Modellierung einer Softwarearchitektur in Virtual Reality mithilfe der Leap Motion

Kevin Döhl

Eine Arbeit zur Erlangung des Akademischen Grades
Bachelor of Science

Universität Bremen

Universität Bremen
Deutschland
15. September 2020

Gutachter
Prof. Dr. Rainer Koschke
Prof. Dr. Gabriel Zachmann

Erklärung

Ich versichere, die Bachelorarbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Bremen, 15.09.2020 

Ort, Datum Unterschrift

Inhaltsverzeichnis

Kapitel 1

Einleitung

Bei der Erschaffung von Software gilt es wie auch bei Gebäuden, sich eine Architektur zu überlegen. Im klassischen Stil erstellt man hierfür u.A. Diagramme, die in der Unified Modeling Language (UML) beschrieben werden. Mit einem UML-Diagramm stellt man die hierarchische Struktur der System/Softwarekomponenten und deren Beziehung untereinander dar. Die Komplexität von UML-Diagrammen kann sehr schnell zunehmen, wodurch sie erst bei genauerer Analyse zu verstehen sind. Ebenso sehen sie dazu auch abstrakt aus, wodurch man erst durch genaues Hinsehen alles versteht. Wie wäre es aber, wenn man die Möglichkeit hätte, solche Diagramme visuell gut und verständlich zu erstellen? Man müsste nicht mehr die aufwendigen Diagramme bauen, sondern nur einzelne Komponenten zusammentragen. Hierfür könnte man eine Welt in der Virtual-Reality nutzen und sich dort eine hierarchische Softwarearchitektur aufbauen.

Das SEE-Projekt der Arbeitsgruppe Softwaretechnik der Universität Bremen ist in der Lage, die komplette Implementierung eines Softwareprojektes graphisch in Virtual-Reality als eine 3D-Stadt darzustellen. Hierfür wird die Spiele-Engine Unity in Kombination mit der Programmiersprache C# verwendet. Die Stadt stellt die Ordnerstruktur und die Dateien in den Ordnern dar. Hierbei werden beispielsweise Kreise oder andere geometrischen Körper gezogen, welche die einzelnen Ordner darstellen und auf den Kreisen befinden sich weitere Kreise, für jeden weiteren Ordner. Auf den Kreisen befinden sich dann einzelne Gebäude, welche die Dateien in jeweiligen Ordner darstellen. Verbindungen zwischen Gebäuden stellen Abhängigkeiten im Quellcode dar.

1.1 Zielsetzung und Aufgabenstellung

Das Ziel der Arbeit ist es, eine Erweiterung für das SEE-Projekt zu erstellen, welches dem Benutzer ermöglicht, eine Softwarearchitektur mit simplen Handgesten während der Laufzeit zu modellieren, abzuspeichern und schon vorhandene Softwarearchitekturen zu laden und weiter zu modellieren. Diese Softwarearchitekturen

sollen dann im späteren Verlauf eines anderen Projektes, als Architekturen für eine *Incremental Reflexion Analysis* (siehe 2.3) zur Verfügung stehen. Diese Analyse prüft eine gegebene Implementierung hinsichtlich der Architekturvorgaben, welche anhand der folgenden Aufgaben umgesetzt werden:

1. Eine Steuerung entwickeln, damit der Benutzer mithilfe von Handgesten Objekte erzeugen und bearbeiten kann.
2. Eine Funktion entwickeln, damit der Benutzer seine gerade erstellte Softwarearchitektur abspeichern kann.
3. Eine Funktion entwickeln, damit der Benutzer eine vorhandene Softwarearchitektur laden und bearbeiten kann.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in 5 Kapiteln: Das erste Kapitel enthält die Einleitung, welche einen Überblick über die Arbeit gibt, sowie die Motivation darlegt. Anschließend werden im zweiten Kapitel Grundlagen über Softwarearchitekturen (siehe 2.1), das SEE-Projekt (siehe 2.2), die *Incremental Reflexion Analysis* (siehe 2.3), die Graph-eXchange-Language (GXL) (siehe 2.4), die Spiel-Engine Unity (siehe 2.5), das Eingabegerät Leap Motion (siehe 2.6) und Tools mit ähnlichen Funktionen (siehe 2.7) erklärt. Im dritten Kapitel wird aufbauend auf den Grundlagen die Designidee der Implementierung erklärt und darauf folgend die Umsetzung der Ideen dargestellt. Abschließend werden in Kapitel 4 die Ergebnisse der Evaluation für die Steuerung mit der Leap Motion dargestellt und mit einem Fazit aus den Ergebnissen wird die Arbeit beendet.

Kapitel 2

Grundlagen

Im folgenden Kapitel werden die Grundlagen dargestellt, die für das verstehen der Arbeit notwendig sind. Hierbei werden die Themengebiete Softwarearchitekturen, das SEE-Projekt, die *Incremental Reflexion Analysis*, die Graph eXchange Language, Unity und der Leap Motion Controller erläutert und Tools beschrieben, die ebenfalls eine Funktion der Archtiekturmodellierung besitzen.

2.1 Softwarearchitektur

Es existieren eine Menge Definitionen von Softwarearchitekturen aber es existiert eine grundlegende und standardisierte Definition des Verbandes *Institute of Electrical and Electronics Engineers*, kurz *IEEE*, aus dem Jahre 2000, welche wie folgt lautet:

> Eine Softwarearchitektur ist die grundlegende Organisation eines Systems verkörpert in seinen Komponenten, deren Beziehungen untereinander und zu der Umgebung und die Prinzipien, die den Entwurf der Evolution leiten.[IEE00]

Somit ist bei der Planung und Aufbau einer Softwarearchitektur zu beachten was für Komponenten genau benötigt werden, ob diese weitere Komponenten beinhalten und wie die verschiedenen Komponenten miteinander in Beziehung stehen. Unter der Beziehung von Komponenten versteht man u.A., dass die Implementierung einer Komponente einen Funktionsaufruf aus der in Beziehung stehenden Komponente ausführt. Hierbei soll ebenfalls auch noch darauf geachtet werden, dass die Softwarearchitektur während der Entwicklungsphase wartbar ist, wodurch sich der Aufwand bei nötigen Änderungen verringert.

2.2 Das SEE-Projekt

Das SEE-Projekt wird innerhalb der Arbeitsgruppe Softwaretechnik an der Universität Bremen unter der Leitung von Prof. Dr. Rainer Koschke entwickelt. Hierbei handelt es sich um ein Tool, was es dem Benutzer ermöglicht, Informationen aus einer Codeanalyse von Softwareprojekten visuell in einer dreidimensionalen Welt mithilfe der Unity-Engine darzustellen. Hierbei werden die Informationen mithilfe einer *Code-City*(vgl. [Lim+19]) dargestellt, wodurch der Benutzer eine bessere Übersicht über die komplexen Informationen aus der Codeanalyse erhalten soll.

In Abbildung 2.1 wird dargestellt, wie der Detailgrad einer Code-City dargestellt und durch Gruppierungen von Elementen angepasst werden kann. Die einzelnen Komponenten werden hier durch Blöcke dargestellt und visualisieren anhand ihrer Größe, Form und Farbe unterschiedliche Eigenschaften. So kann beispielsweise die Darstellung eines Dateisystems eines Softwareprojektes mit einzelnen Blöcken für Dateien und Gruppierungen von Blöcken einen Ordner im Dateisystem realisiert werden. Um weitere Eigenschaften, wie das Alter oder Dateigröße, darzustellen, kann die Farbe oder Größe der Blöcke verwendet werden.

Abbildung 2.1: Vier Unterschiedliche Detailstufen einer Code-City[Lim+19]

Zurzeit werden drei Funktionen im SEE-Projekt unterstützt. Zum einen können statische Informationen einer Software visuell dargestellt werden und zum anderen werden verschiedene Versionen einer Software, sowie das Laufzeitverhalten, anhand einer Animation dargestellt. Also wann beispielsweise welche Klasse oder Datei benutzt wird. Neben dieser Arbeit sind zwei weitere Funktionen in Arbeit: einmal die Funktion, die *Incremental Reflexion Analysis* auf einer Architektur und Implementierung laufen zu lassen (siehe Kapitel 2.3) und eine Steuerung in der Spielwelt

mithilfe von Spracheingaben zu realisieren.

2.3 Incremental Reflexion Analysis

Die durch diese Arbeit entstehenden Softwarearchitekturen sollen dafür genutzt werden, dass ein gegebenes Implementierungsmodell auf diese Architektur abgebildet werden soll. Daraufhin soll die *Incremental Reflexion Analysis*[Kos13] anhand dieser Eingaben die Softwarearchitektur evaluieren und mögliche Unterschiede feststellen. Diese Abbildung wird auch *Mapping* genannt. Diese Analyse baut auf die *Reflexion Analysis* von Murphy *et al.*[MNS01] auf. Die Analyse von Murphy überprüft anhand des Mappings, ob das Implementierungsmodell die Regeln der Softwarearchitektur eingehalten hat. Die Ergebnisse der Prüfung werden durch drei Zustände dargestellt: *convergence*, *absence* und *divergence*. Eine *Convergence* ist vorhanden, wenn das Implementierungsmodell eine Abhängigkeit besitzt, welche auch in der Architektur vorhanden ist, also die Regeln der Architektur eingehalten worden sind. Eine *Absence* ist vorhanden, wenn in der Architektur eine Abhängigkeit existiert, diese aber nicht im Implementierungsmodell vorhanden ist. Zuletzt ist eine *Divergence* vorhanden, wenn das Implementierungsmodell eine Abhängigkeit besitzt, welche aber nicht in der Architektur vorhanden ist. In diesem Fall hat das Implementierungsmodell die Regeln der Architektur gebrochen. Der Verlauf dieser Analyse wird in Abbildung 2.2 dargestellt.
Die Analyse von Murphy hatte ein Problem, welches durch die *Incremental Reflexion Analysis* behoben worden ist. Bei einzelnen Änderungen im Implementierungsmodell, der Architektur oder im Mapping muss die Analyse die komplette Abbildung erneut durchlaufen. Dies kann bei großen Systemen einen großen Aufwand haben. In der *Incremental Reflexion Analysis* von Rainer Koschke[Kos13], wird bei Änderungen nicht erneut die komplette Abbildung betrachtet, sondern nur die Teile, welche von den Änderungen betroffen sind. Hiermit wird ein großer Aufwand erspart und es ermöglicht die Erschaffung von Tools, wie in der parallel laufenden Arbeit von David Wagner, ein Implementierungsmodell auf eine Architektur abzubilden und sofortiges Feedback von der Analyse zu erhalten, ob diese Implementierung und Architektur konsistent sind.

2.4 Graph eXchange Language

Die Graph eXchange Langauge[Ric] (GXL) ist der De-facto-Standard im Bereich der Softwarearchitektur und des Reverse Engineerings um Graphen zu beschreiben. Mithilfe dieses Formates soll es möglich sein, Graphen unabhängig von Plattform, Tools und Programmiersprachen auszutauschen und dessen Informationen zu verarbeiten. GXL basiert auf XML (Extensible Markup Language), einer Auszeichnungssprache die sowohl für Menschen, als auch Maschinen lesbar ist, und durch eine Dokumen-

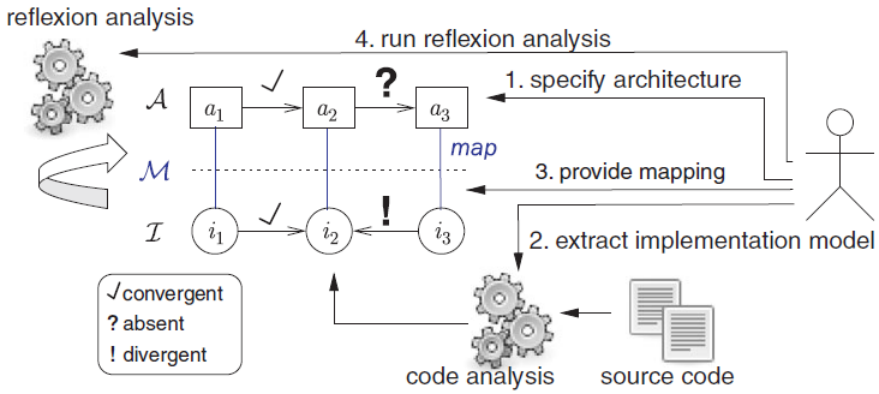

Abbildung 2.2: Verlauf der *Reflexion Analysis*[Kos13]

tentypdefinition (engl. Document Type Definition, DTD)[1] beschrieben wird.

Im SEE-Projekt werden GXL-Dateien benutzt, um Informationen aus einer Codeanalyse abzuspeichern und diese Informationen dann für das Visualisieren einer Implementierung zu benutzen. In dieser Arbeit werden die GXL-Dateien dafür benutzt, Softwarearchitekturen abzuspeichern oder zu laden. Hierfür wird der Inhalt der GXL-Datei in einer internen Graphenstruktur geladen. Diese Struktur besitzt Knoten und Kanten. Knoten sind Elemente/Objekte, deren Verbindung untereinander durch Kanten repräsentiert wird. Hierbei wird bei den Kanten zwischen zwei Arten unterschieden: gerichteten Kanten, welche eine Hierarchie vorgeben, und ungerichtete Kanten, welche keine Hierarchie vorgeben. Bei den Knoten wird zwischen Wurzelknoten und normalen Knoten unterschieden. Ein Wurzelknoten stellt im Fall eines Implementierungsgraphen das Stammverzeichnis (engl. *root* directory) der Software dar und bei einem Architekturgraphen die jeweiligen Hauptkomponenten.

Im folgenden Teil wird der Aufbau einer solchen GXL-Datei erläutert, wie sie in dieser Arbeit verwendet wird.

Jede GXL-Datei benötigt ein **gxl-element** und ein **graph-element** (siehe Listing 2.1). Hierbei stellt das **graph-element** die Sammlung der Informationen des jeweiligen Graphen in Form von **node-element** (Knoten) und **edge-element** (Kanten) bereit. Das Attribut *id* sagt aus, um was für einen Graphen es sich handelt. In dieser Arbeit liegt der Schwerpunkt auf der Softwarearchitektur, demnach wird die id mit *Architecture* beschrieben.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE gxl SYSTEM "http://www.gupro.de/GXL/gxl-1.0.dtd">
3 <gxl xmlns:xlink="http://www.w3.org/1999/xlink">
4   <graph id="Architecture" edgeids="true">
5     -
6     -
7     -
```

[1] http://www.gupro.de/GXL/dtd/gxl-1.0.html

```
8       -
9     </graph>
10  </gxl>
```

Listing 2.1: GXL-Rootelemente

Das **node-element** (siehe Listing 2.2) stellt die Funktion bereit, um Knoten eines Graphen und dessen Informationen darzustellen. Ein Knoten eines Graphen sollte eindeutig sein. Dies wird gewährleistet, indem sich im **node-element** das eindeutige *id*-Attribut befindet, wodurch es nicht erlaubt ist, Knoten mit derselben ID zu erstellen. Ein **node-element** besitzt immer ein **type-element**, dieses sagt aus, um was für eine Art Knoten es sich handelt. Hierbei handelt es sich um ein *Architecture_Component*-element also eine Softwarearchitekturkomponente. Knoten können aber auch bestimmte Attribute beinhalten, diese werden mit dem **attr-element** angegeben. Innerhalb dieses Elements wird dann der Wert dieses Attributes gesetzt, dieser kann ein String, Integer, Float, Bool und weitere Datentypen[2] sein. In dieser Arbeit besitzen die Knoten immer ein Attribut *Source.Name* und *Linkage.Name*. *Source.Name* beinhaltet den Namen des Knotens, welcher dem Benutzer visuell angezeigt wird, während *Linkage.Name* dafür benutzt wird, jeden Knoten ein eindeutiges Attribut zu geben um Knoten mit demselben *Source.Name* unterscheiden zu können.

