IFS INSTITUTE FOR SOFTWARE

OST Ostschweizer Fachhochschule

Software Visualisierung durch VR mit Webtechnologien

Studienarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Herbstsemester 2021

Autoren:	Thomas Hindermann, Joel Hirzel
Betreuer:	Prof. Dr.-Ing Frieder Loch
Projektpartner:	IFS – Institute for Software, OST Campus Rapperswil

Abstract

Virtuelle Realität (VR) gewinnt zunehmend an Bedeutung. Der technologische Fortschritt öffnet Türen für neue Anwendungen und verspricht benutzerfreundlichere Hardware. Die meisten VR-Anwendungen sind plattformspezifisch und somit nur auf spezifischen VR-Geräten ausführbar. Eine weitverbreitete Lösung, um plattformunabhängige Applikationen zu entwickeln, ist die Verwendung von Webtechnologien. Ziel dieser Arbeit ist es deshalb herauszufinden, wie gut sich eine VR-Applikation mit heutigen Webtechnologien umsetzen lässt.

Nach einer kurzen Recherche bezüglich VR und verwandten Themen stellte sich die Frage, welche Technologien für die Umsetzung zur Verfügung stehen. Dazu wurden mögliche Kandidaten anhand verschiedener Kriterien miteinender verglichen. Gleichzeitig wurde mit der Software City ein konkreter Anwendungsfall gefunden und nach möglichen Lösungsansätzen gesucht. Um die Handhabung der Technologien besser einschätzen zu können, wurde jeweils ein Prototyp einer minimalen Software City mit den drei vielversprechendsten Kandidaten erstellt. Babylon.js konnte die Technologiewahl für sich entscheiden und wurde mittels der Software City genauer erprobt.

Das Resultat ist ein funktionierender Prototyp, der einen neuen Blick auf seinen geschriebenen Java-Code erlaubt. Eine interaktive Stadt mit praktischen Funktionen wird durch spielerische Elemente belebt.

Das Vertrauen in die gewählte Technologie bestätigte sich schnell und ermöglichte eine unkomplizierte Umsetzung. Babylon.js stellte sich als ein intuitives, mächtiges Open Source Framework heraus und unterstützt den Entwickler ideal beim Erstellen einer VR-Anwendung. Ein Hindernis im implementierten Prototypen stellte die Benutzerfreundlichkeit dar, was nicht ausschliesslich auf die unbekannte Handhabung von VR-Applikationen zurückzuführen ist. Einige Lösungen dafür konnten bereits erläutert werden, andere bedürfen tieferer Recherche. Des Weiteren kam die Hardware öfters an seine Limiten, was eine unangenehme VR-Erfahrung bescheren kann. Hier braucht es wohl noch etwas Zeit, Hardware und Software gleichermassen, um ein gutes Erlebnis zu ermöglichen.

Management Summary

Ausgangslage

Der Anlass dieser Arbeit ist die zunehmende Bedeutung von VR. Die Ausführungen dieses Dokuments stellen eine Machbarkeitsstudie für VR-Applikationen dar. Der Fokus liegt dabei nicht auf nativen Applikationen, sondern auf webbasierten VR-Anwendungen. Durch die inhärent plattformübergreifende Natur von Weblösungen, lohnt sich deshalb eine detailliertere Untersuchung.

Vorgehen

Anfängliche Recherchen sollen die Nomenklatur klären. Um einen Überblick über die aktuellen Technologien zu gewinnen, zielt ein Teil der Recherchen darauf ab, diese in einen Vergleich zu stellen. Anhand einer Software City Anwendung wird dann eine ausgewählte Technologie, in diesem Fall Babylon.js, erprobt und genauer analysiert. Eine ausführliche Planung und Analyse der Software City stellt sicher, dass die Umsetzung einem roten Faden folgt. Die Endergebnisse und Erfahrungen werden festgehalten und diskutiert.

Ergebnisse

Babylon.js stellte sich als eine angenehme Lösung für die Umsetzung von VR-Applikationen heraus. Durch den Fokus des Frameworks auf Einfachheit bietet sich die Technologie für die meisten Entwickler an, die plattformübergreifende VR-Anwendungen implementieren wollen. Einzig die Performance liess zu wünschen übrig. Das liegt aber zu einem gewissen Grad auch an der leistungsschwächeren Hardware im Vergleich zu Desktop PCs. Die Software City konnte erfolgreich implementiert werden. Das Erlebnis, wenn man in die Stadt hineingeht, ist immersiv und ermöglicht dem Benutzer seinen Code in einer Art und Weise zu erleben, wie es bisher nur wenige kennen dürften.

Danksagung

Wir möchten in erster Linie unserer Betreuungsperson Herrn Prof. Dr.-Ing Frieder Loch danken. Die Zusammenarbeit mit ihm hat uns, nicht zuletzt auch aufgrund seiner positiven und zuvorkommenden Art, Freude bereitet. Er hat uns stets tatkräftig unterstützt und gab immer wieder tolle Ideen und nützliche Inputs, wie die Applikation erweitert werden könnte. Bei der Planung der Wochensitzungen war er immer flexibel und nahm sich sogar für spontane Treffen Zeit mit uns. Auch für das Gegenlesen der Arbeit und sein jeweils ausführliches Feedback wollen wir uns bedanken.

Auch den Teilnehmern des Usability Tests möchten wir für ihr wertvolles Feedback danken. Durch ihre Ausführungen konnten wertvolle neue Einblicke in die Applikation gewonnen werden, die uns selber nicht aufgefallen wären.

Des Weiteren möchten wir unserem Kommilitonen Abinas Kuganathan danken, der uns im Bereich UI-Design einige Inspirationen geliefert hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

1.1 Beschreibung des Problems

Virtuelle Realität (VR) gewinnt zunehmend an Bedeutung. Die Anwendungsfälle reichen von Unterhaltungsanwendungen, wie Computerspielen, 3D-Kino oder virtuellen, sozialen Welten, bis hin zu Trainingsumgebungen. Auch Freizeitparks haben die Vorzüge von Virtual Reality für sich entdeckt, indem sie Achterbahnen mit Virtual Reality Brillen kombinieren. Abseits der Unterhaltungsanwendungen kann man auch in der Businesswelt Gebrauch von VR machen. Businessmeetings in VR ermöglichen ein immersiveres Erlebnis als klassische Videokonferenzen. Präsentationen oder 3D Modelle von Produkten oder Prototypen können hochgeladen und interaktiv genutzt werden. Im Bereich Bildung gibt es die Möglichkeit, mittels VR sichere Trainingsumgebungen zu erzeugen, ohne die Konsequenzen der realen Welt tragen zu müssen. Auch die Datenvisualisierung profitiert von den Möglichkeiten mit VR, indem die Daten auf natürliche Weise auch in der dritten Dimension dargestellt werden können.

Die Umsetzung solcher Applikationen basiert meistens auf proprietären Technologien und ist somit automatisch an bestimmte Plattformen gebunden. So läuft eine für die Oculus Quest entwickelte Applikation nicht automatisch auch auf einer Vive. Durch den Einsatz von Webtechnologien werden sich offene und plattformübergreifende Lösungen erhofft.

1.2 Ziel der Arbeit

In dieser Arbeit soll die Frage beantwortet werden, wie gut sich die virtuelle Realität mit Webtechnologien umsetzen lässt. Es werden verschiedene Tools und Frameworks miteinander verglichen, um dann mit Hilfe der Implementation eines Prototypen ein Fazit zu ziehen. Die Testapplikation beschäftigt sich mit einer unkonventionellen Art Software zu visualisieren. Die Idee ist es, Softwareprojekte in Form von Software Cities darstellen zu können.

1.3 Aufbau der Arbeit

Im anschliessenden Kapitel geht es darum, Virtual Reality zu erläutern und Anwendungsfälle zu analysieren. Das dritte Kapitel befasst sich mit den verschiedenen Webtechnologien. Es werden verschiedene Technologien vorgestellt und anhand eines Kriterienkatalogs miteinander verglichen. Auserdem wird der Anwendungsfall einer Software City geplant,

um damit später eine ausgewählte Technologie zu erproben. Daraufhin werden die funktionalen und nichtfunktionalen Anforderungen des Anwendungsfalls festgelegt. Die Softwarearchitektur ist im fünften Kapitel beschrieben. Das Kapitel Umsetzung schildert die Vorgehensweise während der Entwicklung und beschreibt in einer Designdokumentation die visuellen Aspekte des zu implementierenden Anwendungsfalls. Die Resultate inklusive Qualitätsmanagement werden im Kapitel Resultat veranschaulicht. Das letzte Kapitel diskutiert die Implementation des Anwendungsfalls und bewertet kritisch die angewandte Technologie. Es wird dort ein Fazit gezogen und die anfängliche Frage beantwortet, wie gut sich eine VR-Applikation über Webtechnologien umsetzen lässt. Der Projektplan befindet sich im Anhang und erläutert die Vorgehensweise während des Projekts.

Kapitel 2

Begriffserklärung und Anwendungsfälle der Virtual Reality

Übersicht

Im Bereich der Virtual Reality existiert eine Vielzahl an Begriffen, die teilweise unterschiedlich interpretiert werden. Ziel dieses Kapitels ist es, einen Überblick zu verschaffen, was die Fachliteratur vom Begriff Virtual Reality versteht, sowie dessen Einordnung im Reality-Virtuality Continuum zu klären. Ausserdem werden Anwendungsfälle diskutiert und der konkrete Anwendungsfall der Software City genauer beschrieben.

2.1 Grundlagen der Virtual Reality

Wenn man von virtuellen Umgebungen spricht, können verschiedene Abstufungen gemeint sein. Das eine Extrem beinhaltet keinerlei virtuelle Elemente und ist demnach gleich der Realität. Das andere Ende entspricht einer komplett simulierten Umgebung, ohne jeglichen Einfluss der realen Welt. Dazwischen befindet sich das, was allgemein als Augmented Reality und Virtual Reality bekannt ist. Dieser Zusammenhang zwischen Realität und Virtualität wird auch als *Reality-Virtuality Continuum* [Milgram et al., 1994] bezeichnet und ist in der Abbildung 2.1 dargestellt.

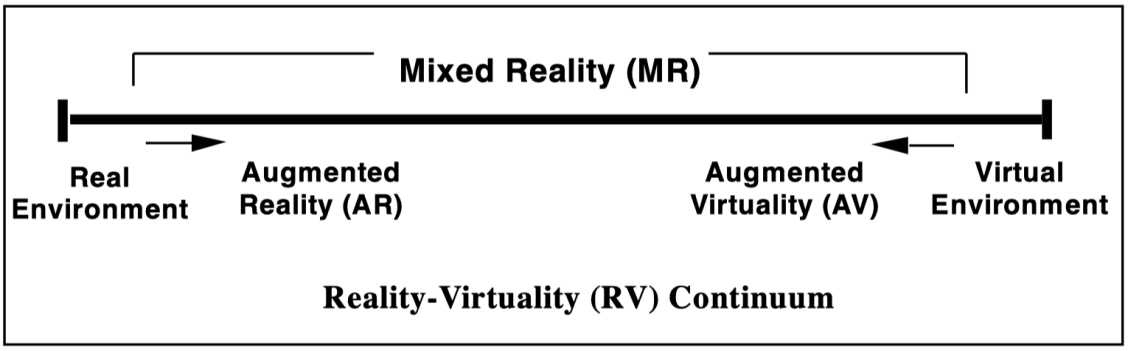


Abbildung 2.1: Reality-Virtuality Continuum

Zu beachten ist, dass die Virtual Reality in der Abbildung in den Bereich Augmented Virtuality fällt. Grund für diese Namenswahl ist, dass man die Virtualität gewissermassen durch reale (d.h. nicht modellierte) Bilddaten oder Objekte (beispielsweise die Hand

des Nutzers als Steuerelement) erweitert (engl. augment). Auf der anderen Seite wird die Realität um virtuelle Elemente erweitert, bis die beiden Seiten in der Mitte aufeinandertreffen. Mit der Weiterentwicklung der Technologien können virtuelle Welten immer realistischer nachgebildet werden. Dadurch kann es in Zukunft immer unklarer werden, ob man durch eine AR-Brille nun reale oder computergenerierte Bilder sieht. Schon heute werden in günstigen Umgebungen reale und virtuelle Bilder erstaunlich gut kombiniert[1]. Das Zusammenspiel von virtuellen und realen Objekten wird durch die Mixed Reality beschrieben und ist somit der Überbegriff von Augmented Reality und Virtual Reality.

Um diesen Abschnitt abzurunden, konkretisieren wir nun unser Verständnis von VR und AR. Augmented Reality fügt, über eine Live Ansicht, der Realität digitale Elemente hinzu. Virtual Reality bedeutet ein vollständiges Eintauchen in eine simulierte Welt, wobei die physische Welt visuell ausgeblendet wird.

2.2 Technische Umsetzung der Virtual Reality

Wie sehen nun solche Technologien aus, über die man in die Welt von VR und AR eintauchen kann und wo liegen die Herausforderungen?

Herausforderungen

Konkret werden in Augmented Reality-Brillen, für den maximalen Grad der Realraumdarstellung, transparente Bildschirme eingesetzt. Das ermöglicht die überzeugendste erweiterte Realität, indem die computergenerierten Bilder direkt in den realen Raum gerendert werden. Die Herausforderung besteht im Tracking des Kopfes mit geringer Latenz und einer präzisen Blickpunktanpassung des Benutzersichtfeldes. Speziell bei transparenten Displays ergibt sich der Nachteil, dass das Auge einen anderen Blickpunkt als die Kamera der Brille hat. Eine weitere Problematik bei der Interaktion von computergenerierten Bildern und realen Objekten entsteht bei überlappenden Objekten. Das virtuelle Objekt fügt sich nur dann nahtlos in die Realität ein, wenn es auch verschwindet, falls es von einem realen Objekt "überdeckt" wird.

Alternativ kann die Realität auch über eine Kamera erfasst werden und auf einem nicht-transparenten Bildschirm dargestellt werden. So kommt es beispielsweise in Smartphones zur Anwendung. In der Abbildung 2.1 wäre diese Variante etwas näher an Virtual Environment als die mit den transparenten Bildschirmen.

Mixed Reality Mixed Reality Headsets basieren auf Technologien, die für Smartphones entwickelt wurden, wie zum Beispiel ein Gyroskop, Beschleunigungssensoren, Bildschirme oder Prozessoren, da dort ähnliche Anforderungen gelten [Wikipedia, 2021c]. Um bei der Virtual Reality einen 3D-Effekt und somit ein Gefühl der Immersion zu erzeugen, müssen zwei Bilder generiert werden, die sich minimal in der Perspektive unterscheiden. Um das jeweilige Bild auf das korrekte Auge zu bringen, existieren verschiedene Varianten. Eine Möglichkeit ist es, für jedes Auge ein separates Display zu verbauen.

[1] Beispiel IKEA App: https://www.architectmagazine.com/technology/ikea-launches-augmented-reality-application

Degrees of Freedom Im Zusammenhang mit Mixed Reality Brillen taucht häufig der Begriff Degrees of freedom (auch DOF, deutsch Freiheitsgrade) auf. Damit sind die verschiedenen Möglichkeiten gemeint, über die sich Objekte im Raum bewegen können. In einem 3 dimensionalen Raum gibt es 6 DOF. 3 für die Translationsbewegungen und 3 für die Rotationsbewegungen. Generell verfügt ein Mixed Reality Headset über 6 DOF, um die maximale Immersion zu gewährleisten.

Umsetzungen

Die Verwendung von Smartphone-Technologien ermöglichte vor allem in der Anfangsphase die Entwicklung preiswerter Headsets für unabhängige Unternehmen wie zum Beispiel Oculus[2]. Ihre Oculus Rift[3] konnten sie so über Kickstarter finanzieren.

Für die Interaktion mit der virtuellen Welt sind spezielle Eingabegeräte denkbar. Dazu gehören die 3D-Maus[4], Datenhandschuhe[5], Motion Controllers[6] oder optische Tracking Sensoren[7]. Einige Eingabegeräte integrieren haptisches Feedback an Hände oder andere Körperteile. So kann sich der Benutzer über mehr Sinnesempfindungen in der virtuellen Welt orientieren und realistischere Simulationen durchführen.

Ein Beispiel einer klassischen Virtual Reality Brille wäre die Oculus Rift oder Oculus Quest[8] Reihe. Die Oculus Quest ist ein sogenanntes standalone device und kann eigenständig und kabellos verwendet werden. Sie unterstützt Positionstracking über sechs Freiheitsgrade, wobei eine Reihe von Kameras an der Vorderseite der Brille anstelle von externen Sensoren verwendet werden. Das macht die Brille mobiler und intuitiver in der Anwendung als noch die Oculus Rift. Durch das Tracking über die Weitwinkelkameras weiss die Brille, wo sich die Controllers befinden. Dieses Tracking wird mit Beschleunigungssensoren im Headset und in den Controllern kombiniert, sowie mit AI-Algorithmen erweitert, um den Bewegungspfad vorherzusagen, sobald sich ein Controller ausserhalb des Kamerasichtfeldes befindet.

Ein bekanntes Beispiel einer Augmented Reality Umsetzung ist die HoloLens beziehungsweise HoloLens 2 von Microsoft. Unter anderem verfügt die Brille über eine Intertiale Messeinheit (IMU), vier "environment understanding" Sensoren, eine TOF-Kamera[9], eine 2,4-Megapixel Fotokamera, ein Array mit vier Mikrofonen und einem Umgebungslichtsensor.[10]

2.2.1 Zusammenspiel mit Webtechnologien

Nachfolgend wird genauer auf die Rolle von Webtechnologien im Bereich Virtual Reality eingegangen. Der Vorteil von Webtechnologien ist die Plattformunabhängigkeit der Anwendungen. Was in Desktop Umgebungen bereits zur Normalität geworden ist, lässt sich auch in die Welt der Virtual Reality übertragen. Ermöglicht wird das durch WebXR.

[2] Eines der ersten Unternehmen, die das Virtual Reality Headset massentauglich machten
[3] Siehe https://en.wikipedia.org/wiki/Oculus_Rift
[4] Siehe https://en.wikipedia.org/wiki/Computer_mouse#3D_mice
[5] Siehe https://en.wikipedia.org/wiki/Wired_glove
[6] Siehe https://en.wikipedia.org/wiki/Motion_controller
[7] Siehe https://en.wikipedia.org/wiki/Motion_capture
[8] Siehe https://en.wikipedia.org/wiki/Oculus_Quest
[9] Time of Flight Kamera
[10] Siehe https://en.wikipedia.org/wiki/Microsoft_HoloLens

OpenGL und OpenGL ES

OpenGL ist eine plattformübergreifende Graphics API und spezifiziert ein Standard-Interface für Grafikkarten [AndroidDocumentation, 2021]. Die API wurde von der Khronos Group, einer non-profit Organisation, entwickelt und gemanaged. OpenGL ES ist eine Variante der OpenGL-Spezifikation, die für Embedded Systems gedacht ist. Das Betriebsystem der Oculus Quest ist Android und Android unterstützt mehrere Versionen der OpenGL ES API.

WebGL

WebGL stammt ebenfalls von Khronos und ist eine weitverbreitete Lösung für die Entwicklung von 3D-Webseiten. Die API basiert auf der Open Graphics Library for Embedded Systems (OpenGL ES), ECMAScript und dem HTML5 Standard.

WebXR

Das XR in WebXR steht für "Extended Reality". Dieser Begriff umfasst VR, AR und weitere Technologien, wie beispielsweise mobile Geräte mit Positionstracking. WebXR ist eine Gruppe von Standards, die ein gemeinsames Ziel haben, das Rendering von 3D Szenen auf Hardware zu untersützen, welche für die Darstellung von VR oder AR Bildern entwickelt wurde. Definiert wurde dieser Standard von der Standardisierungsgruppe W3C. WebXR selbst ist keine rendering Technologie und unterstützt auch kein Managing von 3D-Daten [MDNWebDocs, 2021].

WebXR Device API

Das WebXR Device API implementiert den Kern des WebXR Featuresets, damit die Ausgabegeräte die 3D Szenen rendern können.

Durch WebXR wird es Webentwicklern somit ermöglicht, VR- als auch AR-Applikationen einmalig für verschiedene Hardware zu entwickeln. Die meisten Anwendungen verwenden dabei WebGL, oder eine Abstraktion davon, als Rendering Engine. Benutzer dieser Applikiationen müssen dafür nur einen Browser mit implementierter WebXR Device API starten, ohne zusätzliche Software installieren zu müssen [TK, 2021].

WebXR Lifecycle

Die meisten Applikationen, die WebXR verwenden, folgen einem wohldefinierten Lebenszyklus [WebXR, 2021]:

1. Abfrage, ob der gewünschte XR-Modus unterstützt wird.
2. Wenn eine Unterstützung verfügbar ist, wird der Benutzer auf die XR-Funktionalität hingewiesen.
3. Das "user-activation" Event signalisiert, dass der Benutzer XR verwenden möchte.
4. Request einer immersiven Session an das Gerät
5. Die Session wird verwendet um einen Rendering-Loop auszuführen, der die Sensordaten aktualisiert und grafische Bilder erzeugt, die auf dem XR-Gerät angezeigt werden.

6. Fortfahren, bis der Benutzer signalisiert, dass er den XR-Modus verlassen will.
7. XR-Session beenden.

WebXR Browserunterstützung

In der Tabelle 2.1 befindet sich eine Auswahl an Browsern, die WebXR unterstützen.

Tabelle 2.1: WebXR Browserunterstützung

Browser	Headsets	Beschreibung
Firefox Reality	Oculus Headsets, HTC Vive, HoloLens	Der Browser wurde von Mozilla speziell für VR entwickelt.
Oculus Browser	Oculus Headsets	Dieser Browser wurde von Oculus für ihre eigenen Brillen entwickelt
Chrome	-	Chrome als Desktop Anwendung bietet sich speziell für die Entwicklung gut an, da über eine Emulatorextension ein WebXR Device emuliert werden kann.
Edge	HoloLens 2	-

Grafikkarten

Dadurch dass GPUs mittlerweile auch auf mobilen Geräten realisierbar sind, profitieren auch VR-Headsets. So können Berechnungen der 3D-Welten von der umfangreichen Parallelisierung der GPUs Gebrauch machen. Dank WebXR funktioniert das auch für Webanwendungen. Die WebXR API erweitert die WebGL API, welche wiederum auf den Spezifikationen von OpenGL ES basiert, und ermöglicht grafiklastige Aufgaben auf die Grafikkarte auszulagern.

2.3 Anwendungsbeispiele

Virtual Reality

Der häufigste Anwendungsfall von Virtual Reality ist in der Unterhaltung. Computerspiele, 3D-Kino oder virtuelle, soziale Welten eignen sich perfekt dafür. Auch in der Businesswelt kann VR von Vorteil sein. Präsentationen oder 3D Modelle von Produkten oder Prototypen können hochgeladen und interaktiv genutzt werden. Beispiele für VR-Anwendungen in der Bildung sind Anatomie, Flugsimulationen, Architekturdesign, Fahrschule und Gesundheitspflege [Wikipedia, 2021c].

Ein weiteres beliebtes Anwendungsgebiet für VR-Applikationen ist Training. Durch VR lassen sich sichere Trainingsumgebungen erzeugen, in denen man ohne die Konsequenzen der realen Welt tragen zu müssen, üben kann. Solche Trainingsumgebungen versprechen, flexibel und langfristig kosteneffizient zu sein [Loch et al., 2019].

Training mit VR Die Informationen dieses Abschnitts wurden dem Paper [Loch et al., 2019] entnommen. Die meisten virtuellen Trainingssysteme werden für Montageprozessen oder Wartungsarbeiten verwendet. Man kann den Testpersonen die verschiedenen Schritte detailliert aufzeigen. Wichtig ist, dass die Trainingsumgebung ein exaktes Modell der Maschine, mit der man zusammenarbeitet, umfasst. So können die Trainingspersonen das

Gefühlt erhalten, effektiv vor Ort zu sein. Wie unten beschrieben, ist das nicht immer die beste Wahl. Für eine realitätsgetreue Simulation der Umgebung sind nicht nur die visuellen Aspekte, sondern auch die Akkustik, sowie haptisches Feedback wichtig. Bei Bedarf wäre es auch denkbar Gerüche zu simulieren. Arbeitsschritte können verbal oder visuell über Pfeile kommuniziert werden. Das Ausführen der Arbeitsschritte geschieht mittels klicken eines Knopfes auf dem Controller oder durch eine bewegungsbasierte Interaktion.

Das Paper beschreibt auch, wie Trainingsumgebungen für ältere Personen angepasst werden könnte, um den Rückgang der Wahrnehmungsfähigkeit und kognitiven Fähigkeiten auszugleichen. Die Abbildung 2.2 zeigt eine nicht angepasste Trainingsumgebung. Ältere Menschen haben wegen den ähnlichen Farben in den 3D-Modellen Mühe, die Komponenten der Maschine auseinanderzuhalten. Ausserdem gibt es durch die detailreiche Darstellung der Modelle zu viele Informationen auf einmal.

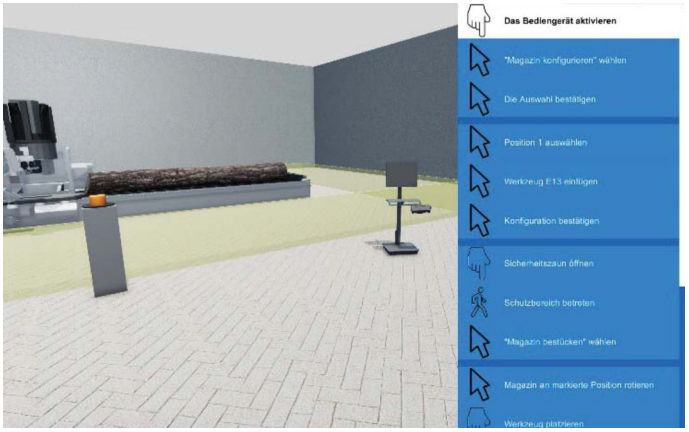


Abbildung 2.2: Nicht angepasste VR-Trainingsumgebung

Eine angepasste Trainingsumgebung ist in der Abbildung 2.3 ersichtlich. Die Farben sind konstrastreicher ausgewählt und die Modelle stark vereinfacht. Auch die Arbeitsschritte werden vereinfacht dargestellt. So kann ein grösserer Lerneffekt für die ältere Testgruppe erzielt werden.

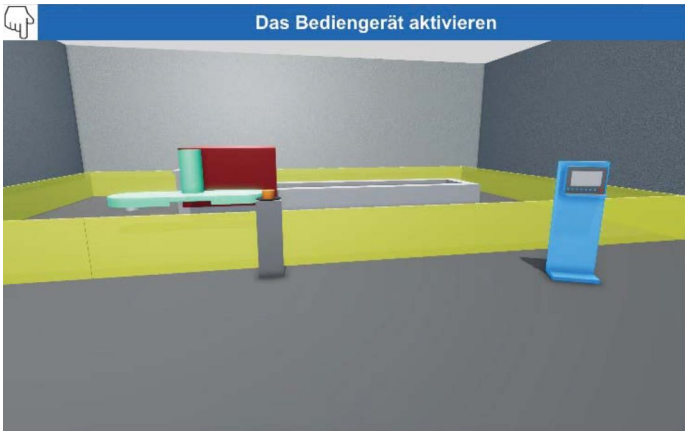


Abbildung 2.3: Angepasste VR-Trainingsumgebung

Augmented Reality

Durch den stärkeren Fokus auf die Realität in Augmented Reality Anwendungen ergeben sich auch hier eine Vielzahl an Anwendungsmöglichkeiten. In medizinischen Umgebungen können beispielsweise Daten von Ultraschall oder CT-Scans mit dem tatsächlichen Körper überlagert werden. In der Architektur kann AR helfen, Gebäude direkt auf dem Grundstück zu visualisieren, bevor mit dem Bau begonnen wird. In der Schule kann den Schülern zu den Textbüchern zusätzliche Informationen in Form eines Bildes, Videos oder 3D-Modells eingeblendet werden. Schriftliche Berechnungen von Studenten in der Mathematik oder Zeichnungen in der Geometrie können live analysiert und die Lösung in-place eingeblendet werden. Handgeschriebene Aufsätze können gescannt und auf Rechtschreibfehler überprüft werden. Auf der Strasse können Navigationssysteme direkt auf der Windschutzscheibe die Route anzeigen, anstatt auf einem separaten Display. Bei Personen, dessen Namen man vergessen hat, kann man mithilfe von Gesichtserkennung den Namen einblenden [Wikipedia, 2021a].

2.4 Anwendungsfall Software City

2.4.1 Software Visualisierung

Die Idee der Software Visualisierung in VR ist nicht neu. Es gab schon mehrere Versuche und experimentelle Umsetzungen in diesem Gebiet [Clinton, 2019]. Die meisten Projekte verfolgen grundsätzlich dasselbe Ziel: Man will die Übersicht auch in grossen Softwareprojekten gewährleisten, indem der Source Code optisch ansprechend dargestellt wird. Dabei galt der Fokus primär auf der Visualisierung statischer Eigenschaften, wie Code Metriken, die unter anderem auch für die Erkennung von Code Smells nützlich sein können. Zur Visualisierung von Code sind verschiedene Metaphern denkbar. Beispiele wären eine galaktische oder Sonnensystem Metapher oder geografische Metaphern wie etwa Inseln [Clinton, 2019]. In dieser Arbeit wurde die City Metapher ausgewählt.

2.4.2 Die City Metahper

Der Vorteil einer Metapher liegt darin, dass dem Betrachter die Situation bereits vertraut vorkommt. Aus der Tatsache, dass die reale Welt als Grundlage dient, folgt, dass der Mensch auf sein natürliches, intuitives Verständnis der physischen Welt zurückgreifen kann. Das vereinfacht die Aufnahme von neuen Informationen und kann zu einer schnelleren und besseren Einschätzung der Situation führen [Russo dos Santos et al., 2000].

Eine Studie hat es sich zur Aufgabe gemacht, die Auswirkungen einer Software City auf das Programmverständnis zu testen und experimentell nachzuweisen. Experimente haben gezeigt, dass sich die Verwendung von CodeCity[11] (Abbildung 2.4) positiver auf die Korrektheit und Geschwindigkeit der Arbeit auswirkt, als herkömmliche Visualisierungstools [Wettel et al., 2011]. Den Probanden wurden dabei Aufgaben gestellt, die ihr Verständnis der Software überprüften.

[11] Siehe https://wettel.github.io/codecity.html

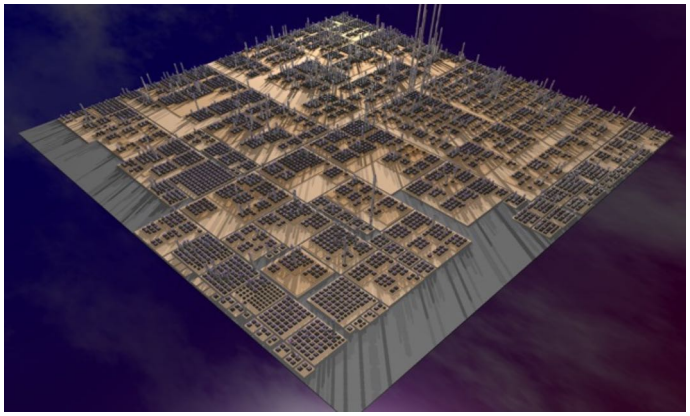


Abbildung 2.4: CodeCity

2.4.3 Parametrisierung der City

Die Frage stellt sich, was man durch eine virtuelle Stadt genau zeigen will. Wenn man eine Programmiersprache wie zum Beispiel Java betrachtet, wird ein Projekt in Packages gegliedert. Diese können weitere Packages oder einzelne Files mit Klassen enthalten. In der Software City könnten die Packages eine Art Stadtviertel darstellen und die Klassen wären die Gebäude darin. Die Dimensionen der Gebäude könnte man mit den verschiedensten Metriken einer Klassen parameterisieren. Beispielsweise könnte die Höhe der Klasse für die Anzahl Zeilen der Klasse stehen und die Breite des Gebäudes für die Anzahl Funktionen [Clinton, 2019]. Es wäre auch denkbar die Farbe der Gebäude anzupassen nach Anzahl Kommentarezeilen. Ein Java Programm hat mindestens eine Einstiegsfunktion. Die Klasse, die diese Funktion beinhaltet könnte speziell dargestellt werden, zum Beispiel in Form eines Stadthauses. Auch Abhängigkeiten und Vererbungshierarchien könnten dargestellt werden.

2.4.4 Techniken für das Software City Layout

Es existieren verschiedene Techniken, wie man das Layout umsetzen kann. Zwei mögliche Varianten sind nachfolgend aufgelistet. Genauere Beschreibungen dieser Varianten mit eigenen Beispielen, wie die konkrete Umsetzung aussehen könnte, befinden sich im Kapitel 3.2.1.

Treemap

Treemapping [Wikipedia, 2021b] ist eine Methode zur Darstellung von hierarchischen baumartigen Darenstrukturen mit Hilfe von Rechtecken. Jeder Zweig des Baumes bildet ein Rechteck, das durch seine Kinder wiederum in kleinere Rechtecke gekachelt wird. Verschiedene Ausprägungen dieses Algorithmus sind über die Zeit entstanden, wobei jede Variante ihre eigene Vor- und Nachteile hat. Unterschieden werden diese Algorithmen in den folgenden Kategorien:

1. **Ordnung**: Beibehaltung einer gewissen Ordnung der Eingabedaten. Sprich Daten, die in der Source nahe beieinander sind, sind auch in der Treemap nahe beieinander.

2. **Seitenverhältnis**: Die Rechtecke sollten idealerweise ein Seitenverhältnis von ca. 1 aufweisen.

3. **Stabilität**: Änderungen in der zugrunde liegender Datenstruktur sollten sich auch in der Treemap wiederspiegeln. Eine kleine Änderung in der Source soll optimalerweise nicht die gesamte Treemap anderst darstellen, sondern nur eine lokale Änderung aufzeigen.

Man kann nicht alle Aspekte gleichzeitig optimieren. Für die Zwecke einer Software City, wäre die wichtigste Metrik ein gutes Seitenverhältnis. Der Squarified Treemap Algorithmus optimiert genau das und bietet sich deshalb gut für diesen Anwendungsfall an. Um die Evolution einer Software City zu betrachten wäre eher der Aspekt Stabilität von Bedeutung.

Street Layout

Das Prinzip des Street Layout Algorithmus [Steinbrückner and Lewerentz, 2010] ist simpel. Jeder Ordner bildet eine Strasse und die Elemente dieses Ordners werden am Strassenrand nebeneinander aufgestellt. Da ein Ordner auch Unterordner beinhalten kann, entsteht ein Geflecht von Strassen. Am Strassenrand befinden sich entweder Gebäude oder weitere abzweigende Strassen. Die Abbildung 2.5 zeigt die Umsetzung von EvoStreets mit dem StreetLayout Algorithmus [Clinton, 2019].

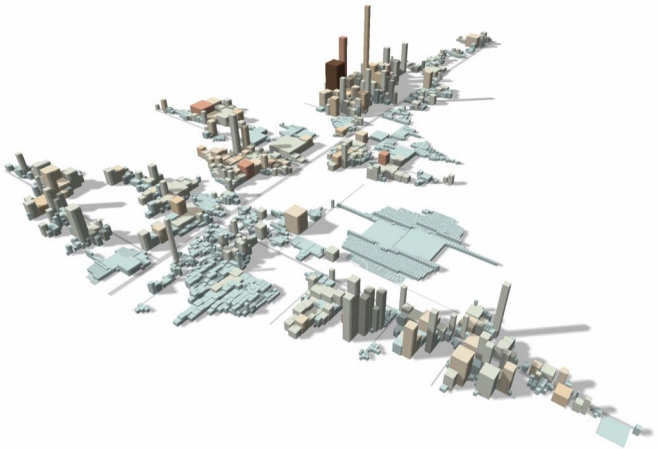


Abbildung 2.5: Beispiel einer Stadt mit dem Street Layout Algorithmus

Zusammenfassung

In diesem Kapitel wurden die Grundlagen der Virtual Reality erläutert. Die Begriffe Virtual Reality, Augmented Reality und Mixed Reality wurden abgeklärt. Auch technische Umsetzungen dieser Technologien wurden aufgezeigt und anhand von konkreten Beispielen genauer beschrieben. Zudem wurde das Zusammenspiel von Virtual Reality und Web Technologien verdeutlicht, sowie die Begriffe OpenGL (ES), WebGL und WebXR behandelt. Zum Schluss beschäftigte sich das Kapitel mit dem konkreten Software City-Anwendungsfall, welcher für den Rest dieser Arbeit weiter verfolgt wird.

Kapitel 3

Wahl der Anwendung und Technologie

Übersicht

Dieses Kapitel bildet die Grundlage für die spätere Umsetzung. Es werden verschiedene Webtechnologien untersucht und verglichen. Anhand eines selbst definierten Kriterienkatalogs werden die verschiedenen Technologien analysiert und bewertet. Anschliessend wird der Anwendungsfall Software City genauer spezifiziert, so dass eine ausgewählte Technologie mit Hilfe dieses Anwendungsfalls erprobt werden kann. Für die drei vielversprechendsten Technologien wird jeweils ein Prototyp einer minimalen Software City erstellt. Aus den dazugewonnenen Erkenntnissen dieser Prototypen wird dann die Technologie auserkoren, die sich am besten für die Umsetzung der Software City eignet.

3.1 Technologievergleich

Nicht ohne Grund erfreuen sich Webtechnologien einer grossen Beliebtheit. Sie sind vielversprechend wenn es darum geht, plattformübergreifende Lösungen zu entwickeln. Durch die Entwicklungen der letzten Jahre sind aus den klassischen Websites neue, dynamischere Webseiten entstanden, die nicht mehr bei jedem Buttonklick die Seite neu laden müssen. Diese Ansätze sind unter den Namen Single Page Application (SPA) oder auch Progressive Web App (PWA) bekannt.

Die Idee ist nun, dass sich diese Vorteile auch in den Bereich der virtuellen Realität übertragen lassen. Anstatt mit proprietären Technologien zu arbeiten und somit automatisch eine geringere Hardwareunterstützung folgt, hat man die Möglichkeit, seine Applikation als Web-App umzusetzen. Alles was die VR-Brillen dann noch benötigen, um die App auszuführen, ist ein installierter Webbrowser. WebXR[1] ist eine Gruppe von Standards, die zusammen das Ziel haben, das Rendering von 3D Szenen auf Hardware zu unterstützen, die für die Darstellung virtueller Welten (VR) oder die Erweiterung der Realität durch Hinzufügen grafischer Bilder (AR) entwickelt wurde. Es ist das Ziel dieser Arbeit herauszufinden, wie gut sich VR-Applikationen mit Hilfe dieser Standards und einer Beispielimplementation umsetzen lässt.

[1] Nachfolger von WebVR

3.1.1 Technologiekandidaten

Eine Auswahl an VR-Technologien soll einen Überblick über aktuelle Technologien verschaffen und deren Funktionsweise aufzeigen.

Game Engines

Es gibt eine Vielzahl an Möglichkeiten, um selbst eine VR-Applikation zu erstellen. Unterstützt wird man dabei unter anderem von:

- Unity
- Unreal Engine
- PlayCanvas
- Wonderland Engine

Viele solcher Engines sind zwar frei zugänglich, jedoch fallen bei den meisten davon Lizenzgebüren an, oder man bezahlt für eine Pro-Version, um alle Features freizuschalten.

Der Vorteil von solchen Engines liegt darin, dass man relativ schnell und unkomlpiziert eine Szene erstellt hat, auch ohne grosse Vorkenntnisse. Ebenfalls kann man diese vielseitig einsetzen. So können damit abseits von Spielen auch Filme erstellt und Animationen gestaltet werden oder als Hilfsmittel im Architektur- und Ingenieurwesen verwendet werden.

Nachfolgend werden Frameworks aufgelistet, durch die man 3D-Szenen fürs Web erstellen kann. Die aufgeführten Frameworks dienen nicht zur 3D-Modellierung, sondern nur zur Darstellung.

Three.js

Die JavaScript 3D Library Three.js ist eine der bekanntesten, wenn es um die Erstellung von 3D-Grafiken für den Browser. Three.js baut auf WebGL auf. Das Open Source Projekt wird in einem Repository auf GitHub gehostet und hat zur Zeit über 1,5k Contributors. Three.js wurde 2010 veröffentlicht. Die Library kann als ein npm-Package oder in der CDN-Version verwendet werden.

Lizenz: MIT

A-Frame

Aufgebaut auf Three.js stellt A-Frame ein unabhängiges Open-Source Framework für 3D, VR und AR Erfahrungen dar. Ursprünglich konzipiert von Mozilla bietet, A-Frame nun seine eigene Webseite[2] mit detaillierter Dokumentation und einem Guide für Anfänger.

Lizenz: MIT

[2] Siehe: https://aframe.io/

Babylon.js

Babylon.js ist 2013 von zwei Microsoft Mitarbeitern als Hobbyprojekt gestartet worden und hat seit dem eine umfassende Entwicklung erlebt. Der Source Code ist in TypeScript geschrieben und fügt sich somit nahtlos in eigene TypeScript oder JavaScript Projekte ein. Das Framework wird über zwei Versionen distribuiert. Eine npm-Version und eine CDN-Version. Die Babylon.js 3D-Engine basiert direkt auf WebGL und ist deshalb auf allen Webbrowsern ausführbar, die WebGL unterstützen. Zahlreiche Spiele wurden bereits mit Babylon.js implementiert[3], darunter auch eine ältere Implementation von Minecraft[4]. Auf ihrer eigenen Webseite[5], sowie auf ihrer GitHub Seite steht eine umfangreiche Dokumentationen bereit. Babylon.js integriert WebXR Für die Entwicklung von VR- und AR-Applikationen.

