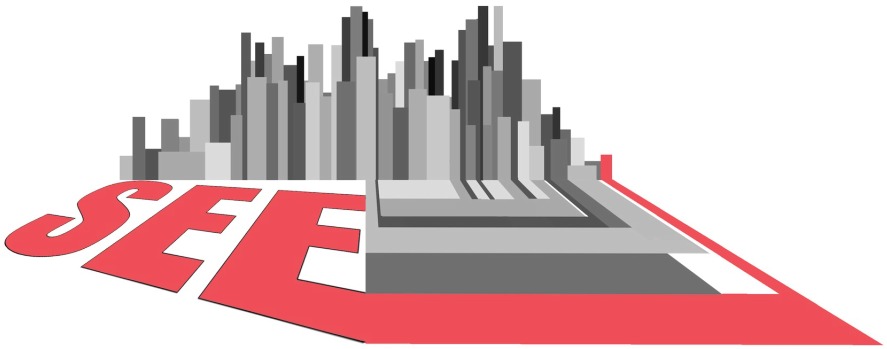

Projekt SEE

Software-Entwicklung im 21. Jahrhundert

Moritz Blecker	Lino Goedecke	Nick Seedorf
Thore Frenzel	Leonard Haddad	Robin Theiß
Thorsten Friedewold	Simon Leykum	Sören Untiedt
Falko Galperin	Christian Müller	Jan Woiton

31. Mai 2021

Inhaltsverzeichnis

Gender-Hinweis

Aus Gründen der besseren Lesbarkeit wird auf die gleichzeitige Verwendung der Sprachformen männlich, weiblich und divers verzichtet. Sämtliche Personenbezeichnungen gelten gleichermaßen für alle Geschlechter.

1 Abstract

Vor dem Hintergrund aufkommender Komplexität beim Debuggen von Software im Allgemeinen, d.h. der Implementierung der vorausgegangenen Architekturbeschreibung, und der Tatsache, dass oft mehrere Entwickler parallel an konkreten Implementierungen arbeiten und sich darüber austauschen wollen, entstand unter der Leitung von Prof. Dr. Koschke die Idee des studentischen Projekts SEE. SEE ist ein auf der Game Engine Unity basierendes Programm zur Visualisierung von Software, das für die kollaborative Entwicklung auf verschiedenen Plattformen genutzt werden kann, um nicht nur den Status quo, sondern auch die vorausgegangene Entwicklung eigener Software zu analysieren.

In diesem Abschlussbericht des studentischen Bachelorprojekts wird erörtert, auf welchen aktuellen Forschungsständen aufgebaut wurde, wie in dem studentischen Projekt die Software von SEE als konzertiertes, kollaboratives Projekt konsekutiv um weitere Features erweitert wurde und abschließend, welche möglichen Veränderungen und Erweiterungen es in der Zukunft geben soll. Aufbauend auf den vorherigen Arbeiten des studentischen Projekts arbeiteten zwölf Entwickler inklusive der zwei Betreuer an der Entwicklung der Software und führten einerseits gänzlich neue Interaktionen ein und überarbeiteten und verbesserten andererseits bestehende Komponenten. Folgende Inhalte und Funktionen wurden im Rahmen des Bachelorprojekts SEE umgesetzt:

- Es wurde ein vollständig neues User-Interface-Framework integriert, welches plattformunabhängig eingesetzt werden kann. Auf Grundlage dessen konnte ein Player-Menu erarbeitet werden, welches dem Nutzer Interaktionen mit den Code-Cities ermöglicht, wie z. B. die Darstellung des Sourcecodes einzelner Komponenten, das Hinzufügen neuer Kanten, das Hinzufügen und Bearbeiten von Knoten, das Skalieren von Knoten und auch das Löschen von Knoten und / oder Kanten.
- Des Weiteren wurden alle Funktionen im Multi-User-Betrieb des Player-Menus implementiert. Hierfür wurde ein SEE-internes Netzwerkframework genutzt, welches nun das ortsunabhängige, kollaborative Arbeiten verschiedener Entwickler in Echtzeit ermöglicht.
- Die Entwicklung an dem Bereich der Software-Evolution lässt sich in zwei Bereiche unterteilen. Zum einen wurde das User-Interface zur Interaktion und zum Highlighten von Veränderungen überarbeitet, zum anderen wurden gänzlich neue Funktionen integriert, aber auch vorhandene verfeinert.
- Als ein weiteres Ziel des Projekts galt es, SEE nicht nur auf dem Desktop und in der virtuellen Realität, sondern auch in Augmented Reality (AR) nutzen zu können.

Deshalb wurde SEE für die HoloLens 2 konfiguriert und ist nun auch in der AR verfügbar.

- Damit SEE langfristig auch in der Praxis eingesetzt werden kann und um den Stand der Forschung präsentieren zu können, wurden auch einige Dinge zur Außendarstellung von SEE unternommen, wie die Erstellung einer Webseite sowie die Präsentation des Projekts vor externen Vertretern und auch anderen Informatikern. In Zukunft soll die Software auf Praxistauglichkeit überprüft werden und Hand in Hand mit Nutzern weiterentwickelt werden.

2 Einleitung

2.1 Motivation

Kollaborative Softwareentwicklung gewinnt seit einigen Jahren zunehmend an Relevanz. Das hängt unter anderem damit zusammen, dass Projekte immer komplexer werden, Softwarelösungen immer umfangreicher und so das Klischee des Programmierers, der isoliert an seiner Software arbeitet, nicht mehr aktuell ist. Die anhaltende Corona-Pandemie intensiviert diesen Trend zunehmend, da Entwickler nun häufig lediglich über ihre Endgeräte mithilfe des Internets mit anderen Entwicklern kommunizieren können. Zusammenarbeit bei der Entwicklung bietet viele Vorteile, stellt die Teilnehmer jedoch vor einige Herausforderungen.

- **Komplexität von Software:** Quellcode anderer Entwickler vollständig zu durchdringen ist häufig zu komplex und gar nicht zwangsläufig zielführend. Es muss vielmehr ein Austausch über Pakete, Komponenten und Aufrufstrukturen zwischen diesen Komponenten auf höherer Abstraktionsebene bestehen.

- **Paralleles Arbeiten:** Das Arbeiten mithilfe von Tools wie *Zoom*[1], Screensharing innerhalb dieser und Versionskontrollsystemen wie *Git*[2] bietet zwar die Möglichkeit, dieselben Elemente wie andere Entwickler zu sehen, ermöglicht jedoch keine Möglichkeit des Eingriffs und berücksichtigt auch keine aktuellen Veränderungen im eigenen Quellcode, ohne umständliches „Pushen“ und „Pullen“ mithilfe der Versionskontrolle.

- **Kommunikationskanäle:** Durch Desktopumgebungen und Webcams sowie Headsets wird zwar grundlegende Kommunikation zwischen Entwicklern unterstützt, es besteht allerdings keine Möglichkeit Software „anzufassen“ oder ihre Architektur gemeinsam im Raum zu betrachten.

Sinnvolle, alle Herausforderungen umfassende Lösungen für diese Probleme existieren bisher nur in geringem Maße, hiervon umfassen die meisten Angebote jedoch lediglich

[1] https://zoom.us/ (letzter Abruf: 31.5.2021)
[2] https://git-scm.com/ (letzter Abruf: 30.05.2021)

kleine Teile der Anforderungen und bieten kaum Integrationsoptionen für Lösungen anderer Probleme.

Aus diesen Herausforderungen begann die AG Softwaretechnik der Universität Bremen um Prof. Dr. Rainer Koschke mit der Entwicklung der Software SEE — Software Engineering Experience — welche kollaboratives Arbeiten an Software auf verschiedenen Endgeräten in AR/VR und auf der herkömmlichen Desktopumgebung ermöglichen soll und Lösungsansätze für alle im oberen Bereich beschriebenen Probleme bieten soll. Auf die detaillierteren Funktionen soll im Folgenden noch genauer eingegangen werden (siehe Sektion 2.2).

Das Ziel des Bachelorprojektes SEE war es nun also, die bestehende Basis von bereits mehr als 30.000 Zeilen Code zu durchdringen und um viele nützliche Features zu erweitern, sodass die Software SEE langfristig zur kollaborativen Softwareentwicklung und dem besseren Verständnis komplexer Software genutzt werden kann. Die detaillierten Ziele werden in ihrem vollen Umfang in den nachfolgenden Kapiteln thematisiert, nachdem SEE und die Ausgangslage von SEE zum Projektbeginn zunächst genauer beschrieben wurden.

2.2 SEE — Ausgangslage

Die Idee dieses Projekts basierte auf der Erstellung einer virtuellen Umgebung, die beispielsweise Büros im Homeoffice ersetzen könnte. In dieser virtuellen Umgebung sollte man sowohl die Möglichkeit der Kommunikation, als auch die des Aufbereitens des eigenen Codes mithilfe einer visuellen Darstellung haben. Diese Darstellung stützt sich dabei auf das Modell der Code-Cities (siehe Sektion 3), welches dem Nutzer ermöglichen kann den eigenen Code als ein Stadtlayout mit Gebäuden anzeigen zu lassen. Außerdem wird mit dem Projekt SEE ebenfalls die unterschiedliche Nutzbarkeit mit verschiedenen Endgeräten genauer untersucht, hierbei liegt der Fokus vor allem auf VR- und AR-Technologien. Dabei wird untersucht, ob AR/VR in diesem Zusammenhang einen positiven Beitrag leisten können.

Zu Beginn des Bachelorprojekts arbeiteten wir uns deshalb mit einer auf Unity basierenden Version von SEE ein. Ursprünglich wurde für das Projekt die Game-Engine *Unreal Engine 4*[3] genutzt, aber aufgrund günstigerer Voraussetzungen wurde SEE zur Game-Engine *Unity*[4] transferiert. Da SEE zu diesem Zeitpunkt eine Art Plattform für mehrere Forschungsarbeiten war, gab es schon zahlreiche fundamentale Funktionen. Somit war es bereits möglich, eine Main-Scene rendern zu lassen und sich in dieser zu bewegen. Auch waren bereits erste Hover-Effekte und Features zur Darstellung von Kanten (siehe Sektion 4.6.3) implementiert. Dadurch war es ebenso möglich, eine Code-City zu rendern und diese auf Attribute wie „Name“ und „Position“ zu untersuchen. Die eindrucksvollste Funktion war es jedoch, das Spiel mithilfe einer VR-Brille starten zu können. Damit war SEE eine geeignete Basis zur Entwicklung der vier wesentlichen Anwendungsfälle:

[3] https://www.unrealengine.com/ (letzter Abruf: 31.5.2021)
[4] https://unity.com/(letzter Abruf: 30.05.2021)

1. Die erste Kategorie von Anwendungsfällen ist hierbei der Architekturvergleich. Hierbei ist die Vergleichbarkeit des *Ist*- und des *Soll*-Zustands der wesentliche Schwerpunkt.

2. Der nächste Anwendungsfall ist die Evolution, welche es ermöglicht, den Werdegang einer Software darzustellen. Dies wird mithilfe einer sich dauerhaft verändernden Stadt dargestellt, bei der Änderungen mit farblicher Untermalung symbolisiert werden.

3. Beim Debugging hingegen ist es möglich, Kommunikation innerhalb eines Programmes in der visualisierten Stadt nachzuverfolgen. Dabei liegt das Hauptaugenmerk auf der Kommunikation der einzelnen Komponenten zur Laufzeit.

4. Im letzten Anwendungsfall geht es um die Qualitätsmetriken. Diese zeigen an, wie viele Codesemells, wie z. B. Code-Duplikate der jeweilige Code hat. Je dunkler die Rottöne dargestellt werden, desto schwächer ist die innere Qualität des Codes.

Auf dieser Grundlage baute das Bachelorprojekt SEE im weiteren Verlauf auf und formulierte seine Ziele.

2.3 Ziele

Die Zielsetzung von SEE wurde zum Beginn des Projektes in einer Einführungsveranstaltung dargestellt. So können die Ziele in drei Kategorien unterteilt werden:

1. Die Erweiterung der bestehenden Architekturprüfung (Sektion 2.3.1)

2. Neue Funktionen und eine intuitivere Benutzeroberfläche für die Evolution (Sektion 2.3.2)

3. Die Unterstützung der HoloLens 2 (Sektion 2.3.3)

2.3.1 Architekturprüfung

Die Architekturprüfung beschreibt den Vergleich von der geplanten Architektur mit der aktuell vorliegenden, umgesetzten Architektur. So soll es z. B. ermöglicht werden, die Architektur mit mehreren Teilnehmern zu besprechen und zu bearbeiten. Außerdem soll es ermöglicht werden, den Quelltext von Knoten anzuzeigen, Divergenzen und Konvergenzen einzusehen sowie eine Visualisierung von Code-Metriken zu integrieren. Dabei wurden die folgenden Ziele zu Beginn festgelegt:

- Modellierung und Veränderung der Architektur
  - Hinzufügen neuer Knoten und Kanten
  - Löschen existierender Knoten und Kanten
  - Ändern der Schachtelung

- Anzeigen des Quelltexts der Komponenten
- Abbildung von Knoten der Implementierung auf Knoten der Architektur durch *Drag&Drop* (VR) bzw. *Copy&Paste* (Desktop)
  - Räumliche Einbettung mit freier Positionierung und Skalierung
  - Kennzeichnung bereits abgebildeter Knoten
- Darstellung der Veränderungen der Reflexionsanalyse
  - Besondere Hervorhebung neuer und gelöschter Knoten und Kanten
  - Anzeigen des Quelltexts der Divergenzen
- Reaktion auf Reflexionsergebnis
  - Hinzufügen von Architekturkanten, um Divergenzen zu Konvergenzen zu machen
  - Beseitigen von absenten Architekturkanten
  - Verändern der Abbildung
- Integration der Metrik-Charts (z. B. Anzahl der Verstöße)
- Unterstützung des Mehrspielermodus

2.3.2 Evolution

Die Evolution beschreibt die Darstellung von Code-Cities und deren strukturelle Änderungen in Abhängigkeit der Zeit, bzw. genauer den Revisionen, die aufeinanderfolgend im Verlaufe der Zeit erstellt wurden. So soll es ermöglicht werden, wenn die Informationen der zeitlichen Änderung der City vorliegen, diese Änderung wie in einem Film animiert abspielen zu lassen. Dabei sollen Bedienelemente, wie man sie von der Videowiedergabe kennt, das Vor- und Zurückspulen, Pausieren und Starten ermöglichen. Die folgenden Punkte wurden dabei festgelegt:

- Einlesen von Metriken aus CSV-Dateien für jede Revision
- Auswahl der darzustellenden Knotentypen
- Mitbewegen existierender Kanten in Animation
- Hervorheben neuer, gelöschter und geänderter Kanten
- Bessere Hervorhebung neuer, gelöschter und geänderter Knoten
- GUI-Kontrollen wie bei einem Videoplayer (Start, Stopp, Timeline-Slider)
- Setzen von Markierungen in der Timeline, zu welchen gesprungen werden kann
- Integration der Metrik-Charts
- Unterstützung des Mehrspielermodus
- Inkrementelle Layouts der Graphen

2.3.3 HoloLens

Die HoloLens 2 soll ein noch intuitiveres Interagieren mit den Code-Cities ermöglichen. Dazu muss das bestehende Projekt auf die HoloLens portiert werden. Die bereits bestehenden Funktionen der Evolution und Architekturprüfung sollen dabei auf der HoloLens mit den dafür vorgesehenen holografischen Benutzersteuerungselementen dargestellt werden. Die folgenden Punkte wurden dafür zu Beginn festgelegt:

- Code-City im Raum beliebig platzieren und skalieren
- Zooming und Panning der Code-City
- Selektion von Komponenten in der Code-City
- Integration der Metrik-Charts
- Quelltext der Komponenten der Code-City anzeigen
- Unterstützung des Mehrspielermodus
- Im späteren Verlauf: Anwendungsfälle *Architekturprüfung* und *Evolution* unterstützen

Der zeitliche Rahmen erstreckt sich auf acht Monate, die für das Bachelorprojekt geplant waren. Innerhalb dieses Zeitrahmens war geplant, möglichst viele der oben genannten Neuerungen zu implementieren, wobei es auch zu Abweichungen kommen konnte, da zeitliche Verzögerungen oder Planungsänderungen das Bearbeiten von anderen auftretenden Aufgaben erfordern oder den Fokus auf andere Themen verschieben. Außerdem musste innerhalb dieser Zeit eine Abschlusspräsentation vorbereitet und auf dem Bachelorprojekttag vorgetragen werden sowie ein Abschlussbericht (dieses Dokument) angefertigt werden.

3 Forschungsantrag

Hinweis: Diese Sektion wurde von den Projektbetreuern Rainer Koschke und Marcel Steinbeck verfasst. Es handelt sich um einen Forschungsantrag.
Der Fokus unseres Antrags ist die Unterstützung kooperativer Verstehensprozesse in der Software-Wartung und -weiterentwicklung im Kontext verteilter Teams mit Hilfe neuer Medien und Technologien der Software-Visualisierung. Beim kooperativen Verstehen versuchen mehrere Beteiligte gemeinsam, sich ein Modell des Aufbaus und/oder des Verhaltens eines Programms zu erarbeiten. Dabei können sie wechselseitig von den unterschiedlichen Perspektiven und Expertisen anderer Gruppenmitglieder profitieren. Hierbei ist es meist hilfreich, ein gemeinsames Modell zu externalisieren – also für alle sichtbar und explizit zu machen, um die Gefahr von Missverständnissen und Fehlinterpretationen zu verringern. Häufig beginnen kooperierende Verstehende deshalb auf ganz natürliche Weise damit, ad hoc Skizzen auf einem Whiteboard zu erstellen, wenn sie

sich in einem gemeinsamen Raum befinden. Solche Modelle können zum Beispiel Aspekte des statischen Aufbaus eines Programms in Form von Architekturbeschreibungen oder auch des Verhaltens in Form von Aufrufsequenzen oder Zustandsautomaten sein. Dabei kommen sowohl informelle Notationen (z.B. Boxen und Pfeile) als auch formale Notationen (z.B. UML-Statecharts) zum Einsatz. Nicht selten werden die diskutierten Sachverhalte jedoch aus dem Gedächtnis der Beteiligten rekonstruiert, was in vielen Fällen nur eine ungenaue Idealisierung oder gar völlig Unzutreffendes wiedergibt. Zutreffende Informationen lassen sich für bestimmte Aspekte des Aufbaus und Verhaltens eines Programms mittels statischer oder dynamischer Programmanalysen gewinnen. Diese sind dann häufig jedoch zu detailreich und müssen erst zusammengefasst und abstrahiert werden, damit sie für die Beteiligten zugänglich und verständlich werden. Aus diesem Grund werden die Ergebnisse statischer und dynamischer Programmanalysen in der Regel grafisch aufbereitet. Agieren die Gruppenmitglieder im Verstehensprozess zudem räumlich verteilt, müssen Distanzen durch geeignete Technologien überbrückt werden. In der Praxis kommen hierbei zum Beispiel generische Videokonferenzsysteme oder geteilte virtuelle Whiteboards zum Einsatz.

Im beantragten Projekt werden wir die Frage untersuchen, wie kooperatives Programmverstehen in verteilten Teams in der Software-Wartung und -weiterentwicklung mithilfe von modernen konvergenten Visualisierungstechnologien besser unterstützt werden kann. Unter technologischer Konvergenz verstehen wir an dieser Stelle die enge Integration zuvor unzusammenhängender Technologien. Konkret wollen wir sowohl herkömmliche Desktop-Hardware (Computer, Tastatur und Maus) und Tablet-Geräte als auch fortschrittlichere Hardware für virtuelle (VR) und erweiterte (AR, engl. *augmented*) Realität so integrieren, dass Software-Entwicklerinnen und -Entwickler ein zweckdienliches, einheitliches und zutreffendes Bild ihrer Software über räumliche Distanzen hinweg bekommen. Es soll ein gemeinsamer virtueller Raum erschaffen werden, in dem sich die Beteiligten in exemplarischen Anwendungsszenarien in der Weiterentwicklung von Software verständigen können. Diesen virtuellen Raum können die Beteiligten mit dem Gerät ihrer Wahl (Desktop, Tablet, VR oder AR) „betreten“. Bei den exemplarischen Anwendungsszenarien werden wir ein großes Spektrum realistischer Anwendungsfälle in der Software-Weiterentwicklung abdecken. Dazu gehören das Debugging, die Analyse der inneren Qualität, die Nachvollziehung früherer Entwicklungen und die Architekturprüfung.

Der Trend zur Software-Entwicklung in verteilten Teams ist schon lange zu beobachten und wurde durch die aktuellen Einschränkungen der Pandemie nur noch einmal verstärkt [EKP16]. Es ist davon auszugehen, dass sich dieser Trend auch in der Zeit nach der Pandemie fortsetzt. Die Visualisierung von Software für Gruppen statt nur Einzelnen ist dabei wenig erforscht. Nach unserer Kenntnis gibt es keine Arbeiten, die Software-Visualisierung mit Hilfe der Integration der heute verfügbaren, heterogenen, oben angeführten Technologien untersucht haben.

Stand der Forschung und eigene Vorarbeiten

In diesem Abschnitt stellen wir den Stand der Forschung in den wesentlichen Bereichen dar, die für unseren Untersuchungsgegenstand besonders relevant sind. Dazu gehören Arbeiten zur Visualisierung von Software mittels der Stadt-Metapher und mit Hilfe von VR/AR. Visualisierungen auf Basis der Stadt-Metapher erfreuen sich in der Forschung und auch Praxis großer Beliebtheit, weshalb sie für uns den Ausgangspunkt unserer Visualisierung bilden. Wir wollen im Projekt keine gänzlich neue Visualisierung erschaffen, sondern vielmehr eine gut geeignete, bewährte Darstellungsform durch konvergente Technologien für die Zwecke des kooperativen Verstehens in Teams optimal weiterentwickeln. Aus diesem Grund legen wir auch einen besonderen Schwerpunkt in der Erörterung des Stands der Forschung auf Untersuchungen zu Visualisierungen für Gruppen statt einzelnen Betrachtern, d.h. solche, die kooperatives Verständnis zum Ziel haben. Da das Feld der Software-Visualisierung immens groß ist und sich nicht in diesem Antrag erschöpfend darstellen lässt, verweisen wir für einen allgemeinen Überblick zur Software-Visualisierung auf die einschlägigen Literaturübersichten [PBS93; BK01b; BK01a; Kos03b; CG08; TC09b; TC09a; Die01; Die10; CZ11; Nov+13; BHK15; Mer+18; Ham+20].

3.1 Code-Cities

In der Darstellung von Aspekten von Software muss häufig neben quantitativen Eigenschaften wie etwa der Größe einer Komponente auch die Hierarchie der Software, die sich zum Beispiel aus geschachtelten Namensräumen ergibt, wiedergegeben werden. Eine sehr frühe Darstellung, die beide Aspekte zugleich berücksichtigt, sind die so genannten *Treemaps* [JS91]. Hier wird die Hierarchie durch ineinander geschachtelte Rechtecke dargestellt. Eine Metrik für Blätter der Hierarchie (ein innerstes Rechteck) wird durch den Flächeninhalt des Rechtecks abgebildet. Es gibt eine Reihe von Layout-Algorithmen (z.B. *Slice&Dice*, *Squarified*, *Striped* etc.), die die Rechtecke letztlich aber alle so anordnen, dass die Treemap auf einer vorgegebenen Fläche platzsparend und auf einen Blick sichtbar gemacht wird. Sie variieren in Bezug auf andere Optimierungskriterien wie zum Beispiel das angestrebte Seitenverhältnis der Rechtecke. Den ursprünglich zweidimensionalen Treemaps kann noch eine Höhe der Rechtecke hinzugefügt werden, um eine weitere Metrik abzubilden. Solch dreidimensionale Treemaps erwecken bei einem menschlichen Betrachter das typische Bild einer nordamerikanischen Innenstadt, die aus gitterförmig angeordneten Hochhäusern besteht. Diese Ähnlichkeit hat dreidimensionalen Treemaps für Software den Namen *Code-City* beschert [AWP97]. Diese metaphorische Darstellung hat sich schnell nachhaltiger Beliebtheit in der Forschungsgemeinde erfreut [KM00; KM00; Cha+02; Bal+04; PBG03; MFM03; WL07; WL08b; WL08a; FRH15; FKH15a; BSB15; Mer+17a; MBN18; SWD18; Mer+19; Lim+19; SKR20]. Statt des ursprünglichen Treemap-Algorithmus wurden auch alternative Layouts und Ausdrucksformen entwickelt, die aber der City-Metapher selbst treu geblieben sind, wie zum Beispiel die *EvoStreets* [SL10; Ste13]. Durch zusätzliche visuelle Attribute wie Farbe und Texturen lassen sich weitere Sachverhalte einer Software in Code-Cities darstellen [Lim+19]. Hier kann man

insbesondere auch gut die Metapher einer Stadt bedienen, indem man beispielsweise die Anzahl von Methoden einer Klasse, die als Gebäude dargestellt sind, durch Stockwerke ausdrückt. Relationen zwischen den in einer Code-City dargestellten Softwareelementen werden zumeist in Form von Kanten dargestellt [Kos03a]. Werden diese aber lediglich auf dem direkten Weg der durch die Kanten verbundenen Elemente gezeichnet, ergibt sich bei Systemen mit vielen Relationen wegen der vielen Überschneidungen ein nicht mehr lesbares Bild. Eine übersichtlichere Darstellung, die auch die der Software zugrundeliegende Hierarchie gut berücksichtigt, bieten die von Holten vorgeschlagenen hierarchisch gebündelten Kanten, die mittels B-Splines beschrieben werden [Hol06; Hol09]. Die Kontrollpunkte der B-Splines werden hierbei so gewählt, dass sie die verschiedenen Ebenen der Hierarchie reflektieren, in denen die verbundenen Elemente verortet sind. Alle Kanten, die über dieselbe Hierarchieebene verlaufen, teilen sich dabei den gleichen Kontrollpunkt, was zur Bündelung wie durch einen Kabelbinder führt.

Code-Cities sind längst auch in der Praxis angekommen und werden mittlerweile selbst von kommerziellen Werkzeugen zur Visualisierung der Änderungshistorie und Code-Qualität verwendet. Beispielsweise haben die Firmen *Hello2omorrow* und *Serene* solche Darstellungsformen implementiert. *Serene* ist hier ein besonders erwähnenswertes Beispiel, weil die Firma direkt aus der Forschung am Hasso-Plattner-Instituts (HPI) hervorgegangen ist [Boh10]. Auch für die weitverbreitete Plattform für statische Analyse zur Bewertung der inneren Softwarequalität *SonarQube* gibt es ein Plugin namens *SoftVis3D* für *Code-Cities* auf Basis von *Treemaps* und *EvoStreets*. Leider liegen nach unserer Kenntnis noch keine wissenschaftlichen Studien vor, wie und mit welchen Auswirkungen diese Visualisierungen tatsächlich in der Praxis verwendet werden. Teilweise scheinen diese Visualisierungen durch ihre Ästhetik auch aus Gründen der Werbewirksamkeit eingesetzt zu werden. Wenn sie in der Entwicklung tatsächlich eingesetzt werden, dann wohl meist zur Kommunikation auf der Management-Ebene (bei *Serene* ist das der so genannte *Digital Boardroom*). Aus unserer Sicht ist damit das Potential von Software-Visualisierung in kooperativen Prozessen aber noch nicht völlig ausgeschöpft. Wir denken, dass sie auch tief in den Entwicklerteams verankert werden sollte, damit diese ein gemeinsames Bild ihrer Software haben, um eine bessere Zusammenarbeit zu ermöglichen.

3.2 Code-Cities in VR/AR

Dreidimensionale Code-Cities wurden und werden zumeist auf zweidimensionalen Displays in Pseudo-3D dargestellt. Mit Hilfe stereoskopischer Displays, wie sie in *Head-Mounted Displays (HMD)* für VR verbaut sind, lassen sie sich aber auch für das menschliche Auge authentisch dreidimensional wahrnehmen. Die Darstellung von Code-Cities in VR ist dabei eine keineswegs neue Idee. Umgesetzt wurde sie bereits im Jahre 2000 [KM00; Cha+02]. An die Verwendung von VR war damals wie heute auch die Hoffnung geknüpft, dass sich die Vorteile von VR, die außerhalb der Softwaretechnik bereits beobachtet werden konnten [Cha+98], auf dieses Themenfeld übertragen lassen. Weiter befördert wurde die Software-Visualisierung mit VR auch durch Fortschritte der Technologien und eine größere Verfügbarkeit der Geräte. Seit Erscheinen des

Übersichtsartikels von Knight und Munro im Jahre 2000 [KM00] wurden Code-Cities in Pseudo-3D und VR für eine ganze Reihe verschiedener Aspekte von Software eingesetzt. Dazu gehören der statische Aufbau [Cap+17; Mal+01; PBG03; Pan+07b; FKH15a; Kha+17; Mer+17b; SB17] genauso wie das dynamische Verhalten [Wal+13; Fit15; FRH15; Oga+17; FRW17; Mer+19]. Die für die Softwaretechnik spezifisch gesammelten Erfahrungen werden ergänzt durch den Wissensbestand, der außerhalb der Softwaretechnik und auch außerhalb der Informatik mit Pseudo-3D und VR erarbeitet wurde [Bow+99; RCB07; RSM92]. Studien mit HMDs haben beispielsweise einen positiven Effekt für die Orientierungsfähigkeit in dreidimensionalen Umgebungen gezeigt [Cha+98]. Es gibt jedoch auch Studien, bei denen ein positiver Effekt nicht nachgewiesen werden konnte. Sousa Santos et al. etwa haben bei einem Experiment zur Navigation beobachtet, dass die Teilnehmer zwar sehr positiv von VR eingenommen waren, aber in einer Desktop-Umgebung mit 2D-Display, Maus und Tastatur tatsächlich besser navigieren konnten. Ruddle et al. haben die Navigation in computer-simulierten Welten mit HMDs und Desktop-Umgebungen verglichen [RPJ99; RP04]. In ihrem ersten Experiment, in dem die Teilnehmer durch verschiedene Räume in einem virtuellen Gebäude navigieren mussten, konnten sie eine höhere Geschwindigkeit bei den Teilnehmern mit HMD beobachten [RPJ99]. In einem zweiten Experiment haben sie propriozeptives Feedback und seinen Einfluss auf die Navigation in einem virtuellen Labyrinth untersucht [RP04]. Dabei konnte für die Träger der HMDs kein signifikanter Vorteil festgestellt werden. Diese Untersuchungen haben nur eine geringe Relevanz für unsere eigene geplante Forschung, da wir nicht beabsichtigen, dass sich Entwickler durch maßstabsgetreu große Städte bewegen sollen, so dass man eigentlich eine Landkarte bräuchte, um sich zu orientieren. Stattdessen schwebt uns vor, dass sich alle Entwicklerinnen in einem virtuellen Raum um einen Tisch versammeln, auf der die Code-City vor ihnen ausgebreitet ist. Dies ermöglicht es, dass sich die Entwicklerinnen jederzeit sehen können und sich allenfalls im selben virtuellen Raum um einen Tisch herum bewegen müssen. Der Grund dafür, warum wir diese sich teilweise widersprechenden Studien in Domänen außerhalb der Informatik hier anführen, ist es, dass hiermit erkenntlich wird, dass es keineswegs klar ist, ob VR stets die in sie gesetzten Hoffnungen sicher erfüllt und dass hier noch immer Grundlagenforschung notwendig ist. Dies berücksichtigen wir insofern in unserem beantragen Projekt, dass wir verschiedene Technologien integrieren und miteinander vergleichen werden. Neben VR/AR werden wir auch stets „klassische“ Umgebungen mit Desktop oder Touchpads in unseren Entwicklungen und Untersuchungen berücksichtigen und gegeneinander abwägen.

Eine interessante Beobachtung am Rande zur aktuellen Forschung im Bereich der Visualisierung mit Hilfe der Stadtmetapher ist es, dass gerade in Deutschland eine hohe Anzahl von Forschungsgruppen diese nutzen: Die Gruppe um Kollege Lewerentz an der Universität Cotbus, wo die sogenannten *EvoStreets* entwickelt wurden [SL10; Ste13], die Gruppe von Kollege Döllner am Hasso-Plattner Institut in Potsdam, die verschiedene Verbesserungen der *Code-Cities* vorgeschlagen und evaluiert haben [Lim+19], die Gruppe von Kollege Hasselbring an der Christian-Albrecht Universität zu Kiel, die in Deutschland als Pioniere der *Code-Cities* für die Darstellung dynamischer Information [Wal+13; Fit+13; FSH14; Fit+15; FKH15b; Fit15; FKH17; FRH15] und in virtueller Realität gelten [FKH15a;

ZKH19] sowie das Visualisierungsinstitut der Universität Stuttgart, die verschiedene Aspekte der Interaktion mit *Code-Cities* in VR untersuchen [Mer+17a; MBN18; Mer+19; Mer+17b]. Dadurch können sich leicht Möglichkeiten zur Vernetzung mit anderen Forschungsgruppen innerhalb Deutschlands auf diesem Gebiet ergeben.

3.3 Kooperative Visualisierung

Bereits in den 90er Jahren stellten Wood et al. [WWB97] eine Systemarchitektur vor, die als Basis für beliebige (Datenfluss-)Visualisierungen genutzt werden kann. Mit Hilfe dieser Architektur, so die Autoren, könnten bestehende Visualisierungen leicht um einen kollaborativen Aspekt erweitert werden (im Fokus standen Visualisierungen zur Darstellung wissenschaftlicher Daten). Als besondere Neuerung heben die Autoren hervor, dass es mit dem vorgestellten Verfahren nicht länger notwendig sei, dass sich die Betrachter (z.B. Mitglieder einer Arbeitsgruppe) physisch an einem Ort zusammen finden müssen. Stattdessen sei es möglich, nach Belieben unabhängig (und verteilt) voneinander zu arbeiten. Neben der Beschreibung der Systemarchitektur implementierten die Autoren das Verfahren als Erweiterung für den `IRIS Explorer` [Fou95] und demonstrierten das Vorgehen in einem fiktiven Szenario.

Ein früher Versuch, kollaborative Visualisierung im Bereich des Programmverstehens umzusetzen, findet sich in der 2005 veröffentlichten Arbeit von Kot et al. [Kot+05]. Die Autoren unternahmen den Versuch, auf Basis der Computerspiele-Engine *Quake 3* eine Anwendung zur Visualisierung von Programmcode in einer dreidimensionalen Welt zu erstellen. Die grundlegende Idee besteht darin, Quelltextdateien als interaktive *Entitäten* in einer statischen 3D-Welt darzustellen. Diese können dann von Nutzern (Spielern) eingesehen, bewegt und (z.B. in logischen Gruppen) arrangiert werden. Da *Quake 3* selbst bereits netzwerk- und somit mehrbenutzerfähig ist, lässt sich die Visualisierung ohne weiteren Aufwand kooperativ nutzen. Die Autoren zeigten durch ihre Umsetzung, dass es grundsätzlich möglich und vorteilhaft ist, Computerspiele-Engines als Basis für Visualisierungsanwendungen zu nutzen. Vorteilhaft ist insbesondere die Wiederverwendung vieler der von der Engine bereitgestellten Funktionalitäten. Auch wir werden in unserem Projekt deshalb auf einer existierenden Spiele-Engine aufbauen. Eine formale Usability-Studie wurde von den Autoren nicht durchgeführt; sie beließen es bei Ad-Hoc-Beobachtungen von Nutzern, die teilweise Verbesserungsmöglichkeiten existierender Feature aufdeckten. Eine berichtenswerte Beobachtung, die die Autoren machten, war jedoch, dass die Nutzer sich die virtuelle Welt schnell zu eigen machten und in einer Weise zur Zusammenarbeit nutzten, die sich zwar auf natürliche Weise aus dem Aufbau der virtuellen Welt ergab, jedoch von den Autoren weder so vorhergesehen noch durch besondere Features explizit unterstützt wurde: Die teilnehmenden Entwickler begannen von sich aus damit, Blöcke, die hier Dateien darstellten, zu anderen Entwicklern zu tragen, um sie ihnen zu zeigen.

Jung et al. [JDP20] konzipierten und implementierten eine Anwendung zur Visualisierung von Softwaresystemen in VR. Als Metapher für die Darstellung statischer Strukturen (wie beispielsweise Klassen und Pakete) wurde das Konzept der Code-City gewählt. Neben der Darstellung einzelner Gebäude (für die zu visualisierenden Elemente

einer Software) können Gebäude auch in Distrikte zusammengefasst und diese Distrikte visuell hervorgehoben werden. Als unmittelbarer Vorteil der Distrikte geben die Autoren an, dass die Hierarchie der visualisierten Software (Klassen in Pakete etc.) erhalten bleibt, da Distrikte rekursiv ineinander verschachtelt werden können. Zusätzlich unterstützt die Anwendung die Darstellung von Funktionsaufrufen (Traces). Diese werden mittels Kanten (Kreisbögen) hervorgehoben, deren Darstellung – anders als bei den Gebäuden und Distrikten – dynamisch ist, d.h. Nutzern der Anwendung wird ermöglicht, den Verlauf einer Aufrufsequenz von Funktionen live mitzuverfolgen. Die entwickelte Visualisierung setzt sich somit aus einem statischen Teil (die Stadt) und einem dynamischen Teil (animierte Aufrufkanten) zusammen.

Anders als bisherige Ansätze in VR bietet die Anwendung von Jung et al. zudem die Möglichkeit an, dass die Visualisierungen von mehreren Personen gleichzeitig genutzt werden kann (Kollaboration). In einem kontrollierten Experiment wurde der Forschungsfrage nachgegangen, ob und wie VR einen Mehrwert bei der Software-Visualisierung bieten kann. Gegenstand des Experiments war ein Szenario, in dem ein *Coach* einem *Newbie* die Architektur eines für den *Newbie* unbekannten Systems erläutert (in dem Szenario gab es immer genau einen *Coach* und einen *Newbie*). Die Teilnehmer des Experiments (die *Newbies*) wurden in zwei Gruppen aufgeteilt. Die eine Gruppe (VR-Gruppe) spielte das Szenario in VR durch, die andere Gruppe (Kontrollgruppe) an einem regulären Bildschirm. Während die VR-Gruppe das durch die Anwendung bereitgestellte Kollaborationsfeature nutzte (d.h. *Coach* und *Newbie* bewegten sich gemeinsam, aber jeweils an einem eigenen Endgerät befindlich durch die Stadt), nutzte die Kontrollgruppe das Programm *Jaeger*, wobei der Bildschirm des *Coaches* über ein Audio-/Videokonferenzsystem mit dem *Newbie* geteilt wurde (auf Wunsch seitens des *Newbies* passte der *Coach* den sichtbaren Ausschnitt der Visualisierung an). Das entspricht in etwa dem heutigen Umgang mit Videokonferenzsystemen bei der Kollaboration. In beiden Fällen konnte der *Coach* die Namen der Objekte der Stadt (die Namen der Klassen) für den *Newbie* einblenden. Darüber hinaus gab es drei Beispiele für Funktionsaufrufsequenzen (dynamische Visualisierung), die zum Verständnis der *Newbies* beitragen sollten.

Im Anschluss an das Szenario füllten die Teilnehmer zwei Fragebögen aus. Der erste Fragebogen enthielt dabei Verständnisfragen zur untersuchten Anwendung, um zu überprüfen, ob die durch den *Coach* genannten Informationen reproduziert werden können. Der zweite Fragebogen hingegen setzte sich aus Fragen zusammen, mit Hilfe derer die Teilnehmer die Visualisierung (VR bzw. regulärer Desktop) bewerten konnten. Im Ergebnis zeigte sich, dass im Mittel über alle Fragen des ersten Fragebogens hinweg beide Gruppen ungefähr gleich gut abgeschnitten haben (Median: 58 % Korrektheit der VR-Gruppe und 56 % Korrektheit der Kontrollgruppe). Es zeichneten sich jedoch bei zwei Fragen stärkere Unterschiede ab (einmal war die VR-Gruppe deutlich besser, einmal die Kontrollgruppe). Den Ergebnissen des zweiten Fragebogens nach schienen die Teilnehmer der Kontrollgruppe den Erläuterungen des *Coach* etwas besser folgen zu können. Zugleich hatten die Teilnehmer der Kontrollgruppe im Schnitt weniger das Gefühl, etwas Neues gelernt zu haben. In Bezug auf die Fokussiertheit während des Szenarios zeichneten sich keine wesentlichen Unterschiede ab.

