreis neben dem Wertefeld geklickt wird. Durch Faltung ausgeblendete Elemente werden grundsätzlich nicht angezeigt.

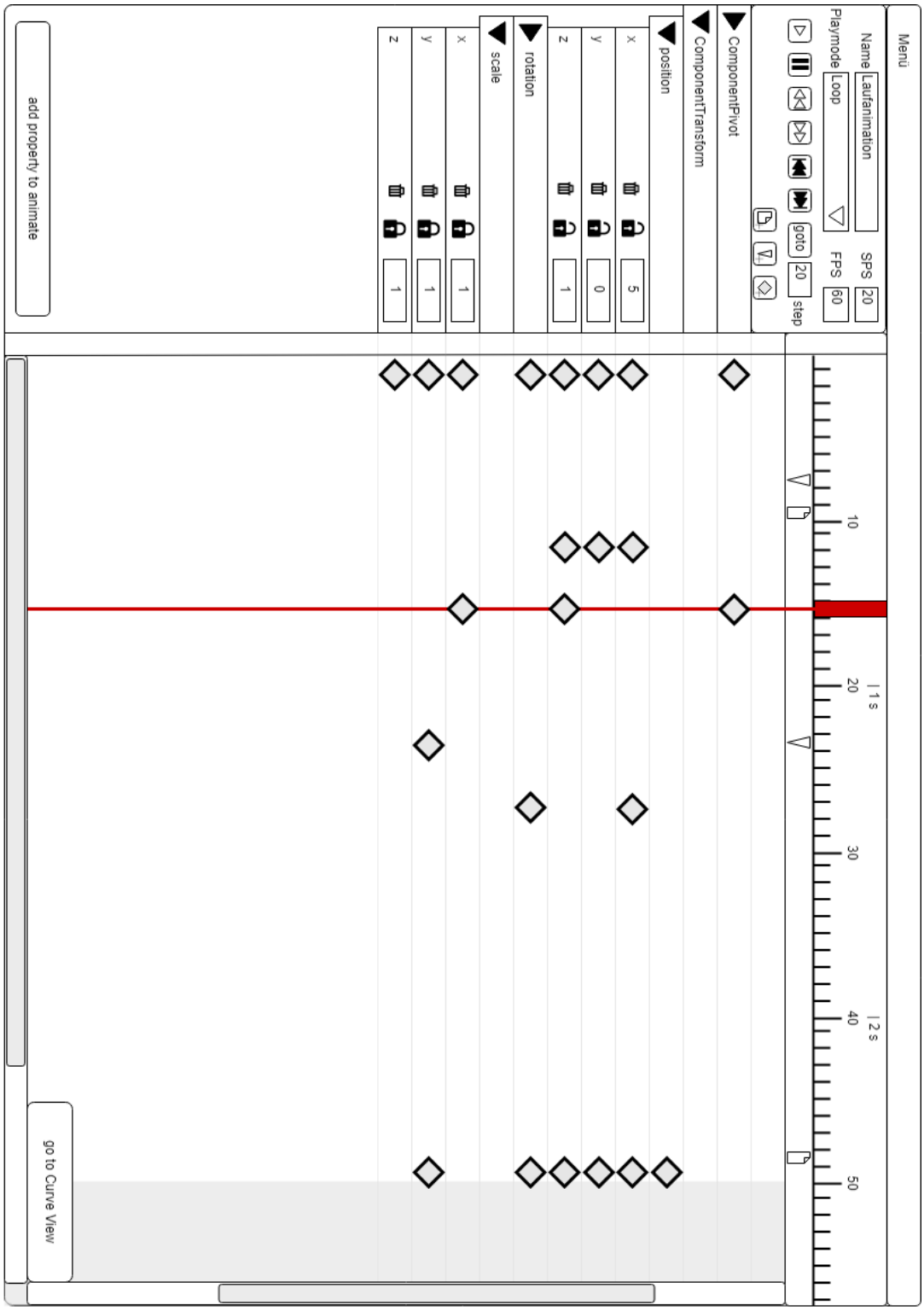

Abbildung 4-3 Konzept Dopesheet

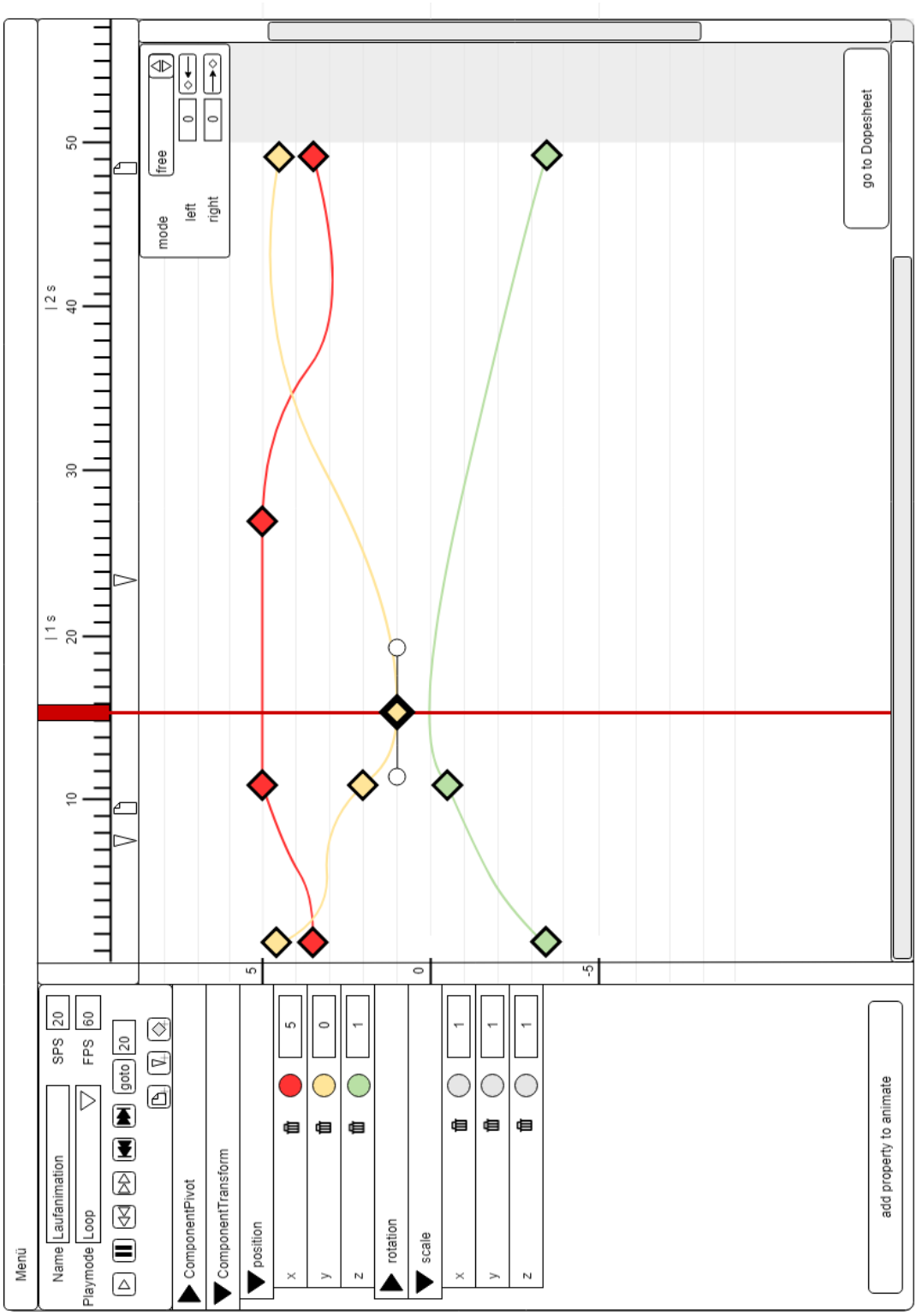

Abbildung 4-4 Konzept Curve View

4.4.2 Klassenstruktur Animationseditor

Die Struktur des Animationseditors ist im Vergleich zu dem Vektoreditor stark reduziert, da dieser zwar viele Optionen und Schaltflächen, aber nicht so viele unterschiedliche Bearbeitungsmöglichkeiten hat. Die Interaktionsmöglichkeiten des Nutzers beschränken sich auf die Input Elemente mit vorgefertigten Aktionen sowie den Keys, welche sich lediglich auswählen und bewegen lassen. Darum wurde hier auf einen ähnlich komplexen Aufbau wie beim Vektoreditor verzichtet.

Auch an dieser Stelle werden Teile des UI Systems von FUDGE verwendet. Diese Verbindungen wurden aus Gründen der Übersichtlichkeit in den Diagrammen weggelassen.