Lizenz: Apache-2.0

Model Viewer

Model Viewer ist kein Framework um VR-Szenen zu erstellen. Die von Google entwickelte Open-Source Web-Komponente ermöglicht aber eine einfache und mit wenig Code darstellbare Webseite. Dazu muss man dem Model Viewer lediglich ein 3D-Objekt übergeben und sagen, was man damit alles anstellen kann. Ein weiteres tolles Feature ist, dass man mit dem Smartphone Objekte auch einfach im AR-Modus anschauen kann, jedoch wird die Realisierung von VR nicht unterstützt.

Lizenz: Apache-2.0

p5.xr

P5.xr basiert auf der JavaScript Library p5.js[6]. Der Fokus von p5.js liegt darin, das Coding auch für Künstler, Designer und allgemein Anfänger zugänglich zu machen. Aus diesem Grund wird das Framework auch frei zur Verfügung gestellt. Meistens wird p5.js in Kombination mit 2D-Grafiken verwendet. Für die Erweiterung auf 3D-Grafiken wird die WebGL Rendering Engine verwendet. P5.xr ist ein Add-On für die p5.js Bibliothek und kann p5.js-Anwendungen mit Hilfe von WebXR in VR- oder AR-Sketches umwandeln. Die Sketches können dabei wahlweise 2D oder 3D sein.

Lizenz: GNU LGPL

[3] Siehe: https://www.babylonjs.com/games/
[4] Minecraft Classic: https://classic.minecraft.net/?join=gsGbZATuT2UI-R2B
[5] Siehe: https://www.babylonjs.com/
[6] Siehe: https://p5js.org/

React-XR

React-XR[7] ist eine Sammlung an React-Hooks [Dubenko, 2020]. React Anwendungen können damit zu WebXR-Applikationen erweitert werden. Der darunterliegende Renderer bildet react-three-fiber[8]. Dieser Renderer basiert auf Three.js und macht somit Three.js Anwendungen auch in React verfügbar. Die darüberliegende React-XR Schicht erzeugt daraus das VR-Erlebnis. Es können auch Controllers integriert und auf dessen Events reagiert werden, falls beispielsweise mit dem Controller ein 3D-Objekt selektiert wird. React-XR ist nicht zu verwechseln mit React-VR or React-360.

Lizenz: MIT

Verge3D

Verge3D[9] erlaubt es den Benutzern, ihren Inhalt von Modellierungstools wie Blender im Webbrowser anzuzeigen[10]. Die darunterliegende Rendering Engine bildet WebGL. Auch Teile von Three.js werden intern verwendet. Seit der Version 2.10 wird offiziell WebXR unterstützt und ermöglicht es somit, mit Verge3D VR-Anwendungen umzusetzen. Bei der Verwendung von Verge3D unterscheidet sich der Arbeitsablauf grundlegend von den anderen WebGL Frameworks. Man beginnt bei der Modellierung einer Szene, welche dann verwendet werden kann, um das Projekt bei Verge3D zu initialisieren. Die Anwendung kann dann über das Verge3D Netzwerk, oder auf einer eigenen Webseite eingesetzt werden.

Lizenz: Trialware

Blend4Web

Am Anfang seiner Entwicklung war Blend4Web noch unabhängig von Blender und nur als Werkzeug zum Erstellen von WebGL-Inhalten in Blender gedacht. Heute ist Blend4Web eine vollständige, integrierte 3D-Weblösung, bei der Blender weiterhin eine Schlüsselkomponente darstellt. Alle Dienstleistungen im Zusammenhang mit Blend4Web decken auch Blender selbst ab [Blend4Web, 2021].

Lizenz: GNU GPL

3.1.2 Ausschlusskriterien

Gemäss Aufgabenstellung gibt es zwei Ausschlusskriterien:

1. Die Technologie muss Open-Source sein (mindestens Permissive License)
2. Webbasierte Technologie (Plattformunabhängig)

Open Source Definition

Es gibt verschiedene Ansichten, was Open Source bedeutet, dabei begegnet man auch Begriffen wie *Freie Software* und *Copyleft* bzw. *Non-Copyleft*. Grundsätzlich wird Software, welche öffentlich zugänglich ist, bereits als Open Source bezeichnet. Die für diese Arbeit

[7] Siehe: https://github.com/pmndrs/react-xr
[8] Siehe: https://github.com/pmndrs/react-three-fiber
[9] Siehe: https://www.soft8soft.com/verge3d/
[10] Siehe: https://en.wikipedia.org/wiki/Verge3D

erstellte Anwendung soll gemäss der Aufgabenstellung mindestens über eine Permissive License[11] verfügen. Dies bedeutet, dass auch die verwendeten Technologien geringstenfalls diese Vorgabe einhalten müssen. Somit sind folgende Lizenzen für uns geeignet:

- BSD
- MIT
- Apache
- PD
- CC0
- ISC

Ausschlussverfahren der Technologien

Nach den den gegebenen Kriterien müssen bereits einige der zuvor erwähnten Technologien für das Projekt ausgeschlossen werden. Somit kommen

- alle Game Engines (Lizenzierung und/oder nicht Webbasiert),
- Model Viewer (keine VR-Unterstützung),
- Blend4Web (Lizenzierung),
- Verge3D (Lizenzierung)

nicht mehr in Frage. Weiter betrachtet werden

- A-Frame,
- Babylon.js,
- p5.xr (beschränkt durch Lizenzierung),
- React-XR,
- Three.js

Technologien zur Gestaltung von 3D-Modellen wie Blender, welche zwar die Open Source Anforderung erfüllen, aber nicht webbasiert sind, können auch verwendet werden.

[11] Siehe: https://en.wikipedia.org/wiki/Permissive_software_license

3.1.3 Kriterienkatalog

In diesem Kapitel werden die Kriterien aufgelistet, nach der eine VR-Technologie beurteilt wird. Die einzelnen Kriterien werden gewichtet, sodass die für uns relevanteren Kriterien schnell ersichtlich werden. Die drei Technologien, welche das beste Resultat erzielen, werden mit einem Prototyp getestet und daraufhin erneut Bewertet. Ausschlusskriterien werden nicht mehr aufgeführt, da diese schon abgehandelt wurden (Kapitel 3.1.2).
Die Bewertungen werden jeweils in 4 Stufen unterteilt:

- **0**: schlecht/nicht unterstützt
- **1**: kaum unterstützt/erfüllt
- **2**: wird unterstützt/erfüllt
- **3**: gut unterstützt/erfüllt

K1 Unabhängigkeit

Die Realisierung einer Software sollte so unabhängig wie möglich sein. Eigene Anpassungen/Ideen so einfach wie möglich umsetzbar. Man hat von der Technologie wenige Einschränkungen.
Bewertung:

- Die Technologie ist auf bestimmte Funktionen spezialisiert = nicht so frei (eher schlecht)

K2 Entwicklung

Um ein Objekt zu realisieren, werden wenige Codezeilen benötigt. Der Vergleich wird aus den Prototypen gezogen. Ein strukturiertes Arbeiten soll möglich sein, auch bei grösseren Projekten.
Bewertung:

- Ist die Technologie gut geeignet, um ein grösseres Projekt zu realisieren?
- Wird Typescript unterstützt?
- Ist eine brauchbare Dokumentation vorhanden?

K3 Browser

Die Technologie soll möglichst von allen bekannten Browsern unterstützt werden. Die WebXR Standards sollten unterstützt werden.
Bewertung:

- Chrome, Mozilla Firefox, Edge, Safari werden unterstützt

K4 Community

Wie gut die öffentliche Wahrnehmung und wie verbreitet die Technologie ist.
Bewertung:

- Downloadzahlen/Verbreitung werden relativ zu den anderen Technologien verglichen
- Ist die Technologie aktuell? (Häufige Releases?)
- Sind Lösungen auf Stackoverflow oder in anderen Foren zu finden?

K5 Featureset

Wie vielfältig fallen zusätzliche Features aus, die von der Technologie angeboten werden?
Bewertung:

- Können Texturen eingebettet werden?
- Können bestehende 3D-Modelle importiert werden?
- Sind Animationen umsetzbar?
- Wie gut unterstützt das Framework bei der Erstellung eines GUIs? Wichtig dabei ist allerdings, dass das GUI innerhalb der 3D-Welt platziert werden kann. Nur so kann man über VR mit dem GUI interagieren.

K6 Performance

Da viele VR-Geräte nicht über die beste und neuste Hardware verfügen, sollte die Technologie performant sein. Bemerkt man durch ihre Verwendung Einschränkungen?
Bewertung:

- Findet man zu der Technologie Hinweise im Internet, dass Performanceeinbussen in kauf zu nehmen sind?
- Gib es Lösungen um das VR-Erlebnis mit dieser Technologie performanter zu gestalten?

K7 Testbarkeit

Bietet das Framework irgendeine Möglichkeit zur Verifikation und Validation des Source Codes?
Bewertung:

- Stellt das Framework eigene Lösungen für Testing zur Verfügung?
- Sehen die 3D-Szenen nach einer Änderung in der Codebase so aus, wie sie sollten?

Vergabe der Gewichtung

Tabelle 3.1: Kriterienkatalog: Übersicht & Gewichtung

Kriterium	Kurzbeschrieb	Gewicht
K1 Unabhängigkeit	Die Realisierung einer Software sollte so unabhängig wie möglich sein.	1
K2 Entwicklung	Um ein Objekt zu realisieren werden wenige Codezeilen benötigt.	2
K3 Browser	Die Technologie soll möglichst von allen bekannten Browsern unterstützt werden.	2
K4 Community	Wie gut die öffentliche Wahrnehmung und wie verbreitet (Downloadzahl, Releases) die Technologie ist.	3
K5 Featureset	Vielfalt der angebotenen Features der Technologie	3
K6 Performance	Performance der Technologie	1
K7 Testbarkeit	Unterstützung durch das Framework bezüglich Testbarkeit des geschriebenen Codes	1

Da in dieser Arbeit ein grösserer Prototyp geplant wird, der unter Umständen auch noch nach dieser Arbeit weiterentwickelt wird, wäre es von Vorteil, dass das gewählte Framework die Anzahl Zeilen im Source Code nicht in die Höhe treibt (K2 Entwicklung). Alle bekannten Browser unterstützen WebXR. Damit nicht mit einer Technologie gearbeitet wird, welche eigene oder unbekannte Standards verwendet, steht hier (K3 Browser) die Gewichtung ebenfalls auf 2. Um nicht auf einmal auf die Grenzen der gewählten Technologie zu stossen und ein geplantes Feature aufgrund der eingeschränkten Technologie nicht realisiert werden kann, soll ein umfängliches Featureset vorausgesetzt werden. Es ist ebenfalls wichtig, von den Erfahrungen anderer Entwicklern zu lernen. Aus diesem Grund haben die Kategorien K4 Community & K5 Featureset die höchste Priorität.

3.1.4 Technologieneinschätzungen

Technologieneinschätzung zu K1 Unabhängigkeit

Tabelle 3.2: Technologieneinschätzung: K1 Unabhängigkeit

Technologie	Einschätzung	Punktzahl
A-Frame	A-Frame spezialisiert sich nicht auf gewisse Funktionen, es ist darauf ausgelegt einfach Komponenten zu erstellen und rendern.	3
Babylon.js	Keine Spezialisierung	3
p5.xr	*"for artists, designers, educators, beginners, and anyone else"* - p5.xr spezialisiert sich auf Künstler	2
React-XR	Keine Spezialisierung	3
Three.js	Keine Spezialisierung	3

Technologieneinschätzung zu K2 Entwicklung

Tabelle 3.3: Technologieneinschätzung: K2 Entwicklung

Technologie	Einschätzung	Punktzahl
A-Frame	A-Frame an sich ist sehr schlank gehalten. Entwickelt wird hauptsächlich mit HTML Code. Für TypeScript gibt es gewisse Toolkits, welche jedoch kaum Aufmerksamkeiten bekommen. Auf der Homepage von A-Frame steht eine sehr detaillierte Dokumentation zur Verfügung.	3
Babylon.js	Babylon.js ist ähnlich schlank wie A-Frame. TypeScript wird standardmässig unterstützt, da es selbst auch damit geschrieben wurde und in der Dokumentation wird beschrieben, wie man damit arbeitet. Die Dokumentation auf der Homepage ist sehr umfangreich und gut strukturiert	3
p5.xr	p5.xr ist noch in der Aufbauphase und somit noch nicht sehr gross. Zu p5.js stehen TypeScript Packages zur Verfügung, jedoch nicht für p5.xr. Dokumentiert ist vor allem die p5.js Library, zu p5.xr steht lediglich eine kleine Demo bereit.	2
React-XR	React-XR erlaubt die Verwendung von Typescript bzw. TSX. Dokumentiert ist nur sehr wenig, und auch nur im GitHub Repository.	1
Three.js	Für Three.js wird eine TypeScript Boilerplate angeboten, welche immer mehr Aufmerksamkeit in der näheren Vergangenheit erhielt. Eine sehr detaillierte Dokumentation dazu findet man auf ihrer Homepage.	3

Technologieneinschätzung zu K3 Browser

Tabelle 3.4: Technologieneinschätzung: K3 Browser

Technologie	Einschätzung	Punktzahl
A-Frame	Alle ausgewählten Technologien unterstützen WebXR. Edge und Chrome unterstützen WebXR standardmässig, bei Firefox und Safari kann man dies bei Bedarf einschalten	3
Babylon.js		3
p5.xr		3
React-XR		3
Three.js		3

Technologieneinschätzung zu K4 Community

Tabelle 3.5: Technologieneinschätzung: K4 Community

Technologie	Einschätzung	Punktzahl
A-Frame	A-Frame benutzt Stackoverflow als offizielle Q&A-Plattform und man kann sich zu bereits über 2'000 beantworteten Fragen informieren. Der Source Code wird regelmässig angepasst, sodass man auf dem GitHub Repository zum Teil Commits von vor wenigen Tagen sieht. Das Einzige, was nicht aktuell scheint, dass immer noch von WebVR die Rede ist. Die wöchentliche Downloadzahl von A-Frame auf npmjs liegt bei über 6'000.	2
Babylon.js	Auf Stackoverflow findet man einige 100 Fragen zu Babylon.js, dafür existieren umso mehr Fragen im eigenen Forum der Homepage. Der Source Code wird regelmässig angepasst, so dass man auf dem GitHub Repository zum Teil Commits von vor wenigen Tagen sieht. Die wöchentliche Downloadzahl von Babylon.js auf npmjs liegt bei knapp 10'000.	2
p5.xr	Auf Stackoverflow findet man so gut wie nichts zu p5.xr resp. p5.js und ein eigenes Forum wird noch nicht angeboten. Der Source Code auf GitHub wird selten angepasst. Die wöchentliche Downloadzahl von p5.js auf npmjs liegt bei um die 6'000, wobei p5.xr so gut wie nicht gebraucht wird.	1
React-XR	Zu React-XR gibt es fast keine Fragen auf Stackoverflow. Bis auf das GitHub Repository findet man allgemein keine Informationen dazu und dort kann man erkennen, dass seit über einem Jahr keine Änderungen mehr vorgenommen wurden. Die wöchentliche Downloadzahl von React-XR auf npmjs liegt im zweistelligen Bereich.	0
Three.js	Three.js besitzt ein eigenes Forum, aber auch auf Stackoverflow kann man sich an über 14'000 beantworteten Fragen belehren. Auf Github ist anhand der Commits erkennbar, dass hier regelmäsig Updates geliefert werden. Die wöchentliche Downloadzahl von Three.js auf npmjs liegt ungefähr bei 400'000.	3

Technologieneinschätzung zu K5 Featureset

Tabelle 3.6: Technologieneinschätzung: K5 Featureset

Technologie	Einschätzung	Punktzahl
A-Frame	Sieht man in der offiziellen Dokumentation von A-Frame nach, findet sich eine ganze Reihe an zusätzlich unterstützten Features. Texturen lassen sich ohne Weiteres einbetten. Auch Animationen sind umsetzbar. Die umfangreiche API bietet die Möglichkeit die Animationen nach Belieben anzupassen. Externe 3D-Modelle können auch importiert werden. Untersützte Datenformate sind glTF und obj. Beim GUI tut sich A-Frame allerdings schwer. In der Dokumentation findet sich keine Hilfe zur Erstellung von GUIs. Es existiert jedoch eine third-party Library, die den Entwickler bei der Erstellung von GUIs innerhalb der 3D-Welt unterstützt.	2
Babylon.js	Babylon.js bietet ähnlich wie A-Frame eine Menge zusätzlicher Features an. Externe 3D-Modelle können importiert werden. Unterstützt werden glTF, obj und stl Dateien. Modelle können auch mit ihren Texturen importiert werden. Animationen sind vielfältig umsetzbar. Unterstützte Varianten sind Rotationen, Translationen und Transformationen von 3D-Objekten. Babylon.js bietet ausserdem von Haus aus eine umfangreiche Lösung für GUI-Implementationen an. Es kann zwischen einem GUI für 2D-Oberflächen und GUIs in 3D-Umgebungen ausgewählt werden. Babylon.js bietet auch einen umfangreichen Playground inklusive Viewer an, über den neue Features ausprobiert werden können.	3
p5.xr	Auch p5.xr bietet einige Features an. Es können obj und stl Modelle importiert werden. Auf 3D-Objekten können Texuren platziert werden. In der offiziellen Dokumentation konnte keine Unterstützung für 3D-GUIs gefunden werden. Allerdings existieren auch hier third-party Libraries, die diese Funktionalität anbieten. Animationen werden zwar angeboten, jedoch fällt der Umfang weniger vielfältig aus, wie bei A-Frame oder Babylon.js.	2

React-XR	Reactxr unterstützt Interaktionen mit 3D-GUIs, in der Dokumentation konnten aber nur Buttons gefunden werden, die in der 3D-Welt platziert werden können. Allgemein basiert Reactxr auf react-three-fiber, welches als React Renderer für Three.js alle Funktionen von Three.js automatisch auch unterstützt.	2
Three.js	Three.js bietet alle wichtigen Funktionen an. Animationen können erstellt werden und über verschiedene Modi angepasst werden. Charaktere können zum Leben erweckt werden, indem Skelettmodelle hinterlegt werden. Basisfunktionen wie Texturen einfügen und externe 3D-Modelle laden stellen auch kein Problem dar. Importiert werden können glTF, obj, fbx und collada Dateiformate. 3D-GUIs werden nicht out of the box unterstützt. Dazu müssen externe Bibliotheken herangezogen werden.	2

Technologieneinschätzung zu K6 Performance

Tabelle 3.7: Technologieneinschätzung: K6 Performance

Technologie	Einschätzung	Punktzahl
A-Frame	A-Fame ist sich der der Wichtigkeit einer performanter VR-Applikation bewusst und gibt in seiner Dokumentation selber gleich mehrere Tipps und Beispiele, um den Code möglichst performant zu halten. Mit der Technologiespeziefischen Lösung eines States kann man seinen Quellcode optimieren.	3
Babylon.js	In ihrer Dokumentation erläutert Babylon.js dem Entwickler bekannte Gründe, die für eine schlechtere Performance. Ebenfalls werden einem in einem zusätzlichen Kapitel verschiedene Optimierungsmöglichkeiten anhand von Beispielen erklärt.	3
p5.xr	Performance kommt auf der p5.xr-Website nicht zur Sprache.	1
React-XR	React-XR verfügt über keine detaillierte Dokumentation und somit wird Performance auch niergends erwähnt	1
Three.js	Three.js dokumentiert nicht viel zum Thema Performance. Jedoch verfügt ihr WebGL Renderer zum Beispiel über eine Powerpreference-Einstellung, um die Performance etwas zu steuern. Des weiteren wurde in einem Vergleich[12] mit Babylon.js, ausser dass Three.js weniger Memory verwendet, keine relevanten Unterschiede festgestellt.	2

Bezüglich Performance findet man zu keinem dieser Technologien sehr viel Informationen. Desshalb ist es schwierig zu erkennen, ob ein Framework performanter ist als ein anderes. Grundsätzlich macht es jedoch einen besseren Eindruck, wenn in einer Dokumentation die Rede davon ist und man dabei unterstützt wird, als wenn das Thema nie erwähnt wird. Das vermittelt das Gefühl, dass sich Gedanken darüber gemacht werden und in Zukunft vielleicht weitere Optimierungen folgen.

Technologieneinschätzung zu K7 Testbarkeit

Tabelle 3.8: Technologieneinschätzung: K7 Testbarkeit

Technologie	Einschätzung	Punktzahl
A-Frame	A-Frame bietet nur eine Testingmöglichkeit für Performance an. Damit kann die aktuelle FPS Anzahl ausgelesen werden. Mit dem A-Frame Inspector kann man visuell, jedoch nicht automatisiert, eine gewisse Art Testing durchführen.	1
Babylon.js	Babylon.js bietet die Möglichkeit für Regressiontests an. Dazu werden Screenshots von den 3D-Szenen aufgenommen und bei Änderungen mit den neuen Screenshots abgeglichen. Babylon.js führt diese Tests in der Entwicklung der Babylon.js Library aus, ohne eine Unterstützung für Anwender ihrer Library anzubieten. Auch Babylon.js bietet einen Visual Inspector an und erlaubt es die FPS Anzahl auszulesen, um die Performance zu überprüfen.	1
p5.xr	Das zugrundeliegende p5.js hat eine ausführliche Dokumentation, die das Aufsetzen der Testumgebung beschreibt mit simplen Beispielen von Unit Tests. Es konnte allerdings nichts konkretes dazu gefunden werden, wie sich 3D-Szenen testen lassen.	1
React-XR	Für die zugrundeliegende Library React Three Fiber lassen sich Tests schreiben, die die 3D-Szene testen. Dazu wird auch eine knappe Dokumentation zur Verfügung gestellt. Dort wird zum Beispiel überprüft, ob eine Szene die korrekte Anzahl an Meshes hat. Auch Interaktionen mit den Meshes können getestet werden.	2
Three.js	Three.js zeigt in ihrer Dokumentation eine kleine und simple Variante, wie man automatisiertes Testing durchführen könnte. Jedoch sind diese Tests bescheiden, indem man zum Beispiel nur schaut, ob ein Objekt undefined ist oder nicht, oder einen Wert prüft. Dies kann mit allen Technologien gemacht werden und ist unabhängig von Three.js.	1

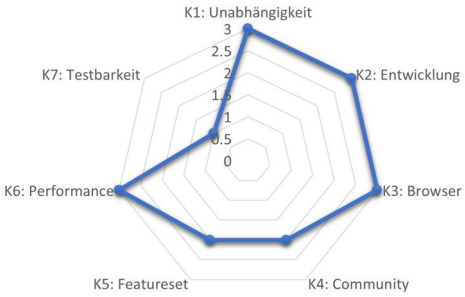
3.1.5 Auswertung der Technologieneinschätzung

Die Tabelle 3.9 fasst die Punktzahlen zusammen und weisen den Kategorien noch ihre Gewichtung zu.

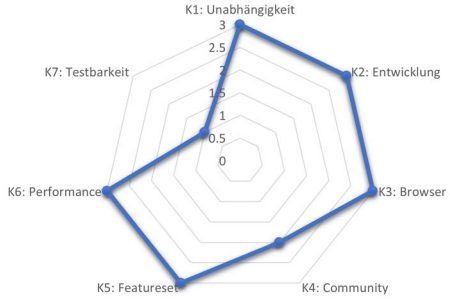
Tabelle 3.9: Auswertung der Technologieneinschätzung

Kriterium (Gewichtung)	A-Frame	Babylon.js	p5.xr	React-XR	Three.js
K1 (1)	3	3	2	3	3
K2 (2)	3 = 6	3 = 6	2 = 4	1 = 2	3 = 6
K3 (1)	3	3	3	3	3
K4 (3)	2 = 6	2 = 6	1 = 3	0	3 = 9
K5 (1)	2 = 6	3 = 9	2 = 6	2 = 6	2 = 6
K6 (1)	3	3	1	1	2
K7 (1)	1	1	1	2	1
Total:	**28**	**31**	**20**	**17**	**30**

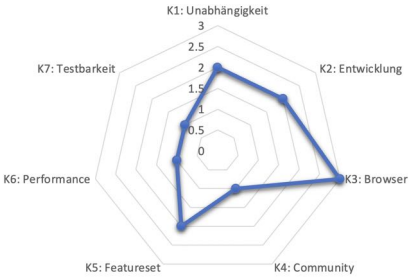
Zur besseren Übersichtlichkeit werden die Resultate noch in Form von Netzdiagrammen abgebildet. In den Grafiken wurde allerdings auf eine Gewichtung verzichtet.



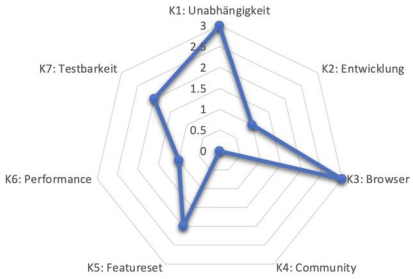
(a) A-Frame Netzdiagramm



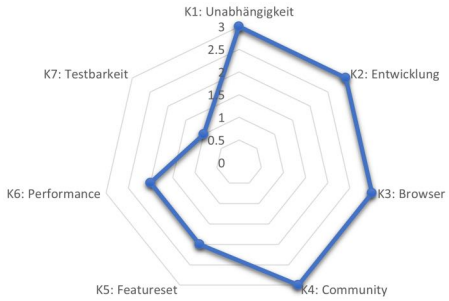
(b) Babylon.js Netzdiagramm



(c) p5.xr Netzdiagramm



(d) React-XR Netzdiagramm



(e) Three.js Netzdiagramm

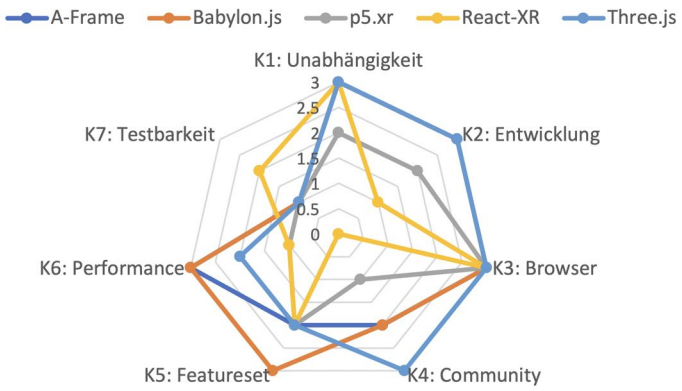


Abbildung 3.2: Netzdiagramm Technologieauswahl

3.1.6 Fazit

In den Netzdiagrammen oben ist zu erkennen, dass sich Three.js, A-Frame und Babylon.js in etwa auf dem gleichen Level befinden. Dabei kann jede Technolgie seine eigenen Stärken auspielen. Während Three.js mit der grössten Community weit verbreitet ist, hat Babylon.js das umfangreichere Featureset. A-Frame andererseits wurde speziell für VR-Anwendungen entwickelt. React-XR und p5.xr waren im Vergleich entweder gleichauf mit den anderen Technologien oder haben schlechter abgeschnitten. Einzig was das Testing angeht, konnte React-XR mit React Three Fiber mehr überzeugen. Im Abschnitt 3.3 wird für die Kandidaten Three.js, A-Frame und Babylon.js jeweils ein Prototyp realisiert, um eine endgültige Entscheidung zu treffen, mit welcher Technologie die Software City umgesetzt werden soll.

3.2 Planung des Anwendungsfalls

Eine allgemeine Beschreibung der Software Visualisierung durch Software Cities befindet sich im Kapitel 2.4. In diesem Abschnitt geht es nun darum, die Implementation grob zu planen und allfällige Probleme zu erkennen.

3.2.1 Das Software City Layout

Es bieten sich verschiedene Strategien an, die Stadt aufzubauen. Sieht man die Software als eine Hierarchie von Komponenten an (Abbildung 3.3), können beliebige Darstellungen von Tree Datenstrukturen als Grundlage für das Stadtlayout dienen.

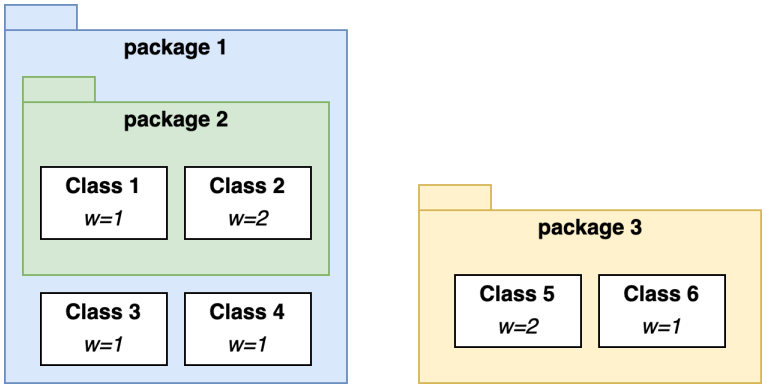


Abbildung 3.3: Hierarchischer Aufbau von Softwarekomponenten

Eine Variante könnte eine Treemap sein. Da es aber nicht der Fokus der Arbeit sein soll, einen speziellen Algorithmus für das Stadtlayout zu erarbeiten, werden zwei Ideen von bestehenden Software Cities genauer angeschaut. Die Implementation soll eine einfache Austauschbarkeit dieses Algorithmus ermöglichen. Folgender Baum bildet die Grundlage für unsere Varianten und bezieht sich auf die Abbildung 3.3.

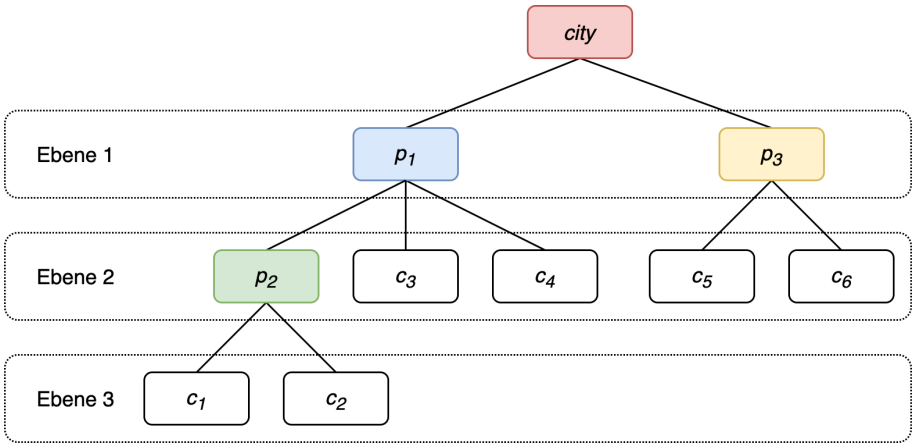


Abbildung 3.4: Baumstruktur der Softwarekomponenten

Variante 1: Squarified Treemaps

Die Squarified Treemap [Bruls et al., 2000] hat das Ziel die Kacheln (Grundstücke eines Gebäudes/einer Klasse) möglichst quadratisch zu halten. Ein solcher Algorithmus bietet sich deshalb gut für das Layout einer Software City an. Ausgangslage ist das Beispiel in der Abbildung 3.3. Jede Klasse soll nun als ein Gebäude dargestellt werden. Da ein Gebäude, je nachdem wie die Parameterisierung (Kapitel 2.4.3) erfolgt, unterschiedliche Grundstückflächen besitzen kann, wird jedem Gebäude ein Gewicht zugeteilt. Dieses wurde in der Abbildung 3.3 beispielhaft über den Buchstaben w angegeben. Das Gesamtgewicht der Stadt wäre die Summe aller Gebäude, also 8.

Im folgenden Algorithmus wird die Baumdatenstruktur (Abbildung 3.4) nun rekursiv Ebene um Ebene traversiert. Der Start bildet die Ebene 1:

1. Layout der Stadtviertel (Packages) der aktuellen Ebene erstellen nach Squarified Treemap Algorithmus [Bruls et al., 2000]

2. Layout der Strassen als Abgrenzung der Stadtviertel erstellen. Zum Beispiel können die Strassen jeweils unten und links des Grundstücks angebracht werden, falls sich dort nicht schon eine Strasse vom Parent befindet. Tiefer verschachtelte Strassen sind dabei schmaler.

3. Resultate in Datenstruktur speichern und Algorithmus für die Childs wiederholen

Aussehen könnte das in unserem Beispiel wie in der Abbildung 3.5. dargestellt. Zur Übersichtlichkeit wurden die Strassen nicht dargestellt.

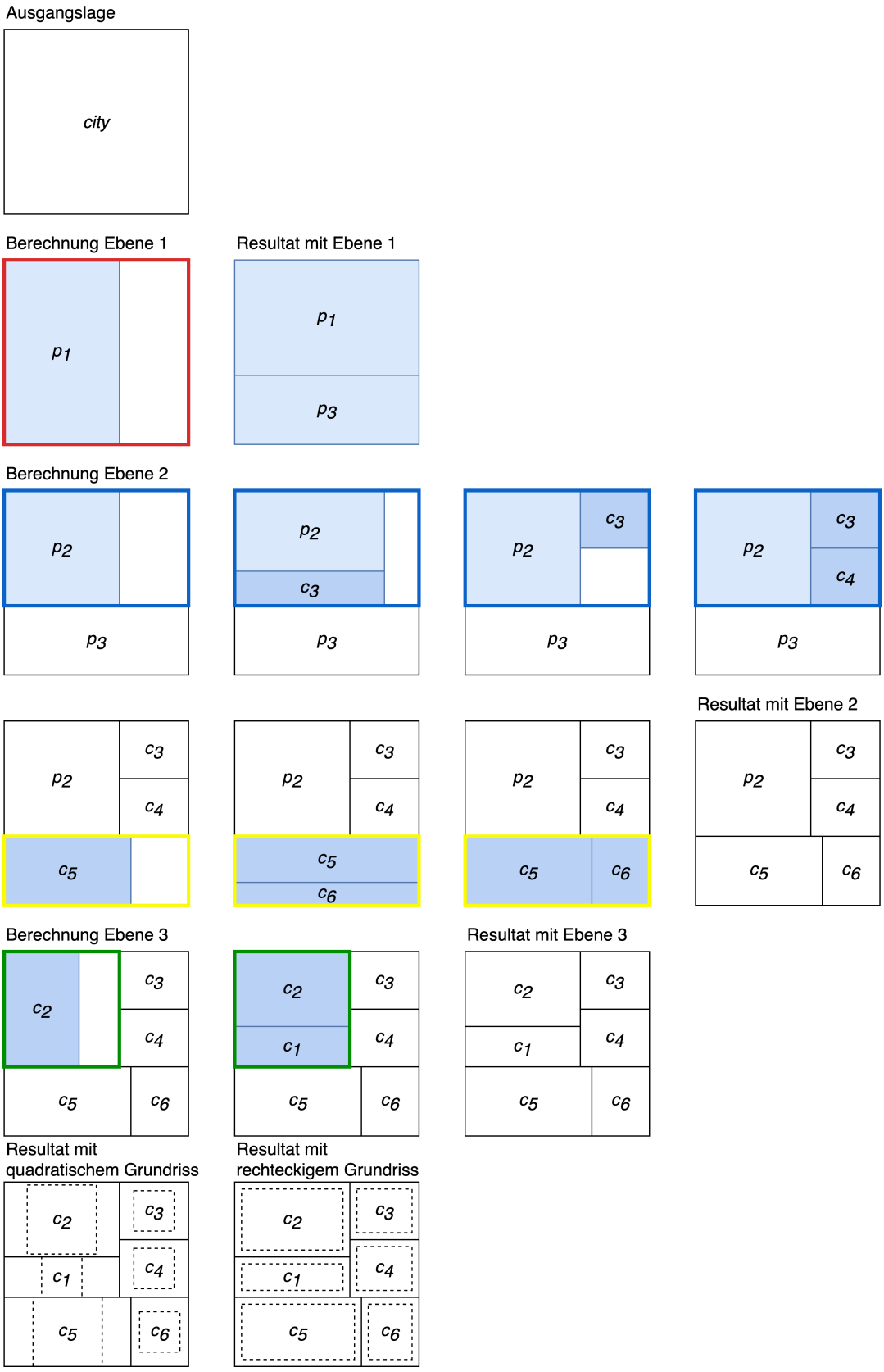


Abbildung 3.5: Stadtlayout Algorithmus Variante Squarified Treemaps

Klassen mit gleichem Gewicht w werden die gleiche Fläche zugewiesen. Gebäudegrundrisse können entweder quadratisch oder rechteckig gewählt werden. Zu beachten ist, dass bei quadratischen Gebäudegrundrissen die effektiven Gebäude je nach Seitenverhältnis des Grundstücks extrem schmal ausfallen können. Heuristiken sagen aus, dass der Squarified Treemap Algorithmus am besten funktioniert (erzeugt Flächen mit Seitenverhältnissen nahe bei 1), wenn man die Elemente vorher nach Gewicht absteigend sortiert. [Bruls et al., 2000]

Die Strassen in der Abbildung 3.6 trennen die Viertel besser ab. Je weiter oben sich die Strasse in der Hierarchie befindet, desto breiter wird sie dargestellt.

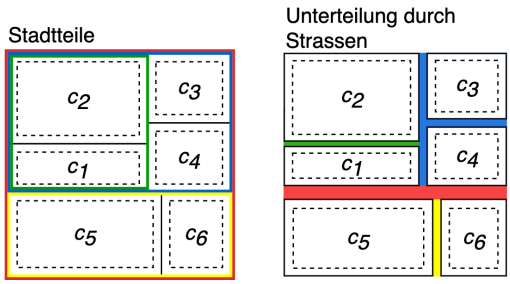


Abbildung 3.6: Variante Squarified Treemaps inklusive Strassen

Variante 2: Street Layout

In der Street Layout Variante ist jedes Package eine Strasse und die Elemente dieses Packages werden am Strassenrand aufgestellt. Diese Elemente können Klassen oder weitere Packages sein. Jedes Element muss seine Dimensionen (breite und länge) dem Parent bekannt geben, damit sich der Parent wiederum seine Strasse zusammenbauen kann. Ein Vorschlag dieses Algorithmus könnte wie folgt aussehen. Das Layout von jedem Package wird für sich erstmal für sich alleine berechnet. Anders als im vorherigen Algorithmus wird hier bei den Leafs angefangen.

Dieser Algorithmus wird für jedes Package durchgeführt. Das Layout soll so rekursiv zusammengebaut werden:

1. Die Childs aufteilen in zwei Collections `leftChilds` und `rightChilds`. Fällt ein Child in die Collection `leftChilds` muss das Child um 90° rotiert werden. In der Collection `rightChilds` muss es um -90° rotiert werden. Rotiert werden können die Punkte mittels einer Rotationsmatrix.

2. Dimensionen des Viertels berechnen:

   - `streetLength` = Gesamtbreite der breiteren Seite (`leftChilds` oder `rightChilds`)
   - `streetWidth` = Je tiefer verschachtelt, umso schmaler
   - `quarterLength` = `streetLength`
   - `quarterWidth` = Breite `leftChilds` + `streetWidth` + Breite `rightChilds`

3. Relative Koordinaten der Childs berechnen, wobei der Bezugspunkt der Mittelpunkt des Viertels darstellt.

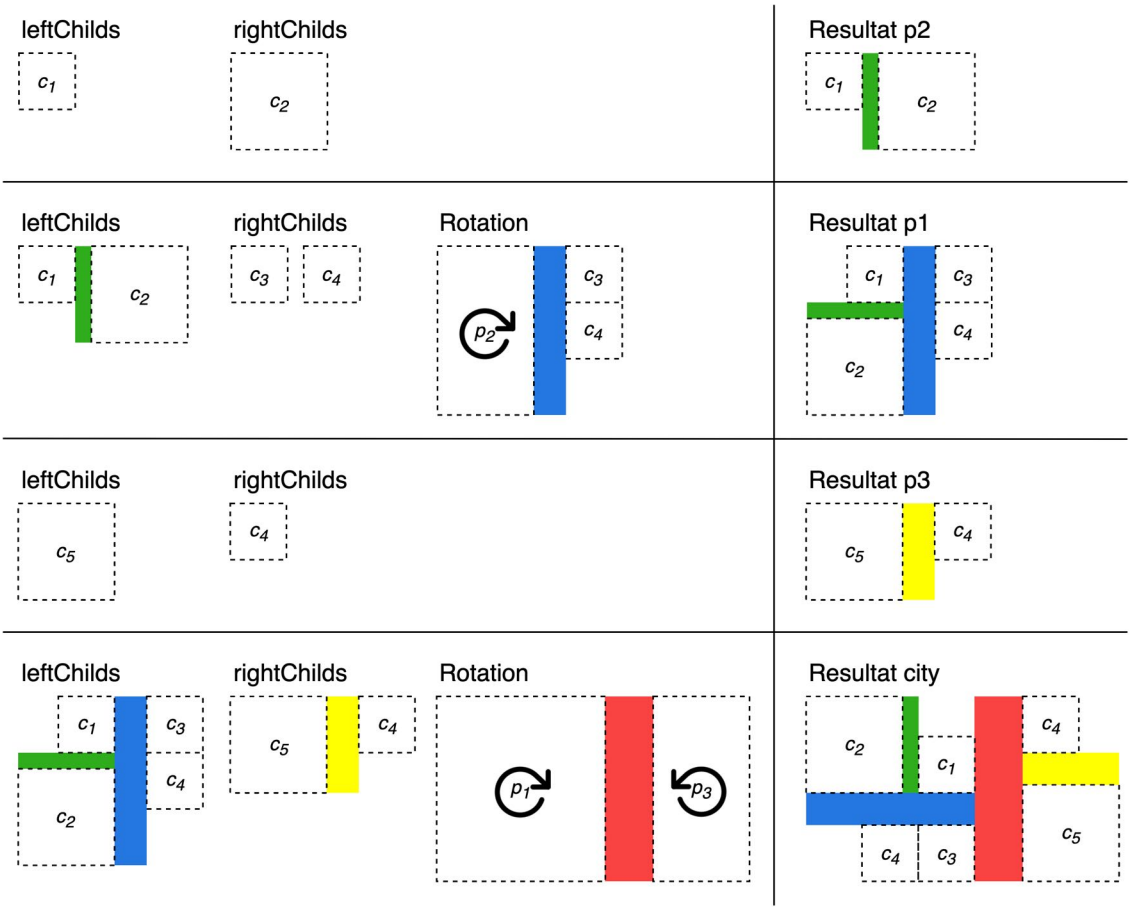


Abbildung 3.7: Stadtlayout Algorithmus Variante Street Layout

Als nächsten Schritt wäre es denkbar, dass man die aufgebaute Struktur nochmals traversiert und für jede Komponente seine absoluten Koordinaten abspeichert. So könnte das Rendering im nächsten Schritt vereinfacht werden. Auch könnte man zwischen den Gebäuden eine Art Padding einfügen. In der Abbildung 3.8 wäre zum Beispiel die rote Strasse die Hauptstrasse der Stadt. Die direkten Kinder, p_1 und p_3, zweigen direkt von der Hauptstrasse ab. Die direkte Kinder von p_1 wären p_2, c_3 und c_4 und zweigen direkt von p_1 ab.