```
<node id="N1523">
      <type xlink:href="Architecture_Component"/>
      <attr name="Source.Name">
        <string>Application</string>
      </attr>
      <attr name="Linkage.Name">
        <string>b7457002-d7c7-45da-898b-12c3921dad6d</string>
      </attr>
</node>
```

Listing 2.2: GXL-Nodeelement

Das **edge-element** (siehe Listing 2.3) stellt die Funktion bereit, die Kanten eines Graphen und dessen Informationen darzustellen. Hierbei werden die Kanten auch mithilfe eines *id* Attributes wie bei den Knoten differenziert. Zusätzlich verfügt es noch über die Attribute *from* und *to*. Dabei wird in *from* die *id* vom node-element des Startknoten hinterlegt, und in *to* die *id* des Endknotens. Dazu besitzt das **edge-element** ebenfalls ein **type-element**. Dieser sagt aus, um was für eine Art von Kante es sich handelt. In Listing 2.3 sieht man die Typen *Source_Dependency* und *Belongs_To*. Ersteres stellt eine einfache Abhängigkeit zwischen zwei Knoten dar, während der zweite Typ eine Hierarchie darstellt also eine *Parent-Child-Beziehung* zwischen zwei Knoten.

```
<edge id="E7851" from="N1576" to="N1549">
     <type xlink:href="Source_Dependency"/>
</edge>
```

[2] http://www.gupro.de/GXL/dtd/gxl-1.0.html

