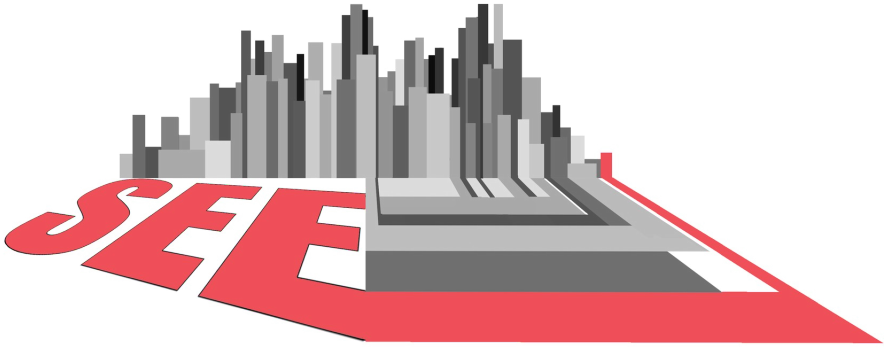


Projektbericht Bachelorprojekt SEE 2022/2023

Thilo Beckord
William Behnke
Zeynep Dilara Degerliyurt
Linus Henkensiefken
Max Julian Herrmann
Alexander Kaiser
Hannes Lennart Kuß
Yvo Muskulus
Ferdinand Rohlfing
Jonas Schramm
Mahmoud Siai
Zhen Yu

15. September 2023

Inhaltsverzeichnis

1 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.

2 Zusammenfassung

William Behnke; Max Herrmann

Der Bericht beschreibt das Bachelorprojekt SEE, das die kollaborative Softwareentwicklung und das Verständnis komplexer Software unterstützen soll. SEE ermöglicht die gemeinsame Analyse und Diskussion von Software in verteilten Teams und ermöglicht die Visualisierung von Softwaredaten im Graphenformat mithilfe der *Code City Metapher*. Das Projekt hatte das Ziel, die bestehende Basis von SEE um nützliche Funktionalitäten zu erweitern und bestehende Problematiken zu beheben, um langfristig weiterhin für die Softwareentwicklung und das Verständnis komplexer Software eingesetzt werden zu können.

Im Rahmen des Projekts wurden verschiedene Inhalte implementiert und Bereiche verbessert, darunter folgende:

Sound Framework	Runtime Transform Gizmos
Charakteranimation und Steuerung	Spiegel
OpenCV Facemask und FaceCam	Runtime Configuration Menu
Solving Reflexion Divergences	Gradual Edge Animations
Incremental Layout	VRIK
HTC FacialTracker	AutoHand
FACSvatar	LiveDocumentation
Port zu Linux und MacOS	

Die Implementierung dieser Inhalte war erfolgreich, und die Ziele des Projekts wurden vollständig erreicht.

Das Bachelorprojekt ermöglichte den Teilnehmern das Erlernen von Projekt- und Zeitmanagement, die Bedeutung guter Programmierkonventionen und den Einsatz geeigneter Programmierwerkzeuge. SEE wurde um Komponenten zur Verbesserung des Nutzererlebnisses erweitert, und einige bestehende Probleme konnten behoben werden.

Für die Zukunft bietet SEE weiterhin eine Plattform für die Erforschung und Evaluation neuer Visualisierungsmethoden für Software sowohl in Desktop- als auch in VR-Umgebungen.

3 Einleitung

3.1 Motivation

Ferdinand Rohlfing

In den letzten Jahren ist die Softwareentwicklung immer umfangreicher geworden, wodurch die Zusammenarbeit in Entwicklerteams an Bedeutung gewonnen hat. Die Zusammenarbeit bei der Entwicklung bietet viele Vorteile, stellt die Teams aber auch vor einige Herausforderungen.

- **Komplexität von Software:** Den Quellcode anderer Entwickler vollständig zu verstehen ist oft zu aufwendig und nicht unbedingt zielführend. Vielmehr ist meist ein Austausch über Pakete, Komponenten und Aufbaustrukturen zwischen diesen Komponenten auf einer höheren Abstraktionsebene notwendig.
- **Paralleles Arbeiten:** Das Arbeiten mit Tools wie Screensharing, z.B. in Zoom, oder Versionskontrollsystemen bieten zwar die Möglichkeit, dieselben Elemente wie andere Entwickler zu sehen, ermöglicht jedoch keine Möglichkeit einzugreifen und aktuelle Änderungen am eigenen Quellcode zu berücksichtigen, ohne umständliches „Pushen“ und „Pullen“ in der Versionskontrolle.
- **Kommunikationskanäle:** Desktop-Umgebungen, Webcams und Headsets unterstützen zwar die grundlegende Kommunikation zwischen Entwicklern, aber es gibt keine Möglichkeit Software „anzufassen“ oder die Architektur gemeinsam im Raum zu betrachten.

Obwohl der Trend der steigenden Relevanz bestehen bleibt, gibt es bisher keine umfassenden Lösungen für die genannten Probleme.
Aus diesen Herausforderungen entwickelt die AG Softwaretechnik der Universität Bremen um Prof. Dr. Rainer Koschke die Software SEE - Software Engineering Experience - die kollaboratives Arbeiten an Software sowohl in Desktopumgebungen als auch in AR und VR ermöglichen soll.

Das Ziel des Bachelorprojektes SEE war daher, die bestehende Basis des bereits existierenden Projektes um viele nützliche Features zu erweitern, so dass SEE langfristig für die kollaborative Softwareentwicklung und das bessere Verständnis komplexer Software eingesetzt werden kann. [1]

[1] Das Kapitel Motivation wurde an [Ble+21] angelehnt.

3.2 SEE

William Behnke

SEE (*Software Engineering Experience*) ist ein *Software Engineering*-Tool zur Visualisierung von Software-Metriken. Es unterstützt verteilte Teams bei der gemeinsamen Analyse von Software indem es Multi-User-Funktionalität einschließlich Sprachchat bietet, mit dem Teammitglieder auf natürliche Weise kommunizieren können, während sie die Software untersuchen. SEE verwendet Graphen in Form von Code City's, um Softwaredaten darzustellen.

Ein langfristiges Ziel von SEE ist es, die Kommunikation und Kooperation zwischen Teammitgliedern zu verbessern, räumliche Distanzen zu überbrücken und das Verständnis von Software in verteilten Teams zu erleichtern.

SEE bietet Entwicklern erweiterte Fähigkeiten, die Qualität ihrer Software zu verstehen, einen umfassenden Überblick zu erhalten und komplexe Architekturstrukturen zu navigieren. Ziel ist es, Software-Entwicklungsteams durch die Bereitstellung einer intuitiven und informativen Umgebung zu unterstützen. SEE wird an der Universität Bremen in Zusammenarbeit mit Axivion[2] entwickelt, einem Unternehmen, das auf statische Codeanalyse und Software-Architekturverifikation spezialisiert ist.

Generell ermöglicht SEE die gemeinsame Analyse und Diskussion von Software in verteilten Teams. Es erleichtert die Zusammenarbeit, verbessert das Verständnis der Softwarearchitektur und fördert die effiziente Kommunikation zwischen den Teammitgliedern.

Zu Beginn unseres Projekts war See bereits eine funktionierende Visualisierungssoftware, die mithilfe einer ausgeklügelten Umgebung und Code City's ihren Grundzweck erfüllte. Wir waren begeistert, auf einer so soliden Basis aufbauen zu können.

Diese Einleitung basiert teilweise auf einem wissenschaftlichen Paper, das im Rahmen des 25. Software Reengineering und Evolution 2023 Workshops verfasst wurde [BK].

[2] **Axivion: https://www.axivion.com/de/**.

4 Konzeption und Implementierung

4.1 Sound Framework

Zeynep Dilara Degerliyurt

Zu jedem Videospiel und Software gehören passende Sound-Effekte, welche dies mehr immersiv und interaktiv machen. Im Fall von einer Software wie SEE sind diese auch wichtig um die Interaktion mit der Architektur realistischer zu gestalten und real-time Feedback zu geben. Optische Veränderungen sind zwar ein Zeichen, dass sich im Spiel etwas getan hat, jedoch sind diese manchmal so subtil, dass sie einem nicht auffallen. Daher ist es besser, mehrere Sinne des Benutzers für das Feedback zu nutzen.

Bisher verfügte SEE über keine Sounds, und auch kein haptisches Feedback, wodurch es außer dem visuellen Feedback keine Reaktionen gab auf Interaktionen mit Graph-Objekten. Die einzigen Sounds, welche im Spiel zu hören waren, sind die Stimmen von anderen Spielern gewesen, wenn SEE im Multiplayer-Modus genutzt wurde.

Das Ziel ist es, ein Framework zu erstellen, mit welchem Geräusche und Musik abgespielt werden können. Hierbei soll es möglich sein, Musik kontinuierlich in einer Endlosschleife abspielen zu lassen für die Szene welche gerade geladen ist, und direktionale Geräusche abzuspielen, so dass es scheint, als würden diese von Objekten selbst in der 3D-Welt ausgelöst werden.

4.1.1 Umsetzung

Um das Framework umzusetzen wurde ein Interface `IAudioManager` erstellt, über welches sämtliche Sound-Funktionen geregelt werden. Dieses verfügt über folgende Funktionen:

- Anpassung der Musiklautstärke.
- Anpassungen der Sound-Effekte-Lautstärke.
- Hinzufügen von Musik in die Musikspieler Playlist (Queue).
- Abspielen von Sound-Effekten.
- Stummschaltung der Lautstärken.

Das Interface wurde so implementiert, dass andere Komponenten von SEE die öffentlichen API Methoden des Frameworks nutzen können für sämtliche Sound-Funktionen, ohne an dem Framework etwas verändern zu müssen. So kann die GUI die Lautstärke allein durch das Interface verändern, ohne weiteren Code in dieses einfügen zu müssen.

Für die Musikspieler-Komponente wird eine `AudioSource` Komponente an das Game-

Object des Spielers gehängt. Diese verfügt über ein Lied, welches in Dauerschleife abgespielt wird, je nachdem welche Szene von SEE geladen ist.

Zum Abspielen von Sound-Effekts hingegen, werden `AudioSource` Komponenten an die GameObjets gehängt (außer in Fällen, wo die Objekte nicht mehr vorhanden sind, z.B. beim Löschen von Objekten) von welchen die Geräusche ausgelöst werden sollen. Dieses wird über die sogenannten `AudioGameObject` Objekte geregelt. Soll ein neues Geräusch von einem GameObject abgespielt werden, so prüft der AudioManager ob das Objekt bereits über ein AudioGameObject verfügt. Ist dies der Fall, so wird in die Warteschlange des Objekts der Sound-Effekt eingefügt, welcher abgespielt werden soll. Ist das Objekt nicht vorhanden, so wird es erstellt und das Geräusch abgespielt.

Das Nutzen dieser Abstraktion erlaubt es, die Sound-Effekte unabhängig von den GameObjects selbst zu verwalten. Da die Audio-Komponenten jedoch an die GameObjects gehängt sind, werden die Geräusche in der 3D-Welt von dort abgespielt, von wo sie ausgelöst wurden. Ist die Warteschlange des AudioGameObjects leer, so wird die Audio-Komponente wieder vom GameObject entfernt.

Darüber hinaus wurden einige Sound-Effekte eingebaut:

- Ein Sound-Effekt, welcher abgespielt wird, wenn über Objekte mit der Maus gehovert wird.
- Ein Sound-Effekt für das Klicken von Objekten.
- Ein Sound-Effekt für das Deselektieren von Objekten.
- Ein Sound-Effekt für das Klicken von Knöpfen in der GUI.
- Ein Sound-Effekt für das Kreide/Malen-Tool (Scribble).
- Hintergrundmusik wurde für die Startszene hinzugefügt.

4.1.2 Fazit

Es wurde ein Sound-Framework erfolgreich hinzugefügt, mit welchem Geräusche und Musik in SEE abgespielt werden können. Des Weiteren wurden einige Sound-Effekte eingefügt für Interaktionen mit Objekten innerhalb des Spiels.

4.2 Runtime Transform Gizmos

Zeynep Dilara Degerliyurt

In den SEE Code-Cities ist es möglich, die Knoten des Graphen zu verschieben, zu vergrößern und verkleinern und diese zu drehen. Dieses soll dazu dienen, die Architektur so zu gestalten, wie es die Software-Architekten für sinnvoll halten, oder um den Fokus auf einige Knoten zu setzen.

Die bisherige Lage erlaubte es, sämtliche obengenannten Aktionen auszuführen. Jedoch wurden diese bisher über eigene Komponenten gesteuert, welche nicht sonderlich gut aussahen, und schwer bedienbar waren. Jedes Gizmo bestand hierbei aus meheren GameObjects, mit einem Update-Listener, welcher in jedem Frame abgefragt hat, ob mit einem der Objekte interagiert wurde, welches nicht sehr effizient war.

Die neuen Gizmos zum Transformieren von Graphknoten sollen besser aussehen, intuitiver und einfacher zu nutzen sein. Sie sollen über ein Unity Plugin verwaltet werden, statt eine eigene Umsetzung zu verwenden. Nach dem Ausführen der Aktionen sollen diese übers Netzwerk zu den anderen Spielern propagiert werden, und sie sollen weiterhin durch Undo und Redo rückgängig gemacht werden können.

4.2.1 Umsetzung

Die Umsetzung begann mit dem Import des Plugins *Runtime Transform Gizmos* [Oct23] des Unity Asset-Stores. Die Assets wurden in SEE eingefügt und eine Assembly-Reference wurde erstellt, um diese in SEE importierbar zu machen (da es über keine Assembly verfügte). Des Weiteren war das Plugin so gedacht, dass es vor dem Start des Spiels manuell initialisiert werden musste. Da dieses in SEE unpraktisch und nicht möglich ist, dank des Wechsels zwischen den Spiel-Szenen, wurde das Plugin angepasst und diesem eine Klasse `RTGInitialiser` hinzugefügt, über welches das Plugin beim Start des Spiels geladen wird.

Um die neuen Gizmos hinzuzufügen, wurden die jetzigen Gizmos Implementierungen aus den Aktionen `ScaleNodeAction` und `RotateAction` entfernt und durch die `ObjectTransformGizmo` von dem Plugin ersetzt. Da diese eigene Update-Action-Listener haben, sind sie effizienter im Vergleich zu den alten Gizmos, welche in jedem Frame auf ein Update gelauscht haben.

In den unteren zwei Bildern Abschnitt 4.2.1 sind die neuen Gizmos zu sehen.

Nach dem Ausführen einer Aktion, wie das Vergrößern/Verkleinern oder Drehen der Graph-Knoten werden die aktuellen States der Objekte abgespeichert, wodurch die Aktionen durch Undo/Redo rückgängig gemacht werden können. Auch werden weiterhin die Aktionen über das Netzwerk propagiert, so dass die Aktionen auch bei anderen Spielern im Multiplayer-Modus ausgeführt werden.

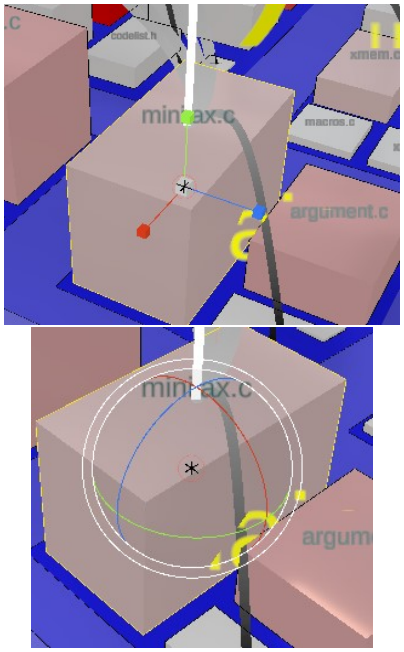


Abbildung 1: Ein Scale- und ein Rotations-Gizmo.

4.2.2 Fazit

Die neuen Gizmos sind eleganter als die manuell-implementierten vorherigen Gizmos und funktionieren effizienter als diese, da sie nur statische Event-Listener nutzen, statt jedes Frame nach Updates zu pollen. Die bisher existierende Funktionalität des Rückkehrens der Aktionen ist weiterhin möglich, und die Propagierung der Aktionen übers Netzwerk bleibt vorhanden.

4.3 Charakteranimation und Steuerung

Linus Henkensiefken

Zum Beginn des Projektes fiel auf, dass sich die Steuerung des Avatars kontraintuitiv, etwas unbeholfen und ungenau anfühlte. Dies wurde behoben, was durch das nun einfacher gestaltete Umschauen in der Welt von SEE dazu führte, dass neue Fehler in SEE entdeckt wurden. Somit wurden nachträglich die Animationen sowie zugehörige Systeme des Avatars korrigiert, überarbeitet und angepasst.

4.3.1 Implementierung, Herausforderungen und Lösungen

Umschauen mit dem Desktop-Avatar Die Überarbeitung des kontraintuitiven Steuerns war eine der ersten Aufgaben in SEE. Deswegen wurden die ersten Treffen dazu genutzt, sich mehr in SEE, besonders in Unity, den dort verwendeten Avatar und dessen Steuerung einzuarbeiten. Danach hat sich leider ein Gruppenmitglied nicht mehr gemeldet, und die Gruppe bestand nur noch aus einer Person. Es wurde das Skript gefunden, welches den Avatar erstellt, und auch das Skript, welches den Avatar steuert. Beim Betrachten im Unity-Editor viel schnell auf, die Kamera dreht sich nicht wie wahrscheinlich gewollt um den Kopf, sondern um die Füße, den Nullpunkt des Avatars. Das Bewegen der Kamera um einen falschen Drehpunkt wurde dann behoben.

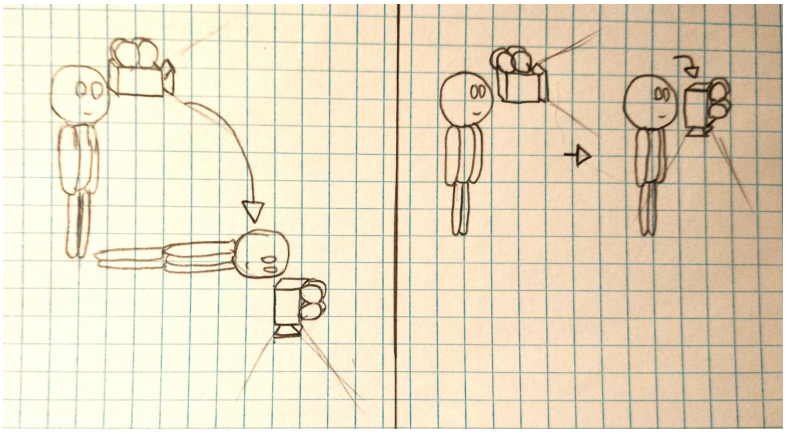


Abbildung 2: Die Kamera dreht sich nicht mehr um den Nullpunkt des Avatars (Links), sondern um sich selbst (Rechts).

Dadurch wiederum viel relativ schnell auf, dass die Kamera sich endlos überdrehen ließ. Eine Neigung von 180 Grad war möglich, womit überkopf geschaut wurde. Normal ist eine Begrenzung der Neigung von ca. 90 Grad nach oben und nach unten, welche dann implementiert wurde.

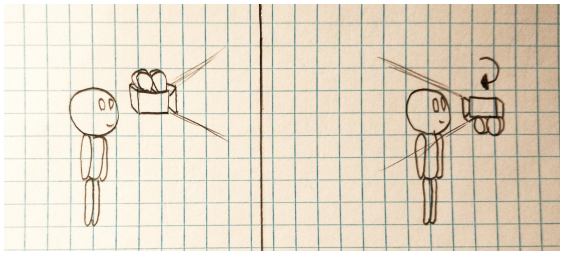


Abbildung 3: Eine vertikale Drehung von mehr als 90 Grad ist selten gewünscht.

Parallel fiel auf, dass durch die Korrektur der Drehung anstatt des Avatars um seine Füße, die Kamera bei seitlicher Drehung nun den Avatar nicht mit dreht, sondern nur sich selbst, und dann den Avatar anschaut. Dies wurde behoben, sodass sich der Avatar wieder mit dreht. Zusätzlich wurde die Kamera auf Augenhöhe vor dem Gesicht des Avatars platziert. Vorher war die Kamera über dem Kopf des Nutzers platziert. Dies geschah, damit der Multiplayer intuitiver ist, indem sich gegenseitig besser angeschaut werden kann. Auch wird direkt dahin gesehen, wohin und von wo andere Nutzer schauen. Zum Beispiel, wenn Nutzer ein Objekt vor die Augen eines anderen Nutzers halten möchten, um es zu demonstrieren.

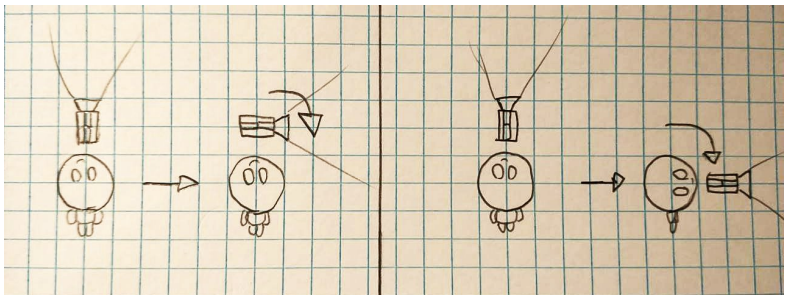


Abbildung 4: Es wird wieder der gesamte Avatar inklusive Kamera gedreht, und nicht nur die Kamera.

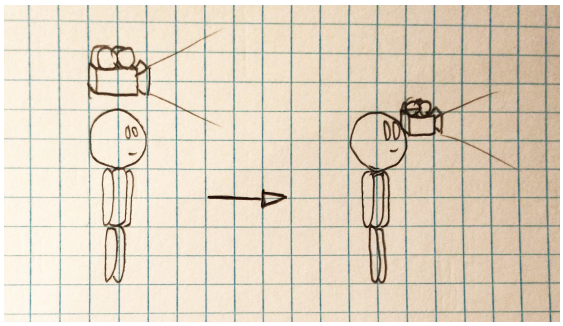


Abbildung 5: Die Kamera findet sich für bessere Immersion nun näher an den Augen des Avatars.

Steuerung des Desktop-Avatars Der Avatar lief sehr langsam, selbst das Rennen war noch langsamer als normales Gehen. Die Geschwindigkeit des Laufens und Rennens wurde angepasst. Dabei fiel auf, dass die Animation, besonders die des Rennens, öfter den Kopf des Avatars so weit nach vorne lehnte, sodass die Kamera im Kopf des Avatars war. Dies sah unschön aus.

Abbildung 6: Das Gesicht des Avatars ist von innen zu sehen.

Es gab nun entweder die Möglichkeit, die Kamera so mit dem Kopf zu verbinden, dass sie sich beim Neigen des Avatars auch nach vorne bewegt, oder die Animation des Avatars so einzustellen, dass er sich nicht so stark neigt. Da der Avatar sich nur minimal in die Kamera lehnte, und genug Spielraum zwischen den Augen, und der tatsächlichen Kamera war, entschieden wir uns, die Animation anzupassen. Dies geschah, da der Vorteil darin bestand, dass die Kamera beim Laufen nicht anfing, nach vorne zu schwenken, oder sich zusätzlich zu bewegen, was eine direktere und simplere Steuerung zum Vorteil hatte.

Animation des Avatars Es wurde sich in das Animationssystem von dem Avatar eingearbeitet, und die Animationen angepasst. Dabei ist aufgefallen, dass diese fehlerhaft sind, es gab kein unterschiedliches Laufen und Rennen wie gewollt, sondern nur Rennen. Schräg rennen sah komplett falsch aus, und links und rechts laufen sowie dann auch Rennen waren vertauscht. All dies regelte der sogenannte “Blend-Tree”, welcher je nach Bewegung und Geschwindigkeit des Avatars aus mehreren Animationen diese so kombiniert, dass sie passen sollten. Je schneller der Avatar sich bewegt, desto mehr wird die Rennanimation anstatt der Laufanimation benutzt, und sobald sich seitlich bewegt wird, werden auch dort wieder andere Animationen mit einbezogen. Der Blend-Tree war falsch konfiguriert. Es wurde kein Unterschied zwischen schnellen und langsamen Bewegungen gemacht. Auch das links Rennen, und rechts Rennen hatte Animationen, die einfach nicht passten, und deswegen durch die Kombination mit anderen Animationen beim Blend-Tree diese ebenfalls falsch ausschauen ließen. Diese beiden Animationen wurden komplett entfernt. Es wurde links und rechts Rennen korrigiert, indem eine beschleunigte Variante des Laufens benutzt wurde. Fast alle Animationen wurden etwas aneinander angepasst, das Neigen beim Rennen reduziert, und die Höhe

der Animationen korrigiert. Die Höhe war vorher unterschiedlich, was dazu führte, dass der Avatar beim Richtungswechsel ggf. etwas nach unten und oben "verrutscht" ist.

Zeigen des Avatars Der Avatar hatte eine Funktion namens "Zeigen", welche den Avatar in eine bestimmte Richtung mit dem Finger zeigen ließ. Zusätzlich wird ein "Laser" aus dem Finger projiziert, damit besser gesehen wird, worauf gezeigt wird.
Bei dem Bearbeiten der Animation fiel der nächste Fehler auf, das Zeigen mit dem Finger in SEE blockierte die Animation des Oberkörpers. Auch beim Nichtzeigen zeigte der Avatar einfach nach unten, obwohl dies nicht gewollt war. Es wurde wohl ursprünglich versucht, die Animation des Zeigens mit der restlichen Animation zu kombinieren. Dies war allerdings wegen Problemen der Unity-Engine nicht so wie eigentlich gewollt möglich. Die Funktion, um Animationen zusätzlich zum Blend-Tree zu kombinieren, hat leider anders als gedacht, einfach nichts getan. Da dies ein Problem der Engine war, überlegten wir uns eine andere Lösung. Die Animation des Zeigens an sich konnte abgespielt werden, wenn die anderen Animationen an den entsprechenden Stellen deaktiviert wurden. Da es optisch am besten aussah, wurden einfach alle Animation des Oberkörpers weiterhin blockiert, allerdings nur während des Zeigens. Und sonst wird die Zeigenanimation blockiert, und die anderen Animationen abgespielt. Somit wird beim Nichtzeigen nicht nach unten gezeigt, sondern wie gewünscht keine Zeigenanimation abgespielt. Beim sonstigen Zeigen wird der Oberkörper sowie die Arme nicht von anderen Animationen beeinflusst. Dies sieht natürlich und gut aus, z. B. beim Rennen richtet der Avatar somit sein Oberkörper etwas auf, um so zeigen zu können.

4.3.2 Fazit (Fortlaufende Ideen, Reflexion des Endzustands und des Ablaufes)

Allgemein führte das innovativere Gestalten der Steuerung von einem Problem zum nächsten, allerdings konnte sich so gut in Unity eingearbeitet werden. Die Hindernisse wurden alle gut überwunden, und Lösungen wurden je nach Anforderungen gut umgesetzt. Die Avatarsteuerung fühlt sich subjektiv nun wesentlich besser an, und der Avatar kann nun objektiv schneller und gezielter bewegt werden. Die Animationen sehen auch passender und schicker aus.
Das Hoch- und Herunterschauen bewirkt allerdings noch keine Animation des Kopfes des Avatars. Der Avatar schaut auch alle paar Sekunden nach links und rechts, wie eine Idle-Animation. Wobei dieses Verhalten durch eine andere Komponente als die Idle-Animation verursacht wird. Dieses Verhalten wird aber in dem Feature FaceCam (Abschnitt 4.5) noch einmal behandelt. "Charakteranimation und Steuerung" wurde durch Linus Henkensiefken umgesetzt.

4.4 Spiegel

Linus Henkensiefken

Da der Nutzer in SEE viel mit seinem Avatar interagiert und auch nonverbal kommuniziert, ist es natürlich, sich etwas mit diesem virtuellen Avatar vertraut zu machen. Wie soll eine Person sonst wissen, wie sie auf andere wirkt, oder z. B., welche Mimiken in VR wie übertragen werden. Dies wird auch oft bei Darstellungen von Leuten gesehen, die einen virtuellen Raum betreten und sich erst mal ihre Hände anschauen, und dann nach einer Spiegelung suchen. Die Idee war, entweder ein kleines Bild im Bild einschalten zu können, welches einen selbst zeigt, oder ein Spiegel in den Raum zu stellen, in dem dann auch mehrere Nutzer sich zusammen selbst betrachten könnten. Das Bild im Bild sollte ca. am Rand oder eine Ecke des Bildschirms oder des VR-HUDs sein, damit es möglichst wenig des restlichen Sichtfeldes blockiert. Der Spiegel sollte groß im Raum stehen, damit ihn jeder leicht entdecken und nutzen kann. Wir entschieden uns für die Umsetzung des Spiegels, und falls die Zeit reicht, wollten wir uns zusätzlich um das Bild im Bild kümmern.

4.4.1 Umsetzung (Implementierung, Herausforderungen und Lösungen)

Schon während der Arbeit mit den Animationen wurde ein simpler Spiegel eingebaut, damit der Avatar bei Animationen leichter beobachtet werden kann. Dieser Spiegel bestand aus einer Fläche, auf der wie bei einem Monitor einer Überwachungskamera, einfach das Bild einer Kamera dargestellt wurde. Die Kamera befindet sich mittig im Spiegel, somit sieht der Nutzer sich bzw. den Avatar selbst, wenn auf den Spiegel geschaut wird. Das Bild musste nur noch links und rechts gespiegelt werden. So ein Spiegel wurde auch später dann in der neuen Standardszene “SEEReflexion” eingebaut, da dieser schon länger vorhanden war. Die Variante wurde zuvor von Yvo Muskulus implementiert und und als Prefab zur Verfügung gestellt

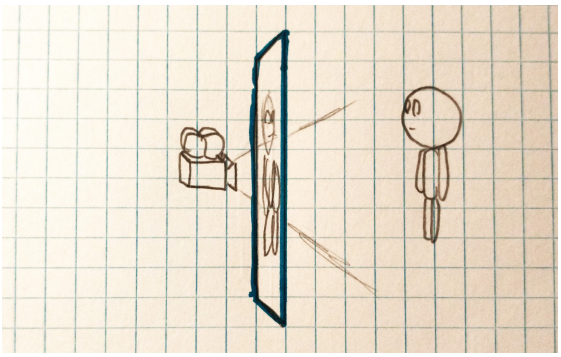


Abbildung 7: Ein simpler Spiegel, bestehend aus einer nicht beweglichen Kamera.

So ein Spiegel ist aber natürlich noch weit von einem realistischen Spiegel entfernt. Je

nachdem wie nah und an welcher Stelle der Nutzer am Spiegel stand, hat der Nuzter sein Gesicht, bzw. das Gesicht des Avatars nicht mehr gesehen. Bei einem echten Spiegel sieht der Nutzer sich immer selber, wenn er geradlinig auf den Spiegel schaut.

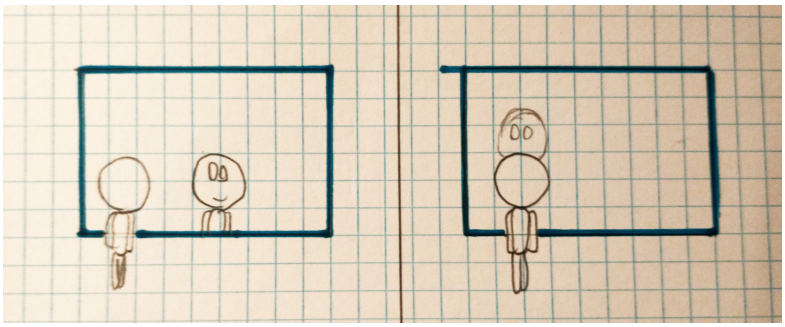


Abbildung 8: Links: simpler Spiegel. Rechts: realistischer Spiegel.

Dies war nicht das einzige Manko. Auch wenn seitlich auf den Spiegel geschaut wird, sollte in etwa das gesehen werden, wo auch ohne Spiegel hingeschaut wird, und nicht das, was geradlinig vor dem Spiegel ist. Dies sollte natürlich auch funktionieren, wenn von oben oder unten auf den Spiegel geschaut wird.

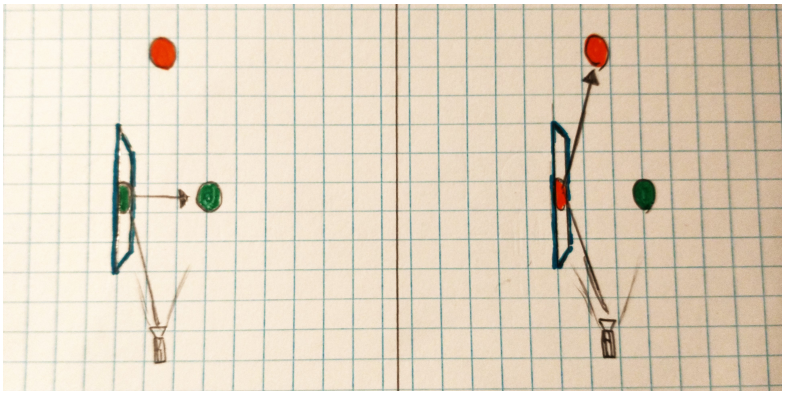


Abbildung 9: Links: simpler Spiegel. Rechts: realistischer Spiegel.

Auch verändert sich die Perspektive je nachdem, wie weit der Betrachter vom Spiegel entfernt ist. Ist er nah dran, hat er ein größeres Sichtfeld, ist er weiter weg und schaut auf den Spiegel, sieht er einen kleineren Ausschnitt.

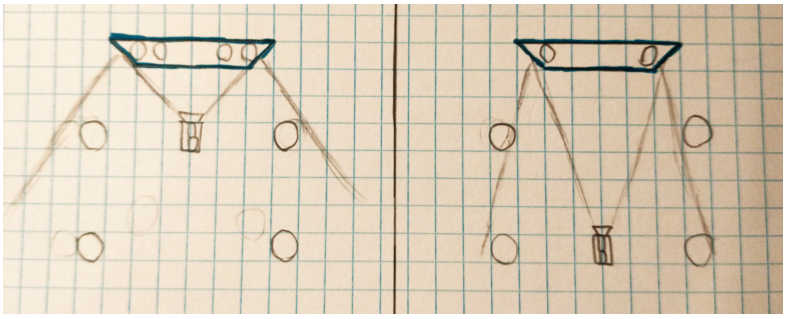


Abbildung 10: Ein realistischer Spiegel ändert den FOV, je nachdem, wie nah der Betrachter sich vor ihm befindet.

Die Umsetzung all dieser Verhaltensweisen eines echten Spiegels benötigte viel Ausprobieren. Es wurden Skizzen dazu aufgezeichnet, wie eine Umsetzung funktionieren könnte, und wie so ein Spiegel überhaupt funktioniert. Dabei wurde immer wieder ein echter Spiegel zur Betrachtung herangezogen. Es stellte sich heraus, dass ein Spiegel sehr gut darstellbar ist, wenn sich einfach eine gespiegelte Version des Avatars vorgestellt wird. Dieser Avatar schaut auch auf den Spiegel, wenn es das Original tut. Er verhält sich exakt wie das Spiegelbild. Jede Bewegung und jeder Winkel, ist gespiegelt, zum Original. Genau so muss sich die Kamera verhalten, welche das Spiegelbild darstellt. Alles, was diese Kamera an der Stelle sieht, wo der Spiegel ist, wäre das, was auf dem Spiegel dargestellt werden müsste.

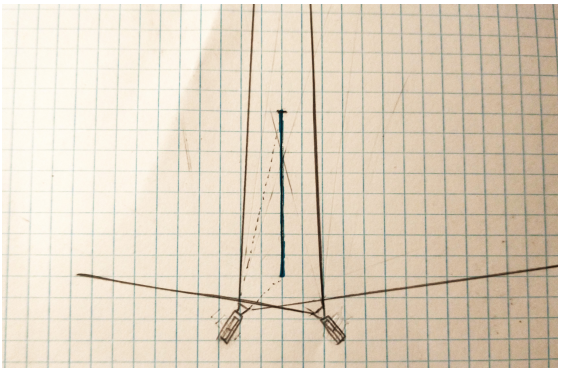


Abbildung 11: Links ist die Kamera, dessen Aufnahme als Textur dem Spiegel dient. Sie ist zur Originalkamera (rechts) gespiegelt. Die gestrichelten Linien stellen den Bereich dar, welcher letztendlich auch als Textur auf den Spiegel dargestellt werden sollten.

Nach diesem Prinzip wurde sich immer mehr an einen realistischen Spiegel herangetastet. Erst wurde die Position der Kamera an die des Spielers, gespiegelt an der Achse des Spiegels angepasst. Dabei fiel auf, dass nur die Entfernung abgeglichen werden muss,

und zusätzlich die Position, wenn der Spiegel als 2D Ebene betrachtet wird, auf der die Position bestimmt werden kann.

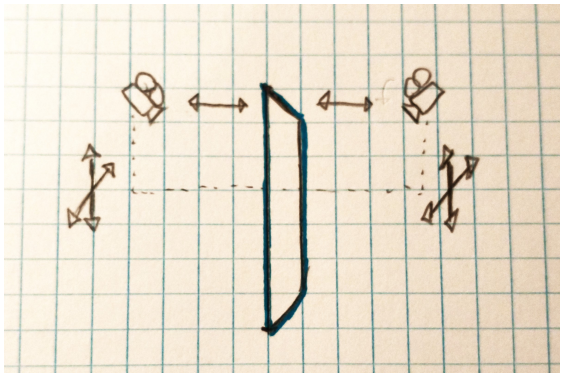


Abbildung 12: Die beiden Kameras sind zur Fläche des Spiegels gespiegelt. Die Information kann als Abstand, inklusive zweidimensionaler Verschiebung, und Drehung der Kamera dargestellt werden.

Das bedeutet, wenn die Winkel und Position des Spiegels mit einberechnet werden, musste nur die Entfernung abgeglichen, die Höhe exakt übernommen, und die von oben gesehene Position mathematisch "gespiegelt" werden. Dies war eine aus meiner Sicht einfachere Berechnung, weswegen ich mich für diese Variante entschied.

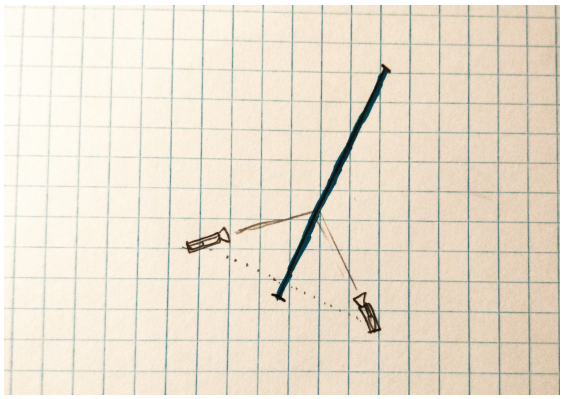


Abbildung 13: Die Höhe bleibt gleich. Von Oben aus wird die zweidimensionale Lage gespiegelt. Dabei wird auch die Drehung der Kamera gespiegelt. Die Neigung bezüglich der Höhe bleibt gleich.

Der Spiegel war nun wesentlich realistischer, das einzige Problem war nur, dass die Spiegelkamera natürlich ggf. ein zu großes oder kleines Blickfeld hatte. Bzw. auch zu viel dargestellt hat, außer dem Bereich, welcher den Spiegel enthielt.