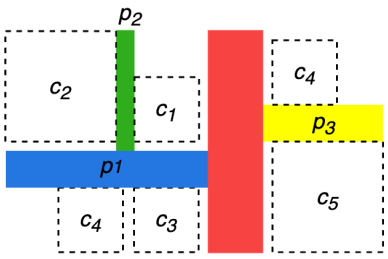


Abbildung 3.8: Variante Street Layout inklusive Padding

3.2.2 Interaktion mit dem Nutzer

Der Nutzer sollte die Stadt von allen Seiten betrachten können. Dazu gehört, dass man die Stadt bei Bedarf rotieren und vergrössern kann. Plausibel wäre auch die Vorstellung, dass der Nutzer gewisse Einstellungen vornehmen kann, wie zum Beispiel die verschiedenen Varianten wie eine Stadt gerendert wird (Abschnitt 3.2.1), oder die Bedeutung der einzelnen Dimensionen der Gebäude 2.4.3. Auch stylistische Änderungen könnte man dem Nutzer überlassen. So könnte er das Styling der Gebäude durch eine Auswahl an vormodellierten 3D-Modellen anpassen. Weiter könnte er die Tageszeit der virtuellen Welt verändern, so dass er die Stadt in einem angenehmen Abendrot betrachten kann.

Selektiert der Benutzer ein Gebäude, könnte in einem Fenster der Source Code der zugehörigen Klasse angezeigt werden. Eine weitere Funktion wäre die Möglichkeit, die Vererbungshierarchie einer Klasse in der Stadt anzuzeigen, indem alles weiss dargestellt wird, ausser die Gebäude, die Teil der Vererbungshierarchie sind.

Als Quelle für die Projektdaten wird GitHub verwendet. Durch die Verwendung von GitHub wäre es möglich die Historie der Stadt anzuzeigen. So könnte der Nutzer in einer Timeline eine Zeit auswählen und die Veränderungen der Stadt über die Zeit beobachten. Das setzt allerdings ein konsistentes Software City Layout voraus, so dass minimale Änderungen nicht zu einer komplett neuen Stadt führen.

3.2.3 Vorgehen zur Generierung der 3D Stadt

Die Erstellung der Stadt beinhaltet verschiedene einzelne Schritte. Grob können diese wie folgt unterteilt werden:

1. Datenabfrage
2. Code Parsing
3. Generierung vom Stadt Layout
4. Renderer

Nachdem der Nutzer die Angaben zu seinem Projekt gemacht hat, beginnt die Applikation mit der Abfrage des Source Codes vom Projekt. Danach übernimmt der Parser und erstellt aus den Source Code Strings die Datenstruktur, mit der Schritt drei das Layout der Stadt erstellen kann. Am Schluss wird die Stadt für den Nutzer auf das Ausgabegerät gerendert.

Jeder dieser Schritte bildet einen State ab, in der sich die Applikation zur gegebenen Zeit befinden kann. Unter Einbezug der restlichen States ergibt sich somit eine State Machine nach Abbildung 5.12.

1. Datenabfrage

Damit der Source Code im Projekt des Benutzers analysiert werden kann, muss das System irgendwie darauf zugreifen können. Der Benutzer hat dabei eine Brille aufgesetzt, und verwendet einen in der Brille eingebauten Web-Browser. Wie bereits oben beschrieben, ist eine mögliche Lösung ein GitHub Repository abzufragen. Der Nutzer gibt die Informationen zu seinem gewünschten GitHub Repository an und über die GitHub REST API kann dann die Abfrage des Projektes gestartet werden. Das Resultat ist eine Baumdatenstruktur des Projektes, wobei die Leafs einen String enthalten, der den Inhalt einer Datei repräsentiert.

2. Code Parsing

Durch das Parsing können die Code Metriken des Source Code ausgelesen und in eine weiterbearbeitbare Form gebracht werden. Für die Umsetzung in diesem Projekt ergeben sich zwei Möglichkeiten: Entweder es wird eine bereits existierende Library verwendet, oder eine rudimentäre, minimale Implementation eines Parsers wird selber umgesetzt.

Da eine Parserimplementation von Grund auf sehr komplex werden kann, wäre der erste Ansatz die logische Wahl. Durch den Ansatz der Single Page Application und der Tatsache, dass auf ein Backend verzichtet wird, müsste die Library eine JavaScript Library sein, damit sie im Browser ausgeführt werden kann. Solche bestehenden Parserimplementationen spezifisch für Java gibt es nicht sehr viele. Die einzige gefundene Library, die diese Anforderungen erfüllen würde, wäre das Node Package java-parser[13]. Dieses kann einen String in eine Baumdatenstruktur parsen, die später traversiert werden kann. Die Traversierung geschieht unter der Anwendung des Visitor Patterns. Im Falle dieser Applikation müsste man den Baum in eine Datenstruktur bringen, die von den späteren Komponenten weiterverarbeitet werden kann.

Der Ansatz, den Parser selbst zu implementieren, bleibt als Fallback Szenario offen. Um dafür nicht zu viel Zeit zu verlieren, würde er mehr rudimentär umgesetzt, sodass mindestens die wichtigsten Anwendungsfälle funktionieren. Die Eigenimplementation würde das Parsing in zwei Teile unterteilen: den Lexer (auch Scanner oder Tokenizer) und den effektiven Parser. Der Lexer wäre dafür zuständig einzelne Tokens, wie Wörter oder Syntaxelemente, aus dem Source Code zu generieren. Der Parser nimmt die Tokens entgegen und erstellt den dazugehörigen minimalen Abstract Syntax Tree.

In beiden Fällen soll der Parser so umgesetzt werden, dass er einfach austauschbar ist. Fortsetzende Arbeiten sollen in der Lage sein den Parser ohne grössere Aufwände durch eine Neuimplementation zu ersetzen.

3. Generierung vom Stadt Layout

In diesem Schritt gilt es, eine der im Abschnitt 3.2.1 beschriebenen Layout Varianten umzusetzen. Im Optimalfall würde man es dem Nutzer überlassen, eine Variante auszuwählen. Am Ende soll es keinen Unterschied machen, welches Layout gewählt wurde, da die Struktur des Outputs in diesem Schritt immer dieselbe ist.

Damit das Layout erzeugt werden kann, wird eine hierarchische Struktur der Codemetriken zu allen Klassen, Interfaces und Enums des Projektes benötigt. Die im vorherigen Schritt erzeugte Datenstruktur beinhaltet diese Informationen.

Die resultierende hierarchische Datenstruktur beinhaltet die 3 möglichen Komponenten der Stadt: Stadtviertel, Gebäude und Strassen. Ein Stadtviertel kann hierbei mehrere Subkomponenten beinhalten. Jede Komponente speichert sich unter anderem seine absoluten Koordinaten und den Namen. Gebäude speichern sich zusätzlich noch die Höhe zusammen mit den ausgelesenen Codemetriken. Eine vollständige Liste findet sich in der Softwarearchitektur im Kapitel 5.1.4. Inspiriert wurde dieser Aufbau durch das Composite Pattern aus dem Design Patterns Buch *Gang of Four*[14].

[13] Siehe https://www.npmjs.com/package/java-parser
[14] Siehe https://en.wikipedia.org/wiki/Design_Patterns

4. Renderer

Der Renderer ist dafür zuständig die Stadt zu zeichnen. Dazu gehören auch Elemente innerhalb der Stadt, wie etwa Bäume, Autos oder Fussgänger.

Um die Stadt zu zeichnen, müssen die übermittelten Daten rekursiv traversiert und jedes Element separat eingezeichnet werden. Bei Bedarf können den Gebäuden und Strassen noch Texturen zugewiesen werden.

3.2.4 Einschränkungen

Bei den unterstützten Programmiersprachen fällt die Wahl auf Java, da wir diese Sprache am längsten kennen. Mit der Strukturierung durch Packages sind auch sinnvolle Stadtlayouts möglich. Die Anwendung an sich würde möglichst simpel gehalten werden, damit der Fokus immernoch auf den VR-Themen bleibt. Das heisst die Implementation wird ohne ein Backend auskommen, in dem zum Beispiel ein Account Management betrieben werden könnte. Beim Rendering der Stadt gilt es eventuell zu beachten, dass grössere Projekte zu rechenaufwändig sein könnten. Der Benutzer soll deshalb darauf hingewiesen werden.

3.2.5 Erweiterungsmöglichkeiten

Diese Applikation wäre auch in einem AR-Setting gut umsetzbar. Dort würde man die Stadt womöglich auf seinem Schreibtisch platzieren und wäre so nicht komplett abgeschottet wie bei der Virtual Reality.

Denkbar wäre auch eine detailliertere Ansicht, in der zum Beispiel Funktionen die einzelnen Gebäude darstellen. So könnte beim reinzoomen auf ein spezifisches Gebäude, das eine Klasse darstellt, das Gebäude ausgeblendet werden und mit kleineren Häusern ersetzt werden, die die Funktionen der Klasse abbilden.

Eine andere Erweiterung wäre die Analyse des dynamischen Verhaltens der Software. So könnte die Gebäudehöhe der Zeit entsprechen, die durchschnittlich in der Klasse verbracht wird.

3.2.6 Namenswahl

Der Name des Prototypen sollte kurz sein, einfach zum Aussprechen und den Anwendungsfall veranschaulichen. Die Wahl fiel deshalb auf **incode**, da man, bei aufgesetztem VR Headset, gewissermassen direkt **in** dem **Code** drin ist, den man analysieren will.

3.3 Prototypische Realisierung

Aus dem Technologievergleich konnte entnommen werden, dass Three.js, Babylon.js und A-Frame die geeignetsten Kandidaten für VR-Anwendungen sind. Um eine entgültige Entscheidung zu treffen, wurde für jedes dieser drei Technologien ein Prototyp erstellt, um die praktische Umsetzung vergleichen zu können. Da der Anwendungsfall eine Software City sein wird, wurde versucht, dessen wichtigsten Elemente in die Prototypen einfliessen zu lassen.
Jeder Prototyp beinhaltet:

- eine Szene mit Strasse und Gebäude
- Texturen
- importierte 3D-Modelle
- UI, um Einstellungen anzupassen
- WebXR-Unterstützung (VR-Modus)

Abbildung 3.9: Prototyp mit Three.js

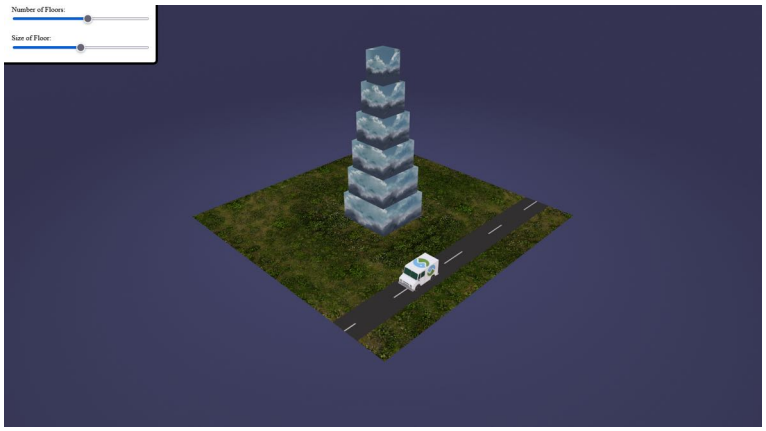


Abbildung 3.10: Prototyp mit Babylon.js

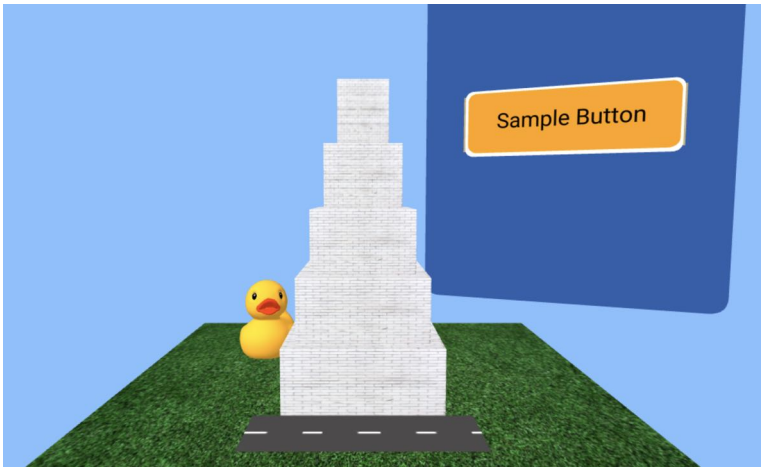


Abbildung 3.11: Prototyp mit A-Frame

Vergleich der Prototypischen Realisierung

Dem jeweiligen Source Code[15] der drei Technologien kann man entnehmen, dass mit bereits wenig Code schon umfangreichere Szenen erstellt werden können. A-Frame kann hier auftrumpfen, indem nur knapp 80 Zeilen Code benötigt wurden, um den Prototypen zu realisieren. Three.js (160) und Babylon.js (200) benötigten hingegen etwas mehr Code. Diesen Unterschied kann man dadurch erklären, dass die Three.js und Babylon.js Anwendung in JavaScript (bzw. TypeScript) geschrieben wurde und A-Frame mit HTML, was tendenziell zu kompakterem Code führt.

Bei allen Technologien konnte wie erwartet problemlos eine Szene mit Objekten erstellt und mit Texturen versehen werden.

Wie man erkennen kann, gibt es in allen Prototypen importierte 3D-Modelle (im glTF-Format), jedoch erkennt man bei der Three.js Variante keine Texturen. Dies könnte mit Sicherheit behoben werden, jedoch wurde bis zu diesem Punkt keine Lösung dafür gefunden.

Alle Prototypen besitzen auch ein kleines UI und können damit die Szene ein wenig anpassen. Jedoch sind die UIs im Three.js und Babylon.js Prototypen in ihrer aktuellen Implementation nicht im VR-Modus benutzbar. In der richtigen Implementation müssten diese also anderst implementiert werden. Für eine Integration im VR-Modus gibt es zumindest für Babylon.js offiziell angebotene Lösungen. Für A-Frame und Three.js müssen externe Libraries bezogen werden.

Im Browser wird bei jedem Prototypen ein *ENTER-VR*-Button sichtbar, womit man in die VR-Ansicht gelangt. Sowohl Chrome als auch Firefox unterstützen die Nutzer hier mit einem Plug-In um eine VR-Brille zu simulieren. Bei Babylon.js und A-Frame gelingt dies ohne Probleme, nur Three.js zeigt beim Eintreten in die VR-Welt nichts mehr an.

3.4 Auswahl der Technologie

Um dieses Kapitel abschliessen zu können, muss nun eine der drei verbliebenen Technologien ausgewählt werden, damit diese Technologie anhand der Software City weiter erprobt werden kann.

[15] Siehe: https://gitlab.ost.ch/sa_webvr/prototypes

3.4.1 Vergleich der Technologien bezüglich dem Anwendungsfall

Um den Anwendungsfall umsetzen zu können, muss die Technologie mindestens folgende Features unterstützen:

- Dynamisches hinzufügen neuer Objekte
- Integration von Texturen
- Interaktionsmöglichkeit für den Benutzer mit Hilfe seiner Controller
- GUI in der 3D-Welt
- Import von externen 3D-Modellen (inklusive Texturen)

3.4.2 Entscheidung

Die Prototypen im Abschnitt 3.3 haben ergeben, dass sich Babylon.js von den drei verbliebenen Technologien wohl am besten eignet. Die Umsetzung war dank einem simplen und intuitiven API gut verständlich und verhielt sich grösstenteils so, wie erwartet. Die Ummsetzung mit A-Frame war durch die Umsetzung mit HTML-Code zwar noch einfacher, jedoch stellten sich dort die interaktiven Teile als umständlich heraus. Das Gebäude konnte nur mit Mühe dynamisch durch neue Blöcke ergänzt werden. Statische Szenen lassen sich mit A-Frame also gut umsetzen, bei Veränderungen muss dann allerdings die Szene mittels DOM-Manipulationen aktualisiert werden, was ungewohnt sein kann.

Der Nutzer der Software City soll innerhalb der virtuellen Welt über ein GUI Einstellungen vornehmen können. Dafür bietet Babylon.js, im Gegensatz zu den anderen Frameworks, bereits eine integrierte Lösung an. Alle drei Technologien bieten eine umfangreiche Dokumentation inklusive Beispielcode an. Die Dokumentation von Babylon.js machte allerdings den Eindruck, etwas beginnerfreundlicher zu sein. Ein interaktiver Playground, über den neuer Code getestet werden kann, konnte in dieser umfangreichen Form nur bei Babylon.js gefunden werden. Umfangreich aus diesem Grund, da er eng an die Dokumentation gekoppelt ist. Ausserdem bietet er beim Tippen eine Autovervollständigung, was automatisch dazu einlädt verschiedene Funktionen auszutesten.

Auch die Tatsache, dass Babylon.js vollständig mit TypeScript kompatibel ist (was A-Frame und Three.js nicht von sich behaupten können) spricht für Babylon.js. Das kann die Einarbeitung der nachfolgenden Teams bei der Projektübergabe wesentlich erleichtern.

Alles in allem macht Babylon.js den Eindruck näher an einer vollständigen Game Engine zu sein als A-Frame oder Three.js, da dort auch fortgeschrittenere Themen wie Szenen- oder Performanceoptimierung zur Sprache kommen. Dieser und die oben genannten Gründe waren schliesslich ausschlaggebend bei der Entscheidung für Babylon.js.

Zusammenfassung

In diesem Kapitel wurden im Technologievergleich zunächst verschiedene 3D-Grafik-Libraries miteinander verglichen, um einen Überblick zu verschaffen. In der Planung des Anwendungsfalls wurde veranschaulicht, wie die konkrete Implementation aussehen könnte. Schlussendlich konnte, auf Basis des Technologievergleichs und des Anwendungsfalls, Babylon.js als vielversprechendste Technologie für die Umsetzung der Software City ausgewählt werden. Diese Technologie soll nun im restlichen Verlauf der Arbeit genauer untersucht werden.

Kapitel 4

Anforderungs- und Domänenanalyse der Software City

Übersicht

In diesem Kapitel werden alle Anforderungen an die Software City beschrieben. Im Abschnitt Funktionale Anforderungen wird zunächst festgelegt, mit welchen Features die Umsetzung der Software City konkret ausgestattet werden soll. Im Abschnitt Nichtfunktionale Anforderungen werden die Qualitätsanforderungen der Anwendung beschrieben. Am Schluss wird in einem Domänenmodell die Problemstellung verdeutlicht.

4.1 Funktionale Anforderungen

Das Use Case Diagramm in der Abbildung 4.1 dient nur der Übersicht. Genauere Beschreibungen der Use Cases im Fully Dressed Format folgen weiter unten. Die weissen Use Cases sind in der Umsetzung fest eingeplant. Die blauen Use Cases können implementiert werden, falls noch genügend Zeit vorhanden ist.

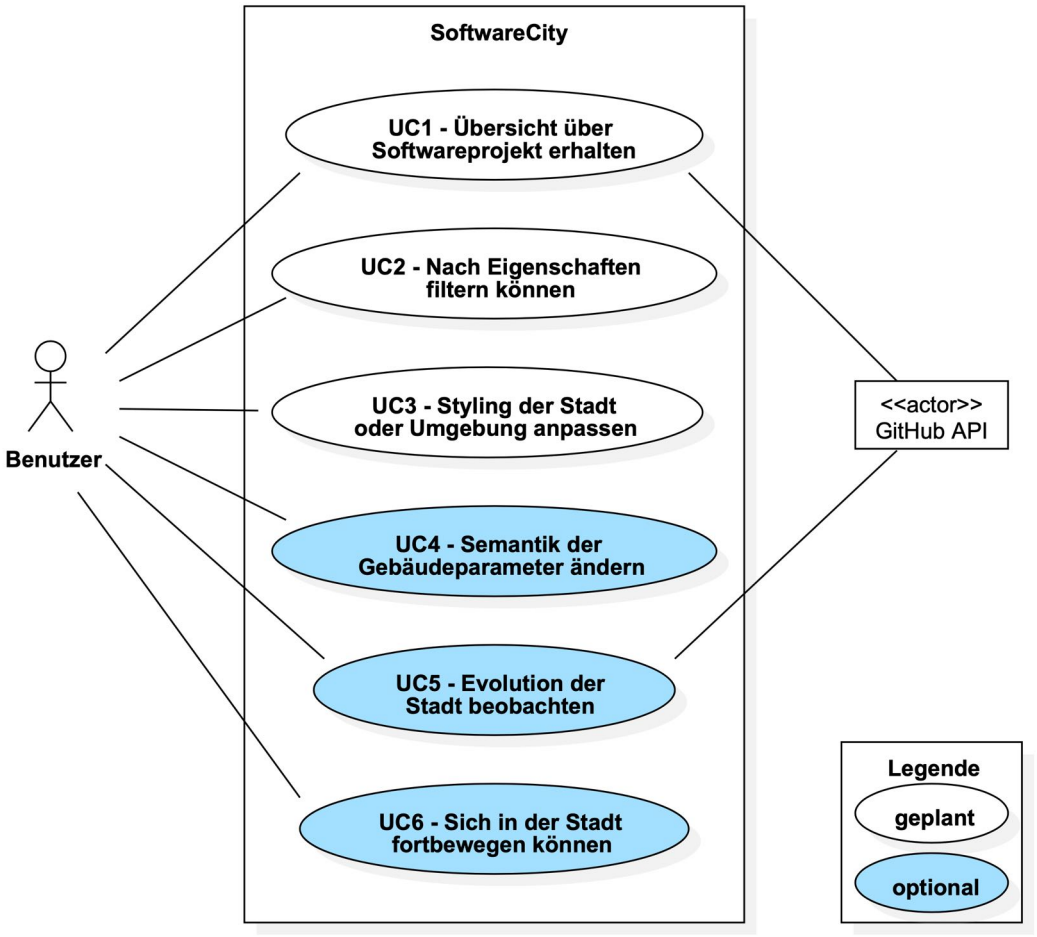


Abbildung 4.1: Use Case Diagramm

4.1.1 Aktoren und Stakeholder

Es interagieren zwei Aktoren mit dem Software System zusammen. Diese sind auch in der Abbildung 4.1 aufgeführt.

Tabelle 4.1: Aktoren

Aktoren	
Benutzer	Der Benutzer stellt den Anwender der Applikation dar. Es werden keine spezifischen Benutzerrollen festgelegt.
GitHub API	Die GitHub API liefert die nötigen Daten in Form von Source Code, damit die Software City überhaupt erzeugt werden kann.

Die Stakeholders haben in irgendeiner Weise Interesse am Projekt.

Tabelle 4.2: Stakeholder

Stakeholder	
Entwickler	Die Entwickler sind verantwortlich für die rechtzeitige Lieferung der Software. Ausserdem erhoffen sie sich, dass sie durch das Arbeiten mit neuen Technologien viel neues dazulernen.
Fachhochschule OST	Die OST stellt die Ressourcen für das Projekt zur Verfügung und ist interessiert an einem guten Ergebnis der Arbeit.
IFS	Das IFS will das Projekt bei Bedarf in weiteren Arbeiten weiterverfolgen.

4.1.2 UC1: Übersicht über Softwareprojekt erhalten

Tabelle 4.3: UC1: Übersicht über Softwareprojekt erhalten

Merkmal	**Beschreibung**
Level	User-goal
Primary Actor	Benutzer
Stakeholders and Interests	Der Benutzer möchte eine Übersicht über die statische Struktur eines Software Projektes erhalten.
Preconditions	Der Benutzer hat sein Projekt in einem public GitHub Repository abgespeichert.
Success Guarantee	Der Benutzer konnte erfolgreich sein Projekt analysieren und versteht jetzt besser wie das Softwareprojekt aufgebaut ist.
Main Success Scenario	1. Der Benutzer besucht die Website der Software City. 2. Der Benutzer gibt der Website seinen GitHub-Username und Name vom Repository seines Softwareprojektes an. 3. Nachdem die Website das Projekt analysiert hat, kann der Benutzer die Stadt betrachten. 4. Auf Wunsch kann der Benutzer die Stadt rotieren und hineinzoomen.

Extensions	1. Der Benutzer will Eigenschaften von spezifischen Stadtteilen genauer betrachten. (a) Der Benutzer selektiert ein Gebäude. (b) Folgende Details werden dem Benutzer angezeigt: (i) Source Code dieses Gebäudes (ii) Detailliertere Metriken wie: Anzahl Codezeilen, Anzahl Methoden, Anzahl Klassenvariablen, Package, Modifiers (iii) Möglichkeit das Gebäude auszublenden 2. Der Benutzer sucht eine konkrete Komponente in der Stadt. (a) Der Benutzer verwendet die Suchfunktion, um seine Komponente zu finden. (b) Die Komponente wird in der Stadt markiert. 3. Dem Benutzer will gewisse Stadtteile ausblenden. (a) Der Benutzer kann für jede Komponente der Stadt definieren, ob sie eingeblendet werden soll oder nicht. 4. Die Analyse vom angegebenen Projekt schlägt fehl. (a) Die Applikation weist den Benutzer mit einer Fehlermeldung darauf hin. (b) Der Benutzer gibt ein alternatives Projekt an oder passt sein bestehendes an. 5. Der Benutzer will ein anderes Projekt begutachten. (a) Der Benutzer kann ein neues Projekt angeben. 6. Optional: Das Projekt ändert sich im Hintergrund. (a) Der Benutzer kann die Stadt aktualisieren.
Frequency of Occurrence	Regelmässig

4.1.3 UC2: Nach Eigenschaften filtern können

Tabelle 4.4: UC2: Codequalität besser beurteilen können

Merkmal	**Beschreibung**
Level	User-goal
Primary Actor	Benutzer
Stakeholders and Interests	Der Benutzer möchte die Codequalität besser beurteilen können und erkennen, welche Codeabschnitte im Software Projekt ein Refactoring benötigen.
Preconditions	Der Benutzer hat seine Stadt bereits geladen.
Success Guarantee	Der Benutzer konnte erfolgreich herausfinden, ob das Software Projekt seine Qualitätsansprüche erreicht.

Main Success Scenario	1. Der Benutzer will die Qualität der Software begutachten. 2. Der Benutzer gibt an, welche Metrik (z.B. Anzahl Zeilen Klasse, Anzahl Funktionen, Anzahl Fields) er analysieren will und definiert einem eigenen Schwellwert. 3. Die Gebäude, die den Schwellwert überschreiten, werden markiert.
Extensions	-
Frequency of Occurrence	Regelmässig

4.1.4 UC3: Styling der Stadt oder Umgebung anpassen

Tabelle 4.5: UC3: Styling der Stadt oder Umgebung anpassen

Merkmal	**Beschreibung**
Level	User-goal
Primary Actor	Benutzer
Stakeholders and Interests	Der Benutzer möchte das rein visuelle Aussehen der Gebäude oder der Umgebung anpassen.
Preconditions	Der Benutzer hat seine Stadt bereits geladen.
Success Guarantee	Der Benutzer konnte erfolgreich das Styling nach seinen Wünschen anpassen.
Main Success Scenario	1. Der Benutzer will das Aussehen der Gebäude ändern (z.B. Textur, Form, Farbe). 2. Der Benutzer wählt sein bevorzugtes Styling aus. 3. Die Applikation übernimmt die Einstellungen und zeichnet die Stadt entsprechend neu.

Extensions	1. Der Benutzer will die Tageszeit ändern. (a) Er wählt eine Zeit. (b) Die Applikation aktualisiert das Umgebungslicht und den Schattenwurf der Stadt. (c) Bei Nacht beginnen Teile der Stadt zu leuchten. 2. Optional: Der Benutzer will das Aussehen der Umgebung ändern (z.B. Textur, Form, Farbe). (a) Er wählt seine neue Umgebung aus. (b) Die Applikation aktualisiert die Szene. 3. Optional: Der Benutzer will das Stadtlayout anpassen. (a) Er wählt ein Layout. (b) Die Applikation aktualisiert die Szene. 4. Optional: Der Benutzer will zwischen verschiedenen Ansichten wechseln. (a) Er kann zwischen der Vorderansicht und Seitenansicht wechseln. 5. Optional: Der Benutzer will die Stadt mehr beleben. (a) Er aktiviert Autos und Fussgänger. (b) In der Stadt erscheinen Autos und Fussgäner.
Frequency of Occurrence	Regelmässig

4.1.5 UC4: Semantik der Gebäudeparameter ändern (optional)

Tabelle 4.6: UC4: Semantik der Gebäudeparameter ändern

Merkmal	Beschreibung
Level	User-goal
Primary Actor	Benutzer
Stakeholders and Interests	Der Benutzer möchte die Bedeutung der Gebäudeparameter ändern.
Preconditions	Der Benutzer hat seine Stadt bereits geladen.
Success Guarantee	Der Benutzer konnte den Dimensionen der Gebäude erfolgreich eine neue Bedeutung zuweisen.
Main Success Scenario	1. Der Benutzer will die Parameterisierung der Gebäude ändern. 2. Der Benutzer weist den Gebäudedimensionen neue Argumente zu. 3. Die Applikation übernimmt die Einstellungen und zeichnet die Stadt entsprechend neu.
Extensions	-
Frequency of Occurrence	Selten

4.1.6 UC5: Evolution der Stadt beobachten (optional)

Tabelle 4.7: UC5: Evolution der Stadt beobachten

Merkmal	Beschreibung
Level	User-goal
Primary Actor	Benutzer
Stakeholders and Interests	Benutzer: Möchte beobachten, wie sich die Stadt über die Zeit verändert.
Preconditions	Der Benutzer hat einen initialen Stand seiner Stadt bereits geladen. Das Projekt im GitHub Repository verfügt über mehrere Commits.
Success Guarantee	Der Benutzer konnte die zeitliche Veränderungen des Softwareprojektes erfolgreich beobachten.
Main Success Scenario	1. Der Benutzer will die Evolution der Stadt beobachten. 2. Der Benutzer wählt eine andere Zeit aus. 3. Die Applikation übernimmt die Einstellungen und zeichnet die Stadt entsprechend neu.
Extensions	-
Frequency of Occurrence	Selten

4.1.7 UC6: Sich in der Stadt forbewegen können (optional)

Tabelle 4.8: UC6: Sich in der Stadt fortbewegen können

Merkmal	Beschreibung
Level	User-goal
Primary Actor	Benutzer
Stakeholders and Interests	Benutzer: Möchte durch die Stadt laufen können und die Stadt von einer anderen Perspektive betrachten.
Preconditions	Der Benutzer hat seine Stadt bereits geladen.
Success Guarantee	Der Benutzer konnte sich erfolgreich durch die virtuelle Stadt fortbewegen.
Main Success Scenario	1. Der Benutzer will durch die Stadt laufen. 2. Der Benutzer markiert seinen Startpunkt innerhalb der Stadt. 3. Der Benutzer wird proportional zur Stadt geschrumpft, an die gewünschte Stelle innerhalb der Stadt teleportiert und kann sich nun frei bewegen. Die Interaktionen mit den Gebäuden werden nach wie vor unterstützt.
Extensions	-
Frequency of Occurrence	Regelmässig

4.2 Nichtfunktionale Anforderungen

In diesem Abschnitt werden die Qualitätsanforderungen der Anwendung genauer beschrieben. Alle nichtfunktionalen Anforderungen wurden gemäss ISO/IEC 25010:2011[1] spezifiziert.

4.2.1 Functional Suitability

Tabelle 4.9: NFR: Reliability

Nr	NFR	Beschreibung	Acceptance Criteria
01	Completeness	Der User muss alle im vorherigen Abschnitt spezifizierten nicht-optionalen Use Cases durchführen können.	System Tests
02	Correctness	Die Projekthierarchie soll bei der erstellten Software City vorhanden bleiben.	System Tests

[1] Siehe https://www.iso.org/standard/35733.html

4.2.2 Reliability

Tabelle 4.10: NFR: Reliability

Nr	NFR	Beschreibung	Acceptance Criteria
03	Fault Tolerance	Wenn die GitHub API nicht verfügbar ist, wird eine Fehlermeldung angezeigt. Bei einem Fehler während des Analyseprozesses vom Software Projekt wird eine Fehlermeldung angezeigt	System Tests
04	Recoverability	Bei einem Absturz des statischen Fileservers, müsste ein Projektmitglied in der Lage sein innerhalb von 12h diesen wieder zum laufen zu bringen.	Bei einem simulierten Absturz den Server in vorgegebener Zeit wieder zum laufen bringen

4.2.3 Performance Efficiency

In diesem Abschnitt wird ein Referenzprojekt erwähnt. Damit ist jeweils das GitHub Repository BlackWidow-Chess gemeint:

- Username: amir650
- Repository: BlackWidow-Chess
- Commit Hash: 1a8716160dc188842377b2d3ea486856c344da10

Tabelle 4.11: NFR: Performance Efficiency

Nr	NFR	Beschreibung	Acceptance Criteria
05	Time Behavior	Die Analyse eines Software Projektes (inklusive Abfrage über das API, bis hin zur gerenderten Stadt) soll nicht allzu lange dauern	Das Referenzprojekt von insgesamt 11740 Zeilen Code mit 59 Klassen, 4 Interfaces und 4 Enums soll im Schnitt in unter einer Minute geladen worden sein.
06	Resource Utilization	Die Szene kann je nach Projektgrösse viele Komponenten enthalten, was der Performance schadet. Der Benutzer soll darauf hingewiesen werden, dass bei grösseren Projekten mit FPS-Drops gerechnet werden muss.	System Tests
07	Capacity	Die Applikation soll ein Referenzprojekt ohne FPS-Drops verarbeiten und darstellen können	Das Referenzprojekt mit 67 Gebäuden soll bei einer stabilen Framerate von 60FPS unterstützt werden.

4.2.4 Compatibility

Tabelle 4.12: NFR: Compatibility

Nr	NFR	Beschreibung	Acceptance Criteria
08	Interoperability	Die Applikation soll auf verschiedenen VR-Browsern laufen und getestet werden	Unterstützt werden folgende Browser: Oculus Browser und Firefox Reality

4.2.5 Usability

Tabelle 4.13: NFR: Usability

Nr	NFR	Beschreibung	Acceptance Criteria
09	Learnability	Die Applikation ist intuitiv bedienbar (selbstbeschreibend)	Usability Test
10	User Error Protection	Wenn ein vom Benutzer angegebenes Repository nicht existiert wird eine Fehlermeldung angezeigt	System Tests
11	User Interface Aesthetics	Die Applikation ist für die Anwendung in 3D-Umgebungen optimiert	Usability Test

4.2.6 Maintainability

Tabelle 4.14: NFR: Maintainability

Nr	NFR	Beschreibung	Acceptance Criteria
12	Modularity	Folgende Komponenten können durch verhältnismässig geringen Aufwand durch neu-Implementationen ersetzt werden: Parser, Software City Layout Builder	Genaue Spezifikation ist vorhanden, welche Daten von einer Komponente zur nächsten übergeben werden müssen
13	Modifiability	Durch Tests wird sichergestellt, dass die Funktionalität durch Modifikationen im Code nicht verloren geht.	Test Coverage für die Business Logik beträgt 70% Statement-Coverage
14	Testability	Alle Test Cases müssen erfolgreich durchlaufen, bevor eine Änderung im Source Code akzeptiert wird.	Pipeline führt Tests erfolgreich aus

4.2.7 Portability

Tabelle 4.15: NFR: Portability

Nr	NFR	Beschreibung	Acceptance Criteria
15	Installability	Der statische Fileserver muss in einer Linux Umgebung installier- und ausführbar sein	Deployment erfolgreich

4.3 Domänenmodell

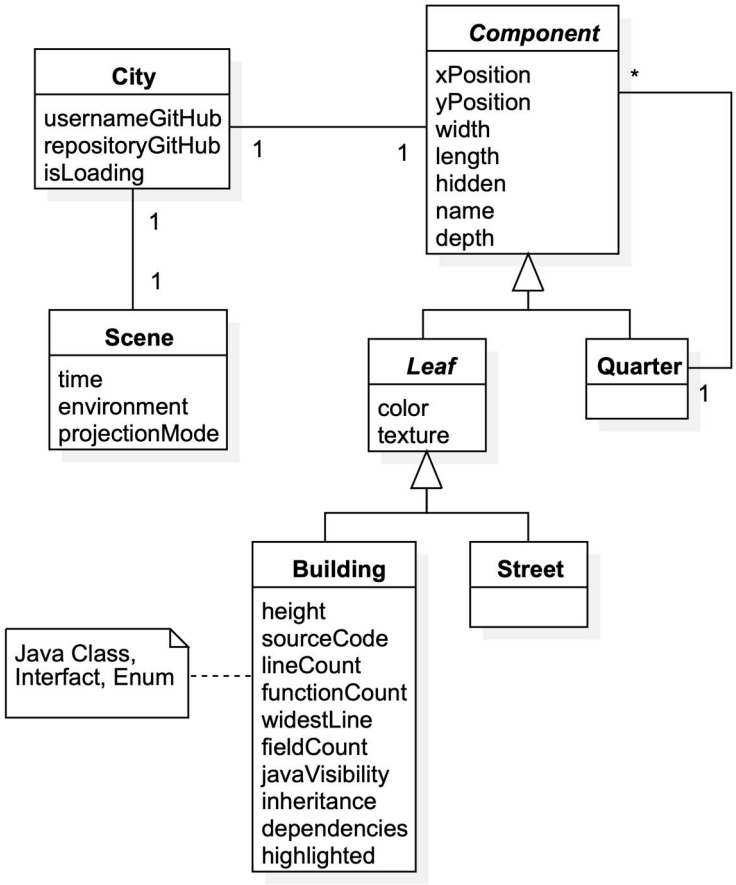


Abbildung 4.2: Domain Model

Das rekursive Design, dass Stadtviertel (Quarter) weitere Komponenenten beinhalten können, ermöglicht eine simple Handhabung der Stadt. Inspiriert wurde dieser Ansatz durch das Composite Pattern.

Zusammenfassung

Dieses Kapitel hat sich mit den funktionalen und nichtfunktionalen Anforderungen an die zu implementierende Anwendung auseinandergesetzt. Die funktionalen Anforderungen wurden in optionale und nicht optionale Anforderungen aufgeteilt. Das Domänenmodell gab einen groben Einblick in die Problemstellung des Anwendungsfalls.

Kapitel 5

Softwarearchitektur

Übersicht

Dieses Kapitel bildet das Fundament, wie die Anwendung umgesetzt wird. Neben der allgemeinen Architektur wird auch das Softwaredesign auf Klassenebene beschrieben. Im Abschnitt Bausteinansicht werden die Komponenten der Software definiert. Der Abschnitt Laufzeitansicht beschäftigt sich mit der Verhalten zur Laufzeit. Danach wird das Deployment beschrieben.

5.1 Bausteinansicht

Die Architektur wird im Stil von C4 dargestellt. Die Idee ist es, die Architektur über die verschiedenen Levels Context, Containers, Components und Code aufzuzeigen (daher die vier "C").

5.1.1 Level 1: System Kontext

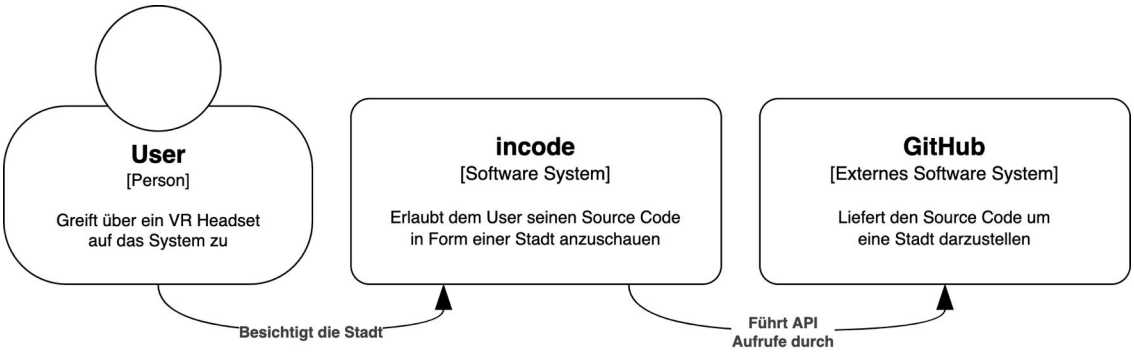


Abbildung 5.1: Level 1 System Kontext

Der User greift auf incode zu und kann mit der Anwendung interagieren. Incode kommuniziert im Hintergrund mit GitHub.