```
<edge id="E8221" from="N1565" to="N1570">
    <type xlink:href="Belongs_To"/>
</edge>
```

Listing 2.3: GXL-Edgeelement

2.5 Unity

Unity ist eine Entwicklungsplattform für Computerspiele. In dieser Plattform ist es möglich, Spiele in Echtzeit zu entwickeln und zu testen. Zur Entwicklung wird ein von Unity bereitgestellter Editor und Microsoft Visual Studio[3] benutzt und z. B. die Programmiersprache C#, für die eine Bibliothek von Unity existiert[4][Tec].
Ebenso bietet Unity an, Spiele auf derselben Codebasis für verschiedene Plattformen zu entwickeln, wie beispielsweise für die Playstation 4, Xbox One, Nintendo Switch und mehr. Der Fokus dieser Arbeit liegt aber auf der Plattform Virtual-Reality in Form der HTC Vive[5]. Hierfür stellt Unity ebenfalls Bibliotheken bereit[6].
Wenn man nur den von Unity bereitgestellten Editor betrachtet, arbeitet Unity mithilfe von sogenannten *GameObjects*. Diese werden zur Realisierung von Objekten im Spiel verwendet. Innerhalb von Unity existiert eine Hierarchie von *GameObjects*, wodurch sich Unity auszeichnet. Mithilfe dieser Hierarchie können *GameObjects* abhängig von anderen *GameObjects* betrachtet werden. Beispielsweise kann es ein *GameObject* geben, was das Modell eines Flugzeuges darstellen soll. Innerhalb dieses Flugzeuges befinden sich die Sitzplätze, welche auch als *GameObjects* dargestellt werden können und in der Hierarchie unter dem Flugzeug angeordnet sind. Wenn sich das Flugzeug nun bewegt, werden die Sitzplätze im Flugzeug sich mit bewegen und die Position im Flugzeug nicht ändern. Eine Beispielhierarchie kann in Abbildung 2.3 betrachtet werden.
Um *GameObjects* mit beispielsweise C#-Scripts anzusprechen, stellt Unity die Klasse *MonoBehaviour*[7] bereit. Mithilfe dieser Klasse ist es nun beispielsweise möglich das oben genannte Flugzeug eine Flugroute zu geben, die er abfliegen soll oder andere Arten von Animationen und mehr. So ein Script wird dann mithilfe des Unity-Inspectors an ein *GameObject* angefügt und läuft dann für dieses spezifische *GameObject*. Ein Beispiel dafür kann in Abbildung 2.4 betrachtet werden.

2.6 Leap Motion Controller

Der in der Arbeit benutzte Controller für das Tracking der Handgesten ist der Leap Motion Controller von der Firma Ultraleap[ult]. Dieser Controller ist ein kleines

[3] https://visualstudio.microsoft.com/de/
[4] https://docs.unity3d.com/ScriptReference/
[5] https://www.vive.com/de/product/vive%20series
[6] https://docs.unity3d.com/Manual/VROverview.html
[7] https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

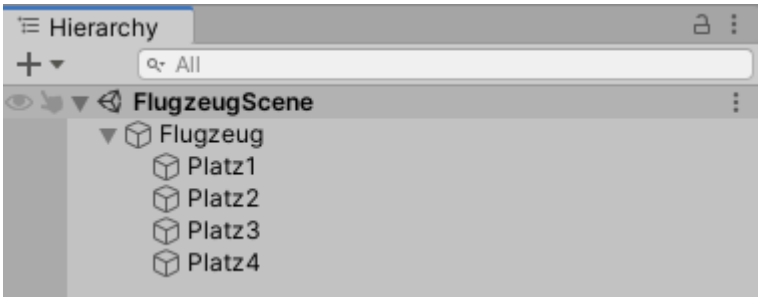

Abbildung 2.3: Beispiel Unity-Hierarchie eines Flugzeuges

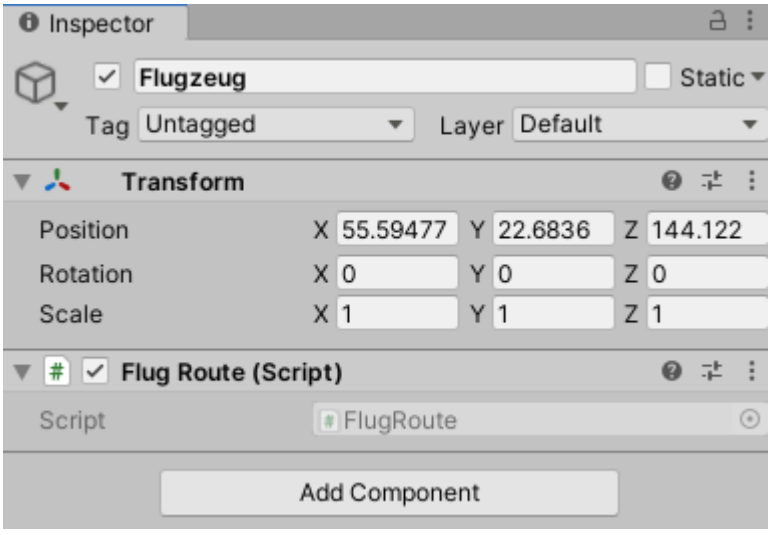

Abbildung 2.4: Beispiel Unity-Inspector eines Flugzeuges

USB-Peripheriegerät welches entweder auf einem Desktop-PC oder Laptop platziert oder, wie in dieser Arbeit, auf einem Virtual Reality Headset montiert wird. Mithilfe seiner zwei Infrarotkameras und drei Infrarot LEDs schafft der Controller es 200 Bilder in der Sekunde aufzunehmen und an die Leap Motion Software zu schicken, welche dann die Aufnahme analysiert und die Hände des Benutzers erkennt. Ultraleap stellt für die Entwicklung mit dem Leap Motion Controller ein Software Development Kit[8] bereit, welches den Entwicklern ermöglicht, den Controller mit Verschiedenen Programmiersprachen ansprechen zu können. Hierbei wird aber zurzeit nur C# und eine C ähnliche Schnittstelle namens *LeapC* aktiv unterstützt. In der Arbeit wird das Software Development Kit C# verwendet. Ebenso werden Plugins für Unity bereitgestellt um den Controller ohne Probleme benutzen zu können.

2.7 Bereits vorhandene Tools

Es ist bereits mit anderen Tools möglich, Softwarearchitekturen in einer zweidimensionalen Sicht zu modellieren. Eine von diesen Tools ist die *Axivion Suite* vom Unternehmen Axivion[AXI]. Dieses Tool basiert auf das Bauhaus-Projekt, welches es ermöglicht, Softwarearchitekturen zu modellieren und eine Softwarearchitekturprüfung durchzuführen. Ebenso existieren noch zwei weitere Tools. Eines davon ist *Lattix Architect* vom Unternehmen Lattix und *Sotograph* vom Unternehmen hello2morrow[Inc][Hel]. Bei den beiden Tools handelt es sich wie bei der *Axivion Suite* um ein Softwarearchitekturprüfungstool, welches dem Benutzer auch die Möglichkeit gibt Softwarearchitekturen zu modellieren. Die Erweiterung für das SEE-Projekt, welches im Rahmen dieser Arbeit erstellt wurde, soll die Softwarearchitekturmodellierung für das *Axivion Suite* in eine 3-Dimensionale Sicht mithilfe der Virtual-Reality bringen und die parallel laufende Arbeit von David Wagner soll dies für die Softwarearchitekturprüfung tun.

[8] https://www.ultraleap.com/product/leap-motion-controller/whatsincluded

Kapitel 3

Konzept und Implementierung

Im folgenden Kapitel wird die Implementierung der Erweiterung des SEE-Projekts aufbauend auf den Designentscheidungen erklärt. Hierbei wird betrachtet, welche Faktoren zu den Designentscheidungen geführt haben und wie die Implementierung aus der Programmiersicht gelöst worden sind. Zuerst werden alle Designentscheidungen der Steuerung, der Import-Funktion und der Export-Funktion dargestellt und darauf folgend die Umsetzung dieser Designentscheidungen.

Um die Softwarearchitektur graphisch darzustellen, wird eine interne Repräsentation eines Graphen verwendet, welche aus dem Datenmodell des SEE-Projekts hervorgeht. Hierbei stellt eine Komponente einer Softwarearchitektur einen Knoten im Graphen dar und die Beziehung zwischen den Komponenten werden durch Kanten dargestellt. Somit werden in den Erklärungen die Begriffe Knoten und Kanten verwendet, womit dann die Komponenten und Beziehungen gemeint sind.

3.1 Anwendungsfälle

Die Nutzer dieser Erweiterung werden im Bereich der Softwareentwicklung tätig sein. Hierbei wird bei der Planung der Softwarearchitektur und während der Implementierungsphase diese Erweiterung ihren Nutzen finden. Folgende Anwendungsfälle sind dabei identifiziert worden:

- Software soll mithilfe einer Softwarearchitektur erstellt werden, welche mit dieser Erweiterung modelliert wird.
- In einem laufenden Softwareprojekt wird die Entscheidung getroffen, jenes Projekt mit dieser Erweiterung erneut zu modellieren oder von einer GXL-Datei der Architektur zu importieren.
- Es wurde bereits eine Architektur in dieser Erweiterung erstellt aber während der Implementierungsphase sind Änderungen an der Architektur nötig. Diese können dann angewendet werden.

3.2 Design

Es wurden zwei Ideen verfolgt, welche das Design beeinflussen. Die erste Idee handelt von einer offnen Spielwelt, wo der Benutzer diese erkunden und neue Knoten und Kanten erstellen kann. Durch die erhöhte Komplexität, die sich durch die Verschachtelung von Knoten ergab, hat sich die Realisierung der Idee in diesem Rahmen als schwierig erwiesen. Innerhalb dieses Abschnittes wird auf eine Alternative eingangen, die die Axivion Suite als Basis verwendet.

3.2.1 Spielwelt

Wie in Kapitel 2.7 erwähnt, soll diese Erweiterung die *Axivion Suite* Softwarearchitekturmodellierung in einer Virtual-Reality 3-Dimensionalen Welt darstellen, um die 2-Dimensionale Eigenschaft des Tools zu simulieren. Um dies zu realisieren wird die Spielwelt als *Whiteboard* dargestellt, was es dem Benutzer ermöglichen soll, Objekte auf diesem zu erzeugen und zu bearbeiten. Dieses Whiteboard soll dem Benutzer aber nicht gezeigt werden, wodurch die Objekte auf dem Whiteboard als schwebend dargestellt werden und farblich von der Hintergrundfarbe der Spielwelt hervorgehoben werden.

3.2.2 Steuerung

Bewegen

Im SEE-Projekt existiert bereits eine Steuerung für die Leap Motion welche es dem Benutzer ermöglicht, durch die *Code City* zu fliegen. Dies tut er, indem er seine linke Zeigefingerspitze an die linke Daumenspitze drückt und dasselbe auch gleichzeitig analog mit dem rechten Zeigefinger und Daumen tut. Die Geschwindigkeit mit der sich der Benutzer bewegt, wird anhand der Distanz zwischen den Zeigefingern bestimmt. Um so größer die Distanz zwischen den zwei Fingern ist, umso schneller bewegt sich der Benutzer und dementsprechend langsamer desto kleiner die Distanz wird.
Somit kann die Steuerung für die Bewegung in der Spielwelt dieser Erweiterung auf der vorhandenen Steuerung aufbauen.

Erzeugen von Knoten

Im *Axivion Suite* ist es möglich Knoten mithilfe der linken Maustaste zu erstellen. Diese haben dann bei der Erzeugung eine feste Größe und können dann durch das Klicken mit der linken Maustaste auf einen Rand des Knotens skaliert werden. Um diesen Vorgang mithilfe von Gestensteuerung zu vereinfachen, soll es möglich sein mithilfe einer Geste einen Knoten zu erzeugen und daraufhin direkt die Größe zu skalieren und festzulegen. Somit wären nur zwei Gesten nötig, eine für das Erzeugen

eines Knotens, mit dieser sollte es dann direkt möglich sein Intuitiv die Skalierung zu steuern und eine für das Bestätigen der Größe für den neuen Knoten.

Bearbeiten/Verschieben der Knoten

Durch *Drag & Drop* ist es in der *Axivion Suite* möglich die Knoten zu verschieben, verschachteln, umzubenennen und die Größe beliebig zu skalieren. Dies kann man mithilfe eines *Raycast*[1] simulieren. Der Benutzer visiert mit seiner rechten Handfläche einen *Raycast*, der für ihn visuell wie ein Laser aussieht, auf einen Knoten und selektiert diesen mithilfe einer Geste mit der linken Hand. Diesen Laser kann man danach auf einen beliebigen Knoten oder einer freien Fläche auf dem Whiteboard richten und den selektierten Knoten verschachteln bzw. verschieben. Das Umbenennen eines Knotens soll mithilfe einer VR-Tastatur gelöst werden, indem der Benutzer in eine virtuelle Tastatur tippt, die vor einem erscheint, wenn ein Knoten selektiert worden ist und eine Geste oder Button für das Erscheinen der Tastatur sorgt. Die Möglichkeit der erneuten Skalierung eines Knotens soll dieselbe Geste benutzen, welche auch bei der Erzeugung der Knoten verwendet wird, um den Benutzer so wenig Gesten wie möglich merken zu lassen.

Erzeugen/Entfernen von Kanten

Durch klicken auf zwei Knoten durch vorheriges Auswählen des Kantentools, wird eine gerichtete Kante zwischen zwei Knoten im *Axivion Suite* erstellt. Um eine ähnliche Funktion zu schaffen, soll der Benutzer hierfür zwischen Knotenerzeugung und Kantenerzeugung differenzieren können. Mithilfe des oben genannten Lasers kann der Benutzer zwei Knoten selektieren, zwischen denen dann eine gerichtete Kante erstellt wird. Der erste Knoten entspricht dann dem Ursprung der Kante und der zweite Knoten das Ziel. Nachdem die zwei Knoten selektiert worden sind, bestätigt der Benutzer mithilfe einer Geste mit der linken Hand die Auswahl und die Kante erscheint.