Im Anschluss an das Experiment führte die Kontrollgruppe das beschriebene Szenario

ebenfalls in VR durch und beide Gruppen (VR-Gruppe und Kontrollgruppe) füllten einen dritten Fragebogen mit Fragen speziell zu der VR-Umgebung aus. Über diesen Fragebogen sollte untersucht werden, wie hilfreich die einzelnen Features der VR-Umgebung während des Szenarios waren. Als besonders hilfreich haben sich dabei zwei Features herausgestellt: (i) die Möglichkeit, den virtuellen Laserpointer des *Coach* (mit dessen Hilfe dieser einzelne Objekte der Stadt auswählen und so deren Namen für den *Newbie* einblenden konnte) zu sehen und (ii) die Möglichkeit, sich in der Stadt durch Teleportation bewegen zu können. Während die Objektnamen ein wichtiger Hinweis für die *Newbies* waren, wurde der Umstand, dass die Namen an die Gebäudewand projiziert wurden, seitens der Teilnehmer im Mittel als weniger hilfreich bewertet.

Interessant sind die Ergebnisse von Jung et al. im Hinblick auf die bereits zuvor durchgeführten Studien von Isenberg et al. [Ise+10] und Anslow et al. [Ans+13; Ans14]. In diesen Studien wurde kollaborative (Software-)Visualisierung in gemeinsamen Umgebungen (*co-located Environments*) untersucht. Zentraler Gegenstand der Untersuchungen ist ein als Tischplatte verbautes Multitouch-Display, welches von mehreren Personen gleichzeitig verwendet werden kann. Auf dem Display konnten mehrere Visualisierungsformen gleichzeitig dargestellt und von den Nutzern interaktiv verwendet werden. Es wurden Experimente durchgeführt, in denen immer Paare von Teilnehmern gemeinsam Aufgaben absolvieren mussten. Als eines der wesentlichen Ergebnisse der Studien kam hervor, dass kollaborative (Software-)Visualisierung *wirklich gemeinsam genutzt* werden sollte – d.h. im Gegensatz dazu, dass einzelne Individuen parallel mit der Visualisierung aber jeder für sich arbeiten. Ein wichtiger Bestandteil der *gemeinsamen Nutzung* besteht laut Autoren darin, Beobachtungen und Ergebnisse mit anderen (verbal) teilen zu können. Diese Ergebnisse fügen sich gut in die Ergebnissen von Jung et al. ein. Im Experiment von Jung et al. wurde zum einen die Möglichkeit, den *Coach* als Avatar in der VR sehen zu können (was eine gemeinsame Nutzung der VR simuliert), im Mittel eher positiv bewertet. Zum anderen konnten *Coach* und *Newbie* während des gesamten Szenarios über ein Telefonkonferenzsystem miteinander kommunizieren. Eine Reihe ähnlicher Arbeiten finden sich im Kontext von *ExploreViz*.

ExploreViz ist ein an der Christian-Albrechts-Universität zu Kiel entwickeltes Framework zur Visualisierung statischer und dynamischer Informationen mit Hilfe der City-Metapher. Dieses zunächst nur in einem Brower für eine Desktop-Umgebung konzipierte Framework wurde von Zirkelbach und Kollegen zu einer VR-Visualisierung ausgeweitet und durch Nutzerstudien evaluiert [ZKH19]. Hierzu haben zunächst zwölf Probanden einzeln die VR-Variante von *ExploreViz* verwendet und anschließend einen Usability-Fragebogen zu ihren subjektiven Eindrücken ausgefüllt. Neben spezifischen Verbesserungsmöglichkeiten einzelner von *ExploreViz* zur Verfügung gestellter Fähigkeiten ergab sich eine insgesamt hohe Akzeptanz bei den Nutzern für VR. In einer darauf folgenden Nutzerstudie wurde der Aspekt der Kooperation innerhalb der virtuellen Welt näher untersucht. Der Versuchsaufbau glich im Wesentlichen der ersten Nutzerstudie, hier jedoch haben elf Paare von Probanden *ExploreViz* gemeinsam in der virtuellen Realität die dargestellten Code-Cities exploriert. Dabei konnten die Teilnehmer, die sich in verschiedenen physischen Räumen befanden, im gemeinsamen virtuellen Raum miteinander sprechen und sich auch zumindest angedeutet sehen. Hierzu wurden das

Head-Set sowie die Controller des jeweils anderen Teilnehmers synchron zu den Kopf- und Handbewegungen in der virtuellen Welt dargestellt. Der anschließend ausgefüllte Fragebogen ergab auch hier wieder eine hohe Akzeptanz bei den Teilnehmern für VR und auch die Unterstützung für kooperatives Explorieren. Ein interessanter Aspekt des qualitativen Feedbacks der Teilnehmer betrifft die Art, wie der andere Teilnehmer in der virtuellen Welt erschien. Die Teilnehmer schätzten es, dass der andere damit sichtbar wurde. Allerdings hätten sie sich lieber eine realistischere, menschlichere Darstellung gewünscht, statt einfach nur die Brille und die Controller einzublenden. Dieses Feedback entspricht der Beobachtung, die schon lange in der Robotics-Community gemacht wurde, dass humanoide Roboter mit einem menschlichen Antlitz viel besser von Menschen akzeptiert werden als nicht-humanoide und gesichtslose Roboter [PR15]. Die Nutzerstudie hat eine prinzipielle Nutzerakzeptanz für VR ergeben und auch gezeigt, dass gerade die Möglichkeit von VR zur Überbrückung räumlicher Distanzen beim Betrachten eines gemeinsamen Bildes einer Software sehr wertgeschätzt wird. Die Studie war jedoch als reine Usability-Studie angelegt, bei der Teilnehmer keine konkreteren Aufgaben meistern mussten, deren Erledigung objektiv quantifiziert wurde. Vielmehr wurde nur das subjektive Feedback der Teilnehmer eingeholt und der Ansatz auch weder qualitativ noch quantitativ gegen alternative Lösungen verglichen.

Es gibt eine weitere Reihe von Arbeiten im Bereich Softwarevisualisierung und Kollaboration (bzw. kollaboratives Verstehen von Software und ihrer Entwicklung) [Mal+01; DL08b; DL08a; DL10; Pan+07a; FPV17] – im weiteren Sinne auch von Stroulia u. a. [Str+13]. Zusammenfassend lässt sich zu diesen Veröffentlichungen sagen, dass die Autoren die Nützlichkeit ihrer entwickelten Visualisierungen hinsichtlich Kollaboration und kollaborativem Programmverstehen nur postulieren. Mit Ausnahme von Stroulia et al. [Str+13] werden diese Annahmen jedoch nicht mit geeigneten Nutzerstudien untermauert. Ein wesentlicher Unterschied zu den von Kot et al. [Kot+05], Jung et al. [JDP20] und Zirkelbach et al. (*ExploreViz*) entwickelten Lösungen besteht zudem darin, dass die Nutzer in den zuletzt genannten Studien [DL08b; DL08a; DL10; Pan+07a; Str+13; FPV17] (mit Ausnahme von [Mal+01]) keine visuelle Repräsentation (z.B. als Avatar) innerhalb der Anwendung haben, sodass die Kommunikation zwischen den Nutzern rein verbal bzw. textuell (z.B. über einen Chat) stattfindet – sofern keine zusätzlichen Videokonferenzsysteme herangezogen werden oder sich alle Nutzer am gleich Ort befinden. Bei der Arbeit von Maletic u. a. [Mal+01] handelt es sich um eine Visualisierungsanwendung (mit dem Namen *IMSOVision*) zur Darstellung objektorientierter Software in virtuellen Welten. Die Anwendung wurde primär für *CAVE* (*Cave Automatic Virtual Environment*) [CSD93; Lei+99; Mel+15] entwickelt, einem (mit mehreren Personen) betretbaren Würfel mit bis zu sechs Projektionsflächen, über die eine dreidimensionale virtuelle Realität innerhalb des Würfels abgebildet werden kann. Wenngleich Kollaboration in dieser Arbeit [Mal+01] kein primärer Untersuchungsgegenstand war, betonen die Autoren, dass kollaboratives Arbeiten grundsätzlich damit möglich sei, da einerseits alle im Würfel befindlichen Personen miteinander kommunizieren und andererseits mehrere *CAVE*-Installationen sogar *remote* miteinander vernetzt werden können.

Eigene Vorarbeiten

Unsere erste eigene Unternehmung, Softwarevisualisierung mit Hilfe der Stadt-Metapher in VR umzusetzen, findet sich in der gemeinsamen Bachelorarbeit von Ganser und Rüdel [GR18]. Ziel der Arbeit war die Umsetzung einer Visualisierung, mit derer i) beliebige Softwaremetriken als EvoStreets und ii) Beziehungen zwischen Elementen (z.B. Abhängigkeiten zwischen Komponenten) mit Hilfe gebündelter Kanten dargestellt werden können. Ein besonderer Schwerpunkt der Arbeit lag zudem auf der Integration des *HTC Vive Virtual Reality Systems*, sodass Nutzer der Anwendung die generierten virtuellen Welten über HMDs erfahren und mittels Hand-Controllern mit diesen Welten interagieren (z.B. Elemente auswählen) können. Die Anwendung, *SCOOP* genannt, wurde als Plug-in der Computerspiele-Engine *Unreal Engine 4* umgesetzt. Das Eingabeformat von *SCOOP* basiert auf dem Standardformat für den Austausch von Graphen *GXL*[5]. Über dieses Format lassen sich beliebige Graphen mit attributierten Knoten und Kanten beschreiben. Die Knoten des Eingabegraphen werden als dreidimensionale Blöcke dargestellt, wobei die im Graph hinterlegten Knotenattribute beliebig auf die geometrischen Eigenschaften *Höhe*, *Breite* und *Tiefe* sowie einem Farbgradienten (als Textur der Blöcke) abgebildet werden können. Die Anordnung der Blöcke wird von *SCOOP* vollständig automatisch errechnet. Auf Basis des generierten Layouts werden die Kanten des Eingabegraphen hierarchisch gebündelt und können ihrerseits über eine farbliche Kodierung weitere Metriken darstellen. Aufgrund der Flexibilität des *GXL*-Formats können in *SCOOP* beliebige Elemente (Klassen, Methoden etc.) und ihre Relationen (Aufrufe, Abhängigkeiten etc.) visualisiert werden. Neben industriellen Anwendungen unterstützen auch viele in unserer Arbeitsgruppe entwickelten Analyseprogramme wie *iClones* [GK09] und *LibVCS4j* [Ste20] *GXL* als Ausgabeformat, so dass deren Ergebnisse in *SCOOP* direkt visualisiert werden können.

Auf Basis der von Ganser und Rüdel [GR18] entwickelten Anwendung wurde in einem kontrollierten Experiment [RGK18] untersucht, ob und inwieweit VR einen Einfluss auf die Orientierung von Nutzern in *EvoStreets* hat. Konkret wurde untersucht, ob i) HMDs dazu beitragen, dass Nutzer effektiver und effizienter in EvoStreets navigieren können, ii) Nutzer mit Vorerfahrung in Computerspielen (insbesondere in solchen, in denen mit einer Tastatur navigiert werden muss) die gestellten Aufgaben effizienter lösen können und iii) ob bei gleichbleibender Struktur das Wechseln der abgebildeten Metriken einen Effekt auf die Effektivität und Effizienz der Navigation hat, d.h. ob Nutzer, die die EvoStreets für eine Software bereits kennen, gleichermaßen effektiv und effizient navigieren, wenn sich ausschließlich die geometrischen Eigenschaften der Blöcke, nicht aber deren Struktur (Anordnung) ändert. Gegenstand des Experiments waren Aufgaben, in denen die 20 Teilnehmer *Zielsuchaufgaben* erfüllen mussten. Im Ergebnis zeigte sich, dass die Primärhypothese des Experiments (HMDs haben einen positiven Effekt auf die Orientierung in *EvoStreets*) abgelehnt werden musste. Die Annahme, dass Nutzer mit Vorerfahrungen in Computerspielen die gestellten Aufgaben effizienter lösen, konnte bestätigt werden – ein positiver Effekt auf die Effektivität konnte jedoch nicht festgestellt werden. Lerneffekte, die dazu beitragen würden, dass in bekannten

[5] http://www.gupro.de/GXL/

EvoStreets (mit anderer Metrikabbildung) effektiver und effizienter navigiert werden kann, konnten nicht beobachtet werden. Ein gegenteiliger Effekt (also eine Verschlechterung der Navigation) konnte ebenfalls nicht beobachtet werden – wir nehmen somit an, dass die Navigation bzgl. Effektivität und Effizienz erhalten bleibt. Hier muss jedoch angemerkt werden, dass die Code-Cities maßstabsgetreu zu einer echten Stadt dargestellt wurden, so dass sich die Probanden wirklich innerhalb einer Stadt bewegten (wobei sie sich in allen drei Dimensionen fortbewegen konnten).

In einem weiteren Experiment [SKR19a] haben wir *EvoStreets* in verschiedenen *Umgebungen* untersucht: VR (HMD und Hand-Controller), 2.5D (Pseudo-3D am Bildschirm mit Maus und Tastatur) und 2D (orthographische Projektion am Bildschirm mit Maus und Tastatur). Ziel dieses Experiments war es, Anhaltspunkte zu finden, ob bestimmte Umgebungen sich besser für die Analyse von Software eignen als andere. Dazu wurden im Rahmen des Experiments drei Aufgaben entwickelt, in denen verschiedene Open-Source-Systeme hinsichtlich ihrer kopierter Quelltexte (Klone/Duplikate) untersucht werden sollten. Wir haben uns in unserem Experiment für die Analyse von Klonen entschieden, da Klone Gegenstand aktueller Forschung sind, leicht von Probanden mit allgemeinen Kenntnissen im Bereich der Software-Entwicklung verstanden werden können und sich gut eignen, um in *EvoStreets* mit Kanten dargestellt zu werden (eine Kante stellt dar, dass die beiden verbundenen Elemente gemeinsame Klone haben). Während des Experiments mussten die 34 Teilnehmer jede gestellte Aufgabe in jeweils einer der drei Umgebungen lösen, d.h. jeder Teilnehmer hat jede Umgebung genau einmal verwendet. Im Vorfeld des Experiment wurde *SCOOP* um zwei Funktionen erweitert: Zum einen wurde die Kantenbündelung verbessert, indem das bis dahin verwendete Spline-Modell von kubisch hermiteschen Splines auf B-Splines umgestellt wurde, was eine hierarchische Kantenbündelung nach Holten [Hol06; Hol09] ermöglicht (die Umsetzung wurde mit der von uns entwickelten Bibliothek *TinySpline* [SK21] realisiert). Zum anderen wurde die Möglichkeit geschaffen, Subsysteme einer Software visuell hervorzuheben. Beide Erweiterungen haben in unserem Experiment Anwendung gefunden. So musste beispielsweise in einer der Aufgaben ermittelt werden, welche beiden Subsysteme der visualisierten Software am stärksten miteinander verklont sind. Obwohl wir keine statistisch signifikanten Unterschiede hinsichtlich der Zeiten, die zur Lösung der Aufgaben aufgewendet wurden, und der Korrektheit der abgegebenen Antworten zwischen den Umgebungen feststellen konnten, konnten wir beobachten, dass die Teilnehmer in der VR-Umgebung dazu neigten, im Vergleich zu der 2.5D-Umgebung einen geringeren Bewegungsradius einzunehmen. Diese Beobachtung haben wir in einer weiteren Studie [SKR19b] näher untersucht.

In der nachfolgenden Studie [SKR19b] haben wir die in unserem zuvor durchgeführten Experiment [SKR19a] erhobenen Positionsdaten ausgewertet und zu Bewegungstrajektorien zusammengefasst. Auf Basis dieser Trajektorien wurde untersucht, ob i) sich die Bewegungen der Probanden in VR und 2.5D hinsichtlich der drei Metriken *Länge*, *Durchschnittsgeschwindigkeit* und *Volumen* voneinander unterscheiden und ii) ob es eine Korrelation zwischen diesen Metriken und den aufgewendeten Zeiten und der Korrektheit der Antworten gibt. Wenngleich keine klar signifikanten Korrelationen ermittelt werden konnten, haben sich unsere früheren Vermutungen [SKR19a] in einigen Teilen

bestätigt: Das Volumen und die Durchschnittsgeschwindigkeit war in VR in zwei der drei Aufgaben signifikant kleiner, die Länge der Bewegungstrajektorien war in einer der drei Aufgaben signifikant kürzer.

4 Konzeption und Implementierung

4.1 User Interface

4.1.1 UI-Framework

Im Verlauf des Bachelorprojekts ist aufgefallen, dass es ein Team geben sollte, welches die an verschiedenen Stellen entstandene UI[6] vereinheitlichen sollte. Ebenso sollte dieses Team eine Art Framework als Abstraktion entwickeln, mit dem sich UI-Komponenten leicht verwenden lassen können, ohne sich um plattformspezifische Details kümmern zu müssen.

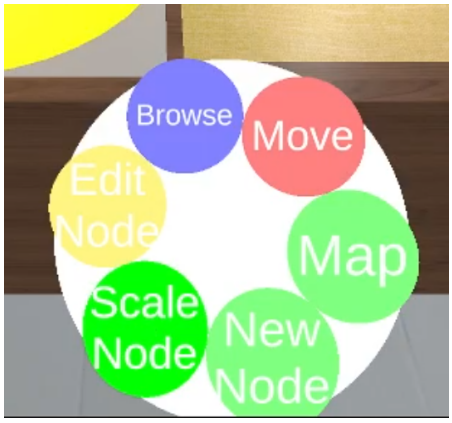

Abbildung 1: Das vorherige *Player Menu*.

Dementsprechend wurde ein abstraktes UI-Framework erstellt, welches die einfache Erstellung plattformabhängiger Komponenten ermöglicht — d. h. z. B. die Erstellung einer „Menü"-Komponente, welche ein plattformunabhängiges Interface zur Erstellung und Verwaltung besitzt, intern aber verschiede Ausprägungen je nach aktiver Plattform (wie Desktop, VR, HoloLens...) verwendet. So können bestehende Komponenten leicht wiederverwendet werden und die spätere Unterstützung weiterer Plattformen von existierenden Komponenten wird deutlich vereinfacht, da nur noch eine weitere Ausprägung des bestehenden Interfaces entwickelt werden muss.

Hierfür wurden die folgenden Komponenten entwickelt (jeweils das plattformunabhängige Interface sowie die Desktop-Ausprägung):

- `Menu`: Oft muss man Nutzern eine Auswahl an Dingen anbieten, wobei jede Wahl einen bestimmten Effekt haben soll. Dazu wurde ein Menü mit mehreren Buttons, welche jeweils beliebige Aktionen ausführen können (siehe z. B. Sektion 4.2), entwickelt. Hierfür gibt es auch noch eine Sonderform `SelectionMenu`, in welcher speziell ein Modus aus einer Liste von Modi ausgewählt werden kann — dies wurde z. B. für die Auswahl des Interaktionsmodus mit den Code-Cities verwendet, welcher in Abbildung 2 betrachtet werden kann. Man vergleiche hierzu auch das vorherige Menü in Abbildung 1.

- `PropertyDialog`: Abgesehen von dem Treffen einer Auswahl aus einer Liste gibt es auch diverse Szenarien, in denen Nutzer Eingaben verschiedener Art machen

[6] Z.B. das in Abbildung 1 auffindbare Menü für die Interaktionsmodi mit SEE.

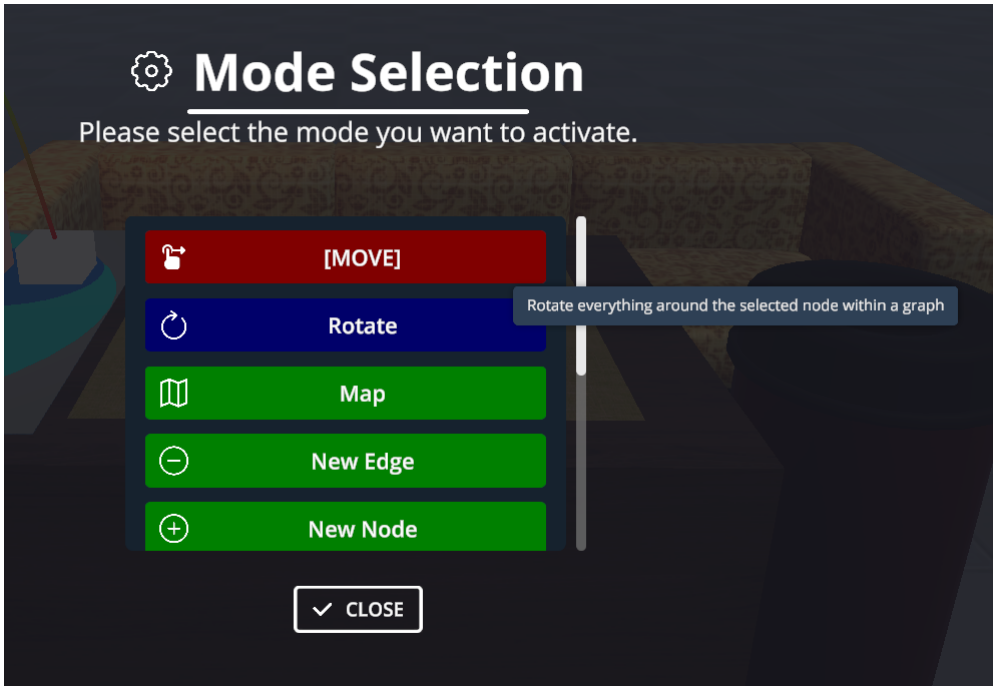

Abbildung 2: Das neue *Player Menu* mit offenem Tooltip.

Abbildung 3: Ein `StateIndicator` in der Ecke des Bildschirms.

müssen, z. B. um Einstellungen zu treffen. Dementsprechend handelt es sich hier um einen Konfigurationsdialog, in dem Werte oder Eigenschaften von Objekten festgelegt werden können. Ein Beispiel hierfür findet sich in Abbildung 11.

- `StateIndicator`: Um Nutzern zu zeigen, welcher Interaktionsmodus mit den Code-Cities gerade aktiv ist, wurde ein Indikator entwickelt, welcher ähnlich wie in zustandsbasierten Editoren wie *Vim*[7] unter Verwendung einer bestimmten Farbe und eines bestimmten Textes in einer Ecke des Bildschirms Informationen anzeigen kann. Wie so etwas aussieht ist in Abbildung 3 zu sehen — dort wird unter Verwendung der Erweiterung `ActionStateIndicator` der aktuelle Interaktionszustand angezeigt.

- `Tooltip`: Um zusätzliche Informationen zu UI-Elementen (etwa Menüpunkten) einzublenden, eignen sich Tooltips. Ein Tooltip ist hier ein Rechteck, in dem beliebiger Text angezeigt werden kann. Das Rechteck kann mit einer Verzögerung

[7] `https://www.vim.org/` (letzter Abruf: 30.5.2021)

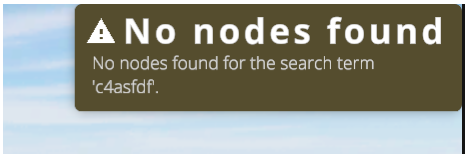

Abbildung 4: Eine Benachrichtigung vom Typ *Warnung*.

ein- und ausgeblendet werden und folgt dem Mauszeiger, sodass es sich dafür eignet, beim Schweben über Elemente Erklärungen einzublenden. Ein Tooltip ist z. B. in Abbildung 2 über dem Menüpunkt *Rotate* erschienen.

- `Notification`: Ein weiteres Problem in SEE war, dass wichtige Benachrichtigungen für den Nutzer oft in Unitys Log-Nachrichten untergingen[8]. Diese Komponente soll eine Alternative dazu bieten, um dem Nutzer im Spiel selbst etwas mitzuteilen. Benachrichtigungen können in verschiedenen Ausführungen über die statische Methode `ShowNotification.Info(titel, text)` genauso leicht erzeugt werden wie mit Unitys `Debug.Log(text)`-Funktion — wie in dessen Familie an Log-Funktionen gibt es auch für die Benachrichtigungen die Varianten *Info*, *Warnung* (siehe hierfür Abbildung 4) und *Fehler*.

- `CodeWindow`: Ein wichtiger Teil in SEE, der noch gefehlt hat, war das Anzeigen des Quellcodes des analysierten Projekts. Da die Umsetzung hiervon umfangreicher war als bei anderen Komponenten gibt es dafür einen eigenen Abschnitt in Sektion 4.1.2.

Alle diese Komponenten erben von `PlatformDependentComponent`, eine Klasse, welche für Unitys `Start`- und `Update`-Methoden jeweils eine Variante pro verfügbarer Plattform anbietet (z. B. `Start`*`Desktop`* oder `Update`*`Hololens`*). Wenn dann die eigentliche `Start`- bzw. `Update`-Methode der Komponente von Unity aufgerufen wird, sorgt der Code in `PlatformDependentComponent` dafür, dass die plattformspezifische Variante der *aktuellen* Plattform aufgerufen wird. Eine `PlatformDependentComponent` wie bspw. der `StateIndicator` kann nun die jeweiligen plattformspezifischen Methoden überschreiben, um für die Plattformen UI-Code festzulegen und kann dabei nach außen hin plattformunabhängige Felder und Methoden bereitstellen, z. B. eine Methode zum Anzeigen oder ein Feld, mit dem die Farbe festgelegt werden kann. In Abbildung 5 ist ein vereinfachtes UML-Diagramm des UI-Codes von SEE zu sehen.

Die Desktop-UI, welche auch in den meisten Abbildungen verwendet wird, wurde dabei mit dem *Modern UI Pack*[9] entwickelt. Für Augmented Reality bzw. die HoloLens sollte das *Mixed Reality Toolkit* (siehe Sektion 4.4) verwendet werden und für Virtual Reality eine Kombination aus dem *VR UIKit*[10] und *CurvedUI*[11].

[8] Hinzu kommt noch, dass der Unity-Log in der ausgelieferten Software im Gegensatz zum Editor, in dem wir entwickeln, gar nicht erscheinen würde.

[9] Siehe https://www.michsky.com/project/modern-ui-pack/ (letzter Abruf: 19.05.2021)

[10] Siehe https://vr-uikit.epibyte.com.au/ (letzter Abruf: 19.05.2021)

[11] Siehe https://assetstore.unity.com/packages/tools/gui/53258 (letzter Abruf: 19.05.2021)

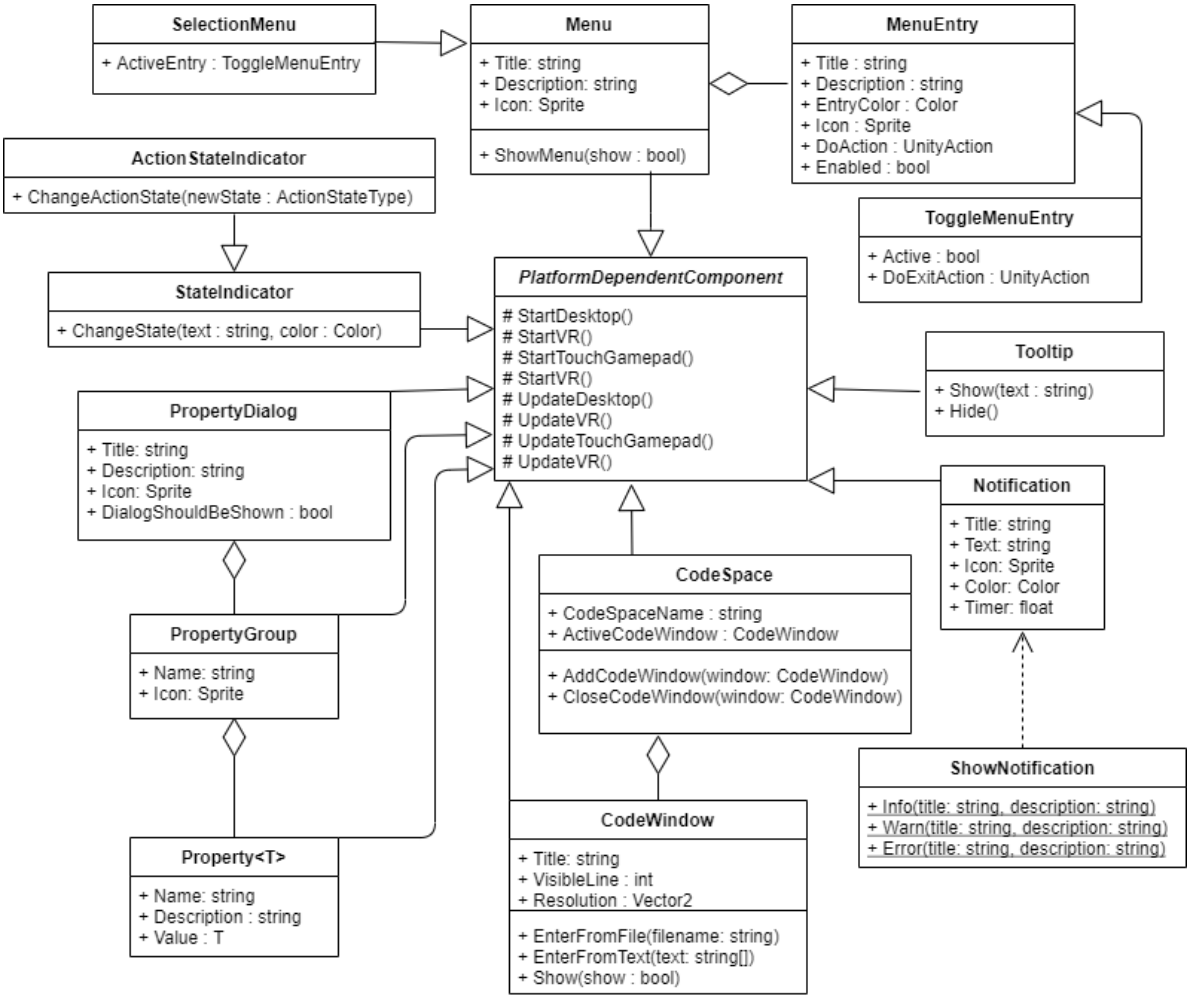

Abbildung 5: UML für das UI-Framework.

4.1.2 Code-Windows

Bei den Code-Windows handelt es sich um eine weitere Komponente des UI-Frameworks, diese bekommt aufgrund ihrer Komplexität jedoch ein eigenes Kapitel. Grundlegend ging es hier um die Darstellung von Quellcode des analysierten Projekts in SEE, es sind jedoch im Verlauf der Entwicklung weitere Features hinzugekommen, die hier auch erläutert werden.

Grundlagen Wenn die GXL-Datei einer geladenen Code-City das `Source.Path`-Attribut enthält, erhält der Spieler die Möglichkeit, durch das Anklicken eines Knotens ein zugeordnetes *Code-Window* zu öffnen. Das Code-Window ist (wie der Name vermuten lässt) ein Fenster, in welchem der Quelltext des Knotens angezeigt wird. Wie so etwas

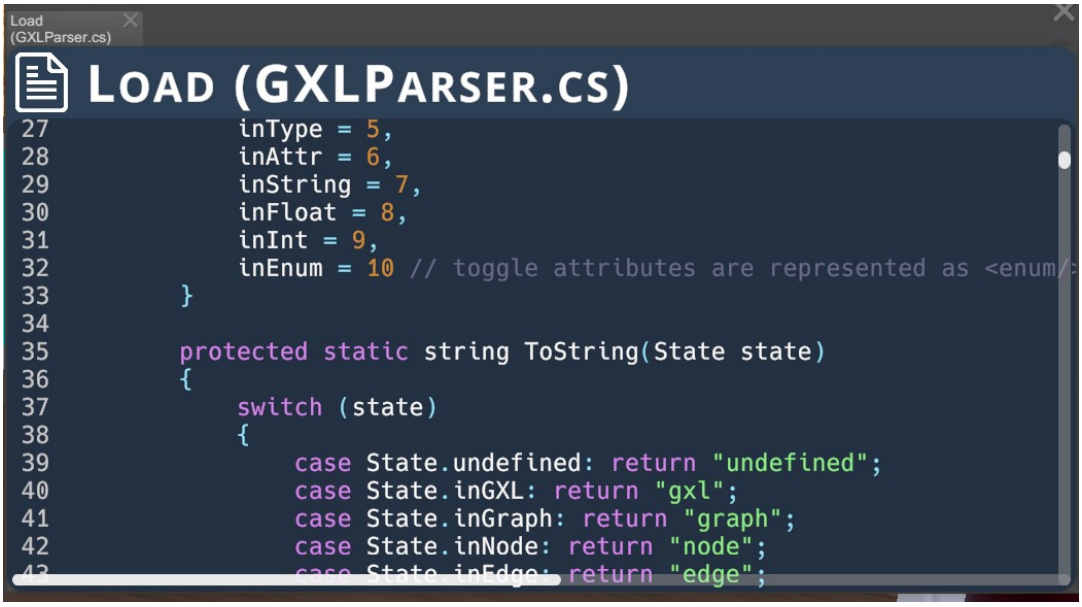

Abbildung 6: Ein *Code-Window* in einem *Code-Space* mit Syntax Highlighting.

aussieht, kann man in Abbildung 6 sehen. Da ein Knoten z. B. auch eine Methode repräsentieren kann (d. h. einen Teil innerhalb einer Datei), wird bei den dazugehörigen Code-Windows dann die Scrollposition an die Position innerhalb der Datei angepasst.

Syntax Highlighting Es werden mit *Antlr*[12] generierte Lexer verwendet, um das Syntax Highlighting umzusetzen. Hierfür wurde ein Datenmodell für Tokens und Sprachen umgesetzt, welches es sehr leicht macht, weitere Sprachen hinzuzufügen. Momentan werden die Programmiersprachen *Java*, *C++*, *C#*, *C* und *Python* unterstützt.

Mehrere Fenster *Code-Spaces* sind definiert als eine Sammlung an offenen Code-Windows. Es handelt sich hier (wie auch bei den Code-Windows) um `Platform-DependentComponents` (siehe Sektion 4.1.1 und Abbildung 5), bisher wurde aber nur die Desktop-Variante implementiert. Dort wurden die Code-Spaces als in der Position und Größe änderbare Fenster realisiert, in denen die Code-Windows als Tabs umgesetzt wurden[13]. Das kann ebenfalls oben in Abbildung 6 gesehen werden. Durch diesen Mechanismus kann leicht zwischen mehreren Code-Windows gewechselt werden.

Multiplayer Eine der wichtigsten Eigenschaften von SEE ist die Fähigkeit, mit mehreren Benutzern kollaborativ zu arbeiten. Dementsprechend darf diese Möglichkeit auch bei der Quelltextanzeige nicht fehlen. Die Umsetzung sieht dabei so aus, dass es einen

[12] https://www.antlr.org/ (letzter Abruf: 30.05.2021)
[13] Hierzu wurde das *Dynamic Panels*-Asset verwendet (https://github.com/yasirkula/UnityDynamicPanels, letzter Abruf: 21.05.2021).

`CodeSpaceManager` gibt, der jedem Spielernamen[14] einen *Code-Space* zuordnet. Bei einer Änderung (z. B. Scrollen zu einer anderen Zeile) wird der Code-Space mit dessen Code-Windows serialisiert — hierfür wurde das *Serialization Proxy Pattern* [Blo18] verwendet, wobei als „Proxy" hier innere Klassen namens `CodeWindowValues` und `CodeSpaceValues` verwendet werden. Diese Objekte werden dann vom SEE-Netzwerkcode (siehe z. B. Sektion 4.5) versendet und bei anderen Spielern vom `CodeSpaceManager` verarbeitet[15].

Spieler haben nun die Möglichkeit, in einem weiteren `SelectionMenu` (siehe Sektion 4.1.1) einen Spielernamen auszuwählen, dessen Code-Space sie betrachten[16] möchten. Das gerade aktive Code-Window und die aktuelle Scrollposition wird dann vom anderen Spieler kontinuierlich übernommen, der Code-Space wird quasi „gespiegelt".

4.2 Architekturvergleich

4.2.1 Aufbau einer Action

Damit ein oder mehrere Benutzer innerhalb des Architekturvergleichs weitere Interaktionsmöglichkeiten mit der Code-City besitzen, wurden für das Player-Menu weitere Interaktionsmöglichkeiten entwickelt. Folgende Actions wurden implementiert:

- **Add-Node-Action:** Eine Action zum Erstellen von neuen Knoten.
- **Scale-Node-Action:** Eine Action zum Skalieren vorhandener Knoten.
- **Edit-Node-Action:** Eine Action zum Bearbeiten von Knoten-Metriken.
- **Delete-Action:** Eine Action zum Löschen von Knoten und/oder Kanten.
- **Add-Edge-Action:** Eine Action zum Erstellen von neuen Kanten.
- **Draw-Action:** Eine Action zum Ziehen von Linien zwecks Markierung einzelner Komponenten.
- **Hide-Action:** Eine Action zum Verstecken und Hervorheben von zusammenhängenden Knoten und Kanten im Graphen.

Im Folgenden wird die Konzeption, die Implementierung und das Design sowie der Weg zur Entwicklung des aktuellen Standes für jede Action dargestellt. Dabei werden auch Probleme beleuchtet, die während der Entwicklung aufgetreten sind und gelöst werden mussten. Zur Veranschaulichung der implementierten Struktur der Actions wurde in Abbildung 7 ein vereinfachtes UML-Diagramm erstellt.

[14] Momentan ist der Spielername die IP-Adresse des jeweiligen Benutzers.

[15] Hier gibt es zwei mögliche Betriebsmodi: Entweder der gesamte Dateiinhalt wird über das Netzwerk übertragen, oder nur der Dateipfad. Letzteres setzt voraus, dass alle Spieler die Dateien besitzen.

[16] Dieses Fenster kann auch tatsächlich nur betrachtet, aber nicht geschlossen werden. Die Scrollposition wird ebenfalls nicht übertragen, wenn einem dieses Fenster nicht „gehört".