5.1.2 Level 2: Container Diagramm

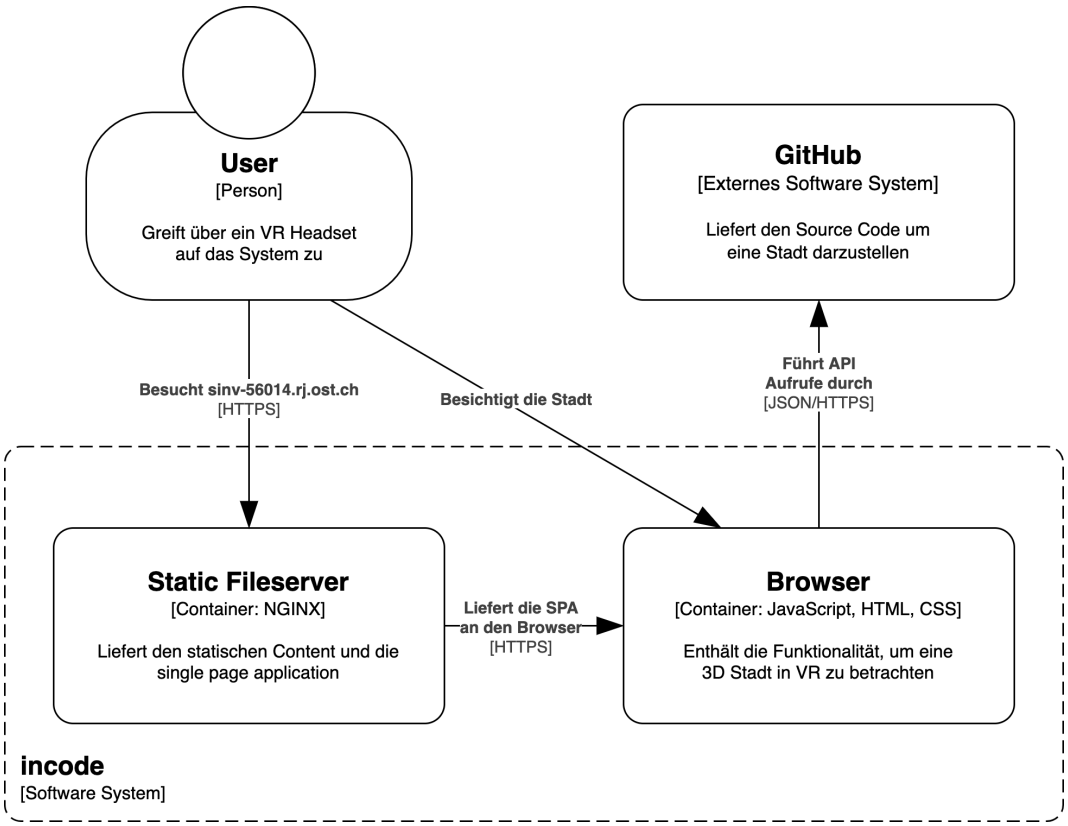


Abbildung 5.2: Level 2 Container Diagramm

Incode besteht aus dem statischen Fileserver und der Single Page Application. Der statische Fileserver wird mit NGINX umgesetzt. Er wird so konfiguriert, dass er alle HTTP Anfragen automatisch auf HTTPS weiterleitet. Die Single Page Application wird mit native TypeScript und Babylon.js umgesetzt. An den Browser werden schlussendlich HTML, CSS und JavaScript Files ausgeliefert.

5.1.3 Level 3: Komponenten Diagramm

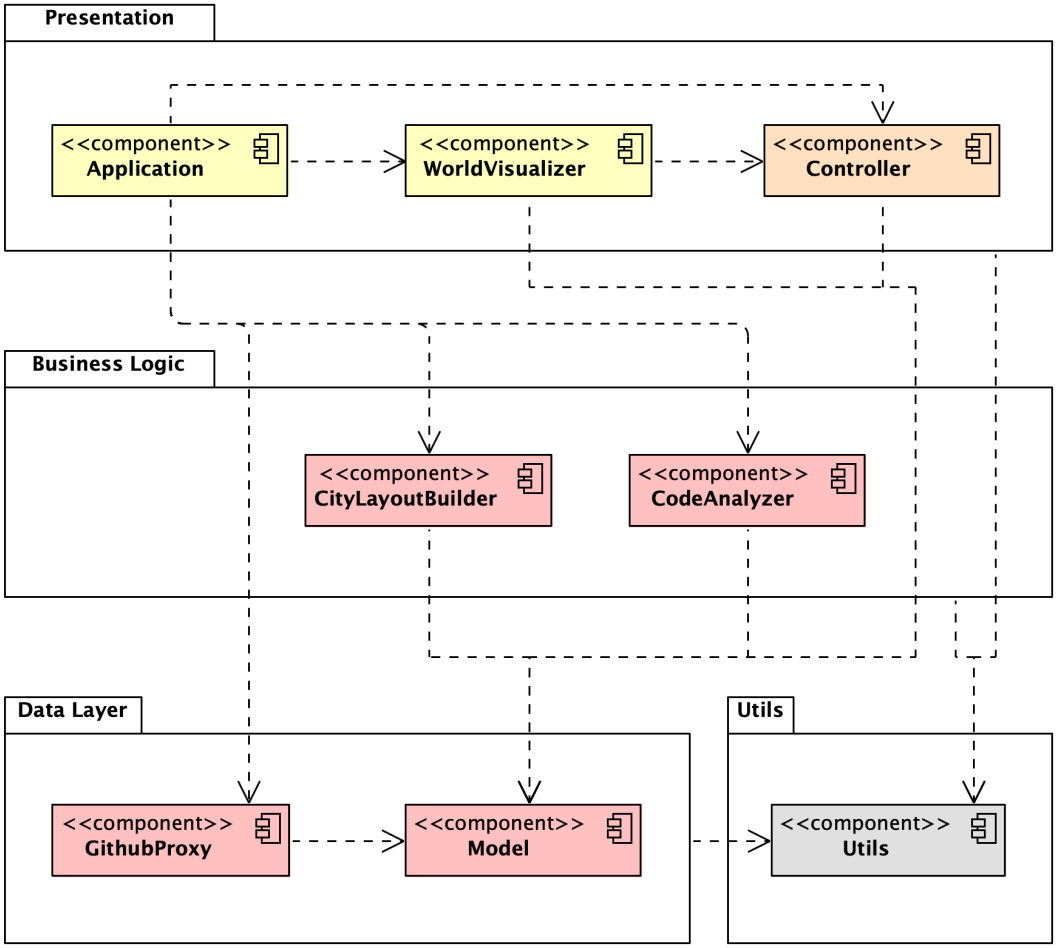


Abbildung 5.3: Level 3 Komponenten Diagramm

In der Abbildung 5.3 wird die Single Page Application genauer betrachtet. Die Realisierung erfolgt als Schichtenarchitektur. Die *Presentation*-Schicht kümmert sich um alles, was mit der Darstellung zu tun hat. In der *Business Logic* werden die Algorithmen abgelegt, die zur Erstellung der Stadt nötig sind. Der *Data Layer* besorgt die Daten von der GitHub REST API und beherbergt Daten, die von den oberen Schichten zwischengespeichert werden müssen.

Die Komponenten passen ebenfalls in das MVC Design Pattern. Die gelben Komponenten sind die View, orange ist der Controller und rot das Model von MVC.

In der Utils Komponente werden Helper Klassen abgelegt, die sonst nirgendwo reinpassen.

5.1.4 Level 4: Klassendiagramm

Nachfolgend wird genauer auf die einzelnen Komponenten von Level 3 eingegangen. Bei der Darstellung der Klassen und Interfaces wurde zwecks Verständlichkeit nur das wichtigste abgebildet. Auch die Auflistungen der Klassenvariablen und Methoden sind bei Weitem nicht abschliessend.

Application

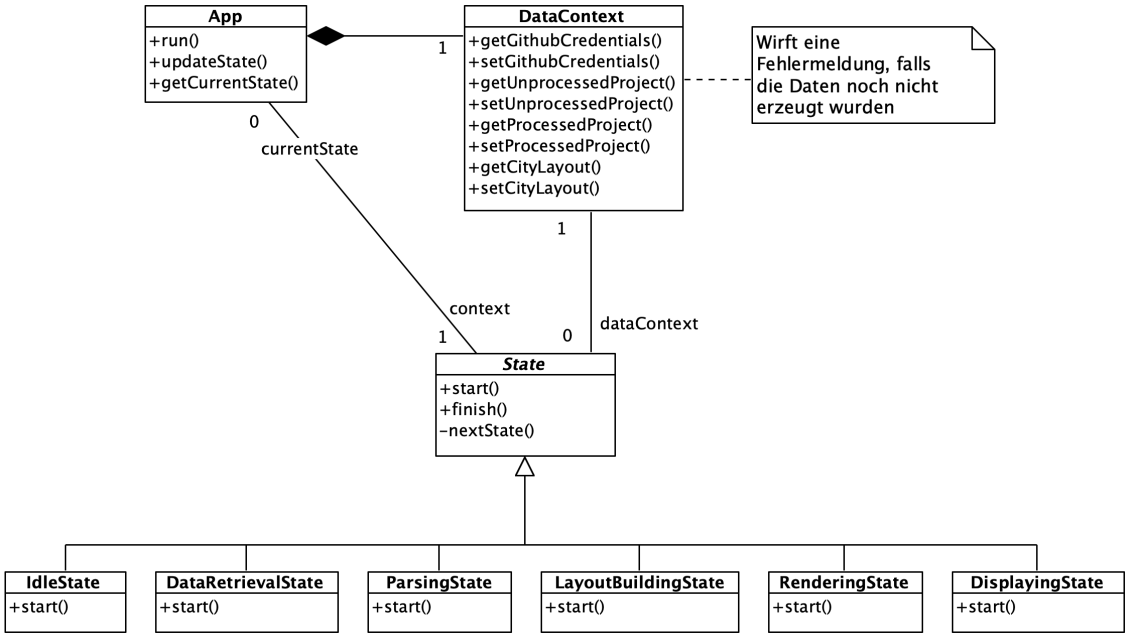


Abbildung 5.4: Level 4 Klassendiagramm Application

Die `App` wird zu Beginn hochgefahren und orchestriert alle anderen Komponenten. Da das ganze System eine State Machine ist (Abschnitt 5.2.1), wird von der `App` einen State gehalten. Die Umsetzung erfolgt nach dem State Pattern. Wird die `App` erzeugt, ist ihr initialer State der `IdleState`. Dort wird so lange verweilt, bis der User sein GitHub Repository angegeben hat. Auf dem `IdleState` wird dann `finish` aufgerufen, was intern über `nextState` den nächsten State triggert. In diesem Fall würde in den `DataRetrievalState` gewechselt werden. Allgemein weiss jeder State selber, welcher State als nächstes folgt. Eine Übersicht über die Abfolge der States findet sich im Abschnitt 5.2.2. Über den `DataContext` können die States ihre Arbeitsresultate mit den anderen States teilen.

GithubProxy

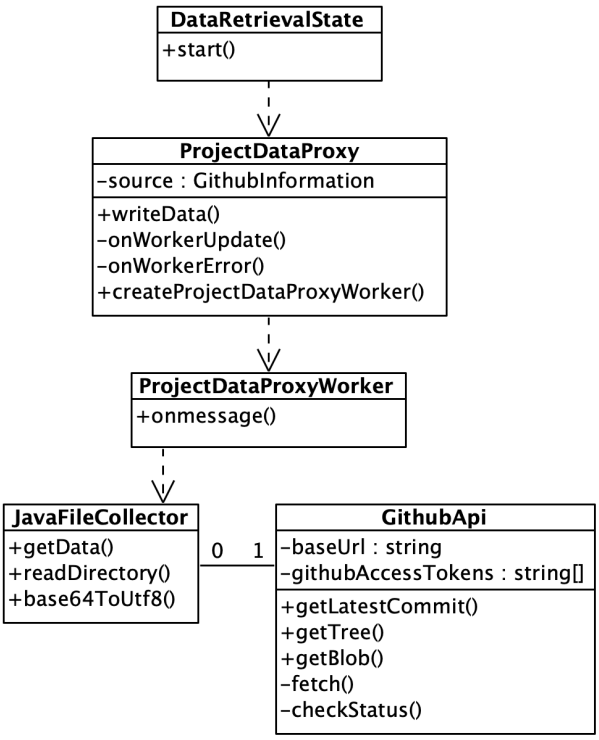


Abbildung 5.5: Level 4 Klassendiagramm GithubProxy

Der `ProjectDataProxy` fährt den `ProjectDataProxyWorker` hoch, welcher einen Web Worker darstellt. Dieser startet den `JavaFileCollector` welcher sich um die Abfrage des GitHub Projekts kümmert. In der `GithubApi` findet die effektive Abfrage zur GitHub REST API[1] statt. Es werden alle Javafiles (Files mit der .java Endung) eingelesen. Die Base64 codierten Files der GitHub API werden mit der `base64ToUtf8` Methode decodiert. Damit alle Files gefunden werden, wird das GitHub Repository rekursiv nach den Javafiles durchsucht. Das bedeutet, dass jedes gefundene Javafile über eine separate Abfrage geladen wird. Das ist nicht optimal, wurde aber so gewählt, da die alternativen Abfragen bezüglich Fileanzahl und Filegrösse Limiten aufweisen. In der `GithubApi` werden (falls nötig mehrere) GitHub Access Tokens gespeichert. Über ein einzelnes Access Token kann man 5000 Abfragen pro Stunde durchführen. Die `checkStatus` Methode überprüft den HTTP Status der Antwort.

[1] GitHub REST API im Kapitel 6.1.5

CodeAnalyzer

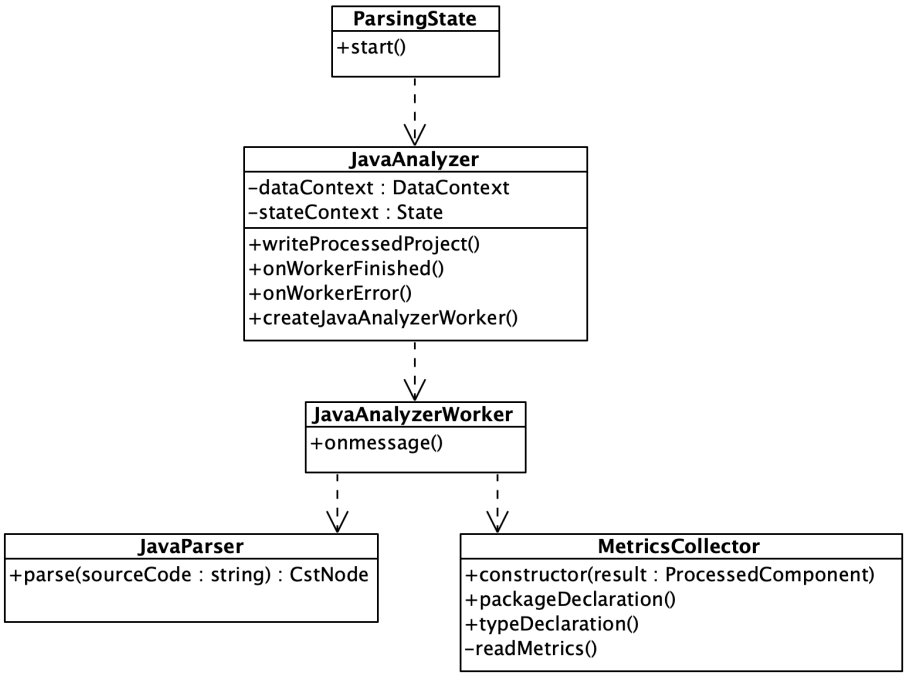


Abbildung 5.6: Level 4 Klassendiagramm Analyzer

Die Komponente *CodeAnalyzer* kümmert sich um die Verarbeitung des rohen Source Codes. Der `JavaAnalyzer` wird vom `ParsingState` aufgerufen. Die Methode `writeProcessedProject` startet den `JavaAnalyzerWorker`. Dieser ist ein Web Worker, der nun im Hintergrund die Daten aufbereitet. Der `JavaParser` entspricht dem npm Modul java-parser[2] und gibt einen CST zurück. Der `MetricsCollector` kann dann mit Hilfe vom Visitor Pattern die gewünschten Daten vom CST auslesen. Er hängt sich bei den Package- und Typendeklarationen von Javafiles rein. Dank dem Composite Pattern muss die rekursive Iteration durch die Javafiles nicht selber geschrieben werden, denn die Datenstruktur weiss selber schon wie sie traversiert wird.

[2] Java Parser basierend auf dem Chevrotain Parsing Toolkit: https://www.npmjs.com/package/java-parser

CityLayoutBuilder

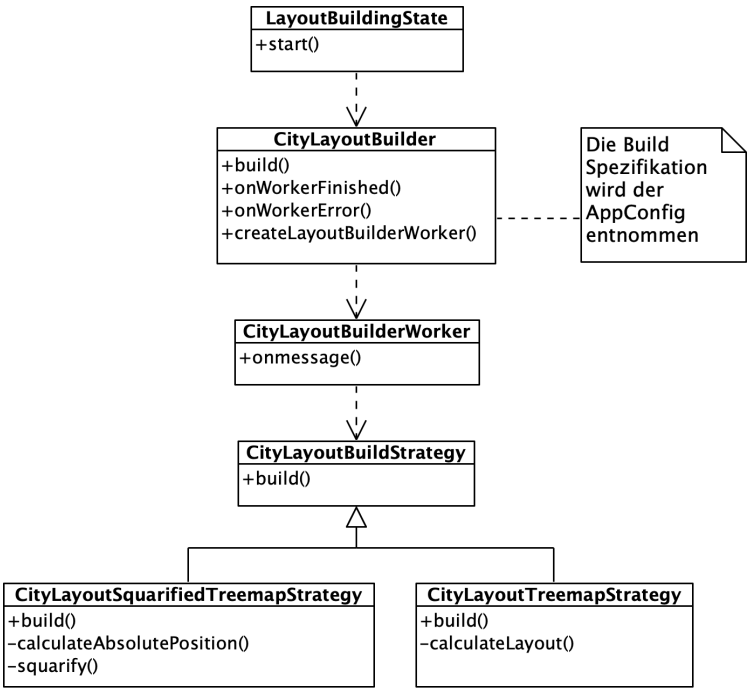


Abbildung 5.7: Level 4 Klassendiagramm CityLayoutBuilder

Der `CityLayoutBuilder` erstellt nicht selber das Layout der Stadt, sondern aktiviert, wie in den Beispielen oben, nur den Web Worker. Der `CityLayoutBuilderWorker` instanziert eine der beiden Layout Algorithmen. In der `AppConfig` (hier nicht abgebildet) wird festgehalten, welche Strategie verwendet werden soll. Zu beachten ist, dass durch die Anwendung des Strategy Patterns die Erweiterung durch neue Layoutstrategien möglich ist. Wichtig ist nur, dass die Strategien den Korrekten Rückgabetypen liefern, damit das Resultat im `DataContext` (hier nicht abgebildet) abgelegt werden kann.

WorldVisualizer

Der *WorldVisualizer* kann in zwei Unterkomponenten aufgeteilt werden. Die eine Unterkomponente kümmert sich um das Rendering der Stadt, die andere um die allgemeine Darstellung und Interaktion mit der virtuellen Welt. Zuerst wird das Rendering der Stadt betrachtet.

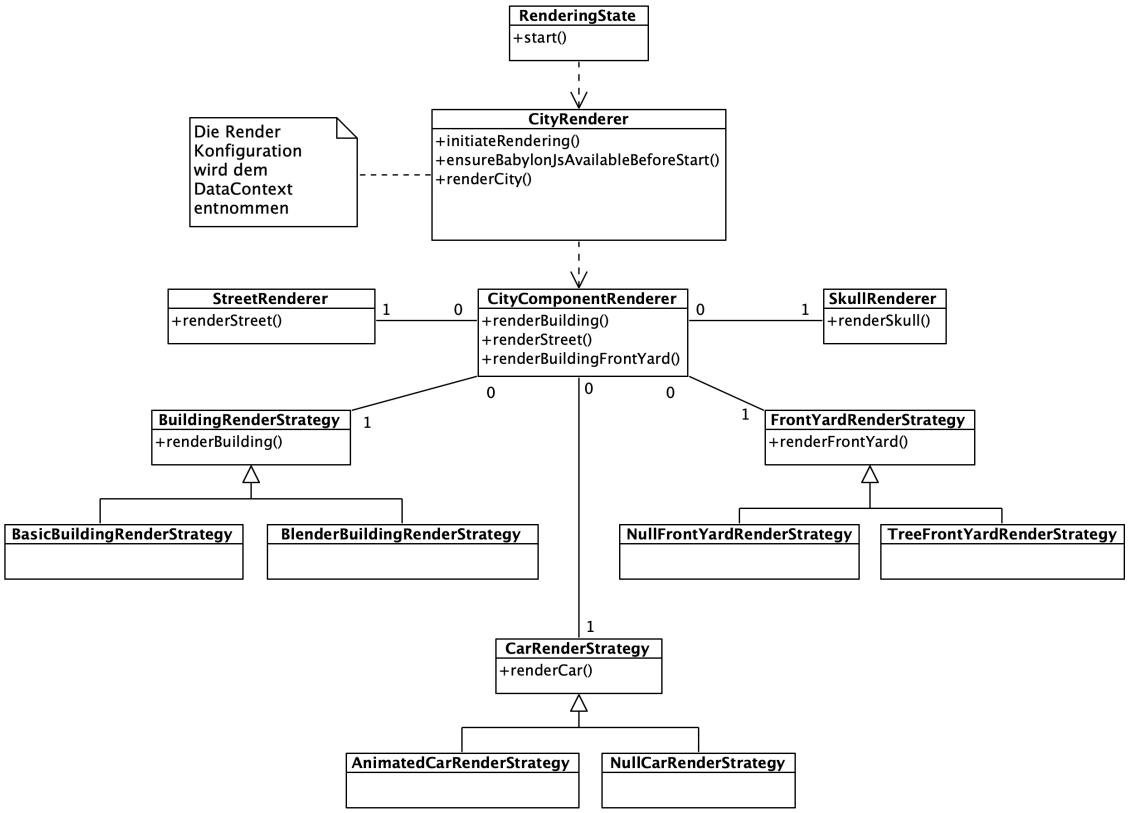


Abbildung 5.8: Level 4 Klassendiagramm WorldVisualizer Renderer

Der `CityRenderer` übernimmt das initiale Rendering der Stadt. Auch wenn sich Einstellungen, wie die Filterung, ändern, rendert diese Klasse die gesamte Stadt neu. Anderst als vorhin kann der CityRenderer diese Arbeit nicht einem Worker Thread übergeben, da das Rendering auf dem Main Thread geschehen muss. Über eine Renderkonfiguration (aus dem `DataContext`, der in der Abbildung 5.8 nicht abgebildet ist) kann das Rendering angepasst werden. So lässt sich je nach Bedarf eine detaillierte Stadt darstellen, oder eine simple Stadt mit primitiven Gebäuden für die schwächere Hardware.

Um nun die Stadt zu rendern, wird keine separate rekursive Methode geschrieben, sondern es wird die Composite Datenstruktur `CityComponent` (Siehe 5.1.4) verwendet. Auf der obersten Komponente wird die render Methode aufgerufen und ein `CityComponentRenderer` übergeben. Die Komponente rendert nun selbständig alle seine Kinder. Der `CityComponentRenderer` enthält alle spezifischen Renderers und zeichnet nun die Stadt.

Die Renderers mit dem *Strategy*-Suffix implementieren das Strategy Pattern. Je nach Konfiguration wird eine andere Renderstrategie ausgewählt. Der `CityComponentRenderer` ist ein transient Service. Heisst bei jeder Verwendung wird er neu erzeugt. So können immer wieder neue Rendererstrategien im Konstruktor übergeben werden. Renderstrategien mit dem *Null*-Präfix implementieren das Null Object Pattern, indem die Strategie dann nichts rendert. So kann auf Nullchecks verzichtet werden.

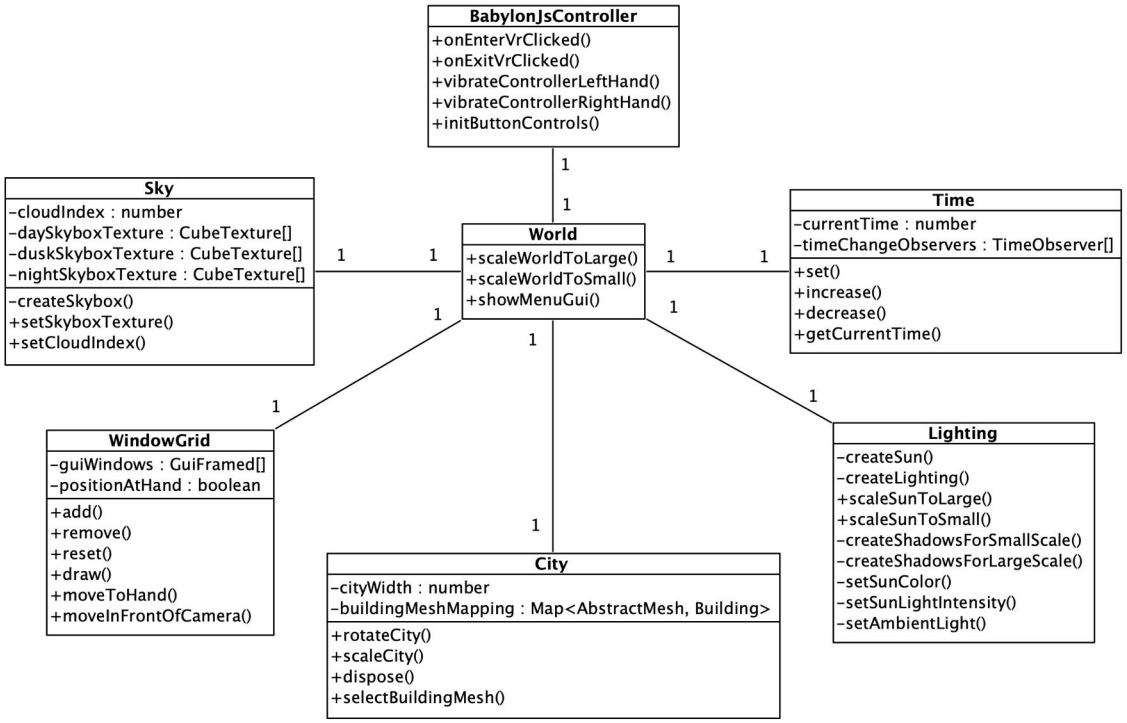


Abbildung 5.9: Level 4 Klassendiagramm WorldVisualizer virtuelle Welt

Die Abbildung 5.9 beschreibt den Aufbau der VR-Welt. Der `BabylonJsController` ist nicht Teil vom *WorldVisualizer*, sondern vom *Controller*, wurde hier aber als Orientierungshilfe eingezeichnet.

World Die `World` ist die Hauptklasse und orchestriert alle anderen Elemente. Die Welt kann über die entsprechenden Funktionen skaliert werden. Diese Funktion ist nötig, damit sich der Benutzer in die Stadt hineinteleportieren kann. Es besteht auch die Möglichkeit das Menu anzuzgeigen.

Sky Wegen den einstellbaren Tageszeiten und der Möglichkeit, einen unterschiedlichen Grad an Wolken im Himmel anzuzeigen ist eine separate `Sky` Klasse nötig. Es wird für jede Tageszeit mindestens eine Skybox (Siehe Kapitel 8.2.1) Textur abgespeichert. Die Textur beim Index 0 ist wolkenlos, während die Texturen der höheren Indexe immer mehr Wolken enthalten. Zu Beginn wird das Attribut `cloudIndex` über die `setCloudIndex` Methode gesetzt. Der Wert von `cloudIndex` ist abhängig von der Formattierung des zu analysierenden Software Projektes. Das Setzen dieses Attributes aktualisiert die Skybox Textur.

WindowGrid Dem Benutzer sollen Interaktionsfenster angezeigt werden. Neben dem Hauptmenu gibt es allerdings auch weitere Fenster, die der Benutzer gleichzeitig dargestellt haben möchte. Dieses Problem löst das `WindowGrid`. Es können verschiedene Fenster übergeben werden und das `WindowGrid` kümmert sich um dessen Positionierung. Im Array `guiWindows` werden die Fenster abgelegt, die dargestellt werden möchten.

Der Benutzer soll wählen können, ob er seine Fenster auf der Hand anzeigen möchte oder an einem fixierten Ort vor seiner Position. Deshalb werden noch die Methoden `moveToHand` und `moveInFrontOfCamera` angeboten.

City Die Stadt mit all ihren Elementen wird in der `City` abgelegt. Das `buildingMeshMapping` speichert die zugehörigen Babylon.js Meshes zu den Business Objekten der `Buildings` ab. Diese `Building` Klasse enthält die Metriken der Gebäude. Falls der Benutzer ein Mesh selektiert, wird über dieses Mapping das zugrundeliegende Building Objekt gefunden, damit die korrekten Metriken angezeigt werden können. Diese Speicherung der Meshes gilt nicht nur für die Gebäude, sondern für alle Elemente, die über den Renderer erzeugt werden und Teil der Stadt sind. So kann beim Rendering (Abbildung 5.8) einer neuen Stadt, über diese Mappings iteriert und jeder Mesh gelöscht werden. Dies würde dann über die `dispose` Methode geschehen.

Lighting Alles was mit der Belichtung und Schattierung zu tun hat, kommt in die `Lighting` Klasse. Falls die `World` skaliert wird, muss die Sonne über die `scaleSunTo...`-Methoden mitskaliert werden. Wird die Tageszeit angepasst, können über die Methoden `setSunColor`, `setSunLightIntensity` und `setAmbientLight` die Farbe der Sonne, dessen Stärke und das Umgebungslicht angepasst werden.

Time In der `Time` Klasse wird die aktuelle Tageszeit abgespeichert. Dank des Observer Patterns können sich andere Klassen hier auf Änderungen der Tageszeit abonnieren. So implementiert unter anderem die `Lighting` Klasse das `TimeObserer` Interface[3].

[3] nicht in der Abbildung 5.9 enhalten

Model

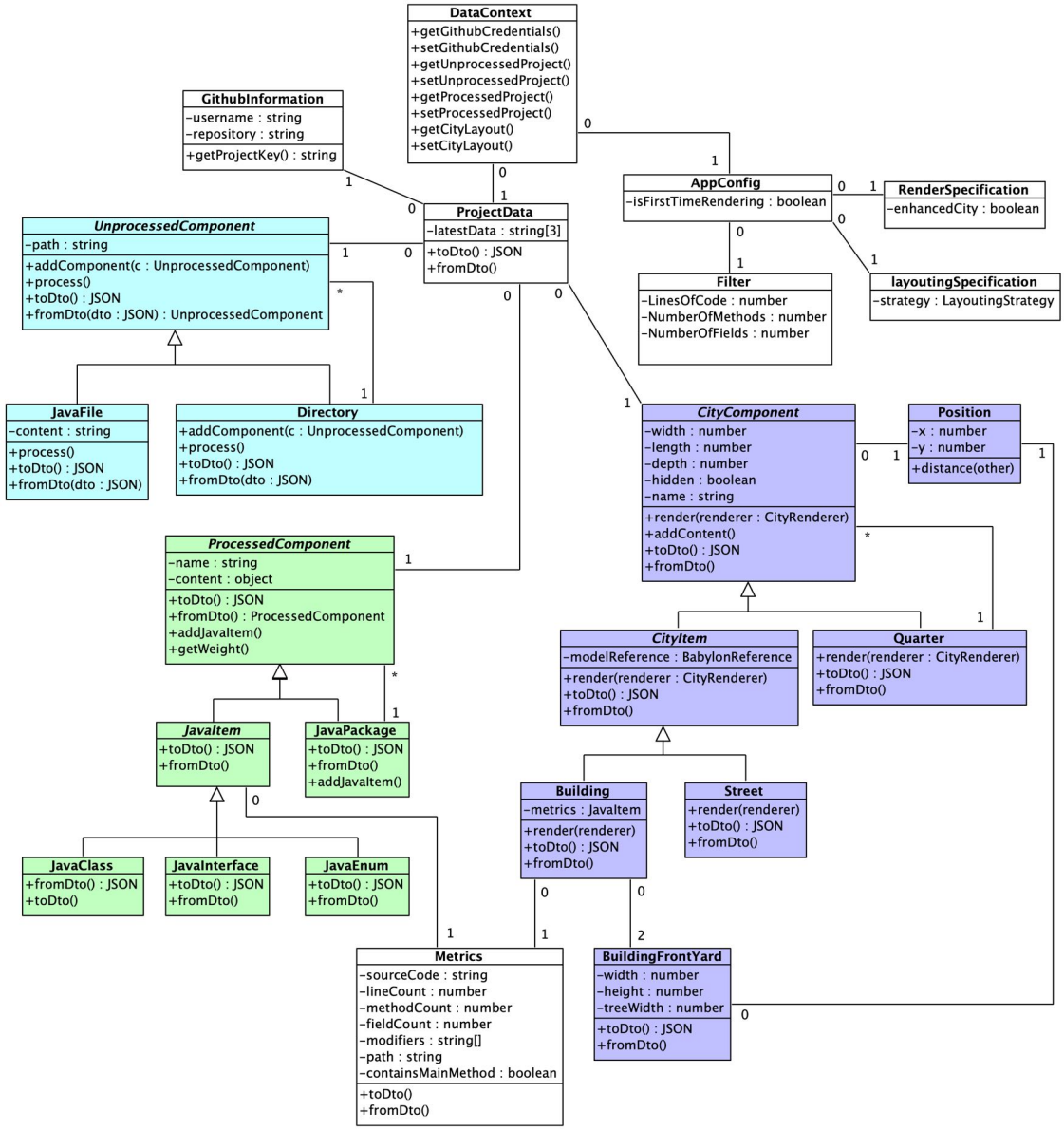


Abbildung 5.10: Level 4 Klassendiagramm Model

Im `DataContext` werden die Stateübergreifenden Daten abgespeichert. `ProjectData` enthält die Projektdaten und wird in den verschiedenen States beschrieben. `UnprocessedComponent` enthält die rohen Projektdaten und wird vom *Data Retrieval* State befüllt. `ProcessedComponent` enthält die verarbeiteten Projektdaten und wird im *Code Parsing* State erstellt. `CityComponent` enthält die Abmasse der 3D Stadt und wird im *Generating Layout* State erstellt. Diese drei Datenstrukturen folgen alle dem Composite Pattern Ansatz. Konfigurationen werden in der `AppConfig` abgelegt. Dort werden die Spezifikationen abgespeichert, wie die Stadt gebaut und die Szene gerendert werden soll.

DTO Die `toDto` und `fromDto` Methoden sollen die Daten serialisieren beziehungsweise wieder in das typisierte Model konvertieren. *DTO* steht dabei für *Data Transfer Object*. So wird die parallele Entwicklung über beide Projektmitglieder vereinfacht. Jeder State, der auf dem Arbeitsresultat des vorherigen States basiert, kann während der Entwicklung temporär ein *DTO* einlesen. Man könnte gewisse Berechnungen so auch einfacher auf einen externen Server auslagern oder bei Bedarf die Arbeitsresultate der States persistieren. Auch das Testing profitiert davon, da man einen abgespeicherten *DTO*-String wieder einlesen kann.

Die Struktur der `UnprocessedComponent` sieht wie folgt aus:

```
{
  directory: {
    path: string,
    content: [
      {
        javaFile: {
          path: string,
          content: string
        }
      }
      ...
    ]
  }
}
```

Die `ProcessedComponent` sieht folgendermassen aus:

```
{
  package: {
    name: string,
    content: [
      {
        class: {
          name: string,
          metrics: {
            sourceCode: string,
            lineCount: number,
            methodCount: number,
            fieldCount: number
            modifiers: string[],
            path: string,
            containsMainMethod: boolean,
          }
        }
      },
      {
        interface: {
          name: string,
          metrics: { ... }
        }
      },
      {
        enum: {
          name: string,
          metrics: { ... }
        }
```

```
      },
      ...
    ]
  }
}
```

Und noch ein Beispiel zur `CityComponent`:

```
{
  quarter: {
    name: string,
    hidden: boolean,
    depth: number,
    dimension: {
      width: number,
      length: number
    },
    position: {
      x: number,
      y: number
    }
    content: [
      {
        building: {
          name: string,
          hidden: boolean,
          depth: number,
          dimension: {
            width: number,
            length: number
          },
          position: {
            x: number,
            y: number
          },
          type: string,
          metrics: { ... },
          frontYards: [
            {
              width: number,
              height: number,
              position: {
                x: number,
                y: number,
              },
              treeWidth: number,
            },
            {
              ...
            }
          ]
        },
        street: {
          hidden: boolean,
          depth: number,
          dimension: {
            width: number,
            length: number
          },
```

```
            position: {
              x: number,
              y: number
            }
          }
        },
        ...
      ]
    }
  }
```

Rendering eines voranalysierten Projektes Die *DTOs* lassen die Möglichkeit offen ein voranalysiertes Projekt rendern zu lassen. So könnte man ein beliebiges *DTO* zur `CityComponent` mittels `fromDto()` konvertieren und danach direkt in den `RenderingState` wechseln, um die Stadt darzustellen. Würde der Benutzer dann die Gebäudesemantik verändern wollen, wäre das allerdings nicht möglich, da der `LayoutBuildingState` die `ProcessedComponent` nicht aus dem `DataContext` abholen kann, weil diese vorher von niemanden geschrieben worden ist. Alternativ könnte man anstatt nur die `CityComponent` zu hinterlegen, direkt alle nötigen Daten der Pipeline über `fromDto()` einlesen. Dann wäre es für den Benutzer wieder möglich die Gebäudeparameter zu ändern.

ProjectData Das Attribut `latestData` in der Klasse `ProjectData` ist ein String-Array der Länge 3. Index 0 repräsentiert die `UnprocessedComponent`, Index 1 die `ProcessedComponent` und Index 2 die `CityComponent`. Wird eine der drei Komponenten mit konkreten Daten abgefüllt, wird im `dataState` der entsprechende Index mit dem aktuellen Projektschlüssel beschrieben. So weiss `ProjectData` immer, welche Daten in den Komponenten aktuell sind und von den States weiterverarbeitet werden können.

Controller

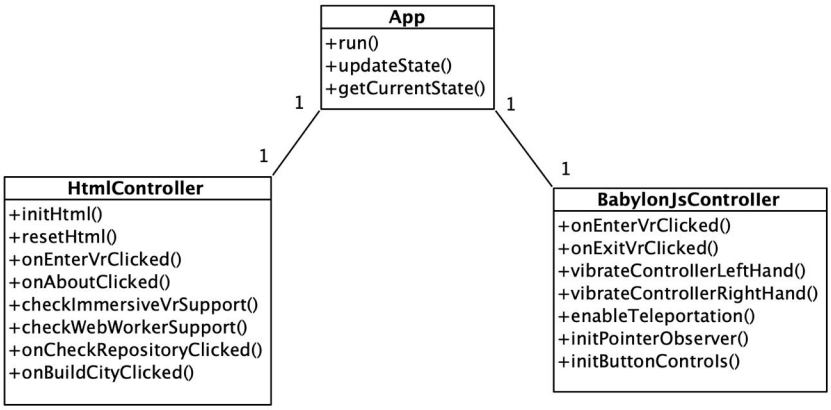


Abbildung 5.11: Level 4 Klassendiagramm InteractionController

Der *Controller* ist das C im MVC. Hier werden UI-Events vom HTML und von Babylon.js entgegengenommen und weiterverarbeitet. Die `initButtonControls` Methode im `BabylonJsController` dient zur Initialisierung der Buttons auf dem VR-Controller. Die `initPointerObserver` Methode registriert sich auf das Event, falls mit dem Controller Pointer ein Babylon.js Mesh selektiert wird.

5.2 Laufzeitansicht

5.2.1 State Machine

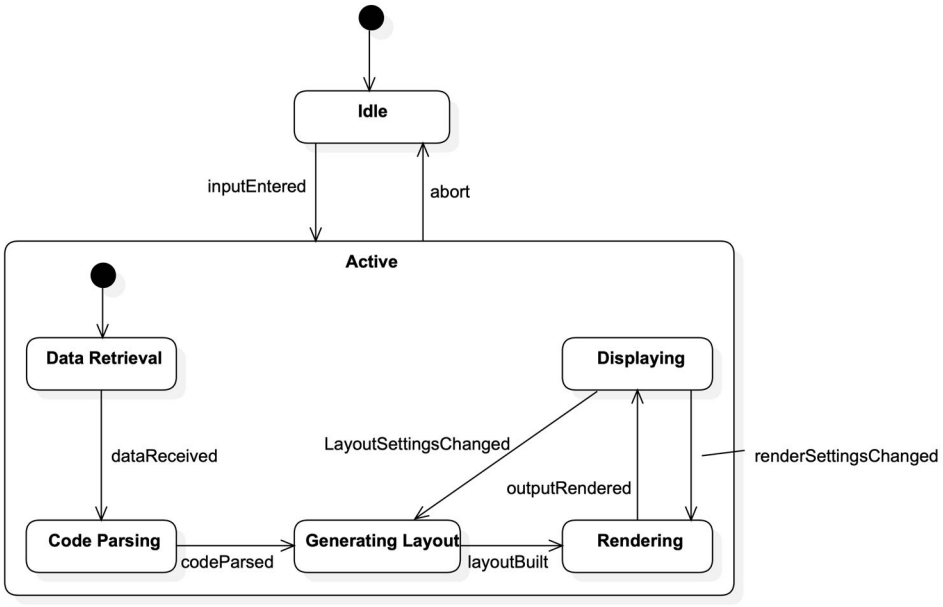


Abbildung 5.12: State Machine

Die Applikation entspricht einer State Machine. Der State *Idle* entspricht dem Zustand, wenn sich der User auf der Startseite der Webseite befindet. Sobald er sein Repository angegeben hat, befindet sich die Applikation im *Active*-State. Dort wird automatisch die Pipeline durchlaufen, um die Stadt zu erzeugen. Am Schluss kann der User im *Displaying*-State die Stadt betrachten. Ändert sich eine Einstellung, die das Layout betrifft, muss das Stadtlayout neu generiert werden. Das ist allerdings einer der optionalen Use Cases. Wenn sich eine Einstellung ändert, die das Rendering betrifft, wird die Stadt neu gezeichnet.

5.2.2 Abläufe

Nachfolgend werden die wichtigsten Abläufe über Sequenzdiagramme visualisiert. Wie bereits im vorherigen Abschnitt erwähnt, sollen diese Diagramme einen Überblick über das System vermitteln und bilden deshalb nur die wichtigsten Aufrufe ab.