3.2.3 Import einer Softwarearchitektur

Falls der Benutzer schon eine Architektur in Form einer GXL-Datei besitzt, soll es eine Möglichkeit geben, dass er diese auf das Whiteboard importieren und bearbeiten kann. Hierfür soll der Benutzer einen Dateipfad zur GXL-Datei angeben und eine Funktion gegeben sein, die Architektur während der Laufzeit des Spiels zu importieren in Form einer Geste oder durch Buttons.

[1] https://docs.unity3d.com/ScriptReference/Physics.Raycast.html

3.2.4 Export einer Softwarearchitektur

Wenn der Benutzer mit seiner Arbeit an der Architektur fertig ist, soll er diese in Form einer GXL-Datei aus dem Spiel exportieren können. Hierfür kann es eine Geste oder einen Button geben, wodurch die virtuelle Tastatur auftaucht und einen Dateinamen für die GXL-Datei erwartet. Nach Bestätigen des Namens wird diese Architektur dann exportiert und unter einem festen Pfad abgespeichert.

3.3 Umsetzung

3.3.1 Spielwelt

Mithilfe des Unity-Editors wurde ein leeres *GameObject* erstellt, welches das Whiteboard darstellen soll. Dieses beinhaltet weder Textur noch Skripte. Innerhalb des Unity-Inspectors wurden die Maße des Whiteboards so festgelegt, dass sie für den Benutzer als eine unsichtbare Spielfläche aussieht.

3.3.2 Steuerung

Um die Steuerung mithilfe der Leap Motion zu realisieren wurde die mitgelieferte C#-Bibliothek für die Unity-Engine benutzt. Mithilfe dieser Bibliothek ist man in der Lage, gezielte Teile der simulierten Hände anzusprechen und kann somit Handgesten abfragen.
Innerhalb dieser Arbeit werden fünf verschiedene Handgesten verwendet, um die Steuerung zu realisieren. Diese Handgesten werden im Klassendiagramm der Klasse **ArchGestures** (siehe Abbildung 3.1) dargestellt. Die Funktionen in dieser Klasse

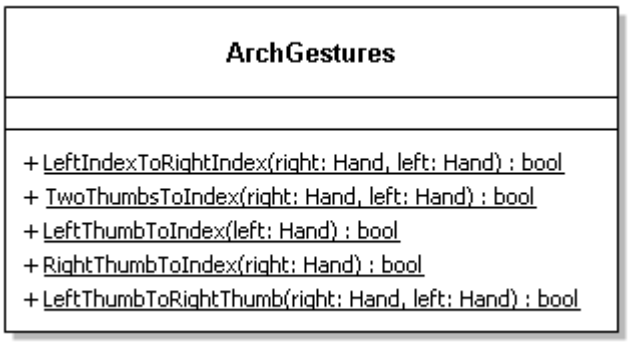

Abbildung 3.1: Klassendiagramm der Klasse **ArchGestures**

werden während der Laufzeit des Spiels in jedem Frame abgefragt, um ein direktes Feedback für die Handgesten zu erhalten.

Um ein besseres Verständnis für die Handgesten zu haben, werden diese nun Anhand der Reihenfolge im Klassendiagramm dargestellt:

- **LeftIndexToRightIndex** stellt die Handgeste dar, in welcher der Benutzer seinen linken und rechten Zeigefingerspitzen aneinander drückt. Ein Beispiel dafür wird in Abbildung 3.2 dargestellt.

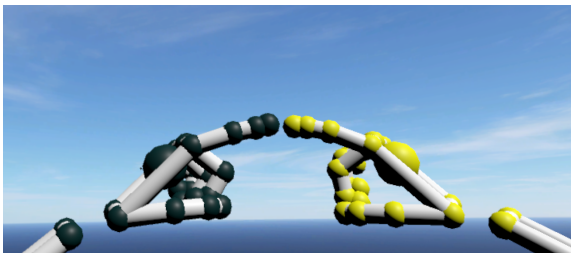

Abbildung 3.2: Handgeste der Funktion **LeftIndexToRightIndex**

- **TwoThumbsToIndex** stellt die Handgeste dar, in welcher der Benutzer seine beiden Daumenspitzen an den jeweiligen Zeigefingerspitzen drückt. Ein Beispiel dafür wird in Abbildung 3.3 dargstellt.

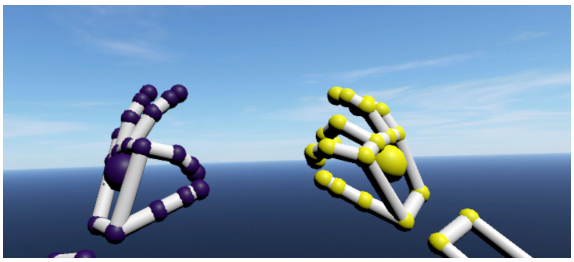

Abbildung 3.3: Handgeste der Funktion **TwoThumbsToIndex**

- **LeftThumbToIndex** stellt die Handgeste dar, in welcher der Benutzer seine linke Daumenspitze an die linke Zeigefingerspitze drückt. Ein Beispiel dafür wird in Abbildung 3.4 dargstellt.

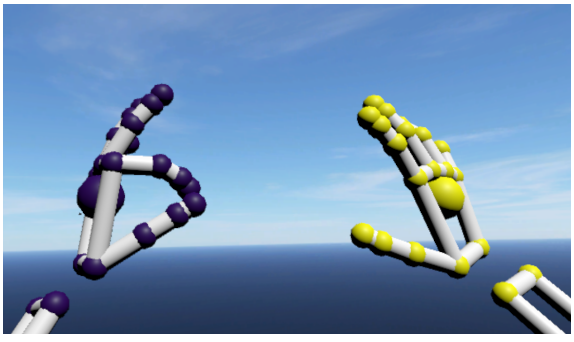

Abbildung 3.4: Handgeste der Funktion **LeftThumbToIndex**

- **RightThumbToIndex** stellt die Handgeste dar, in welcher der Benutzer seine rechte Daumenspitze an seine rechte Zeigefingerspitze drückt. Ein Beispiel dafür wird in Abbildung 3.5 dargstellt.

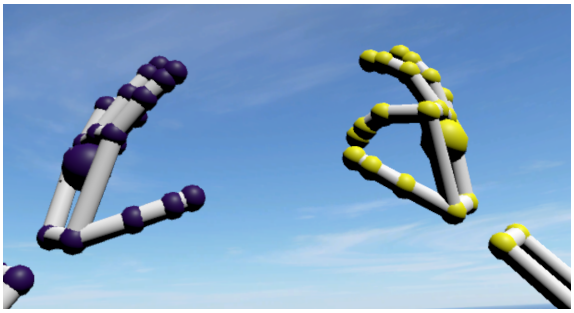

Abbildung 3.5: Handgeste der Funktion **RightThumbToIndex**

- **LeftThumbToRightThumb** stellt die Handgeste dar, in welcher der Benutzer seine linke Daumenspitze an seine rechte Daumenspitze drückt. Ein Beispiel dafür wird in Abbildung 3.6 dargestellt.

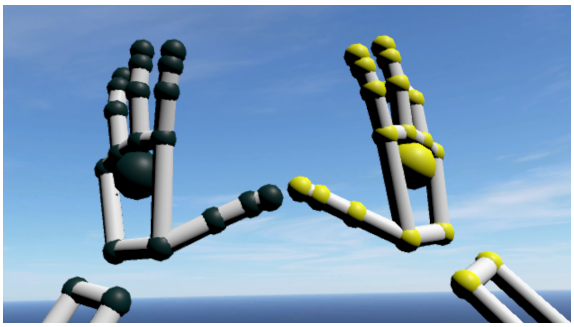

Abbildung 3.6: Handgeste der Funktion **LeftThumbToRightThumb**

Um die verschiedenen Gesten auch für verschiedene Funktionen gleichzeitig zu Nutzen, wurden verschiedene Modi implementiert, um die verschiedenen Funktionen zu unterscheiden. Hierfür erhält der Benutzer ein kleines Raster an Buttons an der linken Hand, welche bei Betrachtung der linken Handfläche erscheinen. Dem Benutzer ist dann möglich, mithilfe der rechten Hand die Buttons zu betätigen und zwischen den Modi zu wechseln. Dieses Raster wird in Abbildung 3.7 dargestellt.

Bewegen

Um die Idee der vorhandenen Steuerung für die Bewegung aus dem SEE-Projekt zu übernehmen, wurde die Klasse **ArchLeapMovement** geschrieben und kann in Abbildung 3.8 betrachtet werden. Es ist möglich sich nach links, rechts, oben und unten zu bewegen. Da man sich in einer 3-Dimensionalen Welt befindet, ist es auch möglich sich nach vorne und nach hinten zu bewegen. Von den Funktionen betrachtet

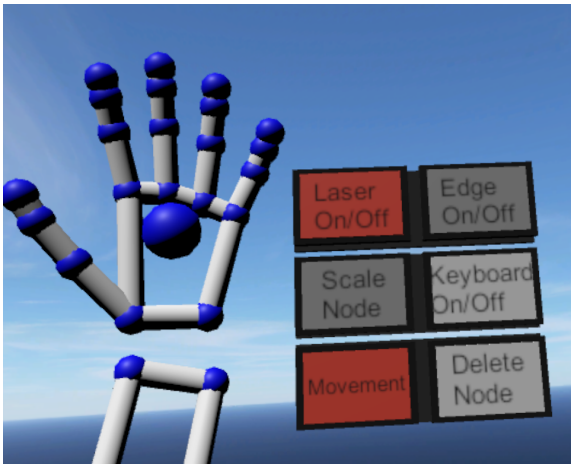

Abbildung 3.7: Raster von Buttons für die verschiedenen Modi.

findet die Bewegung in den Funktionen **Update** und **MoveUpAndDown** statt. In der **Update** Funktion werden im jeden Frame die Gesten abgefragt die für eine Bewegungsrichtung stehen. Diese werden nun kurz aufgelistet:

- Links: **LeftThumbToIndex** als alleinstehende Geste.
- Rechts: **RightThumbToIndex** als alleinstehende Geste.
- Oben/Unten: **LeftThumbToIndex** und **RightThumbToIndex** gleichzeitig, wobei hierbei betrachtet wird, im welchen Bereich sich die simulierten Hände vor dem Benutzer befinden. Befinden sie sich in der Mitte vor dem Benutzer findet keine Bewegung statt, befinden sie sich jedoch auf der oberen Hälfte, bewegt sich der Benutzer nach oben und bei der unteren Hälfte nach unten.
- Vorne/Hinten: **LeftIndexToRightIndex** und **LeftThumbToRightThumb** wobei diese separat betrachtet werden. Ersteres steht für die Bewegung nach vorne und die zweite Geste für die Bewegung nach hinten.

Um die Bewegungen zum Fliegen nach unten und oben zu realisieren, wird die Funktion **MoveUpAndDown** benutzt. Diese wird bei gleichzeitigem benutzen von **LeftThumbToIndex** und **RightThumbToIndex** ausgeführt und prüft, im welchem Bereich sich die simulierten Hände vor dem Benutzer befinden und gibt die Bewegungsrichtung an. Um die Geschwindigkeit der Bewegung zu bestimmen wird, wie auch im SEE-Projekt, die Distanz zwischen den beiden Daumen genommen. Desto weiter sie auseinander sind, desto schneller bewegt man sich in der Spielwelt.

Aus der Benutzersicht betrachtet, muss der Benutzer um sich Bewegen zu können in den Modus für Bewegung wechseln. Dies tut er indem er den Button *Movement* (siehe Abbildung 3.7) an der linken Hand betätigt. Bei Aktivierung von diesem Modus leuchtet der betätigte Button Grün auf und signalisiert dem Benutzer, das er sich nun im Bewegungsmodus befindet und seine Gesten jetzt für die Bewegung benutzt werden. Möchte sich der Benutzer nicht mehr Bewegen, betätigt er den Button erneut und die Grüne Farbe erlischt.

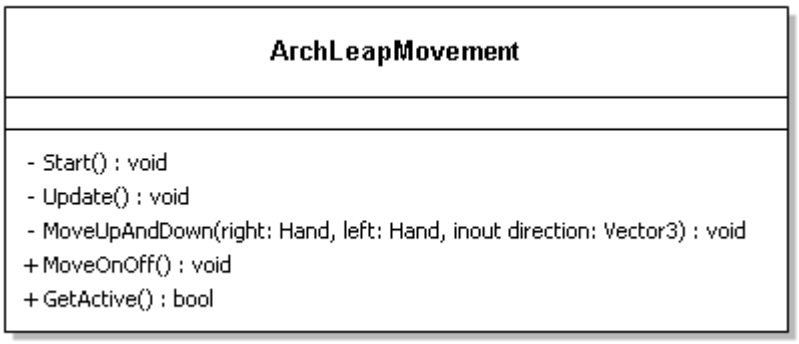

Abbildung 3.8: Klassendiagramm der Klasse **ArchLeapMovement**

Erzeugen von Knoten

Für das Erzeugen von neuen Knoten für den Graphen der Architektur wurde die Klasse **ArchNodeCreator** geschrieben, welche in Abbildung 3.10 dargestellt ist. Die Funktionen die zur Erzeugung eines Knotens benötigt werden, sind die **Update** und **CreateNode**. In der **Update** Funktion werden die Gesten abgefragt, die für das Erzeugen, Skalieren und Bestätigen des Knotens notwendig sind. Um ein Knoten zu Erzeugen wird die Geste **LeftIndexToRightIndex** verwendet und die Funktion **CreateNode** aufgerufen. Diese Funktion erzeugt einen Knoten, das dazugehörige *GameObject* und gibt diesen während der Skalierung einen vorläufigen Namen um auf diesen Knoten im weiteren Verlauf verweisen zu können. Daraufhin findet die Skalierung statt. Der neue Knoten skaliert mithilfe der Distanz zwischen den Zeigefingern, desto größer die Distanz, desto größer wird der Knoten. Die Bestätigung des Knotens findet mithilfe der Geste **TwoThumbsToIndex** statt. Wenn diese Geste während der Skalierung austritt, wird diese beendet und der Knoten nimmt die Größe der jetzigen Distanz an. Nach der Bestätigung wird ein Name für den Knoten erwartet der im Form einer VR-Tastatur abgefragt wird.
Diese Tastatur ist ein Asset aus dem Unity-Assetstore[2] und wurde um die Funktion erweitert, Eingaben der Leap Motion auf der Tastatur entgegenzunehmen. Diese Tastatur wird in Abbildung 3.9 dargestellt.
Um die Implementierung zu vereinfachen, werden Knoten anhand ihrer Namen eindeutig differenziert. Somit ist es nicht möglich, das Knoten mit demselben Namen existieren. Dies wird mithilfe einer Abfrage an den internen Graphen geregelt. Hierbei wird geprüft, ob der neue Name für den neuen Knoten schon von einem anderen Knoten verwendet wird.

Aus der Benutzersicht befindet sich der Benutzer standardmäßig im Modus „Erzeugung von Knoten“. Sobald der Benutzer beispielsweise den Bewegungsmodus benutzt, und diesen dann wieder deaktiviert, befindet er sich wieder im Modus von Erzeugen von Knoten. Somit kann der Benutzer direkt mit dem Erzeugen von Kno-

[2]https://assetstore.unity.com/packages/tools/input-management/osk-onscreenkeyboard-rus-eng-148532

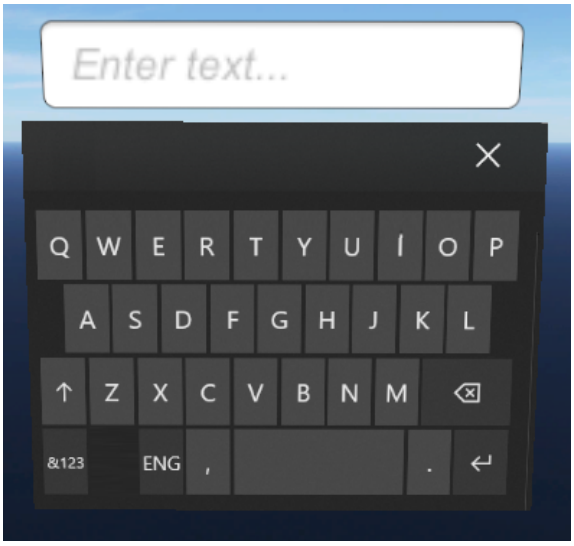

Abbildung 3.9: Die benutze VR-Tastatur.

ten beginnen, wenn er in einer leeren Spielwelt startet.

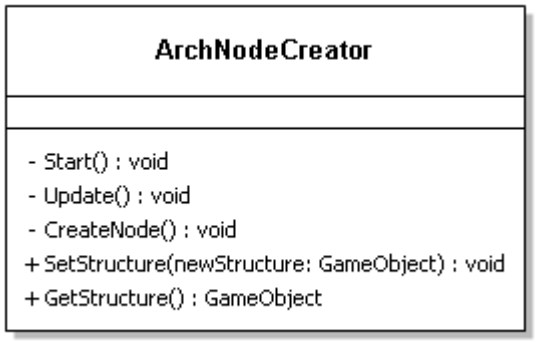

Abbildung 3.10: Klassendiagramm der Klasse **ArchNodeCreator**

Bearbeiten/Verschieben der Knoten

Um Knoten bearbeiten und verschieben zu können, wurde die Klasse **ArchControl** geschrieben. Diese Umfasst die meisten Funktionen im Bereich der Steuerung, da hier viele Bedingungen im Bereich der Verschiebung von Knoten betrachtet werden müssen.
Mithilfe eines Raycasts, der aus der rechten Handfläche erscheint und visuell mithilfe eines *LineRenderers*[3] wie ein Laser dargestellt wird, wird die Funktion ermöglicht, vorhandene Knoten zu selektieren und daraufhin zu verschieben oder zu bearbeiten. Die Funktionen für das Verschieben werden alle in der **Update** Funktion dieser

[3]https://docs.unity3d.com/ScriptReference/LineRenderer.html

Klasse dargestellt. Durch das Verschieben von Knoten ist es möglich, die Hierarchie des zugrundeliegenden Graphen zu bearbeiten, wie beispielsweise einer Komponente einer Softwarearchitektur eine Unterkomponente zuzuordnen oder zu entfernen. Hierbei kann eine Unterkomponente auch weitere Unterkomponenten beinhalten und somit kann eine Komponente eine unendliche Tiefe besitzen. Die oben genannten Bedingungen für das Verschieben von Knoten werden nun einmal kurz aufgelistet:

- Ein Knoten wird auf dem Whiteboard bleibend verschoben.
- Ein Knoten wird vom Whiteboard auf einen anderen Knoten verschoben und wird somit ein Unterknoten.
- Ein Unterknoten wird von seinem Oberknoten auf das Whiteboard verschoben. Somit wird der Unterknoten zu einen alleinstehenden Knoten.
- Ein Unterknoten wird auf seinem Oberknoten verschoben. Hierdurch bleibt der Unterknoten bei dem selben Oberknoten.