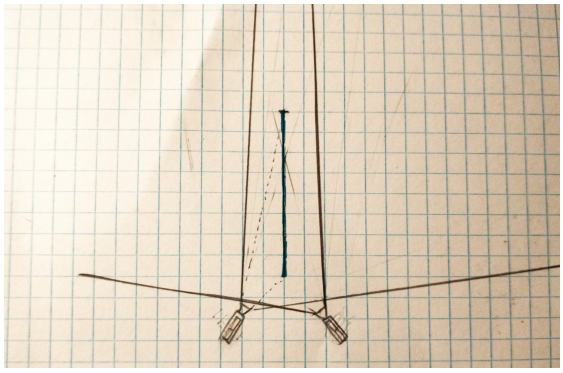


Abbildung 14: Anstatt das der Bereich zwischen den gestrichelten Linien dargestellt wird, wird der gesamte Bereich der durchgezogenen Linien dargestellt.

Die komplett realistische Lösung wäre gewesen, nur ein Ausschnitt der Kamera zu rendern. Dies natürlich in der Auflösung, die der Spiegel haben sollte, also die Pixelanzahl, welche der Nutzer auf den Spiegel sieht. Bzw. wie viel Pixel der Spiegel auf den Bildschirm des Avatars einnimmt. Dies war möglich, dafür würde aber eine Einarbeitung in die Renderpipeline und Shadersprache von Unity nötig sein. Dadurch würde viel Zeit in Anspruch genommen werden. Die Idee war nun, erst einmal einen leicht weniger realistischen, aber funktionierenden Ansatz zu finden. Als Lösung wurde ein Ansatz benutzt, der in etwa das Gleiche erzielte. Er war zwar etwas weniger genau, aber auch nicht so rechenintensiv. Die Spiegelkamera schaute einfach immer direkt so zum Spiegel, dass bei der Anpassung des FOV (Field of View) nur der Bereich des Spiegels abgedeckt wurde.

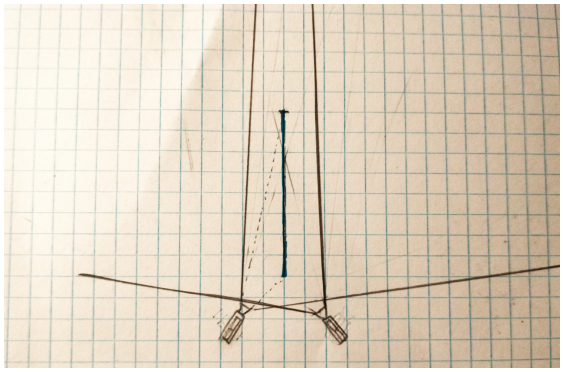


Abbildung 15: Anstatt das der gestrichelte Bereich aus dem gesamten Blickfeld der Kamera (durchgezogene Linien) genommen wird, wird stattdessen das Blickfeld komplett auf den gestrichelten Bereich reduziert.

Nachdem dieser Ansatz getestet wurde, fiel auf, dass der Spiegel nahezu perfekt schien. Es fiel erst gar nicht auf, dass dies kein perfekt realistischer Spiegel ist. Nur wenn

der Betrachter sich sehr nah vorm Spiegel befand, fiel auf, dass er wegen des nicht exakt korrektem FOV, minimal gewölbt erschien. Alle anderen Anforderungen waren allerdings erfüllt. Da die Einarbeitung in die Shader und Renderpipline nun nur noch minimale Effekte erzielen würde, wurde abgestimmt, dass dieser Schritt nicht mehr nötig sei. Außerdem sollte ein Unity-Beispiel ausprobiert werden. Dieses schien alle Funktionen inklusive Eingriff in die Renderpipline haben. Das Beispiel wurde getestet und bearbeitet, funktionierte allerdings leider nicht. Zwischenzeitlich wurde wie schon angemerkt die simple Variante eines Spiegels, die einer Kamera mit angeschlossenem Monitor entsprach in die neue Standardszene "SEEReflexion" eingebaut. Es wurde abgestimmt, dass es bei diesem Spiegel belassen wird, da er ja schon vorhanden ist.

4.4.2 Fazit (Fortlaufende Ideen, Reflexion des Endzustands und des Ablaufes)

Die Arbeit mit dem Spiegel war wirklich sehr interessant und hat viele Überlegungen, Versuche und Lösungen verlangt. Es wurde viel experimentiert und einiges an Erfahrungen gesammelt, auch darüber, wie komplex eigentlich das Sehen und Reflexionen sind. Das Probieren hat viel Spaß gemacht, und ich habe einiges gelernt. Oft dachte ich, der Spiegel sei nun nahezu perfekt, nur um zu merken, dass ich weitere Details nicht bedacht hatte. Auch die Einarbeitung in die Shader und Renderpipeline wären bestimmt spannend und aufschlussreich gewesen, und hätten den Spiegel eventuell letztendlich perfektioniert. "Spiegel" wurde durch Linus Henkensiefken umgesetzt.

4.5 OpenCV Facemask und FaceCam

Linus Henkensiefken

Die Gruppe "OpenCV Facemask" hatte zum Ziel, die Plug-ins "Dlib FaceLandmark Detector" [Eno23a], "OpenCV for Unity" [Eno23c] und "FaceMask Example" [Eno23b] zu untersuchen. Der eigentliche Gedanke dabei war, gegebenenfalls die Mimik des Benutzers auf den Avatar zu übertragen. Alternativ könnte das Gesicht als Bild auf den Avatar gelegt werden.

4.5.1 Umsetzung Facemask

Die Gruppe führte viel Recherche durch, und probierte aus, wie dies möglich sei. Dabei wechselten auch ein paar der Mitglieder der Gruppe. Mit Linus Henkensiefken wurden die Beispiele erneut in SEE implementiert. Relativ schnell fiel auf, dass die Gesichtserkennung von OpenCV zwar relativ akzeptabel funktioniert, die Mimikerkennung aber nur unter optimalen Lichtverhältnissen akzeptabel war. Es wurde trotzdem testweise über Blendshapes mithilfe der über Dlib FaceLandmark Detector Mimik, das Gesicht des Avatars animiert. Da die erkannte Mimik an sich schon fehlerhaft war, sah das Ergebnis auch nicht sonderlich vielversprechend aus. Auf Rücksprache kam auch zu Thema, dass mit Facsvatar eigentlich schon eine Animation der Mimik des Avatars in Planung war. Die Möglichkeit das Gesicht des Nutzers auf das Gesicht des Avatars zu übertragen bestand noch, wurde aber nach längerer Diskussion verworfen. Gründe dafür waren wie folgt meherere. Die Qualität der Webcam und der Gesichtserkennung ließen kein komplett flüssiges Erkennen des Gesichts zu. Dies war zwar kein unüberwindbares Hindernis, allerdings spielte es in der Komplexität und zu erwartenden Qualität eine Rolle. Das Gesicht sollte als 3D-Daten erhoben werden, um nicht nur ein flaches Gesicht zu erhalten. Es wurde in der Diskussion angemerkt, dass Bumpmaps, welche ein 3D simulieren, eine Lösung sein könnte. Diese 3D-Daten, welche dafür benötigt werden, waren aber diese, welche fehlten. Weder OpenCV noch Dlib FaceLandmark Detector hatten Werte über die Tiefe eines Gesichtes. Sie arbeiten komplett 2D. Mit 3D-Daten könnte auch z.B. Displacement Mapping genutzt werden, welches dann aus einer 3D-Textur die Erhöhungen berechnet. Es gab leider keine Möglichkeit, mit den vorhandenen Plug-ins diese 3D-Daten zu bekommen. Wäre eine Lösung oder ein Plug-in gefunden wurden, welches dies könnte, würde noch das Problem der Beleuchtung und Qualität des Bildes der Webcam bestehen. Das Bild würde stark beeinflussen, wie gut dieses dreidimensionale erstellte Gesicht des Avatars mit dem Webcambild aussehen würde. Würde das Gesicht mal falsch erkannt, ohne dass dies rausgefiltert werden könnte, sähe dies bestimmt auch sehr seltsam aus. Die Idee hier war dann, den Avatar einfach einmal ein Gesicht anhand des Webcambildes zu generieren. Dieses Gesicht könnte dann fortlaufend anhand des Bildes der Webcam animiert werden. Auch dafür wären allerdings 3D-Daten über das Gesicht hilfreich. Zusätzlich stand an, dass das Model des Avatars durch ein sogenannten "FACSvatar" (Abschnitt 4.13) ersetzt würde. Dieser sollte mit "Facs" das Gesicht des Avatars wesentlich vielversprechender animieren können. Dar-

aus ließ sich auch schließen, dass eine Erstellung eines Gesichtes des FACSvatar anhand eines Bildes oder Videos, lieber über Facs getan werden sollte. Die Gruppe der Facemask war so praktisch redundant, da die Facs wesentlich vielversprechender waren. Auch fehlten die 3D-Daten und das Model des Avatars sollte sowieso durch ein FACSvatar ersetzt werden, dies würde eventuell dazu führen, dass viel Arbeit doppelt getan würde, und zusätzlich auch ein Teil beim Modellwechsel verworfen werden müsste.

4.5.2 Umstrukturierung zur FaceCam

Es wäre eventuell möglich gewesen, sich der FACSvatar-Gruppe anzuschließen, die Idee war aber, die bisherige Arbeit der Gruppe für ein anderes, sinnvolles und hilfreiches Feature zu erstellen. Unser Ziel von fort an war es, eine von uns so genannte "FaceCam" zu erstellen. Diese sollte lediglich das Gesicht des Nutzers übertragen. Also so wie eine Videotelefonie zusätzlich zum Avatar. Die Idee war es, ein "Screen" zu erschaffen, eines,einen unendlich flacher, von hinten nicht sehbaren, schwebenden Bildschirm, welcher das Gesicht des Nutzers streamte. Dieser sollte vor oder über dem Avatar schweben. Alternativ sollte ausprobiert werden, wie es aussehen würde, wenn der Avatar beispielsweise ein Zylinder als Kopf und gleichzeitig Screen bekommen würde. Dabei wäre dann das Gesicht mehr mit dem Avatar verbunden.

Zu Beginn hatten wir mehrere Punkte, an denen wir ansetzen konnten. Erst einmal gab es noch einige Bugs und Fehlermeldungen mit dem OpenCV-Beispiel in SEE. Dann musste die komplette FaceCam inklusive Video auch im Multiplayer übers Netzwerk funktionieren. Das Gesicht wurde nicht immer zuverlässig erkannt, auch nicht immer nur eins. Da mussten wir Lösungen finden, nicht genau immer das hin und her springende Quadrat der OpenCV als direkten Input der Anzeige des Gesichtes zu nutzen, sondern anhand dieser Daten, eine wahrscheinliche Position des Gesichtes zu erraten bzw. zu errechnen. Das Gesicht sollte ausgeschnitten werden, am besten, da Dlib FaceLandmark Detector zumindest das Kinn relativ gut erkannte, entlang des Kinns, und ungefähr den Rest des Kopfes. Dabei wollten wir dann einen Verlauf zur kompletten Transparenz nutzen, um die ungenauen Daten etwas zu kaschieren. Auch sollte es eine Möglichkeit geben, die FaceCam an und auszuschalten, und ggf. die Kamera, welche das Video aufnimmt, zu wechseln.

4.5.3 Umsetzung FaceCam

Anfangs teilte sich die Gruppe auf, Zhen Yu kümmerte sich um das Aktivieren und Deaktivieren der FaceCam, und fügte dem Beispiel die Funktion hinzu, die FaceCam ein- und auszublenden und dabei die Webcam zu deaktivieren. Danach kümmerte sich Zhen Yu parallel zu Zeynep Dilara Degerliyur, um das detaillierte Ausschneiden des Gesichtes. Dabei wurden wie geplant die Dlib FaceLandmarks, markante Linien ent-

lang des Gesichtes, für Augen, Kinn und Nase, genutzt. Alternativ wurde versucht, den Hintergrund mit anderen Methoden auszuschneiden. Beides hat leider aufgrund auftretender Fehler und Schwierigkeiten nicht funktioniert. Die überbleibenden Aufgaben wurden dann von Linus Henkensiefken übernommen. Die meisten Fehler des in SEE eingefügten Beispiels der OpenCV und Dlib FaceLandmark Gesichtserkennung wurden damit behoben, dass der "AvatarAdatper" nun die FaceCam erstellte, anstatt dass diese einfach in der Unity-Szene vorhanden war. Dies lag wohl daran, dass das Beispiel nun nach der Erstellung der Hauptkamera des Avatars vorhanden war. Dies ist die Kamera, welche dafür zuständig ist, dass der Nutzer die virtuelle Welt von SEE auf dem Bildschirm sehen kann. Auch die restlichen Fehler wurden behoben. Der Code der Beispiele wurde verstanden und nun selber verwendet und angepasst. Als Erstes wurde grundlegend eingebaut, dass, wenn mehrere Gesichter erkannt wurden, diese unterschieden werden konnten. Um nur das Gesicht des Nutzers auf dem Screen darzustellen, wurden die Eigenschaften Tile und Offset der Textur (also dem Webcambild) auf diesem Screen angepasst. Mit diesen Werten ließ sich auf dem Screen mithilfe der Daten von OpenCV an das Gesicht heranzoomen. Die Herausforderung dabei war, nach Verstehen der Werte von OpenCV, welche anscheinend das Gesicht mit Position und Größe auf der Textur bestimmten, diese Werte in Offset und Tile, was lediglich die Verschiebung und Gestauchtheit der Textur angab, umzurechnen.

Abbildung 16: Erster Prototyp mit einer fehlerfreien Gesichtserkennung.

Danach wurde immer direkt wie von OpenCV erkannt das Gesicht dargestellt. Allerdings gab es dabei viele "Sprünge", da das Gesicht kurz auch mal fehlerhaft an anderen Orten erkannt wurde. Außerdem wurde das Gesicht selbst oft zwar richtig erkannt, aber dies mehrmals während einer Sekunde minimal verschoben, obwohl sich das Gesicht so nicht bewegt hatte. Dies führte zusätzlich zu einem merkbaren Ruckeln. Die Lösung war, dass nicht sofort das neu erkannte Gesicht übernommen wurde, sondern sich nur langsam dorthin bewegt wurde. Dies eliminierte tatsächlich beide Probleme und sah sehr professionell aus, hatte leider aber auch zur Folge, dass wenn sich das Gesicht wirklich sehr schnell durch den Raum bewegte, es einen kleinen Moment brauchte, bis das Gesicht auf den Screen wiedererschien. Nach herumprobieren wurde eine Lösung gefunden. Es wurde je nachdem, wie weit ein Gesicht entfernt ist, sich mehr in

diese Richtung beschleunigt. Dabei musste dann noch hinzugefügt werden, dass die Beschleunigung in eine Richtung, auch in eine andere Richtung gedreht werden kann. Dies war wichtig, falls sich in Richtung das Gesichts beschleunigt wird, das Gesicht woanders hinbewegt, nicht erst gegen die alte Beschleunigung in eine falsche Richtung angekämpft werden muss, sondern diese sofort wieder in Richtung des Gesichts zeigt. Dies wirkte optisch sehr gut, und brachte fast immer sofort das Gesicht, ohne zu ruckeln, auf den Screen. Es wurde noch angemerkt, dass der "Kalman-Filter" eine gute Lösung sein könnte. Es wurde sich damit befasst, und es stellte sich heraus, dass der Kalman-Filter an sich kein fertiger Filter, sondern eine Methode ist, mithilfe von Schätzungen und Messwerten ein möglichst realistisches Ergebnis zu bekommen. Diese Schätzungen müssten also erst einmal noch implementiert werden. Außerdem wurden in den OpenCV und Dlib FaceLandmark mehre Beispiele zum Filtern gefunden, auch eine Implementierung des Kalman-Filters. Keiner von diesen funktionierte allerdings merklich zuverlässiger als das Ergebnis von OpenCV direkt. Es wurde also keine weitere Methode, um das Gesicht zuverlässig darzustellen, eingebaut.

Als Nächstes war unser Ziel, all dies auch übers Internet im Multiplayer verfügbar zu machen, da sonst wenig Nutzen in dieser bis jetzt erreichten Funktion sein würde. Der Branch war schon weit hinter dem Master, und da zusätzlich nun ein wesentlich besseres Verständnis des Codes aus den OpenCV- und Dlib-FaceLandmark-Beispielen bestand, entschieden wir uns dazu, erst einmal den Code für das Netzwerk, welches nun ein komplett neues Thema an sich war, zu erstellen, und anschließend den Part mit der FaceCam erneut einzubauen. Der komplexe Teil des Codes war bereits einmal geschrieben worden und müsste dann relativ leicht übernommen werden können. Nachdem sich informiert wurde, wie SEE überhaupt Daten über das Netzwerk sendet, und senden kann, wurde zusätzlich in Erfahrung gebracht, wie Unity an sich Videodaten übertragen kann. Dazu gab es leider noch keine fertigen Lösungen oder Beispiele, die gefunden wurden. Es wurde die Netcode-Dokumentation, welche sich leider noch in der Alpha befand, durchgearbeitet, und einige Beispiele aus dieser Dokumentation umgesetzt, um sich in das Thema einzuarbeiten. Es schien, als könnten wir zumindest versuchen, Einzelbilder zu übertragen. Dies sollte über "RPCs", und "Unreliable RPCs" funktionieren. Das sind Methoden, welche Parameter übers Netz übertragen und dann auf einem Client ausgeführt werden. Die "Unreliable" Variante ist dabei besser für die Videoübertragung geeignet, da sie nicht überprüft, ob ein RPC auch angekommen ist. Es wurde eine Struktur von Host, Clients und Owner FaceCams erstellt, welche in der Lage waren, Beispieldaten untereinander zu streamen. Das war etwas kompliziert, da Clients nur mit dem Server bzw. dem Host-Client kommunizieren durften. Der Host durfte wiederum mit allen Clients kommunizieren. Der Owner war der lokale Client, im Gegensatz zu all den FaceCams, welche von woanders her übertragen wurden. Dem lokalen Avatar, sowie jedem dargestellten Remote-Avatar wurde eine FaceCam zugeordnet. Dies wird für jeden Nutzer bzw. jede Instanz von SEE, die sich verbindet, durchgeführt. Falls es also n Nutzer gibt, gibt es pro Nutzer für jeden anderen Nutzer auch noch mal eine remote Version der FaceCam. Das ergibt also n^2 FaceCam Instanzen. Diese wurden so organisiert, dass jeweils die lokalen Instanzen ihr Video an den Server

bzw. Host senden, außer sie sind selbst der Host, und danach der Host all diese Videos an die Clients sendet, außer der jeweilige Client hat dieses Video schon, da er dies selbst gesendet hat. Anschließend wird das Video auf allen Clients, inklusive des Hosts, auf den FaceCams dargestellt.

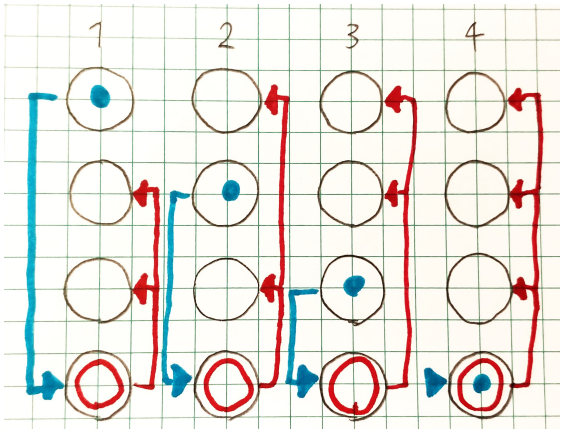


Abbildung 17: Die Spalten 1 bis 4 stellen die FaceCam-Clients pro FaceCam bzw. Avatar da. Der Server ist rot, die Owner blau. Die Pfeile symbolisieren das Senden des Videos.

Nachdem die Struktur fertig war und getestet wurde, war das Ziel, die Beispieldaten, welche übermittelt wurden, durch echtes Video zu ersetzen. Es wurde wieder das OpenCV und DlibFaceLandmark Beispiel eingebaut. Diesmal wurden aber auf möglichst alle überflüssigen Dateien und Codes verzichtet. Auch landete möglichst alles organisiert in einem FaceCam-Ordner, in welchen der Code des Beispiels nicht wie bisher parallel lief, sondern komplett in den eigenen Code eingearbeitet wurde. Als Erstes wurde das erkannte Gesicht wieder auf der FaceCam angezeigt. Dabei wurde nun eine neue Methode gefunden, nicht den Offset und Tile der Textur zu nutzen, sondern nur den Ausschnitt, welcher das Gesicht enthält, freizustellen. Somit sollte auch weniger Bandbreite bei der Übertragung übers Netz genutzt werden. Des Weiteren wurde nun eine Methode eingebaut, um unter mehreren Gesichtern das Gesicht auszuwählen, welches wohl am ehesten dem Nutzer gehört. Es wurde sich entschieden, das Gesicht zu nehmen, welches wahrscheinlich am nächsten an der Kamera ist. Dies wurde umgesetzt, indem das Quadrat der Gesichtserkennung genutzt wurde, welches die größte Fläche hat. Bei der Umsetzung der Netzwerkübertragungen gab es lange Probleme, die einzelnen Frames in JPG-Bilder umzuwandeln. Wenn versucht wurde, die Bilder als Rohdaten zu übertragen, waren diese viel zu groß. Die JPG-Bilder an sich waren allerdings leider immer fehlerhaft. Als sich die Dokumentation angeschaut wurde, hieß es, dass die Textur aus der die JPG-Bilder erstellt werden, nicht komprimiert sein dürfte. Nach längeren Suchen wurde die Funktion im Code gefunden, welche die Textur tatsächlich komprimierte. Diese Funktion wurde ausgetauscht, und es war endlich möglich, das Video als

JPG-Bilder zu übertragen.

Abbildung 18: Beispiel der FaceCam vor dem Gesicht, an die Neigung des Kopfes angepasst (Links). Beispiel über dem Kopf, zum Betrachter ausgerichtet (Rechts).

Zum Zeitpunkt der Veröffenlichung des Berichts steht als Nächstes an, das Webcam-Video wie bei den Beispieldaten nur an alle nötigen Clients zu übertragen. Momentan passiert dies einfach testweise an alle Clients, was so nicht gewollt ist. Außerdem sollte noch von RPC auf Unreliable RPC umgeschaltet werden, und die Framerate, der übers Netz übertragene Bilder, auf ca. 30 pro Sekunde beschränkt werden. Dies sollte nicht zu viel Code erfordern, da hier schon ausprobiert wurde, wie dies möglich ist. Eine schöne Funktion wäre noch, das zufällig wirkende Umschauen des Avatars abzuschalten. Dafür muss nur die sogenannte “Eyes” Komponente abgeschaltet werden. Diese wird aber trotz mehreren Ansätzen bis jetzt nicht wie erwartet aus dem Code heraus angesprochen. Auch sollte der Avatar bei dem Hinauf- und Herabblicken, den Kopf passend dazu neigen. Der letzte wichtige Schritt wäre die Möglichkeit zu implementieren, dass der Nutzer die FaceCam auch aktiveren und deaktivieren kann. Dazu könnte noch die Möglichkeit hinzufügt werden, dass die Position der FaceCam tauschbar wäre. Aus den Tests schien es ganz praktisch, die FaceCam entweder über den Kopf des Avatars oder vor dem Gesicht zu positionieren. Wenn es über den Kopf wäre, würde es sich immer zu dem Betrachter drehen. Somit wäre die FaceCam immer gut sichtbar. Wenn sie vor dem Gesicht ist, wäre es aber intuitiver zu verstehen, wohin der Nutzer dieser FaceCam schaut. Dies ist auch der Grund, warum diese Variante standardmäßig als Erstes ausgewählt sein sollte. Nutzer die sich keine Gedanken über die Möglichkeiten der verschiedenen Positionen der FaceCam machen, oder davon nicht wissen, werden nicht durch die FaceCam über dem Kopf verwirrt. Diese würde nämlich für jeden Nutzer so aussehen, als würden sie direkt angeschaut. Um so Verwirrungen zu vermeiden, ist dieses Positionieren der FaceCam also nur für Nutzer zugänglich, welche die Position der FaceCam extra umschalten. Somit ist die Wahrscheinlichkeit höher, dass diese Nutzer sich Gedanken über die Position der FaceCam machen, und die somit eventuell auftretenden Verwirrungen besser verstehen könnten. Als Buttonbelegung habe ich mich für “O” “L” und “0” entschieden, da diese neben den Button P liegen, welcher für das ebenfalls online interaktive Zeigen zuständig ist. Das “O” direkt daneben aktiviert

und deaktiviert die FaceCam. Mit dem "L" darunter ließe sich die Position wechseln. Die "0" wiederum könnte bei mehreren Webcams bzw. Kameras dazwischen wechseln, und die gewünschte Kamera zu finden. Somit müssten grundlegend alle Funktionen vorhanden sein, welche benötigt werden, um die FaceCam sinnvoll Nutzen zu können.

4.5.4 Fazit (Fortlaufende Ideen, Reflexion des Endzustands und des Ablaufes)

Die Erforschung der Beispiele des OpenCV Dlib FaceLandmark Detectors, sowie das letztendliche Implementieren der FaceCam war wirklich ein großes Projekt an sich. Die Erforschung der Plug-ins und der Möglichkeiten, welche diese bieten, war zusätzlich auch eine sehr lehrreiche Erfahrung. Es wurde viel mit fremdem Code gearbeitet sowie komplett neues Wissen über Funktionen von Unity angeeignet. Auch wurden viele Bugs gefunden und behoben sowie Lösungen gefunden, wo erst einmal gar nicht klar war, wo der Fehler lag. Es wurde Erfahrungen damit gesammelt, größere Projekte aufzuteilen, und Stück für Stück erst einmal das Wichtigste zum Laufen zu bringen. Im Gesamten wurde wirklich sehr viel gelernt und umgesetzt.

In Zukunft kann diese Implementation bestimmt noch viel ausgebaut werden. Es gibt z. B. bessere Komprimierungen von Videos. Auch könnte versucht werden, Video mit dem Sound zu synchronisieren, falls es da Probleme geben sollte. Die Qualität oder Nachbearbeitung des Videos wäre auch noch ein Thema. Auch die Handhabung von besonders schlechter Internetverbindung, oder besonders guter könnte berücksichtigt werden. An sich hat diese Funktion bestimmt Potenzial noch ausgebaut zu werden. Es wurde grundlegend allerdings ein komplexes komplett neues Feature mit all seinen Komponenten entwickelt.

4.6 Runtime Configuration Menu

Mahmoud Siai; Ferdinand Rohlfing; Thilo Beckord

Es sollte möglich sein, die Konfiguration der Code Cities zur Laufzeit vorzunehmen. Dies erhöht die Benutzbarkeit von SEE, da das Vornehmen der Konfiguration in der Szene vor dem Start von SEE nicht nur kompliziert, sondern auch unflexibel ist.

Im Rahmen seiner Bachelorarbeit hat Ruben Smidt [Ble+21, 20ff] bereits eine Version eines solchen Menüs zur Konfiguration implementiert. Dieses musste jedoch nach einigen Änderungen in SEE deaktiviert werden, da es mit diesen nicht mehr kompatibel war. Daher wurde die Konfiguration wieder in der Szene vor dem Start von SEE vorgenommen.

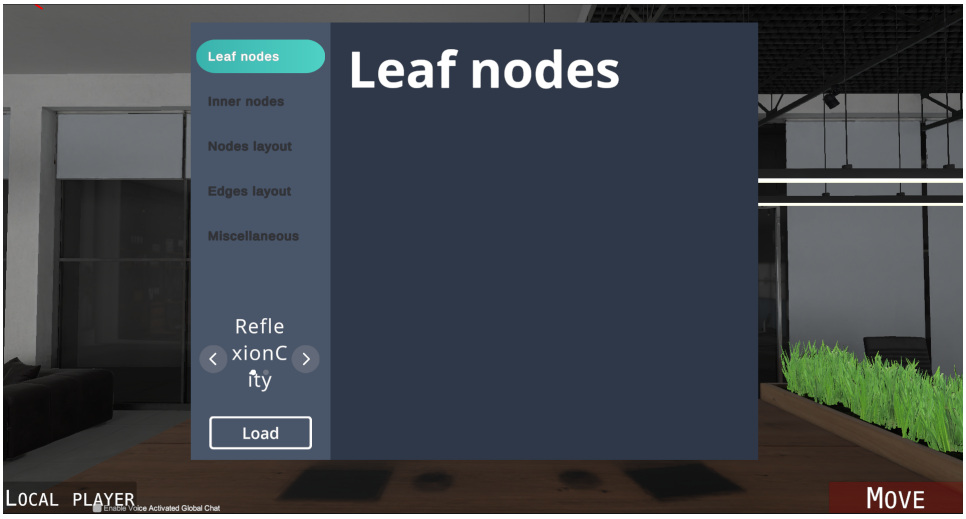


Abbildung 19: Das Konfigurationsmenü von Ruben Smidt [Ble+21, 20ff]

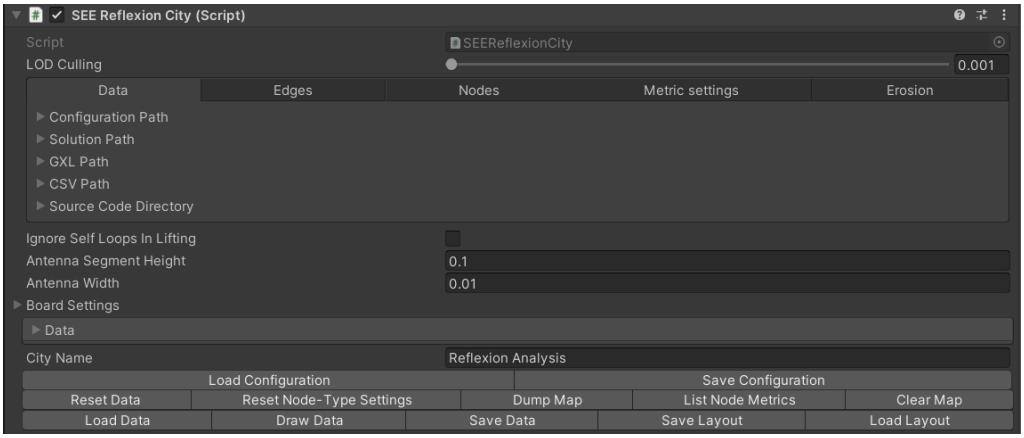


Abbildung 20: Konfiguration einer City im Unity Editor.

Das Ziel war die Implementierung eines übersichtlichen Menüs zur Konfiguration aller Code Cities zur Laufzeit. Ein wichtiger Aspekt ist die Flexibilität des Menüs, sodass dieses auch nach Erweiterungen von SEE möglichst einfach aktualisiert und weitergenutzt werden kann. An der Umsetzung dieser Funktionalität haben Mahmoud Siai, Ferdinand Rohlfing und Thilo Beckord gearbeitet.

4.6.1 UI Design

Mahmoud Siai

Bei der Konzeption des Menüs wurde die Entscheidung getroffen, ein sehr einfaches und VR-zugängliches Menü zu entwerfen, bei dem die Benutzer mit wenigen Interaktionen zum gewünschten Ziel gelangen. Zusätzlich sollen alle Einstellungen aus dem Unity Editor automatisch dem Menü hinzugefügt werden.

Ablauf und Konzeption:
Zu Beginn des Projekts wurde ein Papierprototyp erstellt. Ziel des Papierprototyps war es, die Grundstruktur des Menüs zu definieren und anschließend die notwendigen Prefab zu erstellen.

In der ersten Planungsphase sollten die folgenden Funktionen implementiert werden:

- Erstellen/Löschen eines Tisches
- Laden/Löschen einer City auf einem Tisch
- Bearbeiten von vorhandenen Tischen

Um diese Funktionen zu implementieren wurde ein Prototyp erstellt, der aus mehreren Menüseiten besteht. Die Benutzer sollen eine Übersicht erhalten, in der alle in der Anwendung verfügbaren Tische angezeigt werden. Anschließend kann ein Tisch ausgewählt werden, um eine City zu laden und zu bearbeiten.

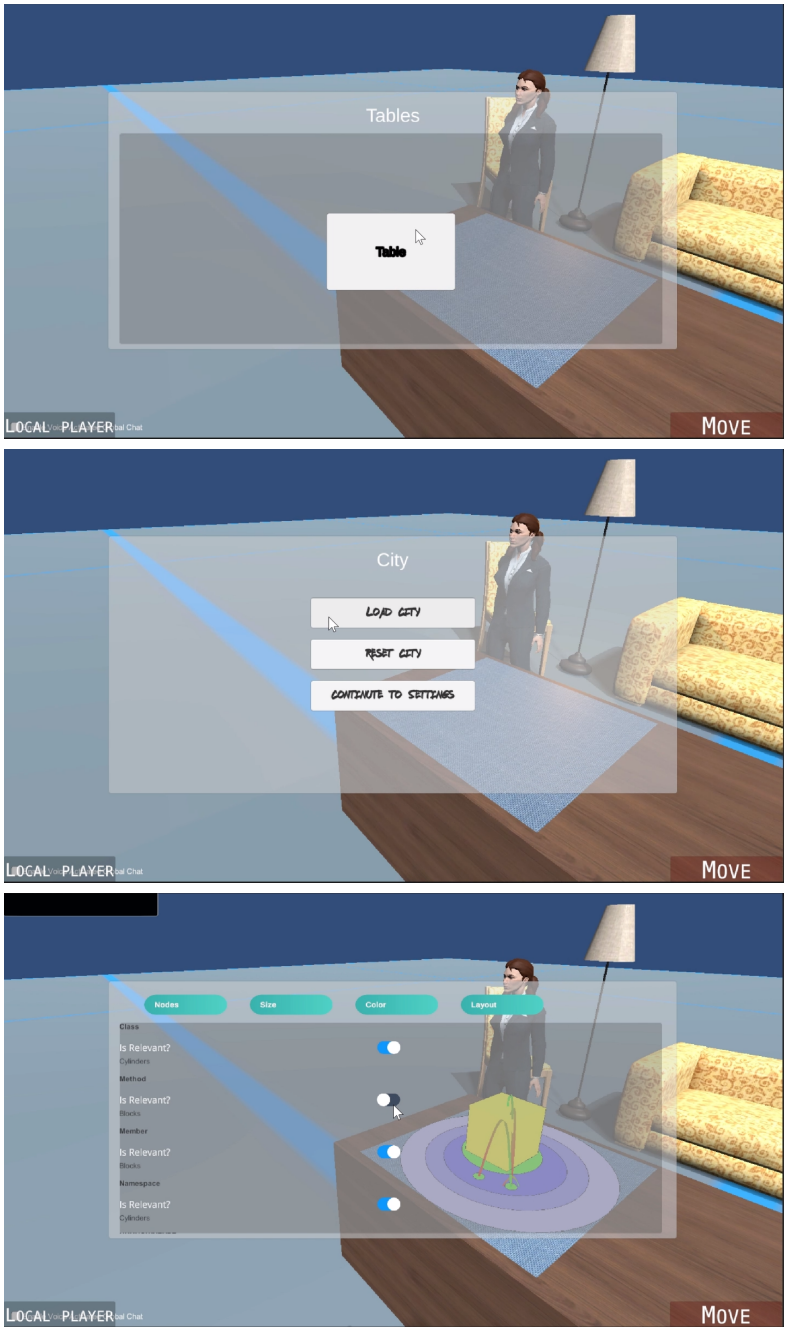


Abbildung 21: Die verschiedenen Seiten der ersten Iteration des Menüs.

Dieser Prototyp wurde schnell angepasst, da sich die Priorität der Funktionalitäten geändert hat. Der Fokus sollte nicht mehr auf dem Erstellen und Löschen von Tischen liegen. Der Hauptfokus des Menüs sollte darauf liegen, Citys auf bestehende Tischen

zu laden und bearbeiten zu können. Durch die Änderung der Prioritäten wurde das Konzept eines verschachtelten Menüs verworfen. Es wurde sich darauf geeinigt, ein Menü zu erstellen, das aus einer Menüseite besteht. So kann sichergestellt werden, dass die Benutzer schnell zu den gewünschten Einstellungen gelangen.

Das finale Menü kann in 5 Bereiche eingeteilt werden.

- City Name und City Switcher (Gelb)
- Tab Menu (Rot)
- Content Menu (Lila)
- Config-Buttons (Pink)
- Menü Buttons (Braun)

Abbildung 22: Finales *Runtime Configuration Menu*

Hierarchie und UI-Framework:

Die Hierarchie, wie in Abb. 23 abgebildet, besitzt die *GameObject*-Hierarchie, die für das UI-Framework erwartet wird. Somit kann das Menü auch für andere Anwendungsfälle genutzt werden. Allerdings müssen gegebenenfalls die Bezeichnungen der Prefabs verallgemeinert werden.

Die Funktionsweise und Anpassungen des UI-Frameworks werden in Abschnitt 4.6.2

vorgestellt.

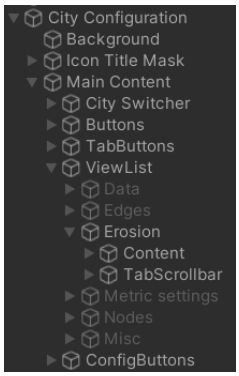


Abbildung 23: Hierarchie des *Runtime Configuration Menu*

City Name und City Switcher:
Der Name der City wird ganz oben im Menü angezeigt. Darunter kann über einen sogenannten `Selector` ein Tisch ausgewählt werden. Anschließend werden die Einstellungen der jeweiligen City geladen und angezeigt.

Tab Menu:
Das Tab Menu ist eine Komponente, die alle verschiedenen Kategorien eines Menüs repräsentiert. Zum Beispiel kann man mit einem Klick auf einen der Buttons alle Einstellungen der jeweiligen Kategorie zu einer City eingeblendet werden. Die so genannten Tab Buttons sind alphabetisch sortiert. Darüber hinaus werden die Buttons abwechselnd in zwei Grautönen gehalten.

Das Tab Menu setzt sich aus zwei Prefabs zusammen: Dem Container, in dem sich die Tab-Buttons befinden und die Tab Buttons selbst. Der Container ist in das Prefab *RuntimeConfigMenu* eingebettet. Dieser besitzt einen scrollbaren Bereich mit einer Scrollbar auf der linken Seite. Die Scrollbar wird erst sichtbar, wenn mehr als acht Tab Buttons angezeigt werden. Die Tab Buttons sind Prefabs, die speziell für den Anwendungsfall erstellt wurden. Auf diese Weise kann sichergestellt werden, dass sowohl die Hover Farbe als auch die Base Farbe einheitlich und auch in der Größe auf das gesamte *RuntimeConfigMenu* abgestimmt sind.

Das Tab Menu hat sich im Laufe der Zeit kaum verändert. In der ersten Iteration war das Tab Menu horizontal angeordnet und nahm den oberen Teil des Menüs ein. Die Prefab Buttons waren weiß und kleiner. Nach dem großen Rework wurden die Buttons nach links verschoben und entsprechend in ein vertikales Menü umgewandelt. Die Buttons hatten eine zufällige Farbe, was jedoch zu Verwirrung führte, da es bei jedem Neustart der Anwendung schwierig war, die richtige Kategorie zu finden. An der Grundstruktur

der Prefabs hat sich im Laufe der Zeit nichts geändert.