Umsetzung der State Machine

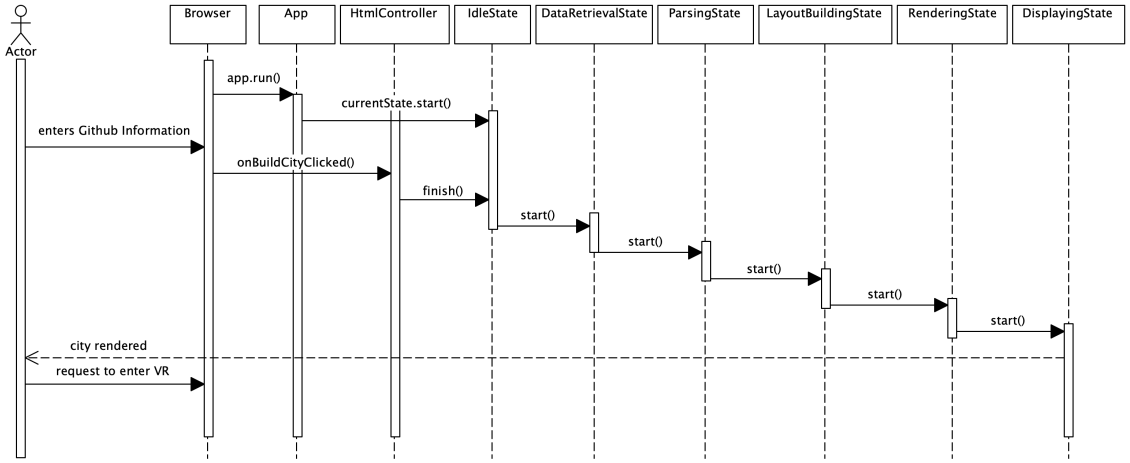


Abbildung 5.13: Sequenzdiagramm Umsetzung der State Machine

Der Browser fährt die ganze Applikation hoch. Der initiale State der Applikation ist der `IdleState`. Wenn der User seine Informationen zum GitHub Repository angegeben hat, wird die Pipeline ausgeführt, bis die Stadt dem Benutzer am Schluss angezeigt werden kann. Wird ein State gestartet, führt er seine Aufgaben durch. In den nachfolgenden Sequenzdiagrammen wird konkreter darauf eingegangen. Ist die Aufgabe innerhalb eines States abgeschlossen, initiiert er selbständig den State Change. Jeder State weiss für sich selber, welcher State folgt. Falls ein State während seiner Berechnung unterbrochen wird, springt die `Application` wieder in den `IdleState` zurück. Bei jedem State Change wird die `Application` benachrichtigt. Zur Übersichtlichkeit wurde bei den States die Ausführung der Aufgaben und die Benachrichtigung an die `Application` weggelassen.

Abfrage der GitHub REST API

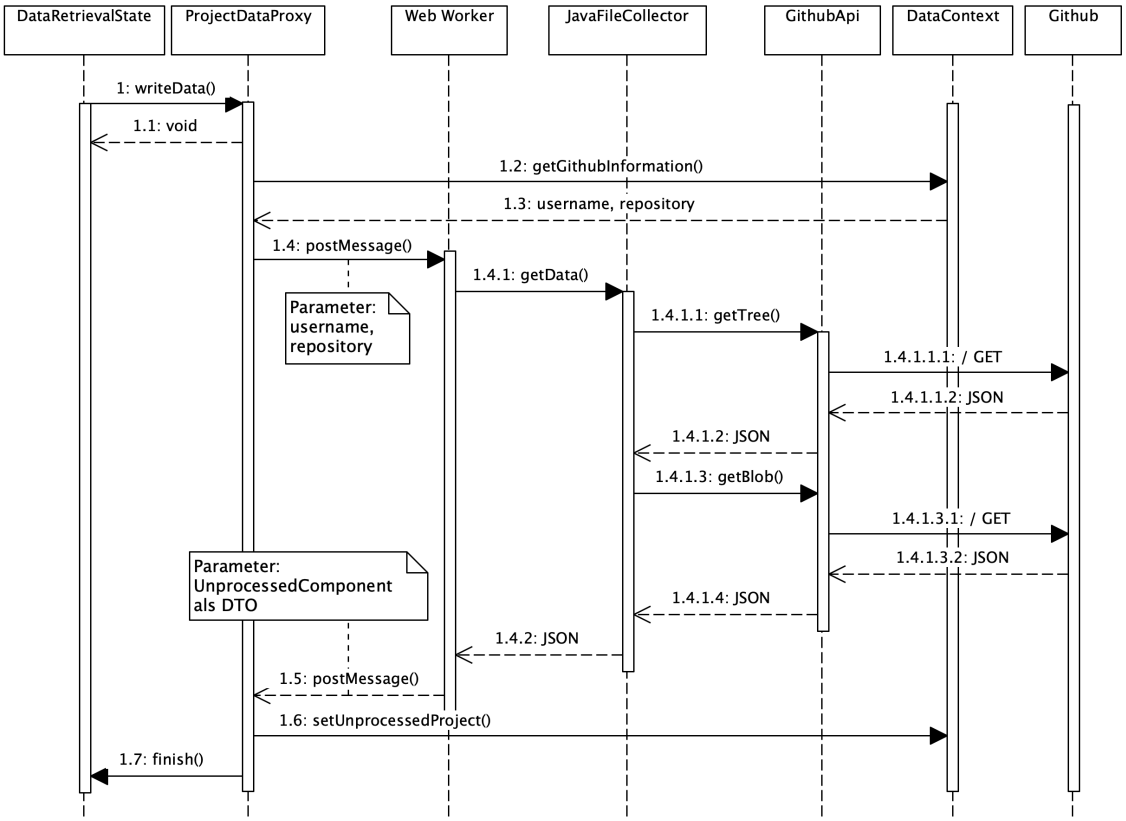


Abbildung 5.14: Sequenzdiagramm Abfrage GitHub REST API

Im `DataRetrievalState` wird das ganze Projekt von der GitHub REST API abgefragt. Der `ProjectDataProxy` startet einen Web Worker, der sich um die Abfrage auf einem anderen Thread kümmert. So bleibt das GUI auch bei grösseren Projekten reaktionsfähig. Dem Web Worker werden der Username und das Repository als Argumente übergeben. Der Web Worker instanziert dann den `JavaFileCollector`, der die rekursiv die Abfragen startet. Über `getTree` wird ein Directory ohne dessen Inhalt geladen. Über `getBlob` wird der Inhalt der gefundenen Javafiles abgefragt. Der `JavaFileCollector` baut sich das Resultat als eine `UnprocessedComponent` zusammen. Ist er damit fertig, konvertiert er das Resultat in ein *DTO*, damit der Web Worker das Resultat dem Main Thread zurückschicken kann. Der Main Thread liest das Resultat über `fromDTO` wieder in eine `UnprocessedComponent` Datenstruktur ein und speichert sein Resultat im `DataContext`. Abschliessend wird die `finish`-Methode des States aufgerufen, damit dieser den nächsten State initiieren kann.

Analyse des Projektes

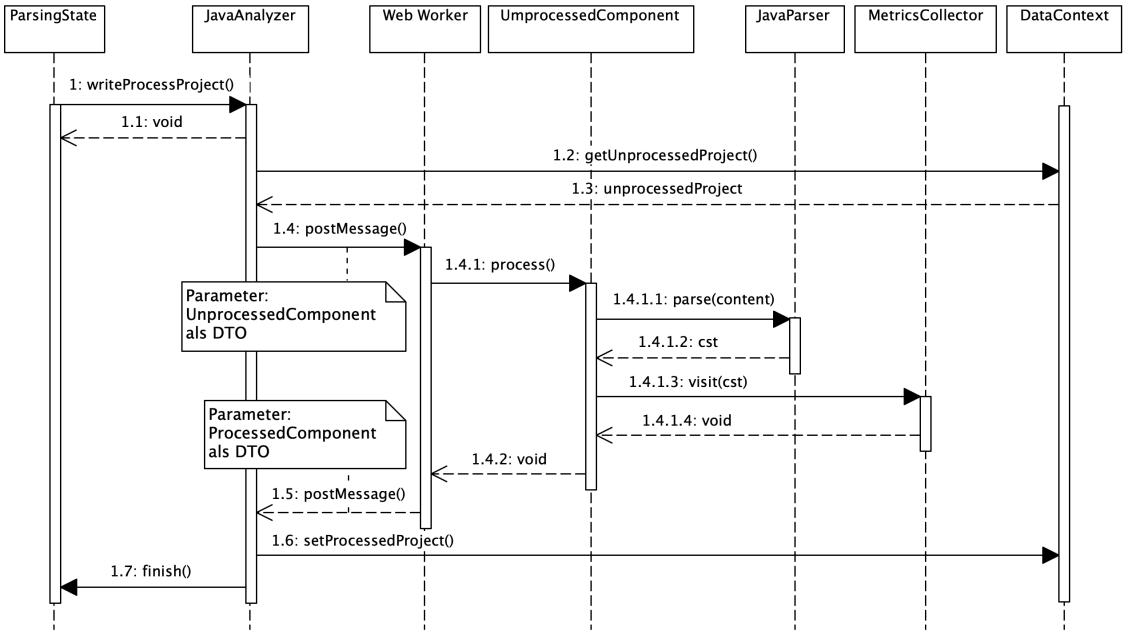


Abbildung 5.15: Sequenzdiagramm Analyse des Projektes

Nachdem die rohen Projektdaten von der GitHub REST API eingelesen wurden, wird das Projekt analysiert und die Metriken ausgelesen. Der `JavaAnalyzer` übergibt einem Web Worker die `UnprocessedComponent` Datenstruktur als *DTO*. Der Web Worker empfängt das *DTO* und liest es über die `fromDTO` Methode wieder in eine `UnprocessedComponent` Struktur ein. Da die `UnprocessedComponent` nun einer hierarchischen Composite Datenstruktur entspricht, muss nur noch auf der oberste Komponente die `process` Methode aufgerufen werden und die Datenstruktur traversiert sich selber. Dabei wird als Seiteneffekt das Resultat beschrieben, das einer `ProcessedComponent` entspricht. Das Resultat wird während der Traversierung jeweils als Parameter weitergereicht. Schlussendlich konvertiert der Web Worker das Resultat wieder in ein *DTO* und schickt es dem Main Thread. Dieser beschreibt den `DataContext` und startet den nächsten State.

Generierung des Stadtlayouts

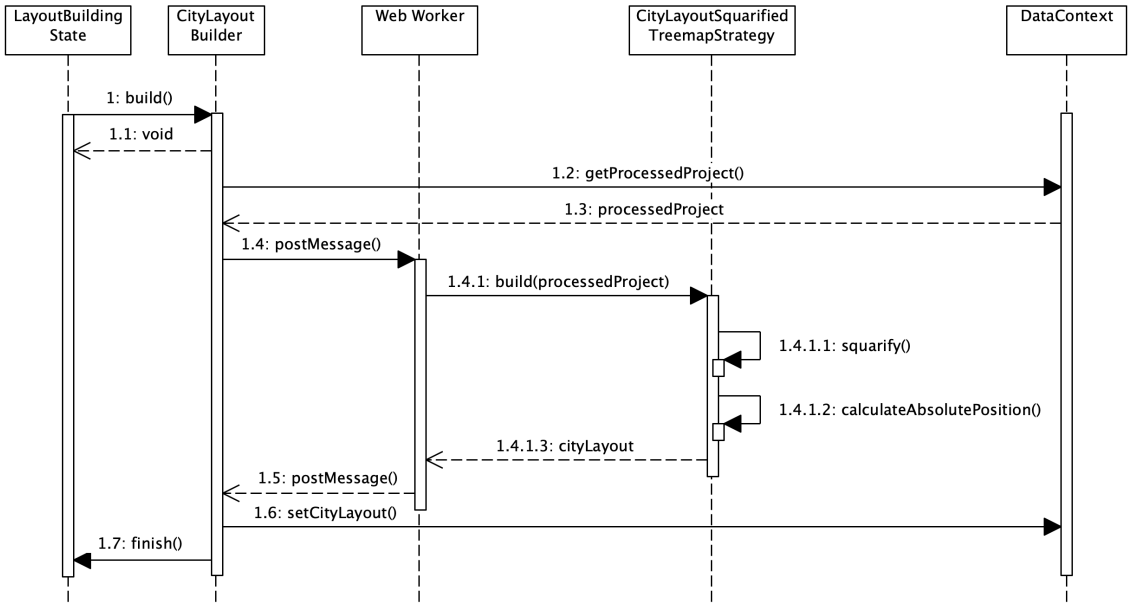


Abbildung 5.16: Sequenzdiagramm Generierung des Stadtlayouts

Die Abfolge beim Generieren des Stadtlayouts ist ähnlich wie vorher. Die Kommunikation zwischen Web Worker und Main Thread läuft wieder über die *DTOs*. In der Abbildung wurde nur die `CityLayoutSquarifiedTreemapStrategy` abgebildet, da diese Strategie die Hauptstrategie ist. Die `CityLayoutSquarifiedTreemapStrategy` besteht aus hauptsächlich zwei rekursiven Methoden und mehreren Hilfsmethoden.

Rendering der Stadt

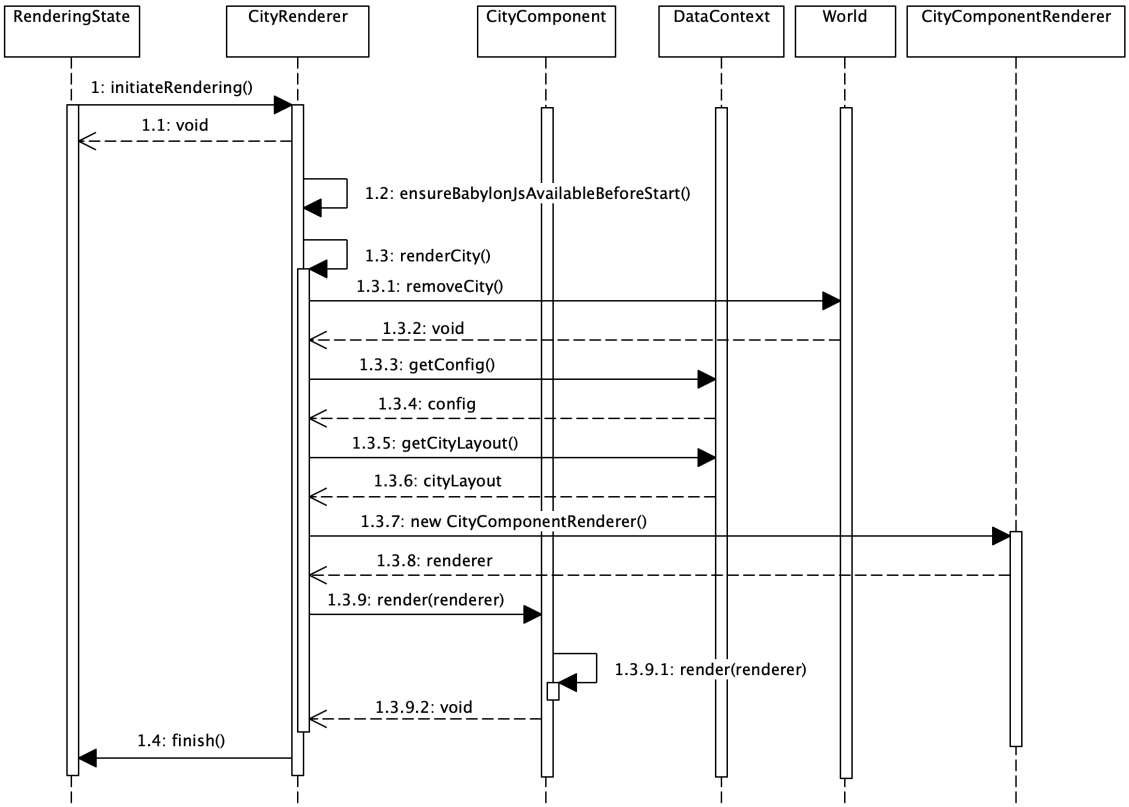


Abbildung 5.17: Sequenzdiagramm Rendering der Stadt

Das Babylon.js Framework wird mittels Lazy Loading nachgeladen. Bevor die Stadt gerendert werden kann, muss sichergestellt werden, dass Babylon.js bereits nachgeladen wurde. Danach kann das Rendering begonnen werden. Zuerst wird die alte Stadt gelösch. Über `getConfig` wird die aktuelle Renderkonfiguration ausgelesen. Mit `getCityLayout` wird die Datenstruktur abgefragt, die die Abmasse der Stadt für das Rendering enthält. Der `CityComponentRenderer` wird nun über die Constructor Injection so instanziert, dass er der Konfiguration entspricht. Dieser `CityComponentRenderer` wird nun als `renderer` der `CityComponent` Datenstruktur übergeben. `CityComponent` entspricht einer Composite Datenstruktur und traversiert automatisch auch durch alle Kinder. Am Schluss steht die gesamte Stadt dem Benutzer zur Verfügung und kann nun damit interagieren.

Nutzer ändert die Konfiguration

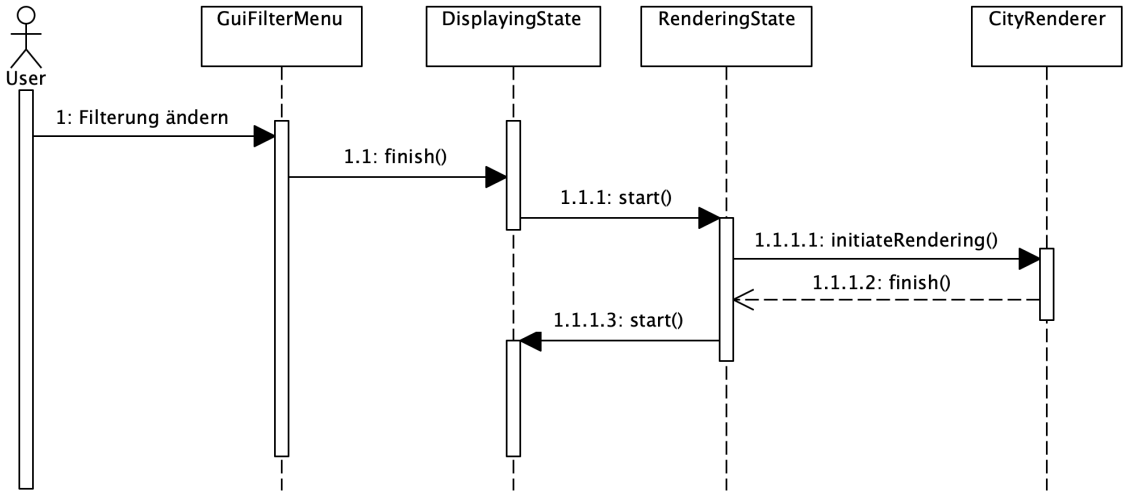


Abbildung 5.18: Sequenzdiagramm Nutzer ändert die Konfiguration

Dieser Abschnitt soll ein Beispiel aufzeigen, was passiert, wenn der Benutzer die Konfiguration ändert. Passt er beispielsweise den Filter der Stadt an, wird automatisch vom `DisplayingState` auf den `RenderingState` gewechselt. Dieser zeichnet die Stadt neu und wechselt dann wieder zurück auf den `DisplayingState`.

5.2.3 Performance

Web Workers

Die Analyse des Projektes kann je nach dessen Grösse relativ viel Zeit in Anspruch nehmen. Damit das GUI nicht einfriert, muss man die Berechnungen auf einen Worker Thread auslagern. Im Browser werden deshalb die Web Workers verwendet.

Tree Shaking

Bei der Auslieferung der Files wird von Tree Shaking[4] Gebrauch gemacht. Dadurch bündelt Webpack nur die Komponenten, die auch wirklich gebraucht werden. So kann die ausgelieferte Filegrösse stark eingeschränkt werden.

Lazy Loading

Trotz Tree Shaking ist die Applikation zu gross, um alles miteinander zu Beginn auszuliefern. Besonders die Babylon.js Library benötigt durch seine Grösse je nach Internetgeschwindigkeit zu viel Zeit in der Auslieferung. Um die User Experience zu verbessern wird deshalb vom Lazy Loading Feature[5] von Webpack Gebrauch gemacht. Ausgewählte Teile der Applikation werden so erst dann ausgeliefert, wenn sich benötigt werden.

[4] Technik zur Eliminierung von Dead Code: https://webpack.js.org/guides/tree-shaking/
[5] Performance Optimierung: https://webpack.js.org/guides/lazy-loading/

Leistung der Brille

Es wird ausnahmslos alles auf dem VR Headset berechnet. Das heisst, dass sich das Headset nebst dem Rendering der 3D Welt, auch um den Abruf der Daten von der GitHub REST API, sowie der Analyse des Java Codes und der Berechnung des Stadtlayouts kümmern muss. Für eine optimale User Experience setzt das speziell bei grösseren Java Projekten voraus, dass das VR Headset über genügend Rechenkapazität verfügt. Die Spezifikationen der Headsets, die in der Enwticklung eingesetzt werden, sind im Kapitel 6.1.1 beschrieben.

Im Falle der Oculus Quest 2 muss sich der Prozessor allerdings nicht verstecken und ist mit einer Basistacktrate von 1.8GHz durchaus dafür geeignet auch grössere Projektanalysen durchzuführen. Um eine endgültige Aussage betreffend der Performance treffen zu können, müssen jedoch auch einen Haufen anderer Faktoren berücksichtigt werden. Es soll deshalb durch Performancetests herausgefunden werden, wie gut sich die Hardware der aktuellen Brillen für die Analyse eignet.

Auch das Rendering der Szene kann je nach Projektgrösse und Hardware ein Problem darstellen. Über Versuche soll herausgefunden werden, wie gut die Hardware mit grösseren Städten klarkommt[6].

Optimierungsmöglichkeiten

Alternativ zur Analyse direkt auf der Brille könnte man die Analyse auf einen leistungsfähigeren Server auslagern. Umsetzen könnte man dies zum Beispiel, indem man einen alternativen State hinzufügt, der die Analyse auf dem Server initiiert und auf das Resultat wartet. Damit der Fokus der Arbeit auf den VR Themen aber nicht verloren geht, wurde darauf verzichtet.

Im aktuellen Design werden alle Schritte der Analyse sequentiell ausgeführt. Zum Beispiel beginnt die Analyse des Java Projektes erst nachdem das Projekt komplett von GitHub abgerufen wurde. Hier wäre es denkbar die Analyse der einzelnen Files zu starten, sobald sie runtergeholt wurden.

5.3 Deployment

Hier werden die allgemeinen Konzepte des Deployments erläutert. Wie das Continuous Deployment aufgesetzt ist, wird im Kapitel 6.1.3 beschrieben.

[6] Die Resultate befinden sich im Kapitel A

5.3.1 Produktion

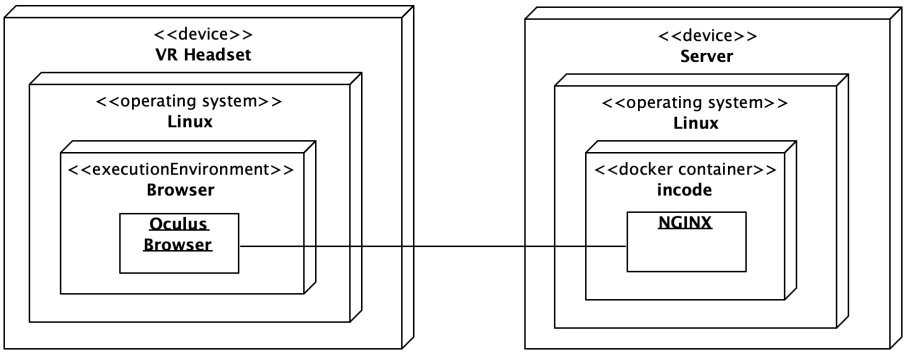


Abbildung 5.19: Deployment Produktion

Abbildung 5.19 beschreibt das Deployment für die Produktion. Auf dem virtuellen Server läuft ein Linux. Dort wird der Docker Container mit dem statischen Fileserver ausgeführt. Die Client Seite wurde hier konkret anhand einer Oculus Quest 2 Brille beschrieben. Diese Brille läuft mit Android und der Standard Browser dort heisst Oculus Browser.

5.3.2 Entwicklung

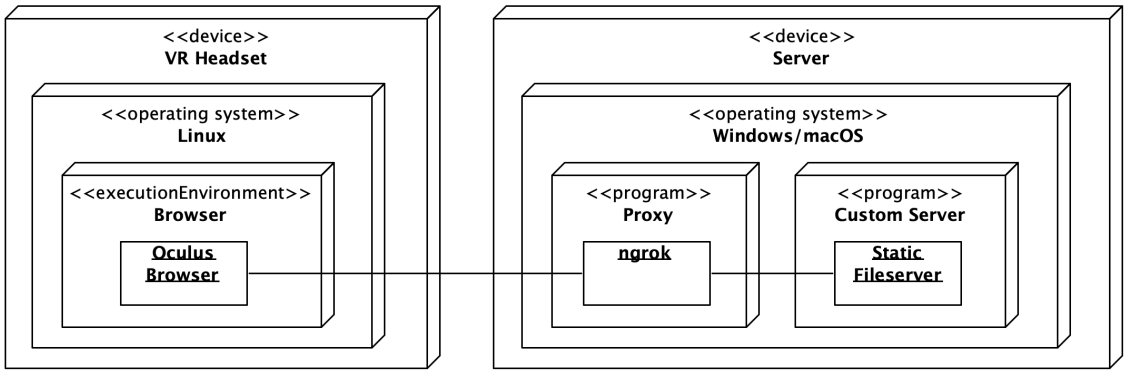


Abbildung 5.20: Deployment Entwicklung 1

Im Entwicklungssetup nach Abbildung 5.20 sind Client und Server wieder getrennt. Es muss deshalb eine HTTPS-Verbindung aufgebaut werden. Um zu umgehen, dass jeder Entwickler ein Zertifikat erstellen muss, wird eine Alternative mit einem Proxy verwendet. Ngrok macht den lokalen Fileserver für das öffentliche Internet zugänglich. Dabei erstellt er einen HTTPS Tunnel. Das VR Headset kann dann die Files verschlüsselt empfangen.

Als Alternative kann man den lokalen statischen Fileserver auch so konfigurieren, dass er direkt eine HTTPS Verbindung zur Verfügung stellt. Bei Webpack kann das in der Konfiguration eingestellt werden.

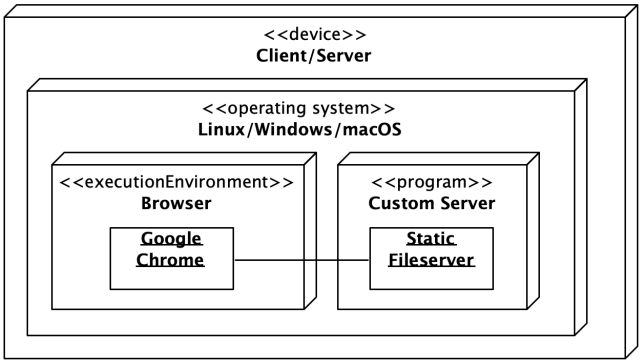


Abbildung 5.21: Deployment Entwicklung 2

Bei der Entwicklung nach Abbildung 5.21 ist kein Aufbau einer HTTPS Verbindung nötig, da bei localhost Adressen auch unverschlüsselte Verbindungen akzeptiert werden. Chrome bietet sich als Browser gut in der lokalen Entwicklung an, da über eine Browserextension ein WebXR Device emuliert werden kann.

5.3.3 Bundling

Die statischen Files werden vor der Auslieferung beim produktiven Deployment gebündelt. Dazu wird Webpack[7] verwendet. Während der lokalen Entwicklung wird das Bundling vom Webpack DevServer[8] übernommen. Dieser lädt auch automatisch die Webseite neu, falls sich Änderungen in den Files ergeben.

5.3.4 HTTPS

Das WebXR Device API Feature ist nur in einem sicheren Kontext verfügbar. Konkret heisst das hier, dass während beim Deployment ein Zertifikat benötigt wird, um HTTPS-Verbindungen anzubieten. Für automatische gratis Zertifikate bietet sich dafür Let's Encrypt an.

Umsetzung

Im Server läuft eine NGINX Instanz, welche neben den normalen Inhalten auch die Zertifikate ausliefern muss. Um überhaupt an die Zertifikate zu gelangen, muss NGINX erst Let's Encrypt anfragen. Ohne Zertifikate lässt sich NGINX aber gar nicht erst starten. Als Lösung wird deshalb zuerst ein dummy Zertifikat erstellt und dieses dann mit dem richtigen ersetzt[9]. Die NGINX Instanz läuft in einem Docker Dontainer. Damit die Zertifikate beim runterfahren des Containers nicht gelöscht werden, wird es über ein Ordner Mapping auf den Server gespiegelt.

Automatische Erneuerung des Zertifikats

Das Zertifikat ist nur für 90 Tage gültig. Das Zertifikat muss dann erneuert werden. Das geschieht am besten automatisch. Der Server wird deshalb so konfiguriert, dass er alle 12 Stunden überprüft, ob eine Erneuerung nötig ist.

[7] Open Source JavaScript Bundler https://webpack.js.org
[8] Siehe: https://webpack.js.org/configuration/dev-server/
[9] Für die HTTPS Verbindung wird diese Anleitung befolgt: https://pentacent.medium.com/nginx-and-lets-encrypt-with-docker-in-less-than-5-minutes-b4b8a60d3a71

Zusammenfassung

Dieses Kapitel veranschaulichte die Softwarearchitektur der Software City. Ausführungen wurden dabei hauptsächlich durch UML-Diagramme illustriert. In der Bausteinansicht wurde die Architketur komponentenweise aufgeteilt. Die Laufzeitansicht beschäftigte sich mit dem Verhalten der Applikation zur Laufzeit. Im letzten Abschnitt wurde das Deployment beschrieben.

Kapitel 6

Umsetzung

Übersicht

Der Abschnitt Infrastruktur behandelt den Aufbau der GitLab Infrastruktur und enthält eine Beschreibung der Umsysteme. Danach wird festgelegt, wie die Entwicklung mit einem VR-Headset konkret aussehen wird. Zum Schluss wird noch die das Design der Andwendung in einer Designdokumentation geplant.

6.1 Infrastruktur

6.1.1 Beschreibung der Hardware

VR Headsets

Entwickelt wird mit der Oculus Quest und der Oculus Quest 2. Nachfolgend werden die wichtigsten Eigenschaften dieser Brillen aufgelistet.

Tabelle 6.1: Oculus Quest: Spezifikationen

Spezifikation	**Beschreibung**
Display	1600 x 1440 Pixel pro Auge
CPU	Qualcomm Snapdragon 835: • Cores: 8 • Taktrate: 2450 MHz
GPU	Adreno 650
Memory	4GB
Storage	128GB
OS	Android

Tabelle 6.2: Oculus Quest 2: Spezifikationen

Spezifikation	Beschreibung
Display	1832 x 1920 Pixel pro Auge
CPU	Qualcomm Snapdragon XR2 Platform: • Cores: 8 (6 Energieeffiziente und 2 Performance Cores) • Taktrate: 1.8GHz • Turbo Taktrate: 2.5GHz • Lithography: 7nm
GPU	Adreno 650 (integrierte Grafikeinheit im Snapdragon XR2)
Memory	6GB
Storage	128GB
OS	Android 10

Virtueller Server

Deployt wird auf dem Server, der die OST zur Verfügung stellt.

Tabelle 6.3: Virtueller Server: Spezifikationen

Spezifikation	Beschreibung
CPU	2 vCPU, Max. 2.2 GHz
Memory	4GB
Storage	50GB
OS	Linux: Ubuntu 20.04 LTS

6.1.2 Continuous Integration

Die integrierte CI in GitLab führt jedes mal, wenn etwas in das Repositoy gepusht wird, das `.gitlab-ci.yml` File aus. Nacheinander werden verschiedene Stages ausgeführt, welche sicherstellen, dass gewisse Kriterien erfüllt werden. Wenn das nicht der Fall ist "failt" die Pipeline und der Entwickler wird informiert.

fetch	Im ersten Stage werden alle Dependencies (npm Packages) heruntergeladen, die im Projekt nötig sind. Um Speicherplatz zu sparen, sind diese Module nicht direkt im Repository abgelegt.
build	Nachdem alle Abhängigkeiten installiert wurden, wird überprüft, ob das Projekt gebaut werden kann.
analyze	Im ersten Teil der Analyse prüft der Linter, ob überall die Regeln für einen sauberen Code eingehalten werden.
	Im zweiten Teil geht SonarQube nochmals durch den Code und sucht nach Code Smells.
test	Dieser Stage führt alle geschriebenen Unit-Tests aus und ist erfolgreich, wenn diese ohne Fehler durchlaufen.
	Zu den ausgeführten Test wird die Coverage über die Files der Businesslogik berechnet.

6.1.3 Continuous Deployment

Bei einem Push auf den production Branch findet ein automatisches Deployment auf den virtuellen Server statt. Folgende Schritte werden durchgeführt:

1. Es wird ein Docker Image gebaut und ins GitLab Container Registry[1] hochgeladen. Die vorherige Version wird überschrieben.

2. Die GitLab CI verbindet sich über SSH mit dem Deployment Server. Der laufende Docker Container wird runtergefahren, das alte Docker Image gelöscht und der Docker Container wieder hochgefahren. Dabei wird automatisch das neuste und aktualisierte Docker Image vom GitLab Container Registry abgefragt.

6.1.4 Verwendete Technologien

Nachfolgend werden die wichtigsten Technologien aufgeführt, die im Projekt verwendet werden. Die Lizenzen der verwendeten Node Packages sind in einer Tabelle im Anhang B.1 aufgeführt.

Tabelle 6.4: Verwendete Technologien

Technologie	**Beschreibung**
TypeScript	Typisiertes JavaScript: https://www.typescriptlang.org
npm	Um die nötigen Packages zu managen: https://www.npmjs.com
Docker	Für ein einfacheres Deployment: https://www.docker.com
NGINX	Für den statischen Fileserver: https://www.nginx.com
Webpack	Zur Bündelung, Optimierung und Minifizierung der Dateien: https://webpack.js.org
Babylon.js	Game Engine zur Darstellung der 3D Welt: https://www.babylonjs.com
Jest	Ermöglich die Definition von Unit und Integration Tests: https://jestjs.io
ESLint	Überprüft den statischen Source Code auf allgemeine Probleme: https://eslint.org
Semantic UI	Bietet vordefinierte UI Komponenten an. Erleichtert das Styling der Website: https://semantic-ui.com
SonarQube	Unterstützung & Sicherstellung der Codequalität: https://www.sonarqube.org/

6.1.5 GitHub REST API

Der Source Code für die Software City wird von der GitHub REST API abgefragt. Damit die Benutzer der Software City sich nicht mit Access Tokens rumschlagen müssen, werden der Einfachheit halber nur public Repositories unterstützt. Folgende GitHub REST APIs werden im Kontext der Applikation benötigt:

- Repositories API
- Git Database API

[1] GitLab Container Registry: https://docs.gitlab.com/ee/user/packages/container_registry/

Die Base URL entspricht für alle API Abfragen

```
https://api.github.com
```

Die folgenden Auflistungen sind bei weitem nicht vollständig. Es werden nur die Elemente aufgelistet, die für das Projekt relevant sind.

Repositories API

Über dieses API wird der aktuellste Commit auf dem Default Branch abgefragt. Für diesen Commit wird der Source Code analysiert und die Stadt gebaut. Dieses API würde theoretisch ausreichen, um den Inhalt der Files für den Source Code abzufragen, jedoch gibt es Limiten wie eine maximale Dateigrösse von 1MB. Aus diesem Grund wird die Git Database API verwendet um Inhalte abzufragen.

```
GET /repos/{owner}/{repo}/commits
```

Tabelle 6.5: GitHub Repositories API: Commits Parameter

Parameter	Typ	Ort	Beschreibung
owner	string	path	Owner vom GitHub Repository
repo	string	path	Name des GitHub Repositories
sha	string	query	SHA oder Branch ab welchem man die Commits auflisten will
per_page	integer	query	Anzahl Commits

Tabelle 6.6: GitHub Repositories API: Commits Response

Response	Beschreibung
commit.tree	Repository Snapshot zu diesem Commit. Dieser Snapshot kann nun rekursiv durchforstet werden.
parents	Array von Parent Commits. Kann benutzt werden, um ältere Commits abzufragen.

Git Database API

Diese API ermöglicht es den Inhalt des Repositories rekursiv abzufragen.

```
GET /repos/{owner}/{repo}/git/trees/{tree_sha}
```

Tabelle 6.7: GitHub Database API: Trees Parameter

Parameter	Typ	Ort	Beschreibung
owner	string	path	Owner vom GitHub Repository
repo	string	path	Name des GitHub Repositories
tree_sha	string	path	SHA des Trees

Tabelle 6.8: GitHub Database API: Trees Response

Response	Beschreibung
tree	Array der Files auf der aktuellen Ebene. Über den Parameter url eines Files kann der Subtree abgefragt werden

```
GET /repos/{owner}/{repo}/git/blobs/{file_sha}
```

Tabelle 6.9: GitHub Database API: Blob Parameter

Parameter	Typ	Ort	Beschreibung
owner	string	path	Owner vom GitHub Repository
repo	string	path	Name des GitHub Repositories
file_sha	string	path	SHA des Files

Tabelle 6.10: GitHub Database API: Blob Response

Response	Beschreibung
content	Inhalt des Blobs
encoding	Encoding des Blobs

6.2 Entwicklungsstrategien

Debugging mit dem VR-Headset

Um die Entwicklung einfacher zu gestalten, muss man auf die Developer Tools des Browsers zugreifen können. Der Oculus Browser in der Oculus Quest 2 bietet keine internen Debugging Möglichkeiten an, um in der Brille zugreifen können. Damit trotzdem Developer Tools verwendet werden können, bietet sicher folgender Workaround an[2]:

1. Das VR Headset über USB mit dem Computer verbinden.
2. Über den Command `adb devices` kann abgefragt werden, ob die Verbindung besteht.
3. Im Headset zur lokalen Website navigieren.
4. Innerhalb von Chrome zu `chrome://inspect/#devices` navigieren.
5. Bei `Discover USB devices` einen Hacken setzen.
6. Auf `Port forwarding` klicken.
7. Bei `Enable port forwarding` einen Hacken setzen.
8. In der Geräteliste unterhalb sollte das Gerät auftauchen. Über `inspect` können nun die Developer Tools gestartet werden.

[2]Diese Anleitung wurde folgender Seite entnommen: https://babiaxr.gitlab.io/tutorials/how_to_launch_and_debug_vr/

Errorhandling

Bei Fehlern werden definierte Error Objekte zurückgegeben. Falls möglich sollen Exceptions mit dem Keyword `throw` propagiert werden. Für eine bessere Übersicht über alle möglichen Errors wird auf Polymorphie gesetzt.

Dependency Injection

Services werden bei der Verwendung falls möglich über den Konstruktor injected. So wird die Testbarkeit des Source Codes sichergestellt.

Codequalität

Bei jedem Push ins GitLab Repository wird in der Pipeline die Codequalität überprüft. Tools wie Sonarqube, ESLint und Tests sollen die Qualität sicherstellen. Ausserdem muss jedes Feature erst einen Merge Request durchlaufen, bevor es endgültig in die Applikation kommt. Entwicklungsumgebungen können die ESLint Probleme direkt schon dem Entwickler anzeigen und auch beheben.

Zirkuläre Abhängigkeiten

Zirkuläre Abhängigkeiten gilt es grundsätzlich zu vermeiden, da so die Wartbarkeit der Codes stark abnehmen kann. Nichtsdestotrotz können Situationen auftauchen, in denen zirkuläre Abhängigkeiten nicht so einfach umgangen werden können. Dies kann beim Composite Pattern der Fall sein, wenn die Superklasse, die Childs kennen muss. Sind in einer zirkulären Abhängigkeit die Klassen auch noch in verschiedene ES6 Module aufgeteilt, können bei der Ausführung korrupte Module entstehen. Um dem entgegenzuwirken, wird in solchen Fällen das Internal Module Pattern angewendet. So kann die Reihenfolge festgelegt werden, wie die Module zu importieren sind. Somit können korrupte Imports umgangen werden.

Externe 3D-Modelle

Um die Stadt aufzuhübschen werden schönere 3D-Modelle nötig sein. Da es zu aufwändig wäre, diese alle selbst zu modellieren, werden sie von externen Quellen bezogen. Im Anhang B.2 sind die verwendeten Webseiten aufgelistet.

6.3 UI Design

6.3.1 Designdokumentation

Die VR-Anwendung ist für passende Hardware, wie ein VR-Headset, gedacht. Innerhalb eines solchen Headsets sieht das Browserfenster vergleichbar zu dem eines Desktop Systems aus. Das Design orientiert sich demnach am Desktop First Ansatz.

Farben

- Primary: #275082
- Secondary: #cec8b0
- Background: #FFFFFF
- Text: #000000

Dark Theme
Im Darktheme Mode werden Background- und Text-Color, sowie Primary- und Secondary-Farben vertauscht.

Logo

Abbildung 6.1: incode Logo

Regular

Abbildung 6.2: incode Schriftzug

Bold

Abbildung 6.3: incode Schriftzug (Bold)

Layout
Zwischen einem Container und seinem Content existiert immer ein Margin/Padding.

Design Library

Es wird SemanticUI als Library verwendet. Soweit möglich werden dessen vordefinierten Komponenten und Icons verwendet.

Font

Semantic UI verwendet Standardmässig sans-serif font *Lato*, welcher auch für die Applikation übernommen wird.