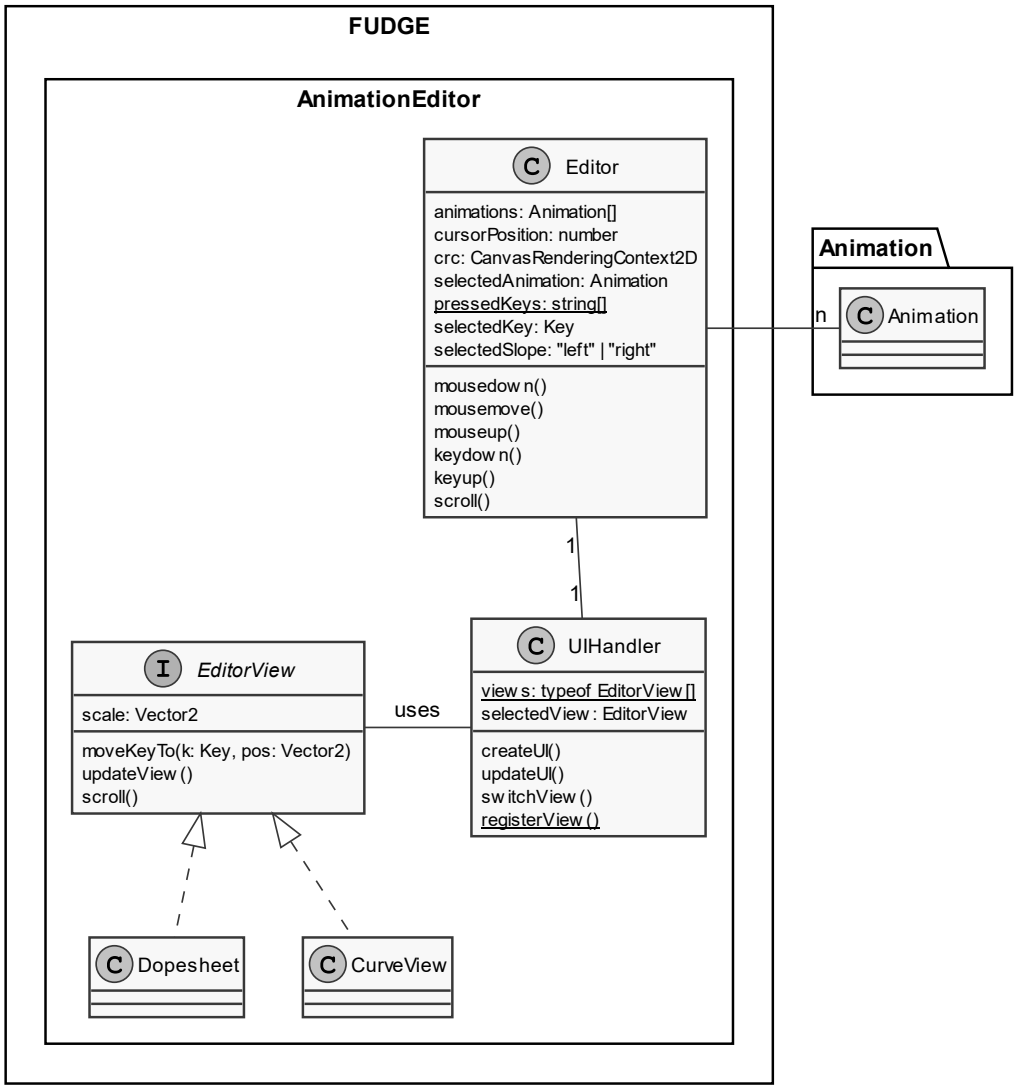

Abbildung 4-5 Klassendiagramm AnimationEditor

Editor

Der Editor bildet wie schon beim Vektoreditor das Kernstück und verwaltet die Animationen sowie andere Werte, wie z. B. die Position des Cursors. Er arbeitet zusammen mit dem UIHandler, um die Grundlage der Funktionalitäten zu bilden. Hier werden außerdem die Nutzereingaben auf dem Canvas, genauer die Verschiebung der Keys und die Views weitergegeben.

UIHandler

Der UIHandler kümmert sich um sämtliche Interaktionen des Nutzers außerhalb des Canvas, insbesondere die Bedienung von HTMLInputElementen. Auch handhabt er die Aktualisierung der Benutzeroberfläche: wenn Elemente hinzugefügt oder entfernt werden, wird dies vom UIHandler verwaltet. Die Views können sich hier ähnlich der Tools im Vektoreditor registrieren und werden so automatisch mit eingebunden.