Content Menu:
Das Content Menu befindet sich in der Mitte des *RuntimeConfigMenu*. Dieser Bereich enthält die eigentlichen Einstellmöglichkeiten der jeweiligen Kategorie. Die Grundstruktur ist in drei Prefabs unterteilt:

- Container im *RuntimeConfigMenu* (*ViewList*)
- *RuntimeSettingsView*
- *RuntimeSettingsObject* und die dazugehörigen Widgets

Der Container im *RuntimeConfigMenu* besteht nur aus einem leeren *GameObject*, in dem die Größe des Objektes definiert wird. Zusätzlich ist der Container in einer für SEE typischen Farbe gehalten, um ein einheitliches Bild zu den anderen Menüs in SEE zu erzeugen.

Jeder Tab Button im Tab Menu ist mit einer eigenen *RuntimeSettingsView* (SettingsView) verknüpft. Die SettingsView wird durch die Interaktion mit dem jeweiligen Tab Button sichtbar gemacht oder durch das Auswählen einer anderen Kategorie verborgen. Die SettingsView besitzt einen scrollbaren Bereich mit `Scrollbar` und eine `Vertical Layout Group`. Damit können die eigentlichen *RuntimeSettingsObject* in der SettingsView übersichtlich angeordnet werden.

Die *RuntimeSettingsObject*s (SettingsObject) enthalten die Einstellungen, einen Namen und einen Button zum Ein- und Ausblenden der darunterliegenden Einstellungen. Im Folgenden werden diese auch als Widgets bezeichnet. Die Widgets können einem SettingsObject zugeordnet sein. Wenn eine Einstellung keine übergeordnete Kategorie besitzt, wird diese am Ende einer SettingsView angezeigt. Gehört eine Einstellung zu einer Kategorie wie z.B. Nodes, so wird diese in ein SettingsObject platziert. Auf diese Weise werden die einzelnen Einstellungen übersichtlicher kategorisiert und dargestellt. Durch einen grauen Hintergrund wird die Zugehörigkeit auch visuell unterstützt. Dabei können die SettingsObjects verschachtelt dargestellt werden. Zum Beispiel bei der Kategorie Nodes. Die Unterscheidung wird zum einen durch die Einrückung und eine dunklere Farbe visualisiert (siehe Abb. 22).

Um eine bessere Übersichtlichkeit zu garantieren, können die SettingsObjects ein- und ausgeklappt werden. Dies wird jedoch nicht rekursiv durchgeführt. Wenn also ein übergeordnetes SettingsObject ausgeblendet wird, bleiben die inneren Einstellungen ausgeklappt. Sie sind aber nicht sichtbar, da das übergeordnete Elternobjekt minimiert wurde. Der *ColorPicker* ist aufgrund der Übersichtlichkeit des Menüs grundsätzlich ausgeblendet und kann durch Interaktion mit dem Icon eingeblendet werden. Die eigentlichen Widgets sind weitgehend nach dem gleichen Schema aufgebaut. Auf der linken Seite befindet sich ein Label, mit dem die Einstellung beschriftet ist, rechts befindet sich die

Einstellung.

Es existieren folgende Widgets:

- *Switcher*
- *Slider*
- *FilePicker*
- *ColorPicker*
- *StringField*

Jedes dieser Widgets wird über das entsprechende Prefab geladen und bearbeitet.

Eine weitere Besonderheit der Widgets ist, dass durch einen Klick auf das Label das sogenannte *SmallEditorWindow* geöffnet werden kann. Mit Hilfe dieses Fensters können Einstellungen vorgenommen werden während mehr von der eigentlichen Szene/Anwendung sichtbar ist. Außerdem ist bei Verwendung des *RuntimeConfigMenu* die Interaktion mit der dahinterliegenden Anwendung blockiert. Die Interaktion mit der Anwendung ist jedoch über das kleine Einstellungsfenster möglich. Das kleine Fenster kann mit einem Klick auf das linke Symbol geschlossen werden. Danach gelangt man zurück in das *Runtime Configuration Menu*. Beim *ColorPicker* müssen die Benutzer mit der grauen Fläche interagieren, um das kleine Menü zu öffnen. Lediglich beim *FilePicker* gibt es keine Möglichkeit, das Widget im kleinen Fenster zu öffnen, da die Interaktion mit dieser Einstellung keine direkte Auswirkung auf die Tische/City hat. Um Änderungen vom *FilePicker* zu übernehmen, muss anschließend immer einer der Config-Buttons gedrückt werden.

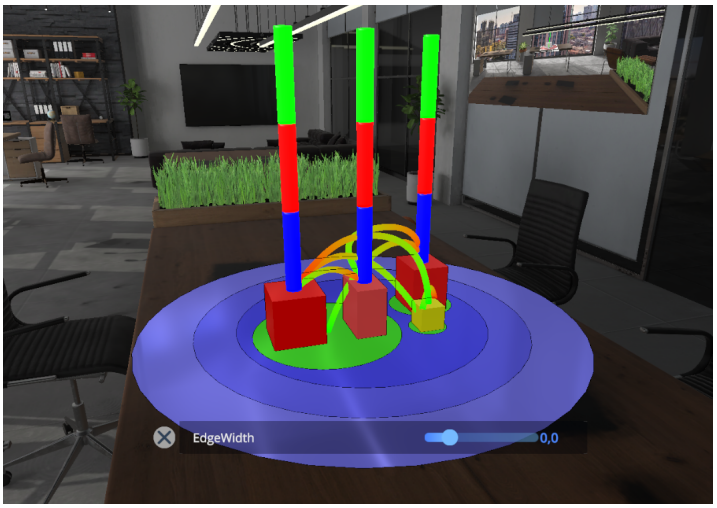


Abbildung 24: Das *SmallEditorWindow*.

Config Buttons:
Die Config Buttons befinden sich auf der rechten Seite des Menüs. Ähnlich wie die Tab Buttons bestehen auch die Config Buttons aus zwei Komponenten:

- Dem Container im *RuntimeConfigMenu* Prefab (ConfigButtons)
- Dem *RuntimeConfigButton* Prefab

Der Container im *RuntimeConfigButton* besteht hauptsächlich aus einer `Grid Layout Group` Komponente, die so eingestellt ist, dass maximal acht Buttons in vertikaler Richtung angezeigt werden können, bis eine weitere Spalte angelegt wird. Diese Designentscheidung wurde getroffen, um dem Benutzer einen schnellen Zugriff auf die Config-Buttons zu ermöglichen. Bei einer zu großen Anzahl von Buttons kann es vorkommen, dass diese aus dem sichtbaren Bereich des Bildschirms herausragen.

Die Config Buttons bestehen aus einem speziell für diesen Zweck erstellten Prefab, welches sich an den Menü Buttons orientiert. Die Größe der Buttons wird nicht durch das Prefab, sondern durch die `Grid Layout Group` bestimmt. Eine weitere Änderung zum normalen Menü Button ist der Wegfall des Icons.

Menü Buttons:
Die Menü Buttons und der jeweilige Container wurden aus den anderen Menüs in SEE übernommen und nicht weiter angepasst.

Probleme bei der Implementierung:
Beim Laden des Menüs entstand das Problem, dass die Widgets in einer *SettingsView* verschachtelt und übereinander angezeigt wurden. Dieser Fehler konnte zur Laufzeit durch mehrmaliges Klicken eines Tab Buttons behoben werden. Es stellte sich heraus, dass `Content Size Fitter` in Verbindung mit `Layout Groups` nicht inflationär eingesetzt werden können.

Abbildung 25: Widgets werden verschachtelt angezeigt.

4.6.2 UI-Framework

Ferdinand Rohlfing

Im vorherigen Bachelorprojekt wurde ein UI-Framework eingeführt, das verschiedene Stellen vereinheitlichen und eine Abstraktion entwickeln sollte, mit dem sich UI-Komponenten leicht wiederverwenden lassen, ohne sich um plattformspezifische Details kümmern zu müssen [Ble+21, S. 20].

Damit wir unser Menü hinzufügen können, musste das UI-Framework erweitert werden. Dabei ist vor allem die Klasse `SimpleMenu` relevant, da dieses von der Struktur mit einem Titel, Beschreibung und Icon unserem geplanten Menü am meisten gleichkommt. Zusätzlich besitzt die Klasse, die Funktionalität Einträge (bzw. Buttons) hinzuzufügen, die dann unterschiedliche Funktionen umsetzen können.

Herausforderungen:
Zwar hat das UI-Framework die Verwendung der UI-Elemente deutlich vereinfacht, dennoch hatte es einige Beschaffenheiten, welche die Erweiterung um das *Runtime Configuration Menu* erschweren.

Dennoch gab es einige Herausforderungen, welche bewältigt werden mussten:

- Hierarchie-Anforderungen
- Aufteilung des `SimpleMenu`
- Events

Hierarchie-Anforderungen:
Das *SimpleMenu* brauchte vom Menü Prefab eine bestimmte Hierarchie mit folgenden GameObjects:

- Das Menü-Icon bei „`Main Content/Icon Title Mask/Content`“
- Einen Container bei „`Main Content/Content Mask/Content`“
  - Darin eine Liste für Buttons bei „`Menu Entries/Scroll Area/List`“

Darüber hinaus musste das Menü Prefab eine `ModalWindowManager`-Komponente das Button Prefab eine `ButtonManager`-Komponente enthalten. Die Notwendigkeit der Komponenten wurde nirgendwo aufgeführt.

Um die Anforderungen an die Hierarchie anpassen zu können, sind die Pfade zu den benötigten Menü Elementen (wie Titel, Icon, etc.) als Attribute im *SimpleMenu* umgesetzt. So können Menüs mit weiteren Spezifikationen, wie das unsere, leichter angepasst werden. Damit die Struktur der bestehenden Menüs nicht angepasst werden muss, haben die Attribute die bisher erwartete Hierarchie als Standardwert.

Zuletzt gab es auch Anforderungen an die Hierarchie für Elemente im Button-Prefab, die schon in der `ButtonManager`-Komponente festgelegt waren. Diese redundanten Anforderungen wurden entfernt, sodass nun der Titel und das Icon des Button Prefab über den `ButtonManager` festgelegt wird, und somit die Anforderungen für die Hierarchie im Button Prefab komplett entfällt.

Aufteilung des *SimpleMenu*:
Bei der Überarbeitung der Hierarchie-Anforderungen fiel auf, dass die Klasse `SimpleMenu` viel Funktionalität besitzt. Die Funktionalitäten können auch auf mehrere Klassen aufgeteilt werden:

Das Simple Menu besitzt zwei große Aufgaben. Auf der einen Seite die generellen Menü Eigenschaften, wie z.B das Festlegen des Titels, der Beschreibung, des Icon sowie das Ein- und Ausblenden des Menüs. Auf der anderen Seite die Verwaltung der Menüeinträge, wie das Hinzufügen und Entfernen der dazugehörigen Buttons.

Da einige Menüs keine Einträge benötigen, ist eine Aufteilung in mehrere Klassen sinnvoll: Das `SimpleMenu` repräsentiert nur noch ein Menü mit Titel, Beschreibung und Icon, welches vom `SimpleListMenu` um die Funktionalität der Menüeinträge erweitert wird.

Events:
Damit die Menüs unabhängig von den Funktionen wiederverwendet werden können, muss es möglich sein, auf Eingaben und Änderungen des Menüs zu reagieren: Beispielsweise stellen viele Unity UI-Elemente die Events `onValueChanged` und `onClick` zur Verfügung. Damit dies auch im UI-Framework möglich ist, werden hierzu einige Events bereitgestellt.

Hierzu zählen unter anderem:

- Das Menü ist fertig initialisiert (`OnMenuInitialized`)
- Das Menü wurde aus- oder eingeblendet (`OnShowMenuChanged`)
- Der Titel, die Beschreibung oder das Icon haben sich geändert
  - `OnTitleChanged`, `OnDescriptionChanged`, `OnIconChanged`
- Über Audio wurde ein Sprachbefehl erkannt (`OnKeywordRecognized`)

4.6.3 Erstellen des Menüs

Ferdinand Rohlfing

Vorheriges Menü:

Das vorherige Konfigurationsmenü von Ruben Schmidt [Ble+21, 20ff] erstellte alle UI-Elemente für die Einstellungen manuell. So musste für jedes Attribut und Methode händisch entschieden werden, ob und wie die Einstellung im Menü repräsentiert wird.

Zwar kann die manuelle Erstellung der UI-Elemente sehr viel Flexibilität bieten, es führte aber eher dazu, dass bei Änderungen der *City*-Klassen die Anpassung des Konfigurationsmenüs umständlich, fehleranfällig und zeitaufwendig war.

So wurden bei Änderungen der *City*-Klassen inkompatible Menü-Teile einfach auskommentiert und neue Einstellung nicht mit ins Konfigurationsmenü übernommen, anstatt das Konfigurationsmenü an diese Änderungen anzupassen. Dies führte graduell dazu, dass im Menü immer weniger Einstellungen bearbeitet werden konnten.

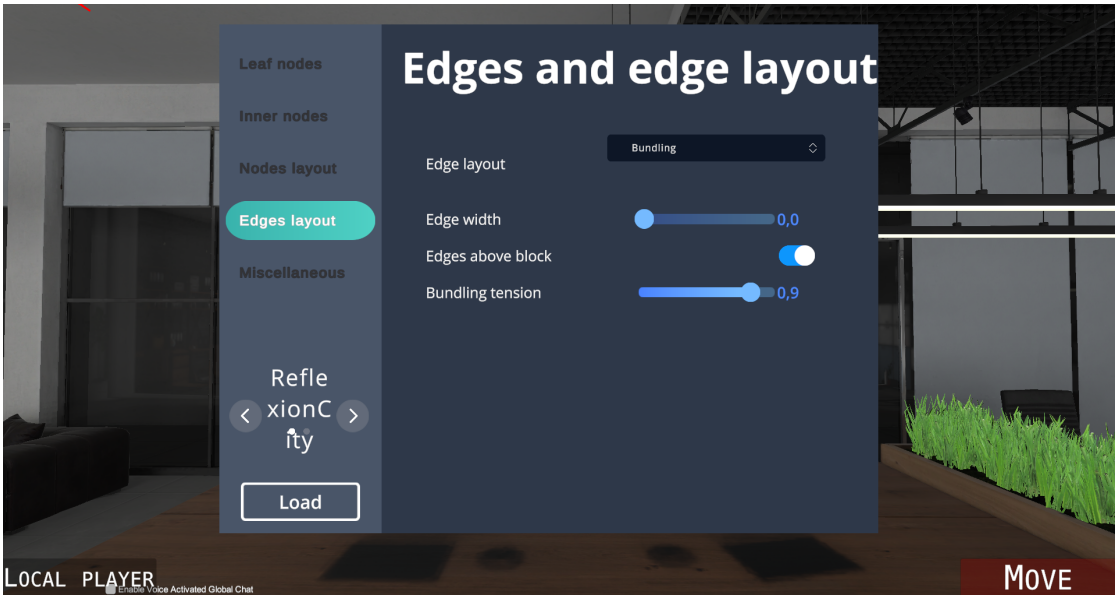


Abbildung 26: Tab bei dem nur noch wenige Einstellungen vorhanden sind

Generelles Vorgehen:

Damit die Problematik des vorherigen Menüs nicht bestehen bleibt, soll sich das neue Menü alle Attribute und Methoden der *City* automatisch beschaffen. Also ohne dass händisch festgelegt werden muss, welche Einstellungen wie und wo im Menü dargestellt werden.

Dabei soll das Menü die Hierarchiestruktur der Attribute widerspiegeln und alle ver-

fügbaren Einstellungen im Menü hinzufügen, ohne dass bei Änderungen das Menü auf diese angepasst werden muss.

Attribute und Methoden:
Die automatische Beschaffung der Einstellungen wird mit Hilfe von *C# Reflection*s umgesetzt. So geht das Menü durch alle öffentlichen (`public`) Attribute und Methoden der *City* und erstellt die dafür benötigten UI-Elemente.

Um dabei nur Attribute und Methoden darzustellen, die für Einstellungen der *City* dienen, werden zuerst alle Attribute und Methoden entfernt, die von einer Elternklasse von `AbstractSEECity` geerbt sind (z. B. von `MonoBehavior`).

Danach werden die Attribute und Methoden durchgegangen: Für jedes Attribut wird `CreateSetting` und für jede Methode wird `CreateButton` aufgerufen. Um hierbei die Reihenfolge zu beeinflussen, können die Attribute und Methoden mit bestimmten Attributen annotiert werden. (siehe Abschnitt 4.6.5) Die benötigten UI-Elemente werden dann in `CreateButton` und `CreateSetting` erstellt.

CreateButton:
Erstellt für jede Methode, die das Attribute `RuntimeButton` hat, ein Config Button auf der rechten Seite des Konfigurationsmenü erstellt (siehe Abschnitt 4.6.1). Dabei werden nur Methoden unterstützt, die keine Parameter haben.

CreateSetting:
Das Verhalten von `CreateSetting` kann in zwei Blöcke aufgeteilt werden:

- Das Erstellen der Widgets für nicht verschachtelte Datentypen
- Das Erstellen der UI-Struktur für verschachtelte Einstellungen

Mit verschachtelten Einstellungen sind Objekte von Datentypen gemeint, die selber Attribute besitzen. So können im Menü nur die Attribute bearbeitet werden und das Objekt selber dient nur als Container für diese Attribute. Da ein Attribut von einem solchen Objekt auch selber wieder verschachtelte Attribute enthalten kann, entsteht dadurch eine mehrstufige Struktur, die sich durch Container-Objekte auch im Menü widerspiegelt (also eine Tiefensuche durch die Attribute, bis ein Datentyp gefunden wird, der nicht als Container für weitere Attribute dient).

Dafür wird zuerst überprüft, ob die Einstellung einen Datentyp besitzt, der nicht weiter verschachtelt ist. Für die Einstellungen wird dann ein passendes Widget erstellt: So zum Beispiel ein File Picker für Dateipfade, ein Slider für Integer und ein Button für *boolean*s erstellt.

Bei verschachtelten Einstellungen wird zuerst ein Container-Objekt erstellt (siehe Settings-

Object). Dannach wird für alle darin enthaltenen Einstellungen wieder die Methode `CreateSetting` aufgerufen, wobei die UI-Elemente dann in den erstellten Container eingefügt werden. Durch das Einfügen der verschachtelten Attribute in das Container-Objekt stellt das *Runtime Configuration Menu* die Hierarchie der Einstellungen dar.

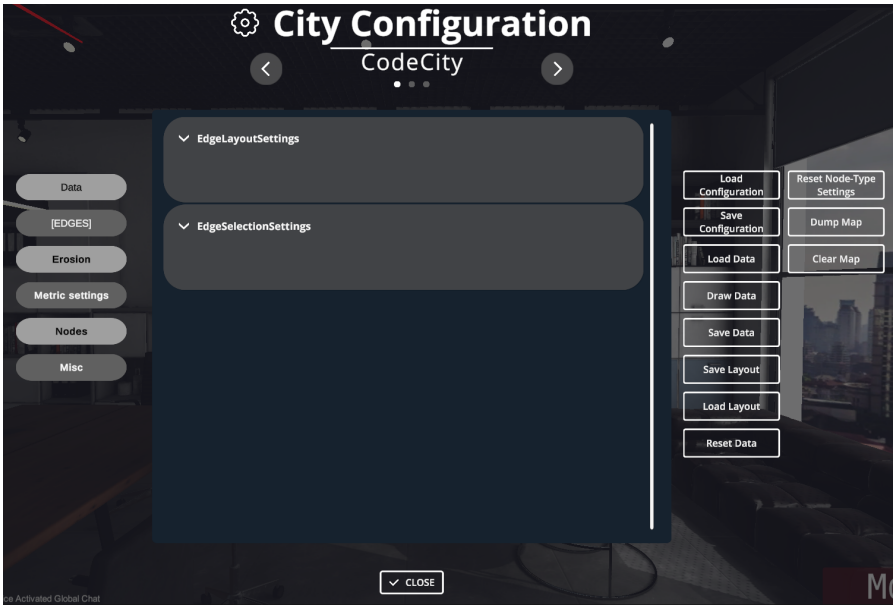


Abbildung 27: Menü mit Verschaltelungstiefe 1

Abbildung 28: Menü mit Verschaltelungstiefe 2

Sammlungen und HashSet:
Sammlung werden von `CreateSetting` wie verschachtelte Objekte behandelt: So wird für die Sammlung ein Container-Objekt erstellt und für die Elemente werden die Widgets in den Container eingefügt.

Für die meisten Sammlungen kann dafür einfach über alle Elemente iteriert werden und für jedes Element nochmal `CreateSetting` aufgerufen werden. Die Vorgehensweise führt aber nur zu einem gewünschten Verhalten, wenn sich die Reihenfolge durch das Ändern des Inhalts nicht ändert, dies ist beim HashSet aber nicht so.

Wenn z. B. ein Element eines HashSets geändert wird, kann das dazu führen, dass sich beim Bearbeiten die Elemente andauernd umsortieren. So haben wir uns dafür entschieden, dass `HashSets` vorerst im Konfigurationsmenü nicht dargestellt werden.

4.6.4 Netzwerk **Thilo Beckord**

Zur Übertragung des Menüs und seiner Änderungen an andere Teilnehmer werden dieselben `Net Actions` genutzt wie für andere Aktionen. Zunächst wurde versucht, eine `Net Action` für alle Einstellungen mit dem Datentyp `object` zu implementieren. Dies scheiterte jedoch daran, dass die `Net Action` die Speichergröße des Datentyps kennen muss. Daher wurde zunächst für jeden Datentyp eine eigene `Net Action` erstellt. Im Rahmen des Reviews des Pull Requests wurden diese `Net Actions` dann zu einer generischen zusammengefasst.

Bei Änderung von Einstellungen wird eine NetAction erstellt, die den Identifier des geänderten Widgets sowie (sofern nötig) die geänderten Werte über das Netzwerk sendet. Beim Empfänger wird ein Event ausgelöst (`SyncField`, `SyncMethod`, `SyncPath`, `SyncAddListElement`, `SyncRemoveListElement`). Wenn der übergebene Identifier mit dem eigenen Widget übereinstimmt, wird dessen Wert aktualisiert.

4.6.5 Attribute **Thilo Beckord**

Die Tab Struktur des Konfigurationsmenüs im Unity Editor (siehe Abb. 20) sowie des *Runtime Configuration Menu* in SEE wurde durch Attribute des Odin Serializers erreicht. Jede Einstellung der Code Cities erhielt dafür ein `TabGroup` Attribut, welches dann im *Runtime Configuration Menu* ausgelesen wurde, um diese Einstellung in den entsprechenden Tab einzuordnen.

Beim Testen des Menüs im Build fiel auf, dass es nur den Tab 'Misc' gab, in den

Einstellungen ohne ein solches `TabGroup` Attribut sortiert werden. Der Grund hierfür ist, dass die Attribute des Odin Serializers ein `Conditional Statement` beinhalten, welches dafür sorgt, dass die Attribute nur im Unity Editor verfügbar sind. Alles, was den Unity Editor benutzt, ist jedoch im Build nicht verfügbar. So mussten die Attribute des Odin Serializers durch eigene ersetzt werden, um die Menüstruktur im Build zu erhalten. Dies betrifft neben den `TabGroup` Attributen auch die `ButtonGroup` Attribute.

Die neu hinzugefügten Attribute `RuntimeTab` und `RuntimeButton` ersetzen dabei nicht die bereits bestehenden Attribute des Odin Serializers, sondern ergänzen sie lediglich. Sie dienen nur zur Erhaltung der Menüstruktur im Build. Das `RuntimeTab` Attribut bekommt als einzigen Parameter den Namen des Tabs, in den die Einstellung sortiert werden muss. D. h. es kann exakt so verwendet werden wie das `TabGroup` Attribut. Für das *RuntimeButton* Attribut gilt dasselbe. Dieses besitzt zusätzlich einen Parameter, der das Label des Button festlegt:

```
[Tooltip("Settings for the edge layout."), TabGroup(EdgeFoldoutGroup),
RuntimeTab(EdgeFoldoutGroup)]
public EdgeLayoutAttributes EdgeLayoutSettings = new();

[Button(ButtonSizes.Small)]
[ButtonGroup(ConfigurationButtonsGroup),
RuntimeButton(ConfigurationButtonsGroup, "Load Configuration")]
[PropertyOrder(ConfigurationButtonsGroupLoad)]
public void LoadConfiguration()
{
Load(ConfigurationPath.Path);
}
```

4.6.6 Erstellen der Widgets

Ferdinand Rohlfing; Thilo Beckord

Das tatsächliche Vornehmen von Einstellungen findet in fünf Widgets statt. Diese werden jeweils in einer eigenen Methode erstellt. Innerhalb dieser Methode wird das zugehörige Prefab instantiiert, der angezeigte Wert der Einstellung gesetzt und die Listener hinzugefügt. Einige Widgets (z.B. der *ColorPicker*) benötigen mehrere Listener, um alle veränderbaren UI Elemente zu überwachen. Da die Listener der Widgets über unterschiedliche Events gesteuert werden, musste für jedes Widget eine eigene Methode kreiert werden. Die Listener sorgen für das Aktualisieren der Werte beim Spieler sowie über die Action `SyncField` bei anderen Spielern. Darüber hinaus wird in diesen Methoden auch die Komponente für das Small Editor Window hinzugefügt.

4.6.7 Fazit

Das zuvor nur im Unity Editor verfügbare Menü wurde erfolgreich zur Laufzeit verfügbar gemacht. Es kann mit dem Keyboard Shortcut 'K' geöffnet werden und implementiert alle Einstellungsmöglichkeiten des existierenden Menüs. Wenn neue Einstellungen hinzugefügt werden, können diese mit unseren Attributen versehen werden. Dadurch werden die Einstellungen korrekt im Menü angezeigt.

Der nächste Schritt für die Weiterentwicklung des Menüs ist die Umsetzung für VR. Außerdem wäre es sinnvoll, die Bezeichnung der Prefabs zu generalisieren, sodass in Zukunft andere Funktionalitäten mit einem Tab Menu umgesetzt werden können.

4.7 Solving Reflexion Divergences

Max Herrmann

Eines der Hauptfeatures von SEE ist die Visualisierung der Ergebnisse einer inkrementellen Softwareanalyse [Kos11].

Eine Reflexionsanalyse eines Softwareprojekts beschreibt die Grade der Entsprechung zwischen einem von einem Softwareentwickler definierten Modell einer höheren Abstraktionsstufe und dem Modell, welches aus der Implementierung einer Software konstruiert werden kann [MNS95]. Voraussetzung für diese Analyse ist, dass zwischen den Komponenten des Implementierungsmodells und den Komponenten des Architekturmodells manuell eine Beziehung, ein sogenanntes `Mapping`, definiert wird. SEE bezieht die Informationen über die vom Entwickler vorgegebene Architektur, die Informationen über das Modell, welches aus dem Quellcode eines Programms konstruiert wurde, sowie ein möglicherweise bereits existierendes Mapping zwischen diesen beiden Modellen aus zur Verfügung gestellten Dateien im GXL-Format, einem XML-basierten Format zur Darstellung von Graphen [HWS00].

Der Implementierungsgraph einer Software kann mithilfe einer statischen Codeanalyse gewonnen werden. Innerhalb von SEE werden keine Annahmen über die Bedeutung der Knoten und Kanten im angegebenen Implementierungsgraph gemacht, dennoch ist das Ziel von SEE die Visualisierung von Software und Softwareanalysedaten (siehe Abb. 29). Dementsprechend sollten die im Implementierungsgraph enthaltenen Knoten Implementierungskomponenten abbilden, und die enthaltenen Kanten bilden dementsprechend Abhängigkeiten zwischen diesen Komponenten ab. In der visuellen Darstellung des Implementationgraph werden die Blattknoten als dreidimensionale Blöcke dargestellt und innere Knoten des Graphen werden als ineinander verschachtelte Flächen visualisiert. Kanten zwischen Knoten werden mithilfe von Röhren dargestellt. Im aktuellen Stand werden die Kanten nur bei Selektion oder einem Hoverevent eines Knotens sichtbar (siehe Abschnitt 4.8).

Ein Architekturgraph, der von einem Nutzer vorgegeben wird, kann innerhalb von SEE erstellt und als GXL-Datei gespeichert werden. Genau wie im Implementierungsgraph sollten Knoten im Architekturgraph für Architekturkomponenten stehen, und Kanten für Abhängigkeiten zwischen diesen Komponenten. Bei der Visualisierung des Architekturgraphen werden die Knoten des Architekturgraphen als ineinander verschachtelte Flächen dargestellt und Kanten zwischen Knoten werden mithilfe von Röhren dargestellt. Die Kanten der Architektur werden im aktuellen Stand des Projekts permanent angezeigt (Abb. 30).

Die beiden Graphen werden aktuell innerhalb von SEE auf einem modellierten Tisch, in einer sogenannten `ReflexionCity` nebeneinander dargestellt (Abb. 31).

Das explizite Mapping der Komponenten der Implementierung auf die Komponenten der Architektur kann der Nutzer nun innerhalb von SEE realisieren, indem man die

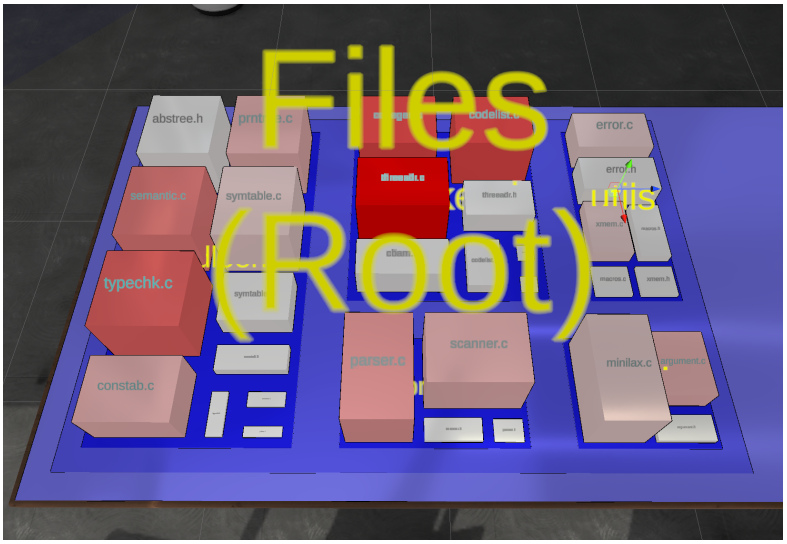


Abbildung 29: Beispiel der Visualisierung eines Implementierungsgraphen.

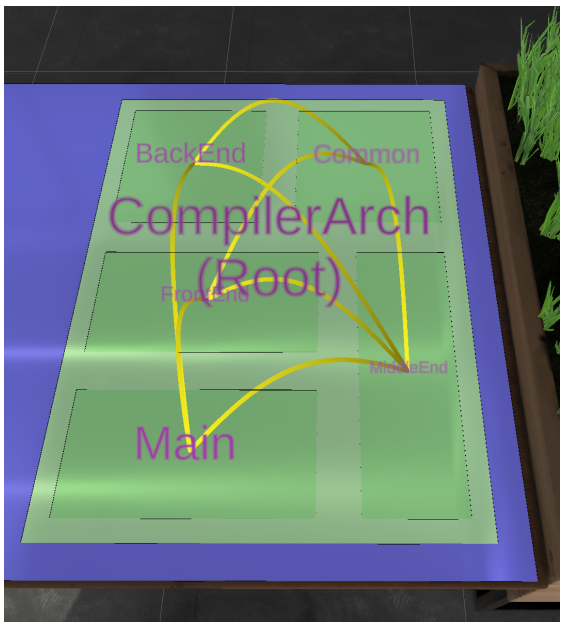


Abbildung 30: Beispiel der Visualisierung eines Architekturgraphen.

Komponenten aus dem Implementierungsgraph auf die Flächen, die die Komponenten des Architekturgraphen darstellen, zieht. Dabei werden die Komponenten aus dem Implementierungsgraph verkleinert auf den Architekturkomponenten platziert. Jede Veränderung löst bedingt durch die inkrementelle Eigenschaft der Reflexionsanalyse [Kos11] eine erneute Analyse aus, und die Kanten im Architekturgraphen verändern sich basierend auf den Ergebnissen der Analyse. Die Kanten innerhalb Reflexiongraph,

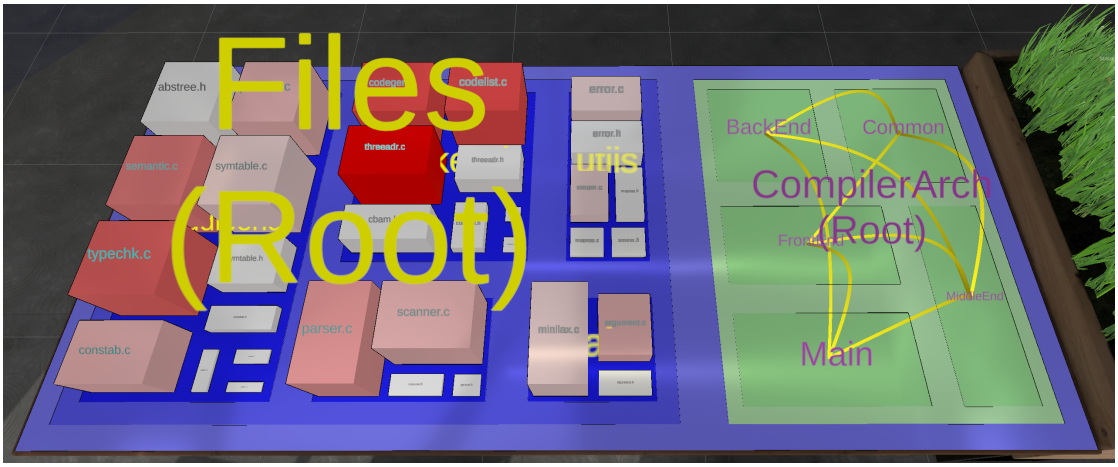


Abbildung 31: Darstellung des `Architecture Table` mit einem Implementierungsgraph (links) und einem Architekturgraph (rechts).

welche die Ergebnisse der Reflexionsanalyse darstellen, können folgende Zustände annehmen:

- Undefiniert (`Undefined`):
  Der initiale Zustand, bevor der Kante ein Zustand zugewiesen wird.
- Erlaubt (`Allowed`):
  Eine Abhängigkeit im Implementierungsgraph, die durch eine Kante im Architekturgraph repräsentiert werden kann.
- Divergent (`Divergent`):
  Eine Abhängigkeit im Implementierungsgraph, die durch keine Kante im Architekturgraph dargestellt wird.
- Abwesend (`Absent`):
  Eine Kante im Architekturgraph, die durch keine Abhängigkeit im Implementierungsgraph repräsentiert wird.
- Konvergent (`Convergent`):
  Eine Kante im Architekturgraph, die durch eine Abhängigkeit im Implementierungsgraph repräsentiert wird.
- Implizit erlaubt (`ImplicitlyAllowed`):
  Eine Abhängigkeit im Implementierungsgraph, die durch eine bereits erlaubte Abhängigkeit, der sie untergeordnet ist, implizit auch als erlaubt gekennzeichnet wird.
- Erlaubt abwesend (`AllowedAbsent`):

Eine Kante im Architekturgraph, die durch keine Abhängigkeit im Implementierungsgraph gedeckt wird, die aber mit dem `Architecture.Is_Optional`-Attribut gekennzeichnet wurde, und somit erlaubt ist.

- Spezifiziert (`Specified`):
  Der initiale Zustand einer Architekturkante, bevor diese durch die Analyse einen anderen Zustand bekommt.

- Nicht *gemapped* (`Unmapped`):
  Eine Abhängigkeit im Implementierungsgraph, die noch nicht *gemapped* wurde, und somit noch keinen Reflexionszustand angenommen hat.

Die Zustände der Kanten nach dem Mapping und nach der Reflexionanalyse werden in SEE durch Anfärbung gekennzeichnet. Architekturkanten, die noch nicht durch eine Implementierungsabhängigkeit bestätigt worden sind (`State: Absent`), werden in SEE gelb eingezeichnet (siehe Abb. 31). Entsteht während des Mappings eine Konvergenz zwischen einer Abhängigkeit im Implementierungsgraph und einer Kante im Architekturgraph, so wird diese Architekturkante hellgrün markiert (siehe Abb. 32).

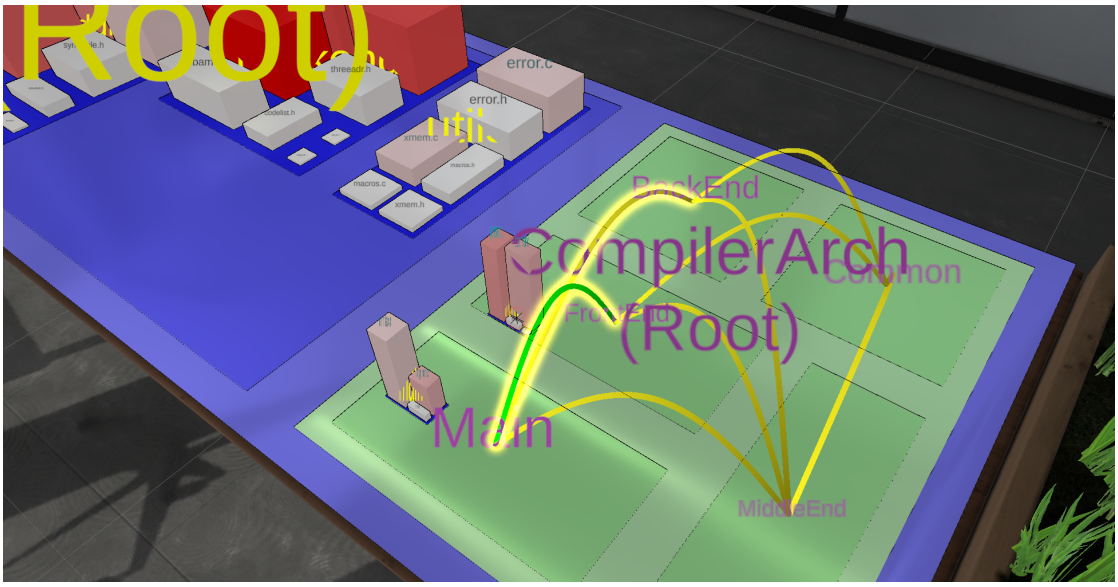


Abbildung 32: Mapping Prozedur - Zwei Komponenten wurden *gemapped*, mit einer konvergenten Abhängigkeit.

Registriert die Reflexionsanalyse eine divergente Abhängigkeit im Implementierungsgraph, die in der Architektur von keiner Kante bestätigt wird, so enthält diese Kante den Zustand `Divergent` und wird rot gekennzeichnet (siehe Abb. 33).