6.3.2 Wireframes

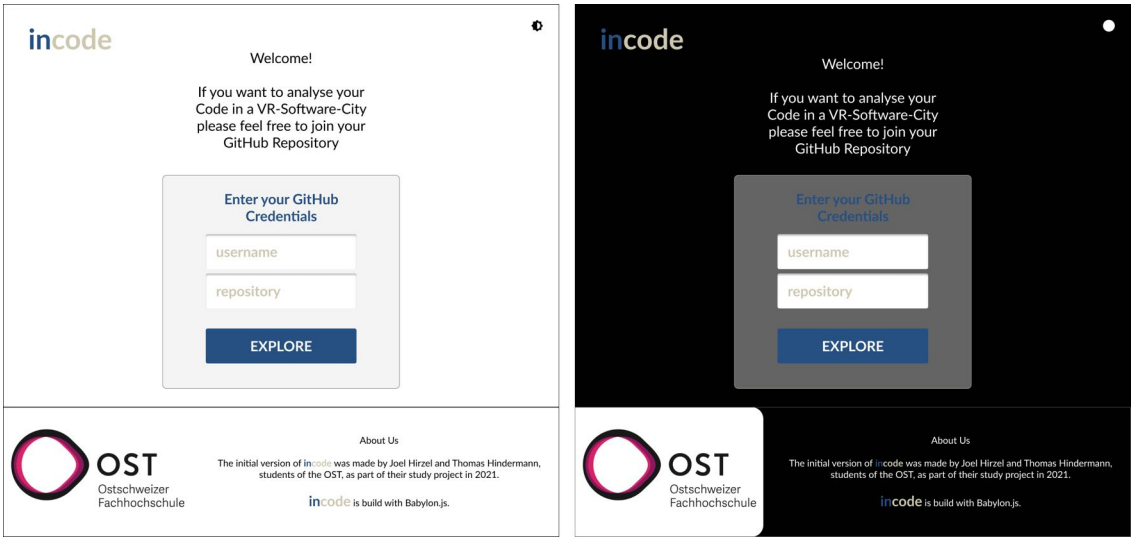


Abbildung 6.4: Wireframes: Homepage

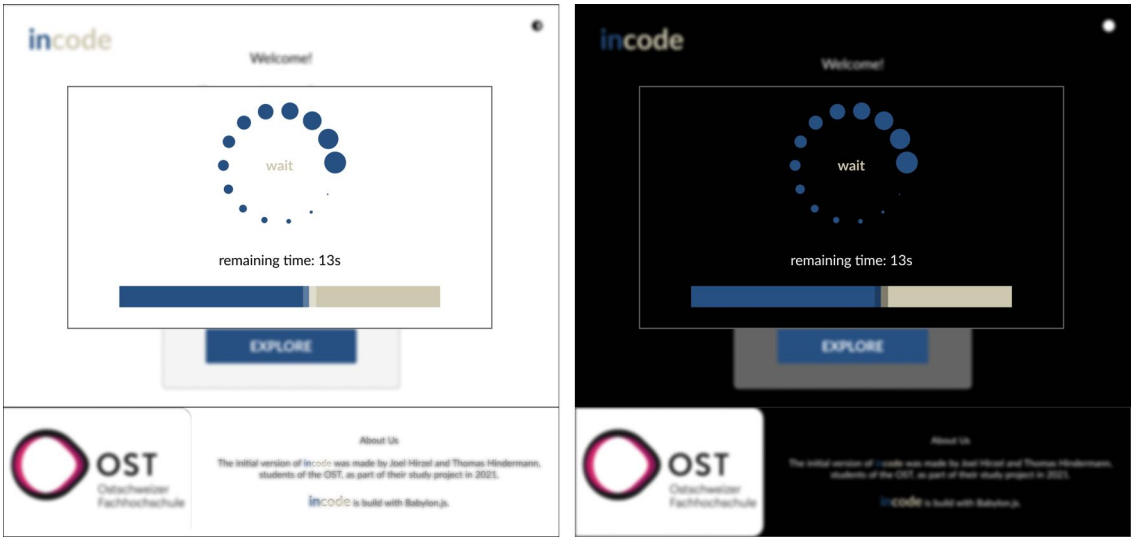


Abbildung 6.5: Wireframes: Loadingpage

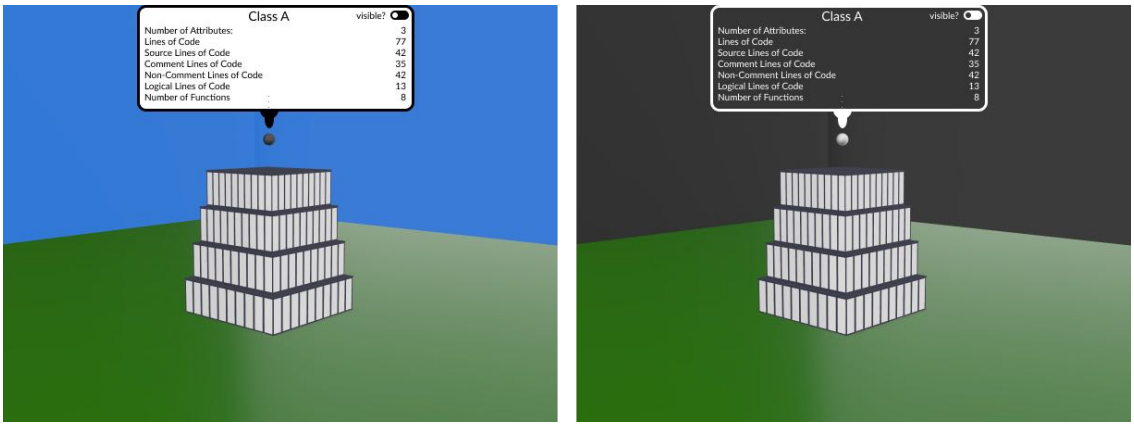


Abbildung 6.6: Wireframes: Komponenten

Abbildung 6.7: Wireframes: UI

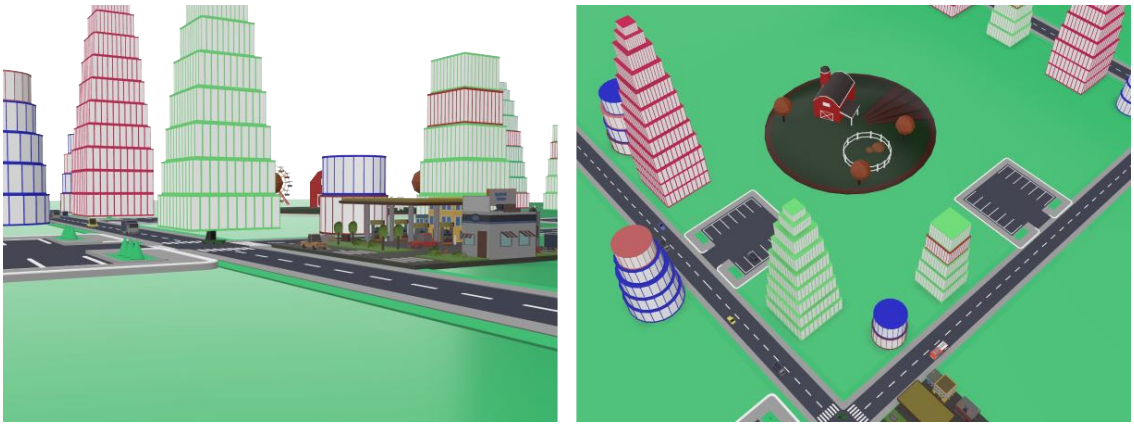


Abbildung 6.8: Wireframes: Symbolbilder der Stadt

Zusammenfassung

Das Kapitel Umsetzung befasste sich mit der Infrastruktur während der Entwicklung und den Entwicklungsstrategien. Auch das UI Design wurde mit Hilfe von Bildmaterial aufgezeigt.

Kapitel 7

Ergebnis

Übersicht

In diesem Kapitel wird das Endergebnis der Implementation anhand von Bildern vorgestellt. Der Abschnitt Qualitätssicherung beschäftigt sich mit der Qualität der Implementation, indem verschiedene Testergebnisse von System Tests, Usability Tests und anderen Nichtfunktionalen Tests präsentiert werden.

7.1 Resultat

Das Resultat der Construction Phase ist ein lauffähiger Prototyp. In dieser Sektion soll dieser Prototyp und dessen Funktionalitäten veranschaulicht werden. Von den geplanten Use Cases aus dem Kapitel 4 konnten fast alle nicht optionalen Use Cases umgesetzt werden. Zusätzlich wurde der optionale Use Case 6 (Kapitel 4.1.7) implementiert, der es dem Benutzer ermöglich sich in die Stadt hinein zu teleportieren. Die nachfolgenden Sektionen sind nach den implementierten Use Cases gegliedert und gehen genauer in die Details ein. Die Beispielstadt bezieht sich auf folgendes GitHub Repository:

- Username: amir650
- Repository: BlackWidow-Chess

Die Beschreibungen beinhalten Bildmaterial, um das Resultat zu veranschalichen. Weitere Bilder befinden sich im Anhang C.

7.1.1 Übersicht der Stadt

Dieser Abschnitt bezieht sich auf UC1 (Kapitel 4.1.2). Wenn der User ein gültiges GitHub Repository angegeben hat, wird ihm die Möglichkeit angezeigt, seine individuelle Stadt aus dem Software Projekt generieren zu lassen. Triggert er diesen Prozess, wird ihm grob über eine Progressbar der Berechnungsfortschritt angezeigt. Während des Analyseprozesses kann der Benutzer bereits in die VR Welt eintauchen.

Abbildung 7.1: Progress Fenster innerhalb der VR Welt

Sind die Berechnungen abgeschlossen, wird dem Benutzer das Resultat in Form einer Miniatur-Stadt präsentiert. Um die Leistungsschwächeren VR-Brillen nicht zu überfordern wird das initiale Rendering minimalistisch gehalten.

Abbildung 7.2: Initiale Stadt

Jedes Gebäude in der Stadt repräsentiert nun eine Klasse, ein Interface oder ein Enum. Die Gebäudedimensionen haben verschiedene Bedeutungen. So steht die Höhe für die Anzahl Zeilen und die Breite für die Anzahl Funktionen. Die Farbe passt sich an, je nachdem ob der darunterliegende Code eine Klasse, ein Interface oder ein Enum ist. Die Himmeltextur wird bewölkt, falls der Code nicht optimal formatiert wurde. Konkret, falls die durchschnittliche Zeilenbreite zu gross ist. Das geschieht allerdings nur bei Tageszeit.

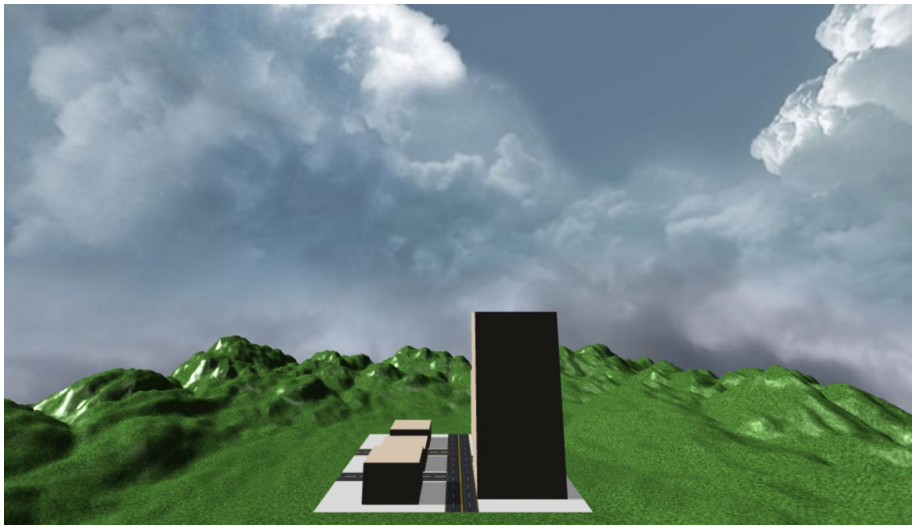


Abbildung 7.3: Bewölkter Himmel

Die Strassen zwischen den Gebäuden trennen die Komponenten besser ab. Je nachdem wie tief verschachtelt im Software Projekt diese Trennung ist, wird eine andere Strasse dargestellt. Die obersten Packages werden durch eine Hauptstrasse getrennt, während tiefer verschachtelte Komponenten durch Kieswege getrennt werden.

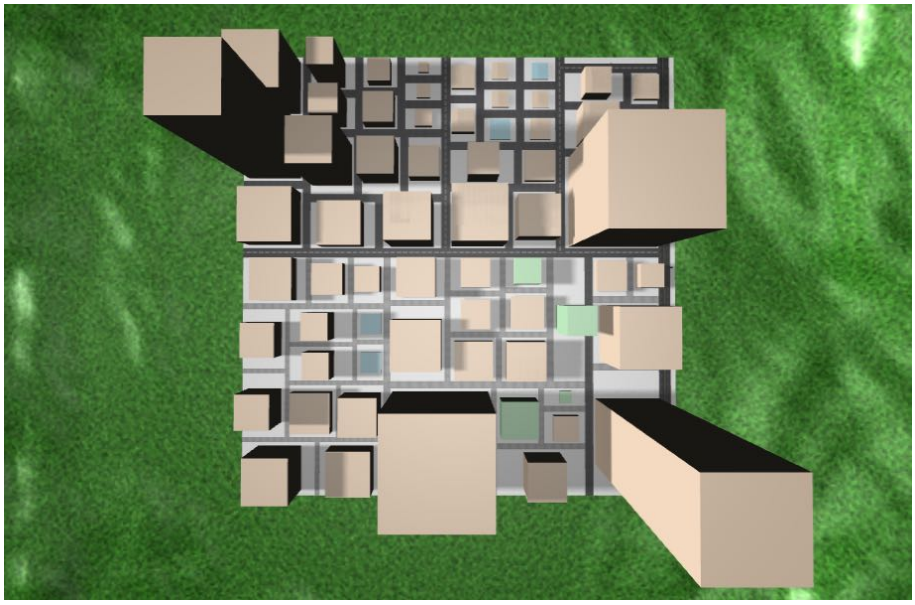


Abbildung 7.4: Stadt aus der Vogelperspektive

So lässt sich mehr oder weniger die Gliederung des Software Projekts erahnen. Die Totenköpfe auf den Gebäuden markieren Komponenten, die gewisse Schwellwerte überschreiten. Diese Schwellwerte sind konfigurierbar.

Selektiert man mit dem Controller ein Gebäude, kann man mehr über diese Komponente herausfinden. In einem neuen Fenster erscheinen die Metriken dieser Komponente.

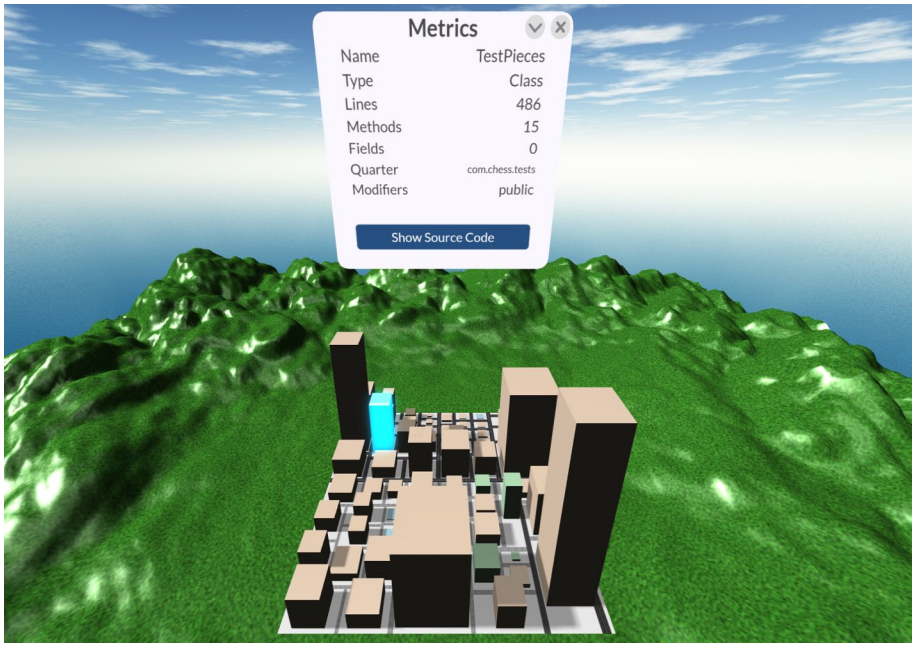


Abbildung 7.5: Gebäude Metriken

Auch der Source Code kann begutachtet werden.

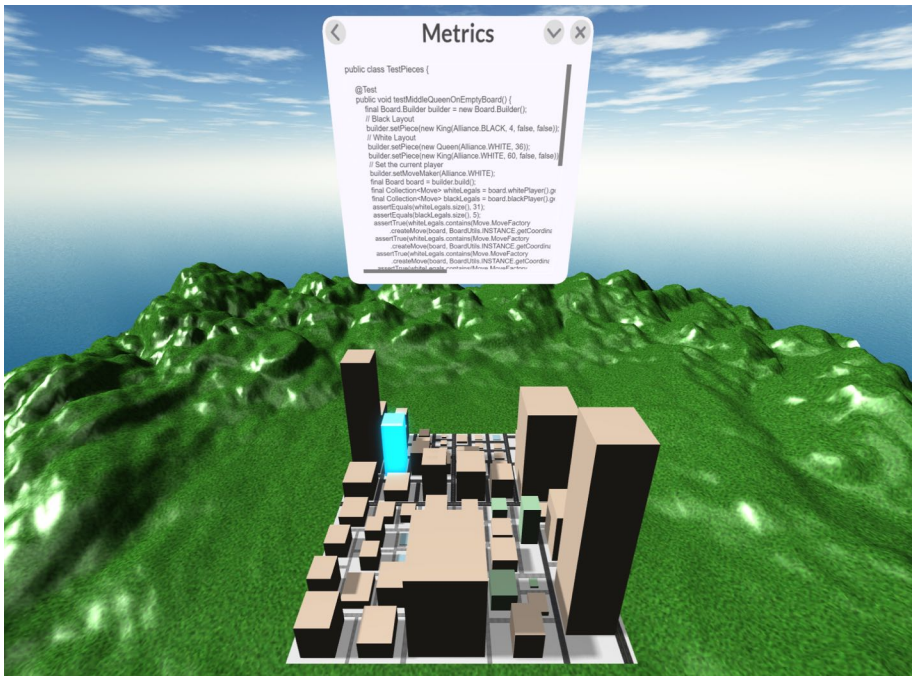


Abbildung 7.6: Source Code

Über die Suchfunktion können spezifische Gebäude gefunden werden. So muss nicht die gesamte Stadt manuell durchsucht werden. In der Abbildung 7.7 wird nach der Klasse Table gesucht. Bei einer erfolgreichen Suche wird das gefundene Gebäude für den Benutzer markiert.

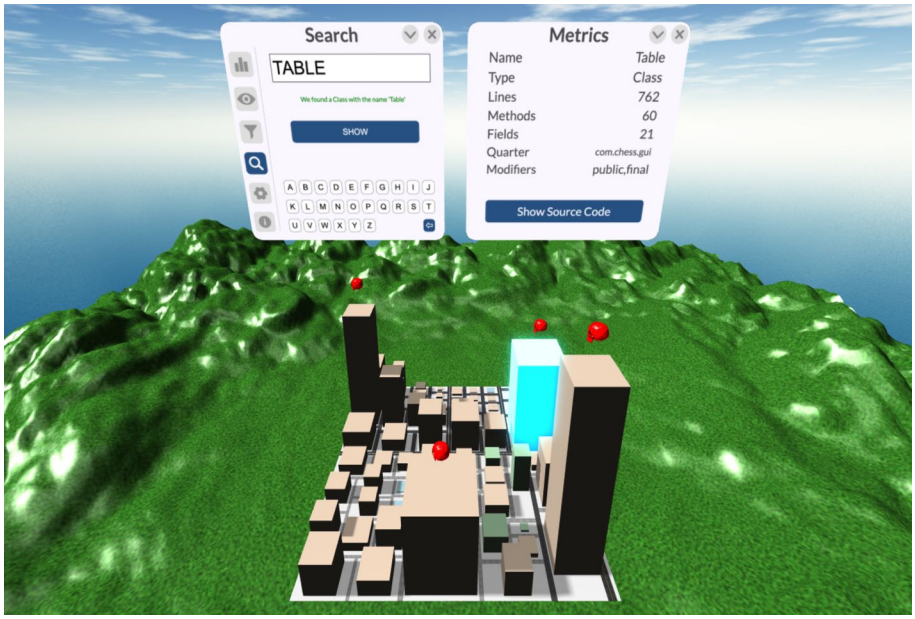


Abbildung 7.7: Suchfunktion

7.1.2 Filtern nach Eigenschaften

Dieser Abschnitt bezieht sich auf UC2 (Kapitel 4.1.3). Falls die Stadt zu überladen wirkt, können Teile auch rausgefiltert werden. In der Abbildung 7.8 wurde der Filter so eingestellt, dass nur Interfaces und Enums angezeigt werden.

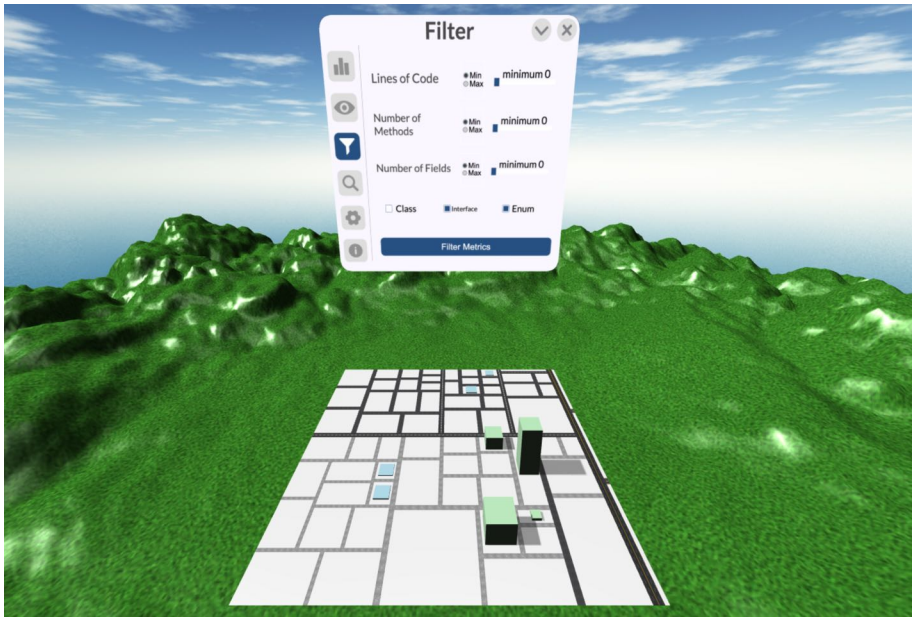


Abbildung 7.8: Filter Menu

7.1.3 Styling der Stadt oder Umgebung anpassen

Dieser Abschnitt bezieht sich auf UC3 (Kapitel 4.1.4). Im Menufenster findet der Benutzer ein Abschnitt, in dem das Aussehen der Stadt angepasst werden kann.

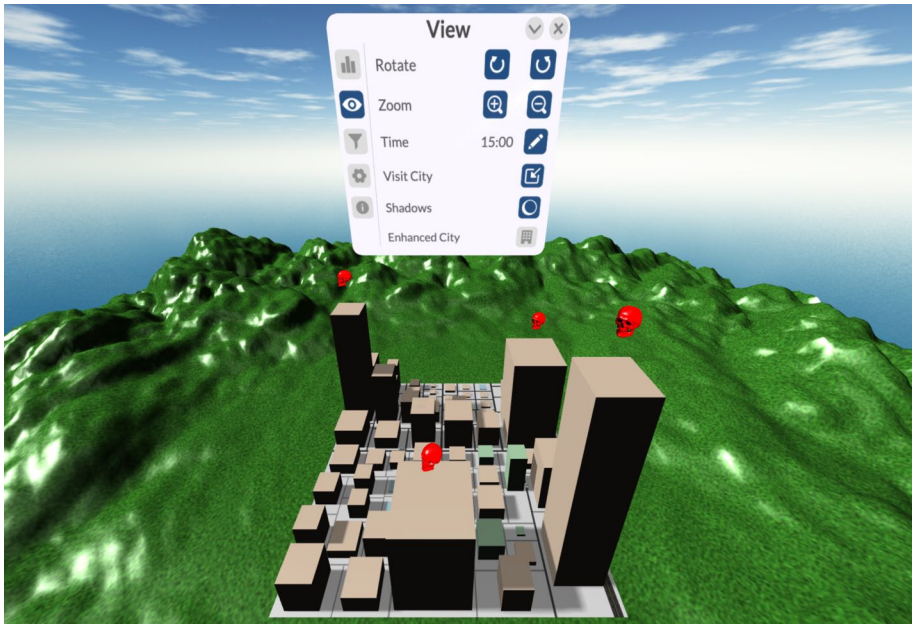


Abbildung 7.9: View Menu

Neben den Funktionen wie dem Rotieren und Zoomen der Stadt kann der Benutzer auch die Tageszeit einstellen. Selektiert der Benutzer diese Funktion, erscheint vor seiner rechten Hand eine Art Zeitrad. Inspiriert wurde dieser Ansatz durch Doctor Strange[1], in dem Zeit auf die selbe Art und Weise manipuliert wurde. Dreht man nun seine Hand, dreht sich auch dieses Rad und ändert somit die Zeit.

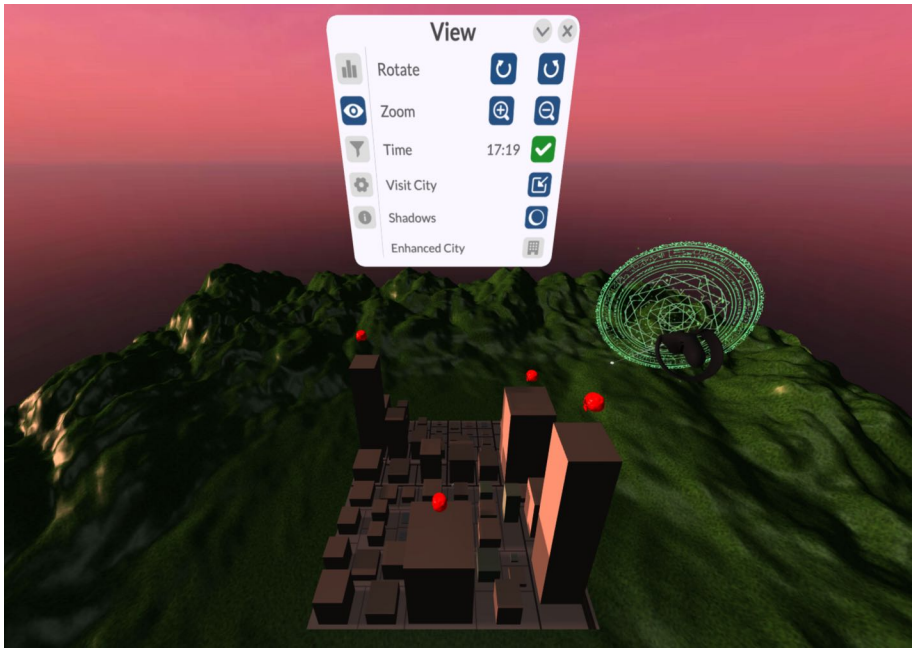


Abbildung 7.10: Stadt im Sonnenuntergang

[1] https://www.imdb.com/title/tt1211837/mediaindex/?ref_=tt_mv_close

So kann man die Stadt auch während dem Sonnenuntergang oder in der Nacht begutachten. Umgebungslicht und Schatten kommen von der Sonne, die sich automatisch der eingestellten Zeit anpasst. Beim initialen Rendering der Stadt passt sich der Sonnenstand automatisch der aktuellen Tageszeit an.

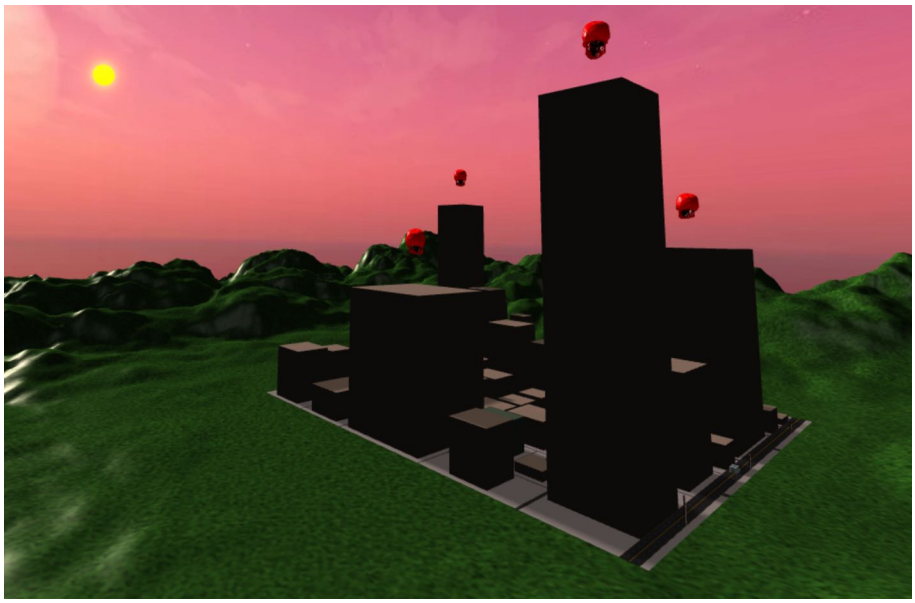


Abbildung 7.11: Sonnenstand

Des Weiteren hat der Benutzer hat die Möglichkeit, die Stadt interessanter zu gestalten. Aktiviert er den Menupunkt *Enhanced City*, werden richtige Gebäude inklusive Bäume, Autos und Fussgänger dargestellt.

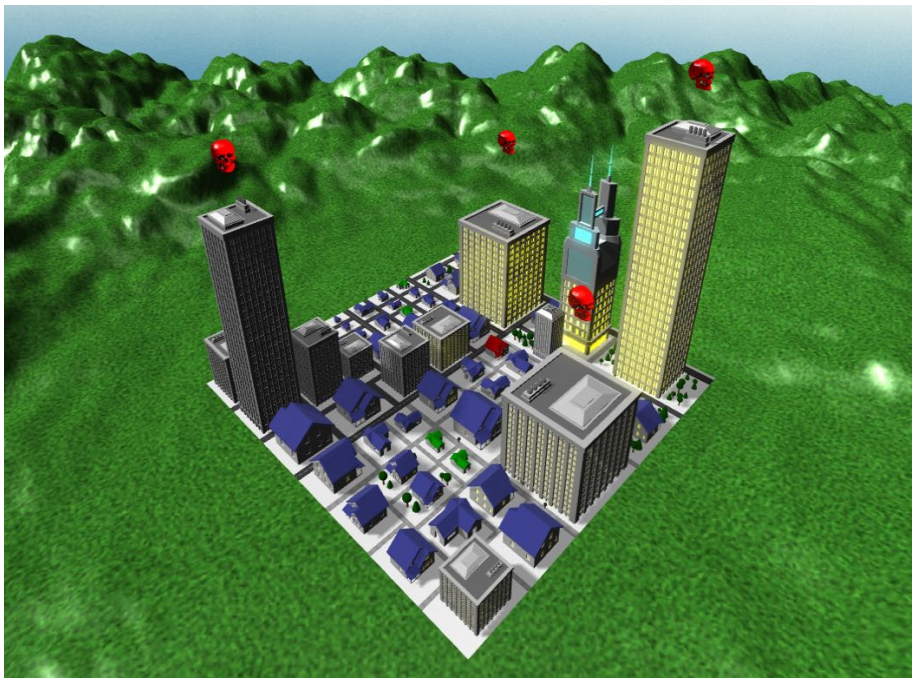


Abbildung 7.12: Verschönerte Stadt

Für die niedrigen Gebäude werden Einfamilienhäuser eigeblendet, während die grossen Gebäude über Wolkenkratzer abgebildet werden. In dieser Ansicht ist auch eine neue Metrik in der Stadt sichtbar. Die Helligkeit der Fenster steht nun für die Anzahl an Klassenvariablen. Im Bild 7.12 ist das Viertel links das Testing Viertel. Wie man sieht, sind die Fenster dort komplett dunkel, was auch Sinn ergibt, da man in Testklassen normalerweise weniger Klassenvariablen benötigt. Das Viertel rechts beinhaltet Gebäude mit der Businesslogik, die entsprechend auch einen State mit Hilfe von Klassenvariablen halten müssen.

Der spezielle Turm mit der blau leuchtenden Konstruktion auf dem Dach entspricht dem Einstiegspunkt in die Applikation. Diese Klasse enthält die Main Methode in Java. Auch interessant kann diese Variante der Stadt bei Nacht sein.

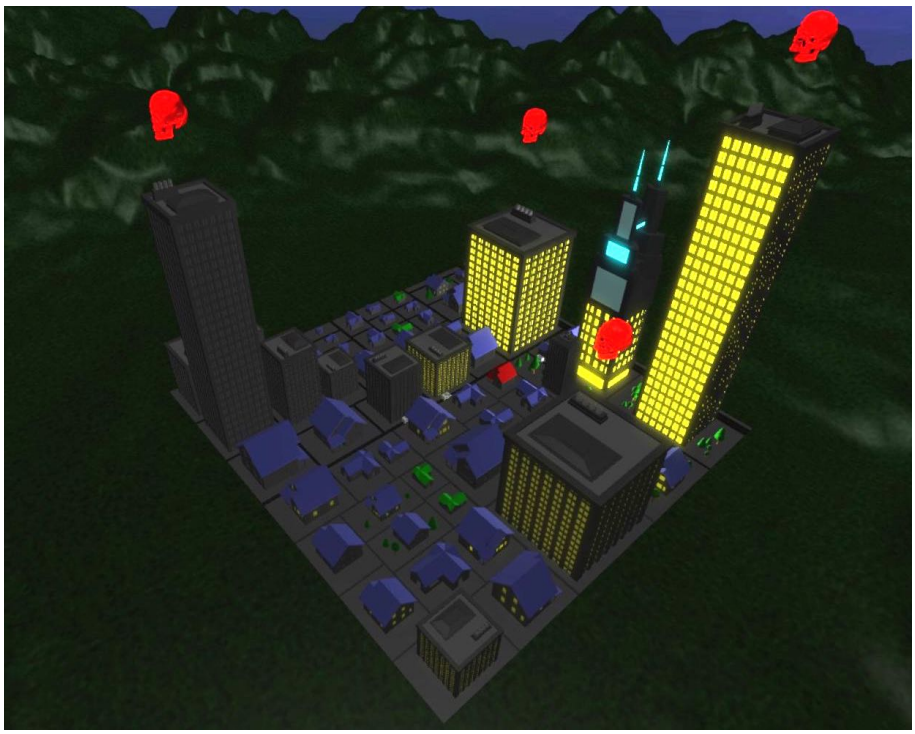


Abbildung 7.13: Stadt bei Nacht

7.1.4 Fortbewegung innerhalb der Stadt

Dieser Abschnitt bezieht sich auf UC6 (Kapitel 4.1.7). Für noch mehr Immersion hat der Benutzer die Gelegenheit, in die Stadt hineinzugehen. Dazu wählt er seinen Spawnpunkt auf in der Stadt aus.

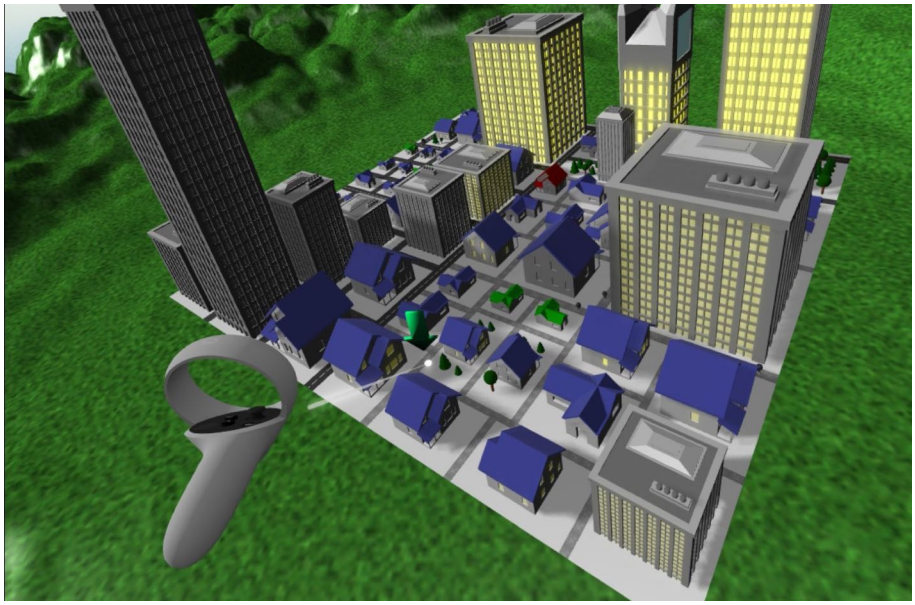


Abbildung 7.14: Spawnpunkt auswählen

Ist der Stadtpunkt ausgewählt, landet der Benutzer in der Stadt.

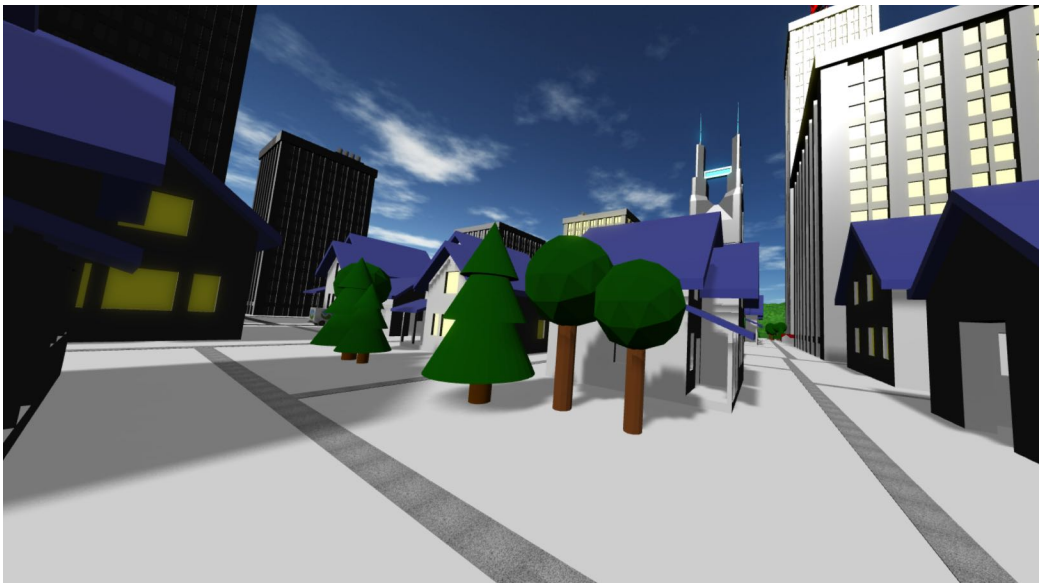


Abbildung 7.15: Innerhalb der Stadt

Natürlich kann der Benutzer in diesem Modus genau so mit der Stadt interagieren, wie sonst auch. Es besteht nun die Möglichkeit, sich in der Stadt fortzubewegen, um die Stadt von den verschiedensten Winkeln zu begutachten. Die GUI-Fenster werden deshalb automatisch beim linken Controller platziert, damit der Benutzer sie immer dabei hat. Bei Bedarf können die Fenster auch wieder an einem fixen Ort platziert werden.

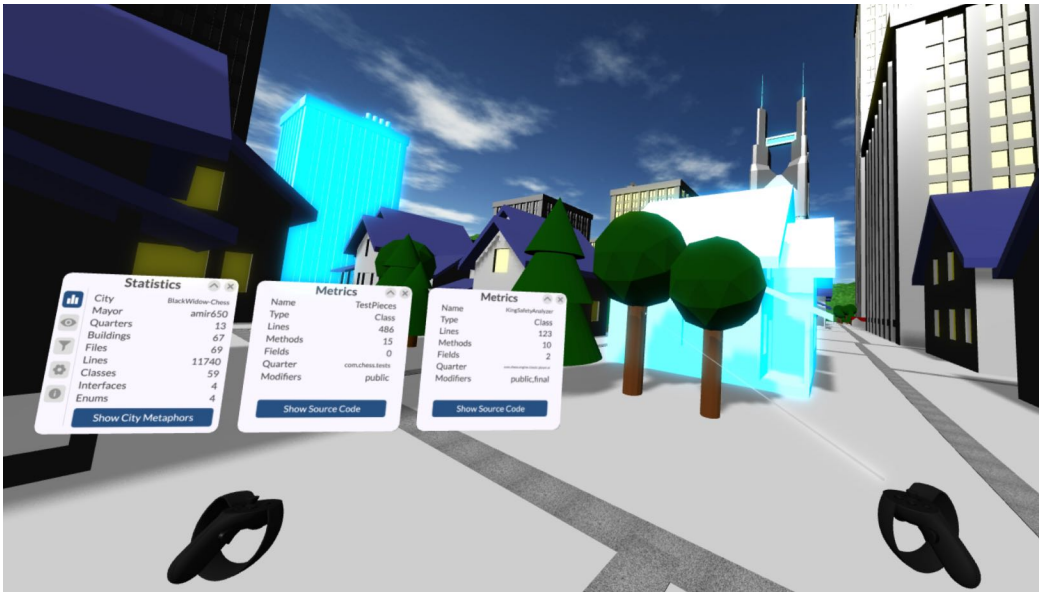


Abbildung 7.16: Selektierte Gebäude innerhalb der Stadt

7.2 Qualitätssicherung

7.2.1 Metriken

Die Source Code Analyse wurde von Sonarqube durchgeführt. Bei jedem Push in den Master Branch gab Sonarqube ein Feedback über die aktuelle Codebase. Es wurde darauf geachtet, die Warnungen zu beseitigen. Das spiegelt sich auch in der Zusammenfassung in der Abbildung 7.17 von Sonarqube wieder. Die Coverage ist hier allerdings bei 0%, weil die Testcoverage über einen separaten Weg ausgelesen wurde.