EditorView

Ein Interface, welches das Hinzufügen von verschiedenen Ansichten erlaubt. Es verlangt von den Views zum einen, ihre Anzeige aktualisieren zu können, und zum anderen die Bewegung von Keys zu behandeln, da diese je nach Ansicht unterschiedlich ausfällt. Diese Bewegung wird von der Editorklasse abgefangen und an den ausgewählten View weitergereicht.

Dopesheet/Curve View

Die bisher geplanten Editoren, welche das EditorView Interface implementieren (s. Kapitel 4.4.1.1 und 4.4.1.2).

4.5 Weitere Überlegungen

4.5.1 Undo

Auch beim Animationseditor soll es die Möglichkeit geben, Geschehenes ungeschehen zu machen. Ähnlich wie beim Vektoreditor, wird auch hier zunächst die Holzhammermethode angewandt, bis sie durch eine bessere Methode ersetzt wird (vgl. Kapitel 3.4.6).

4.5.2 Skalierung und Scrollverhalten

Bei längeren Animationen kann der horizontale wie auch der vertikale Platz schnell zur Einschränkung werden. Darum sollen an der oberen wie rechten Seite des

Editors Scrollbalken angebracht werden, welche eine komfortable Bewegung durch die Ansicht erlaubt. Wie auch schon beim Vektoreditor, soll die Ansicht skalierbar sein. Dabei kann jede Ansicht selbst implementieren, wie diese Skalierung auszusehen hat. Das Dopesheet hat lediglich eine vertikal veränderliche Skalierung wogegen der Curve View auch eine horizontale Skalierung integriert.

5 Umsetzung

Um die resultierende Anwendung so gut wie möglich anhand der Vorgaben von FUDGE umzusetzen, wurde viel Zeit mit Vorbereitung, Planung und Testläufen verbracht. Dabei wurden in vielen verschiedene Richtungen Experimente durchgeführt, selbstverständlich auch solche, die nicht zum Ziel geführt haben. Die besondere Schwierigkeit bestand darin, dass der Kern von FUDGE und die damit verbundenen Gegebenheiten und Umgebungsbedingungen für die längste Zeit noch nicht feststanden und so stets mit Ungewissheit gearbeitet wurde. Somit ging auch ein großer Teil dieser Thesis hinter den Kulissen vonstatten, um ein gemeinsames System zu entwickeln, auf der der eigentliche Inhalt der Thesis dann aufgebaut werden kann.

Die Experimente reichen über Canvas Interaktionen und Testanimationen bis zu WebGL und Electron und können alle auf dem beiliegenden Datenträger eingesehen werden. Es wird an dieser Stelle nicht weiter auf die Experimente eingegangen, da diese keine weiteren Erkenntnisse darlegen können, als jene, welche in Kapitel 3 und 4 bereits ausführlich dargestellt wurden. Die Experimente können auf dem beiliegenden Datenträger eingesehen werden (s. Anhang B

Auf der Basis dieser Experimente wurde die Planung durchgeführt. Dazu gehören insbesondere die bereits besprochenen Klassendiagramme und Skizzen als auch die bisher noch nicht erwähnten Ablaufdiagramme (s. Anhang C). Der Großteil der (Funktions-)Abläufe wurden so bereits komplett geplant. Ein einfaches Beispiel eines Aktivitätsdiagrammes ist in Abbildung 5-1 abgebildet. Damit sollte die Umsetzung, bis auf Kleinigkeiten und unvorhergesehene Probleme, keine große Hürde mehr darstellen, welche allerdings aus Zeitgründen nicht mehr bis zum Abgabezeitpunkt dieser Thesis genommen werden konnte.

Darum ist zum Zeitpunkt der Abgabe dieser Arbeit keiner der beiden geplanten Editoren einsetzbar.