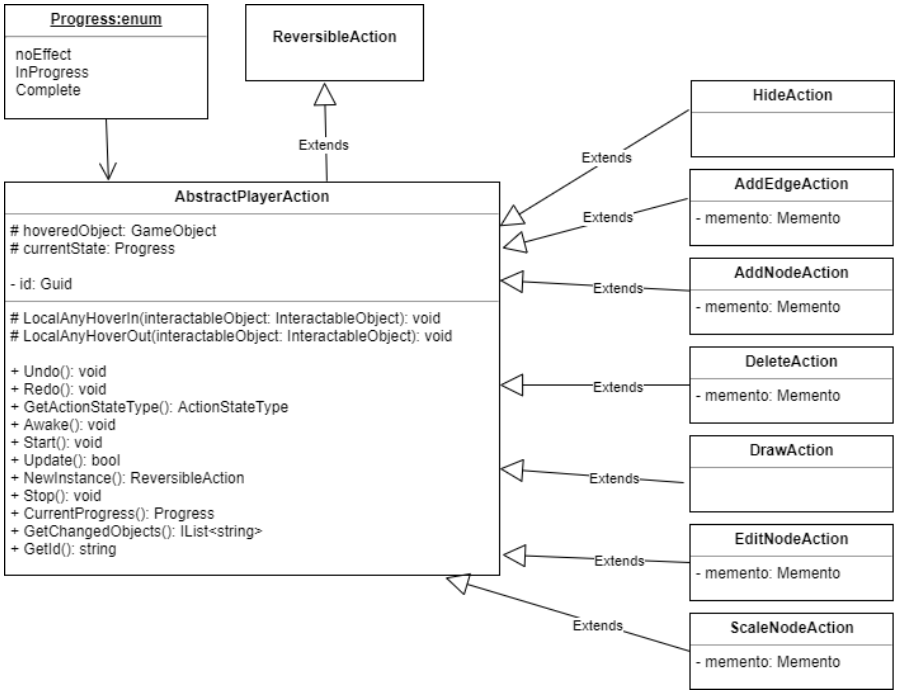

Abbildung 7: UML Diagramm zu den Actions

4.2.2 Add-Node-Action

Zunächst wurde eine Action erstellt, die die Erzeugung von neuen Knoten ermöglicht. Hierbei mussten zunächst konzeptionell einige Dinge geklärt werden. Da jeder Knoten über Metriken verfügt, müssen diese bei der Erstellung eines neuen Knotens initialisiert werden. Es muss also geklärt werden, ob diese mit einem Standardwert initialisiert werden sollen, oder ob der Benutzer Werte für diese Metriken übergeben soll. Danach muss dieser Knoten nicht nur als ein GameObject erzeugt werden, sondern auch in die Graphen-Hierachie eingefügt werden. Ein weiterer wichtiger Punkt bei den Überlegungen zu Beginn war, dass der Benutzer zwischen den Arten eines Knotens differenzieren können will. Konkret bedeutet dies eine Unterscheidung von einem Blatt (dem letzten Knoten eines Zweiges) und einem inneren Knoten, auf den weitere innere Knoten oder Blätter folgen können. Der letzte wichtige Faktor, für den es eine Grundsatzentscheidung bedurfte, war der der Platzierung eines neuen Knotens im Graphen. Nach einigen Überlegungen wurden diese Fragen und Probleme zunächst folgendermaßen beantwortet:

Der Benutzer möchte durch einen Klick auf die jeweilige Stadt auswählen, in welcher Stadt ein neuer Knoten erstellt werden soll. Daraufhin soll sich ein Menü öffnen, in dem er den Namen, den Typen und die Hierarchiestufe (Blatt oder innerer Knoten) des neuen Knotens bestimmen kann. Im Hintergrund soll der Knoten parallel in die Hierarchie des Graphen eingefügt werden. Alle weiteren Metriken bleiben zunächst nicht initialisiert. Abschließend soll nach dem Schließen des Menüs der erstellte Knoten am Mauszeiger

erscheinen und durch einen erneuten Klick innerhalb der neuen Stadt an genau der Position des Mauszeigers abgesetzt werden können. In Abbildung 8 und Abbildung 9 wird die erste Umsetzung dieses Anwendungsfalls gezeigt.

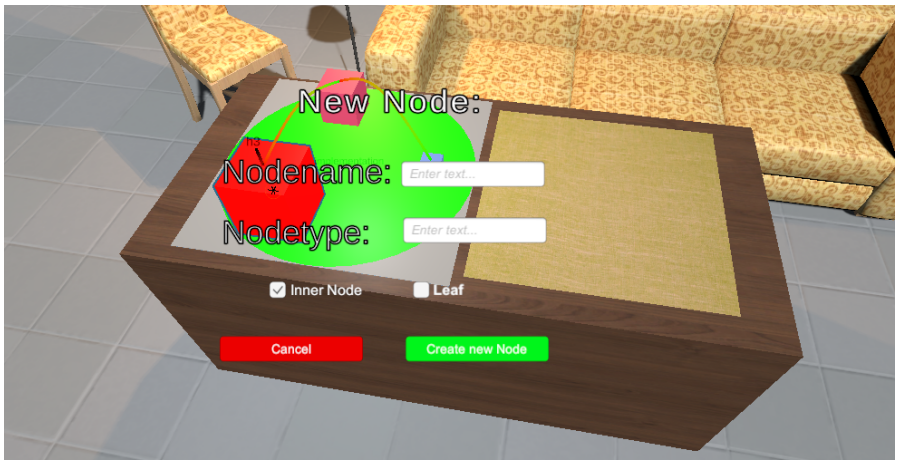

Abbildung 8: Hinzufügen eines Knotens vorher – Eingabe der Metriken

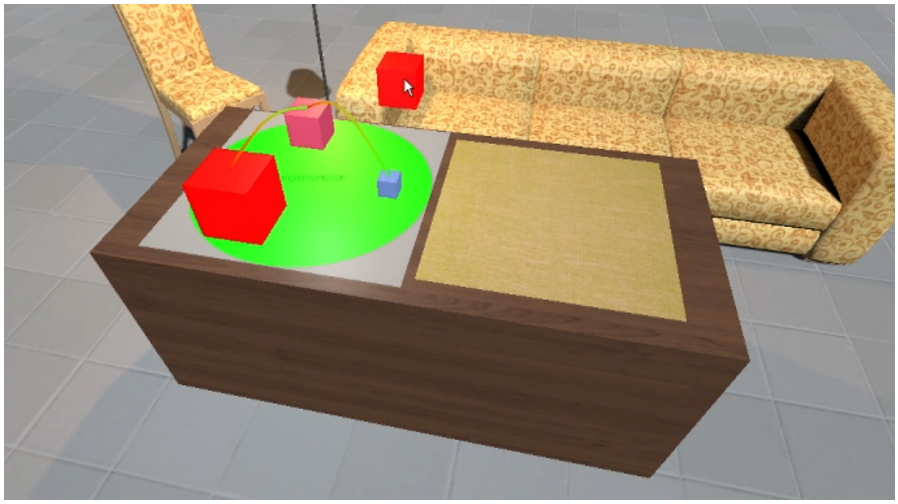

Abbildung 9: Hinzufügen eines Knotens vorher – Platzieren des Knotens

Nach dem Anklicken der Stadt, dessen Wurzel während des Hoverings über die jeweilige Stadt in einer festgelegten Farbe (hier grün) eingefärbt wird, öffnet sich ein Canvas mit darauf enthaltenen Input-Feldern und Checkboxen, in die Werte eingetragen werden können. Zum Zeitpunkt der Entwicklung des User-Interfaces für die Wert-Eingabe war jedoch schon klar, dass dieses Interface in näherer Zukunft durch ein standardisiertes User-Interface (siehe Sektion 4.1.1) ersetzt werden soll. Deshalb handelte es sich hier von Beginn an um ein Provisorium, das hauptsächlich zu Testzwecken

integriert wurde. Nachdem die Werte aus den Eingabefeldern ausgewertet wurden, wozu Unity-interne Kompetenzen zur Erzeugung und Zerstörung von GameObjects, Komponenten und Prefabs erworben werden mussten, erschien der neue Knoten am Mauszeiger. Die Größe des neuen Knotens wurde mithilfe einer Median-Berechnung aller existierenden Knoten bestimmt. Langfristig sollte auch hier das Ziel sein, die grafischen Ausprägungen durch die Bestimmung von Metriken festzulegen. Bei der Bewegung des Knotens durch die Szene gab es ebenfalls einige Probleme, da die Größe während der Bewegung von der Größe der Entfernung des Mauszeigers zum Benutzer abhing. Dies hing mit den Vektoren bzw. Quaternionen zusammen. Die Umrechnung der Mausposition in einen Vektor verändert alle Koordinaten, auch die Distanz von dem Mauszeiger zur Szenen-Kamera. Dieses Problem wurde ebenfalls nicht in diesem Issue behandelt. Nach der Absetzung des Knotens färbte sich die Wurzel wieder zurück in den ursprünglichen Farbton.

Es wird relativ schnell deutlich, dass diese Art der Implementierung zwar eine solide Arbeitsversion bot, aber einige Probleme mit sich brachte, die gelöst werden sollten. Im Verlaufe des Projektes wurden also die Grundannahmen zur Knotenerzeugung überdacht:

Zunächst muss ein Knoten nur initialisiert und der Graphenhierachie hinzugefügt werden. Die Festlegung von Metriken wurde aus dem Prozess der Knotenerstellung in die Actions Scale-Node und Edit-Node ausgegliedert. Die einzig verbleibenden wichtigen Festlegungen für die Erstellung eines Knotens sollen in Zukunft lediglich durch die Auswahl der Stadt und die Auswahl der Position innerhalb der Stadt vorgenommen werden. Hieraus wurde eine neue Version erstellt, die genau diese Prämissen berücksichtigte (siehe Abbildung 10).

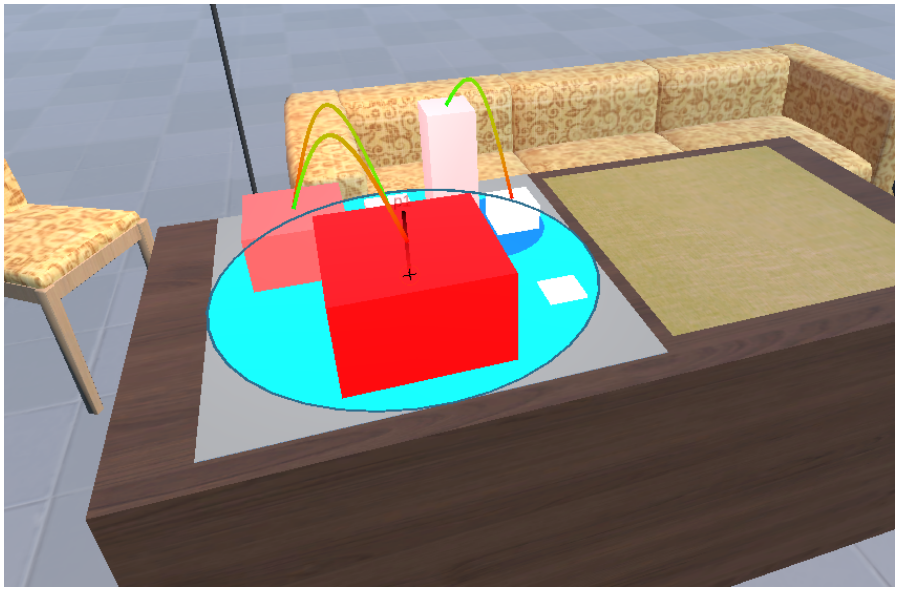

Abbildung 10: Hinzufügen eines Knotens nachher

Aus dieser Implementierung ergaben sich weitere Probleme: Fügt der Nutzer einen neuen Knoten auf einem Blattknoten hinzu, so stapeln sich die Blattknoten übereinander. Normalerweise wird durch diese Veränderung der vorherige Blattknoten ein innerer Knoten. Damit ist auch eine andere Darstellung verbunden, denn im bisherigen Graphen werden die inneren Knoten als Kreise dargestellt, während das Privileg des würfelförmigen Körpers nur für Blattknoten vorbehalten ist. Dementsprechend resultierte dies in einer Anpassung der AddNode-Action und einem weiteren Problem.

Durch die Anpassung wird beim Erzeugen eines neuen Knotens auf einem Blattknoten dieser als Parent gespeichert und in einen kreisförmig dargestellten inneren Knoten umgewandelt. Dafür wird der ursprüngliche Knoten erst gelöscht und dann neu gezeichnet. Das daraus wachsende Problem der Höhenmetrik benötigt ebenfalls einen Lösungsansatz. Die Höhe des Blattknotens vor der Umwandlung ist stellvertretend für eine Metrik. Damit diese nicht verloren geht, wird der innere Knoten nicht als Kreis, sondern als flacher Zylinder dargestellt und das Volumen ergibt sich aus der Berücksichtigung aller vorangegangen Höhen.

4.2.3 Edit-Node-Action

Die Edit-Node-Action verhält sich sehr ähnlich zur Add-Node-Action, mit dem Unterschied, dass hier keine Objekte neu erzeugt werden. Ihre einzige Funktion ist es, Metriken eines Knotens zu editieren. Es wurde also, unter derselben Voraussetzung wie zuvor, dass das User-Interface nur für eine kurze Zeit existiert und danach ersetzt werden soll, ein Canvas mit Input-Fields ähnlich dem in Abbildung 9 erstellt. Auf diesem wurden die Metriken des ausgewählten Knotens angezeigt und konnten mit neuen Werten überschrieben werden. Der zu editierende Knoten sollte per Klick auswählbar gemacht werden, wodurch sich der eben beschriebene Canvas öffnen sollte.

Das wurde genau so umgesetzt und stellte den Beginn der Edit-Node-Action dar. Die einzige Änderung, die daran bisher ausgeführt wurde, ist die Ersetzung des User-Interfaces zur Eingabe der Metriken durch das standardisierte User-Interface (siehe Sektion 4.1.1). Technisch änderte sich bisher nichts. Langfristig wäre das Ziel, alle Metriken des Knotens über diesen Dialog verändern zu können, welche dann direkt in die grafische Repräsentation (Farbe, Größe, etc.) übersetzt werden. Bisher ist lediglich eine Veränderung von Namen und Typ des Knotens möglich, die noch keine Auswirkungen auf den grafischen Teil des Graphens besitzt. (siehe Abbildung 11)

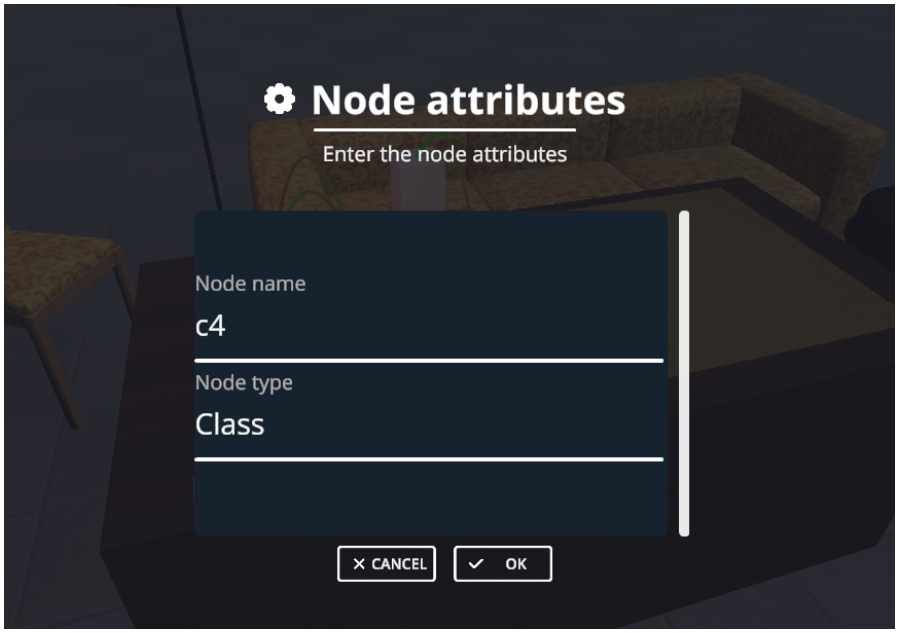

Abbildung 11: Bearbeiten eines Knotens

4.2.4 Add-Edge-Action

Um die Kommunikation zwischen Komponenten bzw. Klassen darzustellen war es weiterhin wichtig, auch neue Kanten zwischen Knoten ziehen zu können. Aus diesem Grund wurde die Add-Edge-Action erstellt. Auch hier war zunächst die Herausforderung, die neu erzeugte Kante ebenfalls in die Hierarchie des Graphens zu integrieren. Dies wurde zügig mittels einer Funktion im Graph-Renderer gelöst. Die zweite Herausforderung stellte die Auswahl der Knoten dar, zwischen denen eine Kante gezogen werden soll. Es wurde eine sehr einfache Umsetzung gewählt, bei der der zuerst angeklickte Knoten den Start der Kante und der danach angeklickte Knoten das Ziel der Kante repräsentierte. Der Startknoten konnte bei falscher Auswahl aber auch aus dem temporären Speicher entfernt werden, sodass ein neuer Startknoten gewählt werden konnte, bevor eine neue Kante gezogen wurde. Auf Abbildung 12 ist im Vergleich zu z. B. Abbildung 10 zu erkennen, wie zwischen dem großen, roten Knoten eine neue Kante zum hohen, lachsfarbenen Knoten gezogen wurde.

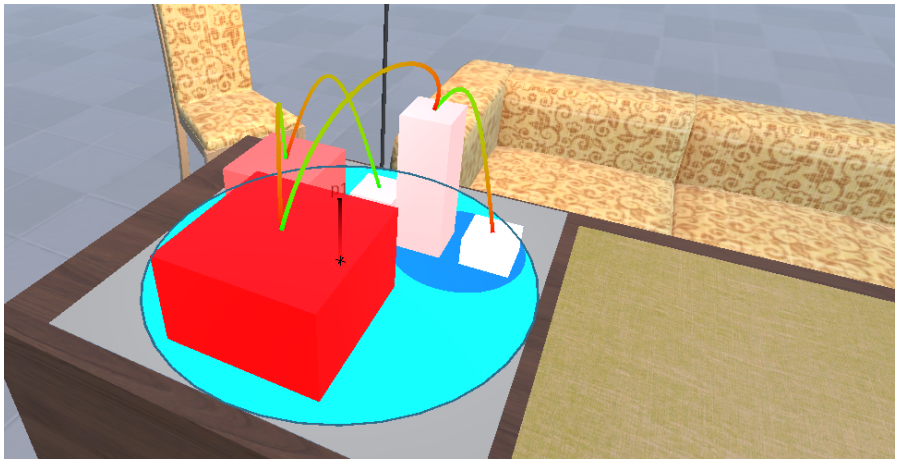

Abbildung 12: Hinzufügen einer Kante

4.2.5 Scale-Node-Action

Um z. B. die Relevanz eines Knotens hervorzuheben, sollte im Rahmen der Scale-Node-Action eine Funktion implementiert werden, die den Knoten in seiner Größe und Form skalierbar macht. Das Konzept, welches hierfür entwickelt wurde, kann folgendermaßen zusammengefasst werden: Ein Knoten sollte nach dem Anklicken über erscheinende Skalier-Punkte an seinen Ecken in seiner Größe veränderbar sein.

Ein Problem, das berücksichtigt werden musste, war jedoch, dass ein Knoten von seinem Vorläufer herunterskaliert werden konnte, wenn an einer Seite des Knotens geschoben und an der anderen gezogen wurde. Da dies aber keine beabsichtigte Funktion des Skalierens war, wurde eine Methode entwickelt, die lediglich eine symmetrische Skalierung um das Zentrum des Knotens ermöglichte. Das bedeutet konkret, dass ein Knoten, der an der linken Seite um Faktor x gestreckt wird, an der rechten Seite ebenfalls um Faktor x gestreckt wird. Dasselbe gilt auch für Stauchungen, nur bei der Höhenskalierung musste dieses Konzept logischerweise nicht berücksichtigt werden. Zu Beginn der Entwicklung gab es zusätzlich zu den acht Sphären an den Eck- bzw. Kantenmittelpunkten des Knotens noch eine grüne Sphäre, die den Skaliervorgang beendete und eine rote Sphäre, die den Skaliervorgang abbrach und die Skalierung des Knotens auf seine vorherige Größe zurücksetzte (siehe Abbildung 13). Im Rahmen der Implementierung eines Undo's (siehe Sektion 4.6.1) wurden diese Sphären allerdings entfernt und ein Klick auf ein anderes Objekt in der Szene beendet nun den Skaliervorgang (siehe Abbildung 14). Nachdem dies abgeschlossen war, wurde dafür gesorgt, dass sich die Kanten, die mit den Knoten verbunden sein können, auch entsprechend der Skalierung verschieben. Dies wurde im Laufe der Entwicklung über ein simples Neuzeichnen der Kanten bei jeder Änderung der verbundenen Knoten realisiert.

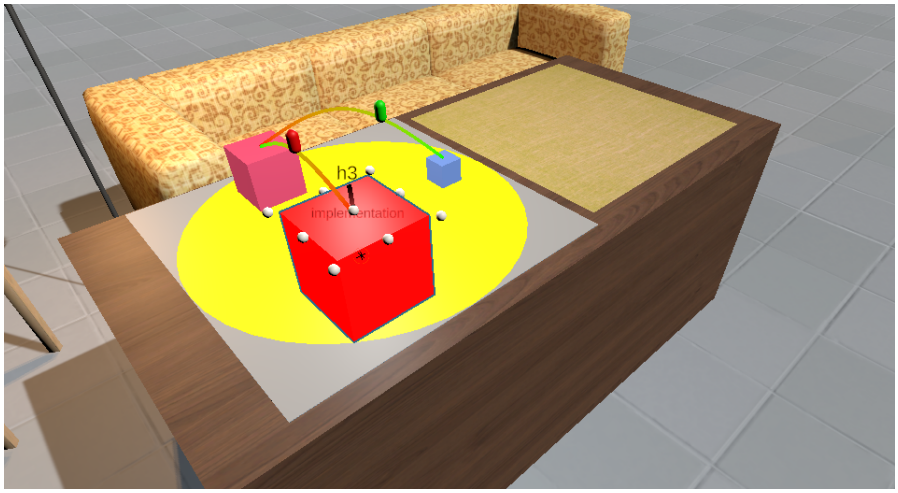

Abbildung 13: Skalieren eines Knotens vorher

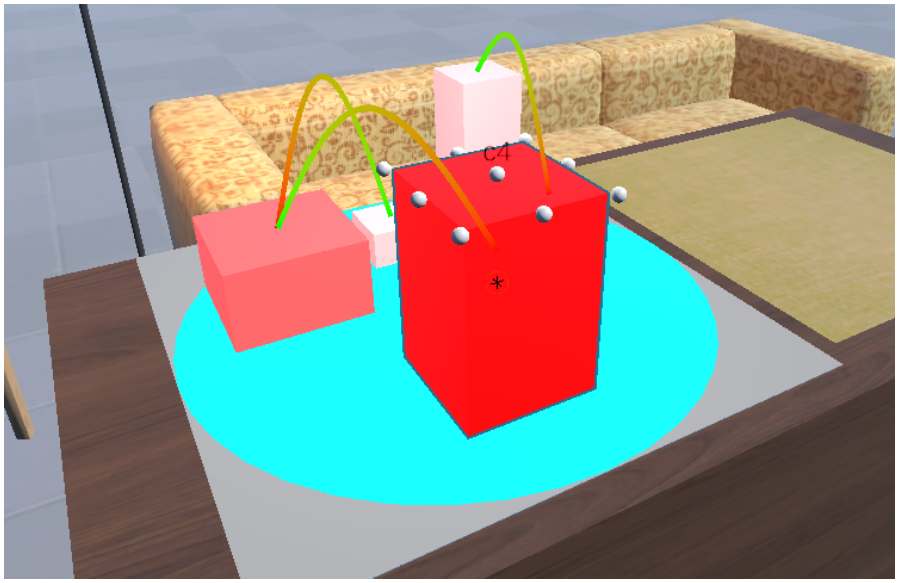

Abbildung 14: Skalieren eines Knotens nachher

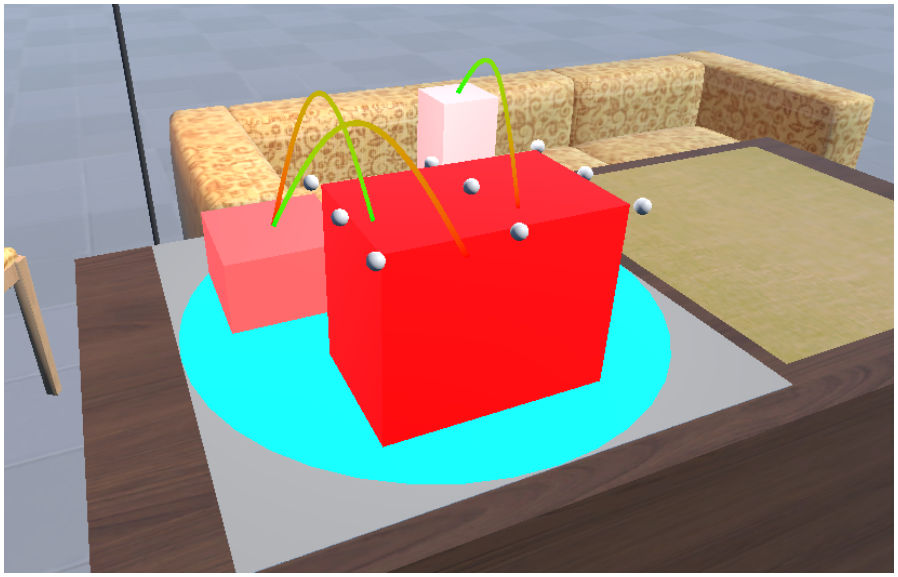

Abbildung 15: Skalieren eines Knotens – gestreckter Knoten

4.2.6 Delete-Action

Damit bspw. obsolete Knoten oder Kanten entfernt werden können, wurde weiterhin eine Delete-Action entwickelt. Diese hat, ähnlich zur Add-Node-Action, zwei wesentliche Aufgaben: Zunächst muss das jeweilige GameObject innerhalb von Unity entfernt werden, damit der Knoten verschwindet. Parallel dazu muss der Knoten oder die Kante aber ebenfalls aus der Hierarchie des Graphen entfernt werden.

Ein weiterer wichtiger Faktor, der bedacht werden muss, ist die Entfernung bestimmter abhängiger Graphen-Objekte. So müssen z. B. alle mit dem Knoten verbundenen Kanten ebenfalls entfernt werden und auch alle Kindknoten und Kanten der Kindknoten gelöscht werden, falls es sich bei dem gelöschten Knoten nicht um ein Blatt, sondern um einen inneren Knoten mit Kindknoten handelt.

Diese Anforderungen wurden zunächst so umgesetzt, dass der gelöschte Knoten bzw. die Kante mit all seinen Abhängigkeiten durch ein Anklicken markiert und durch eine Betätigung einer bestimmten Taste der Tastatur danach sofort gelöscht wurde. Langfristig sollte hier aber ein Konzept entwickelt werden, dass den Knoten für den Benutzer erkennbarer löscht und auch wieder hervorholbar macht (siehe Sektion 4.6.1).

Deshalb wurde mittels einer Co-Routine eine Animation entwickelt, die den Knoten zu einem neu hinzugefügten Mülleimer bewegt und darin verschwinden lässt (siehe Abbildung 16). Die zuvor beschriebene Herausforderung des Löschens mehrerer abhängiger Knoten bzw. Kanten bestand jedoch auch für die Animation der Bewegung zum Mülleimer. Dafür konnte aber auch eine Lösung implementiert werden und ermöglichte so eine bessere Darstellungsart des Löschvorgangs. Ein wesentlicher Unterschied in der Implementierung zur vorherigen Version ist dadurch gekennzeichnet, dass ein GameObject des Graphens nun nicht mehr im eigentlichen Sinne gelöscht, sondern

lediglich aus der Hierarchie des Graphen entfernt wird. Das GameObject wird nach dem Verschieben in den Mülleimer dort nur verkleinert dargestellt.

In Abbildung 16 ist die Animation der Bewegung eines Knotens zum Mülleimer dargestellt. Dieser wird oberhalb des Mülleimers auf eine für die Öffnung passende Größe skaliert und daraufhin in den Mülleimer abgesenkt. Der Verkleinerungsvorgang des Knotens ist dabei reversibel, wird also beim Hervorholen aus dem Mülleimer wieder rückgängig gemacht. In Abbildung 17 wird dieser Vorgang für einen inneren Knoten (der blau gefärbte) dargestellt, dessen Kindknoten zeitgleich mit zum Mülleimer bewegt werden. Die Implementierung dieser gesamten Routinen wurde häufiger bearbeitet. Die früheren Versionen enthielten auf Quelltext-Ebene die gesamten Vorgänge des Löschens aus dem Graphen und die der animierten Verschiebungen der GameObjects in einer Klasse. In der jetzigen Version wurde vor dem Hintergrund des Designprinzips des sogenannten *Seperation Of Concerns* die Animation strikt von den Löschvorgängen und analog dem Hinzufügen der Knoten zum Graphen getrennt. Dieses Muster erweist sich als hilfreich für die Umsetzung der Aktionen im Netzwerk und verbessert zugleich die Les- und Wartbarkeit des Quelltextes.

Abschließend, aber auch in Hinblick auf die Zukunft, blieb ein marginaler Teilaspekt bis dato unbearbeitet. Da Kanten gesondert von Knoten behandelt werden müssen, werden Kanten bisher nicht in den Mülleimer bewegt, sondern lediglich zu Beginn der Animation unsichtbar gesetzt. Das gilt auch für das Löschen einzelner Kanten. Eine sinnvolle Animation hierfür ist aber ebenfalls in Arbeit, um vollständige Konsistenz zu gewährleisten.

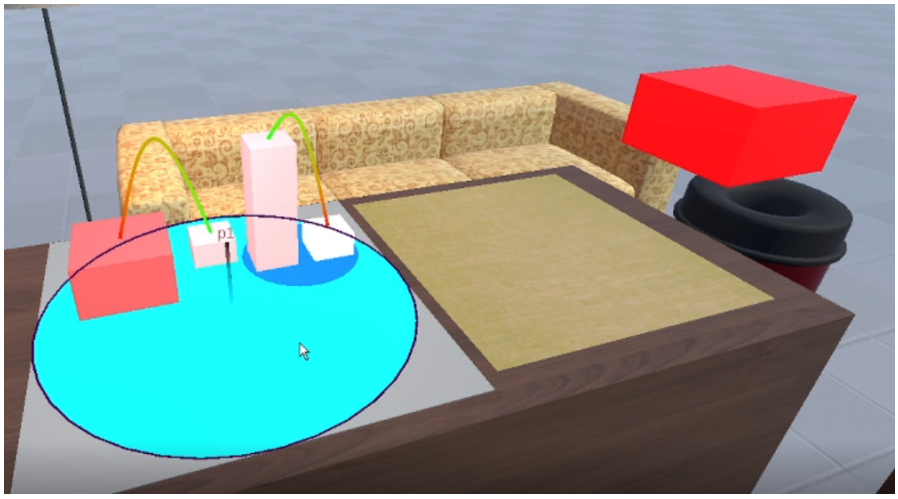

Abbildung 16: Löschen eines Knotens(1)

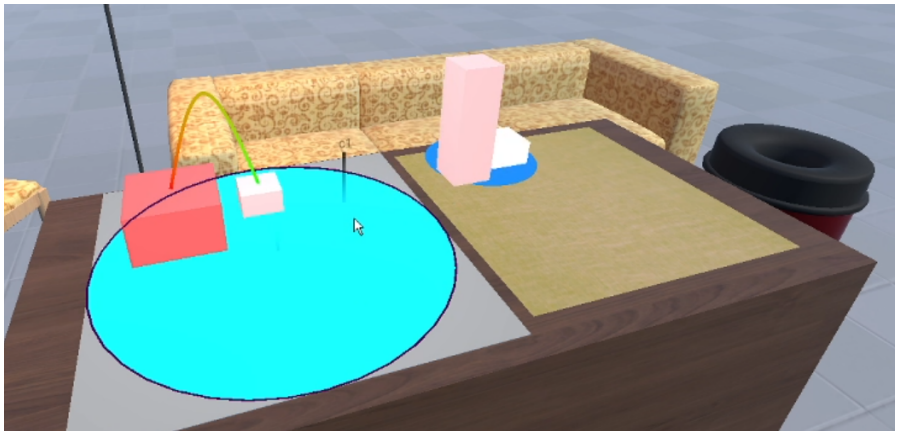

Abbildung 17: Löschen eines Knotens(2)

4.2.7 Draw-Action

Eine Action, die zum Zeitpunkt der Erstellung dieses Berichtes noch in der Entwicklungsphase ist, ist die Draw-Action. Die Funktion dieser Action soll die Markierung bestimmter Komponenten des Graphens sein. Hierfür kann der Nutzer eine beliebig lange Linie mittels gedrückter Maustaste innerhalb der Szene ziehen (siehe Abbildung 18). Das Problem, welches diese Action allerdings verursacht ist, wie bereits beim Erzeugen und Absetzen eines Knotens bei der Add-Node-Action beschrieben, die Perspektive der Linie innerhalb von Unity.

Eine Linie, die gezogen wird, setzt am Mauszeiger an. Wenn dieser nicht in allen Koordinaten exakt bei der jeweiligen Komponente ist, sorgt dies dafür, dass die Markierung nur aus genau diesem Blickwinkel sinnvoll erscheint. Auch die perspektivischen Besonderheiten in Unity sind hier noch nicht vollständig berücksichtigt. So liegen die Knoten perspektivisch oft vor den gezogenen Linien, was darin resultiert, dass diese nicht zu erkennen sind. Dies ist am besten visuell zu veranschaulichen (siehe Abbildung 19). Diese Probleme müssen in Zukunft noch angegangen werden, werden jedoch nicht im Rahmen dieses Projektes abzuschließen sein.

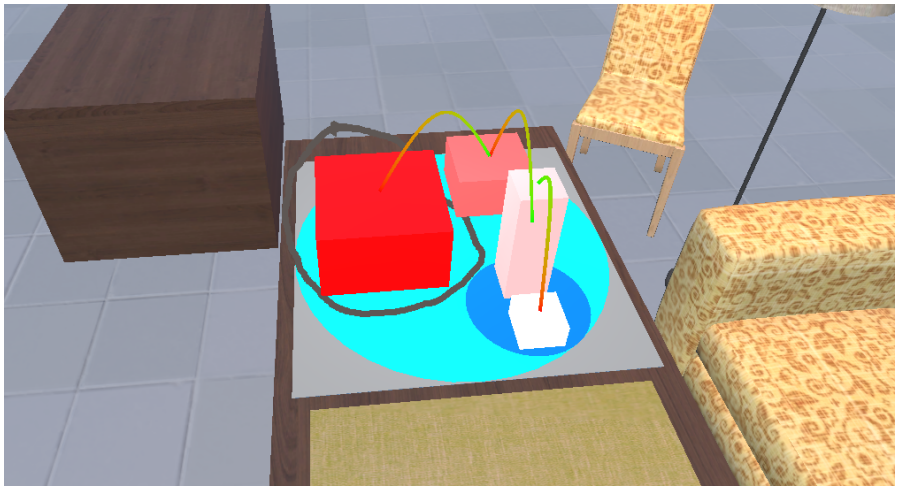

Abbildung 18: Zeichnen einer Linie

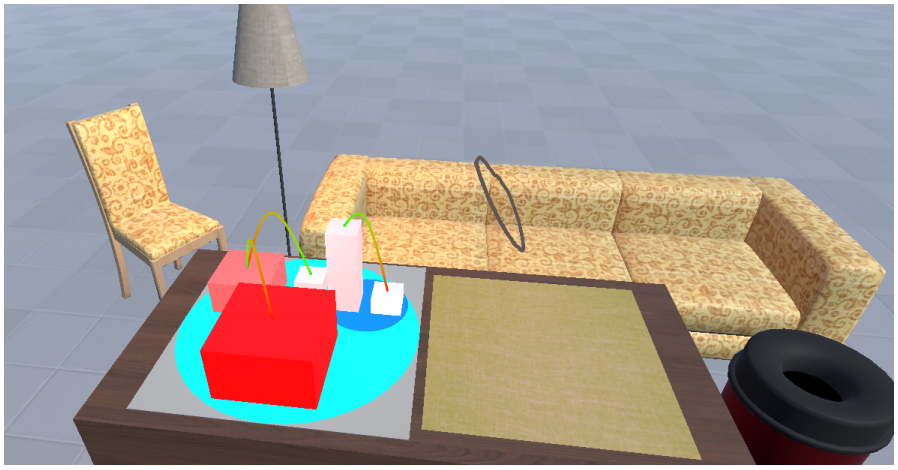

Abbildung 19: Perspektivisches Problem beim Zeichnen einer Linie

4.2.8 Hide-Action

Die Hide-Action dient zum Hervorheben und Verstecken von Kanten und Knoten, um die Übersichtlichkeit innerhalb eines Graphen zu verbessern. Um dies möglichst sinnvoll und nützlich zu gestalten wurden folgende Funktionen implementiert:

- **Hide All:** Versteckt den gesamten Graphen.
- **Hide Incoming:** Versteckt für einen ausgewählten Knoten alle eingehenden Kanten, sowie die Knoten, die mit diesen Kanten verbunden sind.
- **Hide Outgoing:** Versteckt für einen ausgewählten Knoten alle ausgehenden Kanten, sowie die Knoten, die mit diesen Kanten verbunden sind.

- **Hide forward transitive closure:** Versteckt alle Knoten und Kanten, die von dem ausgewählten Knoten über dessen ausgehende Kanten erreichbar sind.
- **Hide backward transitive closure:** Versteckt alle Knoten und Kanten, die von dem ausgewählten Knoten über dessen eingehende Kanten erreichbar sind.
- **Hide transitive closure:** Versteckt alle Knoten und Kanten, die nicht direkt von dem ausgewählten Knoten über dessen ein- und ausgehende Kanten erreichbar sind.
- **Hide selected:** Versteckt alle ausgewählten Knoten, sowie dessen ein- und ausgehende Kanten.
- **Hide unselected:** Versteckt alle unausgewählten Knoten, sowie dessen ein- und ausgehende Kanten.
- **Hide all edges of selected:** Versteckt alle ein- und ausgehende Kanten aller ausgewählten Knoten.
- **Highlight connecting Edges:** Hebt alle Kanten hervor, die eine Verbindung zwischen den ausgewählten Knoten darstellen (siehe Abbildung 22).

All diese Funktionen können dafür genutzt werden, um besser Zusammenhänge und Verbindungen zwischen den einzelnen Knoten innerhalb der Graphen zu erkennen. Im Anschluss der Implementierung wurden alle Funktionen über ein Menü in der UI erreichbar gemacht (siehe Abbildung 20).

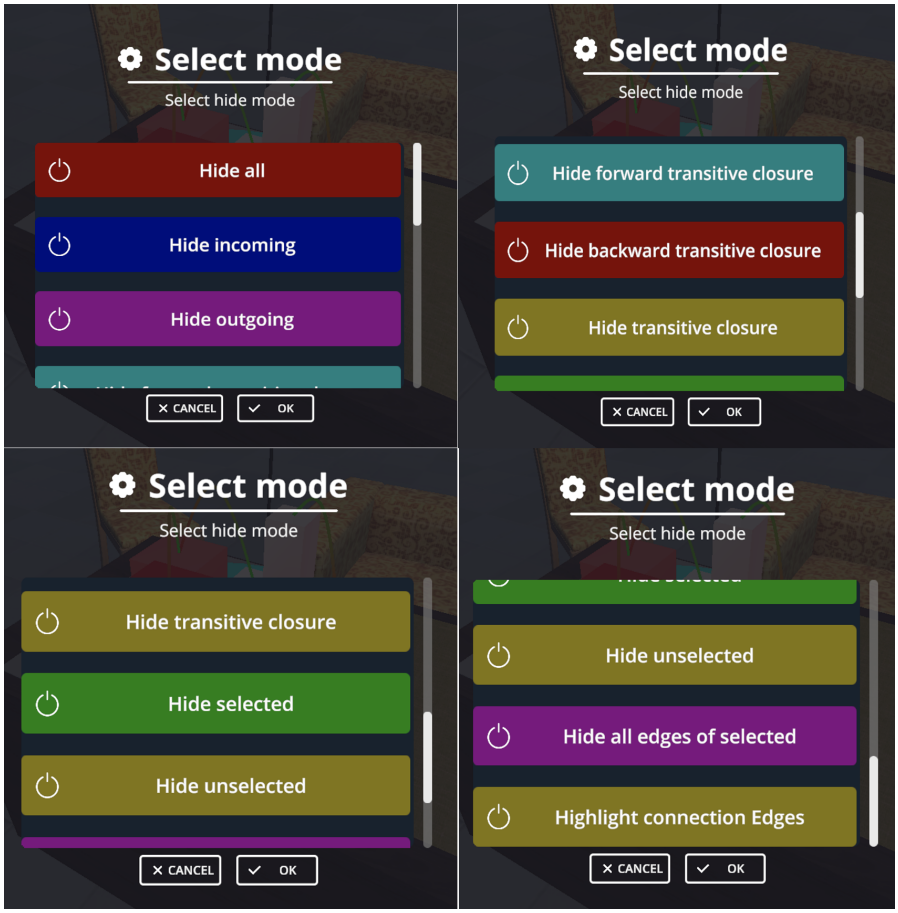

Abbildung 20: Hide Show Menü

Um die angesprochene Selektion einzelner Knoten und Kanten zu vereinfachen wurde dafür gesorgt, dass ausgewählte Objekte opak und unausgewählte Objekte transluzent erscheinen (siehe Abbildung 21).