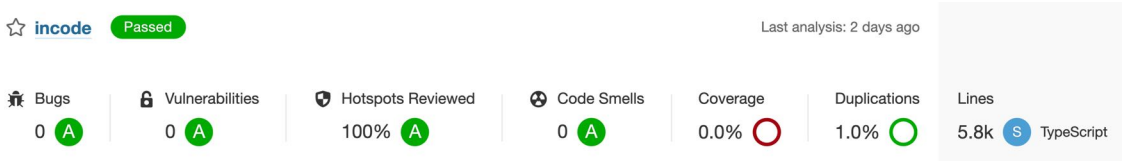


Abbildung 7.17: Messungen von Sonarqube

Zu erwähnen ist, dass zwei Warnungen von uns als False Positive markiert wurden. Bei leeren Funktionen, wie sie beispielsweise im Null Object Pattern vorkommen, hat sich Sonarqube beschwert. Einige non-null Assertions (ein Ausrufezeichen in TypeScript) empfand Sonarqube als redundant, was bei näherer Betrachtung nicht stimmte.

Die Tabelle 7.1 zeigt die von Sonarqube zur Verfügung gestellten Metriken.

Tabelle 7.1: Source Code Metriken

Metrik	Wert
Lines of Code	5822
Lines	6753
Statements	2794
Functions	725
Classes	84
Files	105
Comment Lines	32
Comments (%)	0.5%

7.2.2 Test Coverage

Um die Korrektheit der Anwendung zu überprüfen, wurden bei jedem Push in das GitLab Repository automatisierte Unit- und Integrationtests ausgeführt. Getestet wurde dabei die Business Logik der Applikation. Die Tabelle 7.2 zeigt die zusammengefasste Testcoverage. Eine ausführlichere Tabelle ist im Anhang A zu finden.

Tabelle 7.2: Test Coverage Zusammenfassung

File	% Stmts	% Branch	% Funcs	% Lines
src/businessLogic/ cityLayoutBuilder	86.66	100	75	89.28
src/businessLogic/ cityLayoutBuilder/strategies	96.66	89.53	96.87	96.56
src/businessLogic/ cityLayout-Builder/strategies/areas	100	100	100	100
src/businessLogic/ codeAnalyzer	90.24	75.86	85	91.25
src/dataLayer/githubProxy	86.31	69.56	90	87.09
src/dataLayer/model/ configuration	100	100	100	100
src/dataLayer/model/ projectData	93.18	80	93.75	93.18
src/dataLayer/model/ projectData/layout	70	62.5	45	70
src/dataLayer/model/ projectData/layout/cityItems	74.6	55.55	65	73.77
src/dataLayer/model/ projectData/processed	79.03	82.35	57.14	87.03
src/dataLayer/model/ projectData/processed/javaItems	92.3	92.3	86.66	92.3
src/dataLayer/model/ projectData/unprocessed	70.21	100	60.86	73.33
src/utils/errorhandling	75	100	62.5	75
All files	**86.86**	**81.06**	**77.36**	**87.79**

7.2.3 Validationstests

Die Korrektheit des System wurde auch über Validationstests überprüft. Da die Testergebnisse relativ umfangreich ausfielen, wurden diese im Anhang A festgehalten. Zusammengefasst lässt sich sagen, dass die Implementation die im Kapitel 4 definierten funktionalen sowie nichtfunktionalen Anforderungen fast alle erfüllen kann. Aus zeitlichen und funktionalen Gründen konnten die Features Stadtteile ausblenden oder Farben anpassen nicht implementiert werden. Ausserdem wurde während der Implementation die Leistungsfähigkeit der VR-Headsets überschätzt, was sich negativ auf die Performancetests auswirkte.

7.2.4 Usability Test

Dieses Usability Test Skript basiert auf einem Template von Slava Shestapalov. Zuerst wird der Ablauf des Usability Tests beschrieben. Danach werden die Resultate präsentiert. Der Usability Test basiert auf dem *usability_test* Tag im GitLab Repository. Es sollen folgende Punkte getestet werden:

- Überprüfen, ob das User-Interface benutzerfreundlich ist
- Lernen, welche Anpassungen in nachfolgenden Arbeiten gemacht werden könnten

Intro

Tabelle 7.3: Usability Tests: Intro

Vorgehen	Beschreibung
Das Ziel erklären	• Bei Unklarheiten, was incode ist, erfolgt eine Erklärung. • Bei Unklarheiten, was GitHub ist, erfolgt eine Erklärung. • Es soll herausgefunden werden, wie bequem, nützlich und selbsterklärend incode ist.
Aktivitäten	• Die Sitzung besteht aus zwei Teilen. Der erste Teil ist ein Mini-Interview. Im zweite Teil führt der Proband einige Aktionen mit incode durch.
Regeln für die Durchführung	• Aus Gründen der Genauigkeit dieses Experiments können Fragen erst nach der Sitzung beantwortet werden. • Ziel ist es, incode zu testen, nicht der Proband. Es gibt keine falschen Antworten - Feedback ist willkommen. Der Proband soll möglichst über alles laut nachdenken.

Mini-interview

Es werden die folgenden 3 Fragen über die Person und ihre Vorkenntnisse gestellt:

1. Haben Sie bereits Erfahrungen mit VR?
2. Was ist Ihr beruflicher Hintergrund?
3. Kennen Sie die Konzepte der Objektorientierten Programmierung? (Falls nicht, erfolgt eine kurze Erklärung)

Tasks

Da der Usability Test mit einem frühen Prototypen umgesetzt wird, soll der Test mehr explorativ stattfinden. Sprich die Probanden sollen möglichst selber mit der Anwendung spielen und die Features ausprobieren. Als grober Leitfaden dient die Auflistung der Tabelle 7.4.

Auch wenn VR mittlerweile schon seit einigen Jahren massentauglich gemacht wurde, kennen sich nach wie vor nur die wenigsten Leute gut damit aus. Aus diesem Grund wird, vor dem effektiven Usability Test, jeder Testperson ein Zeitraum von etwa 3 bis 5 Minuten zur Verfügung gestellt, um sich damit Vertraut zu machen. So soll vermieden werden, dass die Bewertung der Usability unter der Neuheit der Technologie leidet.

Kontext Ihr Freund hat Ihnen von incode erzählt. Nun möchten Sie sich selber ein Bild davon machen und besuchen die Webseite mit dem Ziel, ein Softwareprojekt zu analysieren.

Tabelle 7.4: Usability Test: Tasks

Use Case	Tasks
Übersicht über Softwareprojekt erhalten	• Besuchen Sie https://sinv-56014.rj.ost.ch. • Generieren Sie eine Stadt für folgendes Repository: – Username: amir650 – Repository: BlackWidow-Chess • Besuchen Sie die Stadt. • Rotieren und vergrössern Sie die Stadt. • Suchen sie sich ein Gebäude aus und finden Sie heraus, wie viele Zeilen Source Code dieses Gebäude repräsentiert. • Suchen Sie das Gebäude Player". • Sehen Sie eine Bedeutung der verschiedenen Gebäude und Elemente?
Nach Eigenschaften filtern können	• Zeigen Sie nur Klassen an, welche 100 oder mehr Zeilen haben. • Ändern Sie den Qualitäts-Schwellwert für die Anzahl Zeilen auf 500.
Styling der Stadt oder Umgebung anpassen	• Ändern sie die Textur der Gebäude. • Ändern sie die Tageszeit auf 07:00 Uhr.
Sich in der Stadt fortbewegen können	• Gehen Sie in die Stadt hinein. • Bewegen Sie sich in der Stadt. • Verlassen Sie die Stadt. • Verlassen Sie VR.

Outro

Folgende Fragen werden am Ende gestellt:

- Denken Sie, dass Sie einen guten Überblick über das Software Projekt erhalten haben?
- Denken Sie, dass dieses Software Projekt eine gute Qualität hat?
- Wie fühlen Sie sich? Ist Ihnen übel?
- Haben sie allgemeines Feedback oder Verbesserungsvorschläge zur Applikation?

Resultate

Der Usability Test wurde mit 4 Personen durchgeführt. Hier werden die wichtigsten Erkenntnisse zusammengefasst.

Von den Testpersonen waren 3 Personen mitte 20, wobei zwei von ihnen Informatik an der Ost studieren und der Dritte ein Maschinenbaustudium (ebenfalls an der OST)

bereits abgeschlossen hat. Die vierte Testperson war mitte 50 und arbeitet als Software Entwickler mit Embedded Systems zusammen. Keine der Testpersonen hatte bereits grosse Erfahrungen mit VR.

Allgemeine Bedienung Die Testpersonen hatten zunächst Mühe, herauszufinden, wie sich die Applikation bedienen lässt. Es musste zu Beginn einiges erklärt werden, beispielsweise welche Buttons auf dem Controller verwendet werden müssen. Nichtsdestotrotz gewöhnten sich die Testpersonen meistens relativ schnell an die Bedienung und es fiel ihnen zunehmend leichter durch die Applikation zu navigieren. Zu den kritischen Elementen gehörten die Buttons. Einige Leute wollten die Buttons im Menu nicht mithilfe der Laserpointers auswählen und betätigen, sondern indem sie den Controller zum Button bewegten. Es war nicht klar, dass die hauptsächliche Bedienung über den Standardtrigger des Controllers geht.

Der Laserpointer der Controller ist weiss. Die Hintergrundfläche des Menus auch. Das machte es den Nutzern zusätzlich schwierig den Pointer im Menu zu erkennen.

Beim Anschauen von längerem Source Code erscheint eine Scrollbar auf der rechten Seite des Fensters. Die Benutzer fanden schnell heraus, dass man diese Scrollbar greifen und bewegen muss, um zu scrollen. Entgegen ihrer Erwartungen kann man aber nicht den Text greifen und nach oben ziehen, um runterzuscrollen. Ausserdem kam hier der Kommentar, dass man den Source Code gar nicht lesen könnte, da er zu klein sei.

Menu Das Hauptmenu ist standardmässig eingeblendet, wenn die Stadt fertig gerendert wurde. Das Menu kann vom Benutzer ausgeblendet werden. Nach dem Ausblenden war allerdings nicht ersichtlich, dass man das Menu mithilfe eines Buttons auf dem Controller wieder einblenden kann.

Stadt betrachten Wie man Häuser selektiert, um dessen Metriken anzuzeigen, haben alle relativ schnell herausgefunden. Das Rotieren und Zoomen der Stadt war auch selbsterklärend. Die dazugehörigen Controls konnten schnell im Menu gefunden werden. Das Zoomen und Verkleinern der Stadt funktioniert nur in einem vorgegebenen Intervall. Werden die Grenzen dieses Intervalls erreicht, bleibt die Stadt in ihrer aktuellen Grösse. Das hat den Benutzern das Gefühl gegeben, dass der Zoom Button nicht mehr funktioniert.

Das Zoomen der Stadt funktioniert auch über die Mittelfinger Triggers auf den Controllern. Teilweise wurden diese Trigger aus Versehen geklickt, womit der Benutzer verwirrt war, wieso sich die Grösse der Stadt auf einmal verändert.

Ein Benutzer, der es bevorzugte, von oben herab auf die Stadt zu blicken, meinte zu nahe an ihr zu sein. Ihm würde es mehr entgegenkommen, wenn man von weiter weg schräg zur Stadt sieht. Die Begründung dafür war, dass vom ständigen herabschauen Nackenschmerzen entstehen können.

Software City Metapher Erst nach einigem Rumspielen wurden die verschiedenen Metaphern der Software City klar. Dass die Höhe der Häuser eine Bedeutung haben, wurde den meisten schnell bewusst. Die Bedeutung der Häuserbreite war weniger offensichtlich. Teils wurde sich gefragt, ob die Autos oder Bäume eine Bedeutung hätten, was nicht der Fall ist. Das Menu, in dem beschrieben steht, welche Metrik was bedeutet, wurde jeweils schnell gefunden.

In die Stadt hineingehen Teils wollte man in die Stadt reingehen, indem man genügend stark in die Stadt reinzoomt. Teilnehmer, die zuerst mit dem Menu spielten und nicht direkt die Stadt im Fokus hatten, bemerkten jedoch schnell, dass man über dieses Menu einfach in die Stadt gelangen kann. Denen, die sofort eine Stadbesichtigung machen wollten, war es hingegen nicht sofort klar, dass es dazu im Menu eine Funktion dafür gibt, über den man auch den Spawnpunkt in der Stadt festlegen kann.

Der Benutzer kann sich nur an gültige Punkte teleportieren. Der Spawnpunkt wird dem Benutzer bei der Auswahl über einen Pfeil angezeigt. Der Pfeil wird grün eingefärbt, wenn der Benutzer seinen Pointer auf eine gültige Position hält. Ein Haus wäre eine ungültige Position. Das wird über einen roten Pfeil markiert. In einer grösseren Stadt mit vielen Gebäuden gibt es weniger gültige Spawnpunkte, womit der Pfeil häufiger rot angezeigt wird. Das verwirrte die Benutzer, sodass sie sich fragten, weshalb sie keinen Spawnpunkt selektieren können.

Ein Feature ist es, dass die Gebäude während des Selektionsprozesses vom Spawnpunkt immer noch selektiert werden können, womit dann ihre Metriken in einem neuen Fenster erscheinen. Das führte bei einem Probanden jedoch mehr zur Verwirrung.

Zeit einstellen Alle Probanden haben das Menu zur Einstellung der Tageszeit schnell gefunden. Als dann allerdings im Editiermodus das Rad erschien, mit der die Zeit editiert werden kann, war es nicht klar, dass die Hand längs rotiert werden muss (Roll Axis[2]). Als die Testpersonen den Editiermodus abschliessen wollten, war vielen nicht bewusst, wie man das bewerkstelligt. Ausserdem muss man, um den Editiermodus abzuschliessen, den Controller bewegen, mit dem man die Zeit gerade eingestellt hat. Somit wird die Zeit wieder leicht geändert.

Teleportation Einigen war nicht sofort klar, wie sie sich in der Stadt fortbewegen können und versuchten zum Beispiel nur mit Kopfdrehen oder selber herumlaufen zum Ziel zu gelangen. Ein Teilnehmer vermisste es, sich so fortzubewegen, als ob er gehen würde, anstatt von Punkt zu Punk zu springen. Verwirrend war für die Benutzer, dass sie sich in die Häuser hinein teleportieren konnten. Allgemein wäre eine geführtere Fortbewegungsmöglichkeit wünschenswert. Einige Testpersonen erschraken, als sie mitten auf der Strasse auf einmal von einem Auto überfahren wurden.

Gebäude Suchen Das alphabetisch geordnete Keyboard fanden nicht alle Teilnehmer eingabefreundlich und sie wünschten sich eine vertraute CH-Tastatur. Wird ein Gebäude über die Suche gefunden, wird es markiert und seine Eigenschaften in einem Fenster angezeigt. Wenn man in der Stadt drinnen ist, sieht man das markierte Gebäude allerdings nicht direkt, wenn man nicht gerade davor steht. Das macht es schwierig herauszufinden, wo sich das gesuchte Gebäude befindet. Wenig benutzerfreundlich fanden die Teilnehmer, dass der Suchstring komplett mit dem Resultat übereinstimmen muss. Man hätte Gebäude auch auf Substrings matchen können und dann mehrere Resultate gleichzeitig anzeigen. Die Schrift der Fehler- und Erfolgsmeldung war schwer zu lesen, da der Font zu klein war.

Filtern Auch hier war die Schrift teilweise schwer zu lesen. Das Fenster musste näher vor das Gesicht gehalten werden. Allgemein machte das Filtermenu im ersten Augenblick einen überfüllten Eindruck.

[2] https://en.wikipedia.org/wiki/Aircraft_principal_axes

Einstellungen Der "Customize"-Button, um den Schwellwert einzustellen, wurde einmal als irreführend bemängelt. Hier wünschte sich eine Testperson eine andere Bezeichnung. Zudem wurde die Zeitanpassung auch mal im Settingsmenu erwartet und war überrascht, als dies im Viewmenu vorgefunden wurde.

Outro Fragen Zum Schluss wurden die folgenden Fragen von den Testpersonen beantwortet:

Tabelle 7.5: Usability Test: Outro Fragen Antworten

Frage	Antwort
Denken Sie, dass Sie einen guten Überblick über das Software Projekt erhalten haben?	Diese Frage wurde mehrheitlich mit nein beantwortet. Die Begründung war allerdings, dass die Testpersonen gar nicht auf das Software Projekt an sich geachtet haben, sondern mehr auf die Features von incode. Wenn sie mehr Zeit zur Verfügung gehabt hätten, hätten die meisten diese Frage vermutlich anders beantwortet.
Denken Sie, dass dieses Software Projekt eine gute Qualität hat?	Viele beantworteten diese Frage mit ja, da sie nur 4 Totenköpfe in der ganzen Stadt gesehen haben.
Wie fühlen Sie sich? Ist Ihnen übel?	Einer Person wurde es leicht übel, eine andere bekundete bereits nach wenigen Minuten etwas Kopfschmerzen. Das vor allem auch wegen den Laggs, weil durch das zugrundeliegende Software Projekt relativ viele Häuser erzeugt wurden. Den anderen hat diese VR Erlebnis allerdings nichts ausgemacht.
Haben sie allgemeines Feedback oder Verbesserungsvorschläge zur Applikation?	Einige Probanden wünschten sich mehr Interaktionmöglichkeiten innerhalb der Stadt. Beispielsweise sollte man die Totenköpfe abschiessen können, wenn man der Meinung ist, dass dieses Gebäude trotzdem den Anforderungen entspricht. Einer wollte einen Bulldozer einblenden, der die Gebäude entfernt, wenn man die Stadt filtert. Als Orientationshilfe während man sich innerhalb der Stadt befindet, könnte man sich eine Minimap auf der linken Hand anzeigen lassen. Alles in allem hat es den Probanden Spass gemacht, ein Software Projekt mal über eine gänzlich andere Art zu erleben. Die Testpersonen hatten Freude daran, in die Stadt hineinzugehen und die verschiedenen Elemente zu entdecken.

Fazit

Grundsätzlich hatten alle Teilnehmenden Spass am Testen der Applikation. Dies hatte sicher auch damit zu tun, dass VR für alle eine neue Technologie war.

Der grösste Kritikpunkt, der bei allen Teilnehmer zum Vorschein kam, war die Usability innerhalb der Stadt. Viele drückten zu Beginn einfach zufällig nach den Tasten, die ihnen am naheliegensten erschienen. Vor allem der A- respektive X-Knopf wurde oft gedrückt, wenn man etwas auswählen wollte. Dies führt jedoch nur dazu, dass das Menu ein- oder ausgeschaltet wird. Hier würde es einerseits Sinn ergeben, ein Tutorial oder eine Dokumentation zu erstellen, in der man nachlesen kann, wie man welche Funktionen

nutzt. Andererseits würde mehr Feedback dem Nutzer auch helfen zu verstehen, wie die Applikation anzuwenden ist. Was ebenfalls helfen würde, wäre die Fortbewegung innerhalb der Stadt natürlicher zu gestalten. Dazu könnte man die aktuelle Möglichkeit zur Teleportation auf dem einen Controller belassen, während man auf dem anderen Controller das Gehen einer Person simuliert kann.

Zusammenfassung

In diesem Kapitel wurde das Endergebnis der Implementation anhand von Bildmaterial präsentiert. Die Auswertungen der Umsetzung und der verwendeten Technologie folgen dann im nächsten Kapitel Diskussion. Der letzte Abschnitt in dieses Kapitels Qualitätssicherung beschäftigte sich mit der Qualität der Implementation, indem die verschiedenen Testergebnisse präsentiert wurden.

Kapitel 8

Diskussion

Übersicht

Das abschliessende Kapitel dieser Arbeit beschäftigt sich mit der Bewertung der Anwendung und der ausgewählten Technologie Babylon.js. Es werden positive und negative Aspekte gegenübergestellt. Am Ende dieses Kapitels werden Erweiterungsmöglichkeiten für weiterführende Arbeiten dargelegt.

8.1 Bewertung der Implementation

Die Implementation setzt einige Aspekte gut um und andere weniger gut. So sieht der fertige Prototyp durchaus nach einer Stadt aus. Bei genauerem Betrachten fallen allerdings Unschönheiten auf, wie etwa falsche Proportionen bei den Gebäuden. Die Bewertung wird in anwendungs- und entwicklungsspezifische Bewertungen aufgeteilt.

8.1.1 Anwendungsspezifische Bewertung

Was gut funktioniert

Dem Tool kann ein beliebiges Java Projekt angegeben werden, welches analysiert wird, um dann die Stadt darzustellen. Die Darstellung der Stadt funktioniert stabil, jedes Gebäude entspricht einer anderen Komponente des Projekts. Um zu kleine Grundstücke zu vermeiden, wurde die Kurve, die die Breite der Häuser bestimmt, abgeflacht und angehoben. Durch die Sortierung der Komponenten nach Grundstücksfläche vor der Berechnung des Stadtlayouts konnten gleichmässige Quadrate erzeugt werden.

Usability Tests haben ergeben, dass die Präsentation der VR-Welt visuell ansprechend ist. Es lassen sich gut die verschiedenen Stadtviertel erkennen. Ausserdem ist die Stadt durch Animationen von Fahrzeugen dynamisch und zeugt von einer belebten Stadt. Das Erlebnis, wenn man in die Stadt hineingeht, ist immersiv und ermöglicht es dem Benutzer, seinen Code in einer Art und Weise zu erleben, wie es die wenigsten kennen dürften.

Was verbessert werden kann

Aufgrund der Art, wie das Stadtlayout berechnet wird, stimmen die Proportionen der Gebäudebreiten nicht genau. Grund dafür ist das Padding. Damit es innerhalb der Stadt Platz für Strassen hat, wird von jedem Grundstück ein fixes Padding abgezogen. Bei schmalen Grundstücken wirkt sich dieses Padding stärker auf den effektiven Platz aus, der dem Gebäude zur Verfügung steht. Die Grösse des Paddings ist vom Gesamtgewicht des Projekts

abhängig. Der Street Layout Algorithmus (Kapitel 3.2.1) wäre, was die Gebäudeproportionen angeht, optimaler gewesen. So hätte die Stadt je nach Projekt aber stark entartet aussehen können. Sprich, wenn im Extremfall ein Projekt alle Klassen in einem einzigen Package abgelegt hätte, würde die Stadt aus nur einer einzigen langen Strasse bestehen.

Die Strassenkreuzungen werden momentan nicht immer sauber dargestellt. Die Texturen überlappen sich teilweise und schneiden so die Hälfte der anderen Strasse ab. Dort wäre es besser gewesen, wenn man an jeder Kreuzung eine separate Textur für Kreuzungen eingeblendet hätte.

Auch bezüglich der Usability gibt es eine Reihe von Verbesserungspotential. Diese wurden im Kapitel 7.2.4 genauer beschrieben.

8.1.2 Entwicklungsspezifische Bewertung

Die Entwicklung hat grösstenteils gut funktioniert. Es wurde Sonarqube verwendet, um die technischen Schulden gering zu halten.

Weniger optimal ist die Tatsache, dass beim Abfragen der Repository-Daten von der GitHub API für jedes Javafile eine neue Abfrage erstellt wird. Der neue Verbindungsaufbau jeder Abfrage kostet Zeit und verlangsamt den Abfrageprozess. Mittels eines Tokens für GitHub können 5000 Abfragen pro Stunde durchgeführt werden. Über den aktuellen Ansatz werden die 5000 Abfragen schneller erreicht. Im Source Code wurde es für den Notfall allerdings vorgesehen, mehrere GitHub Tokens zu hinterlegen, womit die Anzahl möglicher Abfragen pro Stunde schnell verdoppelt oder verdreifacht werden kann. Ansonsten bietet die GitHub REST API auch einen alternativen Weg an, wie man den Inhalt eines Repositories abfragen kann. Dort kann ein gesamtes Directory auf einmal abgefragt werden, jedoch gibt es Limiten, was die Anzahl Files und die Filegrössen anbelangt. Zusätzlich hätte man über diese Variante auch stets alle nicht-Javafiles heruntergeladen.

Die VR-Brillen haben leider nur wenig Rechenleistungleistung im Vergleich zu einer richtigen Grafikkarte. Die Oculus Quest bietet normalerweise die Möglichkeit an, das Headset mit einem PC zu verbinden, um dessen Grafikkarte zu verwenden. Dieses Feature war in Kombination mit unserer Anwendung jedoch nicht möglich. Nur speziell dafür entwickelte Applikationen, was nicht auf unseren Prototypen zutrifft, können Gebrauch von dieser Funktionalität machen. Verwendet man allerdings die WebXR Emulator Extension für Chrome, greift diese auf die Grafikkarte im PC zu. So lassen sich auch die grossen Städte laggfrei testen. Die Screenshots der grösseren Städte im Anhang C mussten beispielsweise über den Emulator erzeugt werden, da die Stadt im VR-Headset zu stark geruckelt hat. Die Handhabung des Emulators ist allerdings nicht wirklich benutzerfreundlich.

Hat man die Brille aufgesetzt und betrachtet die Stadt, ist man dazu verleitet, sich in alle Richtungen zu bewegen, um Dinge besser zu erkennen oder die Umgebung aus einem neuen Winkel zu betrachten. Die Oculus Quest (2) implementiert mit dem Guardian System[1] eine gute Sicherheitsfunktion. Sie warnt den Brillenträger, falls er aus dem vordefinierten Bereich austritt, indem eine virtuelle Wand eingeblendet wird.

Ein merkwürdiges Verhalten ist bei der Minification durch den Webpack Bundler aufgefallen. Sobald das Node Package *java-parser* minifiziert wurde, konnten das ausgelieferte JavaScript nicht mehr korrekt ausgeführt werden. Aus diesem Grund musste die Minification ausgeschaltet werden, wodurch die ausgelieferten Files grösser wurden.

Der Dark Mode (Kapitel 6.3.2) konnte aus Zeitgründen nicht umgesetzt werden. Die Prioritäten lagen in dieser Arbeit auf den VR Themen und der Software City.

[1] Guardian System von Oculus: https://developer.oculus.com/documentation/native/pc/dg-guardian-system/

Die Umsetzung der Integration Tests zur Business Logik war eine knifflige Angelegenheit. Web Workers lassen sich in Testumgebungen mit Jest[2] nicht so einfach integrieren. Deshalb musste dort vermehrt mit Vererbungen gearbeitet werden. In den abgeleiteten Versionen der Klassen konnte die Auslagerung auf einen echten Web Worker überschrieben werden und auf Fake-Web Worker umgeleitet werden.

8.2 Erfahrungen mit der Technologie

Die nachfolgenden Abschnitte diskutieren die Erfahrungen, die wir mit Babylon.js gemacht haben. Dabei werden sowohl die Fallstricke, sowie auch die positiven Elemente hervorgehoben. Es wird aufgezeigt, welche Features das Framework besonders gut unterstützt, als auch wie intuitiv es sich mit diesem Framework entwickeln lässt.

8.2.1 Babylon.js Features

Die Ausführungen umfassen alle Erfahrungen, die im Verlaufe der Implementation gesammelt wurden. Auch der im Kapitel 3.1.3 definierte Kriterienkatalog wird herangezogen.

3D Transformationen

Die lineare Algebra spielt bei der Entwicklung mit 3D Grafiken eine zentrale Rolle. Hier macht es Babylon.js dem Entwickler einfach, denn dieser muss sich nur minimal um diese Art von Berechnungen kümmern. Babylon.js bietet eine Vielzahl von Möglichkeiten für die Positionierung und Rotation von Meshes. Unterschieden wird von einer Transformation im globalen und im lokalen Raum eines Meshes. Letzteres ist abhängig von der Rotation des Meshes, was nützlich sein kann, falls man ein Objekt anhand seiner Rotation immer in dieselbe Richtung verschieben will. Das konnte in der Implementation beispielsweise bei der Positionierung des Gui Menus auf dem linken Controller verwendet werden. Das Fenster musste dann immer abhängig von seiner aktuellen Rotation nach hinten verschoben werden.

Man hat auch die Möglichkeit, Meshes voneinander abhängig zu machen. Sprich, man kann einem Mesh einen Parent zuweisen. Falls sich der Parent neu positioniert dreht oder skaliert, machen die Childs diese Transformation automatisch auch mit, sodass die relative Positionierung zum Parent immer noch dieselbe ist. Dank diesem Feature war die Berechnung und Positionierung der Gebäude innerhalb der Stadt ein leichtes Unterfangen. Jeder Komponente innerhalb der Stadt wurde derselbe Transform Node als Parent zugewiesen, der sich im Zentrum der Stadt befindet. Ein Transform Node ist ein Objekt, das nicht gerendert wird, aber als Zentrum der Transformation verwendet werden kann. Will man nun die Stadt rotieren oder zoomen, muss man nur diesen Transform Node manipulieren und die gesamte Stadt rotiert, wächst und schrumpft mit. Auch die Positionierung der Sonne konnte so vereinfacht werden. Es musste nur der Drehpunkt in der Mitte der Welt definiert werden. Beim Anpassen der Tageszeit musste so nur noch der Winkel aktualisiert werden.

[2] Siehe https://jestjs.io

Babylon.js bietet seine eigenen Vektor- und Matrix-Datentypen an. Diese Datentypen ziehen sich durch das gesamte Framework hindurch. Das ist zu Beginn etwas gewöhnungsbedürftig, weil man auch die RGB-Farben über diese Vektoren definieren muss. Die konsequente Umsetzung verleiht dem Framework aber einen roten Faden. Falls man seine eigenen Transformationen berechnen will, bietet Babylon.js auf diesen Datentypen auch Vektor und Matrix-Operationen, wie Subtraktion oder Multiplikation an.

Skybox

In Babylon.js lässt sich die Textur des Himmels über eine Skybox anpassen. Das Prinzip ist simpel. Man erzeugt einen Würfel um die gesamte Szene und packt eine Himmeltextur auf jede Seite. Jede Würfelseite entspricht dabei einem separatem Bild. Dieses Prinzip wird auch von anderen Game Engines, wie Unity, verwendet, was dazu führt, dass man viele vorgefertigte Skybox Texturen im Internet findet. Auch Babylon.js selbst bietet einige Texturen an. Die Schwierigkeiten beginnen dann, wenn man seine eigenen Texturen erstellen will. In Fall von incode sollen Wolken angezeigt werden, falls das Softwareprojekt schlecht formatiert wurde. Der Ansatz war, vorgefertigte Texturen zu nehmen und verschiedene Versionen davon zu erzeugen mit mehr Wolken darauf. Dabei gilt es aber darauf zu achten, dass die Übergänge der Bilder zu jeder Würfelseite übereinstimmt, denn sonst sieht der Benutzer direkt, dass etwas nicht stimmt. Normale Bildbearbeitungstools helfen hier nicht besonders weiter.

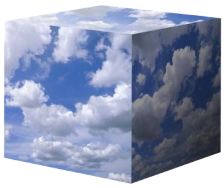


Abbildung 8.1: Skybox

Auch das räumliche Vorstellungsvermögen ist hier gefragt. Denn alle sechs Bilder müssen meistens selbst auf der korrekten Würfelseite und richtig herum rotiert angebracht werden.

Beleuchtung und Schatten

Die Erstellung der Beleuchtung in einer Szene ist meistens selbsterklärend und verhält sich so, wie man es erwartet. Von den insgesamt vier angebotenen Lichttypen wurden in der Implementation das Directional Light und Hemispheric Light verwendet. Das Directional Light wurde in der Sonne eingebaut. Die Sonne gibt ihr Licht somit nicht in alle Himmelsrichtungen ab, sondern nur in Richtung der Stadt, was performanter ist. Zusätzlich kann dieser Lichttyp Schatten von Meshes erzeugen, sodass das Rendering realistischer aussieht.

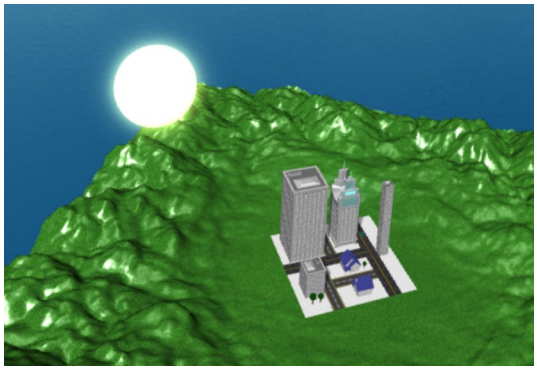


Abbildung 8.2: Sonne als Directional Light Source

Für die Beleuchtung der Stadt während der Nacht wurde das Pointlight ausprobiert. Und zwar sollten alle Häuser, dessen Fenster leuchten, ein eigenes Pointlight erhalten. Bei mehreren Häusern, in denen das Licht brennt, sollten also auch dynamisch mehrere Lichtpunkte erzeugt werden. Babylon.js kam allerdings schon bei wenigen zusätzlichen Pointlights nicht mehr damit klar und deaktivierte zum Teil das Licht der Sonne, womit es dann auch bei Tag dunkel wurde. Aus diesem Grund beliessen wir es bei zwei Lichtquellen. Die der Sonne und eine für die Umgebung, um auch bei Nacht etwas sehen zu können. Pointlights sind allerdings auch die aufwändigsten Lichttypen, da sie ihr Licht im Gegensatz zu den anderen omnidirektional in alle Richtungen ausstrahlen. Dies führt zu einem schnellen und nicht linearen Anstieg des Berechnungsaufwandes.

Die Schattenseite Die korrekte Erzeugung von Schatten ist mit nur wenigen Zeilen zusätzlichem Code realisierbar. Man muss für eine Lichtquelle nur angeben, ob sie Schatten werfen soll, welche Objekte Schatten werfen können und welche Flächen diesen Schatten empfangen. Je nach Setting erscheinen diese Schatten aber nicht zwangsläufig direkt. Gräbt man ein bisschen tiefer in der Dokumentation, findet man, dass der Distanzintervall zur Lichtquelle, in welchem der Schatten geworfen wird, noch angepasst werden muss. Hat man die Distanz angepasst, sind die Schatten aber noch verzogen. Zum Beispiel passt der Schatten der Wand nicht mit dem Schatten des Daches überein. Hierzu können Bias- und Normalbias-Werte angepasst werden. Passt man diese Werte an, gibt es aber einen anderen Nebeneffekt, der sich Shadow Acne nennt. Das führt dazu, dass die Oberflächen der Objekte einen Schatten auf sich selber werfen, was nicht sehr ästhetisch wirkt. Um diesen Effekt zu minimieren, kann entweder die Auflösung der Shadow Map erhöht oder weitere Bias Werte ausprobiert werden.

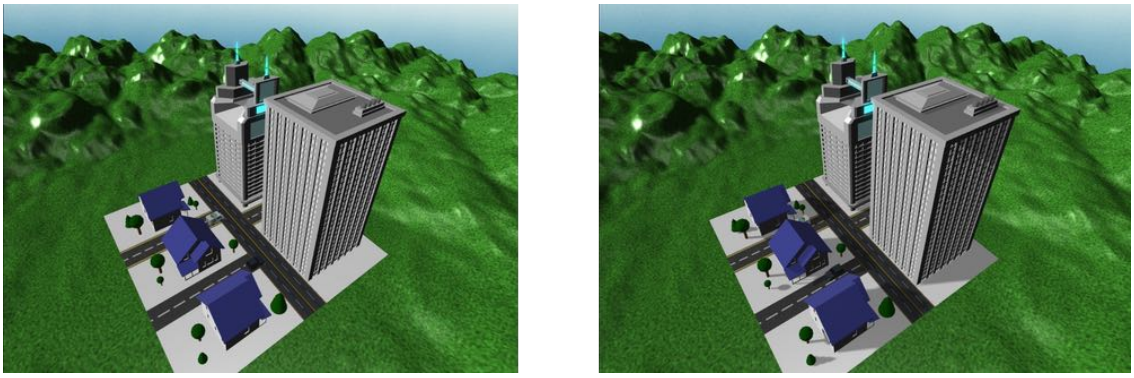


Abbildung 8.3: Die Stadt links ohne Schatten, rechts mit Schatten

In der Implementation wird die Stadt vergrössert, wenn man sich in die Stadt hineinteleportiert. Das führt dazu, dass die Einstellungen der Schatten nun nicht mehr korrekt sind und aktualisiert werden müssen. Durch die Vielzahl an Parametern, an denen geschraubt werden kann, ist die Gefahr gross, dass hier mehr Zeit verbracht wird, als ursprünglich eingeplant war. Dennoch gewöhnt man sich mit der Zeit an die Benutzung und findet auch immer mehr fortgeschrittenere Features wie beispielsweise das Contact Hardening, bei dem der Schatten beim Bodenkontakt schärfer dargestellt wird, als weiter davon entfernt.

Abbildung 8.4: Merkwürdiger Schatten am Haus und bei den Bäumen

Die Bearbeitung der Schatten ist vor allem für Neulinge alles andere als selbsterklärend. Das liegt vermutlich aber auch daran, dass Babylon.js unser erstes 3D-Grafik Framework war, denn die Prinzipien und Probleme wiederholen sich über verschiedene Frameworks.

Animationen

Grosse Animationen sind in der Applikation nicht zu sehen, lediglich einfache Translationen und Rotationen. Eine Animation besteht grundsätzlich aus einer Framerate und der eigentlichen Animation. Die Framerate beeinflusst, wie schnell die Animation vonstattengeht, die eigentliche Animation kann zum Beispiel eine Bewegung entlang der X-Achse bedeuten. Generell definiert man für ein Objekt, wo es zu welchem Zeitpunkt (Frame)

stehen soll. Zwischen zwei definierten Zeitpunkten wird dann eine regelmässige Animation vom Start zum Ziel durchgeführt. Ein Mesh kann mehrere Animationen gleichzeitig ausführen und sogar Geräusche machen. Importierte 3D-Objekte können auch gleich mit ihren eigenen Animationen verwendet werden. Selbsterstellte Animationen überschreiben die anderen nicht automatisch. Das ist in der Stadt bei den Rädern des Milchwagens ersichtlich oder bei den Armen des Lavagolems.

Abbildung 8.5: Milchwagen mit sich drehenden Rädern

Zu Beginn war uns nicht bewusst, dass sich die Animationen auf absolute Positionen im Koordinatensystem beziehen. Intuitiv nahmen wir an, dass eine Animation immer relativ zum Ursprungspunkt des Objekts geschieht. Diese Annahme führte zu offensichtlichem Fehlverhalten des Objekts.

Durch die Art und Weise, wie die Animationen der Fahrzeuge und Fussgänger im Prototypen definiert wurden, rotiert das Objekt schon früher, als es sollte oder erst zu spät. Die Translationen und Rotationen wurden unabhängig voneinander definiert und können somit voneinander wegdriften. Besser wäre hier wahrscheinlich die Verwendung des Path Features von Babylon.js gewesen[3].

Texturen

Die Einbettung von Texturen könnte meistens nicht einfacher sein. Babylon.js verwendet Materialien, um das Aussehen von Oberflächen anzupassen. Einem solchen Material kann eine Textur in Form eines Bildes oder einer einzelnen Farbe zugewiesen werden. Die Darstellung der Texturen auf den Strassen war ein wenig schwieriger in der Handhabung. Da die Strassen unterschiedlich lang sind, konnte nicht jeder Strasse eine einzige lange Textur zugewiesen werden. Die Oberflächen der Strassen mussten aufgeteilt und über eine jeweils verschiedene Anzahl an Strassentexturen abgefüllt werden, damit die Textur nicht verzogen aussah.

[3]Siehe https://doc.babylonjs.com/guidedLearning/workshop/Car_Path#path

Partikel

In der Anwendung wurden nur an einem Ort Partikel eingesetzt. Während der Benutzer die Zeit über das Zeitrad einstellt, erscheinen kleine leuchtende Punkte. Babylon.js bietet eine Fülle an Funktionalitäten nur für Partikel an. Das macht das Partikelsystem von Babylon.js extrem vielfältig und mächtig, aber auch unübersichtlich. Man kann sich schnell darin verlieren.

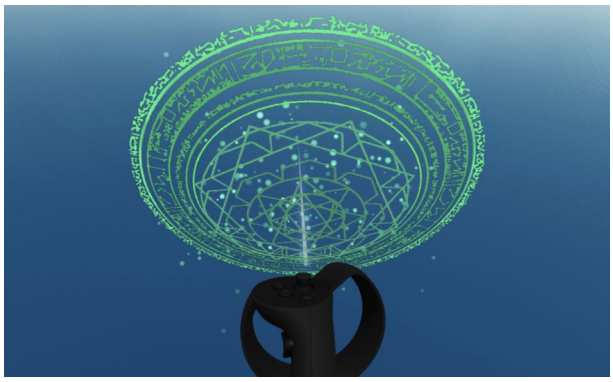


Abbildung 8.6: Zeitrad mit Partikeln

Umgebungslandschaft

Um die Umgebung und das Gesamtbild freundlicher zu gestalten, wollten wir eine Landschaft einbauen. Glücklicherweise kann man diese mit Babylon.js relativ einfach gestalten. Der Meshbuilder ermöglicht das Einlesen einer Heightmap, also einer zweidimensionaler Höhenkarte, und berechnet daraus aufgrund seines Farbverlaufs eine dreidimensionale Dreiecksvermaschung[4].

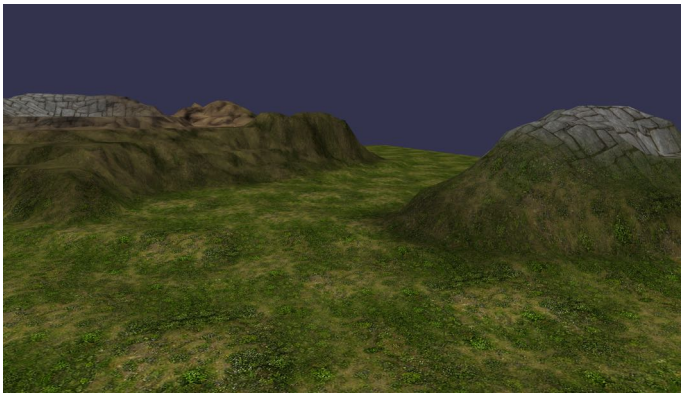


Abbildung 8.7: Heightmap

Auch diesem neu erstellten Mesh können Texturen angebracht werden. Eine einfache, aber effektive Variante ist, wie bei anderen Meshes eine Textur darüberzuspannen. Etwas komplizierter, jedoch auch mit grosser Unterstützung von Babylon.js, wird es dann, wenn man mehrere Texturen auf dasselbe Mesh projizieren möchte. Aufgrund des RGB-Farbverlaufs kann man mehrere Texturen einem Terrainmaterial[5] zuweisen.