- Ein Oberknoten darf nicht auf einen Unterknoten verschoben werden, wenn dieser Unterknoten Teil dieses Oberknotens ist.

Nach Einhaltung dieser Bedingungen existiert nun die Möglichkeit, Knoten erfolgreich zu verschieben. Dies wird bei einer erfolgreichen Verschiebung visuell mithilfe eines Blinkens des Oberknotens dargestellt. Dieses Blinken wird durch die Klasse **ArchBlinkFeedback** zur Verfügung gestellt.

Neben dem Verschieben ist es auch möglich, einen Knoten umzubenennen, die Größe erneut zu Skalieren und Knoten zu löschen. Um Knoten umzubenennen wird zuerst ein Knoten mithilfe des Lasers selektiert und daraufhin die VR-Tastatur mithilfe des *Keyboard* Buttons (siehe Abbildung 3.7) aufgerufen. Hier kann nun der neue Name eingegeben werden und mit der Enter-Taste bestätigt werden, woraufhin sich die Tastatur schließt. Ist der neue Name schon durch einen anderen Knoten belegt, muss ein neuer Name eingegeben werden.
Das erneute Skalieren von Knoten funktioniert mithilfe des *Scale-Modus*. Dieser Verlangt ein vorheriges Selektieren eines Knotens. Dieser Modus wird mithilfe des *Scale-Node* Buttons (siehe Abbildung 3.7) aktiviert und das Buttonraster erhält zwei neue Buttons, diese können in Abbildung 3.11 betrachtet werden. Das Skalieren funktioniert Analog wie bei dem Erzeugen von Knoten, indem man die beiden Zeigefingerspitzen aneinander drückt und auseinander zieht und mithilfe der **TwoThumbsToIndex**-Geste bestätigt. Hierbei ist es nun aber möglich, mithilfe der neuen Buttons die Richtung der Skalierung zu beeinflussen. Durch einmaliges betätigen des *X-/* oder *Y*-Buttons, wird die Skalierung in positiver X-/ oder Y-Richtung stattfinden. Durch erneutes betätigen einer der beiden Buttons findet die Skalierung in negativer Richtung statt. Bei einer dritten Betätigung einer der beiden Buttons wird die Standardskalierung in allen Richtungen stattfinden. Es ist aber nicht möglich, die beiden neuen Buttons gleichzeitig aktiv zu haben. Wenn man sich beispielsweise in positiver X-Richtung befindet und den *Y* Button betätigt, wird die jetzige X-Richtung verworfen und die Y-Richtung verwendet.

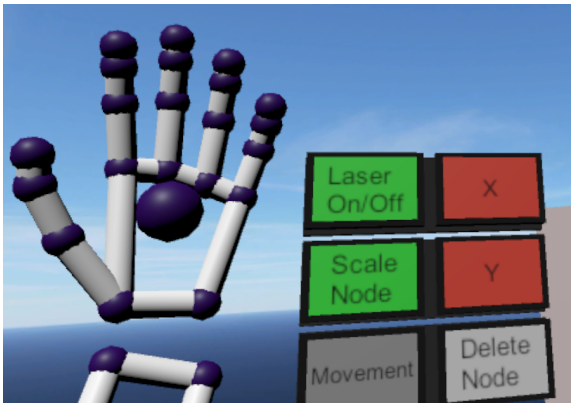

Abbildung 3.11: Das neue Raster im *Scale-Modus*.

Die Funktion für das Löschen von Knoten wird mithilfe des *Delete-Node* Buttons (siehe Abbildung 3.7) realisiert. Durch das Selektieren eines Knotens und durch darauf folgenden Betätigen des Buttons, wird der selektierte Knoten und dessen Unterknoten inklusiver aller Kanten, die in Beziehung zu dem Knoten und alle Unterknoten stehen, entfernt und aus der internen Graphenstruktur gelöscht.

Erzeugen/Entfernen von Kanten

Das Erzeugen und Entfernen von Kanten wird auch in der Klasse *ArchControl* umgesetzt. Hierfür wechselt man durch das Betätigen des *Edge*-Buttons in den *Edge-Modus*. In diesem selektiert man mithilfe des Lasers zwei Knoten. Hierbei steht der erste selektierte Knoten als Ursprung und der zweite als Zielknoten. Durch die Geste **RightThumbToIndex** wird die Kante zwischen den zwei Knoten erstellt. Falls zwischen den zwei Knoten bereits eine Kante in dieser Richtung existiert, wird diese, statt eine neue Kante zu erstellen, gelöscht. Somit ist der Prozess für das Erzeugen und Entfernen einer Kante analog.

3.3.3 Import einer Softwarearchitektur

Um eine vorhandene Softwarearchitektur in Form einer GXL-Datei zu importieren, stellt die Klasse **ArchImportGXL** Funktionen bereit. Diese Klasse kann in Abbildung 3.14 betrachtet werden.
Innerhalb dieser Klasse wird der interne Graph gespeichert. Dieser Graph befindet sich in der Variable *graph*. Zum Start des Spiels wird der Graph als leerer Graph initialisiert und durch erzeugen von Knoten und Verschiebungen bearbeitet. Neben der direkten Bearbeitung ist es möglich mithilfe eines Import-Buttons eine Softwarearchitektur zu importieren (siehe Abbildung 3.12). Durch betätigen des Buttons, wird die GXL-Datei geladen, die der Nutzer vorher als Dateipfad im Unity-Inspektor im Import-*GameObject* eingetragen hat (siehe Abbildung 3.13).
Der Vorgang des Imports wird durch die **Import** Funktion der Klasse gestartet. Hierbei wird zunächst die GXL-Datei traversiert und aus diesen Daten wird ein

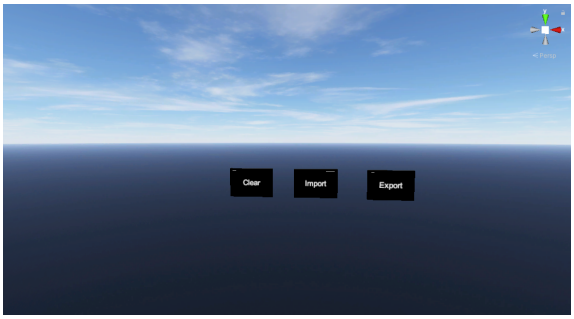

Abbildung 3.12: Die Buttons für das importieren/exportieren/löschen einer Softwarearchitektur

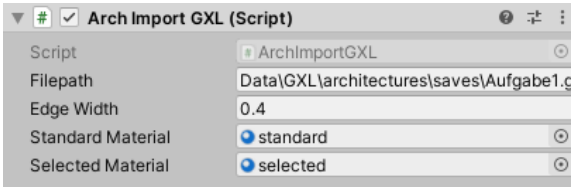

Abbildung 3.13: Unity-Inspektor des Import-*GameObjects*.

Graph erzeugt, welcher in der Variable *graph* gespeichert wird. Daraufhin werden alle Wurzelknoten aus diesem Graphen geholt und zwischengespeichert. Aufbauend auf den Wurzelknoten wird dann der Graph für den Benutzer visuell auf das Whiteboard generiert. Hierfür wird jeder Wurzelknoten mithilfe von einer Tiefensuche traversiert und zu jedem Knoten wird ein Gameobject erzeugt. Aufbauend auf der Anzahl der Unterknoten jedes Knotens werden die Knoten skaliert und die Unterknoten auf diesen positioniert. Als Endergebnis wird dann der vollständige Graph auf dem Whiteboard dargestellt.
Falls eine GXL-Datei benutzt wird, die mithilfe dieser Erweiterung generiert worden ist, wird der Teil der Skalierung und Positionierung anders behandelt. Diese GXL-Dateien halten für jeden Knoten die Position und die Größe, welche sie zum Zeitpunkt des Exportes hatten. Somit können diese Daten ausgelesen und für die Skalierung und Positionierung wiederverwendet werden.

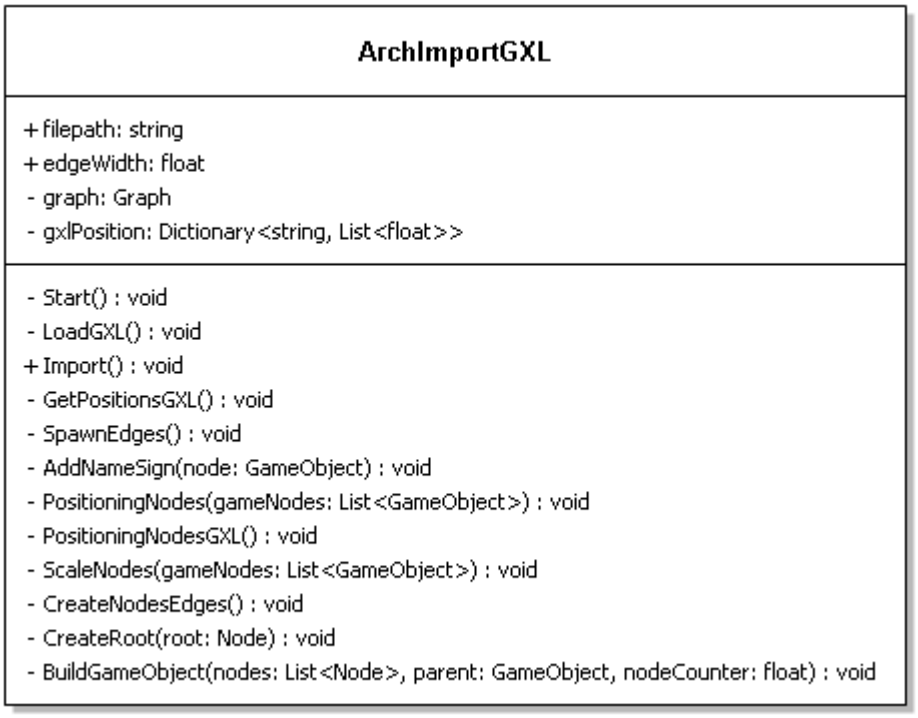

Abbildung 3.14: Das Klassendiagramm der Klasse **ArchImportGXL**

3.3.4 Export einer Softwarearchitektur

Die Funktion des Exports einer Softwarearchitektur wird durch die Klasse **ArchExportGXL** (siehe Abbildung 3.15) und der Klasse **GraphWriter** aus dem SEE-Projekt zur Verfügung gestellt. Durch das Betätigen des Export-Buttons (siehe Abbildung 3.12) wird der Vorgang des Exports gestartet. Hierbei taucht die VR-Tastatur auf und erwartet eine Eingabe für den Dateinamen. Nach betätigen der Entertaste startet der Vorgang des Exports. Zuerst wird jedem Knoten seine jetzige Position und Größe zugeteilt mithilfe von Variablen der Knoten. Daraufhin wird der Graph an den **GraphWrite** übergeben. Dieser baut zuerst den Kopf der GXL-Datei auf und traversiert daraufhin alle Knoten und Kanten des Graphen und schreibt dessen Informationen in die Node-/ und Edge-Tags. Diese GXL-Datei wird dann innerhalb des Projektverzeichnisses unter *Data/GXL/architectures/saves* gespeichert. Falls eine Datei mit demselben Namen in diesem Verzeichnis existiert, wird die alte Datei mit der neuen überschrieben.

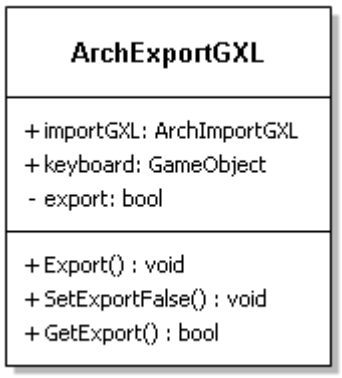

Abbildung 3.15: Klassendiagramm der Klasse **ArchExportGXL**

3.3.5 Probleme, die nicht gelöst worden sind

Es ist möglich mit dem Import eine beliebig große Softwarearchitektur zu importieren. Hierbei existiert aber ein Problem an der visuellen Darstellung der Tiefe, die nicht gelöst werden konnte. Ab der Tiefe sechs eines Knotens werden die weiteren Unterknoten zu klein dargestellt und es nicht möglich diese zu erkennen, selbst wenn man sich näher an die Knoten begibt. Dieses Problem könnte man beispielsweise durch eine Zoom-Funktion oder durch eine Verschachtelung ab einer bestimmten Tiefe lösen.

Ebenso birgt der Leap Motion Controller ein Problem. Sobald die Hände aus der Sicht der Leap Motion verschwinden, werden die Hände nicht mehr getracked. Hierdurch kommt es zur Laufzeit dazu, dass bestimmte Gesten ausgeführt worden sind oder Buttons betätigt worden sind, welche nicht beabsichtigt waren und die Erfahrung mit der Erweiterung beeinträchtigen. Hierfür wurde leider während der Bearbeitungszeit keine Lösung gefunden.

Kapitel 4

Evaluation

Die im Fokus liegende Forschungsfrage, die wir in dieser Evaluation betrachten, lautet:

> Ist die Leap Motion als Eingabegerät für das Modellieren einer Softwarearchitektur aus der Benutzersicht geeignet?

Um dies zu untersuchen, führen wir einen Usability-Test durch, in dem Teilnehmer bestimmte Aufgaben in der implementierten Erweiterung lösen müssen, dabei die Leap Motion verwenden und daraufhin einen Fragebogen beantworten. Bei einem Usability-Test handelt es sich um einen Blackbox-Test[Gle15], deswegen werden die Ausgaben der Implementierung nicht betrachtet, sondern nur das, womit die Teilnehmer interagieren werden.

4.1 Auswahl der Teilnehmer

Durch die derzeitige COVID-19-Pandemie ist das Gebäude der Universität Bremen im Notbetrieb und somit können keine Testdurchläufe in der Universität durchgeführt werden. Dadurch ist die Suche nach Teilnehmern eingeschränkt auf den Bekanntenkreis, die dann zu einem nach Hause kommen und die Evaluation durchführen. Bei der Auswahl dieser Teilnehmer muss beachtet werden, dass diese ein Wissen für die folgenden Themen mitbringen:

1. Die Definition und Aufbau eines hierarchischen Graphen.
2. Was eine Softwarearchitektur ist.
3. Was Komponenten einer Softwarearchitektur sind.

4.2 Aufbau der Testdurchläufe

4.2.1 Geräte

Die Evaluation wird durchgeführt auf einen Rechner mit dem Betriebssystem Windows 10 Pro, einem Intel Core i5-8400 Prozessor, einer GeForce GTX 1060 6GB Grafikkarte, Unity mit der Version *2019.3.14f1* und einer HTC-Vive mit einem HTC Vive Sensor und der anmontierten Leap Motion. Durch die Nutzung von nur einem HTC-Vive Sensor kann es zu Komplikationen während der Evaluation kommen, weil die Hände der Teilnehmer, die sich vor der Leap Motion befinden, die Sicht des Sensors zur HTC-Vive blockieren und somit die HTC-Vive für diesen Zeitpunkt keine Bilder mehr überträgt.

4.2.2 Aufbau der zwei Aufgaben

Um die oben genannte Forschungsfrage beantworten zu können, müssen die Aufgaben für die Teilnehmer der Evaluation alle Steuerungsmöglichkeiten der Erweiterung abdecken, damit die Teilnehmer alles betrachten und bei der Bewertung berücksichtigen können. Hierfür wird die Evaluation in zwei Aufgaben aufgeteilt. In der ersten Aufgabe landen die Teilnehmer in einer leeren Welt (siehe Abbildung 4.1). Hier sollen sie einer Anleitung folgen, welche ihnen eine Softwarearchitektur vorgibt.

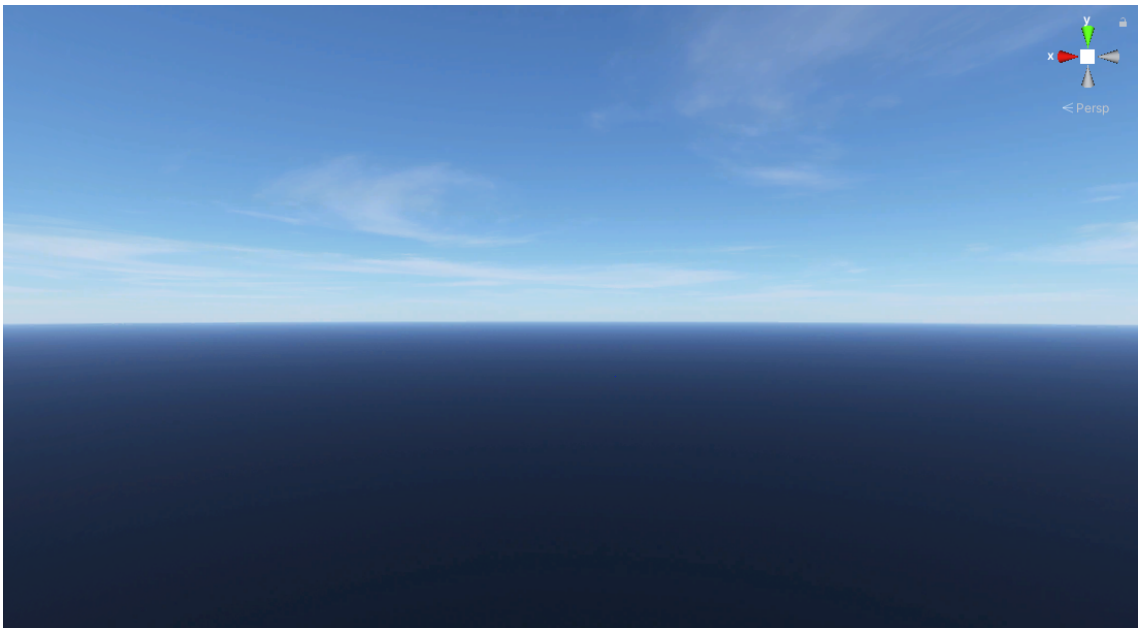

Abbildung 4.1: Eine leere Spielwelt des Systems.

Es wird ihnen nach der Anleitung von einem Beobachter gesagt, was für Objekte sie erzeugen und wie sie miteinander in Beziehung stehen sollen. Hierbei wird eine Softwarearchitektur erzeugt, die der alten abstrakten Twitter-Architektur[1] ähneln

[1] https://www.infoq.com/presentations/Real-Time-Delivery-Twitter/

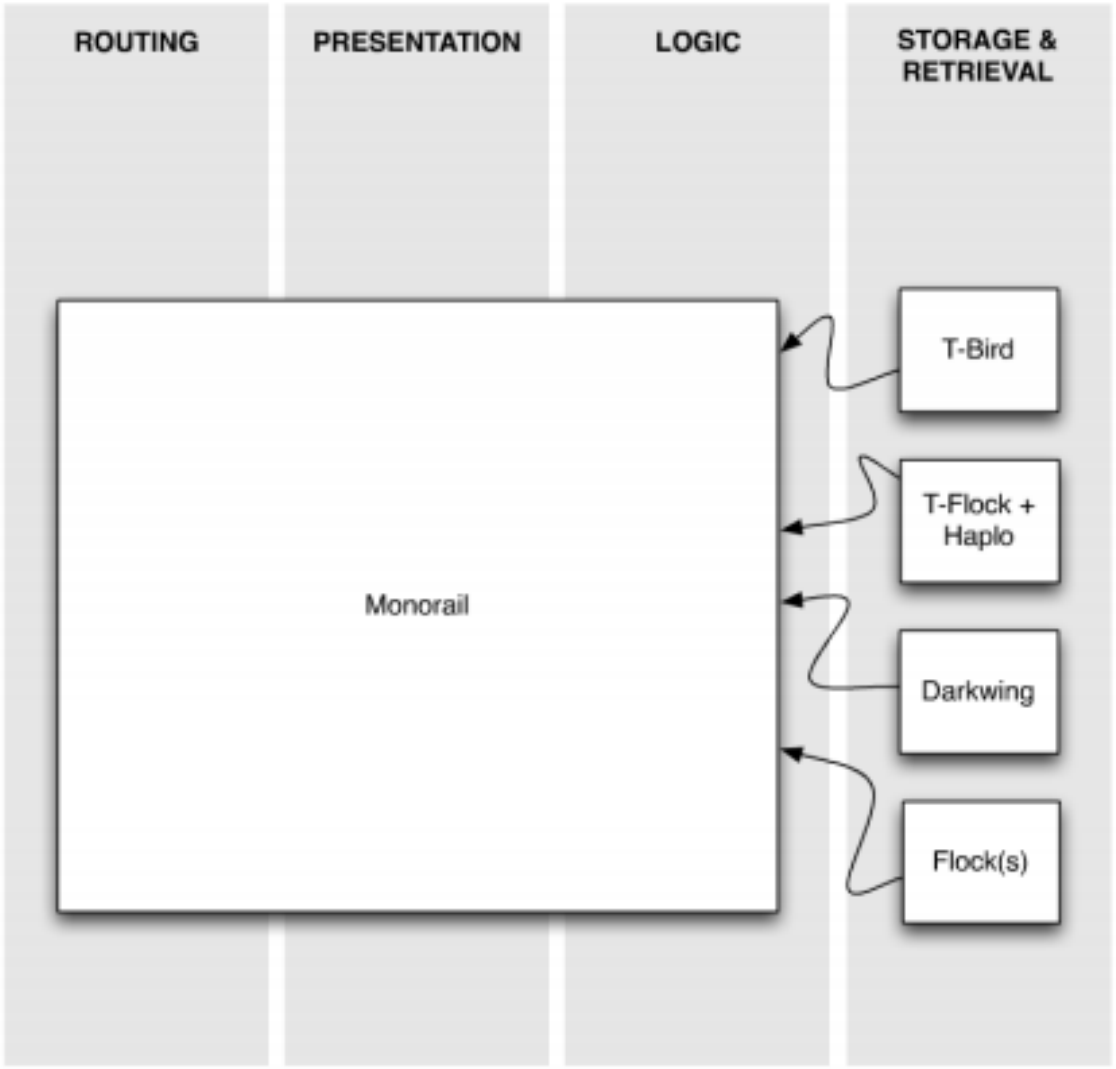

Abbildung 4.2: Die alte Abstrakte Twitter-Architektur

soll (siehe Abbildung 4.2). Die Komponente *Storage & Retrieval* wird in den Aufgaben als *Persistenz* dargestellt, da die VR-Tastatur keine Sonderzeichen besitzt. Die Anleitung für die erste Aufgabe sieht wie folgt aus:

1. Erschaffen Sie die Komponente *Persistenz*. Diese soll vier weitere Komponenten beinhalten, also achten Sie auf die Größe.

2. Erschaffen Sie nun die Komponente *TBird* und packen Sie diese auf die *Persistenz*.

3. Erschaffen Sie nun die Komponente *Haplo* und packen Sie diese auf die *Persistenz*.

4. Erschaffen Sie nun die Komponente *Darkwing* und packen Sie diese auf die *Persistenz*.

5. Erschaffen Sie nun die Komponente *Flock* und packen Sie diese auf die *Persistenz*.

6. Erschaffen Sie nun die Komponente *Monorail* und packen Sie diese neben die *Persistenz*.

7. Erschaffen Sie nun die Komponente *Client* und packen Sie diese über die *Monorail*.

8. Fügen Sie nun Kanten zwischen den folgenden Komponenten ein:
   TBird → *Monorail*, *Haplo* → *Monorail*, *Darkwing* → *Monorail*, *Flock* → *Monorail*, *Monorail* → *Client*

9. Exportieren Sie nun diese Softwarearchitektur indem Sie die *Export*-Taste drücken. Geben Sie ihr einen Namen Ihrer Wahl und bestätigen Sie mit der Enter-Taste.

Da die Teilnehmer die freie Wahl haben, wie sie die Komponenten nebeneinander und aufeinander platzieren können, existieren mehrere richtige Lösungen. Eine mögliche Lösung dieser Aufgabe ist in Abbildung 4.3 zu sehen. Die Teilnehmer

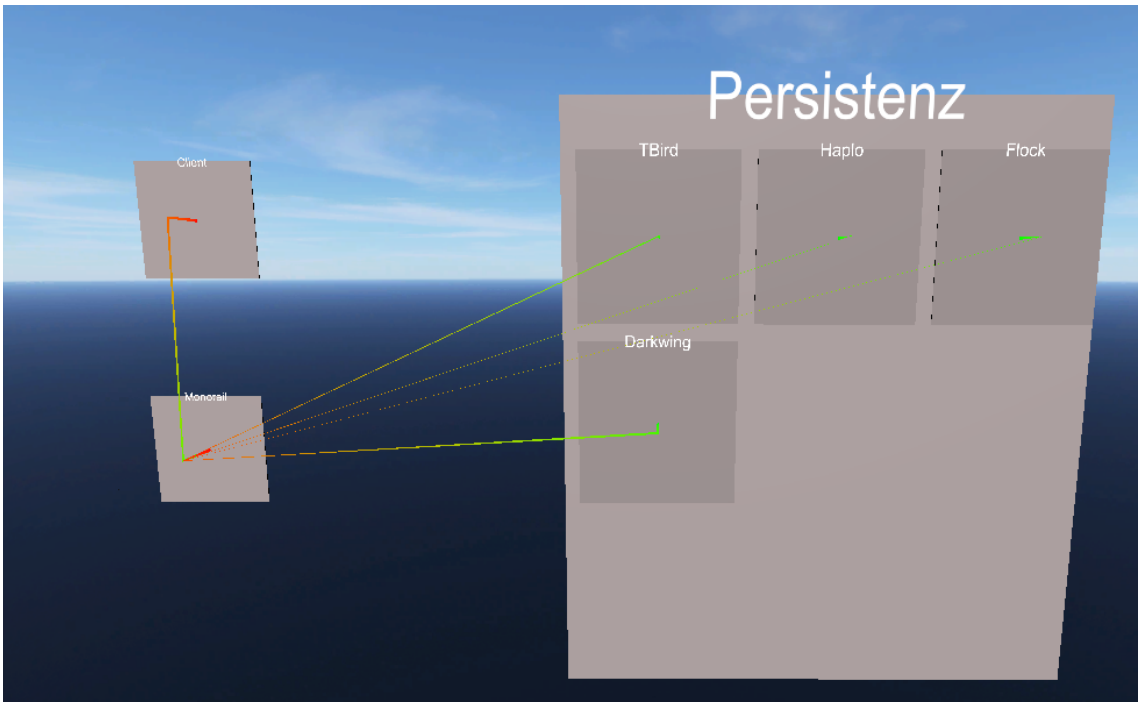

Abbildung 4.3: Eine mögliche Lösung für die erste Aufgabe.

können in dieser Aufgabe in der Lage sein, diese zu meistern, ohne sich in der Spielwelt bewegen zu müssen und die Komponenten erneut zu skalieren oder umzubenennen, Kanten oder Komponenten zu löschen oder eine Softwarearchitektur zu importieren. Hierbei werden also alle Funktionen abgedeckt mit Ausnahme der oben genannten, welche nun den Fokus der zweiten Aufgabe bilden.

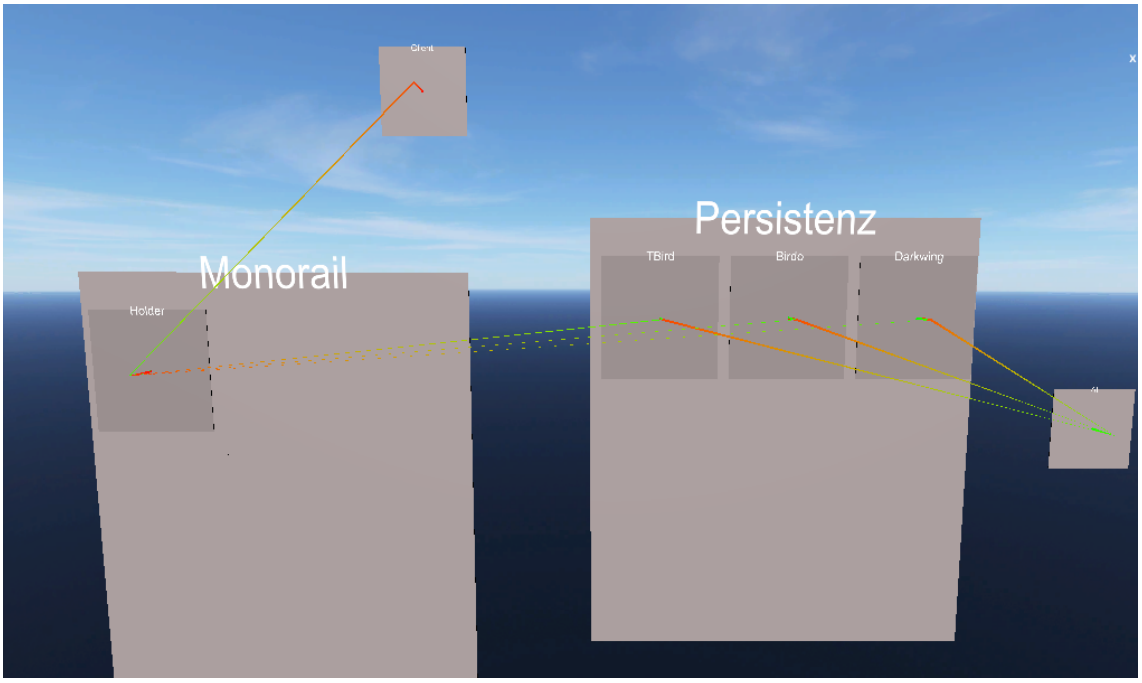

Abbildung 4.4: Eine mögliche Lösung für die zweite Aufgabe.

In der zweiten Aufgabe erhalten die Teilnehmer eine Softwarearchitektur, welche sie importieren und bearbeiten sollen. Die zu bearbeitende Softwarearchitektur wird die sein, welche die Teilnehmer in der ersten Aufgabe erschaffen haben. Eine Beispiellösung für die erste Aufgabe kann in Abbildung 4.3 betrachtet werden. Der Sinn hinter der zweiten Aufgabe ist es, den Teilnehmern Steuerungsfunktionen benutzen zu lassen, die sie in der ersten Aufgabe vielleicht nicht benutzt haben. Die Anleitung für die zweite Aufgabe sieht wie folgt aus:

1. Entfernen Sie alle Kanten, die in die Komponente *Monorail* ein- und ausgehen.
2. Erschaffen Sie eine neue Komponente *Holder*, skalieren Sie die Komponente *Monorail* und packen Sie *Holder* auf *Monorail*.
3. Erschaffen Sie eine neue Komponente *AI* und packen Sie diese neben die *Persistenz* aber nicht auf der selben Seite wie die Komponente *Monorail*.
4. Löschen Sie nun die Komponente *Flock* aus der *Persistenz* und benennen Sie *Haplo* um zu *Birdo*.
5. Fügen Sie nun die folgenden Kanten hinzu:
   TBird → *Holder*, *Birdo* → *Holder*, *Darkwing* → *Holder*, *Holder* → *Client*, *AI* → *TBird*, *AI* → *Birdo*, *AI* → *Darkwing*

Wie auch in der ersten Aufgabe gibt es verschiedene Lösungsmöglichkeiten. Eine Möglichkeit wird in Abbildung 4.4 dargestellt. Hier wurden nun Komponenten erneut skaliert, umbenannt und entfernt, sowie Kanten neu hinzugefügt und entfernt.

Ebenso müssen die Teilnehmer sich in der Spielwelt bewegen, um die Komponente *AI* neben die *Persistenz*-Komponente zu setzen. Damit werden alle Funktionen abgedeckt, welche nicht Fokus der ersten Aufgabe waren.

4.2.3 Durchführung

Vor jedem Testdurchlauf wird die HTC Vive gründlich desinfiziert um den jetzigen Gesundheitsmaßnahmen gerecht zu werden. Daraufhin folgt eine Einweisung:

1. Einwilligungserklärung zur Aufnahme der hier erzeugten Daten.
2. Wie die HTC-Vive richtig aufgesetzt wird.
3. Die Steuerung im Spiel, inklusive bis zu 20 Minuten Probezeit.
4. Erklärung der zwei Aufgaben die mithilfe der Steuerung gelöst werden sollen.
5. Beantwortung von Fragen der Teilnehmer.
6. Durchführung der zwei Aufgaben.
7. Ausfüllen des Fragebogens von den Teilnehmern.

Mithilfe dieser Reihenfolge wird eine einheitliche Durchführung für alle Teilnehmer gewährleistet.

4.2.4 Aufnahme der Daten während der Durchführung

Folgende Daten werden aufgezeichnet:

1. Die Zeit in Sekunden, wie lange ein Teilnehmer für eine Aufgabe gebraucht hat. Da es nicht nur eine richtige Lösung gibt, wird dies mithilfe einer Stoppuhr gemessen. Gestartet wird diese, wenn der Teilnehmer bereit ist und gestoppt, wenn der Teilnehmer meint, dass er fertig ist.
2. Es wird eine Bildschirmaufnahme zu den beiden Aufgaben erstellt, welche die Interaktionen des Teilnehmers im System zeigen, aber nicht der Teilnehmer selbst dargestellt wird.
3. Es wird einen Beobachter geben. Dieser wird die Teilnehmer während der Durchläufe beobachten und seinen Fokus darauf legen, wie sie mit den einzelnen Steuerungsmöglichkeiten zurechtkommen. Ebenso macht er sich Notizen dazu, wie die Teilnehmer auf bestimmte Steuerungsmöglichkeiten reagieren, ob körperlich oder mit Sprache.

Tabelle 4.1: Die Fragen des System-Usability-Scale.

Fragen des SUS
Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.
Ich empfinde das System als einfach zu nutzen.
Ich empfinde das System als unnötig komplex.
Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.
Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.
Ich finde, das es im System zu viele Inkonsistenzen gibt.
Ich kann mir vorstellen, dass die meisten Leute das System schnell beherrschen können.
Ich empfinde die Bedienung sehr umständlich.
Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.
Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.

4. Nach Abschluss der zwei Aufgaben folgt ein Fragebogen in Form eines *SUS* (System-Usability-Scale)(siehe Tabelle 4.1), den jeder Teilnehmer ausfüllen muss.

Die *SUS* wurde von John Brooke im Jahre 1986 entwickelt. Mithilfe dieser Methode ist es möglich, einen *SUS-Score* zu ermitteln, indem die Teilnehmer einen Fragebogen mit zehn Fragen beantworten. Dieser *SUS-Score* befindet sich nach Auswertung der Fragebögen zwischen 0 und 100. Hierbei steht die 0 für ein schlechtes System und die 100 für ein sehr gutes System [Bro96].
Die zehn Fragen im Fragebogen sind aufgeteilt in fünf positive und fünf negative formulierte Fragen. Die Fragen beziehen sich zwar auf das System, nehmen jedoch die persönliche Sicht jedes einzelnen Teilnehmers mit in betracht. Mithilfe einer *Likert-Skala*[Lik32] werden diese Fragen beantwortet. Dabei liegt diese Skala in fünf Bereichen: Der erste Bereich steht für „Stimme gar nicht zu“ und stellt den negativsten Punkt dar. Die weiteren Bereiche sind positiver, wobei der dritte Bereich für eine neutrale Meinung steht, bzw. einer Enthaltung. Der fünfte Bereich stellt mit „Stimme voll zu“ die positivste Antwort dar[Bro96]. Die Fragen des SUS können in der Tabelle 4.1 betrachtet werden.

Neben der SUS existieren auch andere Möglichkeiten, um die Usability von Software zu beurteilen. Beispielsweise den *User Experience Questionnaire*. Mithilfe von Gegensatzpaaren von Eigenschaften werden 28 Fragen gestellt, die der Proband nach

Vollenden des Duchrlaufs ausfüllen muss. Hierbei bewertet der Proband, welche der beiden Eigenschaften er dieser Software eher geben würde[LHS08].
Ein anderer betrachteter Fragebogen war der *NASA Task Load Index*. Dieser Fragebogen betrachtete aber eher die mentale Belastung des Probanden während der Nutzung des Systems mithilfe von sechs Fragen[HS88].

Nach näherer Betrachtung wurde die SUS als Fragebogen für die Bewertung der Usability ausgewählt. Während der *NASA Task Load Index* sich eher weniger mit der Usability als Fokus beschäftigt und der *User Experience Questionnaire* eine weite von 28 Fragen besitzt, um die Usability zu bewerten, stellt sich die SUS, wie auch von John Brooke erwähnt, als eine *Quick and Dirty* Methode[Bro96] dar, um mithilfe von zehn Fragen einen Rückschluss auf die Usability zufinden und der Fokus direkt auf die Benutzbarkeit gelegt wird.

4.2.5 Pilotlauf

Bevor die Studie beginnen kann, wird ein Probedurchlauf der oben genannten Aufgaben mithilfe eines Informatik Studenten durchgeführt. Hierbei soll getestet werden, ob die Aufgaben lösbar sind und die Erklärungen dazu für die Lösungen ausreichend sind. Der Student hatte durch verschiedene VR-Spiele schon Erfahrung in der VR-Welt gehabt und hat durch sein Studium Wissen über UML-Diagramme und dem Aufbau von Softwarearchitekturen sammeln können.
Der Student war in der Lage mithilfe der Erklärungen die beiden Aufgaben zu lösen. Für die erste benötigte er 11 Minuten und für die zweite 12 Minuten. Es ist aufgefallen, das durch das Benutzen von nur einem Sensor für die HTC-Vive, die Verbindung zur VR-Brille oft abbricht und er sich neu vor dem Sensor positionieren musste. Abgesehen von diesem Problem konnte er die Aufgaben gut lösen, wobei sich die Zeiten auf den Verbindungsabbrüchen zurückführen lassen können. Somit rechnet man mit ähnlichen wenn nicht auch längeren Zeiten.

4.3 Auswertung

In der Auswertung wird es eine quantitative Auswertung durch die SUS geben, mit einer anschließenden Diskussion der Ergebnisse. Zuletzt wird es eine qualitative Auswertung geben, in der auf die Kommentare der Probanden eingegangen wird und Merkmale der Aufnahmen erläutert werden. Abschließend werden dann auf die *Threats of Validity* eingegangen. Diese Bedrohungen der Gültigkeit werden anhand verschiedener Faktoren in verschiedenen Arten von Versuchsgruppen dargestellt, und es wird bewertet, wie diese Faktoren Einflüsse auf die Ergebnisse hatten.

An der Evaluation haben insgesamt 6 Testpersonen teilgenommen. Sie fand in den eigenen Räumlichkeiten statt, da sich die Universität zu diesem Zeitpunkt im Notbetrieb befand. Die Durchläufe dauerten zwischen 22 und 30 Minuten, im Mittel

25,5 Minuten. Durch die Nutzung von nur einem Sensor kam es zu technischen Problemen während der Durchführung, die Auswirkung auf die Gesamtzeit hatte. Bei dem Ausfüllen des Fragebogens sind keine Probleme aufgetreten. Nun folgen ein paar Informationen zu den Testpersonen:

- Das Durchschnittsalter der Testpersonen war 23.
- Von den Testpersonen waren zwei weiblich und vier männlich.
- Fünf Testpersonen hatten Erfahrung mit VR-Brillen.
- Drei Testpersonen haben Erfahrungen zum Thema Informatik innerhalb des Studiums sammeln können, während die anderen Teilnehmer keinen Bezug zur Informatik hatten.

4.3.1 Quantitative Auswertung

Das Ziel dieser Evaluation war es herauszufinden, ob die Leap Motion als Eingabegerät für diese Erweiterung aus der Benutzsicht geeignet ist. Um dies zu überprüfen, werden die Ergebnisse der SUS betrachtet. Der Score wird im Bereich zwischen 0 und 100 sein, wobei 0 als schlechte Usability und 100 als beste Usability gilt.