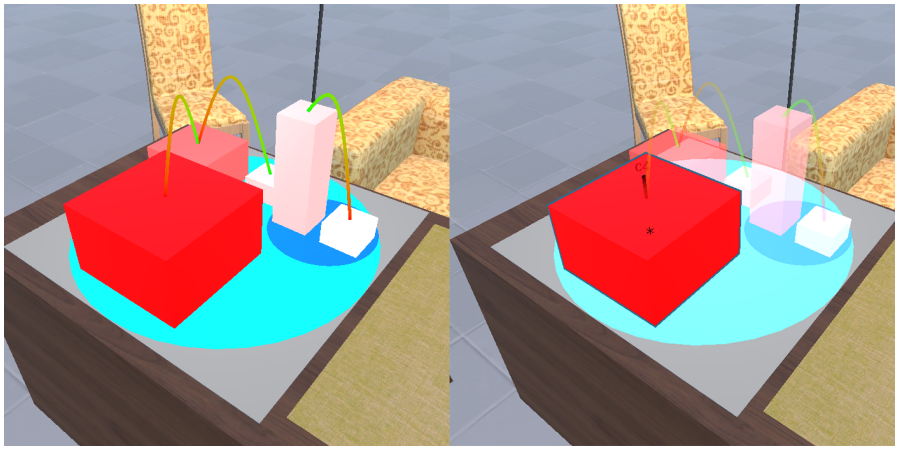

Abbildung 21: Auswahl der Objekte

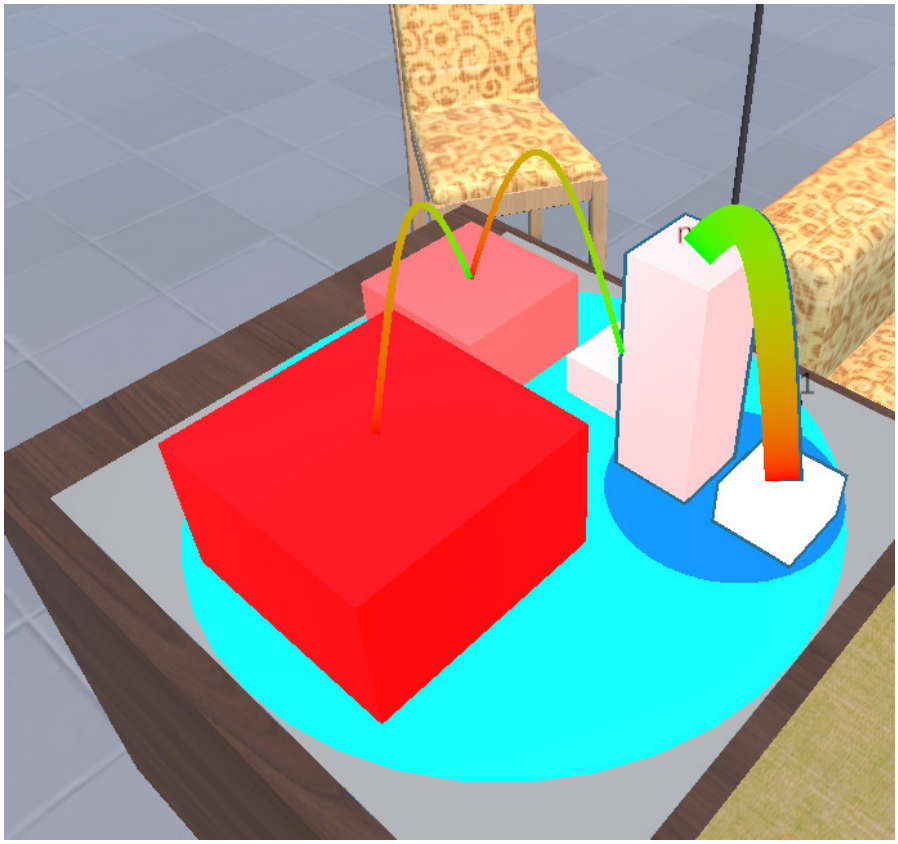

Abbildung 22: Hervorheben der verbindenden Kanten

Mittels dieser Funktionen können Nutzer unwichtige Elemente ausblenden bzw. wichtige Elemente hervorheben.

4.3 Evolution

4.3.1 User Interface / Verbesserungen

4.3.1.1 Verbesserte Auswahl der Graphrevision Zur Verbesserung der Interaktion mit der Evolution wurde eine neue UI implementiert (siehe Abbildung 23 und Abbildung 24). Vorher gab es zur Selektion der Revision ein Dropdown-Menü und je eine Tastaturbelegung für das Anzeigen der nächsten bzw. vorherigen Revision. Das Abspielen bzw. Pausieren und die Geschwindigkeitseinstellung der Animation wurde über die Tastatur geregelt. Dieses sollte intuitiver gestaltet werden, sodass die Interaktion ähnlich wie bei einem Video-Player funktioniert. Die Selektion der Revision sollte also durch einen Schieberegler und das Abspielen bzw. Pausieren und das Einstellen der Geschwindigkeit der Animation sollte über spezielle Knöpfe in der UI geregelt werden. Zudem sollte zusätzlich die Funktionalität eingebaut werden, Markierungen zu setzen, um interessante Punkte zu markieren und diese mit Kommentaren zu versehen. Auch sollte es möglich sein, die Animation nicht nur vorwärts, sondern auch rückwärts abzuspielen.

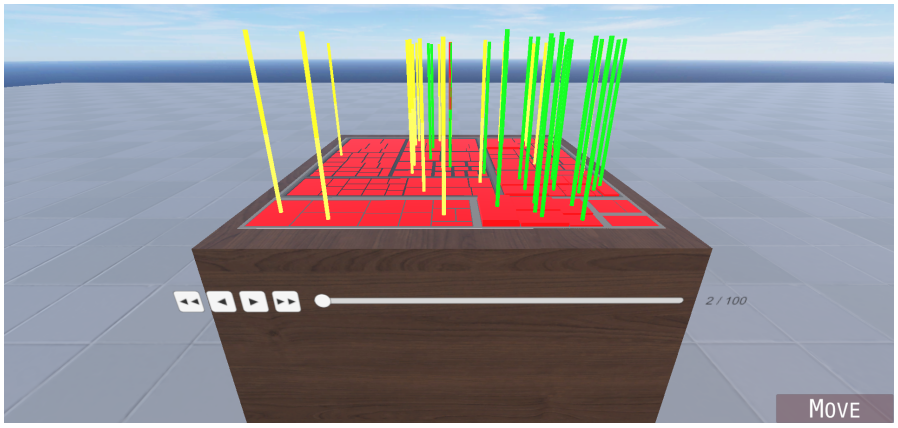

Abbildung 23: Evolution mit neuer UI

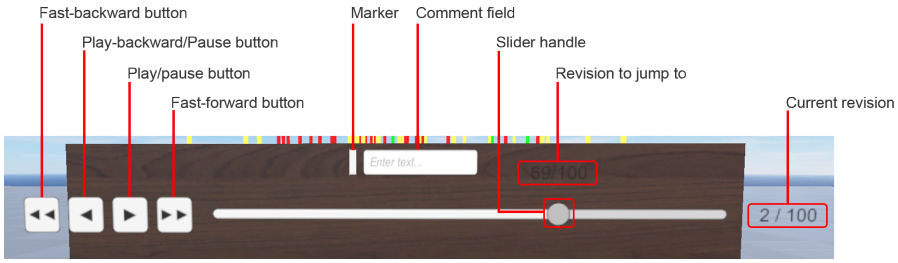

Abbildung 24: Neue UI im Detail (beschriftet)

Wie in den Abbildungen 23 und 24 zu erkennen ist, wurde die UI so implementiert, dass sie an dem Tisch, auf dem die Evolutionsanimation abgespielt wird, befestigt ist. Zudem

wurden seperate Knöpfe für das Vorwärts-Abspielen, Rückwärts-Abspielen, Schneller-Vorwärts-Abspielen und Schneller-Rückwärts-Abspielen hinzugefügt. Das Selektieren der Revisionen wurde durch einen Schieberegler implementiert. Wenn der Nutzer diesen bewegt, wird über diesem die Zahl der Revision angezeigt, zu welcher gesprungen werden würde. Zudem wird am rechten Ende des Schiebereglers die Zahl der aktuell angezeigten Revision dargestellt. Das Einfügen der Markierungen funktioniert durch das Drücken der Taste `Insert` bzw. `Einfg` auf Tastaturen mit deutscher Tastaturbelegung. Mit dem Klicken auf eine Markierung kann diese selektiert und zudem das Kommentarfeld geöffnet werden. Die Markierungen können durch das Drücken der `Del`- bzw. der `Entf`-Taste wieder gelöscht werden. Damit die Markierungen zwischen Sitzungen persistieren werden diese gespeichert, wenn SEE geschlossen wird.

Um die Funktionalität der Knöpfe noch einmal anschaulicher zu machen, wurde Abbildung 25 eingefügt, welche genau erklärt, welcher Knopf welche Funktion hat.

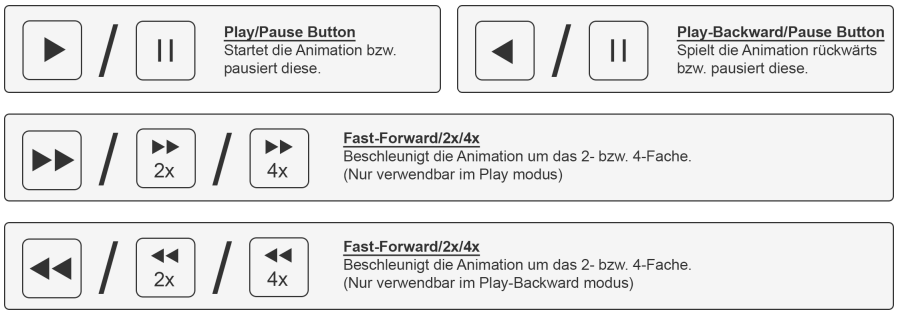

Abbildung 25: Knöpfe der neuen UI

Wie zu erkennen ist, hat also jeder der Knöpfe mehrere Funktionen abhängig davon, wie oft er gedrückt worden ist. Wenn z. B. der Startknopf einmal gedrückt wird, ändert er sich in einen Pauseknopf. Wird er erneut betätigt, wird er wieder zum Startknopf. Genauso funktioniert es bei den Knöpfen, welche die Geschwindigkeit der Animation einstellen. Wenn diese einmal gedrückt werden, wird die Animation in doppelter Geschwindigkeit abgespielt. Bei dem zweiten Drücken dann in vierfacher Geschwindigkeit und bei einem dritten Drücken wieder in normaler Geschwindigkeit.

Für einen Großteil der Interaktionen mit der Evolutionsanimation waren bereits Funktionen implementiert, somit mussten nur die Funktionen der UI implementiert und diese gegebenenfalls mit den bereits existierenden Funktionen der Evolutionsanimation verbunden werden. Eine Ausnahme hiervon war das Rückwärtsabspielen der Evolutionsanimation, dieses musste in der Evolutionsanimation neu implementiert werden.

4.3.1.2 Hervorheben der neuen, gelöschten und veränderten Knoten

In der Evolution war bisher nicht erkennbar, welche Objekte neu erstellt, verändert oder gelöscht wurden. Gefragt war eine Art, auf welche diese Änderungen sichtbar sind. Hierzu wurden die sogenannten „Power-Beams“ eingefügt. Bei den Power-Beams handelt es sich um lange,

runde Lichtstrahlen. Diese Power-Beams scheinen über jedem Objekt in einer Farbe, welche dem Status des Knotens in der aktuellen Revision entspricht.

Diese Farben sind beliebig einstellbar, die Standardwerte wurden folgendermaßen definiert:

- *Hellgrün*: Objekt wurde in der aktuellen Revision neu erstellt.
- *Cyan*: Objekt wurde in der aktuellen Revision verändert.
- *Rot*: Objekt wurde in der aktuellen Revision gelöscht. Hierbei erscheint der Power-Beam da, wo sich zuvor der gelöschte Knoten im Graphen befand.

Diese Farben werden außerdem von den Metric Charts verwendet, wobei jeder Knoten-Name die gleiche Farbe wie sein Knoten im Graph hat (siehe Sektion 4.3.2.4).

Da es sich bei diesen Power-Beams um ein „kosmetisches“ Feature handelt, wurden diese so implementiert, dass so viele ihrer Einstellungen wie möglich veränderbar sind. So kann nicht nur die Farbe der Beams, sondern auch die Höhe und Breite verstellt werden. Diese Veränderungen müssen im Unity-Inspektor geschehen. Der Benutzer kann sich somit entscheiden, ob er längere oder eher kürzere Strahlen haben möchte.

Um die Implementierung der Power-Beams umzusetzen, musste zuerst das Modell eines solchen Beams erstellt werden. Nach der Implementierung wurde dieser weiter dekoriert und dann an die eigentlichen Knoten des Graphen gesetzt, um einerseits die Beams an die korrekten Stellen zu platzieren und andererseits damit die Größe der Beams angemessen ist. Als dieses abgeschlossen war, wurde der komplexeste Bereich dieses Issues angegangen — die Revisionskontrolle. Das Hauptproblem hierbei war, wie eine Revisionsveränderung auf Code-Ebene erkannt werden konnte, um dann die korrekten Aktionen auszuführen. So wurde vorerst einfach pro Frame überprüft, ob es im Graphen noch die gleichen Objekte gibt, welches einen immensen Performanzverlust verursachte. Dennoch wurde dieses vorerst für den Rest der Implementierung verwendet. Um zu ermitteln, welche Knoten sich wie verändert haben, wurden an den Stellen, wo die Knotenbewegungen animiert werden, zusätzliche Funktionalitäten eingebaut. Das Performanz-Problem wurde dann gelöst, indem die zuvor gennante Revisions-Änderungsfunktionalität migriert wurde und nun mit den anderen Animationen des Evolutions-Graphen gemeinsam ausgeführt wird. Zuletzt wurden noch Post-Processing Effekte[17] eingefügt, damit die Power-Beams ein optisch ansprechendes Leuchten bekommen.

[17] Effekte, welche dafür sorgen, dass die Strahlen leuchten.

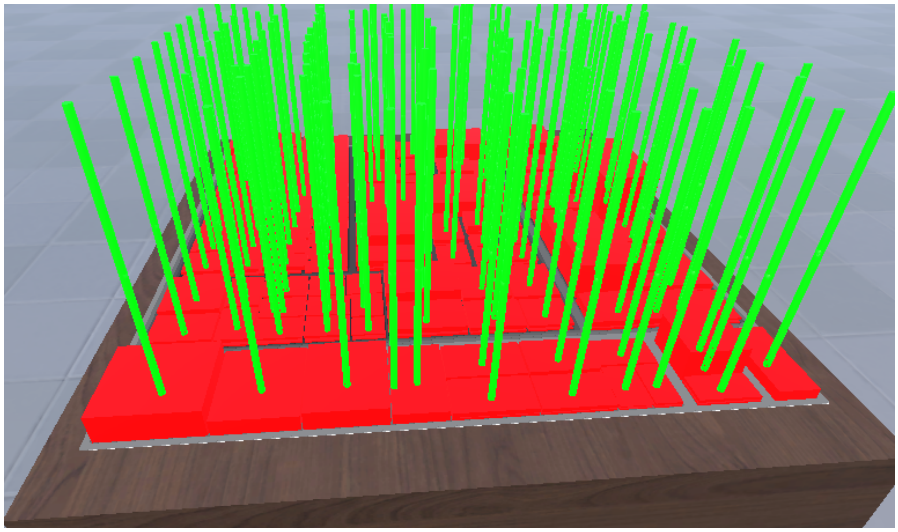

Abbildung 26: Die Power-Beams

4.3.1.3 Integration der Kantenanimation Im Evolution Renderer war der ursprüngliche Zustand, dass sich bei jeder Iteration die Knoten zu ihrer neuen Position bewegen und abschließend die Kanten entsprechend der neuen Positionen berechnet und angezeigt werden. Dieser Zustand sollte dahingehend erweitert werden, dass die Kanten ebenfalls mittels einer Animation zu ihrer neuen Position verschoben werden. Um dieses Verhalten zu erreichen wurden verschiedene Möglichkeiten getestet, jedoch ergab sich recht schnell, dass hierbei in den meisten Fällen ein zu großer Rechenaufwand notwendig gewesen wäre, um eine flüssige Animation zu gewährleisten.

Naiver Ansatz Zunächst wurde dafür gesorgt, dass bei jeder Bewegung eines Knotens alle Kanten neu berechnet werden. Dieses Verhalten funktionierte gut bei kleineren Graphen mit ca. 5–10 Kanten, jedoch wurde schnell ersichtlich, dass bei großen Graphen der Aufwand für die Berechnung zu groß sein würde. Verschlimmert wurde dieser Zustand durch die Implementierung der Mesh-Collider für die Kanten, die dafür genutzt wurden, um Kanten im Graphen auswählbar zu machen (siehe Sektion 4.6.3).

Lösung Um die Performance zu verbessern war die neue Idee, dass man die Kanten bei der Animation der Knoten unbewegt lässt und erst danach einmal für alle Kanten die neuen Positionen berechnet, um diese im Anschluss mittels linearer Interpolation[18] zu verschieben. Das eigentliche Verschieben stellte keine große Herausforderung dar, da alle Kanten über einen Line-Renderer angezeigt wurden, der eine Kante als eine Folge von Punkten realisiert, für den alle Positionen einfach manipuliert werden konnten. Das Hauptproblem war die Anzahl der Punkte auf den Kanten. Da Kanten unterschiedlich

[18] Mit linearer Interpolation ist in unserem Fall das Verschieben eines Punktes an eine neue Position über eine bestimmte Zeit gemeint.

lang sein können, werden sie durch unterschiedlich viele Punkte definiert. Wenn jedoch eine Kante vor der Neuberechnung mehr oder weniger Punkte im Vergleich zu der alten Version hatte, dann konnte die lineare Interpolation nicht funktionieren, da nicht alle Punkte eins zu eins aufeinander abbilden. Dementsprechend musste dafür gesorgt werden, dass Kantenpaare (alte und neue Kante) immer gleich viele Punkte hatten. Glücklicherweise wurden Kanten mittels TinySpline[19] [SK21] als Bézierkurve berechnet. Dadurch war es möglich, anzugeben, wie viele Punkte auf einer Kante liegen sollen. Damit keine Kante durch zu viele (höherer Rechenaufwand) oder zu wenige (eckiges Aussehen) Punkte definiert wird, wurden diese auf der alten und neuen Kante gezählt und daraus dann ein Mittelwert gebildet. Nachdem dies gelöst war konnten die Kanten bei jeder Iteration verschoben werden.

Nachdem die grundsätzliche Implementierung der Kanten nun abgeschlossen war, sollte das zeitgleiche Verschieben der Knoten und Kanten realisiert werden. Dabei entstand relativ schnell ein Problem, das daraus resultierte, dass die Berechnung der Kanten in SEE so implementiert ist, dass Kanten basierend auf den tatsächlichen Positionen der GameObjects der Knoten berechnet werden. Da zu Beginn der Animation jedoch alle Knoten noch an ihrer Ursprungsposition sind, können die Kanten noch nicht berechnet werden. Um dieses Problem zu lösen, wurde die Knotenanimation so verändert, dass die eigentlichen Knoten zu Beginn der Animation dupliziert werden und danach unsichtbar an ihre Zielposition gesetzt werden. Daraufhin werden die neuen Positionen der Kanten für die verschobenen Knoten berechnet. Wenn diese Berechnung abschlossen ist, werden die Kanten zusammen mit den Duplikaten der Knoten zu ihren neuen Positionen verschoben. Abschließend werden die duplizierten Knoten wieder gelöscht und die echten Knoten wieder sichtbar gemacht.

Probleme Bei der Realisierung entstand jedoch das Problem, dass es beim Wechsel der Revisionen des Graphen zu erheblichen Performance-Problemen kam. Dieses Problem wurde in der Folge versucht zu beheben (siehe Sektion 4.3.1.4).

4.3.1.4 Verbesserung der Kantenanimation

Aus der Animation der Kanten (siehe Sektion 4.3.1.3) ergaben sich diverse Performance-Probleme, die in diesem Ticket mittels diverser Maßnahmen angegangen worden sind.

Dynamische Reduktion der Punkte auf der Kante Es wird ermittelt, wie lange jeder Frame des Spiels gebraucht hat, um gerendert zu werden. Sollte dabei entdeckt werden, dass über einen längeren Zeitraum weniger als 30 Bilder pro Sekunde erreicht wurden, dann wird die Anzahl aller Punkte auf den Kanten um die Hälfte reduziert. Um dies zu realisieren, wurde die in der Animation der Kanten (siehe Sektion 4.3.1.3) angesprochene Punkteanzahl pro Kante dahingehend verändert, dass immer nur eine gerade Anzahl an Punkten in Frage kommen.

[19] *Spline Library for a Multitude of Programming Languages* — Marcel Steinbeck (https://msteinbeck.github.io/tinyspline/, letzter Abruf: 26.05.2020)

Bewertung der letzten Animation Bei jeder Animation der Kanten wird die Performance bewertet und fließt in einen Performance-Score ein. Sollte die Bildrate über 60 Bilder pro Sekunde liegen, dann werden bei der nächsten Animation mehr Punkte auf jeder Kante liegen, um das Aussehen der Kanten zu verbessern. Sollten weniger als 30 Bilder pro Sekunde bei der Animation vorherrschen, wird die Anzahl der Punkte bei der nächsten Animation verringert.

Ignorieren der Kanten, die nicht für die Berechnung nötig sind Beim Verschieben der Knoten wird ermittelt, ob sich der Knoten im Vergleich zur vorherigen Version des Graphen tatsächlich bewegt hat. Sollte dies nicht der Fall sein, dann wird dieser Knoten als vernachlässigbar gespeichert. Sobald alle Knoten untersucht wurden und die Kantenanimation beginnt, werden nur die Kanten berechnet, die mindestens einen Knoten verbinden, der bewegt wurde.

Ergebnis Diese Änderungen verbesserten zwar die Bildrate während der Animation, jedoch konnte das Problem des kurzzeitigen Einfrierens des Programms bei größeren Graphen nicht vollständig gelöst werden. Grund dafür ist hauptsächlich die enorme Anzahl von Punkten, die nach wie vor verschoben werden muss.

4.3.1.5 Ersetzen und Kapseln der Animations-Bibliothek „iTween“

Ursprünglich wurde für die Animationen in Unity *iTween*[20] als Framework benutzt. Da dieses Tool allerdings etwas veraltet war und es möglicherweise Bibliotheken gibt, die eine bessere Performance bieten, entwickelte sich der Gedanke, iTween zu ersetzen.

Nachdem wir uns verschiedene Alternativen (*HoTween*[21], *GoKit*[22]) angeguckt und diese verglichen haben, entschieden wir uns dazu, *DoTween*[23] als neue Bibliothek in SEE einzubauen. Primär fiel die Entscheidung auf DoTween, da dieses Framework permanent aktualisiert wird und eine gute Performance, sowie neue Möglichkeiten der Animation in Aussicht stellte. So ist es mit DoTween nun z. B. möglich über sogenannte Sequences mehrere Tweens auf einmal anzusprechen und zu steuern, oder auch mehrere Befehle zu buffern. Die Animationen unterscheiden sich dabei grafisch allerdings nicht großartig von den bereits vorher benutzten Animationen in iTween. Außerdem war die Bibliothek verhältnismäßig einfach zu integrieren, da sie sehr gut dokumentiert ist.
Vorher waren die Aufrufe von iTween direkt in den Quellcode eingebaut, was wir bei der Umstellung auf DoTween änderten. So erfolgten die Aufrufe der Animationsbefehle nun über eine gekapselte Klasse, wodurch es in Zukunft einfacher ist, das Framework zu ersetzen, falls dies nochmal nötig werden sollte. Ebenfalls konnte man Methoden nun gekapselt bearbeiten, ohne dass die Aufrufe der Befehle großartig geändert werden müssen.

[20] http://www.pixelplacement.com/itween/index.php (letzter Abruf: 30.05.2021)
[21] http://hotween.demigiant.com (letzter Abruf: 30.05.2021)
[22] https://github.com/prime31/GoKit (letzter Abruf: 30.05.2021)
[23] http://dotween.demigiant.com/ (letzter Abruf: 30.05.2021)

4.3.2 Neue Funktionen

4.3.2.1 Importieren von zusätzlichen Metriken als CSV-Datei Um eine Code-City zu erstellen, wird vor jeder Szene ein eigener Graph errechnet, der die jeweilige Software, d.h. sowohl die Softwarearchitektur, als auch die Implementierung selbst, mithilfe von Knoten und Kanten darstellt. Die Knoten und Kanten haben spezielle Metriken, wie unter anderem z. B. Lines of Code. Diese Metriken werden für jeden Graphen in sogenannten GXL-Dateien[24] gespeichert. Jede GXL beinhaltet also die notwendigen Daten des zu erstellenden Graphen der Software inklusive der zugehörigen Metriken. Allerdings kann es unter Umständen vorkommen, dass eine CSV-Datei[25] darüber hinaus noch zusätzliche Metriken für die Knoten des Graphen beinhaltet. Dafür wurde auf Quelltextebene eine Abbildung der jeweiligen CSV, falls diese vorhanden ist, auf die ihr zugehörige GXL implementiert und die Metriken, die in der CSV gespeichert sind, ausgelesen. Dazu wurde in dem jeweiligen Ordnerpfad, in welchem die GXL gespeichert ist, gesucht und festgestellt, ob für den exakten Namen der GXL Datei auch analog eine jeweilige CSV existiert. Falls dies der Fall sein sollte, werden sämtliche Metriken ausgelesen. Dafür wurde die Methode, die für das Auslesen der GXL zuständig ist, um die der CSV erweitert. Somit wurden nun diese zusätzlichen Metriken ebenfalls bei der Berechnung des Graphen miteinbezogen und auch z. B. bei der Animation der Revisionen berücksichtigt.

4.3.2.2 Auswahl der darzustellenden Knoten Bevor der Nutzer eine Evolution animieren lässt, sollte er die Möglichkeit haben, die verschiedenen Knotentypen im Editor entweder zu selektieren oder gegebenenfalls zu deselektieren, falls er spezielle Knotentypen in den Graphen der Evolution nicht darstellen möchte. Unter Umständen kann der Nutzer beabsichtigen, nicht sämtliche Knotentypen in die Revisionen der Evolution einfließen zu lassen, sondern lediglich eine kleine Auswahl derer, um sich z. B. einen fokussierten Blickwinkel auf wesentliche Inhalte bei der Analyse zu verschaffen.

Zuvor war es dem Nutzer nicht möglich, bestimmte von ihm ausgewählte Knotentypen für die Evolution nicht zu berücksichtigen, lediglich die Architekturprüfung bot diese Option. Dafür wurde bisher im Hintergrund, bevor der Nutzer die Möglichkeit bekommt, bestimmte Knotentypen auszuwählen, der erste Graph einer Evolution analysiert und standardmäßig wurden alle vorhandenen Knotentypen als selektiert betrachtet. Als Ergebnis hiervon wurden folglich auch alle Knotentypen in der Evolution abgebildet.

Im Zuge dessen wurde die Programmführung für den Nutzer etwas angepasst, sodass er nach dem Laden des Graphen, allerdings vor dem eigentlichen Zeichnen, die verschiedenen Knotentypen in Auswahlkästchen angezeigt bekommt. Die Programmführung

[24] Eine GXL-Datei, folgend GXL bezeichnet, steht für *Graph Exchange Language Format* und beinhaltet alle nötigen Informationen zum Austausch von Graphen zwischen Programmen. Siehe https://userpages.uni-koblenz.de/~ist/GXL/index.php (letzter Abruf: 31.05.2021)

[25] Eine CSV-Datei, folgend CSV bezeichnet, steht für *Comma Seperated Values* und ist ein systemübergreifendes, aber nicht standardisiertes Format, das unkomplizierten Austausch oder auch Sicherung von Daten ermöglicht. Im Fall von SEE wurde es z.B. zur Persistierung von Metriken genutzt. Siehe https://datatracker.ietf.org/doc/html/rfc4180 (letzter Abruf: 31.05.2021)

bezieht sich in dem Kontext auf den Unity Editor. Falls der Nutzer nun Knotentypen deselektiert hat, wurden sie als Konsequenz aus der Datenstruktur entfernt, die vor der eigentlichen Szene bzw. der Evolution zur Graphenberechnung genutzt wird. Somit werden die Knotentypen anschließend bei weiteren Revisionen der Evolution nicht mehr berücksichtigt. Dies ist nun sowohl für die Architekturprüfung als auch die Evolution möglich.

4.3.2.3 Speichern der Auswahl der darzustellenden Knoten

Aufbauend auf den vorherigen Arbeiten, welche die Auswahl der Knotentypen innerhalb der Evolution behandelten, resultierte die Annahme, der Nutzer könnte genau diese Einstellungen bei jedem beliebigen Starten von SEE übernehmen wollen. Dem Nutzer sollte nun also die Möglichkeit gegeben werden, seine persönliche Auswahl, d.h. ob z. B. ein bestimmter Knotentyp selektiert oder auch deselektiert wurde, in Form einer JSON-Datei zu exportieren, sodass entweder der Nutzer selbst oder aber auch andere Nutzer in der Lage sein können, diese Einstellungen vor dem Laden zu importieren, anstatt sie erneut auswählen zu müssen. Das JSON-Format wurde konzeptionell ausgewählt, da es sich um ein von Menschen lesbares Format handelt.

Wir entschlossen uns dafür, es mit Hilfe der `JsonUtility`[26]-Klasse, einer Klasse für die JSON-Serialisierung von Unity Objekten, zu lösen, sodass zuvor der Graph berechnet wird und genau diese Auswahl in Form einer JSON-Datei exportiert wird. Dem Nutzer wurde dafür die Möglichkeit gegeben, diese Datei in den von ihm zuvor ausgewählten Ordner zu exportieren.

Eins der wesentlichen Probleme beim Lösen der Aufgabe war, dass der Parser von `JsonUtility`, den wir für das Einlesen der Daten für die JSON-Datei genutzt haben, allerdings die gesamten Daten des Graphen eingelesen hat und nicht nur jene, die nötig gewesen wären. Ein Problem, dass daraus hätte resultieren können, wäre unter anderem das Überschreiben und eventuelle Konflikte im Graphen durch die zu umfangreiche Datei. Auch wären eventuell andere Einstellungen betroffen gewesen. Als Reaktion darauf wurde ein eigener Parser geschrieben, der dann nur die erwünschte Auswahl einliest und in Form einer JSON-Datei exportiert. Ferner wurde, bevor die eigentlichen Einstellungen importiert wurden, eine Prüfung der alten Einstellung vorgenommen. Falls z. B. bestimmte Knotentypen in der Zwischenzeit hinzugefügt wurden oder es bestimmte Knotentypen in der aktuellen Fassung nicht mehr geben sollte, wird der Nutzer darüber informiert. Dies ist sowohl für die Architekturprüfung als auch für die Evolution möglich.

4.3.2.4 Integration der „Metric Charts“

Die Metric-Charts, also die Tabellen, welche die Knoten des gezeigten Graphen anzeigen, sollten die Funktionalität bekommen, die neuen, veränderten und gelöschten Knoten deutlich darzustellen. Bisher enthielten diese nur die Knotennamen ohne jene Kennzeichnung über den Status der Knoten. Um diese Funktionalität einzubauen wurden Farben verwendet. Damit die Farben eindeutig mit dem gezeigten Graphen übereinstimmen, wurden die Farben aus Sektion 4.3.1.2 benutzt.

[26] https://docs.unity3d.com/ScriptReference/JsonUtility.html (letzter Abruf: 26.05.2021)

Die Metric-Charts zeigen also nun nicht nur normal die Knotennamen, sondern zeigen diese in gefärbter Form, um den Status der Knoten anzuzeigen. Als Standardfarben wurden dieselben Farben festgelegt, wie die in Sektion 4.3.1.2:

- *Grün* — Der Knoten wurde in dieser Revision neu erstellt
- *Cyan* — Der Knoten wurde in dieser Revision verändert
- *Rot* — Der Knoten wurde in dieser Revision gelöscht[27]

Die Komplexität der Implementierung wurde zunächst als gering eingeschätzt, erhöhte sich jedoch drastisch durch das Lazy-Loading der Metric Charts und dem Textmesh-Pro-Mouse-Hover-Features, da die Texte immer neu generiert wurden und somit ihre Properties nicht behielten. Die Implementierung könnte man somit folgendermaßen summieren:

1. Man bekomme von Sektion 4.3.1.2 die Ansicht darüber, welche Knoten sich wie verändert haben
2. Beim Öffnen des Graphen gebe man dann jedem Textobjekt die Farbe, die dieses haben soll, basierend auf dem Status des Knotens
3. Man füge die Textobjekte für die gelöschten Knoten ein und gebe diesen die richtige Farbe

Nun verbleiben zwei Probleme, welche noch gelöst werden müssen:

- Die Farben der Knoten werden beim Mouse-Hovern über die Textobjekte zurückgesetzt
- Beim Scrollen durch den Chart werden manche Textobjekte ausgeblendet und andere haben falsche Farben

Nachdem man also diese Probleme mit intensiver Arbeit und erheblichen Zeitaufwand gelöst hatte, indem man die Text-Farben vor dem Hovern zwischenspeicherte und diese nach dem Hovern wieder angewendet hat, funktionierte alles einwandfrei.

[27] Die gelöschten Knoten werden am Ende des Metric Charts angezeigt, da die anderen Knoten sortiert sind basierend auf ihrem Platz im gezeigtem Graphen.

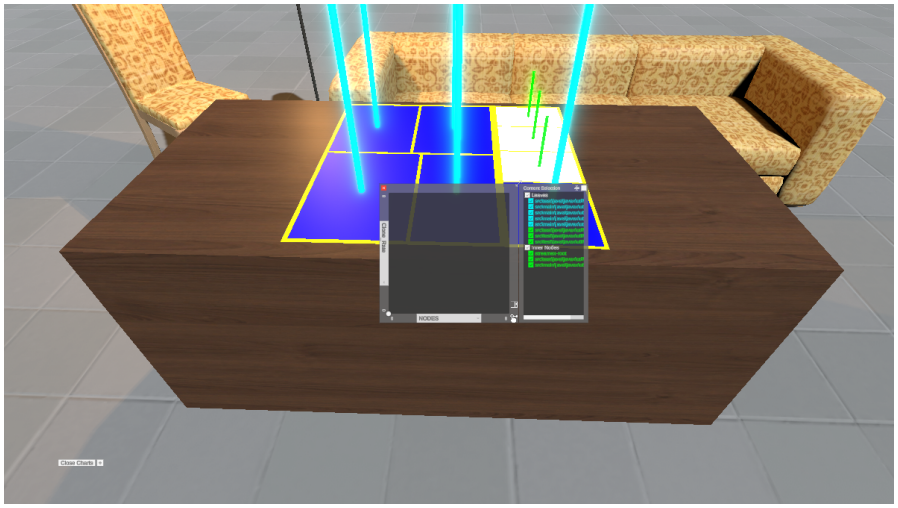

Abbildung 27: Die Metric Charts mit sichtbaren Änderungen

4.4 HoloLens

In diesem Teil wird es darum gehen, wie wir SEE auf die *HoloLens*[28] portiert haben. Die HoloLens ist dabei eine von Microsoft produzierte Datenbrille, mit welcher Informationen als Overlay über die Außenwelt angezeigt werden können[29]. Diese Art von digitalen Interaktionen mit der Umwelt nennt man oft auch AR, eine Abkürzung für *Augmented Reality*, auf Deutsch etwa „erweiterte Realität“. Das Ziel ist hier, die Code-Cities aus SEE durch die HoloLens in der echten Welt zu platzieren, z. B. auf den Schreibtisch des Nutzers und die Möglichkeit zu bieten, mit diesen Objekten zu interagieren, bspw. durch Greifen und Verschieben mit der Hand. Hierbei gibt es diverse zu bewältigende Herausforderungen, wie sehr unterschiedliche Anzeige- und Bedienkonzepte in der AR-Welt im Vergleich zu Desktop-Anwendungen oder die mobile CPU der HoloLens, welche im Vergleich zu z. B. Desktop-CPUs eine merkbar geringere Leistung mit sich bringt.

Als Beispiel für die Verwendung von Code-Cities auf der HoloLens wurde das Projekt *Identifying Usability Issues of Software Analytics Applications in Immersive Augmented Reality* des *Information System Institute* [Bau+20] von der Universität Leipzig betrachtet, in welchem die HoloLens 1 verwendet wurde, um Code-Cities darzustellen. So sollte der bestehende SEE-Code auf eine ähnliche Weise für die Mixed Reality[30] Datenbrille „HoloLens 2“ erweitert werden.

[28] Speziell haben wir SEE auf die *HoloLens 2* portiert. Der Einfachheit halber sagen wir in diesem Dokument aber oft auch einfach nur *HoloLens*, wenn wir uns auf die HoloLens 2 beziehen.

[29] Siehe https://www.microsoft.com/en-us/hololens/hardware (letzter Abruf: 24.05.2021)

[30] *Augmented Reality* (AR) und *Mixed Reality* (MR) sind für die Zwecke dieses Berichts als synonym zu verstehen, der letztere Begriff wird dabei bevorzugt von Microsoft verwendet.

4.4.1 Integration des MRTK

Das *Mixed Reality Toolkit*, kurz MRTK, ist ein Framework von Microsoft, welches Unterstützung von AR-Plattformen (wie die HoloLens) in Unity integriert. Für die initiale Planung haben wir uns in der MRTK-Community informiert, welches Vorgehen dort empfohlen wird, um bestehende Projekte mit dem MRTK zu erweitern und so für die HoloLens 2 verfügbar zu machen. Zusammengefasst wurde empfohlen, das Projekt in mehrere Module aufzuteilen, damit z. B. unnötige Abhängigkeiten auf nicht unterstützten Plattformen auch nicht kompiliert werden. Da dieses Vorgehen eine komplette Umstrukturierung der existierenden Projektmappe zur Folge hätte, wurde dieser Ansatz verworfen. Die von uns gewählte Alternative war stattdessen, das MRTK direkt in das Projekt zu integrieren.

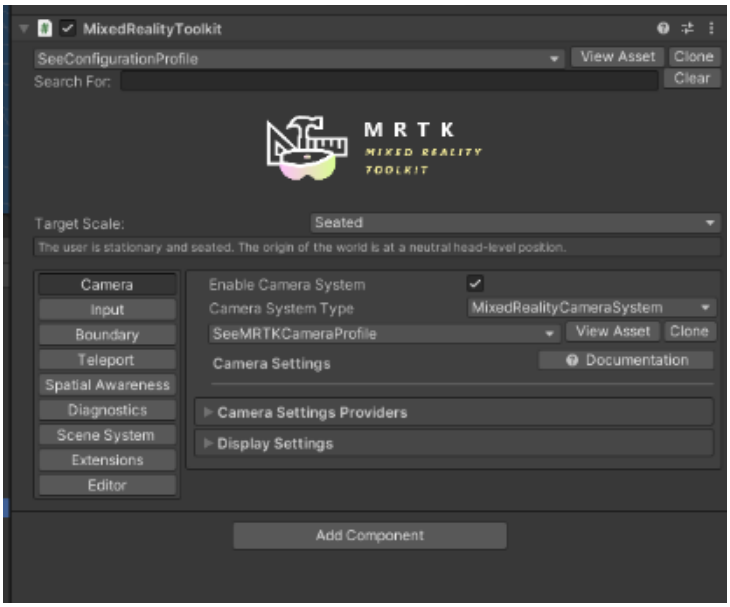

Abbildung 28: Das MRTK-GameObject in Unity.