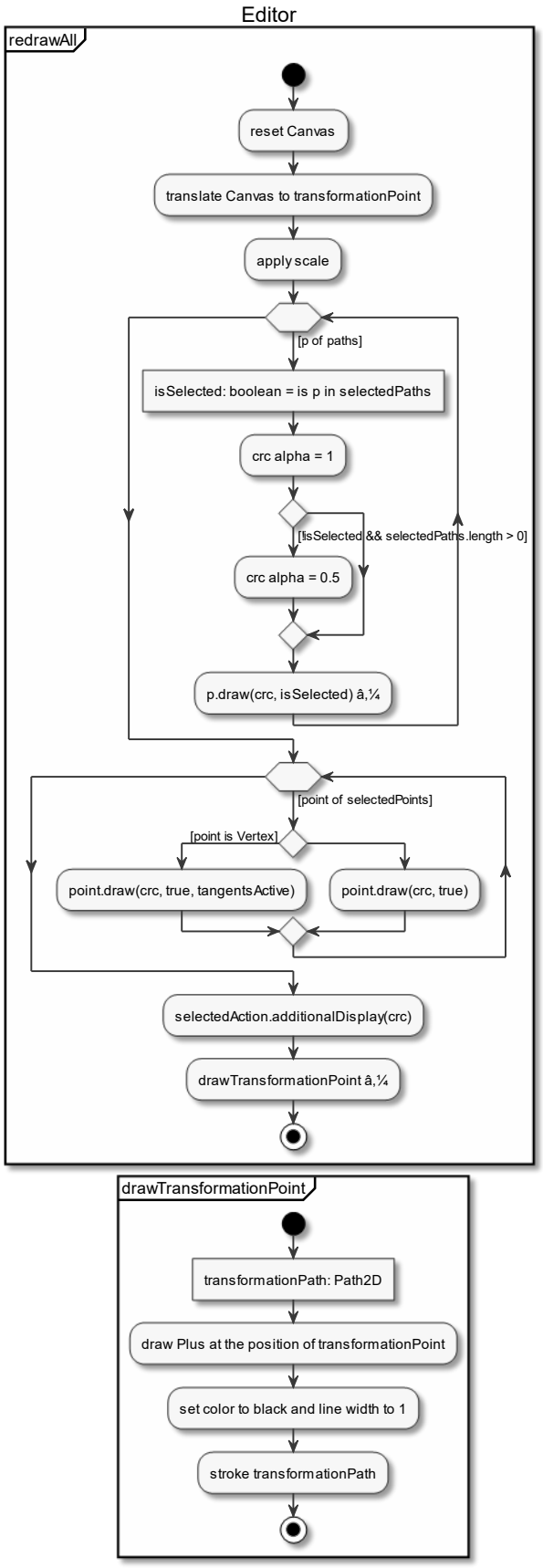

Abbildung 5-1 Anwendungsfalldiagramm der Funktion "redrawAll()" der Klasse VectorEditor.Editor

6 Zusammenfassung

Mit FUDGE soll ein neuartiger Game Editor für die Lehre, insbesondere an der Hochschule Furtwangen, entwickelt werden. Mit seiner Hilfe soll Studenten die Basis der Entwicklung von Spielen und anderen interaktiven Anwendungen beigebracht werden. Auch soll ihnen die Möglichkeit gegeben werden, den Quellcode von FUDGE einzusehen und so auch die inneren Abläufe nachvollziehen zu können.

Damit die Entwicklung von Spielen einfach und vollständig möglich ist, soll FUDGE grafische Editoren beinhalten, welche die Bedienung schnell, einfach und intuitiv gestalten. Zwei dieser Editoren sind der Vektoreditor, zur Erstellung von 2D-Vektorgrafiken zur Nutzung innerhalb von FUDGE als Texturen oder ähnlichem, sowie der Animationseditor zur Erstellung von Animationen für FUDGE Objekte. Die Aufgabe dieser Thesis war es, diese beiden Editoren zu definieren, planen und umzusetzen.

Da FUDGE auf Webtechnologien basiert, sollen die Editoren auch über aktuelle Webtechnologien entwickelt werden. Dabei bieten sich viele Standard-HTML-Elemente an, wie etwa die Zeichenfläche auf dem Canvas oder InputElemente zur Eingabe von Werten.

Beide Editoren wurden ausführlich unter Zuhilfenahme von UML Diagrammen wie Anwendungsfalldiagrammen, Klassendiagrammen und Ablaufdiagrammen durchgeplant und im Kontext von FUDGE eingeordnet. Aus zeitlichen Gründen konnte die Umsetzung zum Ende dieser Arbeit nicht fertig gestellt werden.

Eine große Herausforderung hierbei waren die fehlenden Grundstrukturen, welche erst während dieser Thesis erarbeitet und gelegt wurden. Auch dadurch sind viele Entwicklungen zunächst in eine falsche Richtung gegangen und so fehlte am Ende die Zeit die geplanten Editoren umzusetzen.