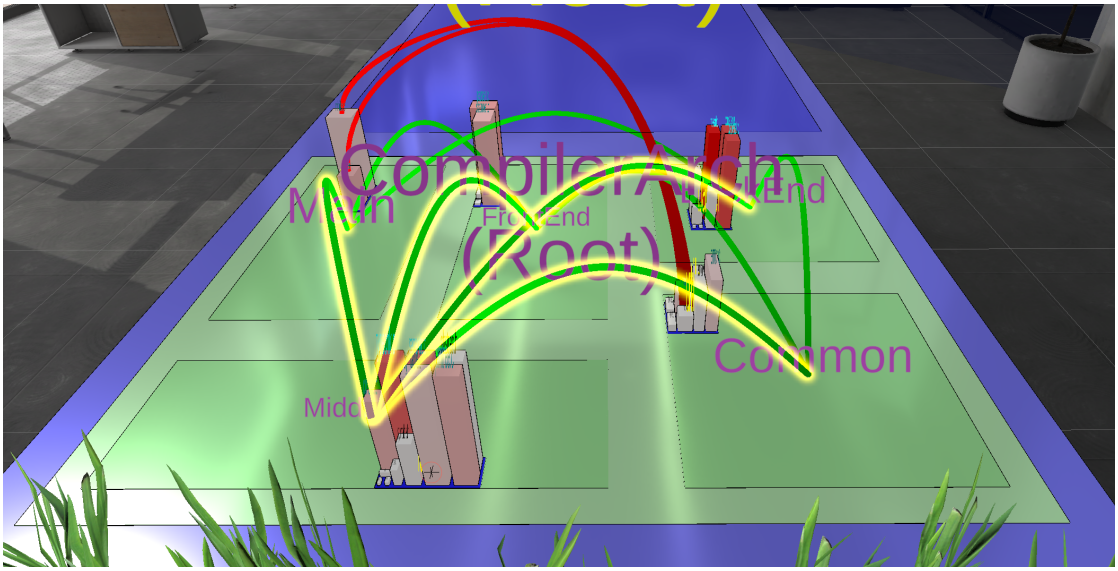


Abbildung 33: Mapping Prozedur - Vollständiges Mapping mit Divergenz(en).

4.7.1 Ziel

Dieses *Feature* beschäftigt sich mit genau diesen Kanten, die nach der Reflexionsanalyse den Zustand `Divergent` angenommen haben.

Ziel ist es, eine `User Action` bereitzustellen, die es dem Nutzer erlaubt, eine ausgewählte Divergenz aufzulösen, indem eine Kante in der Architektur erzeugt wird, die die ausgewählte Abhängigkeit in der Implementierung im Architekturgraph repräsentiert. Diese neue Kante soll beim Speichern des Graphen auch in der Architektur gespeichert werden. Der Hintergedanke dieses *Feature* ist, dass ein Nutzer in einem Szenario, in dem festgestellt wird, dass in der Architektur eine Abhängigkeit nicht definiert wurde, die erst durch Betrachtung der Implementation als nötig erkannt wurde, automatisch hinzuzufügen.

4.7.2 Ausgangslage

Um dieses *Feature* umzusetzen ist ein Verständnis der Repräsentation der einzelnen Graphen (Implementierung, Architektur und Mapping) vorausgesetzt: Die drei Graphen werden von SEE aus den GXL-Dateien geladen (wobei der Mappinggraph nicht unbedingt gegeben sein muss - in diesem Fall wird ein neuer, leerer Mappinggraph generiert). Anschließend werden den Graphen für Architektur und Implementierung künstliche Wurzelknoten hinzugefügt, falls es in den geladenen Daten mehr als einen Wurzelknoten gibt. Im Anschluss daran werden die drei Graphen mit einem weiteren

Wurzelknoten zu einem `ReflexionGraph` zusammengefasst. Dieser `ReflexionGraph` wird bei der Reflexionsanalyse analysiert, und da es sich um inkrementelle Reflexionanalyse handelt, erfolgt eine weitere Analyse bei jeder Veränderung. Da es sich bei SEE und der darin implementierten Reflexionsanalyse um ein sehr ausführlich implementiertes Projekt handelt, sind viele der Funktionen die zur Umsetzung dieses *Feature*s nötig sind, bereits vorhanden. Die Reflexionsanalyse stellt die expliziten (und darauf basierende implizite) Mappings, sowie Funktionen, die diese Mappings nach bestimmten Abbildungen durchsuchen können, bereit (z.B. `MapsTo` in der Klasse `ReflexionGraph`). Des Weiteren werden Funktionen zum Hinzufügen (`AddEdge()`), Entfernen (`RemoveEdge()`) und Überprüfen der Existenz von Kanten in bestimmten Teilgraphen (`IsInArchitecture()` / `IsInImplementation()` / `IsInMapping()` / `IsInReflexion()`) bereitgestellt. Um die aufzulösende Divergenz auswählen zu können, gibt es bereits die Möglichkeit in SEE, Kanten auszuwählen. Um die veränderte Architektur zu speichern, nachdem eine neue Kante hinzugefügt wurde, gibt es bereits die Funktionalität den `ReflexionGraph` in seine Komponenten zu zerteilen und die Graphen für Architektur, Implementierung und Mapping zu speichern.

4.7.3 Umsetzung

Äquivalent zu der im Onboardingprojekt des diesjährigen Bachelorprojekts SEE implementierten `MarkNodeAction` musste für dieses *Feature* eine neue `Action` hinzugefügt werden. Diese `AcceptDivergenceAction` musste außerdem als `ActionStateType` registriert werden, um dem Nutzer im `Actionmenü` angeboten zu werden.

Die `AcceptDivergenceAction` wird aktiviert, wenn eine Kante ausgewählt wird und verfährt grob betrachtet folgendermaßen:

```
1 wenn Gewähltes Objekt ist ein Kante dann
2     wenn Gewählte Kante hat den Status Divergent dann
3         Quelle ← Finde MapsTo der Quelle der Kante in der Tabelle mit
            impliziten Mappings
4         Ziel ← Finde MapsTo des Ziels der Kante in der Tabelle mit impliziten
            Mappings
5         Füge dem Architekturgraph eine neue Architekturkante (Quelle → Ziel)
            zu
```

Algorithmus 1: Auflösen einer Divergenz.

Wählt der Nutzer also eine Kante aus, die eine Divergenz darstellt, so wird diese rote Kante nach der einer Änderung des `ReflexionGraph` folgenden weiteren Reflexionsanalyse zu einer akzeptierten grünen Kante (Abb. 34).

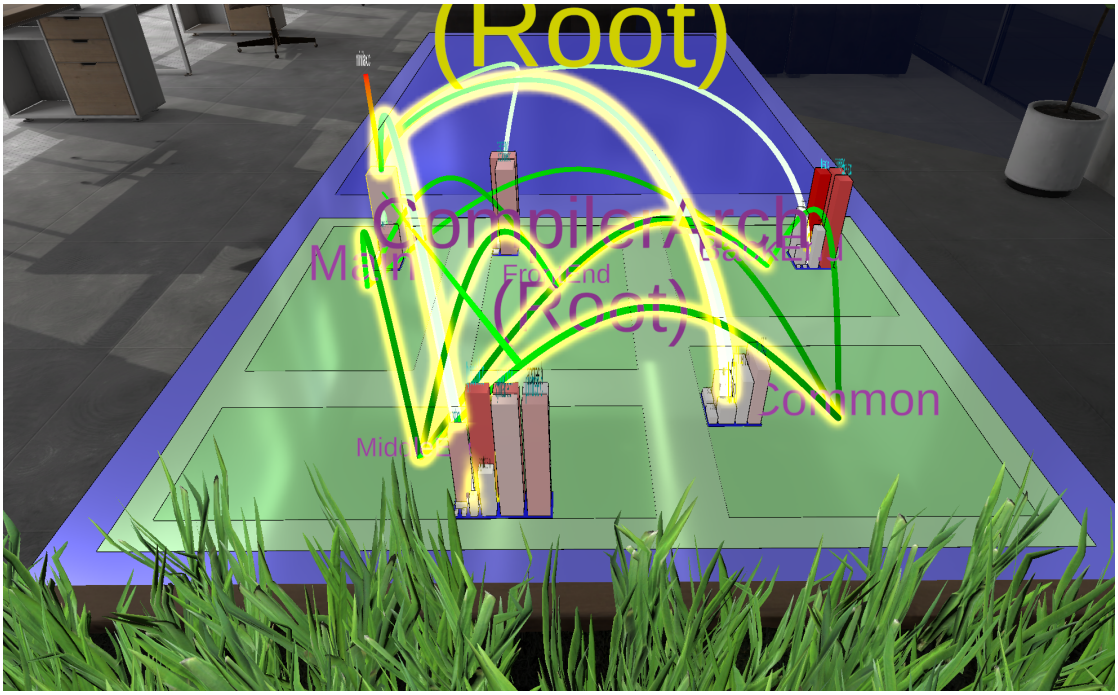


Abbildung 34: Zustand des `ReflexionGraph` nach Ausführen der `AcceptDivergenceAction` und Auswählen einer divergenten Kante.

4.7.4 Fazit

Die Ziele, die dieses *Feature* erfüllen sollte, wurden erfolgreich umgesetzt. Eine vom Benutzer ausgewählte Kante wird überprüft, und wenn es sich um eine Kante handelt, die eine Divergenz darstellt, wird eine diese Divergenz auflösende Kante in die Architektur eingefügt. Der inkrementell aktualisierte Reflexionsgraph führt eine weitere Reflexionsanalyse durch, und die davor als Divergenz erkannte Beziehung wird nun akzeptiert. Die bereits vor diesem *Feature* in SEE enthaltene Funktionalität des Auseinanderbauens des Reflexionsgraphen in Implementierungsgraph, Architekturgraph und Mappinggraph und das Speichern dieser einzelnen Graphen kann mit dem neu implementierten Laufzeitmenü (siehe Abschnitt 4.6) durch einen Klick auf den Save-Knopf erfolgen.

Durch die große Menge an bereits vorhandener Funktionalität im SEE Projekt, vor allem im Bereich der Reflexionsanalyse war die Hauptaufgabe dieses *Feature*s nicht das Schreiben neuer Funktionalität, sondern das Verknüpfen bereits vorhandener Funktionalität innerhalb einer vom User ausführbaren Action.

4.8 Gradual Edge Animations

Max Herrmann; Jonas Schramm

Eines der Hauptziele von SEE ist es, Software mithilfe der Code-City-Metapher zu visualisieren. Dafür werden Software-Projekte analysiert und SEE als Graphen in Form von GXL-Dateien übergeben. In diesen Graphen werden Komponenten der Software (wie z.B. Klassen, Namespaces, Interfaces) als Knoten, und Abhängigkeiten zwischen Komponenten als Kanten dargestellt.

Um diese Graphen zu visualisieren, werden die Knoten innerhalb von SEE als Flächen, Würfel oder Zylinder dargestellt. Ineinander verschachtelte Knoten können z.B. als ineinander verschachtelte Flächen implementiert sein. Abhängigkeiten zwischen Softwarekomponenten, die im Graphen als Kanten mit einem Quell- und einem Zielknoten enthalten sind, werden von SEE als Röhren, beziehungsweise Splinen (Funktionen, die stückweise aus Polynomen zusammengesetzt sind) zwischen dem Quell- und Zielknoten dargestellt. Dabei wird durch Farbgradienten verdeutlicht, welches Ende der Röhre dem Quell- und welches dem Zielknoten zuläuft.

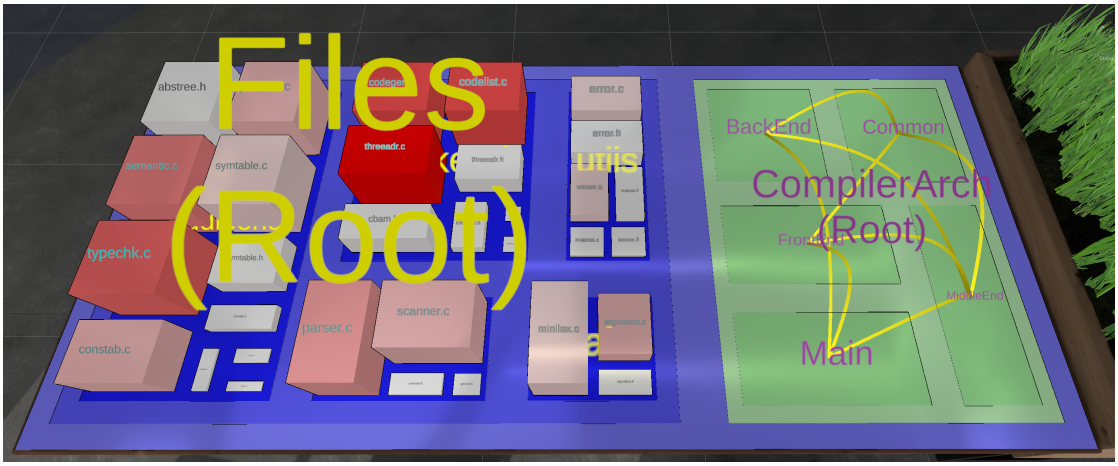


Abbildung 35: Beispiel einer Code-City mit Software-Komponenten als Knoten (Blöcke, Flächen) und Abhängigkeiten zwischen den Komponenten als Kanten mit Farbgradient.

4.8.1 Ziel

Dieses Feature fügt SEE eine neue Art und Weise, Kanten einzublenden, hinzu.

Diese neue Animation baut Kanten kontinuierlich vom Quell-Knoten zum Ziel-Knoten aus Richtung des Quell-Knotens auf, solange dieser Quell-Knoten ausgewählt oder fokussiert wird. Verliert der Quell-Knoten den Fokus, so werden die Kanten wieder graduell abgebaut.

Wird ein Knoten ausgewählt, werden die Kanten aufgebaut und bleiben bestehen, bis dieser Knoten abgewählt wird und den Fokus verliert.

Alle Knoten, die weder ausgewählt, noch im Fokus sind, haben keine angezeigten Kanten, die von ihnen ausgehen, mit Ausnahme von Kanten, die immer angezeigt werden (siehe Abb. 35, rechts).

Dieses Auf- und Abbauen der Kanten soll unabhängig vom festgelegten Kanten-Layout-Typ sein.

4.8.2 Ausgangslage

Bereits vor der Implementierung dieses Feature gab es in SEE eine Animation für Kanten, bei welchem die Kanten kontinuierlich ein- beziehungsweise ausgeblendet wurden. Ein Rahmen, in welchen wir unsere Animation einbauen konnten, war dementsprechend bereits gegeben.

Wie Kanten in SEE dargestellt werden, hängt vom konfigurierten *Kanten-Layout* ab. Die Berechnung der Kanten geschieht mithilfe der Bibliothek TinySpline [SK21a]. Diese C-basierte Bibliothek bietet unter anderem eine API mit Funktionen an, die Sub-Segmente von gegebenen Kanten berechnen. Diese Sub-Segmente werden dann innerhalb von SEE als Splinen dargestellt.

4.8.3 Umsetzung

Nach erfolgreichem Einbinden der neuen Version der TinySpline-Bibliothek konnten wir analog zur bereits existierenden *Fading*-Animation eine *Construction/Destruction*-Animation innerhalb der `EdgeOperator` Klasse definieren.

Um dem Nutzer die Wahl verschiedener Animations-Typen zu bieten, haben wir eine neue Klasse `EdgeAnimationKind` definiert, welche zum Zeitpunkt dieses Berichts nur die Optionen *Fading* (graduelles Ein- und Ausblenden), *Buildup* (graduelles Auf- und Abbauen aus Subsegmenten) und *None* (keine Animation), anbietet.

Die *Construction*- bzw *Descruction*-Action einer Kante werden abhängig von ihrem Selektions- und Hover-Status, bei gegebenem `EdgeAnimationKind` *Buildup* aktiviert.

Wir haben den graduellen Auf- und Abbau von Kanten realisiert, indem wir anstelle der vollständigen Spline adequat kleiner- beziehungsweise größerwerdende Sub-Splinen berechnen. Die gesamte Spline bekam dafür zwei neue Attribute - `SubsplineStartT` und `SubsplineEndT` - welche den Anteil der Sub-Spline an der ganzen Spline mit der Domäne $[0, 1]$ beschreiben. Da sich die Kanten nur von Startpunkt zum Zielpunkt aufbauen

und rückwärts vom Zielpunkt zum Startpunkt abbauen, bleibt der Startpunkt kontinuierlich auf `0.0`. Der Endpunkt `SubSplineEndT` wird von den neu definierten Funktionen `Construction` und `Destruction` bei einem Auswahl- oder Hover-Event kontinuierlich gegen 0 oder gegen 1 verschoben, abhängig vom aktuellen Stand, um einen gleichmäßigen Auf- oder Abbau zu animieren. Bei jedem Update des `SEESpline`-Objekts, welches eine Kante repräsentiert, wird also eine im Vergleich zum vorigen Stand potenziell verlängerte oder verkürzte Sub-Spline dargestellt werden.

Da das Aufbauen einer Kante mit konstanter Geschwindigkeit weniger natürlich für einen Beobachter ist, wurde mithilfe von *Easing*[3] die Aufbaugeschwindigkeit transformiert (siehe Abb. 36). *Easing* bezeichnet das Anpassen von Prozessen wie zum Beispiel Bewegungen, die linear modelliert werden, sodass diese einen natürlicheren, nicht linearen Verlauf haben.

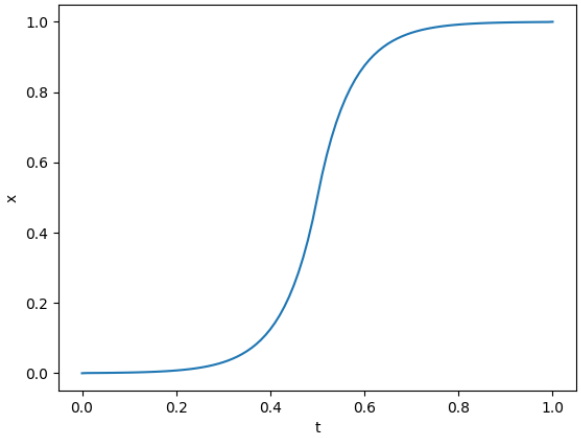
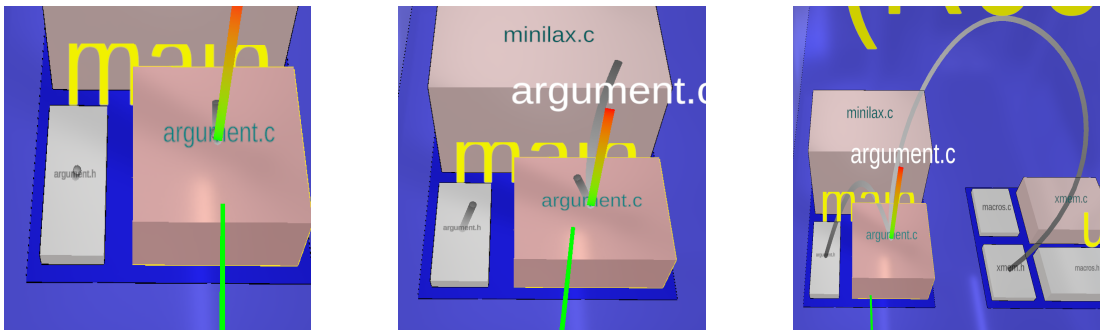


Abbildung 36: Verwendete Easing Transformation



(a) Step 1 (b) Step 2 (c) Step 3

Abbildung 37: Verschiedene Stadien des Kantenaufbaus und -abbaus.

[3] https://easings.net/

4.8.4 Fazit

Die Einbindung der neuen TinySpline API und das Erstellen der neuen Animations-Funktionalität waren erfolgreich. Der Nutzer kann nun zwischen den Kantenanimationen *Fading* und *Buildup* wählen.

Das Aufbauen der Kanten von der Quelle zum Zielknoten gibt dem Nutzer ein direktes Gefühl dafür, in welche Richtung die Abhängigkeit verläuft, während das graduelle Einblenden von Kanten konnte diese Information nicht so klar vermittelt.

Das Verstecken der Kanten zwischen unfokussierten Knoten ist ein guter Weg, die Übersichtlichkeit der Code-City zu gewährleisten.

Die Implementierung dieses Features und das nötige Debugging der TinySpline-Bibliothek bot uns dank der transparenten Arbeitsweise des Autors von TinySpline die Möglichkeit, die Interaktion verschiedener Programmiersprachen zu erleben, und ein Sprachenübergreifendes Debugging zu beobachten.

Eine Möglichkeit, das Feature zu erweitern wäre es, dem Nutzer die Möglichkeit zu geben, die Animation abzuschalten, und alle Kanten einzublenden. Dies wäre vor allem in Code-Cities mit wenigen Kanten ein wertvolles Feature, wenn man sich einen ersten Überblick verschaffen möchte, welche Software-Komponenten wie stark von anderen Software-Komponenten abhängig sind.

4.9 Incremental Layout

Jonas Schramm

SEE bietet die Möglichkeit, Software mithilfe der Software-City-Analogie zu visualisieren. Einen besonderen Anwendungsfall ist dabei sie zeitliche Entwicklung der Software und der erhobenen Metriken.

Ein konkretes Beispiel könnte man wie folgt formulieren: Man hat von einer bestimmten Software durch ein Versionskontrolle-Tool den Entwicklungsstand zu n verschiedenen Zeitpunkten. Zu jedem Zeitpunkt hat man insbesondere Kenntnis von der Dateistruktur der Quelltextdateien und von der *Lines of Code* Metrik jeder Quelltextdatei. Diese Daten möchte man nun geeignet grafisch darstellen mithilfe der Software-City-Analogie. In Abb. 38 kann man die Darstellung einer Software in SEE beobachten. Die grünen

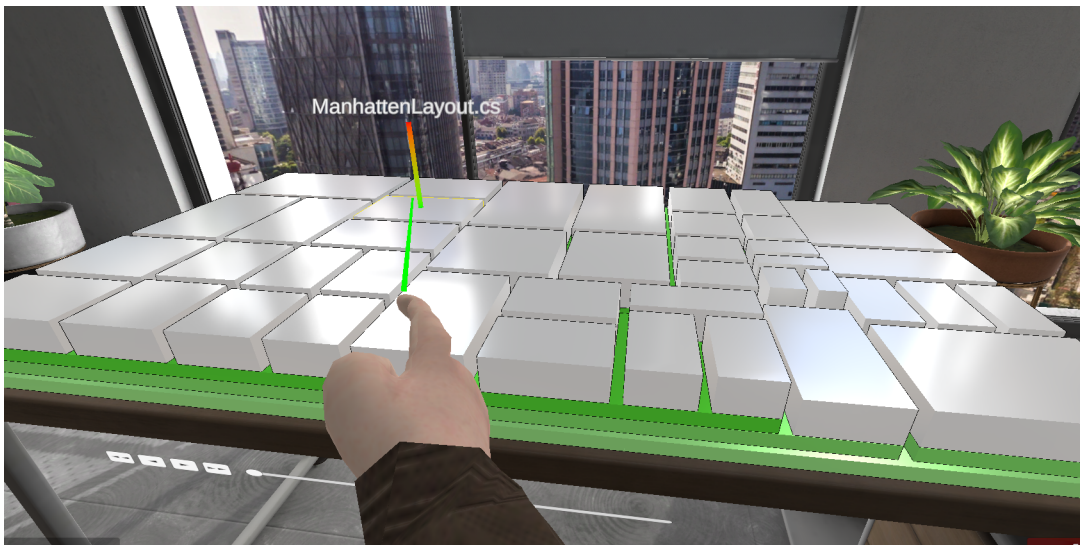


Abbildung 38: SEE als Software-City zu einem frühen Entwicklungsstand

Schichten stellen die Ordnerstruktur dar. Die weißen Blocken repräsentieren je eine C# Datei. Den Repräsentanten eines Ordners als auch einer C#-Datei wird als Node bezeichnet. Farbe und Skalierung der Nodes können also verschiedene Metriken ausdrücken. In diesem Fall steht die Grundfläche einer Node in Relation zur *Lines of Code* Metrik. Die Farbe repräsentiert den Node-Typen und die Höhe ist konstant.

Bei dem Berechnen so eines Layouts stehen verschiedene Aspekte im Vordergrund. Auf der einen Seite soll ein Layout eine gute visuelle Qualität haben. Das bedeutet, dass es einen Betrachter leicht fällt, die dargestellten Zusammenhänge und Größen zu erkennen. Im Fall eines Rechteck-Layouts ist dabei vor allem die Aspect-Ratio der Nodes von Interesse. Beziehungsweise die Aspect-Ratio der Grundfläche einer Node, also das Verhältnis der langen zu der kurzen Seite. Der Hintergedanke ist, dass es einem Beobachter leichter fallt, die Größen von Rechtecken zu vergleichen, wenn diese

annähernd quadratisch sind. Man möchte also innerhalb eines Layouts Nodes haben, die eine möglichst geringe Aspect-Ratio aufweisen.

(a) Layout mit schlechter Aspect-Ratio (b) Layout mit perfekter Aspect-Ratio

Abbildung 39: Beispiel Aspect-Ratio

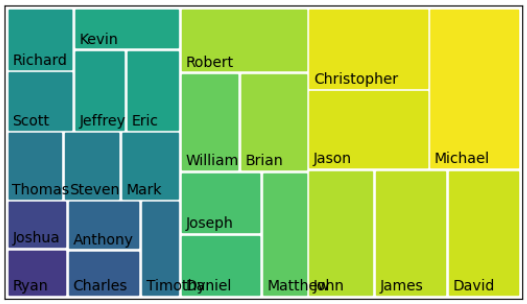
Einem Beobachter fällt es typischerweise schwer zu erkennen, dass in Abb. 39a alle Rechtecke die gleiche Fläche haben, während es in Abb. 39b offensichtlich ist. Auf der anderen Seite sollen die Layouts stabil in der zeitlichen Entwicklung sein. Mit Stabilität bezeichnet man den Grad, mit dem sich ein Layout ändert, in Abhängigkeit einer Änderung in den Daten.

Beispielhaft kann man in Abb. 40 die zeitliche Entwicklung der gängigsten Vornamen in den USA von 1970 zu 1980 sehen, die Einfärbung dient hier nur der besser Unterscheidbarkeit. Das Layout für die Dekade wurde 1980 zweimal berechnet. Im Layout in Abb. 40b wurde die Stabilität in der zeitlichen Entwicklung nicht weiter berücksichtigt, während das Layout in Abb. 40c dem Layout der Dekade 1970 in Abb. 40a deutlich ähnlicher ist. Es ist zu erkennen, dass Nodes wie zum Beispiel *Ryan* und *Joshua*, die in beiden Dekaden vorkommen, von Abb. 40a zu Abb. 40b deutlich mehr ihre Position ändern als von Abb. 40a zu Abb. 40c. Einem Beobachter, der erst das Layout der Dekade 1970 und danach ein Layout der Dekade 1980 betrachtet, wird sich im stabilen Layout deutlich besser zurechtfinden.

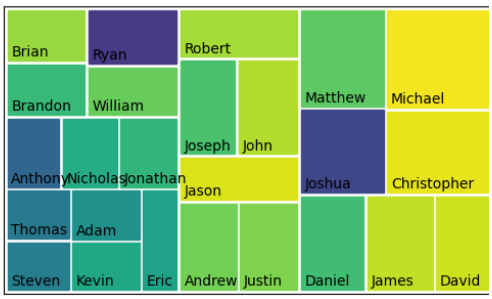
Die Optimierung der beiden Anforderungen Stabilität und visueller Qualität stellt einen Interessenkonflikt dar.

4.9.1 Ziel

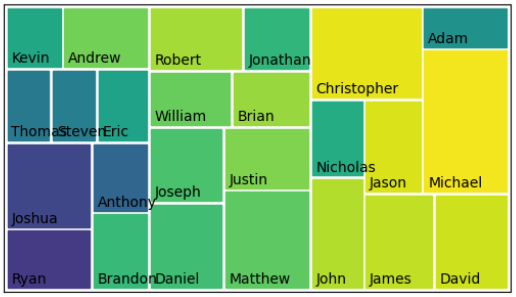
In [SSV18] stellen Sondag et al. einen Algorithmus vor, um das Problem der Stabilität von Layouts anzugehen und dabei gleichzeitig gute Ergebnisse hinsichtlich der visuellen Qualität zu erreichen. Das Ziel des Features *Incremental Layout* ist das Implementieren des Algorithmus von Sondag et al., um so eine die zeitliche Entwicklung darstellen zu können und im Vergleich zu anderen Algorithmen eine höhere visuelle Qualität und



(a) Layout der Dekade 1970



(b) Layout der Dekade 1980, geringere Stabilität zu Abb. 40a



(c) Layout Dekade 1980, höhere Stabilität zu Abb. 40a

Abbildung 40: Die 25 häufigsten männlichen Vornamen in den USA in den Dekaden 1970 und 1980

Stabilität zu erreichen.

4.9.2 Ausgangslage

Beim Anfang der Implementierung des Features verfügte SEE bereits über verschiedene Funktionalitäten, auf die man zurückgreifen konnte.

In SEE ist das Darstellen einer Folge von CodeCitys, als zeitliche Entwicklung bereits implementiert und wurde während des Bearbeitens noch einmal überarbeitet. Die komplette Realisierung eines Layouts in der virtuellen Umgebung ist entsprechend kein Teil des Features, im Gegenteil bietet die Klasse SEE damit bereits eine klare Schnittstelle zwischen dem Berechnen eines Layouts und dem Darstellen.

Weiter sind in SEE bereits mehrere verschiedene Layouts implementiert, die man als Orientierung nutzen kann.

4.9.3 Umsetzung

Die Umsetzung des Features begann mit der Studie der Literatur zu diesem Thema, wobei drei Paper im Vordergrund standen. Zum einen [SSV18] von Sondag et al., in dem die grundlegende Idee der sogenannten Local Moves beschrieben und der Algorithmus vorgestellt wird, der implementiert werden soll.

Mithilfe der Local Moves lässt sich ein Layout in ein anderes ähnliches Layout transformieren, wobei die Änderung der Node-Positionen skalierbar ist, also eine gewisse Stabilität zwischen den Layouts gewährleistet werden kann. Weiter wird ein Algorithmus von Nagamochi und Abe verwendet, der in [NA07] beschrieben wurde und mit dem ein initiales erstes Layout errechnet wird, welches gute Eigenschaften bezüglich der Aspect-Ratios der Nodes hat. Ein weiterer Algorithmus basiert auf [Epp+09] von Eppstein et al. Der Algorithmus ist ein Gradientenverfahren, mit dem die Nodes eines Layouts auf andere Größen angepasst werden können, ohne die relative Position der Nodes zueinander zu verändern.

Bei der Umsetzung der Algorithmen wurde ein weiterer Synthese-Schritt gewählt, nämlich das Programmieren der Algorithmen in `python`. Das erwies sich als sehr hilfreich, da man so nicht nur erstes Ansätze für eine Architektur des Codes erhält, sondern auch erste typische Fehler ausmachen konnte. Es bot die Möglichkeit sicherzustellen, dass man die Algorithmen voll verstanden hat und umsetzen kann, ohne sich dabei um eine Einpassung in die bereits bestehende Codebase kümmern zu müssen. Auch das Debuggen eines `python scripts` hat sich als deutlich einfacher herausgestellt als das Debuggen in Unity.

Neben der Klasse `IncrementalTreeMapLayout`, die, die Schnittstelle zum restlichen SEE-Projekt realisiert und als Art Controller den Algorithmus auf einer höheren Abstraktionsebene ausführt, gibt es weitere Klassen, in denen die einzelnen Teile des Algorithmus implementiert wurden.

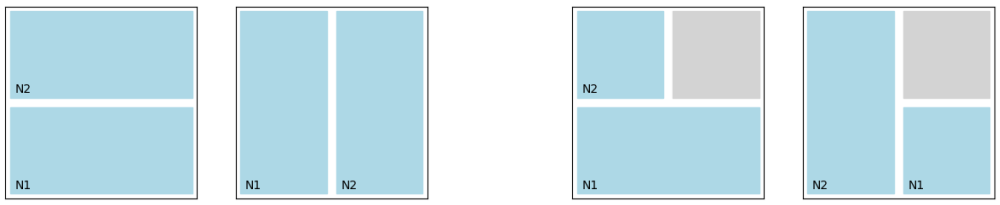
- `static class CorrectAreas` eine Klasse, die den Algorithmus aus [Epp+09] implementiert.

- `static class Dissect` eine Klasse, die den Algorithmus von [NA07] implementiert.
- `static class LocalMoves` eine Klasse, die den Algorithmus aus [SSV18] implementiert.
- `abstract class LocalMove` diese Klasse, symbolisiert einen Local Move.
- `class FlipMove : LocalMove` eine Klasse, die einen Flip Move implementiert.
- `class StretchMove : LocalMove` eine Klasse, die einen Stretch Move implementiert.

Um ein Layout in einer für die Algorithmen sinnvollen Art und Weise darstellen zu können, gibt es noch weitere Klassen:

- `class TNode` repräsentiert einen Datenpunkt und seine Darstellung im Layout.
- `class TRectangle` repräsentiert eine konkrete grafisch Darstellung einer Node.
- `class TSegment` repräsentiert eine Gerade im Layout, die die Rechtecke separiert.

Das Verhalten der Klassen `Dissect`, `CorrectAreas` und `LocalMoves` entspricht im Wesentlichen den in der jeweiligen Literatur vorgeschlagen Algorithmen. Die Klasse `LocalMoves` stellt die Funktion zur Transformation eines Layouts mithilfe von Local Moves bereit. Darüber hinaus implementiert die Klasse auch Funktionen zum Löschen und Hinzufügen von Nodes. Die Klasse `Dissect` erstellt ein Rechteck-Layout durch das rekursive Splitten von Rechtecken. In `CorrectAreas` ist eine numerische Methode definiert. Sie basiert auf der funktionalen Abhängigkeit der Fläche der Nodes von den Segmenten. Die Methode berechnet als Gradientenverfahren in jedem Schnitt die Jacobi-Matrix dieser Funktion und anschließend eine Verschiebung der Segmente, um so die Fläche aller Nodes zu justieren.



(a) Flip Move (b) Stretch Move

Abbildung 41: Die zwei Arten von Loval Moves

Die Klassen `FlipMove` und `StretchMove` realisieren die zwei unterschiedlichen Arten eines Local Moves (siehe Abb. 41). Eine Instanz einer solchen Klasse entspricht dann einem konkreten Local Move, der mit der Methode `localMove.apply()` ausgeführt werden kann.

Aus der Erfahrung der `python`-Synthese ergab sich die Maxime, die Local Moves mit möglichst klaren Schnittstellen zu konzipieren, da hier ein großes Fehlerpotential liegt. Tatsächlich hat sich das saubere Design in dieser Hinsicht auch ausgezahlt. Dafür hat es andere Aspekte verkompliziert, was darin begründet liegt, dass eine Instanz eines `LocalMove` Erben nicht auf einer Kopie des Layouts ausgeführt werden kann. Die Layouts Abb. 40a, Abb. 40b und Abb. 40c wurden mit dem Python-Script berechnet.

Sowohl für die Local Moves, also auch für den CorrectAreas Algorithmus ist es wichtig, dass jede Node Kenntnis hat über die vier Segmente, die das von der Node eingenommene Rechteck begrenzt. Ein Segment wiederum muss Kenntnis haben, über alle Nodes, die an dem Segment anliegen. Um diese Information auf einen möglichst nativen Weg zur gewährleisten, wurden die Klassen `TNode`, `TSegment` und `TRectangle` eingeführt.

4.9.4 Fazit

Der Algorithmus konnte erfolgreich implementiert werden und löst das beschriebene Problem hinlänglich.

Die Laufzeit stellt ein Problem dar. Beim Berechnen von größeren Layouts oder beim Berechnungen mit einer hohen Anzahl von Local Moves zwischen den einzelnen Layouts, wird die Laufzeit untragbar. Weitere Heuristiken und eine effizientere Programmierung, insbesondere eine Restrukturierung einzelner Rechenschritte in parallelisierbare Jobs, können eine sinnvolle Erweiterung sein.

Aufbauend auf den Local Moves kann man versuchen, ein weiteres Qualitätsmerkmal zu optimieren. Kanten werden in dem jetzigen Layout nicht mit einbezogen, obwohl es für die visuelle Qualität von Vorteil wäre, wenn Nodes, die durch Kanten verbunden sind, eher nahe beieinander liegen. Auch dieses Feature würde eine gute Ergänzung bilden.

4.10 VRIK

William Behnke; Yvo Muskulus

FinalIK ist eine Sammlung von Inverse Kinematics-Lösungen und enthält ein VR-IK System, mit dem es möglich ist, die Bewegungen des gesamten Körpers auf einen Avatar zu übertragen [Roo23]. Indem das VR-IK System implementiert wird, können realistischere Charakterbewegungen in der Virtual Reality ermöglicht werden. Diese realistischen Animationen basieren auf Hochrechnungen der jeweiligen Positionen der einzelnen VR Komponenten.

Obwohl SEE bereits ein Aim IK und Look-At IK System für die Desktop-Variante besitzt, ist dieses in der VR Umgebung nur bedingt nutzbar. Um eine immersive Erfahrung in VR zu gewährleisten, ist es erforderlich, das vorhandene System durch das VR-IK System zu ersetzen.

Das Ziel dieses Features besteht darin, durch die Verwendung eines VR Headsets und der dazugehörigen Controller eine vollwertige qualitativ hochwertige Animation, welche den gesamten Körper des Charakters umfasst, zu ermöglichen. Dieses Feature wurde von William Behnke und Yvo Muskulus entwickelt.

4.10.1 Umsetzung

Die Implementierung von VRIK verlief für uns erfreulicherweise unkompliziert. Durch ein Tutorial Video des Erstellers konnte das Feature schnell und einfach implementiert werden [Lan17]. Ein wesentlicher Faktor bestand darin, die Position des VR Headsets und der Controller zu tracken und dem VR-IK Solver zu übermitteln. An dieser Stelle hat Rainer uns geholfen und ein Prefab (*XRRig*) zur Verfügung gestellt, das entsprechende Offsets und Einstellungen enthält. In dem Script *AvatarAdapter.cs* wird dieses Prefab zur Laufzeit instanziiert und die Positionsdaten an den VR-IK Solver in der Methode `SetupVRIK()` übergeben.

Ein weiterer Schritt bestand darin, den ursprünglichen Animator Controller durch den `VRIKAnimatedLocomotion.controller` zu ersetzen, welcher im FinalIK Paket enthalten war. Der ursprüngliche Animator verfügte nicht über ausreichend Animationen, um die gesamte Bewegung des Charakters umfassend zu steuern. Durch die Integration des neuen Controllers konnten wir nun auf ein breiteres Spektrum an Animationen zurückgreifen.