Für die Implementierung musste das MRTK nach der von Microsoft zur Verfügung gestellten Anleitung[31] in das bestehende Projekt eingebunden werden. So muss für die Ausführung auf der HoloLens 2 zuerst Visual Studio mit dem UWP-SDK[32] sowie der HoloLens 2 Emulator neben der bestehenden Unity-Entwicklungsumgebung installiert werden. Mit dem Emulator lassen sich HoloLens-Anwendungen auf einem normalen Windows-Rechner testen, ohne eine echte HoloLens vor Ort zu haben. Der Emulator hat sich dabei in unseren Vergleichen zur echten Hardware identisch verhalten, wodurch wir auf der HoloLens auftretende Bugs, auch ohne das Gerät vor uns zu haben, testen konnten. Zum Zeitpunkt des ersten Einbindens wurden die MRTK-Abhängigkeiten per

[31] https://docs.microsoft.com/de-de/windows/mixed-reality/develop/install-the-tools?tabs=unity (letzter Abruf: 22.05.2021)

[32] Dabei ist UWP eine Abkürzung für *Universal Windows Platform*, eine Plattform für verschiedene Windows-Geräte, inkl. der HoloLens.

Unity-Plugin installiert, welches auf der entsprechenden Github-Seite[33] zur Verfügung gestellt wurde. Im weiteren Verlauf des Projekts wurde das *Mixed Reality Feature Tool for Unity* veröffentlicht und ermöglicht so mittlerweile auch das Updaten des bestehenden Projekts für neue MRTK-Versionen über eine einfach zu bedienende GUI.

Um das MRTK nun in SEE zu verwenden, wurde nach dem Import des Plugins in die Szene ein bereitgestelltes GameObject hinzugefügt (siehe Abbildung 28), welches für die AR-Integration verantwortlich ist. Im späteren Verlauf haben wir SEE so konfiguriert, dass dieses GameObject nur dann geladen wird, wenn die verwendete Plattform tatsächlich auch eine AR-Plattform ist, um das Verhalten anderer Plattformen nicht zu beeinflussen.

Nachdem der erste Start mit dem MRTK erfolgreich war, mussten noch alle für die HoloLens nicht notwendigen Elemente entfernt werden. Außerdem wurde für den Fall, dass mehrere Code-Cities beim Starten in die Szene geladen werden, diese Code-Cities in ein von Unity zur Verfügung gestelltes UI-Element namens `GridObjectCollection` eingebunden. Dieses UI-Element ermöglicht das Anordnen von GameObjects (in unserem Fall die Cities) in einem Gitter bzw. Raster. Durch die Vergabe des „Decoration"-Tags für alle Elemente, die nicht für die HoloLens relevant sind (z. B. dekorative Elemente wie das Sofa), wurde das `PlayerSettings`-Skript erweitert, sodass jene Elemente für die HoloLens ausgeblendet werden und nur noch die Code-Cities selbst sichtbar sind.

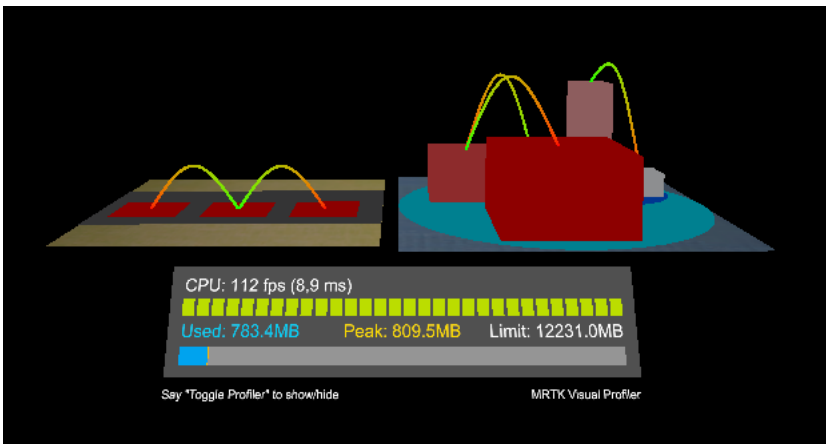

Abbildung 29: Die Darstellung der Code-Cities auf dem HoloLens-Emulator.

Wie das Ergebnis auf der HoloLens aussieht (jedoch mit schwarzem Hintergrund und nicht in der echten Welt) kann in Abbildung 29 betrachtet werden. Nachdem diese grundlegenden Funktionalitäten erfolgreich auf dem Emulator getestet wurden, kümmerten wir uns um die weiterführenden Usability-Verbesserungen, wie das Bewegen und Skalieren von Code-Cities im Raum oder das Anzeigen von Labels per Eyetracking.

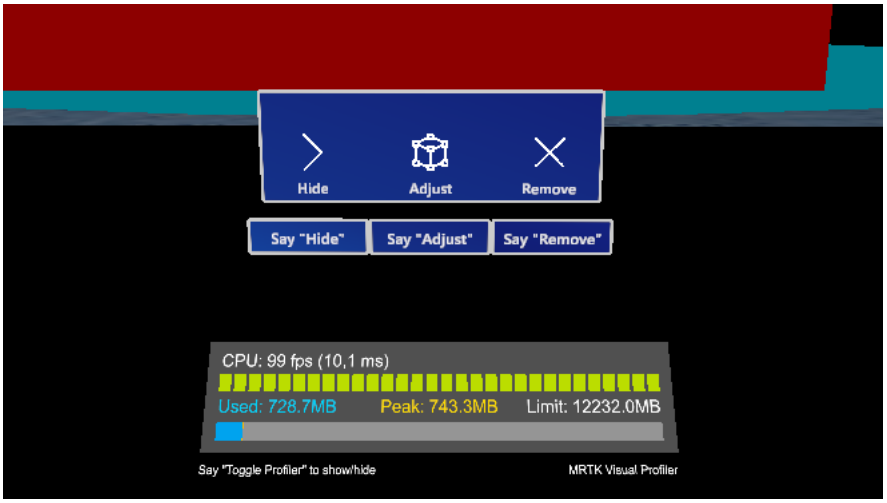

Abbildung 30: AppBar in blau unter der Code-City.

4.4.2 Positionierung und Skalierung in AR

Damit die Code-Cities gegriffen und im Raum beliebig platziert werden können und deren Größe verändert werden kann, musste der Szene eine sogenannte `AppBar`[34] hinzugefügt werden, die beim Interagieren an der aktuellen Code-City „haftet“. Die AppBar besteht aus einer Leiste an Buttons, welche Aktionen an der Code-City auslösen können, z. B. das Löschen dieser.

Die Code-Cities werden jeweils in von uns erstellte Objekte namens `CityContainer` hinzugefügt. Das ist notwendig, damit das MRTK die Höhe der Knoten korrekt erkennen kann. Diesem Container wurde dann eine vom MRTK bereitgestellte `BoundsControl`-Komponente hinzugefügt, mit der bei Auswahl der entsprechenden Option von der AppBar eine Box um die Code-City gezeichnet wird. Diese Box ermöglicht es, durch Greifen und Ziehen mit der Hand die Cities zu bewegen bzw. zu skalieren. Beide Elemente werden als Prefab geladen und den bestehenden Cities hinzugefügt. Die AppBar wechselt dabei die Position, wenn der Fokus auf eine andere Stadt fällt und ist somit immer an der Sichtfront der Stadt. In Abbildung 30 sieht man die AppBar, die BoundsControl um eine Code-City kann man in Abbildung 31 betrachten.

[33] https://github.com/microsoft/MixedRealityToolkit-Unity/releases (letztes Abrufdatum: 22.05.2021)

[34] https://docs.microsoft.com/en-us/windows/mixed-reality/design/app-bar-and-bounding-box (letzter Abruf: 22.05.2021)

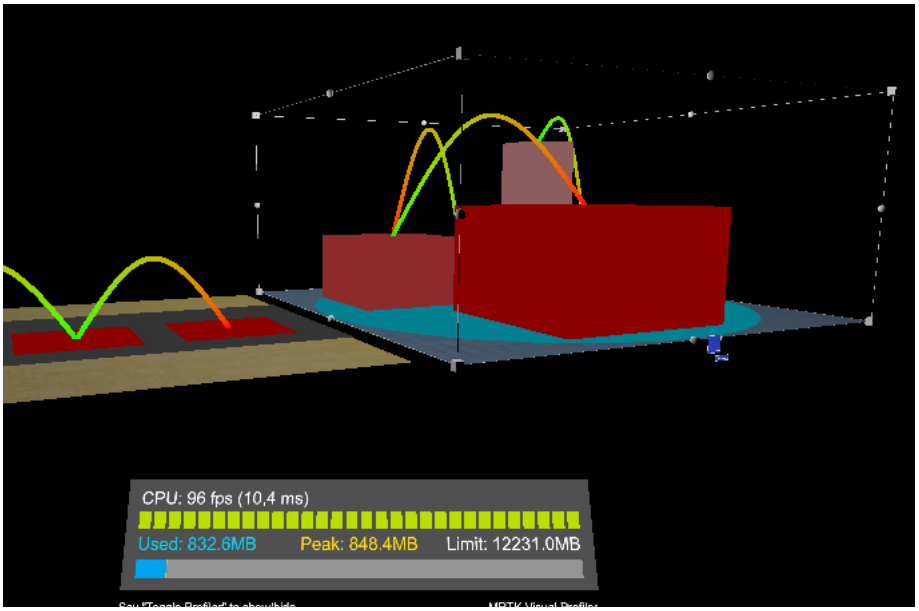

Abbildung 31: BoundsControl um die Code-City herum.

4.4.3 Label-Anzeige per Eyetracking

Ein weiteres Feature, welches wir für die AR-Plattform integrieren wollten, war das Anzeigen von Knotennamen nach Augenkontakt, d. h. wenn man einige Zeit auf einen Knoten schaut, wird über diesem Knoten dessen Name angezeigt. Die Integration selbst war durch Anwendung der entsprechenden Anleitung[35] gut durchführbar, das Eyetracking ließ sich unter Aktivierung einer speziellen Option testen, indem der Mauszeiger als Pseudo-Augenkontakt verwendet wurde.

Die Labels werden nach einer einstellbaren Zahl an Sekunden Augenkontakt eingeblendet und ebenso erst nach einer Verzögerung wieder ausgeblendet. Der Grund hierfür ist, dass durch Augenkontakt gesteuerte Interaktionen nicht sofort ausgelöst werden sollten, da dies sonst sehr desorientierend wirken kann. Bei den Labels wäre insbesondere folgender Ablauf ungünstig, der durch die Verzögerung beim Ein- und Ausblenden nicht mehr auftreten kann:

1. Der Nutzer schaut sich einen Knoten an.
2. Das Label erscheint über dem Knoten.
3. Der Nutzer schaut auf das erschienene Label, um es zu lesen.
4. Da der Nutzer jetzt nicht mehr auf den Knoten sondern auf das Label schaut, verschwindet das Label wieder. Der Nutzer kann es nicht lesen.

[35] https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/tutorials/mr-learning-base-08 (letzter Abruf am 22.05.2021)

Microsoft hat einige Empfehlungen für das Verwenden von Eyetracking auf der HoloLens[36], die die Bedienung für den Nutzer angenehmer machen sollen. Eine solche Empfehlung ist die Existenz von „Fallback Solutions“, also Alternativen, falls das Eyetracking aus verschiedenen Gründen[37] nicht verfügbar ist. Die „Fallback Solution“, die wir in SEE umgesetzt haben, ist in unserem Fall der Finger des Nutzers. Zeigt der Nutzer mit seinem Finger auf einen Knoten, wird mit dem gleichen Mechanismus der Name dieses Knotens eingeblendet. Somit haben HoloLens-Nutzer in SEE auch ohne Eyetracking die Möglichkeit, Namenslabels über den Knoten einblenden zu lassen.

4.4.4 Performance-Probleme auf der HoloLens

Mit Unity 2019 gab es beim Ausführen von SEE Probleme mit der Performance, die sich in sehr geringen Bildwiederholungsraten (≈ 10 FPS[38]) äußerten. Der Versuch, die Renderqualität zu ändern oder die Hologramm-Qualität zu verringern zeigte dabei leider keine Verbesserungen. Auch tiefergehende Analysen per Profiler konnten keine Abhilfe schaffen[39]. Durch das Update auf Unity 2020 haben die eben erwähnten Anpassungen, die unter Unity 2019 nicht gegriffen haben, jedoch Wirkung gezeigt und die Performanceprobleme konnten so gelöst werden. Im besten Fall konnten dadurch sogar Frameraten von bis zu 1000 FPS erreicht werden. Es bietet sich an, diese Frameraten zu begrenzen, um die Prozessorlast zu verringern, was bei mobilen Geräten wie der HoloLens auch die Batterieleistung erhöht.

4.4.5 Paketerstellung im Release- und Mastermodus

Mit Unity 2019 war es für dieses Projekt nicht möglich, im Release- oder Mastermodus eine HoloLens-Applikation zu kompilieren. Es konnte, auch nach weiterer Recherche und einem weiteren Thread im Unity Forum[40], keine funktionierende Lösung gefunden werden. Glücklicherweise konnte auch dieses Problem durch das Update auf die Unity Version 2020 ohne weiteres Zutun gelöst werden.

4.4.6 .NET-Abhängigkeitsprobleme

Beim Update von Unity 2019 auf Unity 2020 wurde das SteamVR-Plugin aktualisiert, was dazu geführt hat, dass die `Valve.Newtonsoft.Json.dll` als Assembly mit in das UWP-Projekt geladen wurde. Da die UWP .NET-Standard-2.0-konforme DLL-Dateien benötigt,

[36] https://docs.microsoft.com/en-us/windows/mixed-reality/design/eye-tracking (letzter Abruf am 22.05.2021)

[37] Z. B. der Nutzer konnte nicht kalibriert werden, die Berechtigung wurde der App nicht gegeben, der Nutzer hat Augenprobleme die das Tracking verhindern, usw.

[38] FPS steht für *Frames per Second* und beschreibt die Anzahl an gezeichneten Bildern pro Sekunde. Eine geringe FPS-Zahl äußert sich somit in einem ruckeligen Bild.

[39] Ein von uns erstellter Forenthread: https://forum.unity.com/threads/uwp-performance-issues.1047482 (zuletzt abgerufen am 22.05.2021)

[40] https://forum.unity.com/threads/uwp-app-cannot-be-build-in-realse-oder-master-mode.1047773/ (letzter Abruf am 22.05.2021)

die von Valve jedoch in einer zu alten .NET-Version kompiliert wurden, wurden als Workaround alle `Valve.Newtonsoft.Json` Abhängigkeiten durch `Newtonsoft.Json` ersetzt. Das hat dazu geführt, dass die Anwendung auf der HoloLens wieder funktionstüchtig war, jedoch resultierte dies in Problemen in der Desktop-Version. Nach einer kurzen Korrespondenz mit den Valve-Entwicklern auf GitHub[41] konnte noch ein Workaround erarbeitet werden, indem das Open Source Projekt `Newtonsoft.Json`[42] für die passende .NET-Version als `Valve.Newtonsoft.Json` kompiliert wurde. Das erforderte den Download des Quelltextes und das Anpassen davon für den `Valve.Newtonsoft.Json` Namespace im gesamten Projekt sowie das Auflösen damit einhergehender Abhängigkeitsfehler.

Nachdem die korrekte DLL-Datei kompiliert wurde, konnte diese in das Projekt eingebunden und für die UWP definiert werden, wodurch die korrekte .NET-Version auch für die UWP genutzt werden konnte. Für die Zukunft bedeutet diese Lösung, dass bei jedem Update die offizielle `Valve.Newtonsoft.Json.dll` durch unsere eigens kompilierte ersetzt werden muss, solange Valve keine neu kompilierte Version erstellt und einbindet. Für den Fall, dass zukünftig auch .NET-Standard 2.0 für UWP von SEE unterstützt wird, wurde eine Version für .NET-Standard 2.0 bereitgestellt, die (falls nötig) später integriert werden kann.

4.5 Multiplayer

Vor dem Hintergrund, dass die Anwendung SEE das kollaborative Arbeiten nicht nur ermöglichen, sondern zugleich auch verbessern soll, bestand die Notwendigkeit, dass sämtliche bereits implementierten Funktionen und Features, die auf den lokalen Endgeräten bereits funktionierten, auch im Netzwerk mit mehreren Teilnehmern in Echtzeit funktionieren sollten. Diese Notwendigkeit bezog sich also vor allem auf die neu integrierten Funktionen, die jeder Spieler ausführen konnte und die somit Änderungen an der lokalen Version von SEE verursachten, vorrangig alle Actions des Player-Menüs.

Grundlegend problematisch war anfangs die Synchronisierung und Serialisierung der jeweiligen Aktionen, respektive der Objekte im Netzwerk. Das Problem bestand darin, dass sich zunächst in die Besonderheiten von Unitys Netzwerkstruktur eingearbeitet werden musste. So handelt es sich dabei um eine Client-Server-Struktur, bei der ein Nutzer zunächst selber einen zentralen Server bereitstellt und sich mit diesem dann automatisch als Client verbindet. Weitere Clients können danach ebenfalls beitreten. So mussten alle Informationen des jeweiligen Clients immer zu diesem zentralen Server versendet werden und von dort an alle Clients verteilt werden.

Es konnte jedoch diesbezüglich dankenswerterweise auf die Vorarbeit eines SEE-Entwicklers zurückgegriffen und aufgebaut werden. Die Verbindung zwischen mehreren Clients konnte bereits hergestellt werden, es gab einen Voice-Chat — jeder Nutzer hatte eine spezifische ID[43] und war den anderen Nutzern in seiner Position und Darstellungs-

[41] https://github.com/ValveSoftware/steamvr_unity_plugin/issues/946 (letzter Abruf am 22.05.2021)

[42] https://github.com/JamesNK/Newtonsoft.Json (letzter Abruf am 22.05.2021)

[43] Diese ID ist momentan die IP-Adresse des Nutzers.

weise durch einen modellierten Kopf sichtbar (siehe Abbildung 32).

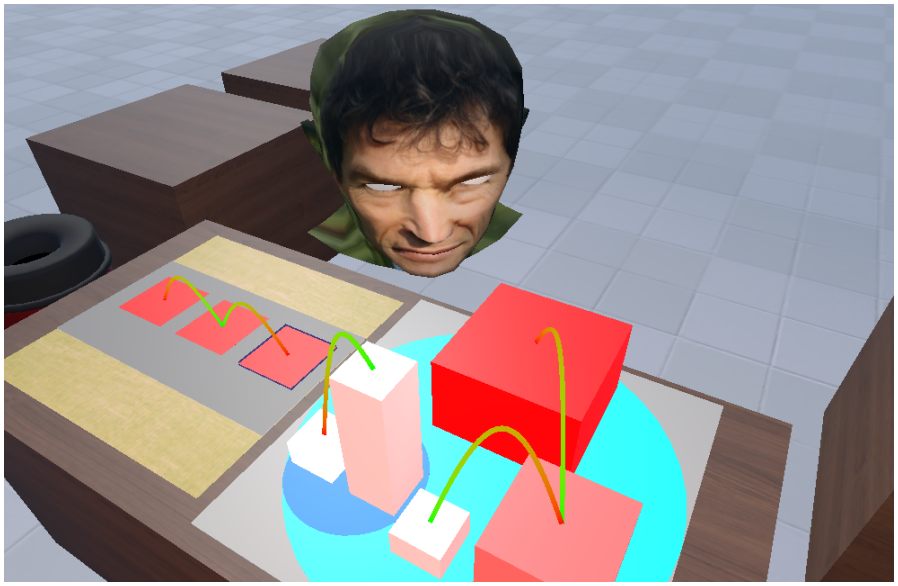

Abbildung 32: SEE in der Netzwerk-Nutzung

Des Weiteren existierte bereits ein implementiertes Framework eines Entwicklers, welches dem Nutzer grundlegend ermöglichte, Code auf jedem Client auszuführen. Dies hat allerdings die Einschränkung, dass man nur primitive Datentypen auf die jeweiligen Clients übertragen kann. Konzeptionell wurde dieses Problem nun in den Klassen des Netzwerks so gelöst, dass ein Objekt, welches eine Änderung erfahren hat, nicht selbst als Objekt, sondern lediglich dessen ID in Form eines Strings an alle Clients im Netzwerk propagiert wird. Weiterhin werden auch alle für die Action relevanten Daten in Form primitiver Datentypen übertragen, wie z. B. Koordinaten und die IDs anderer betroffener Objekte. Anschließend wird auf dem jeweiligen Endgerät das Objekt anhand dieser ID gesucht. Darauffolgend kann von jedem Endgerät respektive allen Clients im Netzwerk die jeweilige Aktion ausgeführt werden, sodass gewissermaßen in Echtzeit alle Clients zu jedem Zeitpunkt auf konsistenten Versionen der jeweiligen Szene arbeiten können. Somit kann es zu keinem Zeitpunkt zu Konflikten bzw. Asynchronitäten zwischen den Nutzern kommen. Der Terminus „gewissermaßen“ bedeutet in dem Kontext lediglich, dass es auf Grund der Latenzen im Netzwerk zu minimalen Verzögerungen kommen kann.

Auf Quelltextebene wurde die Ausführung über das Netzwerk so umgesetzt, dass es fortan für jede spezifische Action mindestens zwei Klassen geben sollte — die Action als solche und die analoge MultiplayerAction, wie z. B. `AddEdge` und `AddEdgeNet`. Die Varianten der Netzwerkskripte greifen dann in Teilen auf die nötigen Methoden der jeweiligen Actions und auf genutzte statische Klassen zu. Abschließend sind sämtliche Aktionen, die den Nutzern von SEE bereits auf den lokalen Endgeräten zur Verfügung standen, auch im Netzwerk verfügbar. Lediglich eine Integration der DrawAction (siehe

Sektion 4.2.7) wurde bislang nicht im Netzwerk umgesetzt. An der Aktion wird allerdings gearbeitet und auch dort ist eine sinnvolle Lösung für den Multiplayer absehbar.

4.6 Weitere Features

4.6.1 Action History

Aus der Entwicklung der Actions entstand nun ein weiteres Problem: Es könnte möglich sein, dass ein Nutzer einzelne Aktionen rückgängig machen möchte, wie z. B. den Löschvorgang eines Knotens, der fälschlicherweise gelöscht wurde, oder auch eine falsche Skalierung eines Knotens etc. Dasselbe gilt auch für fälschlicherweise rückgängig gemachte Aktionen, die erneut ausgeführt werden sollen.

Es muss sich also ein Konzept überlegt werden, bei dem alle atomaren Aktionen eines Nutzers in einer Historie abgelegt werden, in der immer auf das letzte Element zugegriffen wird. Durch eine rückgängig gemachte Aktion (im Folgenden als „Undo“ bezeichnet) muss diese Aktion aus der Historie entfernt werden und einer zweiten Historie hinzugefügt werden, die für die erneute Ausführung von Aktionen zuständig ist (im Folgenden als „Redo“ bezeichnet). Da es sich bei diesem Konzept um ein gängiges Problem in der Softwareentwicklung handelt, welches für viele Anwendungen wie z. B. Texteditoren oder IDEs umgesetzt werden muss, sollte hierfür zunächst ein Standard-Design umgesetzt werden, gemeinhin auch als Entwurfsmuster oder auch Design-Pattern bekannt: Das Entwurfsmuster `Composite`.

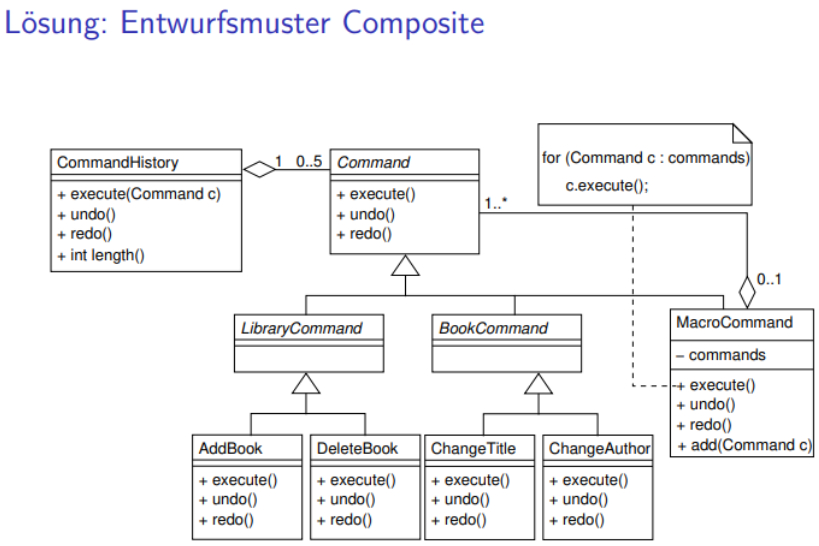

Abbildung 33: Entwurfsmuster: Composite (Aus den Folien von *Softwareprojekt 1* von Rainer Koschke)

Da dieses Entwurfsmuster jedoch nicht explizit für Unity entwickelt wurde, gab es

noch einige wichtige konzeptionelle Anpassungen. So sollten die Actions, welche vorher als von `MonoBehaviour` erbende Klasse erstellt wurden, nicht über das Keyword `new` zur Historie hinzugefügt werden (Vorgabe von Unity), weshalb diese nicht mehr von `MonoBehaviour` erben konnten.

Dies resultierte aber in dem weiteren Problem, dass nun keine Update-Methoden nach dem Konzept von Unity, also jeden Frame aufgerufene Methoden mehr in den Actions ausgeführt werden konnten, da dies von der Vererbung von `MonoBehaviours` abhängig ist. In diesen befand sich zuvor der Code für die Ausführung der Aktionen. Ein weiteres Problem ist dadurch gegeben, dass eine Action nicht bloß abgeschlossen oder nicht existent ist, sondern über das Menü gestartet wird. Zu diesem Zeitpunkt ist die Action weder nicht existent, noch abgeschlossen, sondern wird gerade beliebig lange ausgeführt. Auf dieser Grundlage wurde folgendes Konzept entwickelt:

Jede Action erhält ein Memento[44], welches bei Abschluss der Action mit den relevanten Werten für ein Undo bzw. Redo erzeugt wird. Die Actions selbst erben nicht mehr von der Klasse `MonoBehaviour`, sondern dem Pattern entsprechend von einer `AbstractAction`, die alle relevanten Methoden wie `Undo`, `Redo` und `Update` in einem Interface implementieren. Zusätzlich muss unterschieden werden, wann eine Action nicht existent, wann im Ausführungsstadium und wann abgeschlossen ist. Erst bei Abschluss soll diese der Historie hinzugefügt werden und automatisch eine neue Action derselben Art erstellt und ausgeführt werden. Nun musste für jede Action noch eine konkrete Implementierung für Undo und Redo implementiert werden.

Die Verwaltung der Reihenfolge und Aufrufe der abgeschlossenen Actions erfolgt in der Action-History. Hier existieren zwei Stacks, ein Undo-Stack, in welchem abgeschlossene Actions liegen und ein Redo-Stack, in welchem Undone-Actions liegen. Von hier werden neue Actions gestartet, Undo- und Redo-Aufrufe für die konkreten Actions aufgerufen und auch die Stack-Größen reguliert. So ist ein Redo für Undone-Actions nur dann möglich, wenn keine neue Action auf dem Undo-Stack abgeschlossen wurde.

Nach der konkreten Umsetzung dieses Konzeptes wurde das Issue abgeschlossen und konnte im Offlinemodus problemlos verwendet werden.

Im weiteren Verlauf wurde nun damit begonnen, dieses Feature ebenfalls multiplayerfähig zu machen. Allerdings wurde schnell klar, dass dahingehend einige neue Probleme bestehen.

Das Hauptproblem hierbei besteht darin, wie die Multiplayer-Umgebung von SEE gestaltet ist (siehe Sektion 4.5). Dadurch, dass jeder Spieler lokal Änderungen an der Umgebung durchführen kann, kann in einem lokalen Memento schon eine veraltete Version gespeichert sein, dadurch können Inkonsistenzen beim Interagieren mit der Code-City zwischen den Clients entstehen. Ein Beispiel für dieses Problem wäre folgender Anwendungsfall:

Ein Benutzer 1 erstellt einen Knoten. Ein weiterer Benutzer 2 bearbeitet eben diesen Knoten, indem er ihn umbenennt.

Die konzeptionelle Frage, die hieraus entsteht, ist, was passiert, wenn Benutzer 1 ein Undo aufruft und danach wiederum Benutzer 2 versucht, ebenfalls ein Undo auszuführen.

[44]Siehe z. B. https://sourcemaking.com/design_patterns/memento (letzter Abruf: 31.05.2021)

Das würde bedeuten, dass nach dem ersten Undo der Knoten gar nicht mehr existiert, sodass das zweite Undo in eine Exception laufen würde, oder die Konsistenz vom ersten Undo zerstört. Aufgrund dessen wurde ein neuer Ansatz benötigt, der die Historien aller Benutzer synchronisiert und auch mit Konflikten von Actions umgehen kann. Dazu begannen wir zunächst mit einer Literatur-Recherche. Hier gab es einige, teils sehr komplexe Ansätze, die allerdings unter den gegebenen Voraussetzungen nicht umsetzbar gewesen wären. Deswegen begannen wir auf Basis der Erkenntnisse aus dieser Literatur unser eigenes Konzept zu entwickeln.

Infolge dessen sollten nicht mehr zwei Historien, eine für die Undos und eine für die Redos, sondern nur noch eine Historie in Form eines Ringbuffers als Datenstruktur genutzt werden. Die Idee dabei war, dass man in der Liste nicht mehr nur die einzelnen Actions, sondern auch Meta-Informationen zu diesen findet. Diese Liste wird dann auf den Endgeräten aller Clients synchronisiert, sodass jede ausgeführte Action von jedem Benutzer gefunden werden können. Die Struktur des Ringbuffers bot sich in diesem Fall besonders an, da dieser eine wenig komplexe Integration einer maximalen Historien-Größe ermöglicht. So würden die ältesten Einträge des Buffers durch die neuesten ersetzt werden, sobald eine maximale Größe erreicht wurde.

Die Idee ist nun, dass jede ausgeführte Action bzw. ausgeführtes Undo und ausgeführtes Redo in diese Liste eingetragen werden. Möchte ein Nutzer eine Action rückgängig machen oder ein Redo ausführen, dann wird der Ringbuffer von dem Eintrag der Action auf die ein Undo ausgeführt werden soll bis zum neuesten Eintrag durchsucht. Wenn in einem der veränderten Objekte der neueren Aktionen das gleiche GameObject bearbeitet wurde, dann sollte entweder ein Merge der Aktionen ausgeführt werden, oder das betreffende Undo/Redo blockiert bzw. übersprungen werden. Ein Ringbuffer Eintrag setzte sich aus den folgenden Informationen zusammen:

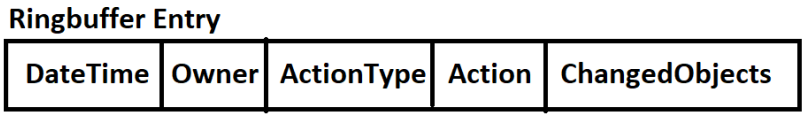

Abbildung 34: Die Struktur eines Ringbuffer-Eintrages vorher

Der eindeutige Schlüssel eines Eintrages sollte aus dem Zeitstempel der Ausführung einer Action und einer ID des jeweiligen Benutzers bestehen. Nach weiteren Überlegungen und Problemen mit der Eindeutigkeit dieses Keys[45] wurde dieser Eintrag durch einen anderen Key ersetzt — jede Action erhielt bei der Erstellung eine UUID[46].

[45] Die Unity-Funktion für Timestamps ist nur auf Sekunden genau, was bedeutet, zwei in der selben Sekunde ausgeführten Actions hätten denselben eindeutigen Schlüssel, was ihn nicht mehr eindeutig macht.

[46] *Universally Unique Identifier* — In der .NET-Umgebung auch als GUID bekannt, siehe https://docs.microsoft.com/de-de/dotnet/api/system.guid?view=net-5.0 (letzter Abruf: 20.05.2021)

Der ActionType beschreibt, ob es sich um eine Action handelt, oder um eine Undone-Action — also ein Undo. Eine Action, die also zuvor auf dem RedoStack gelegen hätte, befindet sich nun in dem Buffer mit dem ActionType UndoneAction. Ein Redo dieses Undos hingegen transformiert dieses wieder zurück in eine normale Action. Aufgrund erhöhter Komplexität der Umsetzung der Historie mit einem RingBuffer wurde dieser jedoch durch eine normale Liste ersetzt. Zusätzlich mussten wir noch feststellen, dass die Übertragung einer Action zum jetzigen Zeitpunkt nur eingeschränkt möglich ist, da nur primitive Datentypen übertragen werden können. Deswegen entwickelte sich ein neues Konzept: Es sollten nur noch Meta-Informationen zu den Actions in einer Historie synchronisiert werden, jedoch nicht mehr die eigentlichen Actions. Auf diese sollte dann jeweils lokal zugegriffen werden. Ein neuer Eintrag für die Historie wurde hieraus entwickelt:

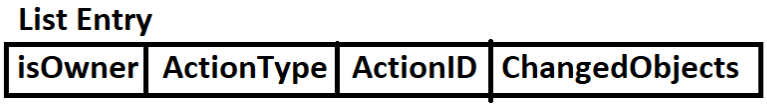

Abbildung 35: Die Struktur eines Listen-Eintrages nachher

Die ID eines Eintrages wurde nun durch die Action-ID dargestellt. Zusätzlich verzichten wir auf das Merging zwischen zwei konfligierenden Actions, da die Grundannahme getroffen wurde, dass es selten zu Konflikten kommt, da das Feature Undo vermutlich eher direkt nach einer versehentlich ausgeführten Aktion benutzt wird und eher selten eine Action nach längerer Zeit rückgängig gemacht wird. Diese Annahme muss jedoch in der laufenden Benutzung noch überprüft werden. Da allerdings wie zuvor angesprochen keine Synchronisierung der eigentlichen Actions in einer Liste stattfindet, benötigt es jeweils noch eine lokale Datenstruktur aller Benutzer, die die Verwaltung der eigens ausgeführten Aktionen übernimmt. Dazu haben wir uns erneut an dem Konzept der klassischen Undo- und Redo-Stacks bedient. Diese werden allerdings, wie zuvor erwähnt, nur lokal gehalten und sind abhängig von der ordnungsgemäßen Synchronisierung der globalen Historie. Sollte ein Undo nicht möglich sein, bekommt der Nutzer eine Fehlermeldung und es wird zur Action gesprungen, die in der Aufrufhistorie vor der konfligierenden Action liegt. In Abbildung 36 wird die konkrete Implementierung unserer globalen Action-History in Form eines UML-Diagramms abgebildet.

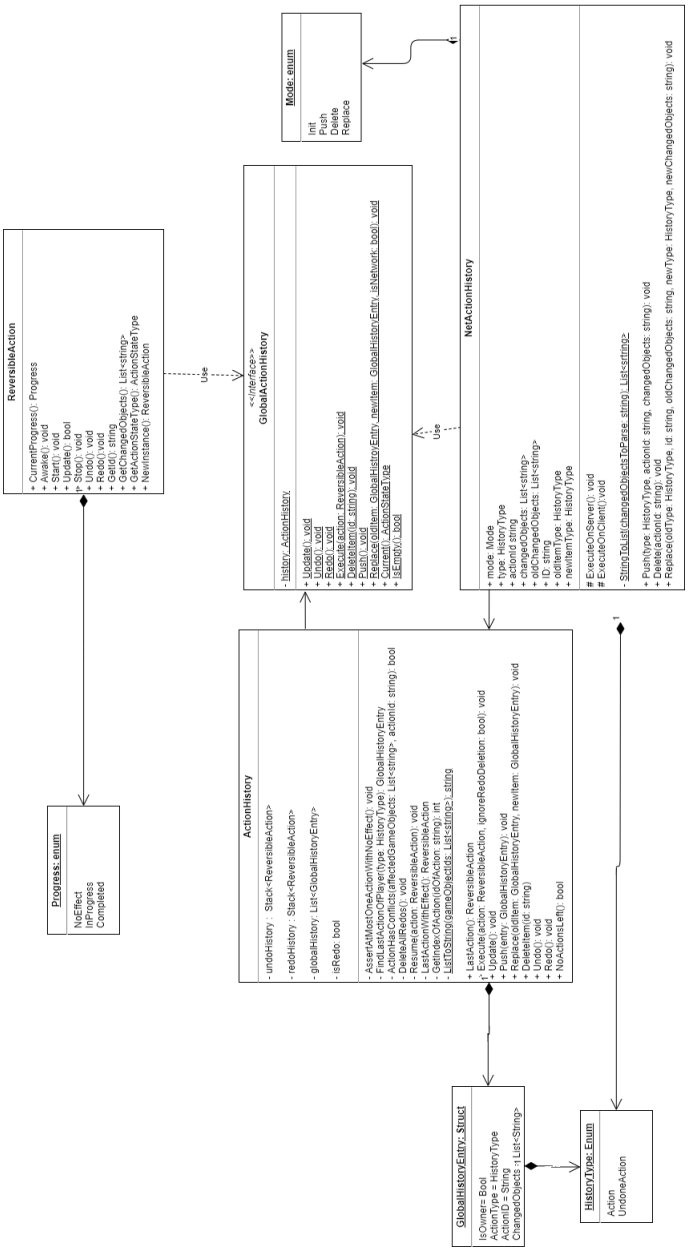

Abbildung 36: UML-Struktur der ActionHistory, um 90° gedreht

4.6.2 Szenen-Setup über Inspektor

In Unity wird das Konzept von *Szenen* verwendet. Eine Szene enthält dabei *GameObjects*, welche mit bestimmten Komponenten versehen werden, die das Verhalten beeinflussen. In SEE gab es mehrere solcher Szenen[47], z. B. die am häufigsten verwendete `MainScene`.

[47] Nach einem Refactoring im Anschluss an dieses Issue gibt es nun nur noch eine Szene.

Manually adding required GameObjects

Your scene must contain the following game objects:

- A Directional Light.
- Some kind of table with a component SEE.GO.Plane upon which the code cities are to be put. Any kind of game object representing this table can be used. The *Scale Factor* of the SEE.GO.Plane must be a factor to adjusting the table's scale units to Unity units. If the table is a Unity cube, the factor should be 1; but there are other kinds of objects for which one scale unit is not exactly on Unity unit. The table object must be tagged by *CullingPlane*.
- A VRPlayer as an instance of the player rig prefab by SteamVR (Assets/SteamVR/InteractionSystem/Core/Prefabs/Player.prefab). Attached to it must be an XRChartAction, XRRay, and XRPlayerMovement component (see below). The name of that game object must be "VRPlayer".
- A game object named "DesktopPlayer" with a Camera, Event System, and Standalone Input Module (comes along with the Event System). In addition to that the SEE components DesktopPlayerMovement and DesktopChartAction must be attached. The attribute *Focused Object* of the DesktopPlayerMovement must be a SEE.GO.Plane component of the table the code cities will be placed on.
- A game object named "PlayerSettings" with SEE's PlayerSetting component.
- A game object named "InControl" created by the menu "GameObject=>InControl=>Manager". This object is required to support gamepads and touchscreens with virtual gamepad controllers. It must have the following child game objects: "Touch Camera", "Touch Stick Left", "Touch Stick Right", and "Touch Button Control". These implement the virtual controllers for touchscreens. They can be created by "GameObject=>InControl=>Touch=>Manager" and then in the inspector when you click on the "InControl" object through the buttons "Create Button Control" and "Create Stick Control". The "Touch Stick Left" should have "Left Stick" as option "Target" and the "Touch Stick Right" should have "Right Stick" as option "Target". The "Touch Button Control" must have "Left Trigger" as option "Target". The targets are the actions defined in Unity's input manager (these were added by InControl)
- A game object named "ChartManager" tagged by "ChartManager" with a component that is an instance of class ChartManager and another component that is an instance of class Animator (the attribute Controller of this Animator must be the ChartManager component -- its sibling). This game object must have a child game object named "ChartsVR" that has a component of class ChartPositionVr attachted to it (the attribute Camera Transform must refer to the transform of the VR Camera). For your convenience, a prefab "ChartManager" exists that you may add to the scene.