7 Fazit und Ausblick

Im Rahmen dieser Arbeit sind ausführliche Pläne für einen 2D-Vektoreditor als auch für einen Animationseditor erarbeitet worden. Es wurden die unterliegenden Strukturen und die benötigten Klassen erdacht und definiert sowie der Großteil der benötigten Methoden geplant, immer unter Berücksichtigung der vorgegebenen Paradigmen der Nutzbarkeit, Erweiterbarkeit und Verständlichkeit.

Beide Editoren sind zum aktuellen Zeitpunkt nicht nutzbar, sollen aber über die kommenden Wochen implementiert und an FUDGE angebunden werden. Außerdem wird FUDGE gerade durch die Arbeit anderer in vielen Bereichen erweitert. Dazu gehören Netzwerkanbindung und Server, Sound, Physik sowie ein übergreifendes UI System. Bis zum Anfang des kommenden Semesters soll FUDGE soweit nutzbar sein, dass es in der Lehre angewendet werden kann.

Ich persönlich habe viel über Webtechnologien, Projektplanung und Spieleeditoren gelernt und konnte so mein bereits vorhandenes Wissen vertiefen und erweitern sowie mir neues Wissen aneignen. Alles in allem hat mir das Projekt viel Spaß gemacht und ich werde mich persönlich daran beteiligen, dass FUDGE bis zum nächsten Semester einsatzbereit ist.

8 Literatur- und Quellenverzeichnis

[1] J. Dell'Oro-Friedl. (2019, Mai, 11). *FUDGE.* [Online]. [Restricted Access].
Verfügbar unter: https://github.com/JirkaDellOro/FUDGE.

[2] GAME. (ohne Datum). Jahresreport der deutschen Games-Branche 2018. [Online].
Verfügbar unter: https://www.game.de/publikationen/jahresreport-der-deutschen-games-branche-2018/

[3] Newzoo. (ohne Datum). Top 100 Countries/Markets by Game Revenues. [Online].
Verfügbar unter: https://newzoo.com/insights/rankings/top-100-countries-by-game-revenues/

[4] Gameswirtschaft. (2018, April, 16). Umsatz-Vergleich 2017: Games deutlich vor Kino und Musik. [Online].
Verfügbar unter: https://www.gameswirtschaft.de/wirtschaft/umsatz-vergleich-2017-games-kino-musik/

[5] WHATWG (Apple, Google, Mozilla, Microsoft). (2019, Mai, 10). HTML. Living Standard. [Online].
Verfügbar unter: https://html.spec.whatwg.org/

[6] S. Faulkner et al. (2017, Dezember, 14). HTML 5.2. W3C Recommendation. [Online].
Verfügbar unter: https://www.w3.org/TR/html52/

[7] Microsoft. (2019, Mai, 15). Typescript. [Online].
Verfügbar unter: http://www.typescriptlang.org/

[8] ECMA International. (2018, Juni). ECMAScript® 2018 Language Specification. [Online].
Verfügbar unter: https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[9] J. Dell'Oro-Friedl. (2019, April, 19). FUDGE. General Guidelines. [Online]. [Restricted Access].
Verfügbar unter https://github.com/JirkaDellOro/FUDGE/wiki/General-Guidelines

Eidesstattliche Erklärung

Ich, Lukas Scheuerle, versichere, dass ich die vorliegende Arbeit selbstständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

_____ Furtwangen, den 31.05.2019

Anhang

A. Tabelle mit Interaktionsmöglichkeiten des Vektoreditors

Diese Tabelle kann auf dem beiliegenden Datenträger unter VektorEditor_Aktionen.pdf eingesehen werden.

B. Experimente

Alle im Rahmen dieser Arbeit durchgeführten Experimente befinden sich auf dem beiliegenden Datenträger. Diese sind mit README Dateien versehen, welche weiterführende Informationen über die jeweiligen Experimente beinhalten.

C. Aktivitätsdiagramme

Alle Aktivitätsdiagramme, die im Rahmen dieser Arbeit entstanden sind, können auf dem beiliegenden Datenträger eingesehen werden.

D. Klassendiagramme

Alle in dieser Arbeit integrierten Klassendiagramme sind auf dem beiliegenden Datenträger in voller Auflösung sowie als Vektordatei hinterlegt.