Nachdem die Probanden ihre Aufgaben in der Erweiterung beendet haben, wurde der SUS-Fragebogen beantwortet. Der jeweilige SUS-Score der Teilnehmer wird in Abbildung 4.5 dargestellt. Hierbei erhält man den tiefsten SUS-Score von 47.5 von Teilnehmer 6 und den höchsten mit 90 von Teilnehmer 2. Somit wäre nach Bangor et al.[BKM09] die Benutzerfreundlichkeit nach Teilnehmer 6 als schlecht dargestellt und nach Teilnehmer 2 als eine gute Benutzerfreundlichkeit. Betrachtet man nun alle SUS-Scores der Teilnehmer und bestimmt den Mittelwert, erhalten wir einen insgesamten SUS-Score von 70,4, was auf gute Benutzerfreundlichkeit schließen lässt[BKM09]. Nach der SUS in Abbildung 4.6, wird diese Bewertung nochmal dargestellt. Als amerikanische Schulnote hätte diese Erweiterung ein C erhalten, beziehungsweise ein „Gut“.

Um einen besseren Überblick zu haben, worin die Teilnehmer laut der SUS eine gute, beziehungsweise eine schlechte Erfahrung gemacht haben, wird die Durchschnittsantwort zu jeder Frage graphisch dargestellt. Hierbei wurden die negativ formulierten Fragen invertiert um die Durchschnittsantworten visuell besser darzustellen. Somit wird die Frage „Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen“ zu „Ich denke, dass ich keinen technischen Support brauchen würde, um das System zu nutzen“ und die Bewertung werden ebenfalls invertiert. Somit wird eine 0 zu einer 4, eine 1 zur 3, eine 2 bleibt 2, die 3 wird zur 1 und die 4 wird zur 0. Dadurch hat jede Frage einen Wert von 0 bis 4, wobei 0 eine schlechte Eigenschaft hat und 4 die beste. Das daraus resultierte Ergebnis wird in Abbildung 4.7 dargestellt. Mithilfe dieser Darstellung ist es nun möglich, die beantworteten Fragen der SUS von allen Teilnehmern auf Konsistenz zu überprüfen.

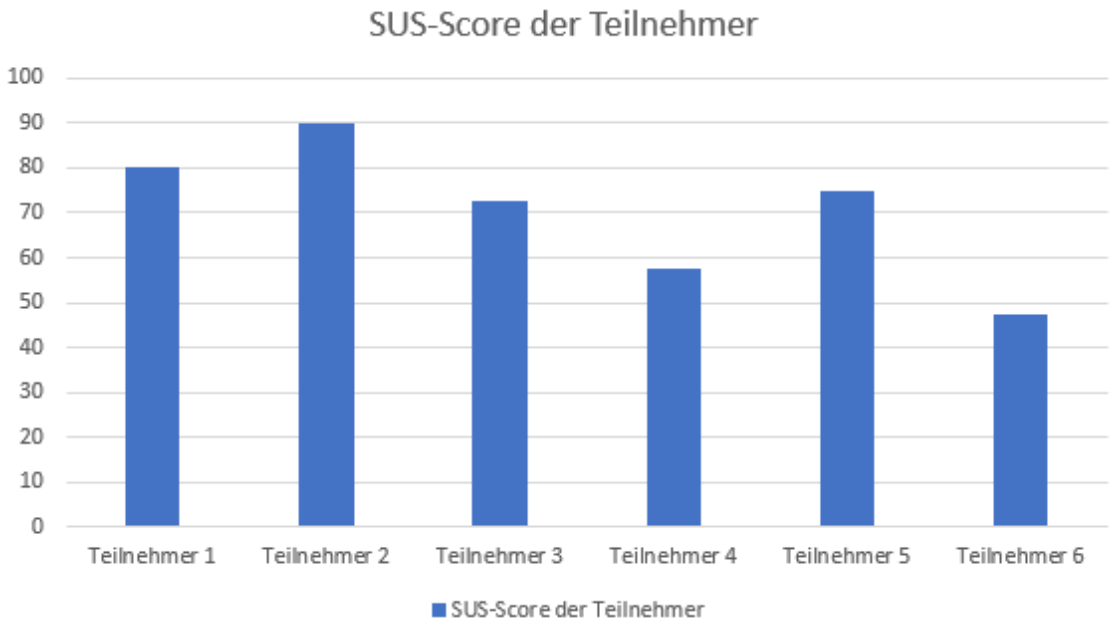

Abbildung 4.5: SUS-Score der einzelnen Teilnehmer

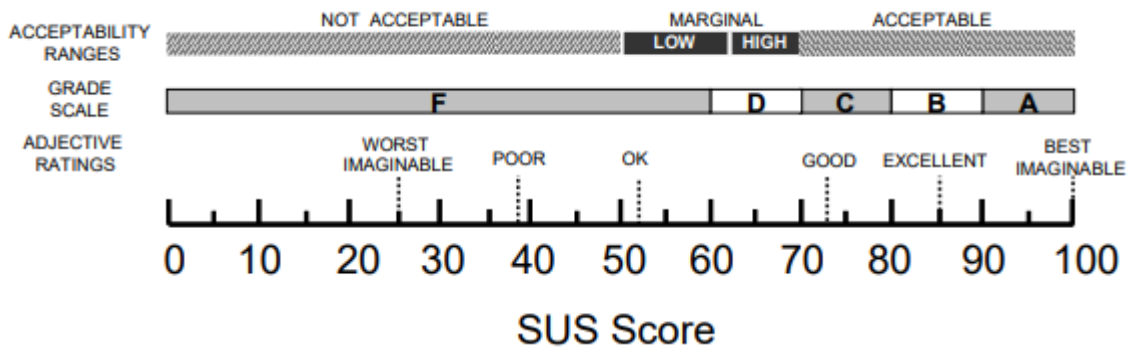

Abbildung 4.6: SUS Referenz nach Bangor et al.[BKM09]

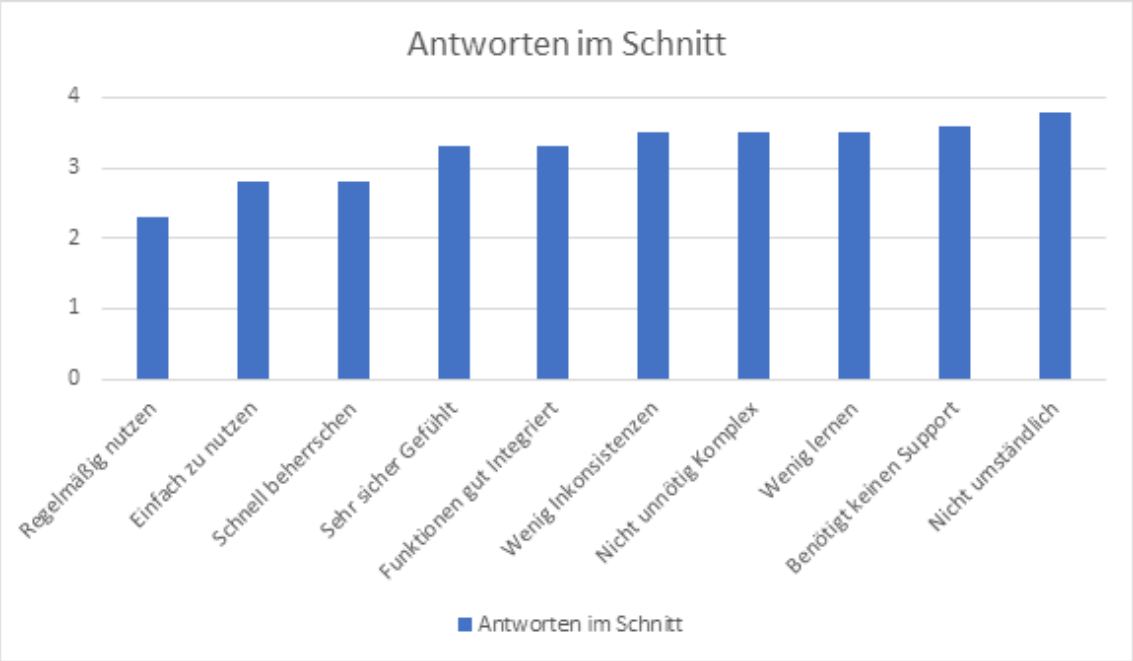

Abbildung 4.7: Antworten des SUS-Fragebogens im Durchschnitt

Auf die Frage, ob die Teilnehmer die Nutzung der Erweiterung als nicht umständlich empfanden, wurde mit einem Score von 3,8 beantwortet und ist somit fast die beste Eigenschaft. Um die Konsistenz dieser Frage zu prüfen, muss man nun die Frage betrachten, ob die Teilnehmer sich sicher bei der Nutzung gefühlt haben und ob, sie Hilfe bei der Lösung von Aufgaben in dieser Erweiterung benötigen. Im Bereich der gefühlten Sicherheit bei der Nutzung gaben die Teilnehmer einen Score von 3,3 und bei der Frage um Hilfe einem Score von 3,6. Hierdurch wird sichtbar, dass die Antworten zu diesen Fragen konsistent sind, da sie sich im selben Wertebereich befinden. Auch zu den Fragen, ob die Funktionen gut integriert sind, ob es wenig Inkonsistenzen gab und ob diese Erweiterung nicht komplex aufgebaut ist, sieht man ein ähnliches Muster. Die erste Frage hat einen Score von 3,3 erhalten, während die anderen beiden einen Score von 3,5 erhalten haben. Somit sind diese auch konsistent von den Teilnehmern beantwortet worden.

4.3.2 Qualitative Auswertung

Nach der quantitativen Auswertung folgt die qualitative Auswertung. Am Ende des Fragebogens gab es eine extra Frage, in welcher die Teilnehmer ihre subjektive Meinung zu der Erweiterung und des Eingabegeräts Leap Motion äußern können. Dazu gab es noch die Möglichkeit Verbesserungsvorschläge zu geben. Auf diese wird im Folgenden eingegangen.

Die Meinung der Teilnehmer zu der Erweiterung war eindeutig. 5 von 6 Teilneh-

mern fanden die Leap Motion als Eingabegerät gut. Es wurde erwähnt, dass sie bis jetzt nur mithilfe eines Smartphones mit Gestensteuerung Erfahrung gesammelt haben. Anders als die Teilnehmer es gewohnt sind, wird für die Leap Motion mehr körperliche Bewegung gefordert. Bei einer Dauer von circa 25.5 Minuten zur Lösung der beiden Aufgaben hatten die Teilnehmer nach der Durchführung schwere Arme gehabt, weil sich die Hände dauerhaft vor dem Leap Motion Sensor auf Kopfhöhe befinden müssen, was sich als anstrengend herausgestellt hat. Somit kann sich auch die Bewertung der Frage, ob das System einfach zu Nutzen sei, mit einem Score von 2,8 erklären. Hierbei wurde aber erwähnt, dass sich dies beim öfteren Benutzen dieser Erweiterung legen würde.

Ein Teilnehmer fand die Leap Motion als Eingabegerät eher unpassend. Er würde zu so einer Erweiterung eher die VR-Controller oder eine Tastaur und Maus Steuerung befürworten, da er den körperlichen Aufwand dieser Aufgabe auf Dauer niemanden zumuten würde.

Besonders positiv wurde das Erlernen der Steuerung bewertet. Während der 20 Minuten Probezeit wurden die Möglichkeiten die diese Erweiterung bietet getestet und konnten ohne Nachzufragen erneut vollführt werden. Die Steuerung wurde als sehr intuitiv dargestellt.

Ebenso gab es Verbesserungsvorschläge, welche im Folgenden aufgelistet werden:

- Bei der Selektierung von Knoten sollte nicht nur der Knoten „leuchten“, sondern auch mithilfe eines Textfeldes oder ähnlichem gezeigt werden, welcher Knoten selektiert wurde, falls man sich von diesem Knoten weit entfernt hat. Dies könnte man dann an das Button-Raster an der linken Hand mithilfe eines *GameObject*, inklusive einer *TextMesh* Komponente für den Namen des Knotens, fixieren.

- Bei der Löschung von Knoten sollte vorher abgefragt werden, ob dieser Knoten wirklich gelöscht werden soll, damit ein nicht gewolltes Löschen verhindert werden kann.
  Hierfür könnte man am Button-Raster neben dem *Delete Node*-Button einen Bestätigungsbutton erscheinen lassen, wenn dieser Button betätigt worden ist. Durch betätigen des neuen Buttons wird dann der selektierte Knoten gelöscht.

- Für das erneute Skalieren eines Knotens sollte man die Seiten des Knotens auswählen können, anstatt mithilfe der X-/ und Y-Buttons die Richtung zu bestimmen. Dies würde die Steuerung der Skalierung intuitiver machen.

- Eine Möglichkeit mehrere Architekturen zu importieren wurde vorgeschlagen. Hiermit ist gemeint, dass der Benutzer beispielsweise eine ältere Version und die aktuelle Version seiner Softwarearchitektur importieren und diese dann anschließend vergleichen kann.

- Bei einem Verbindungsabbruch der Leap Motion, wo sich die Hände nicht mehr im Sichtfeld befinden, sollte verhindert werden, dass ungewollte Gesten

entstehen und ungewollte Knöpfe betätigt werden.

- Die Nutzung von nur einem Sensor für die VR-Brille sorgt für Verbindungsabbrüche. Hier sollten mindestens zwei Sensoren benutzt werden.

Die angesprochenen Verbesserungsvorschläge sind nachvollziehbar und könnten in zukünftigen Versionen dieser Erweiterung umgesetzt werden. Das hier erhaltene Feedback wird in Kapitel 5.3 nochmal aufgenommen, um die Weiterentwicklung des Systems zu erläutern.

Threats to Validity

Mithilfe der *Threats to Validity*, oder auch Bedrohung der Gültigkeit, will man herausfinden, inwiefern die Ergebnisse einer Evaluation gültig sind. Hierfür unterscheidet man zwischen verschiedenen Bereichen: Einmal der internen und einmal der externen Gültigkeit. In den beiden Bereichen existieren nun verschiedene Faktoren, die die Gültigkeit der Ergebnisse beeinflussen können. Die interne Gültigkeit betrachtet Faktoren, welche direkten Einfluss auf das Ergebnis haben können. In diesem Fall also der SUS-Score. Die externe Gültigkeit betrachtet Faktoren, inwiefern diese Ergebnisse auf globaler Ebene repräsentiert werden können[Woh+12].

Im Bereich der internen Gültigkeit unterscheidet man zunächst zwischen zwei Kategorien, einmal bei der Betrachtung von Bedrohungen bei nur einer Testgruppe und mehreren Testgruppen. In der Evaluation dieser Arbeit gab es nur eine Testgruppe, somit werden die Faktoren für mehrere Gruppen nicht betrachtet. Im folgenden werden die einzelnen Faktoren für eine Testgruppe kurz erklärt:

- **History:** Falls die Evaluation an verschiedenen Tagen stattfand und somit nicht alle Teilnehmer die gleichen Voraussetzungen hatten, kann dies einen Einfluss auf die Ergebnisse haben. Beispielsweise, wenn Teilnehmer die Evaluation nach einem langen Arbeitstag durchführen oder direkt am Morgen vor Arbeitsbeginn.
- **Maturation:** Die Dauer der Evaluation kann einen Einfluss auf die Ergebnisse haben. Falls die Teilnehmer sich langweilen oder einen Lernerfolg erleben, kann dies negative oder positive Auswirkungen haben.
- **Testing:** Falls die Aufgaben der Evaluation mehrmals von dem selben Teilnehmer durchgeführt werden, fallen die weiteren Ergebnisse anders aus, als das erste Ergebnis von diesem Teilnehmer. Es soll vermieden werden, den selben Test unter den selben Umständen erneut durchzuführen.
- **Instrumentation:** Dieser Faktor betrachtet die Art und Weise, wie Daten während der Evaluation gesammelt werden. Also beispielsweise durch Fragebögen oder Aufnahmen. Sind diese schlecht umgesetzt worden, werden sie die Ergebnisse auch schlecht beeinflussen.

- **Selection:** Hierbei wird betrachtet, was für einen Einfluss die Auswahl der Teilnehmer auf das Ergebnis hat. Bei einer Gruppe von freiwilligen Teilnehmern werden die Ergebnisse beispielsweise anders beeinflusst, als bestimmte ausgewählte Personen aus einer großen Gruppe von Personen.
- **Mortality:** Falls es während der Evaluation dazu kommt, dass Teilnehmer diese abbrechen, muss betrachtet werden, zu welcher Personengruppe diese angehören. Wenn es beispielsweise bei der Betrachtung von Software zu Abbrüchen von erfahrenen Entwicklern kommt, kann dies die Gültigkeit der Ergebnisse stark beeinflussen.

Nun werden die einzelnen Faktoren der externen Gültigkeit kurz erklärt, diese beziehen sich auf den Bereich Teilnehmer, Ort und Zeitpunkt der Evaluation:

- **Interaction of selection and treatment:** Die Ausgewählte Gruppe von Teilnehmern ist nicht repräsentativ für die Gruppe, auf die sich diese Evaluation beziehen soll. Beispielsweise können an einer Softwareinspektion nicht nur Programmierer teilnehmen sondern auch Tester oder System-Ingenieure.
- **Interaction of setting and treatment:** Falls sich die Evaluation nahe an der industriellen Praxis halten soll und man beispielsweise veraltete Technik verwendet, obwohl in der industriellen Praxis die neuste Technik verwendet wird, kann es einen negativen Effekt auf die Ergebnisse haben.
- **Interaction of history and treatment:** Falls die Evaluation an einem bestimmten Tag oder Zeitpunkt stattfindet, kann es einen Einfluss auf das Ergebnis haben. Wenn es beispielsweise bei einer Befragung über eine bestimmte Software während der Befragungszeit zu einem Absturz kommt, würde der Absturz die Teilnehmer beeinflussen.