If you want to (optional, purely for the aesthetics), you can select "Window=>Rendering=>Lighting Settings" to open the lighting settings. Click on the right dotted circle next to the field "Skybox Material" to select WispySkyboxMat.

Abbildung 37: Vorheriger Wiki-Artikel zum Einrichten von Szenen in SEE.

SEE trifft dabei Annahmen über die Szenen, diese müssen z. B. alle bestimmte GameObjects mit bestimmten Komponenten enthalten, ansonsten funktionieren einige Skripte nicht. Richtet man eine neue Szene ein, muss man diese auf eine bestimmte Weise (siehe Abbildung 37) mit GameObjects, Komponenten und Prefabs (vorgefertigte GameObjects) präparieren, hierzu gab es eine Seite in unserem projektinternen Wiki. Ähnlich kompliziert war es, eine neue Code-City zur Szene hinzuzufügen, da auch hier mehrere GameObjects vorbereitet und mit speziellen Komponenten versehen werden mussten.

Abbildung 38: Ausschnitt der Player Settings im Inspektor

Um dieses aufwändige Setup zu vereinfachen, wurden der *Player Settings*-Komponente zwei Interfaces hinzugefügt (siehe Abbildung 38):

1. Ein Interface, welches lediglich aus einem Button *Setup scene* besteht, welcher die aktuelle Szene mit den notwendigen GameObjects und Komponenten befüllt, sodass diese „spielbereit“ wird.

2. Ein Interface, welches unter Auswahl des City-Typs und Eintragung des Namens eine neue Code-City erstellt und der Szene hinzufügt.

Somit muss einer potentiellen neuen Szene nun nur noch die *Player Settings*-Komponente hinzugefügt werden. Durch diese kann dann alles oben beschriebene automatisch eingerichtet werden.

4.6.3 Selektierung einzelner Kanten

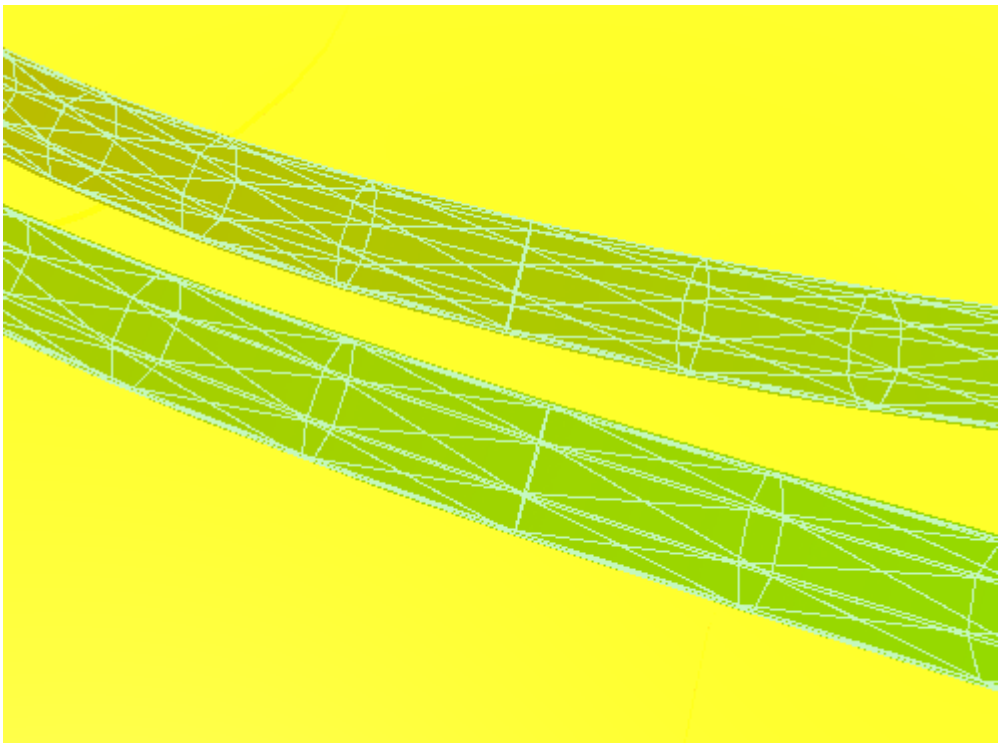

Abbildung 39: Ein Mesh um eine Kante herum.

Nachdem die Knoten einer Code-City bereits ausgewählt werden konnten, fehlte diese Funktion für die Kanten noch. Dies erwies sich aufgrund der spline-förmigen Kanten allerdings als schwieriger als gedacht. Es war zunächst nicht möglich, einen einfachen Collider um die Kanten zu legen, um sie mit der Maus selektierbar zu machen. Deshalb musste ein röhrenförmiges Mesh in Form der Spline berechnet werden, um

diese komplett nachzubilden. Dieses Mesh „liegt“ nun praktisch um die gesamte Kante herum, wodurch erkannt wird, wenn die Maus des Benutzers mit der Kante kollidiert (siehe Abbildung 39). Durch diese Kollisionsabfrage waren Kanten nun für den Benutzer einfach selektierbar.

4.6.4 Visualisierung der Komponenten eines Knotens

Bei der Bearbeitung der Actions (siehe Sektion 4.2) kam die Idee auf, dass zu einem späteren Zeitpunkt die Möglichkeit hinzugefügt werden soll, mehrere Objekte zu einem Objekt zusammenzufalten. Dabei soll erreicht werden, dass generell in den Städten eine bessere Übersicht herrscht, indem zusammenhänge Objekte zu einem großen Block zusammengefasst werden können.

Bei dieser Zusammenfaltung würde dann allerdings das Problem auftreten, dass in einem Objekt mehrere Objekte versteckt werden und die Übersicht über diese dann nicht mehr gewährleistet ist. Um dieser Problematik vorzubeugen sollte eingebaut werden, dass nach der Zusammenfaltung des Knotens an den Gebäudewänden des neuen Objekts grafisch dargestellt wird, was in ihm enthalten ist. Außerdem wurden verschiedene Dacharten für die Knoten erstellt, sodass unterschiedliche Typen von Objekten visuell unterschiedlich aussehen, wodurch diese direkt erkannt werden können (z. B. Klassen versus Interfaces in Java). Dies hat zwar in erster Linie nichts mit dem Zusammenfalten zu tun, trägt aber auch zu der Übersicht bei.

Die Dächer der Knoten werden nur dann erstellt, wenn diese keine zusammengefalteten Knoten sind. Die Dachart kann programmatisch gesetzt werden, je nachdem, welche Art von Objekt durch den Knoten visualisiert wird. Des Weiteren wurden für den Anfang drei verschiedene Dacharten mitgeliefert, es können aber auch beliebige weitere eingefügt werden. Die enthaltenen Dachtypen sind:

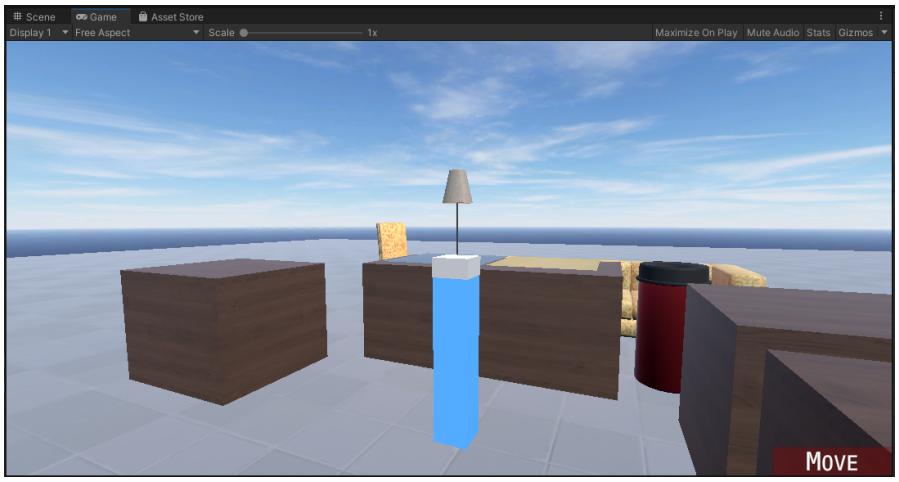

Abbildung 40: Ein rechteckiges Dach

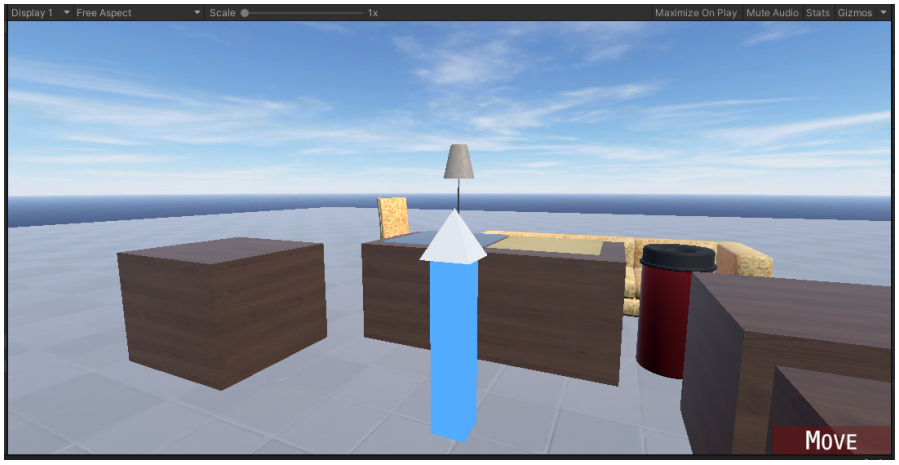

Abbildung 41: Ein pyramidenförmiges Dach

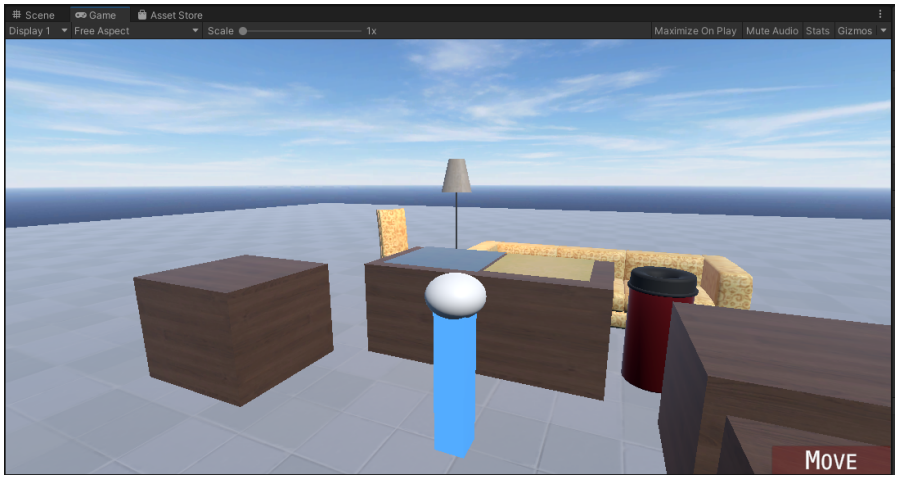

Abbildung 42: Ein kuppelförmiges Dach

Bei zusammengefalteten Knoten sollten die Seiten des Knotens die Höhen-Metrik der in diesem Knoten enthaltenen Kindknoten durch eine TreeMap visualisieren. Die Zusammenfaltung selbst wird, wie bereits erwähnt, erst in einem späteren Issue hinzugefügt. Auf dem Dach des Knotens soll jedoch eine TreeMap das Layout der Kind-Knoten visualisieren. Die Umsetzung war sehr komplex, da viele Koordinaten ausgerechnet werden mussten und die TreeMaps korrekt um die verschiedenen Achsen gedreht werden mussten. Dies konnte mithilfe von geometrischen Berechnungen gelöst werden. Die TreeMaps kalkulieren somit jeweils die Höhe und Tiefe eines Knotens und nutzen diese, um sich richtig zu skalieren. Die Koordinaten des Knotens werden dann verwendet, um die TreeMaps an die korrekte Position zu bringen, welche dann rotiert werden, um auf der Seite des Knotens zu liegen.

Abbildung 43: Ein mit TreeMaps dekorierter Knoten

Bei der TreeMap des Dachs war die Implementierung einfacher umzusetzen, da dieses kein besonderes Rotieren erforderte.

Des Weiteren wurden zwei weitere Visualisierungstypen eingebaut, welche programmatisch anstelle der TreeMaps verwendet werden können. Diese verwenden statt der TreeMaps einfache rechteckige Blöcke, um die Metriken der Kindknoten darzustellen. Ähnlich wie bei den TreeMaps zeigt hierbei der Graph auf dem Dach des Knotens eine Repräsentation der versteckten Knoten, während die Seiten des Blocks die Höhen-Metrik der versteckten Knoten visualisieren.

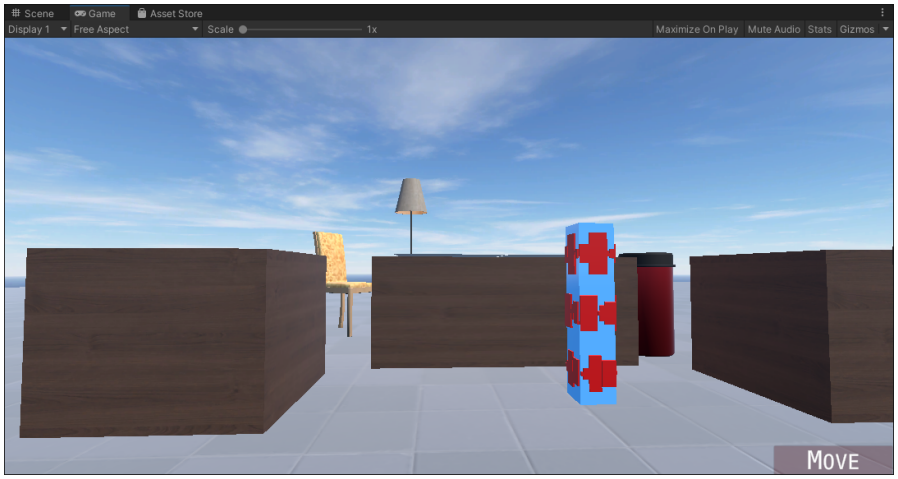

Abbildung 44: Ein dekorierter Knoten

Zudem wurde dann noch eine Testmethode hinzugefügt, um die jeweiligen Methoden zu überprüfen, ohne auf die noch fehlende Funktion warten zu müssen, welche die Zusammenfaltung des Knotens organisiert. Diese kann auch weiterhin verwendet werden, um weitere Knoten-Dekorationen zu testen, da diese erst benutzt wird wenn sie programmatisch ausgewählt wird.

5 Präsentation und Evaluation

5.1 Website

SEE soll in Zukunft Menschen bei der Softwareentwicklung unterstützen. Damit das allerdings auch realisierbar ist, benötigt das Projekt eine gewisse Reichweite. Eine solche startet in der heutigen Zeit grundsätzlich mit einer Onlinepräsenz, bzw. einer Website.

Als ersten Schritt einigten wir uns auf *WordPress*[48] als Content Management System. Zum Inhalt der Website erörterten wir mehrere Schwerpunkte. Diese waren Publications, Use Cases, Supported Hardware, About und ein Bereich, der den Entstehungsprozess von SEE abbildet. Diese ganzen Kategorien haben wir in unserer Gruppe genauer erörtert und präziser formuliert. Dabei einigten wir uns bei der Sprachwahl der Website auf Englisch und ein modernes, aber auch minimalistisches Design. Als Farbpalette wählten wir Blau und Weiß, gemischt mit einigen Grautönen. Weiterhin stellte ein Akzeptanzkriterium die zwingende Abwesenheit von Google Analytics und YouTube-Videos dar.

Ein letzter wichtiger Punkt ist die Datenschutzgrundverordnung (DSGVO). Es war von entscheidender Wichtigkeit in keine gesetzlichen Konflikte zu geraten. Dafür orientierten wir uns an der Datenschutzerklärung der Universität Bremen, da das Projekt im Namen der AG Softwaretechnik der Universität Bremen organisiert wird. Die daraus

[48] https://wordpress.com/de/ (letzter Abruf: 30.05.2021)

resultierende Website (https://see.uni-bremen.de/[49]) wird im Folgenden genauer erörtert.

Start

Der Header einer Website ist elementar für die Navigation und Struktur. Da in einem solchen die Struktur der Seite mittels einer Menüleiste implementiert ist, ist unsere Menüleiste der Struktur unserer Website nachempfunden. So gelangt man auf keine Unterseite beim Klicken eines Menüeintrages, sondern wird automatisch zum richtigen Teil des Onepagers gescrollt. Dadurch wirkt die Website moderner und dynamischer. Ebenfalls haben wir kleine, weiße Boxen genutzt, um schlagwortartig einen Einblick auf SEE zu geben.

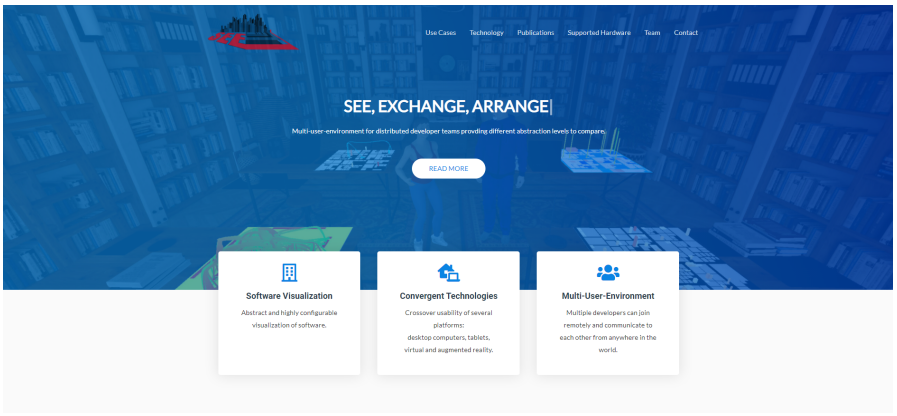

Abbildung 45: UseCases-Teil der Website

Use-Cases

Die Use Cases, also die Anwendungsfälle von SEE, benötigten eine Präsentation, mit welcher dem Besucher der Website ein kleiner und verständlicher Einblick in SEE gewährt wird. Die vier formulierten Use Cases sind: *Debugging*, *Evolution* (siehe 2.3.2), *Architekturvergleich* (siehe 4.2) und *Quality*.

Während diese Use Cases in einem anderen Teil dieser Dokumenation genauer abgefasst sind, geht es in diesem Teil lediglich um die Darstellung selbst. Dabei entschieden wir uns für eine Kachelansicht, in der alle vier Kategorien mit einem unscharfen Hintergrundbild und einer beschreibenden Caption versehen sind. Klickt man eine dieser vier Kacheln an, so öffnet sich ein Modal (ein Popup-Fenster auf der Website), in dem es möglich ist, ein kurzes Video von SEE in laufender Benutzung zu sehen.

[49] Letzter Abruf: 30.05.2021

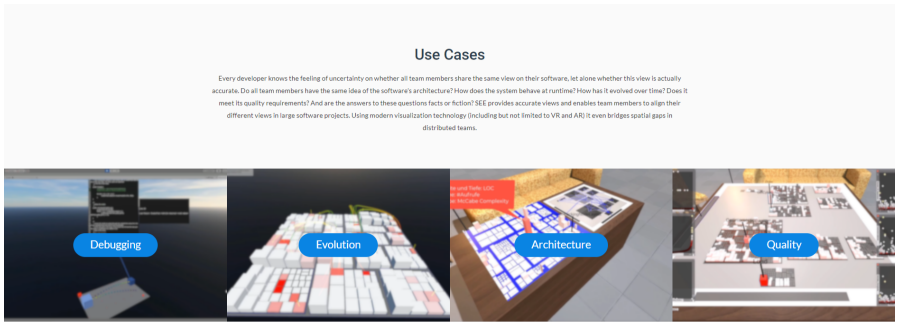

Abbildung 46: UseCases-Teil der Website

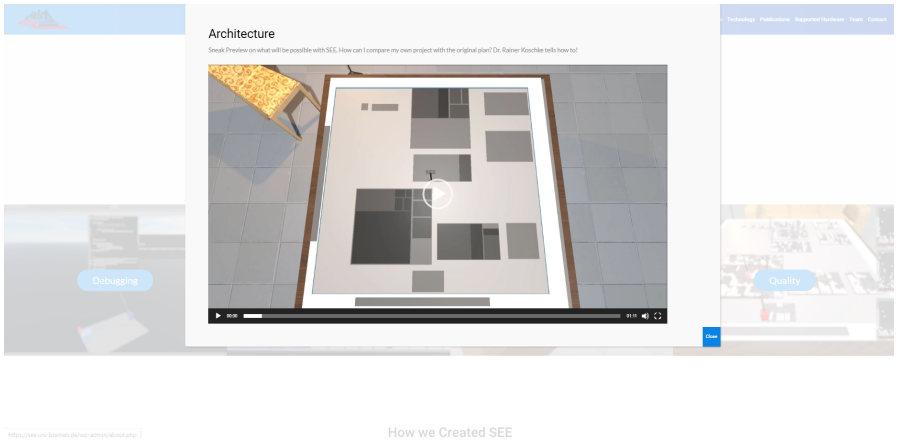

Abbildung 47: UseCases mit geöffnetem Modal

How we Created SEE

In dieser Rubrik galt es, die Fundamente von SEE (z. B. Engine, fremde Plugins, Technologien, etc.) auszuarbeiten und kurz zu erklären. Hier entschieden wir uns für eine Akkordeonansicht, bei der der Besucher selbst entweder eine Übersicht bekommt, oder bei mehr Interesse die Funktion des Aufklappens nutzen kann. Dort findet man genauere Informationen zu SEE.

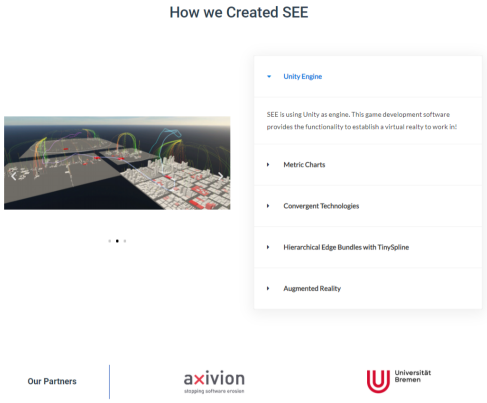

Abbildung 48: How we Created SEE

Publications

Beim Unterpunkt Publications ging es darum, Literatur, die während und rund um SEE entstanden ist (zum Beispiel Paper, Artikel, etc.), übersichtlich darzustellen. Dafür implementierten wir ein Plugin, mit dem es möglich ist, einen Zeitstrahl auf der Homepage abzubilden und sämtliche Informationen in bevorzugter Weise zu zeigen.

Abbildung 49: Publications-Teil der Website

Supported Hardware

Mit Supported Hardware ist eine Auflistung sämtlicher Hardware, mit der man SEE nutzen kann, gemeint. Bei dieser Auflistung ergaben sich erneut vier verschiedene Plattformen: *Augmented Reality*, *Virtual Reality*, *Touchbased Hardware* und *Desktop-PC*. Auch hierbei wird auf die Hintergründe dieser etablierten Schwerpunkte genauer eingegangen.

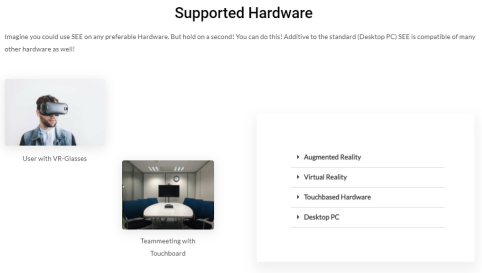

Abbildung 50: Supported Hardware

About

Danach sollten auch alle Teilnehmer des Projektes auf der Website eine Erwähnung finden. Dafür organisierten wir ein Teams-Meeting und nutzten die dortige Saal-Funktion zur Erstellung eines Gruppenbildes. Neben der Einblendung dieses Bildes fügten wir eine Liste der Teilnehmer hinzu.

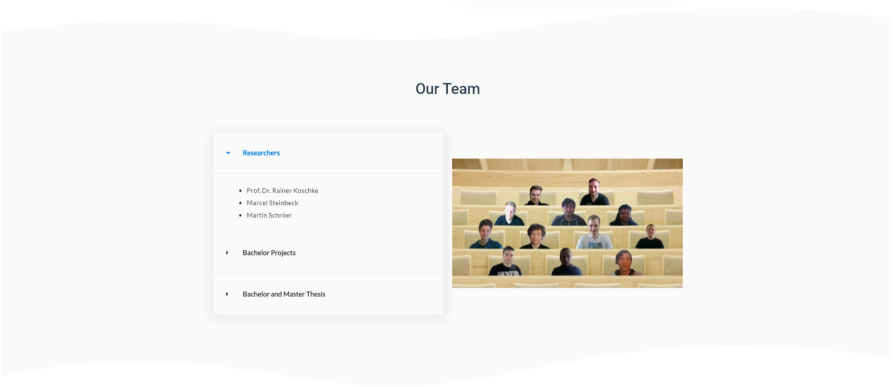

Abbildung 51: About SEE/Our Team

Contact

Abschließend fehlte nur noch eine Komponente, mit der es möglich ist, Kontakt mit der Projektleitung aufzunehmen. Auch bei diesem Part sollte ein schlichter und selbsterklä-

render Aufbau benutzt werden.

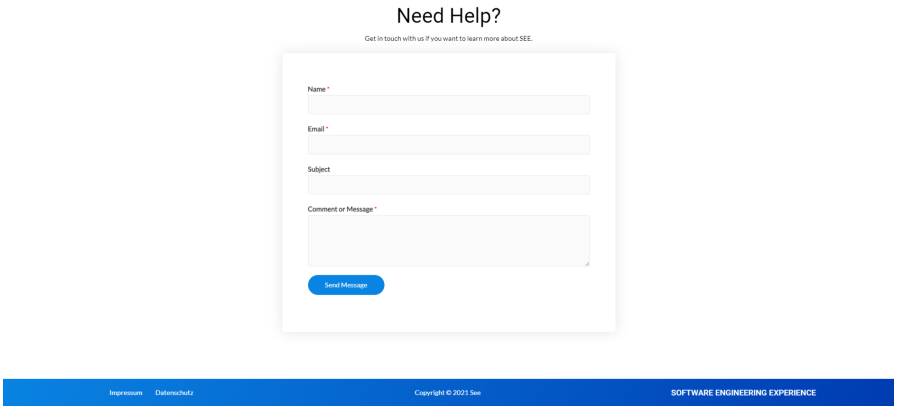

Abbildung 52: About SEE/Our Team

5.2 Abschlusspräsentation und Trailer

Ein Höhepunkt des Bachelorprojekts war der Projekttag, bei dem alle Bachelorprojekte ,wie auch SEE und dessen Ergebnisse, der Öffentlichkeit präsentiert wurden. Da dieser leider aufgrund der Covid-19-Pandemie nicht an der Universität Bremen in Präsenz stattfinden konnte, wurde dieser digital in Form eines Twitch-Livestreams abgehalten. Dafür haben zwei Teilnehmer des Projektes eine Präsentation vorbereitet, welche in 20 Minuten die Problemstellung und SEE als die Lösung für diese Probleme erläutert (siehe Sektion 2). Danach wurde SEE mit all seinen Funktionen noch im Detail vorgestellt. Zur visuellen Unterstützung der Präsentation wurde von Projektteilnehmern und der Projektleitung ein Trailer erstellt. Dazu wurde eine optisch ansprechende Bibliotheksszene zusammen mit einem erstellten Skript verwendet, welches automatische Kamerafahrten ermöglicht. Im Trailer wurden dann einmal alle Anwendungsfälle von SEE dargestellt und beschrieben. Der Trailer ist ebenfalls auf der Webseite[50] von SEE einsehbar, zusätzlich auch auf YouTube[51]. Während unseres Vortrages waren ca. 200 Zuschauer vertreten, wovon einige hinterher auch tiefer greifende Fragen gestellt haben, die von den Präsentierenden beantwortet wurden.

Nach Abschluss der Präsentation war es für alle Zuschauer möglich, die Projektteilnehmer von SEE im Zoom-Raum zu treffen, um über SEE oder allgemein Softwareentwicklung zu sprechen. Ein weiteres Highlight dieser Präsentation war das von einzelnen Projektteilnehmern entwickelte Setup, welches es ermöglichte, SEE mit eigens eingereichter Software im Browser zu testen und so seine eigene Software, dargestellt als Code-City, betrachten zu können.

Dieses Setup funktionierte unter Verwendung von noVNC[52], einem Framework,

[50] https://see.uni-bremen.de/ (letzter Abruf: 30.05.2021)
[51] https://youtu.be/V2WpRtKF5wM (letzter Abruf: 31.05.2021)
[52] https://novnc.com/ (letzter Abruf: 30.05.2021)

welches sich über eine HTML5-Schnittstelle im Browser zu einem VNC-Server[53] auf einem Rechner im Universitäts-Netzwerk verbinden lässt. Dieser Rechner (zusammen mit weiteren Rechnern, für den Fall, dass mehrere Besucher ihre Software parallel einsehen wollen) wurde soweit eingerichtet, dass fremde Software zunächst schnell analysiert und in ein entsprechendes GXL-Format übersetzt werden konnte und daraufhin in SEE auf diesem Server visualisiert werden konnte. Dadurch konnten wir unseren Besuchern einfach einen Link schicken, womit sie SEE im Browser sehen und steuern konnten, nachdem wir ihre Software dort geladen hatten. Dieser Service wurde von einigen Nutzern interessiert angenommen, andere wollten SEE im Allgemeinen ausprobieren und bekamen so die Möglichkeit, mit einer exemplarischen Code-City zu interagieren. Rückwirkend betrachtet wäre es sinnvoll gewesen, frühzeitig in anderen Gruppen und Projekten für diesen Service zu werben, damit dieser noch besser genutzt werden kann und weitere Interessenten aufkommen. Dies wurde dann während des Projekttages nachgeholt.

Zusammenfassend ließ sich der Projekttag für dieses Projekt also trotz der virtuellen Bedingungen sehr gut umsetzen und ermöglichte, besonders durch den Service danach, vielen Nutzern einen umfangreichen Einblick in die Funktionen und Visionen von SEE.

5.3 Feedback und Außenwirkung

SEE soll am Ende eine Software sein, welche Softwareentwicklung verbessern und unterstützen soll. Der Fokus liegt dabei auf der Vergleichbarkeit verschiedener Metriken auf Basis des Codes einer Software. Daher stellte sich die Frage: Wie präsentieren wir SEE auf so eine Art, sodass alle relevanten Informationen verstanden bzw. lückenlos übermittelt werden? Mit dieser Frage beschäftigten sich Rainer Koschke und der Bachelorstudent Jan Woiton. Als Resultat organisierte Jan einen Termin mit Verantwortlichen der IT aus verschiedenen großen Unternehmen, bei welchem wir SEE das erste Mal in einem Unternehmenskontext vorstellen konnten.

Das Gespräch lief einwandfrei, wir erhielten sehr gutes Feedback und wurden dazu ermutigt, die Personen auf dem Laufenden zu halten. Diese interessierten sich am Ende für eine Demo-Version von SEE, um die Software auch testen zu können. Insgesamt war die Präsentation ein voller Erfolg, es wurden neue Informationen über den Eindruck, den SEE hinterlässt, gesammelt, und die Schwerpunkte des Projekts wurden unter Berücksichtigung dieser neu evaluiert. Das Feedback kann folgendermaßen zusammengefasst werden:

Positives

- Die Kernidee des Vergleichens von Visualsierungen verschiedener und konfigurierbarer Metriken sei neu und „fantastisch“.
- Die Umgebung und die Schnittstellen für sowohl Virtual Reality und Augmented Reality (HoloLens) fand Anklang.

[53] VNC steht für *Virtual Network Computing* und ist ein Protokoll zum Zugriff auf Rechner über ein Netzwerk, z. B. über das Internet.

Negatives

- Die Grafik von SEE muss ansprechender werden, vor allem die Avatare (siehe hierzu auch Abbildung 53 in Sektion 7.4).
- Die Interaktionen insgesamt, speziell aber auch die Steuerung sollen, wenn möglich, vereinfacht werden.

Aus dem Erfolg dieses Gesprächs und den erhaltenen Kommentaren haben wir gefolgert, dass in Zukunft als weitere wichtige Quelle für Optimierungsideen von SEE Nutzerfeedback stärker einbezogen werden soll.

6 Projekt- und Selbstreflexion

6.1 Moritz Blecker

Zum Start des Projektes war ich mir ziemlich unsicher, ob das Projekt erfolgreich verlaufen würde. Dies basierte vorrangig auf der Tatsache, dass ich weder Erfahrungen mit der Game-Engine Unity hatte, noch mit der Programmiersprache C#. Dazu kam noch, dass ich zwar C bzw. C++ grundlegend verstanden hatte, jedoch teilweise als eher komplizierter empfunden habe. Diese Hindernisse wurden aber zum Glück schnell aus dem Weg geräumt, da C# doch eher Java als C++ ähnelt. Allerdings erwies sich besonders der Start mit Unity als holprig. Zunächst einmal hatte ich noch nicht herausgefunden, wie man die Visual Studio IDE richtig einrichtet, sodass auch die integrierte auto-completion funktioniert. Wie ich später hören musste, hatten dieses Problem auch andere, wobei ich es abschließend durch einen Wiki Eintrag lösen konnte. Probleme wie diese sind gelegentlich immer mal wieder aufgetreten, diese hätten häufig durch bessere interne Kommunikation vermieden bzw. schneller gelöst werden können. Danach bestand immer noch das Problem, dass ich nicht ganz verstanden hatte, wie Unity-GameObjects und C#-Skripte usw. zusammenhängen, dies habe ich vor allem bei meinem ersten Issue gemerkt.

Nachdem dieses Verständnis jedoch erworben wurde, verlief der weitere Fortschritt immer steiler und ich merkte, wie mir auch die Integration von neuen Features immer leichter fiel. Besondere Fortschritte machte ich, als Lino und Thore sich meinem Issue anschlossen und wir dann gemeinsam über Probleme diskutieren konnten. Diese Art von Pair Programming haben Thore und ich dann auch noch in späteren Issues als sehr positiv wahrnehmen und fortführen können. Natürlich ist es so, dass man grundlegende Dinge eigenständig implementiert, aber gerade bei der Konzeption und beim Debugging komplexerer Anwendungsfälle kann ein persönlicher, oder zumindest virtueller Austausch in einem Gespräch häufig schneller zum Ziel führen. Dies konnte ich auch schon häufig am Anfang durch die Gespräche mit den Betreuern, bei mir war es vornehmlich Rainer, feststellen: So entwickelte es sich oft so, dass ich teilweise Stunden über ein Problem nachgedacht habe, aber als ich dann spontan ein Meeting mit Rainer hatte und ein paar andere Denkansätze dadurch mit an die Hand bekam, es doch innerhalb von wenigen Minuten lösen konnte.

Zur Teamarbeit im Projekt und dem Austausch untereinander bin ich jedoch geteilter Meinung: Einerseits waren die wöchentlichen Meetings sehr gut, um einen groben Überblick über den Verlauf und die Fortschritte des Projektes zu bekommen, anderseits gingen sie des Öfteren zu sehr ins Detail. Im Nachhinein betrachtet muss ich sagen, dass die Meetings um einiges angenehmer gewesen wären, wenn auch das Medium der Webcam regelmäßig von allen genutzt worden wäre. So hätte man meiner Meinung nach ein besseres Kennenlernen ermöglicht. Was ich dagegen als sehr positiv erachtet habe, war der Erfolg der „Weihnachtsfeier“ — also des Spieleabends im Dezember 2020. Hier konnte man noch einmal alle Projektteilnehmer besser kennenlernen. Ein derartiger Abend wäre rückblickend bei so einem solchen Distanz-Projekt eventuell schon direkt am Anfang gemeinschaftsfördernd gewesen. In Zukunft wäre eventuell ebenfalls die Einführung eines „virtuellen Offices“ ein valides, zielführendes Konzept. Dazu könnte man im ersten Schritt einen Zoomraum bzw. Discord-Server einrichten, der jedem Projektteilnehmer einen Arbeitsplatz zum Entwickeln, einem einfachem Gedankenaustausch, oder auch Pausengesprächen mit anderen Entwicklern bieten könnte. Dies wäre ebenfalls gemeinschaftsfördernd und könnte so die Produktivität des Projektes erhöhen.

Langfristig wäre dies natürlich auch in einer VR-Umgebung wie SEE denkbar, obwohl sich der Fokus dieser Funktion natürlich vom eigentlichen Zweck von SEE unterscheidet. Diese Idee ist allerdings zu umfangreich, um sie in diesem Rahmen weiter auszuführen. Sie bietet jedoch eventuell eine angenehme Idee für die Zukunft von SEE, gerade dadurch, dass mobile Workplaces aktuell mehr und mehr in den Vordergrund rücken.

Abschließend finde ich, dass das Projekt ein Erfolg ist. Ich habe für mich selbst viel dazugelernt, sowohl meinen Programmierstil verbessert, als auch mein Repertoire an verwendbaren Techniken zur Lösung bestimmter Probleme erweitert. Auch im Bereich der Teamarbeit habe ich dazu gelernt. Besonders würde ich hier noch die Abschlusspräsentation als neue, persönliche Erfahrung hervorheben, da es für mich eine der ersten Präsentationen in diesem Umfang vor mehr als 200 Leuten war.

6.2 Thore Frenzel

Für mich stellt dieses Projekt insgesamt einen Erfolg dar. Sowohl individuell, als auch im gesamten Projekt haben wir unsere Ziele fast vollumfänglich erreicht. Mein individuelles Hauptziel war es, einen soliden Beitrag zum Projekt zu leisten und darüber hinaus sowohl Fortschritte in spezifischen Bereichen der Implementierung wie Unity und C# zu machen, als auch in allgemeineren Disziplinen der Softwareentwicklung wie Softwarearchitektur, Softwaredesign und weiteren gängigen Praktiken in größeren Softwareprojekten.

Zunächst bedeutete die erstmalige Einarbeitung in eine große Codebasis mit unbekannten Technologien zwar eine große Herausforderung, wurde jedoch von den Organisatoren des Projekts gut moderiert und unterstützt und ermöglichte so nach einigen Wochen Einarbeitungszeit auch produktive Arbeit. Die Kommunikation innerhalb des Projektes war anfangs aufgrund der vielen unbekannten Gesichter und den relativ unpersönlichen Onlinemeetings schwierig. Auch dies wurde aber ebenfalls durch einen gemeinsamen Online-Spieleabend, mehr Kameranutzung in Meetings und ein wenig

Zeit der Zusammenarbeit zunehmend verbessert.