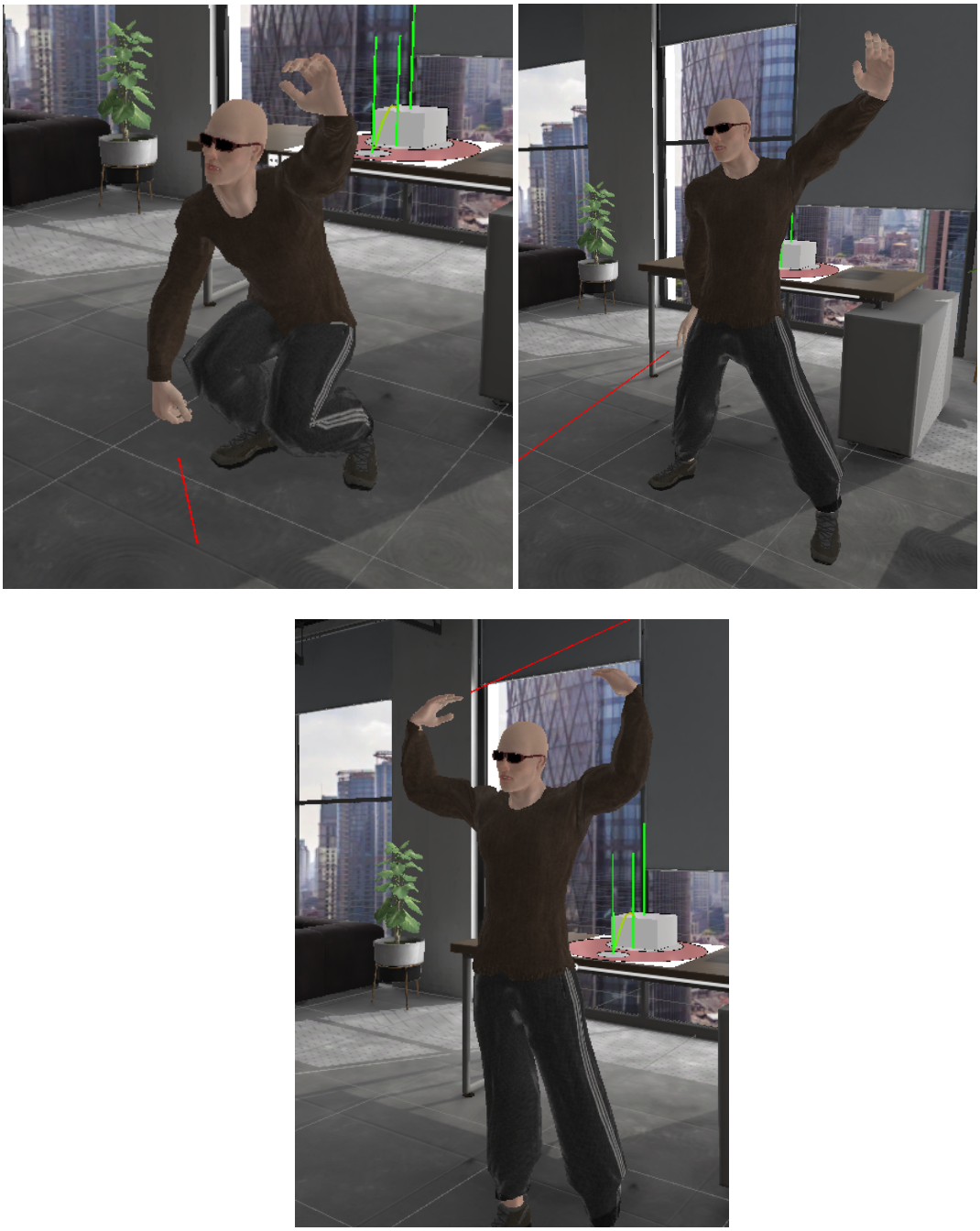


Abbildung 42: Beispiele neuer Animationen.

Ein weiterer Schwerpunkt unserer Arbeit lag auf der allgemeinen Netzwerk-Synchronisation. Es war von größter Bedeutung sicherzustellen, dass die Bewegungen des Avatars zwischen verschiedenen Clients korrekt und nahtlos synchronisiert wurden. Als Grundlage

diente uns hier die Klasse *Synchronizer*, welche regelmäßig Positionsdaten an alle Clients überträgt. Dadurch konnten wir unsere Netzwerkkomponente schnell und einfach implementieren. Zusätzlich wurden Anpassungen am Laserpointer vorgenommen, sodass dieser nun mit VR-IK funktioniert und auf allen Clients synchronisiert wird.

4.10.2 Das Offset Problem

William Behnke

Nachdem das Feature in den Master-Branch gemerged wurde, trat am Projekttag und bei der Implementierung des Features Autohand (Abschnitt 4.12) ein Fehler auf, den wir als „Offset-Fehler“ bezeichneten. Das Offset Problem trat auf, wenn das VR Headset nicht perfekt auf dem Kopf positioniert war, und führte dazu, dass die Hände des Avatars das Tracking verloren. Interessanterweise bemerkten wir das Offset Problem lange Zeit nicht, da wir zu Hause eine auf uns angepasste Kalibrierung des Headsets verwendeten. Erst als wir das VR-System in einer anderen Umgebung testeten, wurde das Offset Problem offensichtlich. Die Hände des Avatars reagierten nicht mehr auf die Bewegungen der Controller und das Tracking war nicht existent.

Um das Offset Problem zu lösen, führten wir verschiedene Anpassungen durch. Wir haben die Offsets für die Controller und das VR-Headset angepasst, um eine genauere Übertragung der Bewegungen zu erreichen. Außerdem haben wir herausgefunden dass es entscheidend ist, dass das Headset fest und gut eingestellt auf dem Kopf getragen wird. Eine sorgfältige Kalibrierung des VR-Systems kann dazu beitragen, dieses Problem zu minimieren.

4.10.3 Fazit

William Behnke

Zusammenfassend lässt sich sagen, dass die Implementierung von VRIK im Großen und Ganzen unkompliziert verlaufen ist, abgesehen von dem OffSet-Problem, das erst nachträglich entdeckt wurde. Das Hinzufügen der VRIK Komponente, der Austausch des Animator Controllers und die Netzwerk-Synchronisation waren vergleichsweise problemlos umzusetzen.

Im Rahmen dieser Arbeit wurden verschiedene Aspekte der Avatar-Implementierung und -Anpassung untersucht und diskutiert. Es wurde festgestellt, dass das Ausblenden des Kopfes, um einen Blick in den Körper zu verhindern, eine mögliche Maßnahme ist, die bei der Entwicklung zukünftiger Avatare berücksichtigt werden sollte. Durch eine feinere Justierung der VRIK Parameter kann die Genauigkeit der Bewegungsübertragung weiter verbessert werden. Hierbei würde es sich jedoch um eine zeitintensive Aufgabe handeln. Darüber hinaus wurde erkannt, dass ein Skript entwickelt werden könnte, um automatisch den Offset anzupassen, sobald die Fehlerposition erreicht wird. Diese Erkenntnis kam jedoch erst spät im Rahmen der Arbeit zur Integration der

Autohand-Funktionen auf. Wir finden die Idee faszinierend, weitere Möglichkeiten zu finden, VRIK anzupassen und zu verfeinern. Hier könnten beispielsweise alternative Eingabemethoden wie Handgesten integriert werden. Dies würde den Benutzern mehr Flexibilität und Immersion bieten und die Bedienung des Avatars intuitiver gestalten. Zusätzlich könnte logischerweise das Feature um weitere Animationsoptionen erweitert werden. Wir finden VRIK stellt allgemein eine gute Grundlage da.

4.11 HTC FacialTracker

William Behnke; Yvo Muskulus

Der HTC Facial Tracker[4] ist ein Zusatzgerät (siehe Abb. 43), das speziell für die Erfassung und Verfolgung von Gesichtsbewegungen entwickelt wurde. Da Gestik und Mimik ein essentieller Bestandteil der kollaborativen Arbeit in SEE werden soll, sind wir dank dieses Gerätes in der Lage, Lippenbewegungen zu verfolgen und auf dem Avatar abzubilden.

Bevor wir mit der Integration des Facial Trackers begonnen haben, gab es keine komplexe Gesichtsanimation. In unserem Projekt SEE haben wir vorher die UMA 2 - Unity Multipurpose Avatare [Gro23] benutzt, welche vor der Implementierung mithilfe der SALSA Lipsync-Erweiterung einen rudimentären Avatar darstellten [Stu23]. Zusätzlich gab es eine einfache Kopfdreh-Routine, um zu verhindern, dass die Charaktere zu statisch wirken.

Die Implementierung des Facial Trackers in unsere Software ermöglicht den Benutzern von SEE die Echtzeiterfassung ihrer Gesichtsausdrücke in der virtuellen Umgebung. Dies trägt zu einer immersiven und interaktiven Erfahrung bei, da neben den visuellen Darstellungen der Code-Visualisierung nun auch der emotionale Zustand der Benutzer reflektiert werden kann. Durch diese Erweiterung entsteht das Potenzial für eine noch effektivere Teamarbeit, da der emotionale Aspekt eine wichtige Rolle in der Kommunikation und im Informationsaustausch spielt. Dieses Feature wurde von William Behnke und Yvo Muskulus implementiert.[5]

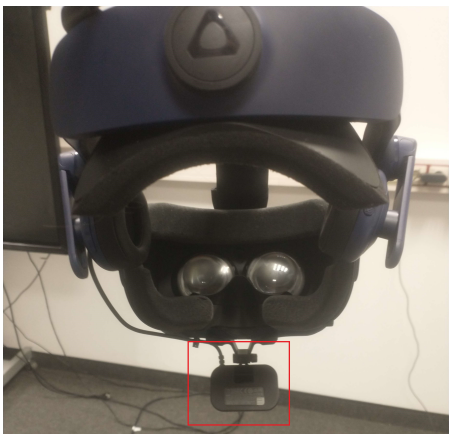


Abbildung 43: Der Facial Tracker am Virtual Reality Headset.

[4] VIVE Facial Tracker: https://www.vive.com/de/accessory/facial-tracker/.
[5] Autor der Einleitung: William Behnke

4.11.1 Umsetzung

Demo-Scene: **Yvo Muskulus**

Zu Beginn des Projektes haben wir uns zunächst mit der Facial Tracker Unity Demo auseinander gesetzt, um die Lippenverfolgung in Aktion zu sehen und einen Eindruck davon zu erhalten, wie der Facial Tracker funktioniert.

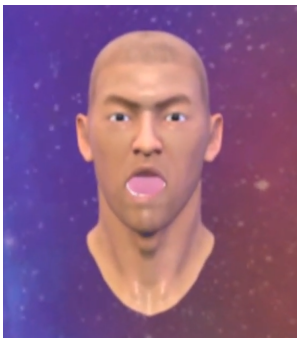


Abbildung 44: Ausschnitt aus der SRanipal Facial Tracker Demo

Als erstes Problem ist dabei aufgefallen, dass der Avatar in der Demo Scene sich grundlegend von dem benutzten Avatar-System in SEE unterscheidet. Damit der Facial Tracker vernünftig arbeiten kann, muss der Avatar Blendshapes besitzen, die von dem Facial Tracker angesprochen werden können. Blendshapes werden benutzt, um ausdrucksstarke Gesichtsanimationen zu realisieren.

In SEE kommen wie bereits erwähnt Unity Multipurpose Avatare zum Einsatz, die keine Blendshapes bieten, die von dem Facial Tracker angesprochen werden könnten [Gro23]. Im Verlauf haben wir uns mit mehreren Möglichkeiten beschäftigt diese Blendshapes hinzufügen. Am Anfang haben wir versucht, in Blender eigene Blendshapes zu erstellen und dem UMA Avatar hinzuzufügen, doch mangels Erfahrung mit Blender hatten wir keinen Erfolg. Für uns war es ebenfalls zu zeitaufwendig, den Umgang mit Blender zu erlernen. Des weiteren haben wir uns mit der Möglichkeit beschäftigt, die Blendshapes des Demo-Avatars auf den UMA-Avatar zu übertragen, doch auch hier scheiterte der Versuch, da es bedingt durch die unterschiedlichen Avatar-Systeme technisch nicht umsetzbar war. Die Lösung des Problems haben wir letztendlich im Unity Forum gefunden [Mar23]. Hier ist zu erwähnen, dass der UMA-Charakter einen sogenannten *ExpressionPlayer* verwendet, um Gesichtsanimationen auf dem Avatar zu realisieren. In dem Forum Eintrag wird beschrieben, wie Fake-Blendshapes kreiert werden können, um den *ExpressionPlayer* zu beinflussen. Wichtig dabei ist es, wie in Abb. 45 zu sehen, dass die Fake-Blendshapes korrekt benannt wurden, damit sie durch den Facial Tracker angesprochen werden können .

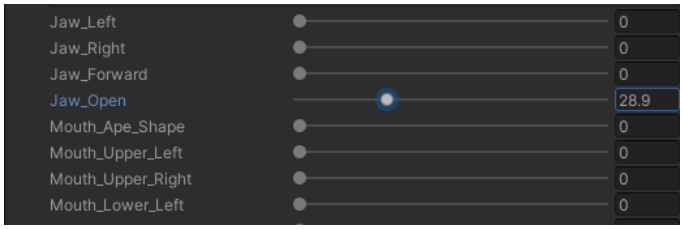


Abbildung 45: Durch diese modifizierte und angepasste Wertetabelle erfolgt eine Mundöffnung.

4.11.2 Blendshapes

William Behnke

Zunächst haben wir mit der Erstellung der sogenannten Fake-Blendshapes, auf Basis des Forum Eintrags, begonnen. Die Fake-Blendshapes wurden entsprechend zugeordnet. Durch diese Zuordnung konnte eine direkte Verbindung zwischen den vom Facial Tracker erkannten Bewegungen und den virtuellen Ausdrücken im Spiel hergestellt werden. Bei der Umsetzung dieser angepassten Fake-Blendshapes traten jedoch Probleme auf, da einige Blendshapes des Facial Trackers nicht mit dem *ExpressionPlayer* dargestellt werden konnten. Dies hatte verschiedene Gründe. Einige Animationen waren nicht in der Originalversion des *ExpressionPlayers* enthalten und mussten daher mithilfe von verschiedenen kombinierten Animationen hergeleitet werden. Andere Blendshapes wiederum passten nicht zur internen Logik des *ExpressionPlayers* oder erforderten weitere spezifische Anpassungen, um korrekt funktionieren zu können. Diese Anpassungen wurden in der Regel durch langsame und mühsame Versuche entwickelt.

Abbildung 46: Beispiele möglicher Mundbewegungen.

4.11.3 SRanipal Software

William Behnke

Ein weiterer wichtiger Aspekt der Implementierung war die Integration von SRanipal, einer Zusatzsoftware, die für die ordnungsgemäße Funktion des Facial Trackers erfor-

derlich ist.[6] SRanipal bietet erweiterte Funktionen und Schnittstellen für die Gesichtsverfolgung und -erkennung. Um den Facial Tracker erfolgreich nutzen zu können, musste die SRanipal-Software in das System integriert und gestartet werden. Die Einbindung des Facial Trackers erfolgte schließlich über den AvatarAdapter, der als Schnittstelle zwischen dem Tracker und dem Avatar-System fungiert. Um sicherzustellen, dass das System ordnungsgemäß funktioniert, wurde dieser entsprechend angepasst. Es wird im Laufe der Startsequenz überprüft ob die SRanipal-Software läuft und ob der Facial Tracker erkannt wurde. Erst dann wurden die Fake-Blendshapes und das Lippenframework hinzugefügt. Diese Vorgehensweise gewährleistet, dass der Facial Tracker nur aktiviert wurde, wenn alle erforderlichen Komponenten erfolgreich initialisiert wurden. Es ist zu beachten, dass das Starten der SRanipal-Software vor der Verwendung des Facial Trackers unerlässlich ist. Falls die Software nicht zuvor gestartet wurde und ein Facial Tracker erkannt wird, friert SEE ein. Es gibt dann solange keine Rückmeldung, bis die SRAnipal-Software gestartet wurde. Sobald diese erfolgreich gestartet wurde, setzt das Spiel seine Ausführung fort. Um den Benutzer darüber zu informieren, werden entsprechende Warnungen im Log angezeigt. Schließlich wurde die Netzwerkkomponente implementiert, um die erkannten Gesichtsbewegungen und Ausdrücke des Facial Trackers an alle Clients zu übertragen. Dies ermöglichte eine konsistente Darstellung der Mimik und Ausdrücke in Multiplayer-Spielen, bei denen mehrere Spieler gleichzeitig involviert sind.

Insgesamt war die Implementierung des HTC Facial Trackers eine anspruchsvolle Aufgabe. Durch die Erstellung von Fake-Blendshapes, die Anpassung auf den *Expression-Player*, die Integration von SRanipal und die Einbindung in den AvatarAdapter wurde eine effektive Lippenverfolgung und -erkennung ermöglicht. Die Übertragung der Daten an alle Clienten gewährleistete eine einheitliche Darstellung der Mimik in Multiplayer-Szenarien.

4.11.4 Fazit

William Behnke

Schlussendlich können wir festhalten, dass der HTC Facial Tracker trotz Hindernisse und Probleme erfolgreich in SEE integriert wurde. Um SEE weiter zu verbessern, schlagen wir trotzdem vor, auf ein anderes Avatar-System umzusteigen, das über eine größere Auswahl an relevanten Blendshapes verfügt. Dieser Vorschlag wurde bereits in Abschnitt 4.13 teilweise realisiert. Durch diesen Wechsel werden wir in der Lage sein, das volle Potenzial der Gesichtsanimationen optimal auszuschöpfen. Mit dieser Umstellung können wir sicherstellen, dass alle gewünschten Blendshapes präzise erfasst und angewendet werden.

Es wäre auch wünschenswert, in Zukunft eine umfassende Gesichtserfassung durch die Integration von Augentracking zu ermöglichen. Dafür fehlt uns aktuell die Hardware.

[6] SRanipal Software: https://developer.vive.com/resources/downloads/.

Ebenfalls erwähnenswert ist, dass nach der Implementierung des Facial Trackers die Integration auf großes Interesse stieß und bei verschiedenen Präsentationen und Veranstaltungen für Aufsehen sorgte.

4.12 AutoHand

William Behnke; Yvo Muskulus

AutoHand ist eine Lösung, die speziell für die Entwicklung von VR-Anwendungen entwickelt wurde. Es bietet ein umfassendes Set an Werkzeugen und Funktionen, um eine realistische Physik-Interaktion in der virtuellen Umgebung zu ermöglichen. Durch die Integration von AutoHand in SEE wird den Benutzern die Möglichkeit gegeben, mit den vorhandenen Nodes auf natürliche Weise zu interagieren [Rob23].

In der VR-Umgebung von SEE war es vor der Implementierung nicht möglich, mit den einzelnen Nodes zu interagieren. Es ist zwar möglich, sich durch den Raum zu bewegen und mit Hilfe eines Laserpointers auf Dinge zu zeigen, doch eine direkte Interaktion ist noch nicht implementiert.

Ziel ist es, durch AutoHand eine möglichst natürliche und intuitive Interaktion mit den einzelnen Nodes und Objekten zu bieten. Dadurch soll die Benutzererfahrung in der VR Umgebung von SEE erweitert und verbessert werden. Dieses Feature wurde von William Behnke und Yvo Muskulus implementiert.

Abbildung 47: Greifen einer Node in VR.

4.12.1 Umsetzung

Recherche: **William Behnke; Yvo Muskulus**

Zu Beginn haben wir uns mit der Demo-Szene von AutoHand vertraut gemacht und

verschiedene Funktionen getestet, wie zum Beispiel das Greifen von Objekten. Zudem wurde das integrierte Teleport-System ausprobiert. Da das Teleport-System Out-of-the-Box funktioniert, werden wir in der Umsetzung nicht genauer darauf eingehen. Des weiteren wurde sich in die Dokumentation von AutoHand eingelesen, um einen Einblick zu gewinnen, wie AutoHand in SEE integriert werden kann [Not23].

Greifbare Nodes: **Yvo Muskulus**

Um Nodes greifbar zu machen und mit der Unity Physic-Engine bewegen zu können, muss jede Node eine Grabbable- und Rigidbody-Komponente besitzen. Dies wurde durch die Methoden `SetupRigidbody()` und `SetupGrabbable()` in *InteractionDecorator.cs* realisiert. Dadurch werden die nötigen Komponenten bei der Generierung der Code-City den Nodes hinzugefügt. Nach ersten Tests ist uns aufgefallen, dass durch das Greifen der Nodes ungewollte Effekte mit anderen oder nahestehenden Nodes ausgelöst wurden. Zum Beispiel wurden die Nodes von der Schwerkraft beeinflusst, oder sind bei Kollision durch den Raum geflogen. Dieses Problem konnte behoben werden, indem die Schwerkraft für jede Node innerhalb der Grabbable Komponente deaktiviert wurde. Des weiteren gibt es die Option *isKinematic* in der Rigidbody-Komponente, wodurch Objekte nicht mehr von der Physik beeinflusst werden. Wir haben uns dafür entschieden, diese Option initial für jede Node zu aktivieren. Unsere Logik dahinter ist es, dass nur das gegriffene Objekt unter dem Einfluss der Physic-Engine steht, um ungewollte Nebeneffekte mit anderen Objekten zu eliminieren. Zusätzlich haben wir sichergestellt, dass das Objekt an seinen Ursprungsort zurückspringt, sollte es in der Luft platziert worden sein.

Nodes bewegen und loslassen: **Yvo Muskulus**

Nachdem die Nodes nun greifbar sind und erste Probleme behoben wurden, haben wir uns damit beschäftigt, die Nodes intuitiv bewegen und loslassen zu können. Hierfür haben wir die Klasse *VrGrabAction.cs* angelegt, welche die Logik dahinter implementieren soll. Dazu bietet die Grabbable Komponente zwei Events: `OnGrab` und `OnRelease`. Mit diesen Events können wir kontrollieren, was passieren soll, wenn die Nodes gegriffen und losgelassen werden. Wie im vorherigen Abschnitt erwähnt, haben wir die Option *isKinematic* initial für jede Node deaktiviert. Diese Option wird nun durch das `OnGrab` Event für die gegriffene Node aktiviert, sodass das Objekt mit Hilfe der Physic-Engine bewegt werden kann. Wird die Node losgelassen, wird dementsprechend durch das `OnRelease` Event diese Option wieder deaktiviert. Somit ist es uns nun möglich, Nodes zu greifen, sie frei durch den Raum zu bewegen und wieder loszulassen. Analog zu der *MoveAction* in der Desktop-Umgebung, soll es möglich sein, intuitiv Nodes auf anderen Nodes zu platzieren und zu verbinden. Da wir ohnehin schon mit der Unity-Physic arbeiten, war die Lösung naheliegend, diese Funktion mit Hilfe von *Collision* zu realisieren. Die Überlegung war es, dass bei Kollision der gegriffenen Node mit einer anderen Node der *Reparenting Prozess* gestartet wird. Umgekehrt wird der *UnReparenting Prozess*

gestartet, sollte eine Node nicht mehr mit einer anderen Node kollidieren.

Collision und Collision Detection: **Yvo Muskulus**

Zunächst haben wir die Kollisionserkennung der Rigidbodys angepasst. Aus der Dokumentation zur Kollisionserkennung geht hervor, dass die besten Ergebnisse erzielt werden, wenn die bewegenden Objekte auf *CollisionDetectionMode.ContinuousDynamic* gesetzt werden [Tec23b]. Für andere Objekte, mit denen diese kollidieren müssen, wird der Wert auf *CollisionDetectionMode.Continuous* gesetzt. In unserem Fall besitzt jeder Rigidbody einer Node initial den Wert *CollisionDetectionMode.Continuous*. Für die gegriffene Node wird der Wert dann in `OnGrab` auf *CollisionDetectionMode.ContinuousDynamic* und nach dem Loslassen in `OnRelease` wieder auf *CollisionDetectionMode.Continuous* gesetzt. Unity bietet zwei Events: `OnCollisionEnter` und `OnCollisionExit`. `OnCollisionEnter` wird aufgerufen, wenn ein Rigidbody begonnen hat, einen anderen Rigidbody zu berühren. `OnCollisionExit` wird aufgerufen, wenn ein Rigidbody einen anderen Rigidbody nicht mehr berührt. Bei der Implementierung dieser Events sind zwei Probleme aufgefallen. Zum einen wurde eine ungewollte Kollision ausgelöst, sollte wie in Abb. 48 die orange hervorgehobene Node (Main) nach oben bewegt werden. Dadurch entsteht beispielsweise Kollision mit allen Nodes, die direkt auf der Main-Node liegen. Dieses Problem konnte durch *CollisionLayer* behoben werden[Tec23a]. Dazu wurde in *GameObjectExtensions.cs* eine neue Methode `SetAllChildLayer()` implementiert, welche die Layer aller Kinder und Kindeskinder eines GameObjects setzt. Somit werden alle Kinder-Objekte auf einen Layer gesetzt, der keine Kollision mit dem Layer der gegriffenen Node auslöst. Zusätzlich wurde die *Layer Collision Matrix* (Abb. 49) in den Project-Settings so angepasst, dass nur noch Kollision zwischen zwei Grabbable Komponenten stattfinden kann.

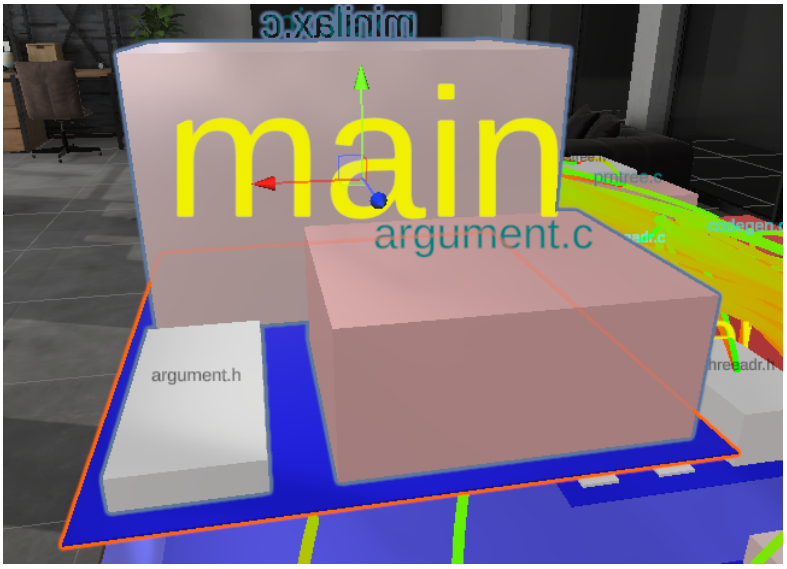


Abbildung 48: Ungewollte Kollision mit den darüber liegenden Objekten.

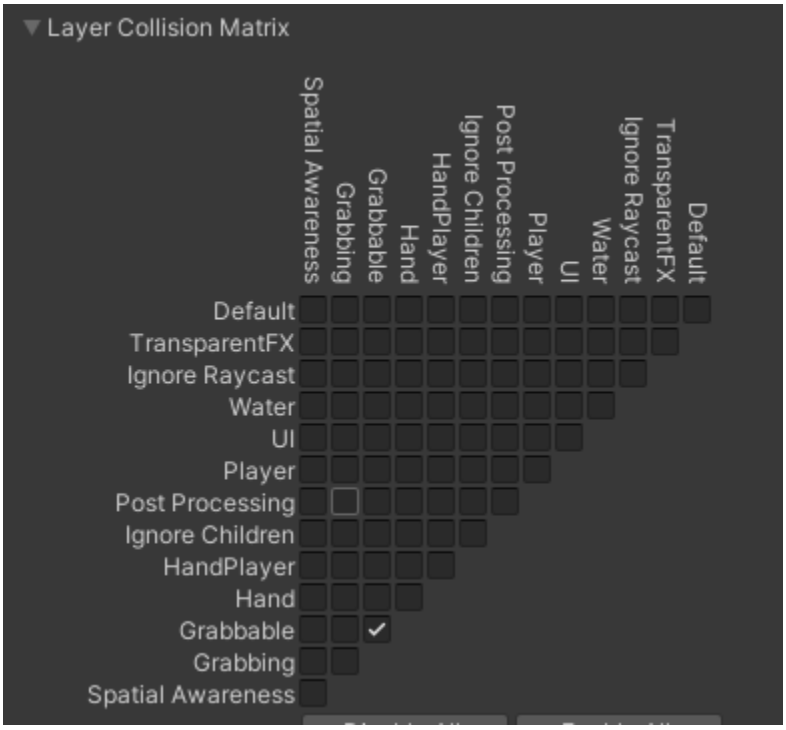


Abbildung 49: Layer Collision Matrix in den Project Settings.

Das zweite Problem bestand darin, wenn ein schnell bewegendes Objekt mit einem sehr dünnen Objekt kollidiert, es sein kann, dass Unity das nicht richtig verbucht. Das hat zur Folge, dass das Objekt durch das dünne „fliegt“ und ggf. noch mehr Kollision mit den darunter liegenden Objekten auslöst. Da dies ein direktes Problem der Unity Physic-Engine ist und wir dies nicht lösen können, haben wir uns dafür entschieden, die einzelnen Nodes dicker zu generieren. Das verringert die Wahrscheinlich, dass dieser Effekt auftritt.

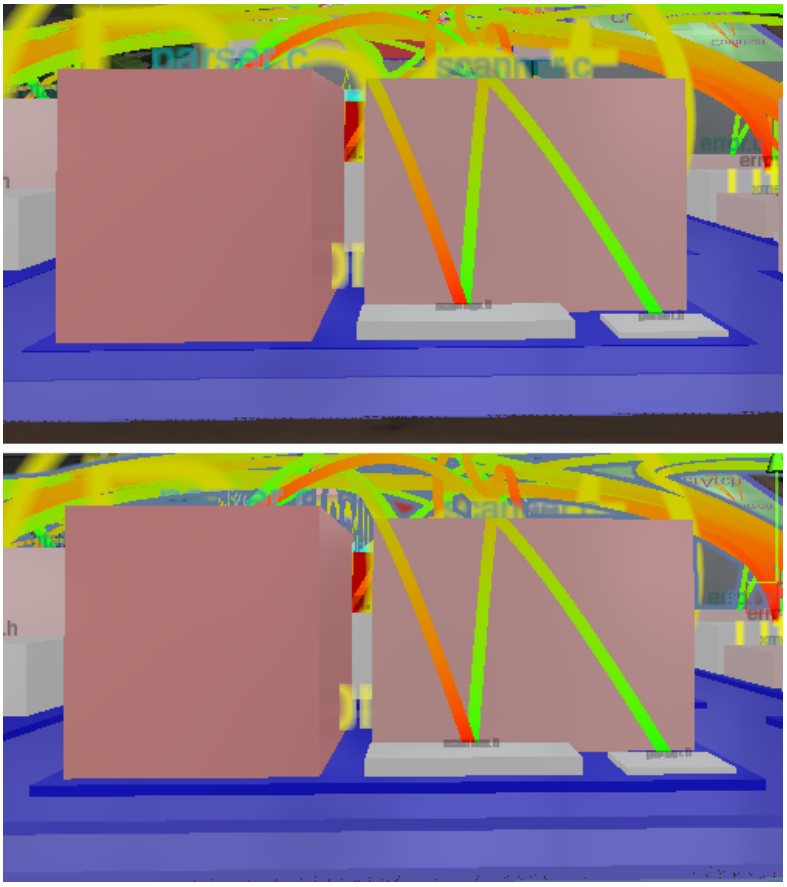


Abbildung 50: Oben die alte Node-Dicke. Unten die neue.

Zusätzlich haben wir Anpassungen an den Physik-Einstellungen vorgenommen, sodass die generelle Kollisionserkennung (*Default Contact Offset*) und die Berechnung dieser (*Default Solver Iterations*) genauer verläuft. Des weiteren wurde durch die Benutzung eines *No Bounce Materials* sichergestellt, dass zwei Objekte bei Kollision nicht voneinander abprallen.

Einbindung in die bestehende MoveAction: **William Behnke**

Die Einbindung in die bestehende MoveAction war ein weiterer interessanter Schritt, welcher auch interessante Hindernisse mit sich brachte. Hauptziel war hier das Reparenting, UnReparenting und der Aktualisierung der Hierarchie. Es gab Schwierigkeiten mit der Hierarchie, welche im Bereich Abschnitt 4.12.1 beschrieben werden. Ein weiteres Problem war die Funktion `putOnAndFit`, bei der einige Objekte immer kleiner wurden, wenn sie neu gegriffen wurden, jedoch wurde das mit der späteren Entfernung der Unparent-Methode irrelevant. Außerdem sind wir auf das Offset-Problem gestoßen,

das im Abschnitt „Offset-Problem“(Abschnitt 4.10.2) genauer erläutert wird.

Reparenting und UnReparenting: **William Behnke**

Die Methode `OnCollisionEnter` wird aufgerufen, wenn eine Kollision mit einem anderen Objekt auftritt. Zusätzlich wird vor dem Start des Reparenting Prozesses überprüft, ob alle von uns gewünschten Voraussetzungen abgedeckt sind. Je nach Ergebnis wird entweder der Reparenting-Prozess gestartet oder die Anfangsposition, -rotation und -skalierung des Objekts aktualisiert. Die MoveAction-Klasse enthält eine Methode namens `Reparent`, die das Reparenting eines Objekts auf einen anderen Node durchführt. Das Objekt wird visuell auf der Ziel-Node platziert und als Ziel markiert. Das ursprüngliche Parent-Objekt wird semantisch zur Ziel-Node umgeändert. Die Methode überprüft auch, ob das Ziel ein Nachkomme des ursprünglichen Parent-Objekts ist, um eine unendliche Schleife zu vermeiden. Je nach Eingabe erfolgt das Reparenting entweder mit Hilfe der `ReflexionMapperSetParent`-Methode oder der `GameNodeMoverSetParent`-Methode. Die `UnReparent` Methode macht das Reparenting rückgängig, indem sie das Objekt zurück zu seinem ursprünglichen Parent-Node verschiebt und das ursprüngliche Erscheinungsbild wiederherstellt. Diese wird aber im aktuellen VR- Kontext nicht benutzt und wird von uns als eine mögliche Fehlerquelle angesehen, welche weitere Nachforschungen mit sich ziehen wird. Zusätzlich mussten wir MoveAction ein wenig entsprechend anpassen, um die spezifischen Anforderungen einer VR-Umgebung zu berücksichtigen.

Das Hierachie Problem: **William Behnke**

Obwohl wir die Größe der Knoten geändert haben, besteht das grundlegende Problem darin, dass das neu verbundene Objekt in der Hierarchie wieder an seine ursprüngliche Position zurückkehrt, nachdem ein neues Objekt gegriffen wurde, das zuvor in Bezug zu diesem Objekt stand. Als Folge davon ändert sich die Größe auch wieder auf ihre ursprüngliche Größe zurück. Im Zuge des Reparenting-Prozesses wurden verschiedene Komponenten angesprochen, und das Problem konnte erst behoben werden, nachdem wir den Reparenting-Prozess vollständig analysiert und jeden einzelnen Schritt in einer kontrollierten Umgebung überwacht hatten und auf nötigste reduziert hatten. So fiel beispielsweise der Unparenting-Prozess komplett raus, da er in VR aktuell nicht benutzt werden muss.

4.12.2 Fazit **William Behnke**

Die Demo beinhaltete die Erweiterung der Funktionalität durch die Implementierung von DistanceGrab und Raycast. Diese Features wurde jedoch bei uns nicht erfolgreich umgesetzt. Hierbei könnte man ansetzen und überlegen, wie man diese Funktionen integrieren könnte. Dies könnte den Benutzern ebenfalls ermöglichen, Objekte auf verschiedene Arten und mit größerer Präzision zu greifen und zu manipulieren. Außerdem haben wir die genaue Fehlerquelle vom Hierachie-Problem nicht identifizieren können,

aber würden gerne hier erwähnen, dass weitere Untersuchungen vom UnReparenting Prozess bei zukünftigen Arbeiten höchstwahrscheinlich helfen könnten.

Zusammenfassend lässt sich sagen, dass dieses Feature Potenzial zur Verbesserung bietet. Die Implementierung von DistanceGrab und oder das Bewegen von Nodes per Raycast könnte die Interaktionsmöglichkeiten erweitern. Es ist ebenfalls wichtig zu erwähnen, dass die Implementierung zum Zeitpunkt dieses Berichts noch nicht komplett abgeschlossen ist. Es fehlt noch die Netzwerkkomponente. Die ActionHistory und die Actionstates müssen ebenfalls noch für VR implementiert werden.

4.13 FACSvatar

Yvo Muskulus

Kommunikation und Mimik ist ein essentieller Bestandteil der kollaborativen Arbeit in SEE. Durch den Facial Tracker haben wir eine Möglichkeit, die Lippen- und Mundbewegungen in SEE auf dem Avatar abzubilden. Das kann durch das sogenannte Facial Action Coding System (FACS) erweitert werden [Pau20]. FACS ist ein Kodierungssystem zur Beschreibung von Gesichtsausdrücken. Dabei werden die Gesichtsausdrücke in einzelne Komponenten von Muskelbewegungen, sogenannten Action Units, zerlegt. Diese Action Units können mit der Hilfe von FACSvatar auf den Charakter übertragen werden [Num23b]. FACSvatar ist ein Open-Source-Projekt auf Github, welches eine Implementierung von FACS in Unity enthält. Es ist ein System, mit dem animierte 3D-Gesichter auf Echtzeit-Videoeingaben reagieren können.

Der HTC Facial Tracker bietet die Möglichkeit, die Mund- und Lippenbewegungen auf den Avatar zu übertragen. Es gibt aber noch keine Möglichkeit, die obere Hälfte des Gesichts darzustellen. Des weiteren bieten die aktuell genutzten UMA-Charaktere nicht die Voraussetzungen, um die Action Units repräsentieren zu können.

Ziel ist es, einen männlichen und weiblichen Avatar zu erstellen, der mit dem Action Coding System kompatibel ist. Darüber hinaus müssen die Avatare Blendshapes bieten, die von dem HTC Facial Tracker angesprochen werden können. Final sollen diese Avatare in die SEE-Umgebung importiert werden. Dieses Feature wurde von Yvo Muskulus entwickelt.

4.13.1 Umsetzung

Erstellung der Avatare:

Aus der Dokumentation von FACSvatar geht hervor, dass mit dem Tool MB-Lab Avatare generiert werden können, die direkt mit dem FACSvatar System kompatibel sind [Num23a]. MB-Lab ist ein Open-Source-Add-On für Blender, um humanoide Charaktere zu generieren [MB-21]. Das Tool wurde ursprünglich von Manuel Bastioni entwickelt und wird nun als Community Projekt am Leben erhalten. Auf dem folgenden Bild sieht man das Tool in Aktion, mit einem generierten weiblichen Avatar.

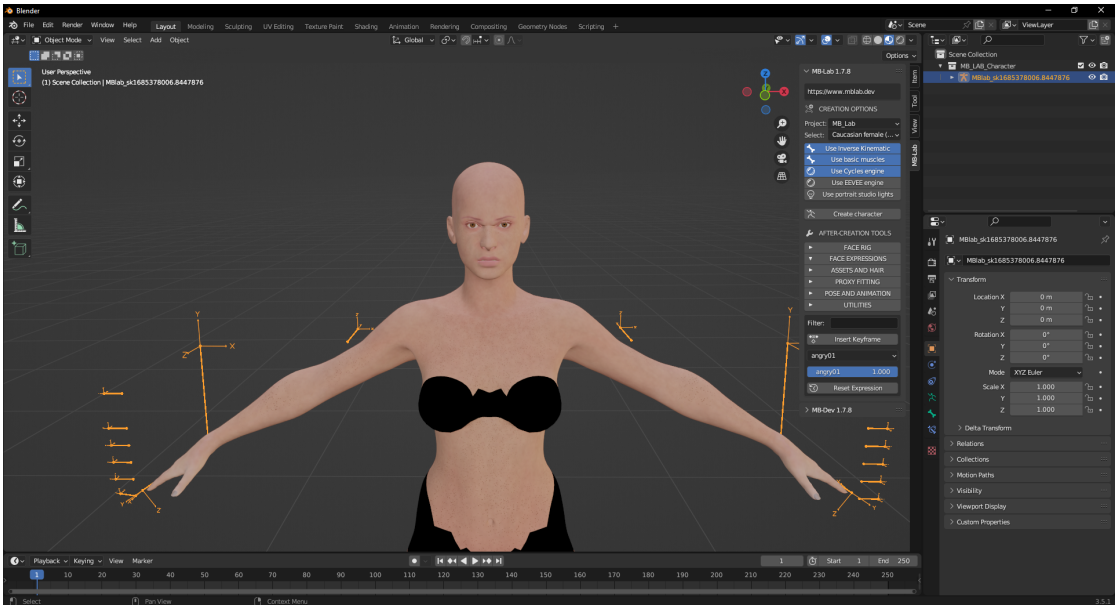


Abbildung 51: Charakter Erstellung in Blender mit dem MB-Lab Tool.