Das Ziel ist es nun abzuwägen, inwiefern diese Bereiche und Faktoren einen Einfluss auf das Ergebnis hatten. Da es sich in dieser Evaluation darum handelt, was für einen Einfluss ein bestimmtes Eingabegerät auf die Benutzbarkeit für diese Erweiterung hat, sollte der Fokus auf die interne Gültigkeit gelegt werden, da diese einen größeren Einfluss auf die Benutzbarkeit haben können[CC79].
Es werden nun alle genannten Faktoren betrachtet und aufgezeigt, inwiefern diese einen Einfluss auf das Ergebnis hatten:

- **History:** Die Evaluation aller Teilnehmer fand an einem Sonntag Mittag/Nachmittag statt. Jeder Teilnehmer hatte einen Zeitslot ausgewählt der für ihn passt um somit das Wohlbefinden der Teilnehmer zu gewährleisten. Dies sollte keinen negativen Einfluss auf das Ergebnis haben.
- **Maturation:** Im Schnitt dauerte die Evaluation 25,5 Minuten pro Teilnehmer. Durch die aufgetretenen Verbindungsabbrüche der HTC-Vive während der Durchführung der Aufgaben, wurde die Dauer der Evaluation beeinflusst. Hierdurch wurde das Ergebnis durch mehrmaliges neu positionieren vor dem Sensor definitiv negativ beeinflusst.

- **Testing:** Da der einzige Unterschied zwischen den zwei Aufgaben das Benutzen von anderen Steuerungsmöglichkeiten war, welche in der ersten Aufgabe nicht benutzt worden sind, hatten die Teilnehmer die Möglichkeit, sich durch die erste Aufgabe für die zweite Aufgabe vorzubereiten. Hierdurch gab es einen positiven Einfluss auf das Ergebnis der zweiten Aufgabe, da die Teilnehmer die Grundsteuerung nun beherrscht haben.

- **Instrumentation:** Die Aufnahme von Daten der Teilnehmer fand mithilfe von Bildschirmaufnahmen und eines Fragebogens statt. Hierbei wurde ein Fragebogen verwendet, der wissenschaftlich als Nutzbar für die Messung von Benutzbarkeit dargestellt worden ist. Hier sollte es keinen negativen Einfluss gegeben haben.

- **Selection:** Durch die COVID-19 Pandemie war man gezwungen, die Evaluation auf den Bekanntenkreis zu reduzieren und konnte somit keine Freiwilligen an der Universität aus dem richtigen Fachbereich suchen, wodurch das Ergebnis negativ beeinflusst worden ist.

- **Mortality:** Während der Evaluation gab es keinen Teilnehmer der den Durchlauf abgebrochen hat. Somit hat dieser Faktor keinen Einfluss auf das Ergebnis.

- **Interaction of selection and treatment:** Wie im Faktor **Selection** erwähnt, musste auf den Bekanntenkeis zurückgegriffen werden, wodurch die Testgruppe nicht genau der Zielgruppe entspricht, welche angepeilt worden ist. Somit ist das Ergbnis nicht aussagekräftig im Bezug auf der gesuchten Zielgruppe.

- **Interaction of setting and treatment:** Im normalen Fall wird die HTC-Vive mit zwei Sensoren geliefert. In diesem Fall stand der Evaluation nur einen Sensor zur Verfügung, wodurch es zu Problemen während der Durchführung kam. Somit hatte dieser Faktor einen negativen Einfluss auf das Ergebnis.

- **Interaction of history and treatment:** Bei den ersten zwei Teilnehmern kam es zu keinen Verbindungsabbrüchen während der Durchführung der Evaluation. Somit hatten diese Teilnehmer eine andere Erfahrung sammeln können als die anderen Teilnehmer. Diese Situation kann man in Abbildung 4.5 betrachten.

Die interne Gültigkeit wurde somit durch drei Faktoren bedroht, welche aber nicht verhindert werden konnte, da es Einschränkungen durch die Technologie und der COVID-19 Pandemie gab. Hätte die Evaluation an der Universität stattgefunden, wäre man in der Lage gewesen, den negativen Einfluss dieser Faktoren zu verhindern. Trotzdem war es aber möglich, diese Ergebnisse vernünftig auszuwerten.
Alle Faktoren der externen Gültigkeit hatten einen negativen Einfluss auf das Ergebnis gehabt. Dadurch ist man in der Lage zu behaupten, dass das Ergebnis nicht auf globaler Ebene aussagekräftig ist.

Kapitel 5

Fazit

5.1 Zusammenfassung der Ergebnisse

Das Ziel der Evaluation war es, die Erweiterung zu testen, ob die Leap Motion als Eingabegerät aus der Benutzersicht geeignet ist. Mithilfe des System-Usability-Scores wurde gezeigt, dass diese Erweiterung von den Teilnehmern einen Score von 70.4 erhalten hat und somit als akzeptabel angesehen werden kann. Hierbei ist aber noch zu beachten, das es nur sechs Teilnehmer an der Evaluation gab. Somit ist die Aussagekraft dieses Ergebnis eher schwach, aber sollte sich bei einer höheren Teilnehmerzahl im selben Bereich wiederfinden.

Es lässt sich somit behaupten, dass die Leap Motion als Eingabegerät für diese Erweiterung aus der Sicht der Benutzerfreundlichkeit einen guten Zweck erfüllt. Um aber ein genaueres Feedback zu bekommen, sollte es einen Vergleich mit einem anderen Eingabegerät geben. Dies hat es im Rahmen der Bearbeitungszeit und der zu dieser Zeit stattgefundenen Corona-Pandemie leider nicht in diese Bachelorarbeit geschafft.

5.2 Persönliches Fazit

Wie in Kapitel 5.1 erwähnt, ist die Benutzerfreundlichkeit der Erweiterung mit der Leap Motion akzeptabel. Da diese Steuerung als akzeptabel bewertet wurde, ist die Arbeit meiner Ansicht nach ein Erfolg gewesen. Es gab natürlich Verbesserungsvorschläge der Teilnehmer der Evaluation um die Steuerung intuitiver, einfacher und sicherer zu gestalten. Im Grunde sind aber die Funktionen wie sie sich im aktuellen Zustand befinden, voll funktionsfähig und im Rahmen einer Bachelorarbeit gut geschafft.

Es freut mich zu wissen, dass es wenig Fehler in der Erweiterung gibt und keine existieren, welche dem Spielverlauf der Erweiterung beeinträchtigen. In der Evalua-

tion ist zumindest kein solcher Fehler aufgetreten.

Das persönliche Feedback der Teilnehmer war ebenso erfreuend für mich. Dass es möglich ist, mit einer Gestensteuerung solch ein Erlebnis zu schaffen, hätten sie sich nicht gedacht. Somit lege ich einen großen Wert auf die Zukunft der Gestensteuerung, da diese meiner Meinung nach schnell erlernbar und intuitiv sein können.

Leider konnte ich gegen das negative Feedback durch die Nutzung von nur einem Sensor für die HTC-Vive nicht lösen, da die Unversität zum Wechsel in den Notbetrieb nur noch einen Sensor auf Lager hatten. Hierdurch kam es vermehrt zu grauen Bildschirmen in der HTC-Vive während der Lösung der Aufgaben, was sich natürlich in der Bewertung des SUS widergespiegelt hat.

Somit würde ich diese Arbeit als ein Erfolg bezeichnen, da alle Ziele, die zu einer funktionierenden Erweiterung nötig waren, auch umgesetzt worden sind.

5.3 Ausblick für die Zukunft

Da das SEE-Projekt sich in stetiger Weiterentwicklung befindet, muss sich diese Erweiterung immer den Änderungen anpassen. Ebenso können auch weitere Features implementiert werden, die Probleme aus Kapitel 3.3.5 gelöst werden oder auch auf die Vorschläge der Teilnehmer eingegangen werden aus Kapitel 4.3.2.

Der Hauptkritikpunkt meiner Meinung nach sind die Verbindungsabbrüche der Leap Motion. Hierbei sollte der Fokus bei der Weiterarbeit an dieser Erweiterung liegen, diese so zu behandeln, dass keine ungewollten Gesten und Betätigungen von Buttons geschehen.

Um das Aussehen von Knoten könnte man sich ebenso Gedanken machen. Vielleicht könnte man diese auch einen bestimmten Stil geben wie auch im SEE-Projekt die Code-Cities. Darauf folgend könnte man die vorhandene Funktion der visuellen Darstellung von Codeänderungen auch für Softwarearchitekturänderungen implementieren und somit beide Änderungen betrachten können.

Auch ein Ziel dieser Arbeit war es, der Arbeit von David Wagner eine visuelle Darstellung einer Softwarearchitektur zu bieten. Somit sollten diese beiden Projekte in Zukunft zusammenarbeiten, um eine jeweils passende Darstellung für jede Situation schaffen zu können.

Abbildungsverzeichnis

Literatur

[AXI] AXIVION. *Axivion Suite*. URL: https://www.axivion.com/de/p/produkte/architektur-analyse-169.html (besucht am 23. 07. 2020).

[BKM09] Aaron Bangor, Philip Kortum und James Miller. „Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale“. In: 4.3 (Mai 2009), S. 114–123. ISSN: 1931-3357.

[Bro96] John Brooke. *SUS - A quick and dirty usability scale*. 1996. URL: https://hell.meiert.org/core/pdf/sus.pdf (besucht am 22. 06. 2020).

[CC79] Thomas D Cook und D T Campbell. *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. English. Houghton Mifflin, 1979.

[Gle15] Corey Sandler Glenford J. Myers Tom Badgett. „Usability (User) Testing“. In: *The Art of Software Testing*. John Wiley & Sons, Ltd, 2015. Kap. 7, S. 143–155. ISBN: 9781119202486. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119202486.ch7 (besucht am 22. 06. 2020).

[Hel] Hello2morrow. *Sotograph*. URL: https://www.hello2morrow.com/products/sotograph (besucht am 29. 06. 2020).

[HS88] Sandra G. Hart und Lowell E. Staveland. „Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research“. In: *Human Mental Workload*. Hrsg. von Peter A. Hancock und Najmedin Meshkati. Bd. 52. Advances in Psychology. North-Holland, 1988, S. 139–183. DOI: https://doi.org/10.1016/S0166-4115(08)62386-9.

[IEE00] IEEE. „IEEE Recommended Practice for Architectural Description for Software-Intensive Systems“. In: *IEEE Std 1471-2000* (Okt. 2000), S. 1–30. DOI: 10.1109/IEEESTD.2000.91944.

[Inc] Lattix Inc. *Lattix Architect*. URL: https://www.lattix.com/products-architecture-issues/ (besucht am 29. 06. 2020).

[Kos13] Rainer Koschke. „Incremental reflexion analysis“. In: *Journal of Software: Evolution and Process* 25.6 (2013), S. 601–637. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.542.

[LHS08] Bettina Laugwitz, Theo Held und Martin Schrepp. „Construction and Evaluation of a User Experience Questionnaire“. In: Bd. 5298. Nov. 2008, S. 63–76. DOI: 10.1007/978-3-540-89350-9_6.

[Lik32] R Likert. „A technique for the measurement of attitudes“. In: Archives of Psychology.140 (1932).

[Lim+19] Daniel Limberger u. a. „Advanced Visual Metaphors and Techniques for Software Maps“. In: *Proceedings of the 12th International Symposium on Visual Information Communication and Interaction*. VINCI'2019. Shanghai, China: Association for Computing Machinery, 2019. ISBN: 9781450376266. URL: https://doi.org/10.1145/3356422.3356444.

[MNS01] G. C. Murphy, D. Notkin und K. J. Sullivan. „Software reflexion models: bridging the gap between design and implementation“. In: *IEEE Transactions on Software Engineering* 27.4 (2001), S. 364–380.

[Ric] Andy Schürr et al. Ric Holt. *Graph eXchange Language*. URL: http://www.gupro.de/GXL/ (besucht am 29. 06. 2020).

[Tec] Unity Technologies. *Unity3D Game Engine*. URL: https://unity.com/de (besucht am 29. 06. 2020).

[ult] ultraleap. *Leap Motion Controller*. URL: https://www.ultraleap.com/product/leap-motion-controller/ (besucht am 29. 06. 2020).

[Woh+12] Claes Wohlin u. a. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. ISBN: 3642290434.