Besonders die gute Erreichbarkeit der Betreuer sorgte dafür, dass das Projekt zunehmend mehr Fortschritte machte. Probleme in Bezug auf Kommunikation, Erreichbarkeit und Mitarbeit unter den Teilnehmern kamen zwar vor, jedoch wurde auch das sehr zügig angesprochen und verändert. Gerade hierdurch zeichnet sich diese Projekt für mich als Erfolg aus. Trotz verschiedener Fähigkeiten, Motivationen und Ausgangsbedingungen wurden am Ende alle Ziele erreicht. Alle Probleme, die bestanden, wurden offen angesprochen, optimiert und verbesserten so sukzessiv die Zusammenarbeit der Teilnehmer.

Da ein Bachelorprojekt neben den eigentlichen Zielen als Lernprojekt auch noch das Erlernen von neuen Fähigkeiten und Zusammenarbeit als Ziel beinhaltet, ist gerade diese Tatsache ein sehr wichtiger Faktor für die Bewertung des Projekts. Dieser Faktor ist allen Teilnehmern gut gelungen. Des Weiteren konnte man speziell von einer Kerngruppe innerhalb kürzester Zeit Feedback zu individuellen Entwicklungen erwarten, aber auch Hilfe und Unterstützung erfragen, was die Arbeit häufig erleichterte und auch angenehmer machte. Regelmäßigen Austausch zwischen den Projektteilnehmern gab es nach kurzer Zeit nicht nur zu den wöchtentlichen Meetings.

Für nachfolgende Projekte würde ich mir persönlich wünschen, dass sich alle Projektteilnehmer noch mehr integrieren und die zuvor angesprochene Kerngruppe noch weiter wächst. Hierfür wäre es eventuell sinnvoll, eine konkrete Pause zu bestimmen, in der keine Leistungen von Projektteilnehmern erwartet werden. Hierdurch könnte die Motivation danach bei allen Teilnehmern wieder steigen, da ein Projekt von nahezu 8 Monaten ohne einer Pause abgesehen von den Weihnachtstagen einen relativ langen Zeitraum der Dauerbelastung bedeutet. Ein weiterer wünschenswerter Punkt wäre eine genauere Definition der Erwartungen an die Teilnehmer, genauso wie regelmäßigere Zwischenstände zur individuellen Leistung.

Abschließend kann selbstkritisch noch angemerkt werden, dass auch ich in Zukunft versuchen sollte, früher den Weg zu anderen Projektteilnehmern zu suchen. Genauso sollte ich versuchen, fachbezogene Probleme frühzeitig bei anderen Teilnehmern anzusprechen, sodass keine Zeit vergeudet wird, die man besser nutzen könnte und man so den gesamten Erfahrungsschatz der Projektteilnehmer besser nutzen kann.

6.3 Thorsten Friedewold

Meiner Meinung nach war das Projekt eine sehr interessante und lehrreiche Veranstaltung, aus der ich viele verschiedene Einsichten mitnehmen konnte. Mein persönliches Ziel war es, in diesem Projekt ein besseres Verständnis für die Entwicklung von 3D-Anwendungen zu gewinnen, da ich auf diesem Gebiet nahezu keinerlei Vorwissen besaß. Zu Beginn fiel mir der Einstieg in das Verständnis zwischen dem Zusammenspiel von Unity und C# durchaus schwer, was die Einarbeitung in die ersten Tickets für mich durchaus zu einer größeren Herausforderung machte. Nachdem diese anfänglichen Berührungsängste jedoch überwunden waren, konnte ich mich deutlich besser in den Aufbau der Software einarbeiten. Ein weiteres Ziel war es, ein besseres Verständnis von kooperativer Softwareentwicklung zu bekommen und die damit einhergehenden

Herausforderungen besser zu verstehen.

Der Verlauf des Projektes wurde aus meiner Sicht durch die Corona-Bedingungen kaum behindert und ich hatte stets das Gefühl, dass trotz der Tatsache, dass nie ein persönliches Treffen zustande kam, eine ausgeprägte Gruppendynamik vorhanden war. Gerade die wöchentlichen Meetings haben mir sehr geholfen einen Überblick zu behalten, an welchen Stellen gearbeitet wurde, was wiederum Überschneidungen schnell aufgedeckt hat. Auch die Kommunikation empfand ich als sehr angenehm, wodurch nie das Gefühl aufkam, dass man mit einer Problemstellung alleingelassen wurde. Persönlich war genau das Gefühl des gemeinsamen Wirkens eine ganz neue Erkenntnis für mich, mit der ich mich rückblickend gesehen anfänglich schwer tat, welche ich aber in Zukunft nicht mehr missen möchte. Die Meetings gingen jedoch zuweilen sehr detailliert auf einzelne Problemen ein. Dies führte bei mir hin und wieder dazu, dass ich keine klare Unterscheidung zwischen allgemein relevanten Dingen und spezifischen Problemlösungen finden konnte. Hier hätte ich mir eine deutlichere Abgrenzung oder ein gesondertes Meeting gewünscht, um mehr Klarheit zu schaffen.

Die Aufgabenstellungen waren sehr interessant und haben mir einiges abverlangt. Gerade die Aufgaben, die ich bearbeitet habe (Siehe Tabelle 1 zu bearbeiteten Issues) drängten mich, mir die Thematik der Komplexität genauer anzuschauen, was mich im Vergleich zum Beginn des Projektes deutlich affiner für den sorgsamen Umgang von Systemressourcen gemacht hat. Jedoch wurde mir diverse Male klar, dass sich nicht alle Aufgabenstellungen mit meinem Wissen lösen lassen, was dazu führte, dass ich mich mehrere Male auch mit Fehlschlägen konfrontiert sah. Gerade in diesen Momenten war die gute Erreichbarkeit der Lehrenden äußerst hilfreich. Dennoch bleiben bei einigen Aufgabenstellungen tiefer führende Verbesserungsmöglichkeiten aus Zeitgründen unausgenutzt.

Die Verständigung mit den Gruppenmitgliedern und Lehrenden war sehr angenehm, wodurch die Arbeit deutlich erleichtert wurde. Fragen wurden schnell beantwortet und um hilfreiche Tipps ergänzt. Es ergaben sich schnell Gruppen von Mitgliedern, die in bestimmten Bereichen äußerst kompetent waren und auf deren Hilfe man zurückgreifen konnte.

Abschließend kann ich sagen, dass ich für die im Projekt gemachten Erfahrungen sehr dankbar bin. Gerade dadurch, dass ich nicht immer alle Probleme auf Anhieb lösen konnte, wurde mir klar, dass es zeit-technisch manchmal besser ist an einer anderen Stelle weiterzuarbeiten und Probleme später neu anzugehen. Darüber hinaus habe ich gelernt, wie wichtig eine gute Kommunikation ist und werde in Zukunft daran arbeiten, mich mehr in Diskussionen einzubringen, um näher am Geschehen zu sein. Außerdem konnte ich auch mein zu Beginn gestecktes Ziel, ein besseres Verständnis für die Entwicklung von 3D-Anwendungen zu erhalten, erreichen. Daher bin ich zusammenfassend äußerst zufrieden mit dem Verlauf des Projektes.

6.4 Falko Galperin

Die Arbeit im Bachelorprojekt hat mir Spaß gemacht und ist fast überall produktiv verlaufen. Trotz der Corona-bedingten Schwierigkeiten ist von den wöchentlichen

Meetings bis zur Kommunikation im Allgemeinen die Teamarbeit meiner Meinung nach sehr gut gelaufen. Da in einem Team von 12–14 Leuten die Koordination oft auch etwas schwieriger umzusetzen ist, waren auch Meetings in sehr kleinen Gruppen (2–4 Personen) außerhalb der wöchentlichen Meetings besonders hilfreich, in denen man sich gegenseitig beim Verständnis und Lösen von Problemen helfen konnte, speziell am Anfang des Projekts, als man noch keinen guten Überblick über die Codestruktur hatte.

Durch das Projekt habe ich auch viel Neues gelernt. Neben den speziellen und notwendigen Sprachen bzw. Frameworks, welche man für dieses Projekt zur Mitarbeit eben lernen muss (z. B. Unity und C#), sind viele der neuen Erkenntnisse aber auch generalisierbar und in zukünftigen Projekten anwendbar. Wegen der Natur von SEE haben dabei wohl alle Teilnehmer des Bachelorprojektes etwas zu Spieleentwicklung[54] und der Analyse von Software und Softwarearchitekturen gelernt. Durch die Portierung auf die HoloLens (siehe Sektion 4.4) habe ich auch einiges über die Integration von Frameworks[55] in bestehende Projekte gelernt, über die Konzepte und „Best Practices" von Mixed-Reality-Interaktionen, sowie darüber, wie ein Projekt dieser Art auf mehrere Plattformen ausgelegt werden kann, ohne den Entwicklungsaufwand z. B. durch Aufspaltung des Projekts zu sehr aufzuhalten. Diesbezüglich hat insbesondere die Arbeit am UI-Framework (siehe Sektion 4.1.1) sehr dazu beigetragen, weil andere SEE-Entwickler hiermit leicht UI-Komponenten verwenden können sollten, ohne sich Gedanken über die speziellen Ausprägungen pro Plattform machen zu müssen.

Organisatorisch habe ich neben der sinnvollen Aufteilung von Aufgaben innerhalb des Teams noch extensiv Erfahrung gesammelt, Feedback in Form von Code-Reviews zu hinterlassen, um Fehler aufzuspüren und anderen Entwicklern dabei zu helfen, besseren Code zu schreiben. Das gilt auch umgekehrt: Die Code-Reviews, die andere bei meinem Programmcode hinterlassen haben, waren auch sehr hilfreich dabei, meine eigene Codequalität zu verbessern.

Ich bin zwar insgesamt relativ zufrieden mit meiner Leistung, aber Verbesserungspotenzial sehe ich bei mir z. B. in drei Punkten:

1. **Dynamische Code-Reviews**: Die Code-Reviews, die ich durchgeführt habe, waren alle statischer Natur — das heißt, ich habe mir den Code meist nur angesehen und durchdacht, um auf mögliche Fehler oder Verbesserungen zu stoßen. Um aber wirklich alle Verbesserungsmöglichkeiten zu finden, wäre es sicherlich einige Male lohnenswert gewesen, den Programmcode auch auszuführen, und so über eine manuelle dynamische Analyse weitere Fehlerquellen zu bestimmen.

2. **Tests**: Insgesamt habe ich für SEE nur drei Testklassen erstellt[56], obwohl sich Tests für einige Komponenten gut umsetzen ließen und auf jeden Fall praktisch wären.

[54] Es handelt sich hier streng genommen natürlich um kein „Spiel", aber in diesem Projekt wurden hauptsächlich Konzepte aus der Spieleentwicklung (z. B. Szenen oder eine dreidimensionale Umgebung) angewendet im Gegensatz zu bspw. traditionellen Desktop-Anwendungen.

[55] In diesem Fall konkret das *Mixed Reality Toolkit* für die HoloLens (siehe Sektion 4.4), aber auch *Dynamic Panels* für die Code Windows (siehe Sektion 4.1.2) und das *Modern UI Pack* für das UI-Framework.

[56] Zwei Testklassen für Menü-Einträge und eine für Konfigurationsdialoge, wobei letztere in Zusammenarbeit mit Rainer Koschke entstanden ist.

3. **HoloLens**: Die Integration der HoloLens-Plattform in SEE wurde zwar durchgeführt und funktioniert im Unity-Editor, allerdings gibt es bei der Verwendung auf einem echten Gerät weiterhin große Probleme und die Plattform lässt sich nicht zufriedenstellend verwenden. Der Grund hierfür liegt hauptsächlich in der Schwierigkeit, SEE auf das Mixed-Reality-Toolkit anzupassen, ohne das Projekt in eine HoloLens- und eine Nicht-HoloLens-Version abzuspalten. Hier wäre es vielleicht gut gewesen, mehr mit projektexternen Experten (z. B. durch Posts in das *Unity-Forum*) zu kommunizieren, um die Probleme zu lösen.

Am Bachelorprojekt selbst sehe ich das Verbesserungspotenzial hauptsächlich in der Anfangsphase. Da sowohl die Benutzung von SEE (zumindest auf dem Stand des Projekts von damals), insbesondere aber auch die Codebasis von SEE relativ komplex ist, wäre hier eine ausführliche Einleitung zu Beginn des Projekts sehr hilfreich gewesen. Diese Einleitung kann — sortiert nach Präferenz — in Form eines interaktiven Meetings, in Form einer Präsentation oder in Form einer Wiki-Seite existieren und sollte vor allem die verschiedenen Möglichkeiten von SEE aus Benutzersicht (z. B. Evolution, Architekturanalyse, „normale“ Code-City...) und den Aufbau des Codes von SEE (z. B. Aufbau der Ordnerstruktur, die `AbstractSEECity`-Klasse, etc.) einführend erläutern. Eine Einleitung in Unity hingegen wurde uns in Form eines E-Books bereitgestellt, das sehr nützlich war, um sich schnell in die Grundlagen von Unity einzuarbeiten.

Abschließend kann ich nur wiederholen, dass das Projekt aus meiner Sicht sehr zufriedenstellend gelaufen ist. Sowohl die Fortschritte in SEE selbst als auch in den Fähigkeiten der Teilnehmer sind klar erkennbar. Ausgehend von den Erfahrungen in diesem Projekt werde ich SEE wahrscheinlich ebenfalls als Masterprojekt wählen.

6.5 Lino Goedecke

Zunächst muss ich sagen, dass ich vor dem Projektbeginn doch (etwas vorsichtig formuliert) größere Unsicherheiten hatte, ob ich den Erwartungen aufgrund mangelnder Kenntnis von Unity trotz der „Einlesephase“ vor dem Projekt, nicht den besten Programmierfähigkeiten generell und keinerlei Kenntnis in der Programmiersprache C#, gerecht werden kann. Nach einer kurzen Einarbeitungsphase in die neuen Technologien stellten sich diese aber aufgrund

- guter Kommunikation mit Kommilitonen
- des Engagements der Betreuer
- der Tatsache, dass C# der Programmiersprache Java strukturell und syntaktisch sehr ähnelt

als nicht ganz zutreffend heraus. Inhaltlich waren alle verteilten Aufgaben zu jedem Zeitpunkt schaffbar und selbst der vorgegebene Zeitrahmen war wider Erwarten größer als gedacht. Falls es bei einigen Issues doch einmal nicht sofort ersichtlich war, wie dies konkret umzusetzen ist, war die Kommunikation mit den Betreuern, wie bereits erwähnt sehr hilfreich. Als Fazit aus der Kommunikation ziehe ich für mich allerdings, dass für

inhaltliche Diskussion entweder ein „Tool“ wie SEE oder ein virtuelles Meeting sehr viel effizienter ist, als inhaltliche Probleme über E-Mail/Mattermost/Telegram oder andere schriftliche Kommunikationskanäle zu klären.

Als weiteren Erkenntnisgewinn ziehe ich darüber hinaus, dass bloßes Einlesen ohne praktische Einarbeitung nebenher innerhalb des Programmierens nur begrenzt hilfreich ist. Das parallele praktische Programmieren unter Zuhilfenahme von theoretischem Background ist vermutlich die effizientere Art und Weise ist, sich in neue Technologien oder Teilaspekte der Programmierung einzuarbeiten.

Generell muss ich sagen, dass mir das Projekt insgesamt und auch die Programmierarbeiten ziemlich viel Spaß gemacht haben, was ich im Vorfeld eher nicht geglaubt hätte und ich mir mittlerweile sogar vorstellen könnte für eine gewisse Zeit in dem Bereich zu arbeiten. Zumindest für kleinere Issues, Bugfixes oder kleine Features — auch wenn ich dann wohl zuvor intensiv an meiner Syntax, Lesbarkeit und generell Kodierrichtlinien arbeiten müsste. Vor dem Projekt war ich eher der Meinung, die Projektleitung und -organisation würde mir mehr Spaß machen, aber mittlerweile könnte ich mir kaum vorstellen, lediglich zu delegieren statt selbst an einigen Aufgaben mitzuarbeiten.

Selbstkritisch müsste ich anmerken, dass es wohl nicht bei der Quantität der geleisteten Arbeit, sondern an der Codequalität, speziell bei der Les- und Wartbarkeit meines Codes durchaus noch Verbesserungspotential gibt, auch wenn ich die meisten Issues inhaltlich gelöst habe. Ferner muss ich anmerken, dass ich, obwohl ich grundlegendes Interesse an Projektmanagement habe, mich aber speziell bei diesem Projekt auch aus persönlichen Zeitgründen zwar aus vielen organisatorischen Aspekten oder Vorbereitungen herausgehalten habe, dennoch anhand der Organisation der Betreuer viel lernen konnte, gerade was das Zeitmanagemment der Issues betrifft.

Abschließend bleibt die Erkenntnis, dass projektbezogene Arbeit gerade im Bereich der Softwareentwicklung, die oft asynchron stattfindet, in einem hohen Maße an der Qualität der Kommunikation innerhalb des Teams gekoppelt ist. Dies hat aber im Bachelorprojekt trotz der widrigen Umstände der Pandemie im „digitalen Semester“ außerordentlich gut funktioniert.

6.6 Leonard Haddad

Als uns das Projekt SEE (damals noch SEA) vorgestellt wurde, wusste ich nicht so ganz was ich mir hierbei vorstellen sollte, allerdings hat mich Rainers Enthusiasmus und das gesamte Konzept sehr dazu motiviert, an diesem Projekt teilzunehmen. Rückblickend betrachtet muss ich sagen, dass es ein echter „Treat“ war, am Projekt SEE teilzunehmen. Die Teilnehmer sind sehr motiviert, was dazu führt, dass das Arbeiten umso mehr Spaß macht. Sie sind auch so nett, dass es gar kein Problem ist, wenn man selbst nicht weiterkommt, da es immer jemanden gibt, der einem helfen würde.

Zu Anfang war es etwas schwer sich in das Projekt einzuarbeiten. Ich hatte zwar schon vorher Spiele mit C# programmiert, dennoch war das Ganze nicht so einfach. Nachdem man sich dann genug eingearbeitet hat, lief dann allerdings alles bestens. Die Arbeit selbst hat mir sehr viel Spaß gemacht, vor allem da ich mich mit den „künstlerischen“ Aspekten (visuelle „Verbesserungen“) von SEE befasst habe. SEE war anfangs etwas sehr

abstrakt und simpel gebaut und brauchte aus meiner Sicht einfach etwas mehr Leben, welches ich dem Projekt hoffentlich geben konnte.

Wir hatten ja leider keine Chance uns persönlich zu treffen, dennoch finde ich, dass der Projektablauf sehr gut geklappt hat. Ich konnte hierbei auch problemlos mit anderen Teilnehmern arbeiten, wobei ich selbst nicht einmal in Deutschland war. Es ist schon unglaublich was die Technik heutzutage so zulässt.

Bevor ich abschließe möchte ich noch einige Verbesserungsvorschläge beifügen, welche ich persönlich gern an SEE sehen würde:

- Custom Keybindings — Es wäre sehr gut, wenn man die Tasten so verstellen könnte, wie man will.
- Die Welt um SEE ist meiner Meinung nach etwas zu abstrakt, es wäre schon etwas netter sich in einem gutaussehenden virtuellen Raum zu treffen.
- Die Bewegung der Charaktere ist etwas unpraktisch, mir wäre eine Standard-Bewegung mit WASD lieber um herumzulaufen.
- Nicht furchteinflößend aussehende Charaktere — die Mitarbeiter-Köpfe welche jetzt verwendet werden, könnten direkt aus einem Horror-Film stammen.

Zum Abschluss möchte ich mich noch einmal bedanken. Danke an Rainer dafür, dass ich an dem Projekt teilnehmen durfte und Danke an die anderen Teilnehmern dafür, dass sie so aktiv und hilfsbereit waren. Das Projekt SEE hat aus meiner Sicht ein großes Potenzial, eines Tages zu einem sehr guten, professionellen Produkt zu werden und ich hoffe es erreicht auch dieses Potenzial. Des Weiteren wünsche ich Rainer und den Teilnehmern, die weiterhin an SEE arbeiten viel Erfolg und ich freue mich dann eines Tages, die Resultate bestaunen zu dürfen.

6.7 Simon Leykum

Meiner Ansicht nach ist das Projekt größtenteils gut verlaufen. Ich hatte mir, bevor das Projekt angefangen hat, gewünscht, im Laufe des Projektes, Kenntnisse im Arbeiten mit Unity, insbesondere in VR und im Programmieren mit C# zu erwerben. Ich hatte, bevor das Projekt angefangen war, noch keine Vorkenntnisse in C#. Somit habe ich insbesondere hier im Laufe des Projektes viel gelernt. Bei der Arbeit mit Unity hatte ich schon leichte Vorkenntnisse, aber konnte diese auch durch das Projekt noch wesentlich erweitern. Die Arbeit mit VR ist mir leider etwas zu kurz gekommen, sodass die meisten Dinge, an denen ich gearbeitet hatte, nicht direkt mit VR zu tun hatten. Die Aufgabenstellungen waren aber dennoch interessant und es hat mir Spaß gemacht, diese zu bearbeiten.

Die Organisation im Projekt schien mir größtenteils sehr gut verlaufen zu sein. Insbesondere die wöchentlichen Meetings haben geholfen, einen Überblick darüber zu bekommen, woran die anderen arbeiten und sich zu organisieren und der eingerichtete Mattermost-Kanal hat geholfen, schnell Fragen beantwortet und Feedback zu seinen Aufgaben zu bekommen. Bei den Meetings hätte ich mir gewünscht, dass manche Dinge ein bisschen länger gekommen und andere Dinge ein bisschen kürzer gekommen wären.

Insbesondere manche Dinge, die dann nur einen kleinen Teil der Gruppe betroffen haben, aber sehr detailliert besprochen worden sind und andere Dinge, die möglicherweise wichtiger gewesen wären, zu kurz gekommen sind. Ich hätte möglicherweise für die Meetings eine Aufteilung in Teilgruppen, die an ähnlichen Aufgaben arbeiten und dann seltenere Meetings mit der ganzen Gruppe, in der alle Teilgruppen vorstellen, woran sie gearbeitet haben, als für sinnvoll erachtet. Auch hatte ich öfter das Problem, dass mir die Funktionsweise von etwas nicht ganz klar war, aber dass das Wiki keine genaueren Informationen gegeben hatte und eine Erschließung über die Dokumentation intensive Einarbeitung gebraucht hat. Ich erachte also ein ausführlicheres Wiki, das bei Änderungen dann auch immer direkt aktualisiert wird, als sinnvoll.

Die Betreuung durch die Lehrenden war sehr hilfreich, diese waren immer zu erreichen und haben bei Problemen oder Fragen schnell geholfen. Was ich mir dennoch gewünscht hätte, wäre ein wenig mehr Feedback während des Projektes zu der individuellen Leistung. Die Arbeit mit den anderen Projektteilnehmern lief ohne Probleme. Fragen wurden immer schnell und kompetent beantwortet und es war immer möglich, konstruktives Feedback zu den eigenen Aufgaben zu bekommen. Auch bei Aufgaben, die zusammen mit anderen Gruppenmitgliedern bearbeitet worden sind, lief die Kommunikation reibungslos und man konnte sich immer auf die anderen Gruppenmitglieder verlassen. Mir ist aber aufgefallen, dass die organisatorischen Aufgaben oft von einem kleinen Teil der Gruppe übernommen worden sind. Hier hätten andere, inklusive ich, sich mehr einbringen können.

Ich konnte meine Aufgaben zwar alle bewältigen, aber sehe durchaus noch Verbesserungsmöglichkeiten, insbesondere bei der Sauberkeit und Effizienz meines Codes. Zudem möchte ich mir angewöhnen, meine Implementierungen mehr zu testen, da es jetzt im Laufe des Projektes mehrmals dazu kam, dass ich noch nachträglich Fehler beheben musste, die erst rückwirkend aufgefallen sind.

Zusammenfassend lässt sich also sagen, dass ich trotz einiger Kritikpunkten das Projekt im Allgemeinen und auch für mich persönlich als Erfolg ansehe. Ich konnte meine Kenntnisse insbesondere in der Programmierung mit C# und dem Arbeiten mit Unity verbessern und einen besseren Einblick bekommen, wie ein solches Projekt organisatorisch abläuft, was mir möglicherweise später bei der Planung eigener Projekte helfen wird.

6.8 Christian Müller

Aufgrund meiner bereits vorhandenen Erfahrungen im Bereich der .NET-Entwicklung habe ich mich für dieses Projekt entschieden, um einen erneuten Einstieg in die Entwicklung von 3D-basierten Anwendungen mit Unity zu bekommen. Als weiteres Ziel hoffte ich, einen Einblick in die Mixed Reality Entwicklung mit Unity zu bekommen, da auch für meine eigenen Projekte AR (Augmented Reality) bzw. MR (Mixed Reality) Anwendungen geplant sind. Hinzu kommt, dass das eigentliche Projekt der Visualisierung von Code in Form einer Code-City auch auf mein Interesse gestoßen ist.

Zum Beginn des Projekts ist mir als .NET-Entwickler die doch sehr ungewöhnliche Projektstruktur aufgefallen. Aufgrund weitverbreiteter Design Patterns in der .NET-Welt

gibt es Projektstrukturen, die Neueinsteigern in Projekten einen schnellen und sofortigen Einstieg ermöglichen. Das ist bei SEE nicht der Fall und hat vermutlich den Grund, dass SEE ursprünglich aus einem mit Unreal Engine entwickelten Projekt entstanden ist und die initiale Entwicklung auch von Nicht-.NET-Entwicklern vorgenommen wurde. Es fehlt beispielsweise eine modulare, .NET-typische, in mehrere Projekte aufgeteilte, für die Zielplattform spezifische Projektstruktur, um plattformspezifische Unabhängigkeit ohne Präprozessoren zu erreichen.

Die Entwicklung mit Unity selbst weicht stark von der .NET-typischen Entwicklung an sich ab, da Unity sich nicht an .NET-Konventionen wie „Naming Conventions“ von Eigenschaften, Variablen oder Methoden hält. Das ist zu Beginn etwas gewöhnungsbedürftig, stellt aber an sich kein Problem dar, würde aber die Arbeit für andere .NET-Entwickler vereinfachen, wenn diese angewendet werden. So ist es auch für .NET-Umgebungen ungewöhnlich, wie Unity ein HoloLens-Projekt erstellt. So wird aus dem Unity-Projekt ein .NET-Projekt für UWP kompiliert, welches dann wiederum kompiliert wird um auf der HoloLens laufen zu können. Dieser doch sehr ungewöhnliche Vorgang ist auch der größte Nachteil von Unity und erklärt, warum die MRTK-Community einen modularen Aufbau empfohlen hat. Der Buildprozess verlängert sich durch das ständige Generieren des gesamten Codes extrem und kann durch einen modularen Aufbau definitiv verkürzt werden und sollte auch in der zukünftigen Planung des Projekts zur besseren Wartung und Produktivitätssteigerung in Betracht gezogen werden.

Insgesamt hat dieses Projekt mir geholfen mein Wissen im Unity-Bereich zu erweitern, darunter das Verwalten von Assemblies in Unity mittels AsmDef-Dateien, was sich grundlegend von anderen .NET-Projekten unterscheidet und hat mir auch die Augen für andere .NET-Engines wie Stride3D und Wave Engine geöffnet, die eine nahtlose Integration in die .NET-Entwicklung ermöglichen. Gerne hätte ich mehr Zeit in dieses Projekt investiert und hätte auch gerne mehr mit anderen Teilnehmern der Gruppe zusammengearbeitet. Leider wurde das aufgrund der aktuellen Pandemielage erschwert und so bin ich hauptsächlich im MRTK-Bereich geblieben. Für zukünftige Projekte plane ich mehr Code-Reviews durchzuführen. Außerdem bin ich aufgrund der Arbeit am MRTK und mit der HoloLens nicht zum Schreiben von Unit Tests gekommen. Dieser Aspekt der Software bedurfte hauptsächlich Blackboxtests in Form von integrativen System- und manuellen Tests, da das Generieren des IL2CPP Codes und das darauf folgende Ausführen auf einem Emulator nicht per Unit Test getestet werden konnte.

Insgesamt war der Verlauf des Projekts sehr positiv. Durch die wöchentlichen Meetings waren alle immer auf dem neusten Stand und auch die Erreichbarkeit aller Beteiligten war immer gegeben. Auch, dass mir ermöglicht wurde meine eigene Software mit SEE zu analysieren, hat nicht nur mir einen neuen Einblick auf mein eigenes Projekt gegeben, sondern hat auch dazu geführt, dass neue Ideen in das Projekt einfließen konnten. Daher würde ich die abschließende Erfahrung mit dem Projekt als sehr positiv bewerten.

6.9 Nick Seedorf

SEE — Ein Projekt zum Anfassen und Erleben. Von den verfügbaren Bachelorprojekten stach SEE für mich heraus. Auch wenn ich mir nicht sicher war, ob ich bei einem so

komplexen Projekt den Durchblick behalten und einen vernünftigen Beitrag dazu leisten könnte, so fand ich das Projekt doch sehr attraktiv, da man hier etwas hatte was praktisch anwendbar, und dadurch gut nachvollziehbar war. Auch fand ich es sehr spannend, dass bei SEE mit Unity eine Spiele-Engine verwendet wird, was gut zu meinem Interesse an Computerspielen passt. Außerdem kam es mir sehr entgegen, dass einige meiner Studienkollegen ebenfalls dieses Bachelorprojekt wählen wollten, wodurch ich mich nicht in einer komplett neuen Gruppe wiederfand, in der ich niemanden kannte.

Die Einarbeitung in SEE erwies sich Anfangs als schwierig, da man sich in einem Projekt wiederfand, was bereits seit längerer Zeit existierte. Dadurch musste man erst einmal durchsteigen wie SEE aufgebaut ist und welche Zahnräder ineinandergreifen, damit SEE so funktioniert wie wir es vorgefunden haben. Hierbei erhielten wir allerdings sofort Hilfe von den Betreuern, die allgemein immer sofort zur Stelle waren wenn jemand Hilfe brauchte oder etwas unklar war, was den Einstieg in das Projekt ungemein erleichterte.
Als es dann so richtig losging und jeder Issues erhielt an denen er arbeiten konnte, kam man dann ziemlich schnell in dem Projekt an und verstand auch mit der Zeit immer mehr was alles miteinander zusammenhing. Dadurch kam man schnell in einen guten Workflow, der durch die wöchentlichen Meetings mit der ganzen Projektgruppe gut abgerundet wurde. So berichtete in den Meetings jeder von den Issues an denen er grade arbeitet und jeder wurde auf den neusten Stand gebracht. Die Meetings förderten außerdem die Interaktion der Gruppe, was anders aufgrund der Pandemie-Situation leider nicht möglich war. Da die meisten Projektteilnehmer sich vor dem Projekt noch nicht so richtig kannten, wäre eine persönliche Interaktion natürlich schöner gewesen als ein Meeting über Zoom, allerdings wurde mit einer Vorstellungsrunde zu Beginn des Projekts so gut es ging Abhilfe geschaffen und im Laufe des Projekts entwickelte sich ein gutes Gruppengefühl, bei dem jeder für den anderen da war wenn Hilfe benötigt wurde.

Mir persönlich hat es bei dem Bachelorprojekt sehr geholfen, dass das Meiste, was man selber bearbeitet hat, visualisierbar und damit „anfassbar“ war. So konnte ich bei den meisten Issues die ich bearbeitet habe in Unity direkt sehen was ich eben programmiert habe und ob dies nun so funktioniert wie gewünscht oder nicht. So konnten Fehler oftmals schnell festgestellt werden oder man sah Dinge die eventuell noch ausbaufähig sind. Außerdem hatte man so immer das Gefühl, dass man etwas zu dem Projekt beiträgt was auch sichtbar ist, und man hat nicht fünf Wochen lang an Anpassungen gearbeitet die im Endeffekt für die meisten anderen nicht wirklich nachvollziehbar sind. So hatte man immer einen Grund zur Freude wenn etwas so funktioniert und auch so aussieht wie man es sich im Vorhinein vorgestellt hat.

Auch ist man immer an seinen Herausforderungen gewachsen. Wenn man mal ein Issue bearbeitet hat, was zu Anfang als sehr schwierig wahrgenommen wurde, hat man es doch immer mit Hilfe der Gruppe oder der Betreuer geschafft, die Aufgaben zu bewältigen, wodurch man einiges dazugelernt hat.

Alles in allem finde ich, dass SEE als Bachelorprojekt eine tolle Erfahrung war und ich selber im Laufe des Projektes einiges gelernt habe. Seien es zum Beispiel die konkreten Programmierfähigkeiten die ich ausgebaut habe oder die tolle Zusammenarbeit in so einer großen Projektgruppe — SEE war eine Erfahrung die ich so vorher noch nicht

gemacht habe.

6.10 Robin Theiß

Das Bachelorprojekt war für mich insgesamt eine sehr hilfreiche und gute Erfahrung. Bei der Suche nach einem Bachelorprojekt stach mir SEE direkt ins Auge. Die Thematik fand ich sehr spannend, gerade durch mein privates Interesse an Visualisierungen und Computerspielen, allerdings war ich mir etwas unsicher, inwieweit ich mich in diesem Projekt zurechtfinden könnte. Nachdem ich dann aber von einem Bekannten eine Empfehlung für SEE bekommen habe und weitere Studienkollegen fand, die ebenfalls Interesse an der Thematik zeigten, fühlte ich mich bestärkt an dem Projekt teilzunehmen. Ich wusste nicht genau was ich von diesem Projekt erwarten sollte, allerdings hatte ich definitiv das Ziel, generelle Fortschritte in Bereichen der Implementierung bezüglich Unity und C# zu machen, als auch mehr Erfahrung in Bereichen der Softwareentwicklung zu sammeln.

Der Anfang des Projektes war definitiv eine große Herausforderung, die bereits vorher vorhandenen Berge von Code in Verbindung mit einer neuen Technologie gestalteten die Einarbeitung als wirklich schwierig. Zwar war die Entwicklungs-Sprache nicht gänzlich anders zu den mir bekannten, allerdings gab es trotzdem, gerade mit der vorher komplett unbekannten 3D-Engine, viel neues zu entdecken und lernen. Diese Herausforderung wurde aber durch die Betreuer deutlich erleichtert, die für alle Fragen immer schnell erreichbar waren und ihr Bestes gegeben haben, um alle Beteiligten in dem Einarbeitungsprozess und auch im folgenden Projektverlauf tatkräftig zu unterstützen.

Nachdem die Einarbeitungsphase mehr oder weniger durchlaufen war, ging es an die Bearbeitung der Issues. Der regelmäßige Austausch wurde dann über wöchentliche Meetings gestaltet, die zu sehr großen Teilen von allen Teilnehmern ernst genommen wurden und es nur selten zu Verspätungen oder Versäumnissen kam. Zu Beginn war die Zusammenarbeit noch etwas schwierig, durch die Pandemie und die dadurch resultierenden Kontaktbeschränkungen war es zu Beginn nicht so leicht ein Gruppengefüge aufzubauen. Auch die Kommunikation untereinander war noch nicht ganz so leicht, da es viele unbekannte Personen waren und durch die reine Online-Kommunikation das zwischenmenschliche etwas zu kurz gekommen ist. Das besserte sich aber im Verlaufe des Projektes merklich und gerade während der Meetings kam es zu immer regeren Austauschen und Beteiligungen. Mittlerweile kann ich sogar sagen, dass sich durch SEE gute Freundschaften entwickelt haben, die hoffentlich noch weit über die Uni hinaus bestehen werden.

Zu Beginn fiel mir die Bearbeitung der Issues wirklich schwer, da ich sehr viel allein versucht habe und durch fehlendes Wissen große Probleme hatte voranzukommen. Nachdem ich dann aber begann mich mit Problemen direkt an Betreuer oder an Mitstudenten zu wenden, wurde es immer besser. Es war zwar nach wie vor nicht einfach diese Aufgaben zu bewältigen, gerade da ich durch dieses Projekt sehr deutlich merkte, dass es mir beim Programmieren definitiv noch an Erfahrung und Routine fehlt, allerdings konnten viele Probleme durch Nachfragen sehr schnell gelöst werden und viel Zeit gespart werden. Gerade als die Kommunikation untereinander immer besser

wurde und sich nicht nur auf bestimmte Personen beschränkte wurde es immer besser.

Zudem war es auch sehr interessant zu sehen, was für eine unterschiedliche Gruppe an Studenten an diesem Projekt beteiligt waren. Egal ob bezogen auf den Studiengang, auf das Alter oder auf den Wissenstand, gab es insgesamt große Unterschiede, trotzdem hat jeder in gewisser Hinsicht seinen Platz gefunden und konnte positiv an dem Projekt mitwirken. Gerade basierend auf diesem Punkt ist das Projekt für mich ein voller Erfolg. Es war wirklich eine schöne und vor allem lehrreiche Erfahrung, aus der ich sehr viel für meine weitere Zukunft mitnehmen kann. Ich würde SEE definitiv jedem Studenten empfehlen, der bei dem Bachelorprojekt wirklich etwas lernen möchte und dabei eine gute Projektumgebung haben möchte, bei der immer jemand erreichbar ist um zu helfen, man aber trotzdem genug Möglichkeiten bekommt, sich selbst zu entfalten und das zu machen worauf man Lust hat.

Für mich selber konnte ich mitnehmen, dass es wirklich hilfreich ist bei Problemen nicht lange herum zu probieren sondern direkt nach Hilfe zu fragen, da man dadurch meist unglaublich viel Zeit und Nerven sparen kann. Außerdem habe ich auch gemerkt wie lange es manchmal dauern kann, vermeintliche leichte Aufgaben zu lösen und dass sich im Bereich des Programmierens Problem auftun können, die man nie erwartet hätte.

6.11 Sören Untiedt

SEE war eine abenteuerliche Reise in ein größeres Softwareprojekt! Die Auswahl des Bachelorprojekts war für mich eine Mischung aus Bauchgefühl und Begeisterung. Das Thema klang sehr spannend und durch mein privates Interesse an Computerspielen und Visualisierungen, fühlte ich mich direkt angesprochen. Auch die Tatsache, dass einige meiner befreundeten Studienkollegen gleiches Interesse zeigten, bestärkte mich in meiner Entscheidung. Meine Anforderungen an das Projekt waren relativ gering, da ich einfach unvoreingenommen herangegangen bin. Meine Ziele jedoch waren relativ klar, lernen mit C# zu programmieren, die Möglichkeiten und Entwicklung mit Unity zu testen und einen Beitrag zum Gelingen des Projekts zu leisten.

Aller Anfang ist schwer, so war es auch bei dieser neuen Herausforderung. Mit einer neuen — wenn auch nicht gänzlich anders zu den mir bekannten — Entwicklungs-Sprache, einem großen Berg von bereits vorhandenem Code und einer vorher unbekannten 3D-Engine, gab es so einiges zu erforschen. Im Nachgang kann ich sagen, dass ich sowohl in der Programmierung, als auch im Umgang mit Unity, vieles dazu lernen gelernt habe.

Nach einer kurzen Einführungsaufgabe zum Verstehen der Codestruktur, wurden Aufgaben an Gruppen oder einzelne Personen verteilt. Für den Austausch untereinander und eine Übersicht über den Fortschritt, wurde ein wöchentliches Treffen ins Leben gerufen. Dieses Meeting war für viele der einzige Kontaktpunkt zu den anderen Projektteilnehmern, da die Pandemie und die damit einhergehenden Beschränkungen das Gruppengefühl und die Atmosphäre des Projekt etwas gemindert haben. Nichtsdestotrotz habe ich viele neue Leute hierdurch kennen gelernt und auch Freunde dazugewonnen.

Ich konnte die meisten meiner Aufgaben mit Kollegen in enger Zusammenarbeit

bewältigen, was zu dem Lern- auch einen Spaßfaktor generierte. Die uns übertragenen Aufgaben stellten kleinere, aber dennoch wichtige Anpassungen am Gesamtprojekt dar, sodass wir uns selbst verwirklichen konnten und auch im Code des Projekts verewigen konnten. Die Bildung von Kleingruppen kann natürlich zur Folge haben, dass es Abgrenzungen innerhalb der Teilnehmer geben kann, ich denke jedoch, dass sich auch neue Gruppen gebildet haben und das Problem größtenteils vermieden wurde.