Mit einem Klick auf *Create character* (siehe Abb. 51) können so leicht weitere Avatare erstellt werden. Außerdem bietet das Tool Optionen, die Charakterproportionen nach seinem Belieben anzupassen. Über den Tab *ASSETS AND HAIR* können zusätzliche Kleidungs- und Haar-Assets importiert werden, die dem zuvor erstellten Charakter hinzugefügt werden können. Dabei ist das *PROXY FITTING* eine große Hilfe, da es die importierten Assets direkt dem erstellen Avatar anpasst und einem sehr viel Arbeit erspart, jenes manuell anzupassen. In Abb. 52 und Abb. 53 wird veranschaulicht, wie eine Jacke hinzugefügt und an den Avatar angepasst wurde.

Abbildung 52: Hinzufügen eines Kleidungsstückes.

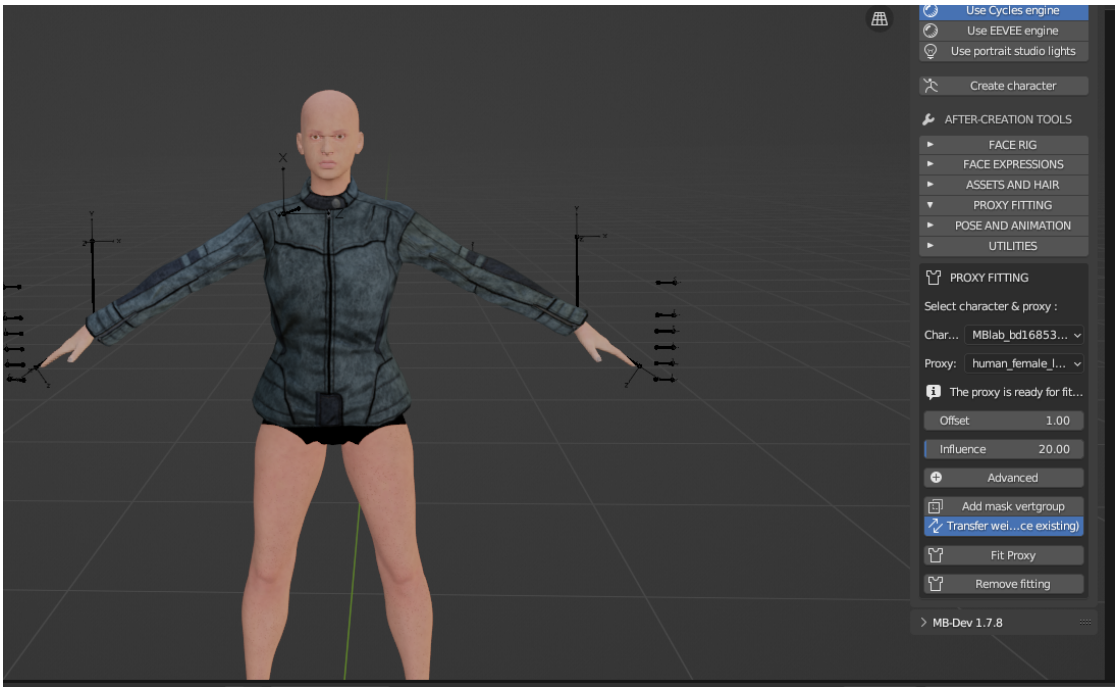


Abbildung 53: Anpassung mit dem Proxy Fitting Tool.

Hinzufügen der Blendhapes für den HTC Facial Tracker:

Nachdem der Charakter erfolgreich erstellt und mit Kleidung und Haaren ausgestattet wurde, müssen letztlich noch die Voraussetzungen geschaffen werden, damit dieser mit dem HTC Facial Tracker kompatibel ist. Der HTC Facial Tracker benötigt eine Reihe an Blendshapes (siehe Abschnitt 4.11.1), um zu funktionieren. Wichtig sind hierbei die korrekten Namen der Blendshapes, damit diese überhaupt angesprochen werden können. Wie zu Beginn der Umsetzung erwähnt, wird der Avatar in MB-Lab direkt mit Blendshapes generiert, welche mit dem FACSvatar System kompatibel sind. Diese Blendshapes können unter anderem genutzt werden, um die Blendshapes für den Facial Tracker zu erstellen.

Blender bietet eine Funktion namens *New Shape from Mix*, mit der ein neuer Blendshape hinzugefügt werden kann. Der neue Blendshape setzt sich aus einer Mischung aller vorhandenen Blendshapes und ihrer Werte zusammen.

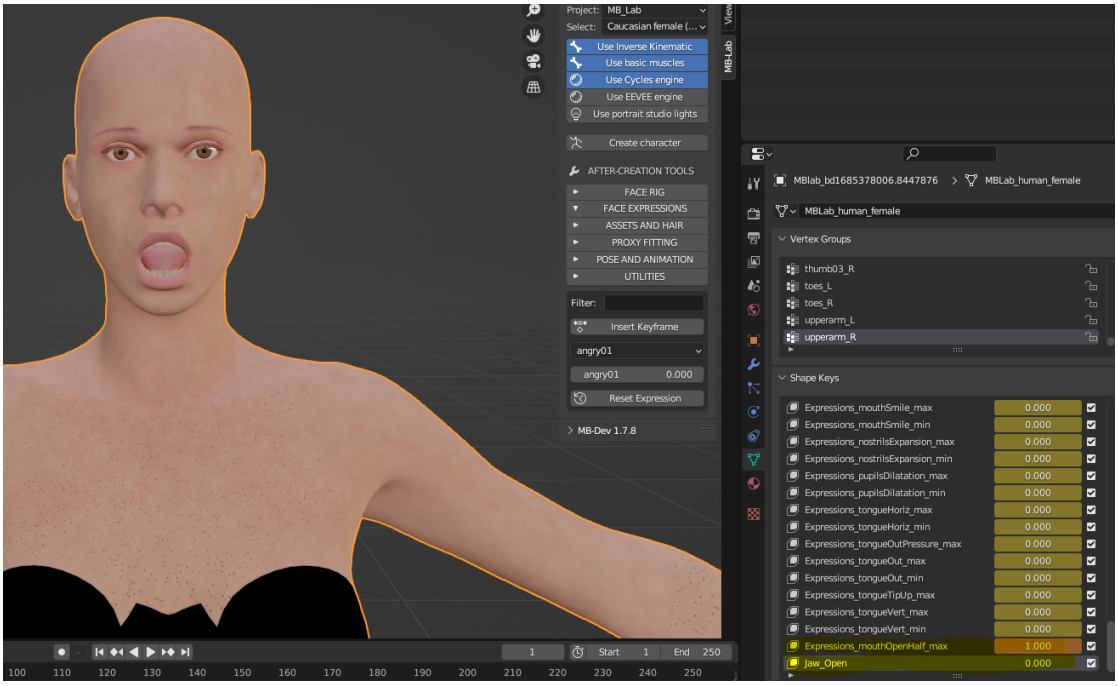


Abbildung 54: Erstellung neuer Blendshapes.

In Abb. 54 ist zu erkennen, dass der Blendshape „Jaw_Open“ aus der Mischung von „Expressions_mouthOpenHalf_max“ erstellt wurde. „Jaw_Open“ kann somit die gleiche Deformation bewirken. Im Endeffekt wurden alle Blendshapes für den Facial Tracker

aus einer Mischung der vorhandenen Blendshapes erstellt.

Export aus Blender und Import in Unity:
Blender bietet die Möglichkeit, den Avatar oder das Projekt als *.fbx* Datei zu exportieren, welche direkt in Unity importiert werden kann. Hierzu wurde unter *„Assets/Resources/Materials/ “* der Ordner *FACSvatar* angelegt, der alle relevanten Dateien zu den FACSvataren samt benutzen Materialien enthält.

Es ist wichtig, dass in den Import Settings der *.fbx* Datei, im Tab Rig, der Animation Type auf Humanoid gesetzt ist. Des weiteren müssen im Materials Tab die richtigen Materials zugewiesen werden. Diese *.fbx* Dateien wurden anschließend in Prefabs umgewandelt und sind unter *„Assets/Resources/Prefabs/Players/“* zu finden.

Damit der Avatar mit SEE funktionieren kann, müssen dem Prefab noch Komponenten hinzugefügt werden. Diese wurden aus den bestehenden UMA-Avataren kopiert und dem Root-Objekt des Prefabs hinzugefügt. Lediglich der *Character Controller* und *Capsule Collider* mussten an die neue Größe der Avatare angepasst werden.

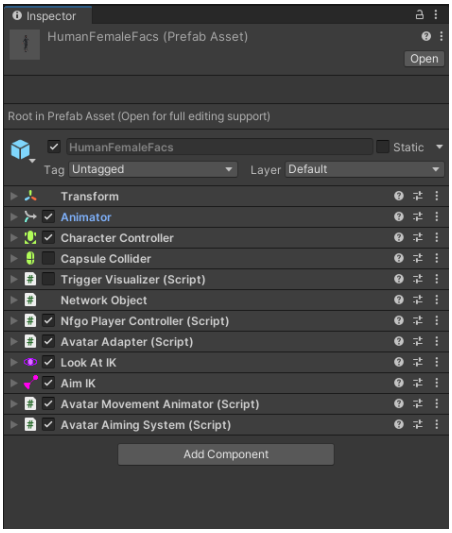


Abbildung 55: Komponenten des weibliches FACS Avatars.

Abschließend wurden die Prefabs im *NetworkManager* (SEEStart Scene) und *PlayerSpawn* (SEEReflexion Scene) hinterlegt und können nun aktiv in der Desktop- und Virtual Reality Umgebung verwendet werden.

4.13.2 Fazit

Im direkten Vergleich der beiden Avatare (Abb. 56) fällt auf, dass der neue FACSvatar deutlich detailreicher und natürlicher wirkt. Durch den höheren Detailgrad kann die Mimik wesentlich besser dargestellt und vom Benutzer erkannt werden. Somit war die Umsetzung erfolgreich.

Mangels Erfahrung in Blender kann der Avatar deutlich schöner gestaltet werden. Gerade die Modellierung der Blendshapes für den Facial Tracker könnte mit etwas Grunderfahrung besser und genauer umgesetzt werden. Außerdem kann durch den Erwerb von zusätzlichen Kleidungs- und Haarassets die Avatare realistischer gestaltet werden.

Abbildung 56: Links der alte UMA Avatar. Rechts der neue FACSvatar.

4.14 LiveDocumentation

Hannes Kuß; Alexander Kaiser

Dieses Feature wurde von Hannes Kuss und Alexander Kaiser bearbeitet

4.14.1 Einleitung

Alexander Kaiser

Das Projekt SEE bietet bereits viele nützliche Tools für die Analyse und dessen Visualisierung von Software. Durch die Darstellung der Software als Codecity ist die Architektur zu jedem Zeitpunkt sehr übersichtlich. Unter anderem kann SEE bereits den Code der Klassen anzeigen. Doch das war uns nicht genug, denn wir wollten, ähnlich wie in einigen Entwicklungsumgebungen, dass Klassennamen und Funktionsaufrufe klickbar sind und auf die jeweiligen Instanzen verweisen, die sich in einem neuen Tab öffnen. Ebenfalls sollen diese Verlinkungen visuell unterscheidbar vom Rest der Dokumentation sein.

Das Ziel war neben den bestehenden Schlüsselwörtern für Klassen- bzw. Methodendokumentationen eigene Wörter einzubinden, um die Visualisierung in SEE anzureichern. Als Beispiel ist das Einbetten von Bildern zu nennen. So könnten Bilddateien, die bestimmte Architekturentscheidungen darstellen, aus einem 'assets'-Ordner im Code verlinkt und in SEE angezeigt werden. Doch auch ganze Use-Cases könnten dokumentiert werden. So könnte man einen dokumentierten Anwendungsfall per Klick starten. Die einzelnen Komponentenaufrufe innerhalb des Use-Cases könnten durch Kanten visualisiert werden. Kanten könnten Beschreibungen beinhalten, um Entwicklern die Aufrufe zu erklären. Innerhalb des Bachelorprojekts, haben wir die Funktion der Verlinkungen umsetzen können und ebenfalls die Grundlage für die Erstellung neuer Schlüsselwörter geschaffen. Für mehr Funktionen hat die Zeit leider nicht ausgereicht.

4.14.2 Motivation

Alexander Kaiser

Viele Entwickler und Unternehmen haben das Problem, dass die Einarbeitung in eine bestehende Software oftmals lange dauert und unverständlich ist. Dies führt zu langen und komplexen Onboardingprozessen, je nach Größe der bestehenden Codebase. Das zugrunde liegende Problem ist die Art des Wissenstransfers. Bisher wird das Wissen durch Erklärungen von Kollegen weitergereicht. Doch auch das Zeigen von Use-Cases trägt zum Wissenstransfer bei. Oftmals werden die Architektur, Komponentenzusammenhänge, Zuständigkeiten und Funktionsumfänge nicht richtig nachvollzogen. Dies liegt daran, dass Onboardings meist sehr theoretisch sind und erst bei der Implementierung von Tickets nach und nach wirkliches Wissen entsteht, da man zu diesem Zeitpunkt erste Kontaktpunkte mit der Software hat. Doch was wäre, wenn man diese Unverständlichkeiten leichter visualisieren könnte? Wenn die Architektur leicht ersichtlich und mehrere Dokumentationen einfach abrufbar bereitstünden? Was, wenn

Funktionsaufrufe der Komponenten bei bestimmten Use-Cases sichtbar werden? Auf all diese Fragen versucht das Feature der Live-Documentation eine Antwort zu finden, um Onboardingprozesse zu verbessern.

4.14.3 Entwurf UI

Hannes Kaiser

Das LiveDocumentation Feature wurde zu Beginn so gedacht, dass sowohl die Dokumentation, als auch die Methodensignaturen dargestellt werden können. Dazu wurde sich zunächst ein grober Entwurf überlegt, der mit AffinityDesigner erstellt wurde und in Abb. 57 zu sehen ist.

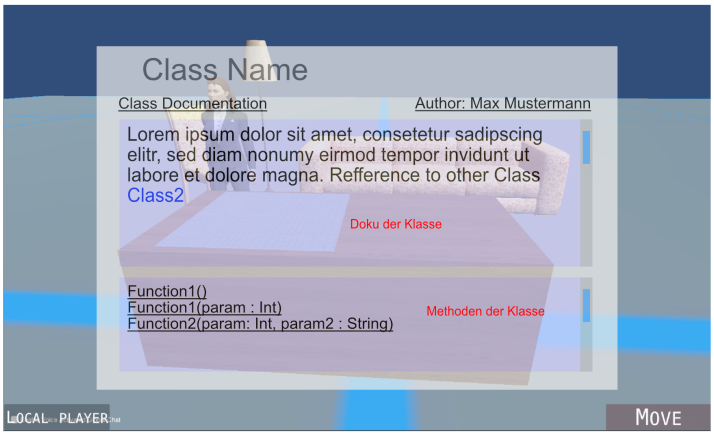


Abbildung 57: Erster Entwurf des LiveDocumentation Fensters

Wie im Bild zu sehen ist, sollte das LiveDocumentation-Fenster aus folgenden Komponenten bestehen:

- Einem Text „Live Documentation“
- Dem Namen der Klasse
- Dem Autor der Klasse
- Der Dokumentation der Klasse - mit Links zu anderen Klassen
- Einer Liste mit den Methoden(signaturen) der Klasse

Das Fenster

Das Fenster soll sich wie in Abb. 57 zu sehen ist, als Overlay zur Scene öffnen, wenn auf einen Knoten in der CodeCity geklickt wurde. In Abb. 57 nicht zu sehen, aber dennoch angedacht war es, das Fenster mit dem „Code Viewer“ von Falko zu verbinden.

Dieser Code Viewer nutzt ein Unity Asset „Dynamic Panel Canvas“, welches ein UI System mit Tabs bereitstellt. Es schien aus Sicht des Entwurfes zu diesem Zeitpunkt sinnlos, ein neues Tab-System zu erstellen, da dies Redundanz bedeuten würde, sowie die gleichzeitige Interaktion mit dem Code Viewer und der LiveDocumentation umständlich machen würde. Daher wurde sich bereits früh dazu entschieden, die LiveDocumentation in der Tab-System von Falkos Code Viewer zu integrieren.

Es soll somit möglich sein, in einem Fenster sowohl Tabs von Code Viewer, als auch LiveDocumentation Tabs zu öffnen.

Klassen Documentation

In dem oberen Teilfenster soll die Dokumentation der Klasse stehen, mit evtl. vorhandenen Referenzen zu anderen Klassen. In C# wäre das z.B. der Inhalt des „<summary>“ tags der C# Dokumentation. Referenzen sollen dabei, wie zu sehen ist, in blauer Farbe dargestellt werden.

Methoden Liste

Es war zunächst die Idee, zum Darstellen der Parameter einen an UML-Diagramme angelehnten Stil zu wählen ([Name des Parameters] : [Typ des Parameters]). Dies sollte einen einheitlichen Stil über alle Sprachen hinweg sicherstellen. Der Ansatz wurde aber später wieder verworfen (siehe Abschnitt 4.14.4).

Es soll jedoch nicht nur die Dokumentation der Klasse angezeigt werden können:
Da Methoden ebenfalls eine Dokumentation besitzen, sollte diese ebenfalls in der LiveDocumentation dargestellt werden können. In diesem Fall wird die Beschreibung der Methode ebenfalls in dem Feld der Dokumentation dargestellt. Die einzelnen Parameter einer Methode können ebenfalls Dokumentation besitzen. Daher sollen die Parameter, mitsamt ihrer Dokumentation in dem unteren Fenster „Methoden der Klassen“ aufgelistet werden.

4.14.4 Implementierung UI

Hannes Kuß

Wie bereits in Abschnitt 4.14.3 erwähnt, soll die LiveDocumentation analog zum Code Viewer im selben Tab-System laufen. Dabei wurde sich mit Falko zu diesem Thema abgesprochen und es wurde eine neue Architektur entwickelt. Da zu derselben Zeit gerade das Paper von Hannes und William geschrieben wurde, wurde die Implementierung von Falko übernommen, auch da Falko einen besseren Überblick über das System des Code Viewers hat.
Die neue Architektur von Falko sieht nun vor, dass von der Klasse `BaseWindow` geerbt werden muss.

Daher wurde die Klasse `LiveDocumentationWindow` erstellt, die von `BaseWindow` erbt. Die Funktionalität (insbesondere die Instanziierung des Prefabs) ist dabei von der Code View Klasse `CodeWindow` inspiriert (insbesondere die Verwendung der Methode `PrefabInstantiator.InstantiatePrefab`) `LiveDocumentationWindow` erstellt dabei eine Instanz von dem Prefab `LiveDocumentation`, welches in Abb. 58 zu sehen ist. Davor ist es allerdings notwendig, dass bestimmte Attribute der Klasse gesetzt werden. Diese sind:

- `SourceName` - Name der Node vom Typ string
- `RelativePath` - Der relative Pfad der Klasse im Projektordner vom Typ string
- `NodeOfClass` - Die Node der Klasse von Type Node

Wie später beschrieben, unterstützt die LiveDocumentation zwei Arten von Fenstern (spezifiziert im enum `LiveDocumentationWindowType`).

- `CLASS` - Zum Anzeigen der Dokumentation von Klassen
- `METHOD` - Zum Anzeigen der Dokumentation von Methoden

Diese beiden Modi unterscheiden sich allerdings in nur wenigen Punkten:
Im Methoden-Modus wird der Text „ClassMembers“ zu „Parameters“ geändert. In der Liste, in der zuvor die Methoden standen, werden dann die Parameter, mitsamt ihrer Dokumentation der Methode aufgelistet. Im Gegensatz zum Klassen-Modus aber können diese Elemente der Liste nicht geklickt werden. Wenn dafür im Klassen-Modus ein Methoden-Feld geklickt wird, öffnet sich ein neuer Tab (dann im Methoden-Modus) mit der Dokumentation der Methode. Das Prefab für die LiveDocumentation ist im folgenden Screenshot abgebildet, in Abb. 58

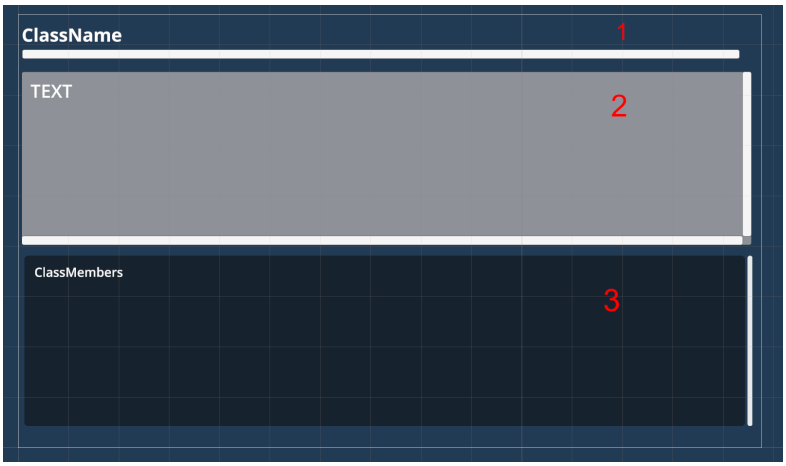


Abbildung 58: Screenshot aus dem Unity Editor des Prefab `LiveDocumentation`

Wie zu sehen ist, wurde das in Abb. 57 noch vorhandene Textfeld mit dem Autor in der Implementierung nicht umgesetzt. Dafür musste sich entschieden werden, da in einigen Sprachen, wie u.a. auch C# keine explizite Angabe des Autors in der Dokumentation spezifiziert ist, im Gegensatz zu Java.
Alle Felder sind in Unitys `ScrollView` eingebettet, damit die LiveDocumentation auch bei veränderter Fenstergröße benutzbar bleibt. In dem Textfeld mit der Nummer 1 soll der Name der Klasse angezeigt werden.
In Textfeld 1 ist nur die horizontale `ScrollBar` aktiv, da der Klassenname nur in seltenen Fällen die Größe des Fensters überragen würde und generell keine Zeilenumbrüche enthält. Somit wäre eine vertikale ScrollBar eher unnötig.
Textfeld 2 hingegen besitzt sowohl eine vertikale, als auch eine horizontale ScrollBar. In diesem Textfeld soll die eigentliche Dokumentation dargestellt werden.
Textfeld 3 besteht aus einer Liste von Textfeldern, in die die Methoden, mit einem Preview von deren Dokumentation geschrieben werden. Ein Beispiel hierfür ist in Abb. 60 zu finden.
Für alle Textfelder wurde dabei analog zum Code Viewer TextMeshPro von Unity verwendet. Um den Text optisch ansprechend zu formatieren, aber auch um Links darzustellen.

Um das Fenster mit den Daten der Dokumentation zu füllen, wurde ein Buffersystem gewählt: Die Dokumentation wird in diesem Buffer stückweise geschrieben.

Das Buffersystem

Dafür wurde die Klasse `LiveDocumentationBuffer` erstellt. Diese besitzt als Variable eine Liste von `ILiveDocumentationBufferItems` `ILiveDocumentationBufferItems` hingegen ist ein Interface, welches die Aufgabe haben soll, einen bestimmten Text (über die Methode `GetPrintableText()`) auszugeben. Dieses Interface besitzt aktuell zwei Implementierungen:

- `LiveDocumentationBufferText`
- `LiveDocumentationLink`

`LiveDocumentationBufferText` wird genutzt, um einfachen Text der Dokumentation auszugeben. `LiveDocumentationLink` hingegen spezifiziert einen Link in der Dokumentation. Dieser Link wird bei Aufruf von `GetPrintableText()` in das RTF-Format, welches TextMeshPro benötigt, umgewandelt. Dabei können der Name des Links sowie das Ziel bestimmt werden – i. d. R. sind diese aber identisch.

Es existiert noch eine weitere Klasse, die von `LiveDocumentationBuffer` erbt:
`LiveDocumentationClassMemberBuffer`. Diese Klasse ist dafür zuständig, die Methodenliste mit Inhalt zu füllen.

`LiveDocumentationClassMemberBuffer` ist ein `LiveDocumentationBuffer` an sich (durch die Ver-

erbung). In diesem Buffer wird die Signatur der Methode gespeichert (z.B. `public Node GetNode(int i)`), mitsamt Links zu anderen Klassen, wie Parameter oder Rückgabetypen. Die Klasse besitzt allerdings noch einen weiteren `LiveDocumentationBuffer` für die eigentliche Dokumentation der Methode sowie eine Liste mit weiteren `LiveDocumentationBuffer` für die Dokumentation der einzelnen Parameter der Methode. Ein Beispiel für den Methoden-Modus ist in Abb. 61 zu finden.

Eine Übersicht der Architektur des Buffersystems im UML Format ist in Abb. 59 zu sehen.

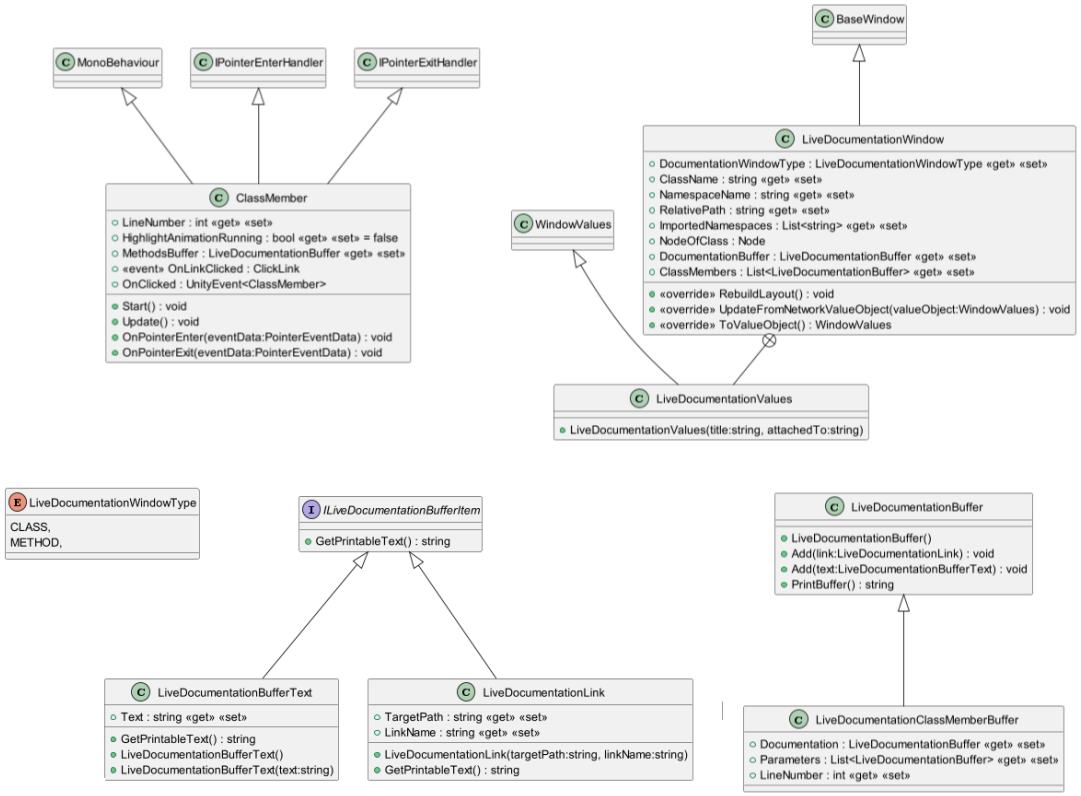


Abbildung 59: UML Klassendiagramm der LiveDocumentation

Der `LiveDocumentationBuffer` wird von dem Parsingsystem gefüllt. (siehe Abschnitt 4.14.5), ebenso wie die Methodenliste mittels des o.g. `LiveDocumentationClassMemberBuffer`. Das Fenster der LiveDocumentation sieht fertig wie folgt aus:

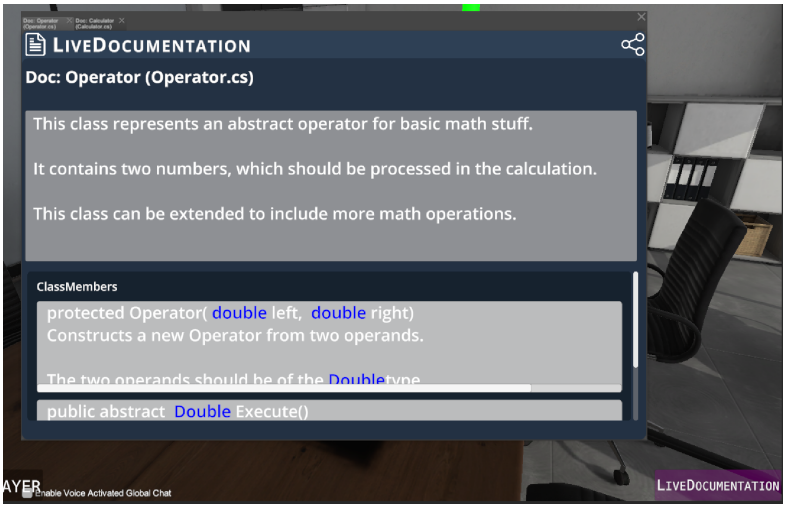


Abbildung 60: Screenshot aus Unity mit geöffneten LiveDocumentation Fenstern

Die Methoden haben bewusst nur eine horizontale ScrollBar, da diese sonst mit der ScrollBar der gesamten Liste an Methoden Probleme gemacht hätte.

Dokumentation von Methoden

Um die Dokumentation einer Methode zu öffnen, muss mit gedrückter Shift-Taste auf das Feld der Methode in der Liste geklickt werden. Shift ist notwendig, damit einfaches Scrollen in dem Feld nicht mit dem Befehl zum Öffnen der Methodendokumentation verwechselt werden kann.

Ein Beispiel von einem LiveDocumentation Fenster, das sich im Methoden-Modus befindet, ist in Abb. 61 zu finden.

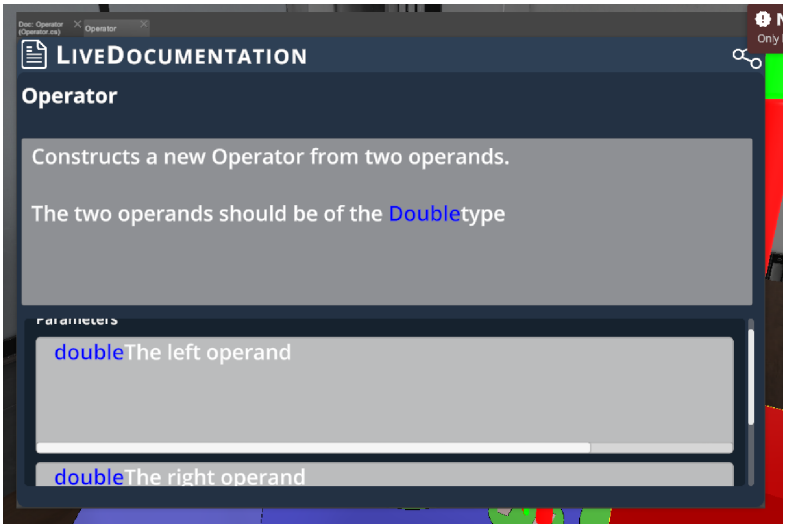


Abbildung 61: Screenshot aus Unity mit geöffneten LiveDocumentation Fenstern im Parameter Modus

4.14.5 Implementierung Parser

Alexander Kaiser; Hannes Kuß

Um einer Methoden- oder Klassendokumentation Information zu entnehmen und auf eine bestimmte Weise darzustellen, bedarf es einer textuellen Analyse. Um die Texte in logisch zusammengehörende Einheiten zu unterteilen, brauchen wir einen Parser, der unsere Wörter liest und an bestimmten Stellen erkennt, welcher Text zu welcher Einheit gehört. Damit wird diese Arbeit nicht selbst machen mussten, haben wir die Library 'ANTLR4' [Par13] verwendet, die bereits im Projekt verwendet wurde.

Daraufhin schrieben wir eine Grammatik, die zunächst Summaries erkannt und geparst hat. ANTLR hat dann anhand dieser Grammatik Lexer und Parser generiert.

Jedoch mussten die Signaturen der Methoden extrahiert werden. Dafür haben wir uns einem bestehenden Parser für C# bedient [ant], mit dessen Hilfe wir die Struktur der C# Methoden analysieren können. Somit können wir wichtige Werte, wie Namen und Typen auslesen und somit auch darstellen.

Im Folgenden konnten wir mithilfe des Parsers programmatisch die Summary einer Klasse bzw. einer Methode ermitteln. Doch dabei blieb es nicht, denn wir wollten im nächsten Schritt auch Verlinkungen erkennen und auf Klick eine bestimmte Aktion durchführen. Dafür haben wir unsere Grammatik angepasst und ANTLR die bestehenden Dateien überschreiben lassen. Nun konnten wir im Code erkennen, ob ein ausgeführter Klick auf einer Klassenverlinkung landete. War dies der Fall, konnten wir weitere Logik im Codewindow ausführen. Zu diesem Zeitpunkt gab es allerdings noch

eine Schwierigkeit: Woher wussten wir, welche Datei wir zu öffnen hatten, sobald ein Nutzer auf eine Verlinkung gedrückt hatte?

Dafür haben wir eine weitere Grammatik geschrieben, die Namespaces aus Dateien auslesen konnte. So konnten wir Verlinkungen zu Klassen und anderen namespaces richtig zuordnen. In Java zum Beispiel gäbe es hierbei keine Schwierigkeiten. Dort existieren keine namespaces und somit hat jeder Dateiname einen genauen Pfad bestehend aus Packages und Dateinamen (und innerer Klasse).

Auf Basis der ausgelesenen Namespaces können wir somit den Ort der verlinkten Klassen stark eingrenzen. Da natürlich alle möglichen Klassen des Projektes zu Scannen (auch diese, die in Form von Libraries eingebunden sind) den Rahmen des Projektes bei weitem übersteigen würde, haben wir uns dafür entschlossen, nur Klassen, die Teil des Graphen sind, zu scannen. Somit werden alle von der geöffneten C# Datei importierten Namespaces, die im Graph vorhanden sind, mittels eines Breitensuche-Algorithmus durchsucht, um die gesuchte Klasse zu finden. Der Dateiname lässt sich über den Graphen auslesen, in dieser Datei können sich aber mehere Namespaces bzw. Klassen befinden. Eine Klasse kann jedoch nicht mehrmals im selben Namespace und zur selben Zeit in derselben Datei - im Gegensatz zu einer „partial class“. Wenn somit in der Datei der richtige Namespace existiert können alle Klassen der Reihe nach der richtigen Klasse durchsucht werden. Insbesondere, da i.d.r der selbe Namespace nicht mehrfach in der selben Datei definiert werden sollte.

Der Parser wurde zudem über das Interface `IExtractor` generalisiert. Indem neue Implementierungen von diesem Interface erstellt werden, lassen sich somit weitere Parser für andere Sprachen hinzufügen. Es ist jedoch leider nicht möglich, „partial classes“ zu öffnen, da hier nicht eindeutig bestimmt ist, welche Datei bzw. Klasendefinition gemeint ist - auch, da die einzelnen Dateien theoretisch verschiedene Dokumentationen besitzen können.

4.14.6 Aufgetretene Probleme

Hannes Kuß

NullReferenceExceptions beim Synchronisieren der Fenster

Es gab ein Problem mit Falkos Implementierung des neuen UI-Systems: Wenn Änderungen an einem Fenster vorgenommen wurden, wurde eine `SyncWindowSpaceAction` gestartet, die den neuen Stand der Fenster an alle anderen Clients weitergeben soll. Bei der Serialisierung einiger Felder in der `SyncWindowSpaceAction` Klasse wurden allerdings einige Attribute zur korrekten Serialisierung nicht gesetzt. Außerdem ist die Klasse `WindowValues` (von der weitere Klassen erben, die den Inhalt der Fenster darstellen sollen) abstrakt gewesen. Abstrakte Klassen werden anscheinend nicht serialisiert.

Dies führte dazu, dass die Variablen von `SyncWindowSpaceAction` beim Empfänger zu `null`

wurden. Durch Entfernen des `abstract` Modifiers, sowie das Hinzufügen einiger Attribute konnte der Fehler aber behoben werden und wurde sogleich per Pull request#597 gemerged.

4.15 Port zu Linux und MacOS

Hannes Kuß

Der Port von SEE zu Linux und MacOS wurde anfänglich von Hannes Kuß und Schahin Schuch übernommen. Schahin Schuch hat aber das Projekt einige Wochen nach Beginn wieder verlassen und Hannes Kuss hat alleine an dem Feature weiter gearbeitet.

4.15.1 Problemanalyse

Bereits vor dem ersten Plenum des Bachelorprojektes wurde das Repository auf GitHub freigeschaltet und die Teilnehmenden wurden dazu angehalten, das Projekt schon einmal zu Clonen und lokal bei sich zu testen. Dabei fiel auf, dass sich das Projekt zwar unter macOS öffnen ließe, allerdings ein Build nicht möglich ist, wie im folgenden Screenshot Abb. 62 zu sehen ist.

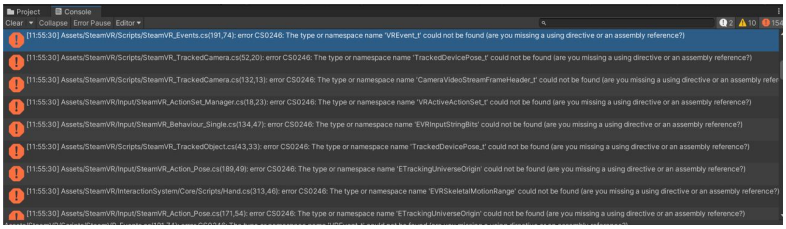


Abbildung 62: Screenshot von den Fehler im Buildprozess

Als dann die möglichen Themen in einer der anfänglichen Plena vorgestellt wurde, habe ich mich dazu entschlossen vorzuschlagen, als Aufgabe das Porting von SEE zu macOS bzw. Linux zu übernehmen. Insbesondere da andere Teilnehmende im Projekt darauf hingewiesen haben, dass das Problem ebenfalls auch unter Linux besteht. Wie bereits in Abb. 62 zu sehen ist, konnte der Build unter macOS nicht durchgeführt werden, weil bestimmte Symbole in der *SteamVR* Bibliothek nicht gefunden werden konnten. Weitere Recherchen haben gezeigt, dass *SteamVR* generell nicht unter macOS kompatibel sei und nur Windows unterstütze [az]. Rainer Koschke hat zudem erwähnt, dass SEE bald von *SteamVR* zu *OpenXR* wechseln würde. Weitere Recherchen hier zeigten aber, dass *OpenXR* ebenso weder macOS, noch Linux unterstützen würde [Uni]. Somit bleibt keine weitere Lösung, als die VR Funktionalität unter macOS und Linux auszuschließen.

Ein weiteres Problem wurde von Marcel Steinbeck herangetragen:
SEE verwendet eine selbst entwickelte native Library *tinyspline* [SK21b] zum Zeichen von Kanten. Diese Library müsste ebenfalls für macOS kompatibel sein - insbesondere für die M1-Variante. Eine Unterstützung für Intel–bassierte macOS Systeme existiere aber bereits. Es wurde hierfür mehrere Meetings mit Marcel Steinbeck abgehalten und die genauen Schritte wurden besprochen:

Zur Anpassung von *tinysplines* müssen die *CMake* Skripte verändert werden, um so neue Builds für macOS M1 bzw. macOS *universal2* zu ermöglichen (eine spezielle Variante von Binaries unter macOS, die sowohl von Intel macOS Systemen, als auch M1 macOS System unterstützt wird).

4.15.2 Lösung der Probleme

SteamVR Probleme

Zur Lösung des SteamVR Problems wurden in Unity die entsprechenden „.asmdef“ von den *SteamVR* Unity Assets so angepasst, dass diese Library unter macOS bzw. Linux nicht eingebunden wird. Dies führte dazu, dass zwar die Symbolfehler verschwunden sind, allerdings neue Fehler im Code des SEE Projektes auftauchten.

Hierbei wurden die Referenzen zu der *SteamVR* Library nicht gefunden, was aber auch logisch ist, da diese zuvor unter macOS ausgeschlossen wurden.

```
1 ...
2 using UnityEngine.UI;
3 using Valve.VR.InteractionSystem;
4 ...
```

Listing 1: Beispiel aus der Klasse AbstractStateIndicator.cs

Diese Referenzen müssten entsprechend unter macOS bzw. unter Linux ebenfalls ausgeschlossen werden.
Dabei wurde sich dazu entschieden, dies über die Präprozessoren in C# zu behandeln. Diese ermöglichen es, bestimmte Teile des Sourcecodes, noch bevor sie an den Compiler kommen, zu löschen. Hierfür wurde der Präprozessor `INCLUDE\_STEAM\_VR` definiert.

Dieser Präprozessor soll nur gesetzt werden, wenn die VR Funktionalität in den Build von SEE aufgenommen werden soll. Somit müssen alle Referenzen zu dieser Library mit solchen Präprozessoren umschlossen werden.

```
1 ...
2 using UnityEngine.UI;
3 #if INCLUDE_STEAM_VR
4 using Valve.VR.InteractionSystem;
5 #endif
6 ...
```

Listing 2: Angewandte Präprozessoren

Wie im Listing 2 zu sehen ist, wurde die Referenz entsprechend so gesetzt, dass der Import nur vorhanden ist, wenn das Präprozessorsymbol gesetzt wurde. Diese Präpro-

zessorenanweisungen müssten nun auch bei allen Code-Zeilen gesetzt werden, die diese Library benutzen. Z.b.

```
1 #if INCLUDE_STEAM_VR
2         private readonly SteamVR_Action_Single throttleAction =
3  SteamVR_Input.GetSingleAction(XRInput.DefaultActionSetName, XRInput.
      ThrottleActionName);
4 #endif
```

Listing 3: Beispiel aus der Klasse XRPlayerMovement

Die meisten Methoden bzw. Teile von Methoden können einfach so ausgeschlossen werden, da sich diese nur auf VR-Relevante Teile beziehen.

Eine Ausnahme hierbei ist die Extension Methode `Color.ColorWithAlpha` (vorkommen z. B. in der Klasse LabelOperator). Diese liegt im Namespace `Valve.VR.InteractionSystem`, welche zu SteamVR gehört. Diese Methode erstellt ein neues `Color` Objekt und weist ihm einen bestimmten Alphawert zu.
Anfänglich war es nicht direkt klar, warum diese Methode genau benutzt wurde. Die Methode wurde in diversen Kontexten benutzt, die keinerlei Zusammenhang mit SteamVR, oder der VR Funktionalität besitzen, insbesondere, da in SEE bereit eine eigene Implementierung *Color.WithAlpha* vorliegt. Nach Nachforschung konnte diese Methode gefunden werden und die alte Methode `Color.ColorWithAlpha` konnte mit der eigenen Implementierung in SEE ersetzt werden.

Wenn in SEE zukünftig neue Klassen oder Methoden hinzukommen, die VR benutzen, müssten diese ebenso analog zu Listing 2 bzw. Listing 3 behandelt werden.

Wenn die VR-Funktionalität unter macOS bzw. Linux deaktiviert werden soll, ist es zudem sinnvoll, den „Toggle Desktop/VR“ Button auszublenden. Dazu wurde ebenfalls mit Präprozessoren gearbeitet, sodass das Hauptmenü nun wie in Abb. 63 aussieht unter macOS bzw. Linux:

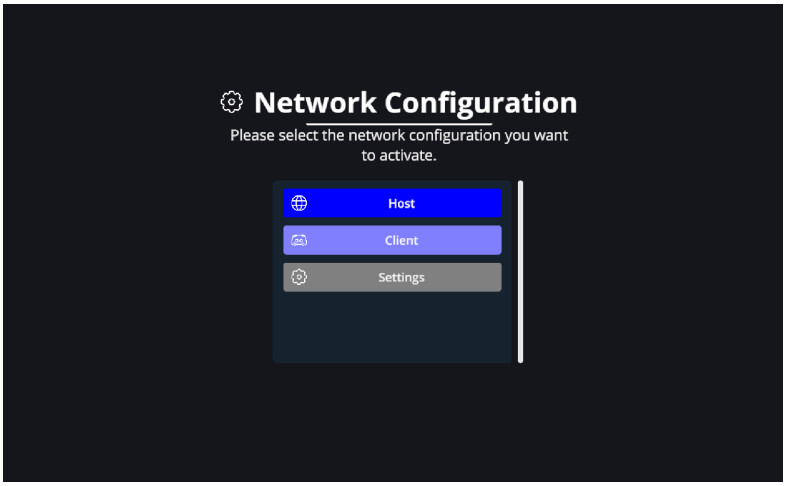


Abbildung 63: Screenshot des Hauptmenüs ohne VR

Es ist zu beachten, dass, auch wenn mit diesem Feature weitere Plattformen eröffnet werden, sowie das Cross-Compiling möglich wird, dass eine VR Version von SEE weiterhin nur unter Windows gebaut werden kann.

Anpassung von *tinyspline*

Wie bereits in Abschnitt 4.15.1 beschrieben, mussten die CMake-Skripte von *tinyspline* so angepasst werden, dass auch M1-MacOS Systemen bzw. universal2 unterstützt wird. Die CMake-Skripte unterstützen bereits die *arm64* Architektur, wie ein kurzer Test zusammen mit Marcel Steinbeck zeigte. Somit muss nur noch eine Version für *universal2* gebaut werden.

In CMake gibt es eine Variable `CMAKE\_OSX\_ARCHITECTURES`, die gesetzt wird, um die Architekturen zu spezifizieren, die unter MacOS gebaut werden sollen. Konkret handelt es sich dabei um eine Liste (z.B. „x86_64;arm64“), diese muss geprüft werden, ob sie sowohl *x86_64*, als auch *arm64* enthält. Dies bedeutet, dass CMake die Binary für universal2 baut.
Da wir SEE u. A. mit einem C# Interface ausliefern, müssen wir die neu gebaute Binary auch in das Nuget-Paket einbinden, da wir dieses auch in SEE benötigen.

In der Dateistruktur von *tinyspline* gibt es die Datei *TargetArch.cmake*, die die Aufgabe hat, die Architektur, für die *tinyspline* gebaut werden soll, zu ermitteln. In dieser Datei wurde eine Prüfung eingebaut, ob beide Architekturen gesetzt wurden. In diesem Fall wird dann die CMake-Variabel *ARCH* auf *universal2* gesetzt.

```
1  # Check for macOS universal2.
2  list(LENGTH CMAKE_OSX_ARCHITECTURES OSX_ARCH_LENGTH)
3  if(OSX_ARCH_LENGTH EQUAL 2)
4    list(FIND CMAKE_OSX_ARCHITECTURES "x86_64" OSX_X86_INDEX)
```

```
5     list(FIND CMAKE_OSX_ARCHITECTURES "arm64" OSX_ARM64_INDEX)
6     if((${OSX_X86_INDEX} GREATER -1) AND (${OSX_ARM64_INDEX} GREATER -1))
7       set(ARCH "universal2")
8     endif()
9   endif()
```

Listing 4: Ausschnitt aus der Datei TargetArch.cmake - Zeile 93 - 101

In dem eigentlichen CMake-Skript von *tinyspline* musste dann eine weitere Prüfung eingebaut werden. Wenn die Variable *ARCH* entsprechend auf *universal2* gesetzt wurde, dann muss die weitere CMake-Variabel `TINYSPLINE\_NUGET\_RID` auf *universal2* gesetzt werden. Diese weitere Variable steuert, in welche Ordnerstruktur die Library in dem Nuget-Paket eingebunden werden soll - in diesem Fall in den Ordner *osx-universal2*. Zuletzt wurde die workflow YAML Datei von der GitHub Action so angepasst, dass eine passende Version für Apple M1 bzw. universal2 gebaut wird. Dazu wurde die o.G. CMake-Variabel `DCMAKE\_OSX\_ARCHITECTURES` benutzt, um einen entsprechenden Job in der Build-Pipeline zu erstellen, der eine passende Version für die *arm64* Architektur unter macOS kompiliert.

Die fertig angepasste Version von *tinyspline* konnte dann in das Unity Projekt eingebunden werden, indem alle alten Binaries von *tinyspline* mit den neuen ersetzt wurden. Dabei ist zu beachten, dass die einzelnen Architekturen der Binaries nur auf den entsprechenden Plattformen bzw. Architekturen eingebunden werden, siehe Abb. 64.

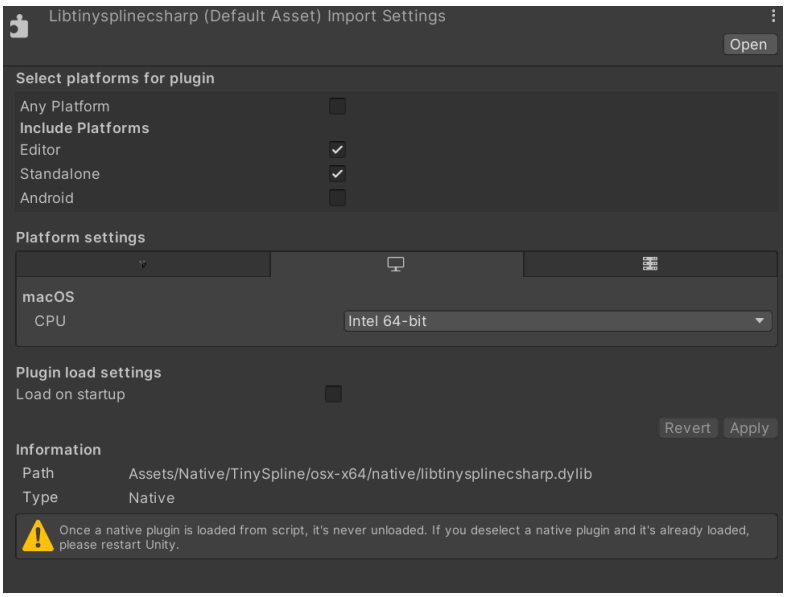


Abbildung 64: Einstellung der tinyspline Binaries

4.15.3 Aufgetretene Probleme und Fazit

Es war zunächst etwas schwer, sich in das System von CMake einzuarbeiten. Nachdem die Grundprinzipien, sowie die Struktur von den *tinyspline* Skripten verstanden wurden, konnte die Aufgabe gelöst werden.

Nachdem *tinyspline* angepasst wurde, wurden die neu erstellten Binaries in das Unity-Projekt eingebunden und alle alten Binaries von *tinyspline* ersetzt. Dabei war es wichtig, dass alle Dateien ersetzt werden, damit unter allen Plattformen die gleiche Version von *tinyspline* läuft. Zuerst wurde allerdings eine fehlerhafte Version von *tinyspline* in das Projekt eingebunden.

Dies fiel aber erst beim Erstellen des Pull-Requestes auf und wurde sogleich behoben.

Ein Ähnliches Problem trat auf, als nach einem Merge vom Master Branch eine Library names „LZMA“ nicht gefunden werden konnte unter macOS in der M1 Variante.

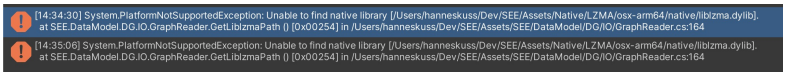


Abbildung 65: Screenshot der Logs im Unity Editor

Wie in Abb. 65 zu sehen ist, fehlen die Binaries für eine Library, für die M1 macOS Variante. Nach kurzer Recherche hat sich herausgestellt, dass es sich um die Library „Joveler.Compression.XZ“ [ied] handelt, die in SEE seit kurzem verwendet wird. Beim Einbinden der Library wurde offenbar die M1 Variante von macOS vergessen. Somit bestand die Lösung dieses Problems darin, dass die aktuellste Version des NuGet Paketes manuell heruntergeladen wurde und die Binaries aktualisiert wurden. Es musste hierbei aber auch analog zu Abschnitt 4.15.2 in der Konfiguration der neuen Binaries die Einstellungen so gewählt werden, dass sie zu den entsprechenden Plattformen passen.

Eine wichtige Lektion daraus ist - auch für die Zukunft von SEE, dass beim Einbinden von neuen Binaries immer darauf geachtet wird, dass alle Plattformen bzw. Architekturen bedient werden. Wenn die CI-Pipeline für weitere Plattformen angepasst wird, bzw. eine fertige Kompilierung von SEE über diese möglich wird, könnten so auch Tests durchgeführt werden. Diese könnten einfach daraus bestehen, ob sich SEE für alle von uns unterstützen Plattformen ohne Fehler kompilieren lässt.

Ein weiteres Problem, das auftrat ist, dass SEE im Editor unter der M1 macOS Version auszuführen zwar funktionierte, allerdings die kompilierte Version nicht. Bei der kompilierten Version wurden keine Kanten gezeichnet und die Knoten ließen sich nicht verschieben, skalieren, etc. Es wurde daraufhin ein „Development Build“ erstellt, indem

Logdaten ausgelesen werden können, mit folgendem Ergebnis:

```
1 Fallback handler could not load library /Users/hanneskuss/Dev/SEE/Build/
2     SEEM1.app/Contents/Frameworks/MonoEmbedRuntime/osx/libtinysplinecsharp.dylib
3 Fallback handler could not load library /Users/hanneskuss/Dev/SEE/Build/
4     SEEM1.app/Contents/Frameworks/MonoEmbedRuntime/osx/libtinysplinecsharp.dylib
```

Listing 5: Ausschnitt aus dem Log einer kompilierten Version von SEE

Der Ordner „MonoEmbedRuntime“ existiert allerdings überhaupt nicht in der Ordnerstuktur der macOS „.app“ Datei und die *tinyspline* Binary liegt an einem anderen Ort. Recherchen brachten zu dem Thema leider keine Lösung. Daher wurde in einem Gespräch mit Marcel angenommen, dass der Fehler bei Seiten von Unity liegt.
Da der Build allerdings in der Intel Version und universal2 funktioniert, wurde sich darauf geeinigt, dass SEE unter macOS nur für Intel bzw. universal2 gebaut werden soll und keine eigenständige Version für M1 macOS Systeme. Da insbesondere die Intel Version unter M1 Systemen auch mit *Rosetta* betrieben werden kann, ohne merkbare Qualitätseinbußen.

5 Individuelle Beiträge

5.1 Thilo Beckord

Ich habe mich für dieses Projekt entschieden, da es sehr praxisorientiert ist und ich schon länger den Bereich der Spieleentwicklung ausprobieren wollte. Des Weiteren war es eines von wenigen Projekten, die auch explizit für Studierende der Wirtschaftsinformatik beworben wurden. Ich hatte bereits rudimentäre Kenntnisse in C# aber keine Erfahrung mit Unity. Die Einarbeitung in letzteres gestaltete sich als überraschend leicht (u.a. da Mahmoud Vorkenntnisse in Unity hatte).

Als Aufgabe habe ich mich für die Umsetzung eines Menüs zur Konfiguration von Code Cities zur Laufzeit entschieden, da mir dies als wichtiger Schritt für die Nutzung von SEE in der Praxis erschien. Im Rahmen dieser Aufgabe habe ich sowohl meine Kenntnisse in C# als auch im Umgang mit Git deutlich erweitert. Letzteres lag vor allem an den zahlreichen Problemen, die viele Teilnehmer zu verschiedenen Zeitpunkten hatten. Diese haben vor allem zu Beginn die Bearbeitung der Aufgabe deutlich verzögert. An dieser Stelle möchte ich hervorheben, dass alle Probleme, auf die wir an unterschiedlichen Stellen im Projekt gestoßen sind, äußerst schnell von Rainer und Falko behoben wurden. Darüber hinaus habe ich wertvolle Erfahrungen in der Arbeit an umfangreichen Projekten mit vielen Beteiligten sammeln können. Dazu hatte ich in meinem Studium bisher nicht die Möglichkeit.

Die Arbeit am *RuntimeConfigMenu* mit Mahmoud und Ferdinand gestaltete sich sehr angenehm. Nach dem Plenum am Freitag haben wir regelmäßig gemeinsam im Labor gearbeitet und uns sonst die Aufgaben sinnvoll aufteilen und anschließend zusammenführen können. Die Zusammenarbeit mit den anderen Projektteilnehmenden beschränkte sich weitestgehend auf die wöchentlichen Plena. Dies liegt in der Natur eines solchen Projektes, bei dem an vielen Features zeitgleich gearbeitet wird. Leider hat sich das meiner Meinung nach negativ auf die Organisation des Projekttages ausgewirkt, bei dem wir als Gesamtgruppe nicht gut funktioniert haben. Eventuell hätte eine andere Gestaltung der Plena hier geholfen. Zu oft wurden dort spezifische Probleme einzelner Gruppen diskutiert statt kurze Updates zu geben, Bezüge des Features zu anderen Projektteilen aufzuzeigen und den Fortschritt in der Vorbereitung des Projekttages zu tracken.

5.2 William Behnke

Ich entschied mich für das Projekt, weil es mich interessierte und ich gerne ausprobieren wollte, wie es ist, in einer VR-Umgebung produktiv zu entwickeln. Während meiner Zeit als Werkstudent hatte ich bereits mit Software-Visualisierungen zu tun und war

gespannt darauf, mehr über SEE zu erfahren.

Die Arbeit in unserem Team war äußerst angenehm und ich war glücklich, Kollegen zu haben, die kompetent und humorvoll waren. Es hat Spaß gemacht, mit ihnen zusammenzuarbeiten, und es gab immer jemanden, der sich meine Implementierungsprobleme angehört hat. Während der Entwicklung war ich besonders beeindruckt davon, wie die Universität mich in allen Aspekten unterstützt hat. Falls Hardware-Probleme auftraten, hatte ich immer eine Ansprechperson, so wurden kaputte HDDs schnell durch SSDs ersetzt und neue Software konnte immer unkompliziert installiert werden. Ein Punkt den ich gerne erwähnen würde ist, dass das Labor eine Lüftungsmöglichkeit benötigt. Die stundenlange Entwicklung in VR und die damit verbundene Bewegung waren für den Raum zu viel. Außerdem hätte ich mich gefreut, wenn wir mehr Gruppen hätten, die an direkten VR-Integrationen gearbeitet hätten. Natürlich gibt es nur eine begrenzte Anzahl an möglichen Features in diesem Bereich, aber als jemand, der viel im Labor war, sah ich von anderen Teammitgliedern nur wenig. Ich hätte mir ebenfalls mehr Interaktion auf Entwicklungsebene mit anderen Gruppen gewünscht beispielsweise in Form eines gemeinsamen zweiwöchigen Exkurs in ein besonders schwer lösbares Problem oder ähnliches. So habe ich mich beispielsweise gefreut, als ich meine Arbeit kurz unterbrechen konnte, um Marcel spontan zu helfen, ausländischen Wissenschaftlern SEE vorzustellen.

Die Einarbeitung in SEE war für mich sehr zeitaufwendig, da ich zuvor weder mit C# noch mit Unity gearbeitet hatte. Aber dank der Hilfe von Yvo, der sich in einem anderen Tempo mit der Thematik auseinandersetzte, lief am Ende alles sehr gut. Negativ überrascht hat mich, dass die Entwicklung in VR viel ausprobieren erforderte und dadurch eine augenscheinlich ineffiziente (und auch physisch anstrengende) Arbeitsweise in meinen Arbeitsprozess integriert war. Diese Gedanken haben mich auch auf die Idee gebracht, mich mit dem Konzept einer zusätzlichen virtuellen VR-Box zu beschäftigen, in der VR-Inputs simuliert werden könnten. Diese Idee werde ich wahrscheinlich auch nach dem Bachelor-Projekt weiterverfolgen.

Ich bin ebenfalls sehr froh, dass ich mir die Möglichkeit gegeben wurde, an dem WSRE in Bad Honnef teilzunehmen. Die Entwicklung eines Papers mit Hannes über die Weihnachtstage empfand ich als äußerst angenehm. Auch wenn es am Ende aus organisatorischen Fehltritten nicht als Studenten-Paper eingereicht wurde, bin ich dennoch sehr glücklich, eine wissenschaftliche Publikation mit meinem Namen zu haben. Die Erfahrungen, die ich gemacht habe, und die Möglichkeit, in Bad Honnef eine Präsentation zu halten, waren meine erste Begegnung mit einem äußerst wissenschaftlichen Umfeld, was mich nachhaltig geprägt hat. Den Projekttag empfand ich persönlich als etwas ernüchternd, da ich bestimmte Aspekte meiner Arbeit perfektionieren wollte. Dennoch war es eine gute Erfahrung, die Spaß gemacht hat. Ich konnte mich angenehm mit potentiellen neuen Mitgliedern des Nachfolgeprojekts unterhalten und meine gesammelte Erfahrung aus Bad Honnef nutzen.

Ich fand die Idee eines Plenums sehr gut und es hat mich gefreut, an den entstandenen Diskussionen teilzunehmen und anderen Gruppen hilfreiche Inputs zu geben. Es war auch eine Gelegenheit, meine eigenen Fortschritte zu präsentieren. Das Plenum spielte eine wichtige Rolle, um einen groben Überblick über die Arbeit der anderen Gruppen zu erhalten. Allerdings wurden sehr detaillierte Implementierungsberichte für Außenstehende schnell unübersichtlich.

Ich fühle mich insgesamt sehr bereichert durch meine Teilnahme an diesem Projekt und bin sehr dankbar dafür.

5.3 Zeynep Dilara Degerliyurt

Das SEE-Projekt hat mich am meisten angesprochen und ich fand es sehr interessant. Es hat mir ermöglicht, die Welt von Unity zu entdecken und C# zu lernen.

Als internationale Studentin konnte ich mich problemlos ins Team integrieren, obwohl ich anfangs unsicher war. Ich habe immer Unterstützung gefunden, wenn ich sie gebraucht habe, und habe mich wohl gefühlt. Die Betreuer waren freundlich und hilfsbereit und meine Teamkollegen waren sehr nett. Wir konnten gut zusammenarbeiten, weswegen ich ihnen allen herzlich danken möchte!

Besonders geschätzt habe ich die Möglichkeit, als jemand, der gerne individuell arbeitet, an spezifischen Teilen des Projekts mitzuwirken. Es war vorteilhaft, meine Aufgaben entsprechend meiner Fähigkeiten auszuwählen zu dürfen. Dadurch konnte ich mich auf die Bereiche konzentrieren, in denen ich den größten Beitrag leisten konnte. Es war auch positiv, dass ich meine Arbeit anpassen durfte, wenn ich einmal nicht weiterkam.

Die Projektplanung war präzise und gut strukturiert, um effektive Kommunikation und regelmäßige Statusaktualisierungen sicherzustellen, ohne Zeit zu verschwenden. Ich fand es äußerst hilfreich, an den Meetings online teilnehmen zu können.

Die Team-Events haben viel Spaß gemacht und ich war erfreut, dass ich sogar ein Bowling-Event organisieren durfte.

Insgesamt war das Projekt eine sehr lehrreiche und angenehme Erfahrung.

5.4 Linus Henkensiefken

Das Bachelorprojekt SEE hat mir gut gefallen, da es einige neue Herausforderungen gab und auch interessante Einblicke in neue Gebiete. SEE als Idee zur Visualisierung des Aufbaus von Software fand ich sehr interessant.Allein die Möglichkeit, sich Software nach Metriken oder anderen Eigenschaften visualisieren zu lassen, ist für mich relativ

unerforscht und meiner Meinung nach ziemlich spannend.Auch die Idee, sich Visualisierungen nicht nur einfach am Bildschirm, sondern interaktiv online und sogar in einer virtuellen Welt, welche man mit Desktop sowie auch VR betreten kann, zu betrachten sowie dort zu interagieren, fand ich sehr logisch und sinnvoll.

Ebenfalls das gemeinsame Arbeiten an einem Softwareprojekt finde ich spannend. In Gruppen habe ich an der Uni nur während Corona gearbeitet, also von zu Hause aus. Wir waren höchstens zu dritt und haben auch nie zusammen an Software gearbeitet. Die Überlegung, mit vielen Leuten an einer größeren, schon vorhandenen Software zu arbeiten, fand ich interessant. Ich war mir nicht sicher, wie so was organisiert wird und ob dies eine größere Herausforderung ist, oder sich für mich einfach fast wie von alleine organisiert.Auch war ich mir nicht genau sicher, woran wir arbeiten würden, was unsere Aufgaben sind und was in SEE der nächste Schritt sein wird. Klar war aber, dass wir mit Unity arbeiten würden, wo ich auch wissen wollte, wozu die Engine fähig war und welche Tools uns dort zur Verfügung standen, auch da ich schon mal Unity vor vielen Jahren gesehen hatte.

Zum Beginn wurde uns SEE gezeigt und wir bekamen einen groben Einblick, was das Programm tut und wie es aussieht. Etwas mit dem Aufbau wurden wir alle durch die gleiche Aufgabe, eine “Mark-Action” zu erstellen, vertraut.Ich dachte erst, die Aufgabe sei etwas viel, da wir ja noch nie mit SEE gearbeitet hatten, allerdings war die Aufgabe dank der Anleitung, und dann mit kleiner eigener Recherche doch relativ klar lösbar.

Nachdem alle ihren Einblick in SEE hatten, sammelten wir Ideen, was man als Nächstes tun könnte. Mir fiel die meiner Meinung nach kontraintuitive Steuerung auf. Auch hatte ich ein paar Ideen, z. B. zu der Farbgebung der “Code-City”, oder dass es sinnvoll wäre, eine Funktion einzubauen, um seinen Charakter zu betrachten. Bei der Kantenanimation und dem interaktiven Verhalten ist mir auch einiges aufgefallen, was man noch umsetzen könnte. Visualisierungen von Informationen fand ich schon immer interessant, gerade wenn sie interaktiv sind.

Bei all den Ideen, welche wir gesammelt hatten, waren auch schon einige, die sich sehr stark auf SEE bezogen und nicht so allgemein waren. Diese sagten mir allerdings nicht so viel, und ich hatte das Gefühl, dass ich persönlich zu wenig Erfahrung in diesen Bereichen hatte, und dachte mir, ich beginne erst einmal mit etwas simplen, um dann mehr Einblicke in SEE zu bekommen. Dann würde ich mich auch an etwas mehr auf SEE-Spezifisches, bzw. auf Software-Analyse Bezogenes heranwagen und konzentrieren. Die Erkennung von Mimik hätte ich mir vorstellen können, aber mit OpenCV/FaceMask, oder auch HTC Facial Tracker hatte ich keine Erfahrung. Zusammenführung von Laufzeit- und Evolutionsanimation sagte mir auch nichts. Auch kannte ich keine inkrementellen Layouts. Unter Laufzeitdaten sammeln und visualisieren konnte ich mir zwar was vorstellen, aber ich wusste nicht, wie die Umsetzung aussehen würde. Wie komme ich überhaupt an Laufzeitdaten? Ähnlich war es bei Konfiguration der

Einstellung während des Spiels. Auf was muss ich da überhaupt zugreifen? Und wie? Integration von Sounds (In Unity) schien mir klar zu sein. “Continuous Integration und Test für SEE (DevOps)” hatte ich noch nie gehört.Ich denke, dass mir das alles nichts sagte, liegt etwas daran, dass ich, auch selbst wenn Erfahrung mit etwas habe, manchmal mir die Fachbegriffe nicht merke. Allerdings auch daran, dass auch wegen Corona sehr wenig Austausch mit anderen Studierenden stattfand. Ich hatte Bedenken, dass ich in lauter Ordnern mit lauter unbekannten Dateien keine Ahnung haben würde, was ich überhaupt mit welcher Software tun muss. Mir kam es so vor, als wären die anderen Gruppenmitglieder wesentlich besser informiert. In Zukunft wäre es vielleicht besser, den Teilnehmern klar zu machen, dass sie eigentlich noch nichts von dem Thema wissen müssen, wenn sie eins auswählen. Bzw. einmal Rückfrage stattgefunden hätte, wer was von alledem versteht, und vielleicht dann sogar Teilnehmer sonst die Lehrkraft allen erklären würden, was dieses Thema ist. So hätte man einen besseren Überblick, wenn man vor der Themenauswahl mehr Infos zu den Themen bekommen hätte. Falls nicht schon jedem bewusst ist, worum es geht.Ich denke, wäre mir von Anfang an bewusst gewesen, wie kompetent und gutmöglichst wir unterstützt werden, und auf alle Fragen möglichst klar und ausführlich geantwortet werden würde, egal ob die Frage simpel oder kompliziert war, dann hätte ich mir bestimmt ein aus meiner Sicht unbekannteres und mir komplexeres Thema ausgewählt.

Um mich erst einmal einzuarbeiten, dachte ich, dass ich die Steuerung überarbeite, das erschien mir als der einfachste Einstieg. Ich musste mich erst einmal etwas tiefer in SEE einarbeiten und herausfinden, wie der Player überhaupt zustande kommt, und welcher Code für ihn zuständig ist. Nach und nach verstand ich dann den Zusammenhang zwischen den Skripten und den sogenannten “GameObjects” in Unity.Nachdem ich dann die Fehler behob, führten diese fortlaufend dazu, dass mir vorher nicht so auffällige Fehler zu Gesicht kamen.

Ich arbeitete mich weiter ein und hatte besonders mit dem Avatar, Animation und der Kamera zu tun. Die Gruppenarbeit kam leider nicht wirklich zustande. Wir waren zu zweit und am Anfang haben wir uns ein paarmal getroffen und versucht, den Einstieg in Unity zu schaffen, dabei konnten wir uns auch gut helfen, danach ist die Person leider nicht mehr aufgetaucht.

Als Nächstes wollte ich dann als Einzelperson wie im Meeting festgelegt den Spiegel umsetzen. Ich schätzte ein, dass meine neu erlangten Unity-Kenntnisse, mir dabei gut zur Seite stehen würden. Außerdem interessierte mich, wie man dies umsetzen kann, und somit hatte ich bereits bei dem Arbeiten mit den Animationen einen Prototyp eingebaut, um die Animation leichter sehen zu können.Ich habe den Spiegel stückweise verbessert, da mir aufgefallen ist, dass ich immer mehr beachten musste wie Blickwinkel, Entfernung und Position. Ich habe Dinge ausgeglichen, z. B. mit dem “Field of View” der Kamera, welche den Spiegel darstellt, so wie Berechnungen zur Drehung und Position der Kamera. Dies war komplexer als gedacht, aber auch spannend und interessant. Das Probieren hat viel Spaß gemacht, und ich habe einiges gelernt.

Ich hatte gehofft, dass ich nun mehr mit SEE bzw. Softwareanalyse zu tun habe, und Leute, die mich dem Thema etwas näher bringen können. Ich wurde dann allerdings einer Gruppe zugewiesen, welche sich um die "OpenCV Facemask" kümmert, was aber auch sehr interessant und vielseitig war.In der Gruppe wurden die Aufgaben lieber verteilt, anstatt an einer gemeinsamen Aufgabe zu arbeiten. Dies war okay, da ich bereits so weit in Unity eingearbeitet war, dass ich auch alleine vorankommen konnte, allerdings hätte mich auch gefreut, mehr von der anderen Arbeit mitzubekommen, um dann mehr voneinander zu lernen und zu verstehen.

Bei dem Thema "Facemask" lernte ich Plug-Ins einzubinden, deren Code zu verstehen und selber zu verwenden. Ich habe viel ausprobiert und geschaut, was möglich ist, und konnte so immer mehr einschränken, wozu diese Add-Ons überhaupt benutzt werden konnten.Es wurde immer mehr klar, dass wir wohl so etwas wie eine "FaceCam" über die Nutzer ihr Gesicht mit anderen teilen konnten entwickeln. Ich musst mich viel mit dem Code der Beispiele sowie mit Unity und SEE auseinandersetzen, um all dies hinzubekommen. Für mich besonders kompliziert war der Umgang mit der nicht ganz zuverlässigen Gesichtserkennung und dafür Lösungen zu finden, sowie das für mich komplett neue Thema das Feature auch mit mehreren Nutzern über das Internet verfügbar zu machen. Sich so neue Themen anzueignen und zu implementieren war für mich eine große Herausforderung und hat auch sehr viel Spaß gemacht.

Dabei sowie bei dem ganzen Projekt habe ich sehr viel gelernt. Das Arbeiten zusammen an einer schon erstellten Software ist wirklich sehr interessant und umfangreich. Ich habe gelernt, dass quasi überall Dinge auffindbar sind, welche noch etwas poliert, ändern, verbessert oder erweitert werden könnten. In einem Projekt mit mehreren Leuten gibt es wirklich sehr viel zu tun. Natürlich könnte es auch sein, dass sich all auf gleiche im Ablauf klar definierte Ziele konzentrieren. Aber so wie wir hier relativ frei zusammengearbeitet haben, kannte ich noch nicht und war für mich auf jeden Fall eine gute, neue, lehrreiche und sinnvolle Erfahrung.

Auch habe ich Erfahrungen gesammelt, wenn es ein Übermaß an Aufgaben gibt, mich besser zu organisieren und die Aufgaben herauszupicken, welche ich favorisieren sollte. Man konnte eher alleine oder als Gruppe arbeiten, wobei in Zukunft ich mehr versuchen würde, mich einer Gruppe anzuschließen, um sich austauschen zu können und ggf. gegenseitig Lösung zu finden, welche man sonst übersieht. Das alleinige konzentrieren auf Dinge, die einem auffallen, hat trotzdem Spaß gemacht.Ich habe wieder gemerkt, wie sehr Austausch helfen kann und in Augenblicken Probleme löst, für die man sonst Stunden gebraucht hätte. Mir ist klar geworden, dass manchmal, wenn der Code von dem eignen Verständnis her funktionieren sollte, dies aber nicht tut, es sehr sinnvoll sein kann, sich die Dokumentation oder Umgebung genauer anzuschauen, mit der man arbeitet. Vieles kann nämlich hinter den Kulissen anders funktionieren, als man erst annimmt.

Etwas schade war, dass Unity bei diesem Projekt sehr langsam war, wenn es geöffnet wurde, etwas verändert wurde oder man SEE starten wollte. Die Funktion, dass nicht jedes Mal alles neu kompiliert, geladen und eingestellt wird, wenn man Code ändert oder zu einem anderen Unity Projekt wechselt, soll man angeblich abstellen können, dies hat bei mir aber leider nicht funktioniert. Die Integration mit Visual Studio war gut, aber musste auch erst einmal richtig eingestellt werden, sonst würden viele Fehler kommen.Die Funktionen, die ich von Unity gesehen habe, wirkten meiner Meinung nach nicht optimal umgesetzt. Es könnte leichter sein, von einzelnen Objekten aus untereinander über Code zu kommunizieren.Die Arbeit mit Git war in unserem Fall wirklich praktisch. Negativ war lediglich, dass der Download bei GitHub und GitLab etwas langsam war, und bei einem Fehler, nach meiner Kenntnis, nicht erneut gestartet werden konnte. Positiv war, dass ich lernte, Issues zu erstellen sowie zu bearbeiten, sinnvolle Branches zu erstellen und zu mergen, sowie allgemein Git mit vielen Leuten gemeinsam zu nutzen.Ich würde sagen, nachträglich wäre es noch schön gewesen, etwas mehr mit SEE und Softwarearchitektur zu arbeiten anstatt mit Unity, aber auch die Arbeit mit Unity war sehr interessant und hat viel Spaß gemacht. Sehr gut fand ich die wirklich professionellen Code-Reviews von Rainer und Falko, bei denen ich viel lernen konnte. Und auch die Fähigkeit, selber Infos zu neuen und fremden Themen zu bekommen, sich dann einzuarbeiten, auch in fremden Code, Fragen zu stellen, und neue Sachen zu implementieren, haben sich verbessert. Dabei habe ich wirklich viel gelernt. Ich finde die Gruppenarbeit so sehr positiv und habe nun noch mehr Motivation, in Zukunft in Teams zu arbeiten.

5.5 Max Herrmann

Ich habe mich für das Bachelorprojekt SEE aufgrund seines praktischen und anwendungsbasierten Charakters und der AG Softwaretechnik, die für das Projekt verantwortlich ist, entschieden. Des Weiteren bot SEE für mich eine Möglichkeit, Software-Entwickler verschiedener Erfahrungslevel mit verschiedenen Präferenzen für Entwicklungsumgebungen und Tools in einem gemeinsamen Projekt zu beobachten, und von deren Lösungen zur Zusammenarbeit an einem Projekt zu lernen.

Nach dem Onboarding-Projekt, welches uns Studierenden die Möglichkeit gegeben hat, sich in SEE, Unity und C# einzulesen und an die neuen Gegebenheiten der Entwicklung zu gewöhnen, haben Jonas Schramm und ich uns für die Bearbeitung des Features Edge Animations (4.8) entschieden, da wir mit diesem Feature zum graduellen Auf- und Abbau von Kanten in der Simulation mit vielen verschiedenen Bereichen von SEE (z.B. Erzeugen von Kanten, Rendering von Kanten, Reflexionsanalyse, Operatoren und Input-Events) in Kontakt kommen konnten.

Nach dem Fertigstellen des Features, und einem akzeptierten Pull-Request, haben Jonas und ich gemeinsam an dem Feature *Incremental TreeMap Layouts* gearbeitet.

Als letztes Feature habe ich mich für die *Divergenzauflösung in Reflexionsanalysen* entschieden, weil die Implementierung dieses Features stark mit der Implementierung der Reflexionsanalyse, dem GXL-Format und der Darstellung gekoppelt ist.

Innerhalb des Bachelorprojekts habe ich zusätzlich als Teil des PM-Teams (Projektmanagementteam) agiert, welches für die Moderation und Planung der wöchentlichen Plena, Zuteilung der Protokolle und allgemeine Management-Tätigkeiten zuständig war.

Des Weiteren habe ich als *Außenminister* für das Projekt SEE bei den Planungs-Meetings für den Bachelor-Projekttag 2023 agiert.

Ein weiteres Nebenprojekt war das Überarbeiten des Projekt-Wikis für kommende Generationen an Studierenden, die sich mit dem SEE-Projekt auseinandersetzen werden. Hierbei wollte ich vor allem erreichen, dass Studierende gute Einstiegspunkte finden können, während sie sich in die in SEE verwendete Projektstruktur, Softwarestruktur, CI/CD-Pipeline, Build-Prozesse und Versionskontrollsystem im Rahmen des Onboarding-Prozesses, einarbeiten.