Des Weiteren war die Gruppe der Studenten bunt gemischt. Sowohl vom Wissensstand oder Studienzeit, als auch vom Alter oder Studiengang. Dennoch wurden persönliche Kompetenzen und Wünsche bei den Aufgaben berücksichtigt, was dem Projekt und dessen Leitung positiv zuzuschreiben ist. Ich konnte zum Beispiel die Website mitgestalten, denn das liegt auch in meinem privaten Interessenbereich, sodass ich hier meine Vorerfahrung nutzen konnte und diese um ein neues CMS-System erweitern.

Zusammenfassend kann ich sagen, war es eine schöne und lehrreiche Zeit, mit vielen Eindrücken und Erfahrungswerten. Für mich ist es ein gelungenes Projekt, da nicht nur die Beteiligten was dazu gelernt haben, sondern auch in gewisser Weise ein Gruppengefühl entstanden ist und diese Gruppe das Projekt SEE auch wirklich verbessern und weiter entwickeln konnte.

Für mich selbst konnte ich mitnehmen, dass es oft hilfreich sein kann, sich vor Beginn einer Aufgabe umzuhören, ob jemand bereits Erfahrungen in dem Bereich sammeln konnte. Nicht nur einmal konnte mir eine Starthilfe durch ein anderes Gruppenmitglied die Arbeit deutlich erleichtern.

6.12 Jan Woiton

SEE hat Potenzial und ist ein fantastisches Projekt. Zu Beginn hatte ich weder Kenntnisse in C#, C, noch in überhaupt irgendeiner Game-Engine, vor allem nicht in Unity. Allerdings reichten die motivierende Ansprache eines Kommilitonen, ein bisschen Fantasie Richtung Matrix und meine persönliche Faszination für Computerspiele im Allgemeinen vollkommen aus, um mich anzumelden. Während der ersten Startschwierigkeiten, wie beispielsweise dem Fehlen von Kenntnissen in irgendwas bei diesem Projekt, übernahm die Projektleitung die Führung und ich konnte mich gut einarbeiten. Anfänglich gab es einige Hemmschwellen, da ich noch nicht alle Leute des Projekts kannte, aber durch einen Online-Zoom-Gaming-Abend mit der Projektgruppe verschwanden diese vollends und ich merkte, wie aufgeschlossen und motiviert alle sind. Über das gesamte Projekt hinweg war es gängig, am Freitagmorgen um 9 Uhr einem Meeting beizuwohnen, dem sogenannten Weekly, bei dem wir uns gegenseitig unsere Fortschritte präsentierten, Fragen stellen konnten und neue Informationen austauschten. In der Zwischenzeit hatte jeder ein Issue oder eine andere Aufgabe, wie beispielsweise das Vorbereiten einer Präsentation. Aufgrund der durchgehenden Erreichbarkeit der Leitung des Projekts war es natürlich auch möglich, zwischendurch Fragen zu stellen und zeitnah zu klären.

Während meiner persönlichen Reise merkte ich, dass mein Interesse an der Verbreitung und Bekanntmachung von SEE immer weiter wuchs. Unabhängig davon hatte ich allerdings doch Probleme mit der Einarbeitung in das Projekt selbst, da es doch schon sehr groß und für mich persönlich recht unüberschaubar schien. Dabei halfen mir

dann Rainer selbst und auch mehrere Kommilitonen und so kam ich auch mit leichten Problemen sehr gut voran. Währenddessen erhielt ich immer mehr Wissen über C# und Unity, über den Aufbau von Software und dem Bewältigen von Issues. Außerdem erfuhr ich das erste Mal, wie es ist, ein Projekt basierend auf der Vorstellung eines Menschen mit einer Gruppe zu verwirklichen. Die Atmosphäre sowie die Mentalität waren und sind immer noch beeindruckend und haben mir eine Zeit geschenkt, die ich sicher niemals vergessen werde. Auch wenn ich hierbei sehr wenige Issues als erledigt vorweisen kann. Durch meine persönliche Intention SEE bekannt zu machen schlug ich gemeinsam mit zwei Kommilitonen den Bau der Website vor und wir setzten dies auch schnell um. Weiterhin habe ich ein Meeting organisiert, in dem ich Projekt SEE einem Head of IT vorstellen konnte. Nach diesem Gespräch erhielt ich ein letztes Issue und mit einer Mischung aus Freude und Trauer fiebere ich dem Ende des Projektes entgegen.

Abschließend gibt es wenig persönliche Anmerkungen zur Optimierung des Projektes. Meiner Meinung nach sollte man das Wiki ausbauen, denn für andere Studenten, welche mit einem mir ähnlichem Kenntnisstand in dieses Projekt kommen könnten, sollte es eine Literatur zum Nachlesen geben. Des Weiteren soll für die Zukunft auch ein Nutzer SEE benutzen und dies wäre mit einem optimierten Wiki einfacher.

Anfänglich bekamen alle Kommilitonen inkl. mir die gleiche Aufgabe zur Einarbeitung. Das hat mir sehr gut gefallen und hier lernten wir uns Kommilitonen alle schon etwas näher kennen. Darüber hinaus habe ich meine Fähigkeiten im Bereich Kommunikation und Vertrieb ausbauen können und habe Einblicke in Businessmeetings bekommen.

Zusammenfassend ist SEE ein fantastisches Projekt mit einer stabilen Struktur, einem permanenten Fortschritt und aus meiner Sicht ein Projekt, welches hoffentlich in naher Zukunft sehr vielen Menschen bei der Entwicklung von Software unterstützen wird!

7 Fazit und Ausblick

7.1 Bearbeitete Issues

In Tabelle 1 auf Seite 92 im Anhang (Sektion 9) werden alle bearbeiteten GitHub-Issues von allen studentischen Projektteilnehmern alphabetisch und chronologisch aufgelistet.

7.2 Erreichte Ziele

7.2.1 Architekturprüfung

Für die Architekturprüfung wurden große Teile der anfänglich geplanten Aufgaben im zeitlichen Rahmen des Bachelorprojekts umgesetzt. Die geplanten Punkte „Modellierung/Veränderung der Architektur“, „Abbildung von Knoten der Implementierung auf Knoten der Architektur durch *Drag&Drop* (VR) bzw. *Copy&Paste* (Desktop)“, die Mehrbenutzer-Unterstützung und das Anzeigen von Quelltext wurden wie geplant umgesetzt. Für die Darstellung der Veränderungen der Reflexionsanalyse konnten nicht alle geplanten Funktionen umgesetzt werden: Das Hervorheben von neuen und gelöschten Knoten und Kanten sowie das Anzeigen des Quelltextes für Divergenzen wurde nicht

implementiert. Die Punkte „Reaktion auf Reflexionsergebnisse“ und die Integration der Metrik-Charts wurden nicht implementiert. Für die Multibenutzer-Unterstützung wurden dafür mehrere neue Interaktionsmöglichkeiten (Player Actions, siehe Sektion 4.2) neu hinzugefügt, die zu Beginn des Projekts nicht abzusehen waren. Da die Entwicklung davon viel Zeit in Anspruch genommen hat und als wichtiger eingeschätzt wurde, sind die eben genannten Punkte zum Hervorheben nicht umgesetzt geworden.

7.2.2 Evolution

Für die Evolution konnten nahezu alle geplanten Neuerungen implementiert werden. Eine Ausnahme bildet die Multi-Benutzer-Unterstützung, die bisher nur für die Architekturprüfung umgesetzt wurde. So konnte die Benutzeroberfläche maßgeblich verbessert werden und wie geplant über eine wie bei einer digitalen Videowiedergabe bekannten Benutzeroberfläche gesteuert werden. Außerdem wurde ein Wechsel des Animationsframeworks durchgeführt und somit das bisherige iTween durch DoTween ersetzt. Des Weiteren wurde eine performante Animation für die Kanten implementiert.

7.2.3 HoloLens

Die Erweiterung des bestehenden Projekts für die HoloLens 2 ist mit Abschluss des Projekts erfolgreich gewesen. Durch anfängliche Probleme, gegeben durch die Verwendung von Unity 2019, sowie Kompatibilitätsprobleme von eingebetteten Assemblies konnten nicht alle geplanten Neuerungen für die HoloLens implementiert werden. So existiert zwar der Code für den Mehrspielermodus, dieser konnte aber noch nicht auf der HoloLens getestet werden. Im Falle von „Zooming und Panning“ wurde dieses zunächst verworfen, da die Interaktion mit einer City grundsätzlich anders funktioniert. So kann bereits eine City auf der HoloLens beliebig im Raum platziert und skaliert werden und daher konnte ein zusätzliches „Zooming und Panning“ nicht ohne weiteres implementiert werden.

Für die Metrik-Charts und die Quellcode-Anzeige müssen noch in dem neuen UI-Framework (Sektion 4.1.1) die HoloLens-Ausprägungen umgesetzt werden. Für die HoloLens existieren bereits vordefinierte und für die HoloLens optimierte Benutzerelemente im MRTK, die für die Darstellung von Menüs und Inhalt, wie z. B. Charts oder Quelltext, zuständig sind. Die Selektion von Elementen wurde durch Eyetracking und durch das Zeigen mit dem Finger umgesetzt und zeigt im aktuellen Stand den Namen des ausgewählten Elements auf einem darüber schwebenden Label an. Insgesamt ist jedoch eine Verwendung von SEE auf der HoloLens 2 möglich und bietet auch erste Interaktionsmöglichkeiten, die in Zukunft noch erweitert werden können.

7.3 Fazit

Insgesamt kann das Projekt als Erfolg bezeichnet werden. Für alle in Sektion 2.3 genannten Teilbereiche der Ziele konnten die absolut notwendigen Implementierungen, sowie einige weitere Funktionen implementiert werden. Für die Architekturprüfung

und Evolution konnten nahezu alle geplanten und zusätzlich noch aus dem Verlauf des Projekts heraus neu entstandenen Funktionalitäten hinzugefügt werden. Für die Unterstützung der HoloLens 2 konnte nicht alles implementiert werden, was aber an der Anzahl der dafür zugewiesen Personen im Vergleich zu den anderen Zielkategorien nachvollziehbar ist. Jeder Teilnehmer hat etwas sinnvolles gelernt und SEE ist im Vergleich zur Ausgangssituation stark verbessert und erweitert worden. Somit wurde dieses Projekt sehr zufriendenstellend beendet, wobei SEE natürlich auch über das Bachelorprojekt hinaus weiterentwickelt wird.

7.4 Ausblick

Abbildung 53: Verstörender Avatar.

Für die Zukunft können noch viele weitere Aspekte bedacht werden, um das Potential von SEE weiter auszubauen. Momentan lassen sich Cities nur aus dem Unity-Editor heraus laden und konfigurieren. Stattdessen sollte eine City dynamisch aus der Anwendung heraus geladen und gestartet werden können. Für einen sinnvollen Testbetrieb mit echten Anwendern ist das unabdingbar, damit diese ihre eigenen Cities laden, analysieren und besprechen können. Es wird auch die Funktion des Speichern des aktuellen Standes oder die Möglichkeit, Notizen zu erstellen (Whiteboard oder Sticky-Notes), um Informationen schnell festzuhalten, benötigt. Für eine noch bessere Integration in den Arbeitsablauf des Entwickelns und Debuggens wäre auch die Integration von SEE direkt in die verwendete IDE sinnvoll. So könnte man aus der Entwicklungsumgebung heraus den entsprechenden Knoten in SEE öffnen und die Abhängigkeiten prüfen oder die Implementierung mit der Architektur vergleichen, ohne den Knoten erst in SEE suchen zu müssen.

Im Mehrbenutzer-Modus sollten die Avatare in Form von weniger abschreckenden[57] (siehe Abbildung 53) Platzhaltern verbessert werden. Außerdem kann durch die Übertragung von Mimik auch das kollaborative Arbeiten verbessert werden, da so eine angenehmere und natürlichere Umgebung zum Zusammenarbeiten entsteht. Wie das umgesetzt werden kann, muss noch weiter untersucht werden, da bisher kein frei erhältliches Gerät die gleichzeitige Aufnahme von Mimik während der Nutzung unterstützt. Außerdem wird noch nicht der Multi-Benutzer-Modus innerhalb der Evolution unterstützt, welche das Betrachten von zeitlichen abhängigen Veränderungen der City mit mehreren Personen ermöglicht. Eine weitere Idee, die in Betracht gezogen wird, ist die Integration und Visualisierung von Daten aus Versionskontrollsystemen (z. B. *Git*) und Issue-Trackern (z. B. *GitHub*).

In Anbetracht der zu unterstützenden Hardware bietet sich neben der noch tieferen

[57] „Verstörend“ laut Feedback von mehreren externen Testern, die die Platzhalter-Avatare gesehen haben.

Integration der HoloLens 2 an, weitere AR-kompatible Geräte wie Smartphones zu unterstützen. Diese bieten die Möglichkeit, mit der Kamera AR-Inhalte live auf dem Display anzuzeigen und damit zu interagieren. Apple bietet in iOS das ARKit und Google bietet auf Android das ARCore an, um AR-Anwendungen zu entwickeln. Ob diese in Unity integriert werden können, sollte auch untersucht werden.

8 Danksagung

An dieser Stelle möchten wir uns bei all denjenigen bedanken, die uns während des Projektes so tatkräftig unterstützt haben. Ein besonderer Dank geht an die Leitung dieses Projekts: Prof. Dr. Rainer Koschke und Marcel Steinbeck. Dank ihrer ausgezeichneten Organisation, Zuverlässigkeit und Hilfsbereitschaft war es einem jeden Teilnehmer des Projektes zu jeder Zeit möglich, in kürzester Zeit ein Gespräch zu suchen oder bei einem Problem Unterstützung erwarten zu können. Weiterhin gilt der Dank allen Teilnehmern dieses Projektes. Über die Dauer des Projektes sorgten alle für ein positives und angenehmes Arbeitsklima und alle zeigten sich äußerst engagiert und hilfsbereit.

9 Anhang

9.1 Issue-Tabelle

Tabelle 1: Die von jeder Person bearbeiteten Issues.

Person	ID	Issue-Titel
Moritz Blecker	*#61*	Show source code
	#115	User can add, scale, and name new node
	#186	AddEdgeAction should become multiplayer capable
	#187	Draw new Edges should be Multiplayer compatible
	#213	Make undo global
	#253	GlobalUndoHistory does not act Deterministic
	N/A	Abschlusspräsentation
Thore Frenzel	*#115*	User can add, scale, and name new node
	#119	Evolution: The metrics in the CSV files should be read in and applied to the visual graph nodes
	#120	Evolution: The node types contained in the GXL and to be visualized should be selected for the animation of the evolution
	#138	Im- and export user settings, such as selected and deselected nodetypes into a .json File

	#180	Deletion of nodes or edges needs an Undo or a confirmation dialogue
	#210	Undo and Redo AddEdge
	#212	Undo and Redo EditNode
	#213	Make undo global
	#226	PlayerMenu won´t be updated after undoing the last action
	#253	GlobalUndoHistory does not act Deterministic
	N/A	Organisation des Abschlussberichts
Thorsten Friedewold	*#121*	Incoming and outgoing edges should be moved along with a node
	#153	Hide/show edges
	#177	Changing the animation order for edges and nodes
	#231	Performance degradation due to edge animation for short animation lags
	#251	Improve the performance of edge animation
	#259	If a node is scaled, its incoming and outgoing edges must be adjusted
	#332	Calculation of the edges after scaling a node is not working if the user scales an inner node
Falko Galperin	*#61*	Show source code
	#100	Label above Node
	#132	Integrate Mixed Reality Toolkit (MRTK)
	#133	HoloLens: Make code cities movable and scalable
	#135	HoloLens: Source.Name label above gazed GameObject
	#148	Allow creating SEECity GameObject via Unity Inspector
	#166	Implement consistent UI
	#194	Verify no rendered code cities are committed
	#243	Scrolling via mouse wheel is too slow in UI elements
	#245	Syntax highlighting for source code shown in source viewers would be useful
	#252	Allow searching for nodes by name
	#267	UI: Implement notifications
	#317	Add script to fix CodeFacts.gxl paths
	N/A	Fernsteuerung von SEE am Projekttag
Lino Goedecke	*#115*	User can add, scale, and name new node

	#119	Evolution: The metrics in the CSV files should be read in and applied to the visual graph nodes
	#120	Evolution: The node types contained in the GXL and to be visualized should be selected for the animation of the evolution
	#138	Im- and export user settings, such as selected and deselected nodetypes into a .json File
	#180	Deletion of nodes or edges needs an Undo or a confirmation dialogue
	#204	Rework network-deleteAction
	#224	DeleteAction-UndoAnimation
	#255	ScaleNode-Undo-Redo
Leonard Haddad	*#5*	Use and decoration of own cubes
	#122	Evolution: Better highlighting of new, deleted, and changed nodes
	#124	Evolution: Integration of metric charts
	#166	Implement consistent UI
	N/A	Abschlusspräsentation
Simon Leykum	*#123*	Evolution: Better UI for
	#153	Hide/show edges
	#215	The play buttons and markers and the revision slider for a code-city evolution are no longer available
	#223	Creating an edge when no edge layout was selected in the inspector causes null pointer dereference
	#232	NullReferenceException when moving through graph with slider
Christian Müller	*#132*	Integrate Mixed Reality Toolkit (MRTK)
	#133	HoloLens: Make code cities movable and scalable
	#135	HoloLens: Source.Name label above gazed GameObject
	#166	Implement consistent UI
	#168	HoloLens: Fix performance issues
	#178	[HoloLens] [UWP] Can not be built in release and master mode
	#182	Hololens AppBar does not show up
	#228	It is not possible to build for Hololens with Valve dependencies
Nick Seedorf	*#114*	Replace iTween by newer animation framework
	#154	Highlighting selected edges
	#245	Syntax highlighting for source code shown in source viewers would be useful

	N/A	Website
Robin Theiß	*#5*	Use and decoration of own cubes
	#184	DeleteAction should be multiplayer capable
	#117	Existing nodes and edges can be removed by the user
Sören Untiedt	*#114*	Replace iTween by newer animation framework
	#154	Highlighting selected edges
	#245	Syntax highlighting for source code shown in source viewers would be useful
	N/A	Website
Jan Woiton	*#116*	User can add new edge from node to node
	#246	If a node was added, its parent should be visualized as an inner node
	N/A	Website

9.2 Literatur

[Ans+13] C. Anslow u. a. „SourceVis: Collaborative software visualization for co-located environments“. In: *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. Sep. 2013, S. 1–10. DOI: 10.1109/VISSOFT.2013.6650527.

[Ans14] C. Anslow. „Reflections on Collaborative Software Visualization in Co-located Environments“. In: *2014 IEEE International Conference on Software Maintenance and Evolution*. 2014, S. 645–650. DOI: 10.1109/ICSME.2014.115.

[AWP97] Keith Andrews, Josef Wolte und Michael Pichler. „Information Pyramids: A New Approach to Visualising Large Hierarchies“. In: *IEEE Conference on Visualization*. 1997, S. 49–52. ISBN: 0-8186-8262-0.

[Bal+04] Michael Balzer u. a. „Software Landscapes: Visualizing the Structure of Large Software Systems“. In: *IEEE TCVG Symposium on Visualization*. Eurographics Association, Jan. 2004, S. 261–266. DOI: 10.2312/VisSym/VisSym04/261-266.

[Bau+20] David Baum u. a. *Identifying Usability Issues of Software Analytics Applications in Immersive Augmented Reality*. 2020. arXiv: 2008.06099 [cs.HC].

[BHK15] Hamid Abdul Basit, Muhammad Hammad und Rainer Koschke. „A survey on goal-oriented visualization of clone data“. In: IEEE, 2015, S. 46–55.

[BK01a] Sarita Bassi und Rudolf K. Keller. „Software visualization tools: Survey and analysis“. In: IEEE. 2001, S. 7–17.

[BK01b] Sarita Bassil und Rudolf K Keller. „A Qualitative and Quantitative Evaluation of Software Visualization Tools“. In: *Proceedings of the Workshop on Software Visualization*. IEEE. 2001, S. 33–37.

[Blo18] Joshua Bloch. *Effective Java*. 3. Aufl. Boston, MA: Addison-Wesley, 2018. ISBN: 978-0-13-468599-1. URL: https://www.safaribooksonline.com/library/view/effective-java-third/9780134686097/.

[Boh10] Johannes Bohnet. „Visualization of Execution Traces and itsApplication to Software Maintenance“. Dissertation. Hasso-Plattner-Institut, Universität Potsdam, Okt. 2010. URL: https://hpi.de/fileadmin/user_upload/fachgebiete/doellner/publications/2011/BOH11/2011-phd-dissertation-bohnet.pdf.

[Bow+99] Doug A. Bowman u. a. „Maintaining Spatial Orientation During Travel in an Immersive Virtual Environment“. In: *Presence: Teleoper. Virtual Environ.* 8.6 (1999), S. 618–631. ISSN: 1054-7460. DOI: 10.1162/105474699566521. URL: http://dx.doi.org/10.1162/105474699566521.

[BSB15] Gergõ Balogh, Attila Szabolics und Árpád Beszédes. „CodeMetropolis: Eclipse over the city of source code“. In: Sep. 2015, S. 271–276. DOI: 10.1109/SCAM.2015.7335425.

[Cap+17] Nicola Capece u. a. „Visualising a Software System as a City Through Virtual Reality“. In: *Augmented Reality, Virtual Reality, and Computer Graphics*. Hrsg. von Lucio Tommaso De Paolis, Patrick Bourdot und Antonio Mongelli. Cham: Springer International Publishing, 2017, S. 319–327. ISBN: 978-3-319-60928-7.

[CG08] Sheelagh Carpendale und Yaser Ghanam. *A survey paper on software architecture visualization*. Technical Report 2008-906-19. University of Calgary, 2008. DOI: http://dx.doi.org/10.11575/PRISM/30491.

[Cha+02] Stuart M. Charters u. a. „Visualisation for Informed Decision Making; from Code to Components“. In: New York, NY, USA: Association for Computing Machinery, 2002, S. 765–772. ISBN: 1581135564. DOI: 10.1145/568760.568891. URL: https://doi.org/10.1145/568760.568891.

[Cha+98] S. S. Chance u. a. „Locomotion Mode Affects the Updating of Objects Encountered During Travel: The Contribution of Vestibular and Proprioceptive Inputs to Path Integration“. In: *Presence: Teleoper. Virtual Environ.* 7.2 (1998), S. 168–178. ISSN: 1054-7460. DOI: 10.1162/105474698565659.

[CSD93] Carolina Cruz-Neira, Daniel J. Sandin und Thomas A. DeFanti. „Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE“. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. Anaheim, CA: Association for Computing Machinery, 1993, S. 135–142. ISBN: 0897916018. DOI: 10.1145/166117.166134. URL: https://doi.org/10.1145/166117.166134.

[CZ11] Pierre Caserta und Olivier Zendra. „Visualization of the static aspects of software: A survey“. In: 17.7 (2011), S. 913–933.

[Die01] Stephan Diehl, Hrsg. *Software Visualization: International Seminar Dagstuhl Castle, Germany*. Lecture Notes in Computer Science. Springer, 2001.

[Die10] Stephan Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, 2010.

[DL08a] M. D'Ambros und M. Lanza. „A Flexible Framework to Support Collaborative Software Evolution Analysis“. In: *2008 12th European Conference on Software Maintenance and Reengineering*. 2008, S. 3–12. DOI: 10.1109/CSMR.2008.4493295.

[DL08b] Marco D'Ambros und M. Lanza. „Churrasco : Supporting Collaborative Software Evolution Analysis“. In: 2008.

[DL10] Marco D'Ambros und Michele Lanza. „Distributed and Collaborative Software Evolution Analysis with Churrasco“. In: *Science of Computer Programming* 75.4 (2010). Experimental Software and Toolkits (EST 3): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT 2008), S. 276–287. ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2009.07.005. URL: https://www.sciencedirect.com/science/article/pii/S0167642309001105.

[EKP16] C. Ebert, M. Kuhrmann und R. Prikladnicki. „Global software engineering: evolution and trends“. In: 2016, S. 144–153.

[Fit+13] Florian Fittkau u. a. „Live trace visualization for comprehending large software landscapes: The ExplorViz approach“. In: *VISSOFT*. 2013, S. 1–4. DOI: 10.1109/VISSOFT.2013.6650536.

[Fit+15] Florian Fittkau u. a. „Comparing Trace Visualizations for Program Comprehension through Controlled Experiments“. In: 2015, S. 266–276. DOI: 10.1109/ICPC.2015.37.

[Fit15] Florian Fittkau. „Live Trace Visualization for System and Program Comprehension in Large Software Landscapes“. Dissertation. Christian-Albrechts-Universität zu Kiel, 2015. ISBN: 3739207167.

[FKH15a] Florian Fittkau, Alexander Krause und Wilhelm Hasselbring. „Exploring software cities in virtual reality“. In: 2015, S. 130–134.

[FKH15b] Florian Fittkau, Alexander Krause und Wilhelm Hasselbring. „Hierarchical software landscape visualization for system comprehension: A controlled experiment“. In: 2015, S. 36–45. DOI: 10.1109/VISSOFT.2015.7332413.

[FKH17] Florian Fittkau, Alexander Krause und Wilhelm Hasselbring. „Software landscape and application visualization for system comprehension with ExplorViz“. In: *Information and Software Technology* 87 (2017), S. 259–277. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2016.07.004.

[Fou95] David Foulser. „IRIS Explorer: A Framework for Investigation“. In: *SIGGRAPH Comput. Graph.* 29.2 (Mai 1995), S. 13–16. ISSN: 0097-8930. DOI: 10.1145/204362.204365. URL: https://doi.org/10.1145/204362.204365.

[FPV17] M. Ferenc, I. Polasek und J. Vincur. „Collaborative Modeling and Visualization of Software Systems Using Multidimensional UML“. In: *2017 IEEE Working Conference on Software Visualization (VISSOFT)*. 2017, S. 99–103. DOI: 10.1109/VISSOFT.2017.19.

[FRH15] Florian Fittkau, Sascha Roth und Wilhelm Hasselbring. „Explorviz: visual runtime behavior analysis of enterprise application landscapes“. In: 2015, S. 1–13.

[FRW17] F. Fernandes, C. S. Rodrigues und C. Werner. „Dynamic Analysis of Software Systems through Virtual Reality“. In: In Spanish. Nov. 2017, S. 331–340. DOI: 10.1109/SVR.2017.52.

[FSH14] Florian Fittkau, Phil Stelzer und Wilhelm Hasselbring. „Live Visualization of Large Software Landscapes for Ensuring Architecture Conformance“. In: 2014. DOI: 10.1145/2642803.2642831. URL: https://doi.org/10.1145/2642803.2642831.

[GK09] Nils Göde und Rainer Koschke. „Incremental Clone Detection“. In: Best Paper Award von 70 eingereichten Beiträgen. 2009, S. 219–228.

[GR18] Johannes Ganser und Marc-Oliver Rüdel. „Orientierung in VR-basierten Software-Cities mit hierarchisch gebündelten Abhängigkeiten“. Bachelorarbeit. Fachbereich 3, Universität Bremen, 2018.

[Ham+20] Muhammad Hammad u. a. „A systematic mapping study of clone visualization“. In: 37 (2020), S. 100266. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2020.100266. URL: http://www.sciencedirect.com/science/article/pii/S1574013719302679.

[Hol06] Danny Hubertus Rosalia Holten. „Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data“. In: 12.5 (Sep. 2006), S. 741–748. ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.147.

[Hol09] Danny Hubertus Rosalia Holten. „Visualization of graphs and trees for software analysis“. Diss. Technical University of Delft, 2009.

[Ise+10] P. Isenberg u. a. „An exploratory study of co-located collaborative visual analytics around a tabletop display“. In: *2010 IEEE Symposium on Visual Analytics Science and Technology* (2010), S. 179–186.

[JDP20] Florian Jung, Veronika Dashuber und Michael Philippsen. „Towards Collaborative and Dynamic Software Visualization in VR“. In: *Proceedings of the International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP*. INSTICC. SciTePress, 2020, S. 149–156. ISBN: 978-989-758-402-2. DOI: 10.5220/0008945201490156.

[JS91] Brian Johnson und Ben Shneiderman. „Tree-Maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures“. In: *Proceedings of the Conference on Visualization*. IEEE Computer Society Press, 1991, S. 284–291. ISBN: 0-8186-2245-8. URL: http://dl.acm.org/citation.cfm?id=949607.949654.

[Kha+17] Pooya Khaloo u. a. „Code Park: A New 3D Code Visualization Tool“. In: *Software Visualization (VISSOFT), 2017 IEEE Working Conference on*. IEEE. 2017, S. 43–53.

[KM00] Claire Knight und Malcolm Munro. „Virtual but visible software“. In: IEEE. 2000, S. 198–205.

[Kos03a] R. Koschke. „Software Visualization in Software Maintenance, Reverse Engineering, and Reengineering: A Research Survey“. In: *Journal on Software Maintenance and Evolution* 15 (März 2003), S. 87–109.

[Kos03b] Rainer Koschke. „Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey“. In: 15.2 (2003), S. 87–109.

[Kot+05] Blazej Kot u. a. „Information Visualisation Utilising 3D Computer Game Engines Case Study: A Source Code Comprehension Tool“. In: *Proceedings of the 6th ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction: Making CHI Natural*. CHINZ '05. Auckland, New Zealand: Association for Computing Machinery, 2005, S. 53–60. ISBN: 1595930361. DOI: 10.1145/1073943.1073954. URL: https://doi.org/10.1145/1073943.1073954.

[Lei+99] J. Leigh u. a. „Visualization in teleimmersive environments“. In: *Computer* 32.12 (1999), S. 66–73. DOI: 10.1109/2.809253.

[Lim+19] Daniel Limberger u. a. „Advanced Visual Metaphors and Techniques for Software Maps“. In: Sep. 2019, S. 1–8. ISBN: 978-1-4503-7626-6. DOI: 10.1145/3356422.3356444.

[Mal+01] J. I. Maletic u. a. „Visualizing object-oriented software in virtual reality“. In: Mai 2001, S. 26–35. DOI: 10.1109/WPC.2001.921711.

[MBN18] Leonel Merino, Alexandre Bergel und Oscar Nierstrasz. „Overcoming Issues of 3D Software Visualization through Immersive Augmented Reality“. In: 2018, S. 54–64.

[Mel+15] Natalia Melnikova u. a. „CAVE 3D: Software Extensions for Scientific Visualization of Large-scale Models“. In: *Procedia Computer Science* 66 (2015). 4th International Young Scientist Conference on Computational Science, S. 679–688. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.11.077. URL: https://www.sciencedirect.com/science/article/pii/S1877050915034262.

[Mer+17a] Leonel Merino u. a. „CityVR: Gameful Software Visualization“. In: *(TD Track)*. 2017, S. 633–637.

[Mer+17b] Leonel Merino u. a. „On the Impact of the Medium in the Effectiveness of 3D Software Visualizations“. In: *Software Visualization (VISSOFT), 2017 IEEE Working Conference on*. IEEE. 2017, S. 11–21.

[Mer+18] Leonel Merino u. a. „A Systematic Literature Review of Software Visualization Evaluation“. In: 144 (Juni 2018). DOI: 10.1016/j.jss.2018.06.027.

[Mer+19] Leonel Merino u. a. „PerfVis: Pervasive Visualization in Immersive Augmented Reality for Performance Awareness“. In: *ACM/SPEC International Conference on Performance Engineering*. 2019, S. 13–16.

[MFM03] Andrian Marcus, Louis Feng und Jonathan I. Maletic. „3D Representations for Software Visualization“. In: New York, NY, USA: Association for Computing Machinery, 2003, S. 27–36. ISBN: 1581136420. DOI: 10.1145/774833.774837. URL: https://doi.org/10.1145/774833.774837.

[Nov+13] Renato Lima Novais u. a. „Software Evolution Visualization: A Systematic Mapping Study“. In: 55.11 (Nov. 2013), S. 1860–1883. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2013.05.008. URL: http://dx.doi.org/10.1016/j.infsof.2013.05.008.

[Oga+17] Katsuya Ogami u. a. „Using High-Rising Cities to Visualize Performance in Real-Time“. In: *Software Visualization (VISSOFT), 2017 IEEE Working Conference on*. IEEE. 2017, S. 33–42.

[Pan+07a] T. Panas u. a. „Communicating Software Architecture using a Unified Single-View Visualization“. In: *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*. 2007, S. 217–228. DOI: 10.1109/ICECCS.2007.20.

[Pan+07b] Thomas Panas u. a. „Communicating software architecture using a unified single-view visualization“. In: *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*. IEEE. 2007, S. 217–228.

[PBG03] Thomas Panas, Rebecca Berrigan und John Grundy. „A 3d metaphor for software production visualization“. In: IEEE. 2003, S. 314–319.

[PBS93] Blaine A. Price, Ronald M. Baecker und Ian S. Small. „A Principled Taxonomy of Software Visualization“. In: *Journal of Visual Languages & Computing* 4.3 (1993), S. 211–266. ISSN: 1045-926X. DOI: https://doi.org/10.1006/jvlc.1993.1015. URL: https://www.sciencedirect.com/science/article/pii/S1045926X83710153.

[PR15] Akanksha Prakash und Wendy A Rogers. „Why Some Humanoid Faces Are Perceived More Positively Than Others: Effects of Human-Likeness and Task“. In: *International Journal of Social Robotics* 7.2 (Apr. 2015), S. 309–331. DOI: doi:10.1007/s12369-014-0269-4.

[RCB07] Bernhard E. Riecke, Douglas W. Cunningham und Heinrich H. Bülthoff. „Spatial updating in virtual reality: the sufficiency of visual information“. In: *Psychological Research* 71.3 (Mai 2007), S. 298–313. ISSN: 1430-2772. DOI: 10.1007/s00426-006-0085-z. URL: https://doi.org/10.1007/s00426-006-0085-z.

[RGK18] M. Rüdel, J. Ganser und R. Koschke. „A Controlled Experiment on Spatial Orientation in VR-Based Software Cities“. In: Sep. 2018, S. 21–31. DOI: 10.1109/VISSOFT.2018.00011.

[RP04] Roy A Ruddle und Patrick Péruch. „Effects of proprioceptive feedback and environmental characteristics on spatial learning in virtual environments“. In: *International Journal of Human-Computer Studies* 60.3 (2004), S. 299–326. ISSN: 1071-5819. DOI: https://doi.org/10.1016/j.ijhcs.2003.10.001. URL: http://www.sciencedirect.com/science/article/pii/S1071581903001733.

[RPJ99] Roy A Ruddle, Stephen J Payne und Dylan M Jones. „Navigating large-scale virtual environments: what differences occur between helmet-mounted and desk-top displays?“ In: *Presence: Teleoperators & Virtual Environments* 8.2 (1999), S. 157–168.

[RSM92] J. Wesley Regian, Wayne L. Shebilske und John M. Monk. „Virtual Reality: An Instructional Medium for Visual-Spatial Tasks“. In: *Journal of Communication* 42.4 (1992), S. 136–149. ISSN: 1460-2466. DOI: 10.1111/j.1460-2466.1992.tb00815.x. URL: http://dx.doi.org/10.1111/j.1460-2466.1992.tb00815.x.

[SB17] Andreas Schreiber und Marlene Brüggemann. „Interactive Visualization of Software Components with Virtual Reality Headsets“. In: *Software Visualization (VISSOFT), 2017 IEEE Working Conference on*. IEEE. 2017, S. 119–123.

[SK21] Marcel Steinbeck und Rainer Koschke. „TinySpline: A Small, yet Powerful Library for Interpolating, Transforming, and Querying NURBS, B-Splines, and Bézier Curves“. In: *IEEE International Conference on Software Analysis, Evolution and Reengineering*. Accepted. IEEE Computer Society Press, 2021.

[SKR19a] Marcel Steinbeck, Rainer Koschke und Marc-Oliver Rüdel. „Comparing the EvoStreets Visualization Technique in Two-and Three-Dimensional Environments: A Controlled Experiment“. In: 2019, S. 231–242. DOI: 10.1109/ICPC.2019.00042.

[SKR19b] Marcel Steinbeck, Rainer Koschke und Marc-Oliver Rüdel. „Movement Patterns and Trajectories in Three-Dimensional Software Visualization“. In: 2019, S. 163–174. DOI: 10.1109/SCAM.2019.00027.

[SKR20] Marcel Steinbeck, Rainer Koschke und Marc-Oliver Rüdel. „How EvoStreets Are Observed in Three-Dimensional and Virtual Reality Environments“. In: 2020, S. 332–343. DOI: 10.1109/SANER48275.2020.9054802.

[SL10] Frank Steinbrückner und Claus Lewerentz. „Representing development history in software cities“. In: ACM, 2010, S. 193–202.

[Ste13] Frank Steinbrückner. „Consistent Software Cities: supporting comprehension of evolving software systems“. Diss. Cottbus: Brandenburgischen Technischen Universität Cottbus, Juni 2013. URL: https://opus4.kobv.de/opus4-btu/frontdoor/index/index/docId/1681.

[Ste20] Marcel Steinbeck. „Mining Version Control Systems and Issue Trackers with LibVCS4j“. In: 2020, S. 647–651.

[Str+13] E. Stroulia u. a. „Interactive Exploration of Collaborative Software-Development Data“. In: *2013 IEEE International Conference on Software Maintenance*. 2013, S. 504–507. DOI: 10.1109/ICSM.2013.102.

[SWD18] Willy Scheibel, Christopher Weyand und Jürgen Döllner. „EvoCells - A Treemap Layout Algorithm for Evolving Tree Data“. In: *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. 2018, S. 273–280.

[TC09a] A. R. Teyseyre und M. R. Campo. „An Overview of 3D Software Visualization“. In: 15.1 (Jan. 2009), S. 87–105. ISSN: 1941-0506. DOI: 10.1109/TVCG.2008.86.

[TC09b] Alfredo R. Teyseyre und Marcelo R. Campo. „An overview of 3D software visualization“. In: 15.1 (2009), S. 87–105.

[Wal+13] Jan Waller u. a. „Synchrovis: 3D visualization of monitoring traces in the city metaphor for analyzing concurrency“. In: 2013, S. 1–4. DOI: 10.1109/VISSOFT.2013.6650520.

[WL07] R. Wettel und M. Lanza. „Visualizing Software Systems as Cities“. In: *IEEE International Workshop on Visualizing Software for Understanding and Analysis*. Juni 2007, S. 92–99. DOI: 10.1109/VISSOF.2007.4290706.

[WL08a] R. Wettel und M. Lanza. „Visual Exploration of Large-Scale System Evolution“. In: *2008 15th Working Conference on Reverse Engineering*. Okt. 2008, S. 219–228. DOI: 10.1109/WCRE.2008.55.

[WL08b] Richard Wettel und Michele Lanza. „CodeCity: 3D Visualization of Large-scale Software“. In: *Companion of the 30th International Conference on Software Engineering*. ICSE Companion '08. Leipzig, Germany: ACM, 2008, S. 921–922. ISBN: 978-1-60558-079-1. DOI: 10.1145/1370175.1370188. URL: http://doi.acm.org/10.1145/1370175.1370188.

[WWB97] J. Wood, H. Wright und K. Brodlie. „Collaborative visualization“. In: *Proceedings. Visualization '97 (Cat. No. 97CB36155)* (1997), S. 253–259.

[ZKH19] Christian Zirkelbach, Alexander Krause und Wilhelm Hasselbring. *Hands-On: Experiencing Software Architecture in Virtual Reality*. Research Report 1809. Christian-Albrechts-Universität zu Kiel, Jan. 2019. URL: http://eprints.uni-kiel.de/45728/.