Vor allem in der finalen Phase des Projektes hatte ich die Möglichkeit, über meine Herangehensweise im Verlauf der Projektzeit zu reflektieren.

Zu Beginn des Projektes hätte ich mich im Kontext des Onboarding-Prozess verstärkt mit der Software-Struktur von SEE, der Funktionsweise von Unity und den speziellen Sprachfeatures von C# (z.B. partielle Klassen, `out`-Parameter, `async / await`) auseinandersetzen und diese im kleineren Rahmen testen sollen. Des Weiteren hätte ich mich mit der Verwendung eines Debuggers in Kombination mit Unity verstärkt vertraut machen müssen.

In meiner Funktion als Mitglied des PM-Teams hätte ich die Moderation der wöchentlichen Plena genauer planen und umsetzen sollen. Vor allem Diskussionen, die nicht im Fokus des ganzen Bachelorprojekts lagen, sondern nur für einzelne Gruppen relevant waren, hätten wir früher und vehementer unterbinden sollen. Ich hätte mich außerdem stärker für eine höhere Qualität der Präsentation von Gruppenfortschritten innerhalb der Plena mit Einbindung verschiedener Medien (z.B. PDF-Präsentation, Videos, Bilder) einsetzen müssen, um die Plenumszeit effektiver zu nutzen.

Obwohl Unit- und andere Tests einen Teil des Projekts SEE darstellen und zu den essentiellen Werkzeugen eines Softwareentwicklers gehören, bin ich mit diesen wenig in Kontakt gekommen, und habe keine eigenen Tests geschrieben.

Da es sich bei SEE um ein für Studierende komplexeres Softwareprojekt handelt, wäre eine längere und intensivere Onboarding-Phase mit einem erhöhten Maß an Studierende-Betreuer-Interaktion ein gutes Mittel, um Studierenden einen leichteren Einstieg in die Entwicklung innerhalb von SEE zu ermöglichen. Die Onboarding-Phase könnte mit interaktiven Präsentationen von Studierenden über einzelne Aspekte von SEE oder dem

gemeinsamen Bearbeiten mehrerer kleiner Issues mit den Betreuern im Stil eines *Showcase Issues* erweitert werden, um den Studierenden die anfängliche Orientierungslosigkeit innerhalb des Projektes und der Softwarestruktur zu nehmen. Dies könnte auch eine hervorragende Möglichkeit sein, SEE mithilfe von SEE zu präsentieren.

In diesem Zusammenhang kann man das weitestgehend aktualisierte Wiki und wie man neues Wissen in dieses einpflegen, bzw. vorhandenes Wissen aktualisieren sollte, mit den Studierenden üben.

Ich habe gelernt, wie wichtig es ist, seine eigene Entwicklungsumgebung und Tools (z.B. Debugger, Language Server-Anbindung, Versionskontrollsystem) zu kennen und einzusetzen. Des Weiteren habe ich gelernt, wie wertvoll ein durchdachtes und diszipliniertes Einlesen in die Projekt- und Softwarestruktur ist, und wieviel Zeit und Fehlversuche man sich im Endeffekt spart.

Zusammengefasst wurden meine Erwartungen an das Bachelorprojekt, meine eigenen Arbeitsweisen zu reflektieren und auf einem realen Projekt anwenden zu können, Erfahrungen mit verschiedenen Programmiersprachen und Entwicklungsumgebungen zu sammeln, kritische Diskussionen über gemeinsame Code-Abschnitte zu führen, meine Pairprogramming- und Projektmanagement-Skills zu erweitern und Erfahrungen in einem größeren Team von Entwicklern zu machen, vollständig erfüllt. Basierend auf den gemachten Erfahrungen konnte ich mir neue persönliche Ziele setzen, um in zukünftigen Projekten direkter und klarer kommunizieren und schneller und sicherer produktiv sein zu können.

5.6 Alexander Kaiser

Für SEE entschied ich mich, nachdem ich den Trailer und den Vorstellungstext gelesen hatte. Ich hatte bereits Erfahrung in der Programmierung mit Spieleengines, allerdings nicht auf einem so hohen Niveau. Zuvor hatte ich mit Unity und der Unreal-Engine kleinere Projekte realisiert. Dies hat großes Interesse geweckt größere Projekte mithilfe einer Spieleengine umzusetzen - daher habe ich mich unter anderem für SEE entschieden.

Ein weiterer Grund war, dass ich mich in meinem Unternehmen schon an einigen Stellen mit statischer Codeanalyse beschäftigt hatte. Ich weiß, wie wichtig das Monitoring der Codequalität ist, da ich auf der Arbeit sowohl mit Analysetools gearbeitet, als auch ein eigenes Auswertungstool für Analysekennzahlen für interne Projekte geschrieben habe. Ich wollte unbedingt bei diesem völlig neuen Ansatz von SEE mitwirken und einen echten Mehrwert für die zukünftige Softwareentwicklung schaffen.

Doch anfangs hatte ich Schwierigkeiten in das Projekt zu finden. Zwar gab es eine Einführungsaufgabe, die mich schnell wieder in die Programmierung mit Unity brachte, allerdings war es langfristig schwierig einen echten Mehrwert zu schaffen. Das Feature

der Live Documentation benutzt an einigen Stellen bereits bestehenden Code, doch aufgrund meines sehr überschaubaren Projektwissens, habe ich zunächst probiert eigene Strukturen aufzubauen, obwohl bereits bestehende hätten verwendet werden sollen. Auch kam es oft zu Kompilierfehlern, ausgelöst durch git. Das Wiki gab mir an dieser Stelle wenig Aufschluss über mögliche Lösungen. Nicht nur mir ging es so. Nach Absprache mit den anderen Gruppenmitgliedern, wird das Wiki angepasst und unter anderem um auftretende Probleme und ihre Lösungen erweitert.

Das Finden von Terminen war in unserem Team leider ein Problem, da beide Teammitglieder viel eingespannt waren. Dadurch waren die Absprachen unregelmäßig. In Zukunft würde ich Weeklys bzw. Jour Fixes abklären, um eine Regelmäßgikeit in die Absprache hineinzubringen. Doch ansonsten hat mir die Arbeit im Team sehr gut gefallen.

Die gesamte Arbeit in der Projektgruppe „SEE“ hat mir überaus gut gefallen. Ich konnte nach Absprache an einem selbst ausgedachten Feature entwickeln, wodurch ich sehr motiviert war. Das Feature schafft eine solide Basis für weitere Anpassungen im Rahmen der Dokumentation. Ich freue mich darauf hier anzuknüpfen und innerhalb meiner Bachelorarbeit die Live Documentation um Use-Cases zu erweitern.

5.7 Hannes Lennart Kuß

Am Anfang hatte ich kaum Erfahrungen mit Unity oder der Arbeit mit Game-Engines, aber ich hatte zumindest etwas Erfahrung mit C#. Daher war ich zunächst unsicher, ob ich mich wirklich für SEE anmelden sollte, als die Projekte vorgestellt wurden. Die Idee von SEE, insbesondere die kollaborative Softwarevisualisierung und der VR-Aspekt, hat mich jedoch so begeistert, dass ich mich entschieden habe, die Herausforderung anzunehmen.

Rainer stellte zu Beginn ein Einführungsbuch über Unity vor [SW17], und ich setzte mich daran, mich mit dem Unity-System vertraut zu machen. Dabei stellte sich heraus, dass meine anfänglichen Bedenken nicht ganz unbegründet waren - ich hatte definitiv Schwierigkeiten, mich in Unity einzuarbeiten und auch die komplexe Struktur von SEE war nicht leicht zu verstehen.

Der Onboarding-Task hat mir jedoch sehr geholfen, da er eine recht einfache Aufgabe war, mit der ich mich in die Projektstruktur von SEE einarbeiten konnte.

Besonders gut fand ich, dass Teambuilding-Maßnahmen angeboten wurden. Obwohl ich leider nur an einem der beiden Events teilnehmen konnte, war es schön, die anderen Teilnehmenden des Projekts auch außerhalb der Arbeit kennenzulernen.

Als Rainer den Student-Paper-Wettbewerb vorstellte, wollte ich daran teilnehmen. Ich

dachte, es wäre eine gute Vorbereitung auf meine Bachelorarbeit und würde mir mehr Erfahrung im wissenschaftlichen Schreiben verschaffen. Da das Paper [BK] über die Weihnachtspause geschrieben wurde, konnte ich mich gut darauf konzentrieren und war mit dem Ergebnis sehr zufrieden. Natürlich war es ärgerlich, dass es einen Fehler bei der Abgabe gab, aber ich war sehr erfreut, dass das Paper dennoch akzeptiert wurde.

Die Reise mit William nach Bad Honnef zum Vortrag des Papers war ebenfalls sehr schön. Auf dem WSRE konnten wir das Projekt SEE vorstellen und interessante Gespräche mit den anderen Teilnehmenden der Konferenz führen.

Ein negativer Punkt war die Terminfindung innerhalb des kleinen Teams. Es war manchmal schwierig, einen gemeinsamen Termin zu finden, an dem wir in Ruhe über die Themen diskutieren konnten. In Zukunft werde ich auch proaktiver darauf achten, regelmäßige Termine zum Austausch des aktuellen Standes festzulegen. Auch die Absprachen bezüglich der Aufgabenteilung möchte ich zukünftig besser und häufiger gestalten.

Es war auch schwierig für mich, meine eigene Leistung in dem Projekt einzuschätzen. Ich hatte oft Angst, dass meine Leistung nicht ausreichen würde, um das Projekt mit einer guten Note abzuschließen. Das Zwischengespräch hat zwar diese Angst beseitigt, aber ich fand, dass es etwas spät stattgefunden hat. Natürlich kann man im frühen Stadium eines Projektes den Arbeitsstand nicht genau absehen.

Insgesamt hat mir das Bachelorprojekt jedoch sehr gut gefallen. Ich konnte mich intensiver mit C# beschäftigen, was mich auch in meiner zukünftigen Karriere begleiten wird. Außerdem kann ich mir gut vorstellen, meine Bachelorarbeit im Kontext von SEE zu schreiben.

5.8 Yvo Muskulus

Bei der Wahl des Bachelorprojektes stieß SEE für mich heraus, da die Idee der Softwarevisualisierung und auch die vorhandene Umsetzung für mich sehr interessant ist. Insbesondere die Umsetzung in Virtual Reality und die verbundene Arbeit damit hat mich sehr gereizt.

Anfangs habe ich mir große Sorgen gemacht, ob ich den Erwartungen gerecht werde und mich schnell in das komplexe Projekt einarbeiten kann, da ich zunächst keine Erfahrungen mit der Game-Engine Unity und der Programmiersprache C# besaß. Zu der Zeit beschränkten sich meine Programmierkenntnisse auf die Sprachen Java und C++. Durch die vorgeschlagene Lektüre von Rainer konnte ich mir einen groben Überblick darüber verschaffen, was die Grundfunktionen von Unity sind und wie die Game-Engine aufgebaut ist [SW17]. Außerdem habe ich erkannt, dass C# sehr ähnlich zu Java

ist, wodurch ich mit einem positiven Gefühl in das Projekt starten konnte.

An dieser Stelle möchte ich die Onboarding Aufgabe hervorheben, in der wir eine „Marking-Action“ für Nodes implementieren sollten. Durch diese Aufgabe habe ich die Grundlagen der SEE-Implementierung kennengelernt und einen Einblick in die Struktur und den Code von SEE erhalten. Das hat mir persönlich den Einstieg in das Projekt enorm erleichtert und mir die meisten meiner Sorgen genommen.

Nichts desto trotz habe ich bei der Bearbeitung meines ersten Issues gemerkt, dass ich mich noch tiefer mit der Materie beschäftigen muss und ich viel Zeit investiert habe, um mir ein Verständnis für die Codebasis und Virtual Reality aufzubauen. Nachdem dieses Verständnis jedoch vorhanden war, verlief die weitere Implementierung immer besser und auch die Integration der neueren Features (siehe Abschnitt 8.1) fiel mir deutlich leichter. Positiv habe ich hier die Unterstützung durch die Betreuer wahrgenommen, die bei den Problemen sehr schnell helfen konnten und mir neue Denkansätze und Lösungswege aufgezeigt haben.

Zu dem Projekt gehört natürlich auch die Teamarbeit, Management und Organisation. In meinem Team gab es einige Probleme in der ersten Hälfte des Projektes, was Kommunikation und Absprachen angeht. Es hätte unserer Gruppe gutgetan, wenn sich jemand als Team-Leader präsentiert hätte, der das ganze leitet und führt. Durch die Begleitveranstaltung „Projektmanagement“ ist mir dies klar geworden und ich habe in der Folgezeit versucht die Rolle zu übernehmen. Das hat meiner Auffassung nach auch ganz gut geklappt. Dementsprechend verlief die Kommunikation und Teamarbeit in der Folgezeit deutlich besser.

Sehr positiv fand ich die Teambuilding-Maßnahmen, die ein besseres Kennenlernen ermöglicht und zum Wohlfühlfaktor innerhalb der Projektgruppe deutlich beigetragen haben. Durch die wöchentlichen Meetings habe ich einen Einblick in die Arbeit der anderen Teams erhalten, doch teilweise war es sehr schwierig, den einzelnen Gruppen zu folgen, da einem selber das tiefergehende Wissen in dem Kontext gefehlt hat. Als Verbesserungsvorschlag finde ich die Idee kleiner Präsentationen sehr gut, um die Arbeit der einzelnen Teams besser zu vermitteln.

Negativ könnte man auffassen, dass ich das ganze Projekt über nur mit meinem Partner gearbeitet habe und in den einzelnen Teams so gut wie gar nicht gewechselt wurde. Für mich persönlich hat das aber kein Problem dargestellt und ich bin zufrieden damit. Die Möglichkeit hat bestanden, sich nach dem Abschluss eines Issues umzuorientieren.

Für mich ist es sehr schwierig gewesen, meine eigene Leistung einzuordnen, auch im Hinblick auf die spätere Benotung. Das Zwischengespräch hat zwar einen Eindruck davon vermittelt, dennoch stellte sich mir die ganze Zeit die Frage, ob meine Arbeit ausreichend ist, bzw. ob ich mehr arbeiten muss, um meine Ziele zu erreichen. Vielleicht wären regelmäßigere Zwischenstände zur individuellen Leistung sinnvoll, damit das ganze persönlich besser eingeordnet werden kann.

Insgesamt hat mir der Verlauf des Projektes sehr gut gefallen, insbesondere die praktische Arbeit mit Virtual Reality hat mir sehr viel Spaß gemacht. Ich konnte meine Kenntnisse in C# und der Unity Game-Engine deutlich erweitern und fühle mich mittlerweile sehr selbstbewusst im Umgang damit. Wie schon weiter oben angemerkt, gab es einige Probleme im Bezug auf das Teammanagement. In Zukunft werde ich versuchen, früher Initiative zu ergreifen und gute Kommunikation und Absprache im Fokus zu haben. Abschließend kann ich für mich sagen, dass das Projekt ein voller Erfolg war und die Weiterentwicklung von SEE deutlich zu erkennen ist. Ich freue mich sehr darauf, meine Bachelorarbeit in dem Kontext schreiben zu können.

5.9 Ferdinand Rohlfing

Vom Bachelorprojekt hatte ich im Vorfeld das Interesse, Erfahrungen in einem umfangreicheren Softwareprojekt zu bekommen. Da SEE schon im Vorfeld für meine Verhältnisse eine große Codebase hatte, bin ich über meine Bachelorprojekt-Wahl von SEE sehr froh, da ich hier einige lehrreiche Einblicke und Erfahrungen sammeln konnte.

Doch die Größe des Projektes kam leider auch mit Eigenarten, die mir nicht so gut gefallen haben: So störte die Kompilierungszeit von Unity den Arbeitsfluss teilweise stark, da man bei Codeänderungen manchmal 1-2 Minuten warten musste. Dazu zählen auch die Probleme mit Git, auf die ich zu Beginn des Projektes auch gerne verzichtet hätte.

Dagegen haben mir die Plena im Großen und Ganzen auch sehr gefallen, da man so informative Einblicke in die Fortschritte der anderen Gruppen bekommen konnte, ohne sich dabei mit den einzelnen Themen selber befassen zu müssen. Dennoch wurden manchmal einige Themen für meinen Eindruck nach zu umfassend im Plenum besprochen, da die Diskussionen häufig für die meisten Teilnehmer nicht relevant waren und meist mit einer deutlich kleinen Anzahl an Personen geklärt werden könnte.

Darüber hinaus hat mir vor allem die Dynamik in der Gruppe *Runtime Config Menu* mit Thilo und Mahmoud sehr gefallen: So haben wir sehr häufig im Labor gemeinsam gearbeitet, unter anderem immer freitags nach dem Plenum. Das hat zu einer sehr angenehmen Zusammenarbeit geführt, bei der ich das Gefühl hatte, dass *alle* Gruppenmitglieder ein starkes Interesse an der Bearbeitung der Aufgaben haben, was bei Gruppenarbeiten leider sehr selten ist.

5.10 Jonas Schramm

Ich habe mich dafür entschieden, dass Projekt SEE zu belegen, da ich Erfahrung beim Arbeiten an größeren Projekten sammeln wollte. Insbesondere interessierte mich dabei

der Workflow in einer größeren Gruppe, die Gestaltung von Schnittstellen zwischen Arbeitsgruppen und das Einarbeiten in ein bereits bestehendes Projekt. Außerdem wollte ich mehr über sinnvolle und gängige Konventionen beim Programmieren lernen, sowie eine gute Praxis beim Entwickeln einer Architektur miterleben.

Während des Projekts habe ich an zwei Features gearbeitet. Zunächst habe ich mit Max J. Herrmann das Feature `Gradual Edge Animation` implementiert, im Anschluss habe ich das Feature `Incremtal Layout` bearbeitet, wobei anfangs Max J. Herrmann auch hier mit mir zusammen gearbeitet hat, habe ich später dieses Feature alleine weiter entwickelt. Während des gesamten Projekts habe ich außerdem als Teil des PM-Teams mitgewirkt.

Das Feature `Gradual Edge Animation` war für mich eine naheliegende erste Aufgabe, um einen Einstieg in das Projekt zu finden. Es war ein klar umrissenes Problem und hat an meinem Interesse an numerischer Interpolation angeschlossen. Nach dem Abschluss dieses Features hatte ich Interesse an einer anspruchsvolleren Aufgabe und auch auf etwas mit einem theoretischen Hintergrund. Das hat mich zum Feature `Incremental Layout` geführt, welches ich mit großem Interesse bearbeitet habe.

Rückblickend wurden viele meiner Erwartungen an das Projekt erfüllt.

Ich konnte in verschiedenen Bereichen der Software-Entwicklung Erfahrungen sammeln, wobei gerade der Einblick in ein größeres Projekt einen Einmaligkeitswert innerhalb meines Studiums besitzt. Ich konnte so einen neuen Einblick in die Problematiken und Lösungen erlangen, die beim Arbeiten an einer umfangreicheren Software auftauchen.

Der Austausch mit anderen Entwicklern, die an SEE gearbeitet haben, hatte einen großen Wert für mich und hat mir die Möglichkeit gegeben, meinen eigenen Workflow zu verbessern.

Die Organisation der verschiedenen Arbeiten in Teams innerhalb des Projekts sowie die Abstimmung zwischen den Teams und innerhalb eines Teams hat mir neue Anregungen gegeben, wie das Zusammenarbeiten effizienter gestaltet werden kann.

Darüber hinaus bot das Software-Projekt mir die Gelegenheit, mich mit C# auseinandersetzen und den unerwarteten aber willkommenen Einblick in die Theorie von Layouts.

Das Projekt hätte in einigen Bereichen reibungsloser verlaufen können.

Der Einstiegsprozess in das Projekt hat zwar erste Fragen beantwortet, konnte allerdings keinen guten Überblick die gesamte Software vermitteln. Das fehlende Verständnis über Teile von SEE korrelierte mit der gleichzeitigen, mangelnden Erfahrung mit Unity zum Nachteil der Teilnehmenden.

Ein gepflegtes Wiki auf GitHub über SEE, über Unity im Kontext von SEE sowie über bekannte Probleme und Lösungen hätten einen großen Wert gehabt. Ein etwas ausführlicherer Einstiegsprozess hätte hier ebenfalls einen positiven Effekt.

Das wöchentliche Plenum ist ein weiterer Punkt, bei dem Verbesserungspotenzial existiert. Die Gespräche im Plenum sind zu häufig in Detailfragen abgerutscht und die einzelnen Beiträge hatten nicht den Mehrwert für die Teilnehmer, den sie hätten haben können, unter anderem durch die mangelnde Verwendung unterstützender Medien.

Eine Verbesserung wäre ein anderes Vortrags-Format. Zum Beispiel hat das Begleiten der einzelnen Beiträge mit einer kurzen Beamer-Präsentation (3 bis 5 Slides) nicht nur den Vorteil der visuellen Unterstützung, sondern auch dass der Beitrag vorher kurz durchdacht wurde und einen festen Rahmen hat.

Diese Kritik muss ich zunächst auf mich selbst anwenden, nicht nur als Teilnehmer, sondern auch als Mitglied des PM-Teams.

5.11 Mahmoud Siai

Ich habe mich für dieses Projekt entschieden, weil es ein praxisorientiertes Projekt sein sollte. Im Nachhinein kann ich dem nur zustimmen. Normalerweise kommt man im Rahmen des Studiums nicht mit so großen Projekten in Berührung. Auf der einen Seite ist die Größe des Projektes sehr gut, da man so mit einer großen bestehenden Codebase in Berührung kommt und lernen muss, damit umzugehen. Auf der anderen Seite nimmt es auch viel Zeit in Anspruch, sich in das Projekt einzuarbeiten. Dies war auch eine der größten Schwierigkeiten im Projekt.

Dadurch, dass ich schon während meines Studiums im kleinen Ausmaße mit Unity in Kontakt gekommen bin, blieb mir die anfängliche Überforderung mit der Engine erspart. Dennoch hatte ich recht große Probleme, mich in die Codebase einzuarbeiten, da ich erst etwas später in das Projekt eingestiegen bin. Die Onboarding-Aufgabe hätte mir dabei sicherlich geholfen. Aufgrund des erschöpften Github Datenlimits und des späten Einstiegs in das Projekt konnte ich diese jedoch nicht machen. Dennoch hätte ich mir eine etwas größere Einführung in SEE gewünscht, um unter anderem die Funktionalitäten von SEE kennenzulernen.
In diesem Projekt konnte ich insbesondere meine Kenntnisse in Unity und vor allem in C# vertiefen. Meine Hauptaufgabe bestand darin, ein UI für ein Menü zu bauen, was mir sehr geholfen hat, mein Wissen über UI Design und der verschiedensten UI-Komponenten in Unity zu erweitern. Des Weiteren habe ich auch wieder etwas über Git gelernt, da ich mich vor allem durch die Probleme mit Git-LFS noch tiefer damit auseinandersetzen musste.
Im Zuge dessen möchte ich mich noch bedanken, dass ich das Projekt in meinem Masterstudium als General Studies ablegen durfte und somit mein praktisches Wissen

erweitern konnte.

Das Plenum und die Betreuer sind eine große Bereicherung für das Projekt. An sich hat mir das Plenum sehr gut gefallen, ich hätte mir nur gewünscht, dass spezifische Probleme nach dem Plenum besprochen werden, da nicht immer alle involviert sind und es sehr viel Zeit in Anspruch genommen hat. Hier hätte ich jedoch als Mitglied der Projektmanagementgruppe eingreifen können. Besonders hervorheben möchte ich noch, dass das direkte und detaillierte Feedback, unter anderem beim Pull Request, von den Betreuern sehr hilfreich und lehrreich ist.

In der Gruppe „RuntimeConfigMenu“ haben wir ein Menü realisiert, mit dem wir zur Laufzeit Cities bearbeiten und laden können. Die Einarbeitung in die bestehende Codebase hielt sich hier in Grenzen, da wir ein bestehendes Menü komplett erneuert haben. Zu Beginn des Projektes hatten wir allerdings große Probleme mit Git (-LFS), sodass wir vor allem in den ersten zwei Monaten des Projektes nicht vernünftig arbeiten konnten. Nach dem Git-Rewrite konnten wir uns ohne große Probleme dem Menü widmen und haben es auch aus meiner Sicht zufriedenstellend umgesetzt. Besonders loben möchte ich die Gruppenarbeit in meiner Gruppe, die Aufgabenverteilung hat sehr gut funktioniert und durch die sehr vielen gemeinsamen Pair Programming Sessions hat das Projekt auch sehr viel Spaß gemacht.

5.12 Zhen Yu

Ich habe mich vor allem für dieses Projekt entschieden,um das Programmieren in einem praktischen Projekt besser kennenzulernen. Darüber hinaus hatte ich zwar wenig Kontakt zu VR-Spielen, habe dafür in den letzten Jahren aber immer mehr Interesse gebildet, sodass ich mich über ein Projekt im Bereich VR gefreut habe.

Am Anfang habe ich zusammen mit Sandra an der Mimik mit Hilfe von OpenCV/FaceMask für Desktop gearbeitet. Dafür haben wir gemeinsam opencv für Unity gelernt, wobei wir uns vor allem auf verwandte Beispiele im Internet bezogen. Dabei trafen wir einmal pro Woche und kommunizierten, später zog sich Sandra aber leider aus der SEE-Gruppe zurück. Daraufhin musste ich eine längere Zeit alleine programmieren, wobei ich mich um VTuber kümmern wollte, also um das Gesicht der 3D-Humanoid-Modellierung zu ändern, ich bin dabei an einigen Problemen hängengeblieben.

Dann trat Linus dem Team bei, lieferte neue, von WebCam übernommene Ideen und plante die Arbeit neu. Ich habe zuerst die Schaltfläche zum Umschalten der WebCam hinzugefügt. Nachdem mit Linus meiner Gruppe zugewiesen wurde und die Umsetzungen zusammengefügt wurden, habe ich versucht, den überflüssigen Hintergrundteil des Bildes zu löschen ,um nur das Gesicht beizubehalten. Zunächst habe ich dafür das Bild entsprechend der Gesichtsgröße skaliert, das lief zwar erfolgreich, aber leider nicht flüssig. Linus hat bei der Modifizierung mitgeholfen. Durch spätere Hinzufügungen

von Shadern wurden noch verbleibenden Kanten entfernt.

In diesem Projekt habe ich mich eingehender mit der Verwendung von C# und Unity befasst. Dabei habe ich Besprechungsprozesse und die Kommunikation in einem Großprojekt kennengelernt. Die Kommunikation zwischen mir und den Teammitgliedern war jedoch nicht ausreichend, was es schwierig machte, die falsche Richtung des Denkens zu erkennen und erschwerte die Weiterverfolgung bei den Aufgaben. So muss ich die Kommunikation mit meinem Teamkollegen noch stärken, damit der Fortschritt in zukünftigen Projekten verbessert werden kann. Gleichzeitig muss ich weitere verwandte Aspekte im voraus lernen, um während des Projekts keine zusätzliche Zeit für ergänzendes Lernen aufwenden zu müssen.

6 Fazit und Ausblick

Max Herrmann; William Behnke

Fazit Insgesamt kann das Bachelorprojekt SEE 2022/2023 als erfolgreich aufgefasst werden. Die in Abschnitt 2 aufgezählten Features wurden implementiert und befinden sich teilweise schon im `master branch` des *SEE Repository*. Die Hauptfunktionalität von SEE war bereits vor Beginn des diesjährigen Projekts implementiert und vollständig, was uns die Möglichkeit bot, flexibel an aktuellen Issues zu arbeiten, oder neue Funktionalität rund um das Nutzererlebnis zu implementieren. Die Feature-individuelle Fazite der verschieden zusammengesetzten Gruppen befinden sich in den Abschnitten "Fazit" der Unterabschnitte von Abschnitt 4.

Die Teilnehmer diese Bachelorprojekts konnten durch die vielseitigen Bereiche und regelmäßigen Plenarsitzungen viel über die Notwendigkeit guten Projekt- und Zeitmanagements, die Wichtigkeit guter Programmierkonventionen und Nutzung adequater Programmier-Werkzeuge lernen. Die Software SEE ist um einige Nutzererlebnisverbessernde Komponenten erweitert worden, und ein Teil der existierenden und bekannten Probleme konnte behoben werden.

Ausblick SEE verspricht auch weiterhin als Forschungsprojekt Studentierenden sowie Mitarbeitern des Projekts eine Plattform zu bieten, neue Visualisierungs-Methoden für Software, sowohl im Desktop- als auch im VR-Raum zu testen und zu evaluieren. Durch seinen kontinuierlichen Charakter hat SEE aktuell viele Bereiche, in welchen es weiterentwickelt werden kann. Das neue Laufzeit-Menü kann mit neuen, interaktiveren Methoden für die VR-Welt erweitert werden. Mit der Entwicklung elektronischer Technologie, Programmierebene und VR-Produkten sollte das 3D-Modell des Charakters auch in der Lage sein, einen flexibleren Körperausdruck mit der Bewegung des Körpers in der Realität zu erzielen.Es kann möglich sein, die gleiche barrierefreie Kommunikation wie in der Realität zu erreichen.....

7 Danksagung

Max Herrmann

Wir, die Mitglieder des diesjährigen Bachelorprojekt SEE, möchten uns bei allen Menschen bedanken, die uns während der Projektzeit in jedweder Art tatkräftig unterstützt haben.

Wir möchten uns insbesondere bei der Projektleitung des Projekt SEE, Prof. Dr. Rainer Koschke für die uns zugutegekommene Betreuung, Aufmerksamkeit und Vermittlung von Fachwissen und Erfahrnung, sowie bei unseren Betreuern Marcel Steinbeck und Falko Galperin für ihre Unermüdlichkeit und Geduld im Beantworten von Fragen jeglicher Art und die direkte und indirekte Betreuung während unserer Arbeit an von SEE.

Desweiteren möchten wir uns bei allen Teilnehmern des Bachelorprojekts für ihr Engagement und Hilfsbereitschaft, zielorienterte Kommunikation und ein angenehmes und positives Arbeitsklima bedanken.

Literatur

[ant] antlr. *antlr/grammars-v4*. URL: https://github.com/antlr/grammars-v4/tree/master/csharp.

[az] alexeystrakh und zite. *Unable to use SteamVR on MacOS with the Unity Editor and the Unity Plugin*. URL: https://github.com/ValveSoftware/steamvr_unity_plugin/issues/710.

[BK] William Behnke und Hannes Lennart Kuß. *Collaborative software visualization with SEE*.

[Ble+21] Moritz Blecker u. a. „Projekt SEE“. 2021.

[Eno23a] Enox Software. *Dlib FaceLandmark Detector | Integration | Unity Asset Store*. 2023. URL: https://assetstore.unity.com/packages/tools/integration/dlib-facelandmark-detector-64314?aid=1011l4ehR&utm_campaign=unity_affiliate&utm_medium=affiliate&utm_source=partnerize-linkmaker (besucht am 01. 06. 2023).

[Eno23b] Enox Software. *FaceMask Example | Tutorials | Unity Asset Store*. 2023. URL: https://assetstore.unity.com/packages/templates/tutorials/facemask-example-79999 (besucht am 01. 06. 2023).

[Eno23c] Enox Software. *OpenCV for Unity | Integration | Unity Asset Store*. 2023. URL: https://assetstore.unity.com/packages/tools/integration/opencv-for-unity-21088 (besucht am 01. 06. 2023).

[Epp+09] David Eppstein u. a. „Area-Universal Rectangular Layouts“. In: *Proceedings of the Annual Symposium on Computational Geometry* (Feb. 2009). DOI: 10.1145/1542362.1542411.

[Gro23] UMA Steering Group. *UMA 2 - Unity Multipurpose Avatar | 3D Characters | Unity Asset Store*. 2023. URL: https://assetstore.unity.com/packages/3d/characters/uma-2-unity-multipurpose-avatar-35611 (besucht am 29. 05. 2023).

[HWS00] R.C. Holt, A. Winter und A. Schurr. „GXL: toward a standard exchange format“. In: *Proceedings Seventh Working Conference on Reverse Engineering*. 2000, S. 162–171. DOI: 10.1109/WCRE.2000.891463.

[ied] ied206. *Joveler.Compression.XZ 4.2.1*. URL: https://www.nuget.org/packages/Joveler.Compression.XZ.

[Kos11] Rainer Koschke. „Incremental reflexion analysis“. In: *Journal of Software: Evolution and Process* 25.6 (28. Juni 2011), S. 601–637. DOI: 10.1002/smr.542.

[Lan17] Pärtel Lang. *FINAL IK TUTORIAL - Setting Up VRIK for Steam VR*. 2017. URL: https://www.youtube.com/watch?v=6Pfx7lYQiIA (besucht am 01. 06. 2023).

[Mar23] Andrew Marunchak. *UMA - Unity Multipurpose Avatar on the Asset Store!* 2023. URL: https://forum.unity.com/threads/uma-unity-multipurpose-avatar-on-the-asset-store.219175/page-210 (besucht am 29. 05. 2023).

[MB-21] MB-Lab. *MB-Lab*. 2021. URL: https://mb-lab-community.github.io/MB-Lab.github.io/ (besucht am 29. 05. 2023).

[MNS95] Gail C. Murphy, David Notkin und Kevin Sullivan. „Software reflexion models". In: *Proceedings of the 3rd ACM SIGSOFT symposium on Foundations of software engineering - SIGSOFT '95*. ACM Press, 1995. DOI: 10.1145/222124.222136.

[NA07] Hiroshi Nagamochi und Yuusuke Abe. „An approximation algorithm for dissecting a rectangle into rectangles with specified areas". In: *Discrete Applied Mathematics* 155.4 (2007), S. 523–537. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2006.08.005. URL: https://www.sciencedirect.com/science/article/pii/S0166218X06003817.

[Not23] Notion. *Notion – The all-in-one workspace for your notes, tasks, wikis, and databases*. 2023. URL: https://earnestrobot.notion.site/earnestrobot/Auto-Hand-f78a404f6baf4d85ab705c0d1f92c30e (besucht am 31. 05. 2023).

[Num23a] NumesSanguis. *MB-Lab (Manuel Bastioni LAB) — FACSvatar 0.4.0-alpha documentation*. 2023. URL: https://facsvatar.readthedocs.io/en/latest/avatars/mblab.html (besucht am 29. 05. 2023).

[Num23b] NumesSanguis. *NumesSanguis/FACSvatar: An Open Source Modular Framework From Face to FACS Based Avatar Animation (Unity3D / Blender)*. 2023. URL: https://github.com/NumesSanguis/FACSvatar (besucht am 29. 05. 2023).

[Oct23] Octamodius. *Runtime Transform Gizmos | Modeling | Unity Asset Store*. 2023. URL: https://assetstore.unity.com/packages/tools/modeling/runtime-transform-gizmos-125537 (besucht am 31. 05. 2023).

[Par13] Terence Parr. *The Definitive ANTLR 4 Reference*. 1st. Raleigh, NC: The Pragmatic Bookshelf, 2013. URL: https://www.antlr.org/.

[Pau20] Paul Ekman Group. *Facial Action Coding System*. 2020. URL: https://www.paulekman.com/facial-action-coding-system/ (besucht am 29. 05. 2023).

[Rob23] Earnest Robot. *Auto Hand - VR Physics Interaction | Game Toolkits | Unity Asset Store*. 2023. URL: https://assetstore.unity.com/packages/tools/game-toolkits/auto-hand-vr-physics-interaction-165323 (besucht am 29. 05. 2023).

[Roo23] RootMotion. *Final IK | Animation Tools | Unity Asset Store*. 2023. URL: https://assetstore.unity.com/packages/tools/animation/final-ik-14290 (besucht am 29. 05. 2023).

[SK21a] Marcel Steinbeck und Rainer Koschke. „TinySpline: A Small, yet Powerful Library for Interpolating, Transforming, and Querying NURBS, B-Splines, and Bézier Curves“. In: *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2021, S. 572–576. DOI: 10.1109/SANER50967.2021.00068.

[SK21b] 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.

[SSV18] Max Sondag, Bettina Speckmann und Kevin Verbeek. „Stable Treemaps via Local Moves“. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2018), S. 729–738. DOI: 10.1109/TVCG.2017.2745140.

[Stu23] Crazy Minnow Studio. *SALSA LipSync Suite | Animation Tools | Unity Asset Store*. 2023. URL: https://assetstore.unity.com/packages/tools/animation/salsa-lipsync-suite-148442 (besucht am 29. 05. 2023).

[SW17] Carsten Seifert und Jan Wislaug. *Spiele entwickeln mit Unity 5: 2D- und 3D-Games mit Unity und C# für Desktop, Web & Mobile*. 3., aktualisierte Auflage. München: Hanser, 2017. ISBN: 978-3-446-45197-1. URL: http://www.hanser-fachbuch.de/9783446451971.

[Tec23a] Unity Technologies. *Unity - Manual: Layer-based collision detection*. 2023. URL: https://docs.unity3d.com/Manual/LayerBasedCollision.html (besucht am 31. 05. 2023).

[Tec23b] Unity Technologies. *Unity - Scripting API: Rigidbody.collisionDetectionMode*. 2023. URL: https://docs.unity3d.com/ScriptReference/Rigidbody-collisionDetectionMode.html (besucht am 31. 05. 2023).

[Uni] Unity. *OpenXR Plugin*. URL: https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.7/manual/index.html.

8 Anhang

8.1 Issue-Tabelle

Tabelle 1: Die von den einzelnen Teilnehmern bearbeiteten Issues.

Person	**Issue ID**	**PR ID**	**Issue-Titel**
Thilo Beckord	#503	#509	Configuration of SEE-Cities during Runtime
William Behnke	#497	#542	HTC Vive Facial Tracker Implementation
	#524	#557	A local VR avatar needs a different way to propagate its movements to remote clients
	#544		Use new Unity XR subsystem, get rid of SteamVR
Zeynep Dilara Degerliyurt	#501, #509	#547	Implement Game Sounds, Implement sound framework
	#560		Live transmission of the face, in front of the avatar.
	#571	#605	Replace scale gizmos by Runtime Transform Gizmos
	#572		Integrate grid placement system plugin
	#606	#609	Fix continuous sound bug
Linus Henkensiefken	#510	#521, #552	Improved desktop camera control
	#522	#523	Players should have the possibility of seeing themselves
	#532	#552	Camera attached to player avatar should not show the head or hair of the avatar
	#560		Live transmission of the face, in front of the avatar
Max Herrmann	#498	#558	Gradual Edge Animation
	#599		Synchronize Edges from Implementation into Architecture in Reflexion Cities
Alexander Kaiser	#611		LiveDocumentation : Enhancing Class Documentation in SEE
Hannes Lennart Kuß	#496	#577	Porting to MacOS/Linux

	#611	#597	LiveDocumentation : Enhancing Class Documentation in SEE
Yvo Muskulus	#497	#542	HTC Vive Facial Tracker Implementation
	#524	#557	A local VR avatar needs a different way to propagate its movements to remote clients
	#544		Use new Unity XR subsystem, get rid of SteamVR
	#589		Integrate FACSvatar
Ferdinand Rohlfing	#503	#509	Configuration of SEE-Cities during Runtime
	#541	(#509)	UI Framework Overhaul
Jonas Schramm	#498	#558	Gradual Edge Animation
	#563		Incremental Layouts
Mahmoud Siai	#503	#509	Configuration of SEE-Cities during Runtime
Zhen Yu	#560		Live transmission of the face, in front of the avatar.