[4] Siehe: https://desktop.arcgis.com/de/arcmap/10.3/manage-data/tin/fundamentals-of-tin-surfaces.htm

[5] Beispiel im Playground: https://playground.babylonjs.com/#E6OZX#7

Erstellung eines GUI

Ein Grossteil der Interaktion findet über das GUI statt. Babylon.js bietet von Haus aus eine umfassende Lösung für das GUI an. Für die VR-Applikation war es wichtig, dass das GUI innerhalb der 3D Welt platziert werden kann. Dies kann man über zwei Wege erreichen. Zum einen kann das Standard 2D Babylon.js GUI auf die Oberfläche eines Mesh projizieren werden. Zum anderen stellt Babylon.js ein separates 3D GUI (Abbildung 8.8) zur Verfügung, über das die Buttons als 3D Formen in der Welt eingefügt werden können.

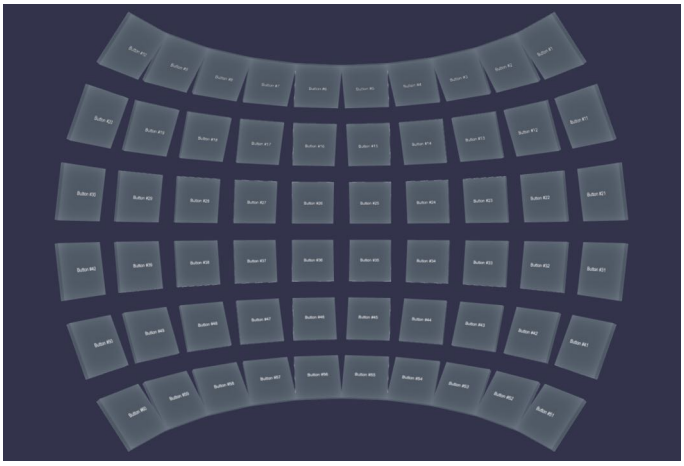


Abbildung 8.8: Beispiel eines Babylon.js 3D GUIs

Controls Spielt man ein bisschen mit den beiden Varianten, wird schnell offensichtlich, dass die Möglichkeiten beim normalen GUI für 2D-Oberflächen viel umfangreicher ausfallen. Das war der Grund, wieso diese Variante schlussendlich auch in der Anwendung verwendet wurde. Alle wichtigen Interaktionselemente eines Formulars werden hier direkt schon zur Verfügung gestellt. Neben den standardmässigen Labels und Text Buttons gibt es auch Image Buttons, Text Inputs, Radio Buttons, Checkboxes, Sliders und mehr. Sogar einen Color Picker haben sie im Angebot. Eine Progress Bar wird allerdings nicht angeboten. In der Implementation musste diese mit dem Slider simuliert werden. Auch der animierte Übergang, wenn es einen Fortschritt gibt musste selbst implementiert werden. Scroll Views werden ebenfalls unterstützt. Bei vielen Zeilen beginnt die Scrollanimation jedoch stark zu ruckeln.

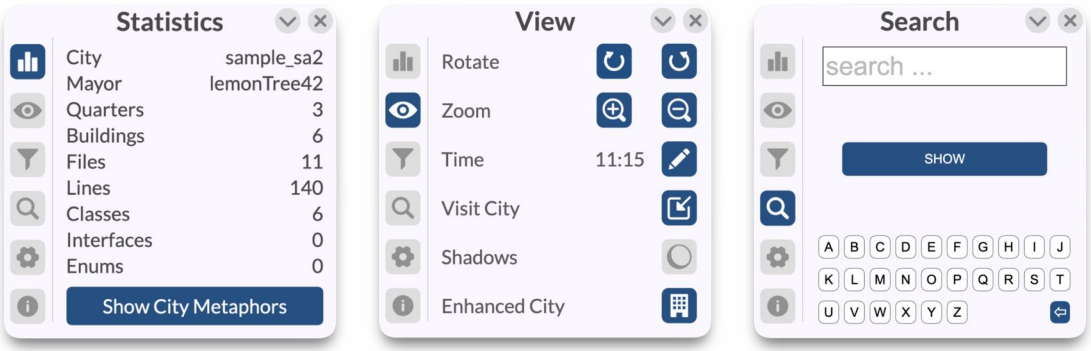


Abbildung 8.9: GUI des Prototypen

Layout Zur Anordnung der Controls können Container, wie ein Grid oder Stackpanel, verwendet werden. Dessen Verwendung ist bequem und leuchtet ein. Controls können beliebig in diese Container platziert werden. Container können auch in andere Container platziert werden, um eine hierarchische Struktur aufzubauen. So konnte relativ einfach das in der Abbildung 8.9 ersichtliche Layout erzeugt werden. Die zwei Buttons oben rechts mitsamt Titel sind ein eigener Container. Alles was unten an Content folgt, sind weitere Child Container.

Styles Um Codeduplikationen zu verringern, bietet Babylon.js an, Styles zu definieren. Diese Styles können dann den Controlls zugewiesen werden.

Unscharfe Schriften Was schnell aufgefallen ist, war, dass die Schriften schnell schwer zu lesen sind. Vor allem ein weiter entferntes GUI wird unscharf. Die Schriftgrössen mussten deshalb stark vergössert werden, was zu weniger Platz für den eigentlichen Inhalt führte. Diese Unschärfe bezieht sich nicht nur auf die Schrift auf dem GUI, sondern allgemein kriegt man das Gefühl, dass Babylon.js die Elemente in der 3D Welt weniger scharf rendert, als beispielsweise Three.js[6] (Siehe Abbildung 8.10). Bei einer genaueren Recherche wird klar, dass es sich hier um Default Einstellungen handelt und die Szene mit den richtigen Einstellungen auch schärfer dargestellt werden kann.

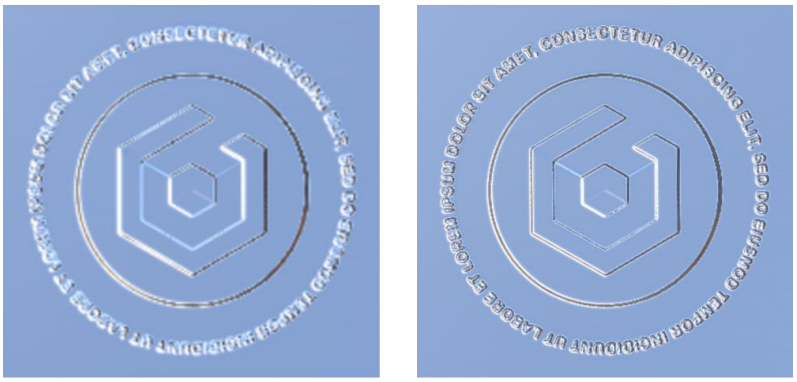


Abbildung 8.10: Schärfe der Bilder: Links Babylon.js, Rechts Three.js

Varianten in der Entwicklung Das 2D GUI kann man über drei Varianten definieren. Entweder man schreibt alles in Form von JavaScript oder TypeScript Code, oder man gibt die Struktur in einem separaten XML-File vor, oder man designt das GUI im GUI-Editor (Abbildung 8.11), exportiert die Struktur in ein JSON Format und importiert dieses JSON im Code. Dieser GUI-Editor ist zurzeit noch im Alpha Stadium. Aufgrund von verschiedenen Babylon.js Versionen (Abschnitt 8.2.2) funktionierte die GUI-Editor Variante leider nicht mit der im Prototypen verwendeten npm-Version zusammen. Der Ansatz mit den XML-Files ist wahrscheinlich die verbreiteste Variante (beispielsweise angewandt in Android, WPF oder in der Webentwicklung) und würde das Prinzip der Separation of Concerns am besten unterstützen. Unglücklicherweise gab es einige Komplikationen mit Webpack, der die XML-Files nicht korrekt verarbeiten konnte. Aus diesem Grund wurde auf die GUI-Definition über den Code ausgewichen. Das war ziemlich gewöhnungsbedürftig, funktionierte schlussendlich aber auch.

[6] Siehe https://forum.babylonjs.com/t/how-to-achieve-quality-like-with-three-js/12872/10

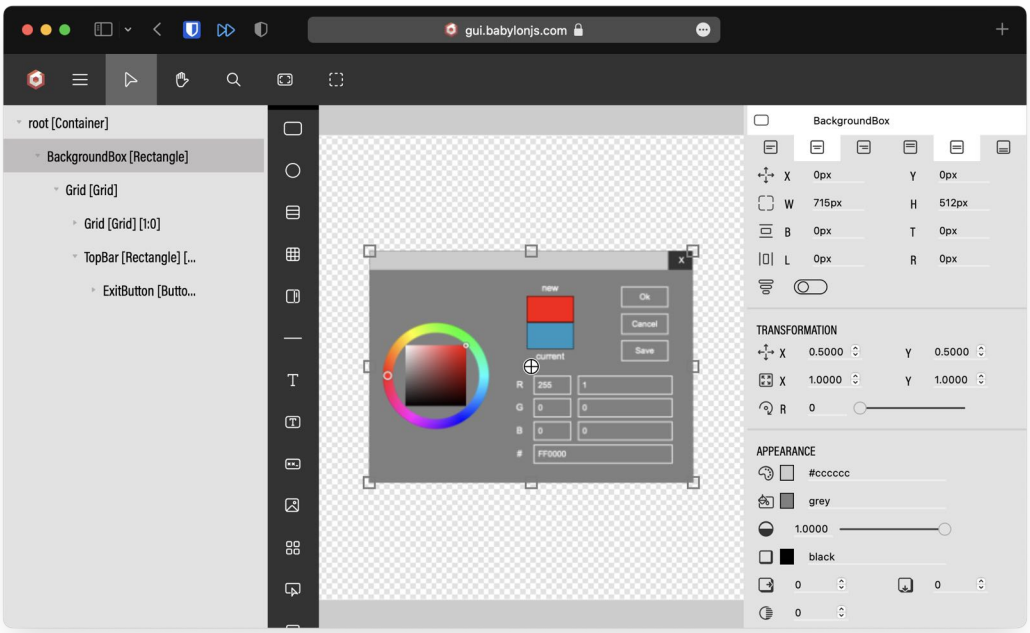


Abbildung 8.11: Babylon.js GUI-Editor

Verzogenes GUI Die GUI-Fenster in der Implementation sind alle quadratisch. Das hat den Ursprung darin, dass das Fenster nicht in nur eine Dimension wachsen konnte, ohne dass es den gesamten Inhalt verzogen hat. Trotz detaillierter Recherche konnte keine Lösung gefunden werden, ein Seitenverhältnis von beispielsweise 16:9 zu erzeugen, ohne dass die Schriften verzerrt wurden.

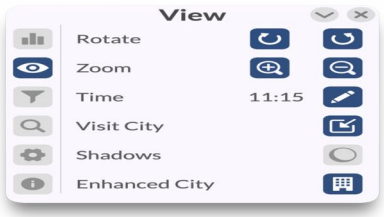


Abbildung 8.12: Beispiel eines verzogenen GUIs

VR Kamera

Die Positionierung und Ausrichtung der Kamera funktioniert beim erstmaligen Betreten der VR-Welt gut. Geht man allerdings aus der VR-Welt raus und dann wieder rein, zeigt die Kamera je nachdem in eine komplett neue Richtung. Der Grund für dieses Verhalten konnte nicht herausgefunden werden.

Einbinden von externen 3D Modellen

Über den Scene Loader können 3D Modelle diverser Filetypen in die Szene importiert werden. Die Files, die direkt ohne Probleme geladen werden konnten, waren *.glTF*, *.glb* und *.babylonjs* Files. Aus uns unbekannten Gründen sind *.obj* und *.blend* Files nach dem Import nicht in der Szene erschienen, obwohl dies gemäss Dokumentation möglich sein sollte. In den *.glTF & .glb* Fileformaten können auch Animationen hinterlegt werden, welche problemlos von Babylon.js ausgelesen werden. So war es zum Beispiel ohne Aufwand möglich, Räder der Autos drehen zu lassen.

Grössere Schwierigkeiten gab es beim Anpassen der Modelle. Da bei uns jede Strasse unterschiedlich gross ist, müssen die darauf fahrenden Fahrzeuge entsprechend skaliert werden. Die Skalierung an sich ist kein Problem, jedoch macht es Babylon.js einem nicht einfach die Abmasse eines Ojekts auszulesen. Womit unsere Lösung für dieses Unterfangen nicht besonders schön ausfiel. Die Abmasse der Modelle wurden manuell ausgelesen, dann hardcodiert im Soure Code festgehalten und somit die Breite des Objekts auf 1 skaliert. Nun hatten alle Objekte dieselbe Breite und konnten alle vom Renderer gleich behandelt werden.

Dass gewisse Datenformate der 3D-Modelle beim Import nicht angezeigt wurden, hinderte uns aber nicht daran, diese Objekte zu nutzen. Mit Hilfe von Blender kann man solche 3D-Objekte einfach in das gewünschte Format konvertieren.

Kombination mit Blender Falls man eigene 3D-Objekte modellieren will, ist Blender ein empfehlenswertes, kostenloses Tool. Man kann sein Modell von dort aus in ein beliebiges Format exportieren, dass von Babylon.js eingelesen werden kann. Babylon.js bietet sogar ein Exportplugin für Blender an, durch das die Modelle direkt in *.babylonjs* Files exportiert werden können. Auch Komprimierungen an den Modellen können vorgenommen werden, damit die Auslieferung der statischen Files zum Benutzer schneller vonstatten geht. Die Materialien, die in Blender definiert werden, findet man genau so auch in Babylon.js wieder, wenn man die Modelle importiert. Dadurch können die Materialien auf den Meshes auch nach dem Import noch verändert werden. So war es in der Implementation möglich die, Helligkeit der Fenster dynamisch anzupassen. Nicht nur Materialien können im Nachhinein aktualisiert werden, sondern auch die Form, Positionierung und Rotation der Meshes. Bei den Häusermodellen der Stadt wurden die Modelle mittels Blender normalisiert. Das heisst, dass die Breite aller Häuser nun einem Meter entsprechen. So konnte die Skalierung in Babylon.js später überall gleich behandelt werden.

WebXR

Dank der WebXR Device API (Kernimplementation des WebXR Featuresets: Kapitel 2.2.1) können Entwickler VR- und AR-Erlebnisse erstellen. Im Kontext dieser Arbeit wurde WebXR nur im VR-Modus getestet. Babylon.js integriert einen umfangreichen WebXR Support. Babylon.js kann über wenige Zeilen mitgeteilt werden, dass die aktuelle Szene in eine WebXR-Session umgewandelt werden soll. Szenen können also unabhängig von WebXR erstellt und später in eine WebXR Experience umgewandelt werden. Die Implementation der Anwendung weiss somit grösstenteils gar nicht, dass sie in einer VR-Umgebung läuft. Im Klartext bedeutet das, dass man als Entwickler die Features und Interaktionen mit der 3D Welt losgelöst vom VR-Kontext entwickeln kann. Man implementiert die An-

wendung wie eine normale WebGL Applikation und hat somit automatisch Zugriff auf das gesamte Featureset von Babylon.js. Erst zum Schluss könnte angegeben werden, dass man nun auch VR-Umgebungen unterstützen will. Babylon.js kümmert sich beim Starten der WebXR-Session darum, dass die VR-Features korrekt initialisiert werden.

Starten einer WebXR-Session Bei der Erstellung der Default WebXR Session initialisiert Babylon.js automatisch auch Features wie Input Controller mit der Möglichkeit zur Pointer Selection oder Teleportation. Ausserdem wird ein HTML-Button erstellt, der diese WebXR Features aktiviert, sobald man auf ihn klickt.

In der Anwendung sollte die WebXR Session zuerst von Grund auf selber initialisiert werden, weil dann vermieden wird, dass ein HTML-Button auf die Webseite gerendert wird. Das hat allerdings nicht wie gewünscht geklappt. Probleme bereitete vor allem das Darstellen der Controller. Das lag unter anderem auch daran, dass die API an dieser Stelle anders als sonst, nicht wirklich intuitiv zu bedienen war. Deshalb wurde dann trotzdem die Default WebXR Session verwendet und das HTML Canvas Element mitsamt Button ausgeblendet. Um die Session zu starten wird nun ein Klick auf den gerenderten und versteckten HTML-Button simuliert.

WebXR-Controller Wenn die Controller mal geladen wurden, ist es wiederum einfach, die Buttonklicks auszulesen. Auch haptische Feedbacks können problemlos an die Controller gesendet werden. Babylon.js erkennt sogar automatisch, um welche Controller es sich handelt. In unserem Fall hat Babylon.js auch die Oculus Quest 2 erkannt und automatisch die zugehörigen 3D Modelle geladen (Abbildung 8.13). So sieht der Benutzer seine physischen Controller in der virtuellen Welt wieder ohne dass der Entwickler einen grossen Aufwand hätte betreiben müssen. Sogar Buttonklicks auf den Controllern werden von Babylon.js realitätsgetreu nachanimiert.

Abbildung 8.13: 3D-Modell eines Oculus Quest 2-Controllers

8.2.2 Entwicklungsspezifische Erfahrungen

Eignung für grössere Projekte

Ein grosses Plus für Babylon.js ist, dass das Framework mit TypeScript implementiert wurde. Sprich als Entwickler kann man immer auf eine typisierte Library zugreifen. Das hilft ungemein, wenn man eine grössere Codebase schreiben will. Von den 5.8k Zeilen Source Code sind etwa 3k Zeilen Entwicklung mit Babylon.js. Das gilt zwar immer noch

als ein eher kleines Projekt, jedoch kann daraus mehr oder weniger abgeleitet werden, wie gut sich ein Projekt mit Babylon.js skalieren lässt. Nach unserer Einschätzung eignet sich Babylon.js durchaus für grössere Projekte. Man merkt, dass beim Framework viel Wert auf Einfachheit gelegt wurde. Viele Features kann man in wenigen Zeilen implementieren, während man zum Beispiel bei Three.js mehrere Zeilen benötigt[7]. Schlussendlich ist das grösste Kriterium für ein erfolgreiches grösseres Projekt, wie wahrscheinlich in jedem Projekt, ein gutes Software Design. Und hier kann Babylon.js nicht viel ausrichten, jedoch bieten sie durch den TypeScript Support und der Benutzerfreundlichkeit eine gute Unterstützung.

Verschiedene Versionen von Babylon.js

Babylon.js kann über verschiedene Varianten in ein Projekt eingebunden werden. Es kann zum Beispiel die CDN-Version verwendet werden, die auch von den von Babylon.js zur Verfügung gestellten Playgrounds verwendet werden. Incode verwendet die npm-Version. Leider kam es während der Entwicklung häufig vor, dass gewisse Funktionalitäten in der npm-Version schlicht nicht vorhanden waren, die von der CDN-Version allerdings angeboten wurden. So konnte beispielsweise das über den GUI-Editor entworfene GUI nicht geparst werden, denn die benötigte Parsefunktion existierte zu diesem Zeitpunkt nur in der CDN-Version. Allgemein war das nur in den seltensten Fällen ein Problem, dass die API nach offizieller Dokumentation nicht der effektiven Library entsprach.

Entwicklung mit einem VR-Headset

Den Code der VR Applikation während der Entwicklung immer auf einer VR-Brille zu testen, kann mühsam sein. Der von Mozilla entwickelte WebXR-Emulator[8] für Chrome und Firefox beschleunigt die Entwicklung enorm. Babylon.js unterstützt diesen Emulator offiziell und das Babylon.js Team nutzt diesen auch selbst um Babylon.js zu entwickeln. Zu beachten ist aber, dass der Emulator vor allem in Bezug auf die Controller stark eingeschränkt ist. Von den sechs Buttons eines Oculus Quest 2-Controllers können vom Emulator nur zwei getriggert werden.

Dokumentation und Community

Die Dokumentation ist ausführlich, aber trotzdem kompakt. Man findet sich schnell zurecht. Codebeispiele werden vielfach durch Playground Demos visualisiert. Das macht die Dokumentation zu einer ausgezeichneten, aber leider oftmals auch einzigen[9] Informationsquelle während der Entwicklung.

Playground Der Babylon.js Playground ist ein live Editor für Babylon.js Szenen. Mit der live Ansicht auf der rechten Seite eignet er sich optimal dafür, neue Features auszuprobieren bevor man sie im eigenen Projekt umsetzt. Die Code Completion im Editor bietet dem Entwickler einen guten Überblick über die vorhandenen Features. Nicht immer offensichtlich ist es, wie der Code dann in das eigene Projekt eingeflochten werden soll. Mit mehr Erfahrung wird es aber immer klarer. Auf den Playground wurde während der Entwicklung häufig zurückgegriffen.

[7] https://javascript.plainenglish.io/webgl-frameworks-three-js-vs-babylon-js-36975d915694#9a89

[8] Siehe https://blog.mozvr.com/webxr-emulator-extension/

[9] Siehe Community

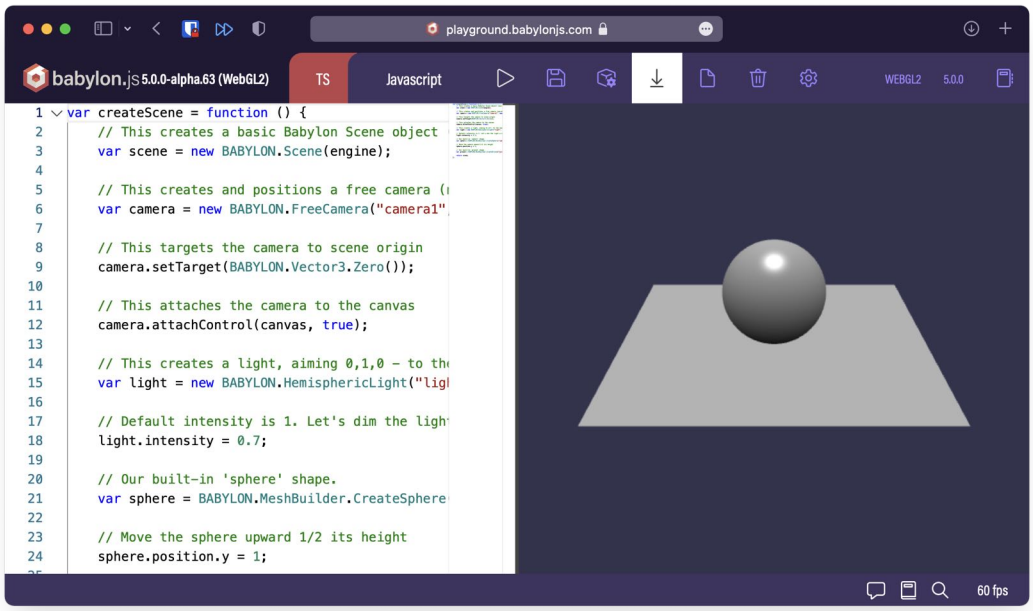


Abbildung 8.14: Babylon.js Playground

Community Babylon.js besteht aus einer kleinen, aber aktiven Community. Stellt man eine Frage in ihr eigenes Forum, kriegt man meistens innert Kürze eine Antwort. Aus unklaren Gründen ist Babylon.js viel weniger weit verbreitet, als Frameworks wie Three.js oder A-Frame. In der Abbildung 8.15 wurde das Suchinteresse dieser drei Framworks bei Google verglichen. Das Bild spricht für sich.

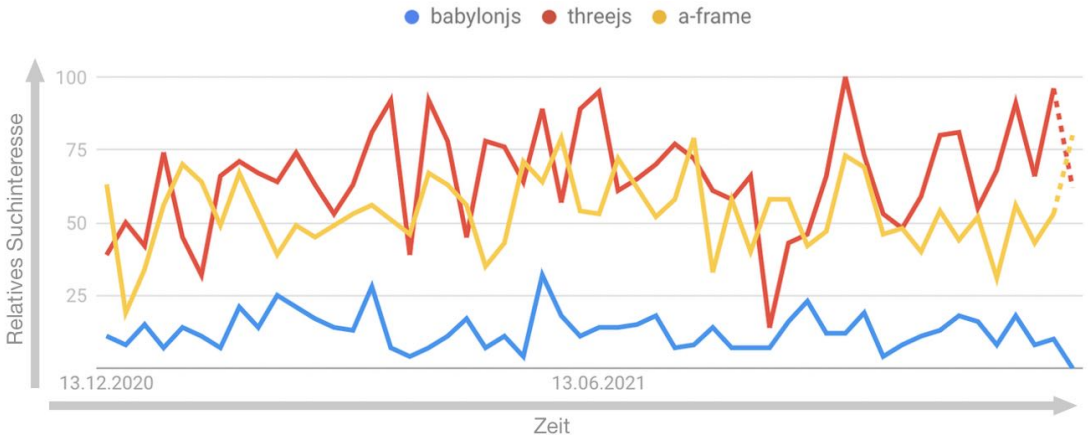


Abbildung 8.15: Vergleich von Babylon.js zu Three.js und A-Frame

Die Werte der y-Achse geben das Suchinteresse relativ zum höchsten Punkt im Diagramm an. Der Wert 100 steht für die höchste Beliebtheit dieses Suchbegriffs. Der Wert 50 bedeutet, dass der Begriff halb so beliebt ist und der Wert 0 bedeutet, dass für diesen Begriff nicht genügend Daten vorlagen[10].

[10]Für eine genauere Erklärung siehe Google Trends: https://trends.google.com/trends/?geo=CH

Durch dieses niedrige Interesse an Babylon.js findet sich auch weniger Hilfe abseits der offiziellen Babylon.js Seiten wie etwa Stackoverflow. So hat man häufig keine andere Wahl, als in der offiziellen Dokumentation nachzulesen, was jedoch sowieso für jedes Projekt empfehlenswert sein dürfte. Probleme kriegt, man dann wenn Teile der Dokumentation nicht ausführlich genug sind. Aufgrund des konstant niedrigeren Interesses für Babylon.js, ist die Frage berechtigt, wie die Zukunft dieses Frameworks aussieht. Was Hoffnung gibt, ist die kontinuierliche Weiterentwicklung von Babylon.js. Regelmässig werden neue Versionen veröffentlicht mit Bugfixes und neuen Features[11].

Testing

Das Testing des Babylon.js Projekts stellte sich als umständlich heraus. Babylon.js an sich stellt keine richtigen Tools zur Verfügung, mit denen man sein Projekt testen kann. Babylon.js selbst testet ihre Playgrounds über Validation Tests, indem sie automatisch Screenshots vom HTML Canvas Element erzeugen und bei Codeänderungen die Screenshots auf Änderungen überprüfen. Es wäre denkbar, dass man denselben Ansatz für sein eigenes Projekt verfolgt. Da sich dies als zu aufwändig herausstellte, wurde auf die Validation Tests verzichtet. Als Alternative, um das UI dennoch zu testen, wurden Usability Tests durchgeführt. Der Fokus der automatisierten Tests lag in dieser Arbeit auf der Business Logik.

Performance

Um die Szene performanter zu machen, nimmt Babylon.js selbst schon einige Optimierungen vor. So werden weiter entfernte Texturen automatisch unscharf dargestellt. Die zwei grössten Faktoren auf die Performance hatten die grosse Anzahl an Meshes in der Szene und dessen Schatten. Um den Schatten performanter zu machen konnte die Auflösung der Shadow Map runtergeschraubt werden. Zu stark war dies allerdings auch nicht möglich, da die Schatten an den Kanten ausgefranst wurden. Die Anzahl an Meshes hängt von der Grösse des analysierten Softwareprojektes ab und wurde von uns auch nicht wirklich optimiert. Die Komplexität der einzelnen Meshes konnte jedoch verringert werden, was sich positiv auf die Performance ausgewirkt hat. Eine Möglichkeit, die Babylon.js anbietet, wäre das Freeze Feature. Damit können alle Objekte in der Welt eingefroren werden und ändern somit ihre Position und Rotation nicht mehr. Allerdings würden dann die Animationen der Autos nicht mehr funktionieren.

WebGPU Die schwächere Performance, als man es von Desktop Umgebungen kennt, war unglücklich. Die ganz grossen Städte (über 100'000 Zeilen Code) können auf dem VR-Headset fast nicht betrachtet werden. Auch bei der Komplexität der importierten Meshes, wie den Gebäuden oder Bäumen, musste man Kompromisse eingehen. Das Versprechen, das hinter WebGPU[12] steht, ist eine signifikat schnellere API, die eine low level Kontrolle der Grafikressourcen von Javascript aus ermöglicht. Es wäre interessant gewesen, die Performance der Anwendung damit zu testen, denn WebGPU soll ab der Babylon.js Version 5.0 unterstützt werden. Verwendet wurde in dieser Arbeit die Version 4.2. Nach den Angaben von Babylon.js soll die Version 5.0 im April 2022 erscheinen[13].

[11] https://github.com/BabylonJS/Babylon.js/releases?page=1

[12]

[13] Siehe Babylon.js Forum https://forum.babylonjs.com/t/babylonjs-5-milestones-planning/22464/2

Lazy Loading und Tree Shaking Die Babylon.js Library ist zu gross, als dass sie beim Besuch der Webseite komplett mitgeladen werden kann. Es musste deshalb auf Lazy Loading ausgewichen werden. Neuerdings bietet Babylon.js für ihre Library auch Tree Shaking an. In Kombination mit einem Bundler wie Webpack werden beim Import von ES6 Modulen nur die Teile gebündelt, die auch wirklich verwendet werden.

8.2.3 Nicht verwendete Features

Babylon.js ist zu umfangreich, als dass alle Features getestet werden konnten. Nicht behandelt wurden beispielsweise die Integration von Audio oder Physiksimulationen. Es existiert auch eine Vielzahl an Möglichkeiten die Szene aufzuhübschen, wie PBR Materialien, Lens Flares, oder Sprites. 2D-Texturen könnte durch Normalmapping einen 3D-Effekt verliehen werden.

8.2.4 Fazit

Babylon.js fühlte sich zu Beginn fremd an. Beide Teammitglieder kannten sich vorher allerdings auch nicht mit 3D Framworks aus. Wir können es uns vorstellen, dass der Einstieg einiges einfacher wäre, wenn man bereits Erfahrungen in der 3D-Grafik Entwicklung mitbringt. Beschäftigt man sich aber eine Weile mit dem Framework, geht die Verwendung schnell ins Blut über. Ausserdem macht es Spass die verschiedenen Features auszuprobieren. Die Lernkurve ist zu Beginn speziell für WebGL-Neulinge etwas steiler, flacht dann aber schnell ab, was unter anderem auch der übersichtlichen Dokumentation zu verdanken ist.

8.3 Ausblick und Erweiterungsmöglichkeiten

Das Resultat dieser Arbeit ist ein funktionierender Prototyp, über den man ein beliebiges Java Projekt analysieren und überblicken kann. Nichtsdestoweniger ist diese Arbeit nur die Spitze des Eisberges an verschiedenen Ideen und Möglichkeiten. Die Metapher der Software City kann in vielerlei Hinsicht ergänzt und verbessert werden.

8.3.1 Mögliche Funktionale Features

Minimap Eine Idee wäre es, dem Benutzer eine Minimap einzublenden, während er durch die Stadt läuft. Dies könnte bewerkstelligt werden, indem von der gerenderten Stadt von oben initial ein Foto geschossen, abgespeichert und auf dem Menufenster eingeblendet wird. Das Foto müsste dabei mit einer orthografischen Kamera aufgenommen werden[14]. Ein dynamischer Punkt auf der Minimap würde die aktuelle Position der Benutzers in der Stadt anzeigen.

Alternativ könnte die Stadt auch rekursiv Element um Element auf die Minimap gezeichnet werden. Das wäre vermutlich aus Performancegründen weniger gut denkbar.

Gebäudeschilder Eine weitere Idee ist, dass man direkt bei den Gebäuden schon eine Anzeige einfügt, was das Gebäude genau darstellt. Beispielsweise könnte ein Schild mit dem Namen der Klasse und bei Bedarf auch weiteren Informationen vor das Gebäude platziert werden.

[14] Ein von Babylon.js unterstütztes Feature: https://doc.babylonjs.com/typedoc/classes/babylon.camera#orthographic_camera

Bedeutungen der Autos und Bäume In der aktuellen Implementation haben Objekte wie Autos oder Bäume noch keine Bedeutung und dienen nur der Dekoration. Hier wäre es denkbar, diesen Komponenten eine Bedeutung zu verleihen. Eine nicht ganz triviale Idee wäre es, dass die Autos die Imports von Packages und Klassen darstellen. So könnte der Import einer spezifischen Klasse dadurch abgebildet werden, dass ein Lieferwagen ein Package von einem Gebäude an das andere Gebäude liefert.

Mehrere Benutzer Ein aufwändigeres Feature wäre es, den Benutzern die Möglichkeit zu geben, zu zweit eine Stadt zu besuchen. Man könnte dann gemeinsam durch die Stadt laufen und das Projekt analysieren. Dabei wären die Avatare der beiden Teilnehmer ersichtlich, damit man sieht, wo sich der andere befindet. Da man das wahrscheinlich mit Websockets umsetzen würde, müssten grössere Teile der Architektur umgeschrieben oder angepasst werden.

Zeitreise Ein zusätzliches Feature, das im Vergleich einfacher zu implementieren wäre, ist die Zeitreise. Aktuell kann zwar die Zeit verändert werden, allerdings bezieht sich das nur auf den Sonnenstand. Hier könnte man die Stärken von Git ausspielen und auch die Stadt zu einem älteren Commit darstellen. Da die Analyse für einen einzelnen Commit allerdings lange dauert, müsste man eine Art Puffer vorsehen, in dem die vorberechneten älteren Commits abgelegt werden.

Customization Man könnte dem Benutzer auch die Möglichkeit für mehr Customizing geben. Die aktuellen Bedeutungen der Gebäudedimensionen und Gebäudefarben und so weiter sind fest vorgegeben. Für die Analyse des Softwareprojektes wäre es von Vorteil, wenn man selbst anpassen kann, welche Bedeutungen die verschiedenen Dimensionen haben.

Cyclomatic complexity Die Metrik der Cyclomatic Complexity könnte auch in das Projekt eingeflochten werden. Gebäude, die einen schlechten Wert erzielen, könnten als eine Ruine dargestellt werden, während Gebäude mit guten Werten als ein Topmodernes Gebäude dargestellt wird.

Rundfahrten Wenn man sich in der Stadt aufhält, will man teilweise nicht selbst überall hinlaufen. Da würde sich eine Option anbieten, in einen Bus zu steigen, der durch alle Viertel der Stadt fährt. Der Benutzer müsste nur den Bus anwählen und schon fährt er mit. Über ein Display innerhalb des Busses könnte das aktuelle Viertel angezeigt werden, mit Metriken zu diesem Viertel. Auf Wunsch der Benutzers könnte der Bus anhalten, damit der Benutzer aussteigen kann, um sich selbst umzuschauen, bis er dann wieder weiterfahren will.

In die Gebäude hineingehen Aktuell stehen die Gebäude für ganze Klassen, Interfaces oder Enums. Es gibt aber keine metaphorische Möglichkeit für den Benutzer, eine Klasse für sich genauer zu analysieren. Dies könnte man dadurch ermöglichen, dass man in die Gebäude hineingehen kann und dort beispielsweise die Funktionen visualisiert.

Darkmode Geplant, jedoch nicht umgesetzt, wurde ein Darkmode in der Browseransicht sowie im Immersive-Mode. Dies wäre vor allem auf der Homepage noch mit überschaubarem Aufwand machbar.

8.3.2 Mögliche Nichtfunktionale Features

Abfrage der GitHub Daten Die Anwendung kann auch abseits der funktionalen Features verbessert werden. So muss momentan für die Abfrage des GitHub Repositories mit Abstand am meisten Zeit aufgewendet werden. Das liegt zu einem grossen Teil daran, dass jedes Javafile eine separate Abfrage ist. Man könnte dem Benutzer auch die Möglichkeit geben, das Repository als ZIP-File selbst runterzuladen und auf incode hochzuladen, damit die Ordnerstruktur des Repositories komplett lokal analysiert werden kann.

Auslagern der Berechnungen Falls man Zugriff auf einen performanten Server hat, könnte man die Analyse des Projektes mit der Berechnung des Stadtlayouts auf einen Server auslagern. Die Webseite würde dann nur noch das Resultat als *DTO*[15] empfangen.

Voranalysierte Projekte Auch denkbar wäre es dem Benutzer eine Auswahl an voranalysierten Projekten zu geben und als *DTO* abzuspeichern. Dieses *DTO* müsste dann nur noch gerendert werden und der Benutzer sieht die Stadt ohne den Analyseprozess im Vorfeld.

Verbesserung der Bedienbarkeit Beim Usability Test wurde ersichtlich, dass das Fortbewegen innerhalb der Stadt nicht immer intuitiv ist. Man könnte somit einen Modus anbieten, in dem der Benutzer nur der Strasse entlang laufen kann. Auch könnte man für die Teleportation Snappoints[16] auf den Strassen errichten, damit die Fortbewegung mehr geführt ist.

8.4 Fazit

Diese Arbeit sollte sich mit der Frage auseinandersetzen, ob sich eine VR-Applikation gut mit Webtechnologien umsetzen lässt. Da Webtechnologien inhärent plattformübergreifende Lösungen sind, soll die Anwendung so automatisch hardwareneutraler werden.

Angefangen mit einer Vorstudie über VR, den verschiedenen WebGL Frameworks und einer Analyse und Planung des Anwendungsfalls wählten wir Babylon.js zu unserem Favoriten, um unsere Software City zu realisieren. Die Umsetzung der Software City wurde über verschiedene Tests validiert und verifiziert. Abschliessend wurden unsere Erfahrungen mit Babylon.js anhand von Beispielen erläutert. Es ist geplant, dass diese Arbeit weitergeführt wird. Im Abschnitt 8.3 wurde auf mögliche Erweiterungen hingewiesen.

Die Umsetzung der VR-Applikation mit VR-Technologien entpuppte sich als eine intuitive Angelegenheit. Da keines der beiden Teammitglieder Erfahrungen mit VR mitbrachte, konnte auch kein direkter Vergleich zur Umsetzung einer nativen VR-Applikation gezogen werden. Dies wäre speziell in Hinblick auf die Performance interessant gewesen. Trotzdem hatten wir im Allgemeinen nicht das Gefühl, in der Funktionalität in irgendeiner Form eingeschränkt zu werden. Das Zusammenspiel zwischen Hard- und Software funktionierte abseits von kleineren Schwierigkeiten einwandfrei. Diese Probleme lassen sich teilweise auch auf Babylon.js zurückführen. Die geplanten Features der Anwendung konnten wie geplant umgesetzt werden, wobei sich Babylon.js als ein sehr zweckmässiges Tool erwies.

Was ist also nun die Antwort auf die anfängliche Frage, wie gut sich die virtuelle Realität mit Webtechnologien umsetzen lässt? Wie so häufig lässt sich eine solch allgemeine Frage nicht abschliessend beantworten. Was allerdings mit Sicherheit gesagt werden kann,

[15] Data Transfer Object: Siehe Kapitel 5.1.4
[16] https://doc.babylonjs.com/divingDeeper/webXR/WebXRSelectedFeatures#snap-to-hotspots

ist, dass die Möglichkeiten der Entwicklung sehr vielfältig ausfallen. Für crossplatform Lösungen eignet sich die Kombination von Webentwicklung und VR sehr gut. Einzig was die Performance angeht, lässt der aktuelle Stand der Technologien noch zu Wünschen übrig. In Zukunft wird das mit grosser Wahrscheinlichkeit immer weniger ein Problem darstellen. Die Entwicklung einer VR-Applikation mit Webtechnologien ist also für jeden zu empfehlen, der eine plattformübergreifende Lösung sucht und zu Kompromissen in der Performance bereit ist.

Literaturverzeichnis

[AndroidDocumentation, 2021] AndroidDocumentation (2021). Opengl es. https://developer.android.com/guide/topics/graphics/opengl.

[Blend4Web, 2021] Blend4Web (2021). Blend4web und blender. https://www.blend4web.com/en/technologies/blender.

[Bruls et al., 2000] Bruls, M., Huizing, K., and van Wijk, J. (2000). Squarified treemaps. In *de Leeuw W.C, van Liere R. (eds) Data Visualization 2000*. Springer, Vienna.

[Clinton, 2019] Clinton, J. (2019). *The City Metaphor in Software Visualization*. Václav Skala - UNION Agency.

[Dubenko, 2020] Dubenko, O. (2020). Write your first vr web application using react and webxr. https://blog.dubenko.dev/react-xr/.

[Loch et al., 2019] Loch, F., Böck, S., Zou, M., and Vogel-Heuser, B. (2019). Adapting virtual training systems for industrial procedures to the needs of older people. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 783–788.

[MDNWebDocs, 2021] MDNWebDocs (2021). Webxr device api. https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API.

[Milgram et al., 1994] Milgram, P., Takemura, H., Utsumi, A., and Kishino, F. (1994). Augmented reality: A class of displays on the reality-virtuality continuum. *ResearchGate*.

[Russo dos Santos et al., 2000] Russo dos Santos, C., Gros, P., Abel, P., Loisel, D., Trichaud, N., and Paris, J. (2000). Metaphor-aware 3d navigation. In *IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings*, pages 155–165.

[Steinbrückner and Lewerentz, 2010] Steinbrückner, F. and Lewerentz, C. (2010). Representing development history in software cities. In *Proceedings of the 5th International Symposium on Software Visualization*, SOFTVIS '10, page 193–202, New York, NY, USA. Association for Computing Machinery.

[TK, 2021] TK (2021). What is webxr and why webvr is dead. https://redstapler.co/what-is-webxr/.

[WebXR, 2021] WebXR (2021). Webxr device api explained. https://immersive-web.github.io/webxr/explainer.html#lifetime-of-a-vr-web-app.

[Wettel et al., 2011] Wettel, R., Lanza, M., and Robbes, R. (2011). Software systems as cities: A controlled experiment. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, page 551–560, New York, NY, USA. Association for Computing Machinery.

[Wikipedia, 2021a] Wikipedia (2021a). Augmented reality. https://en.wikipedia.org/wiki/Augmented_reality.

[Wikipedia, 2021b] Wikipedia (2021b). Treemapping. https://en.wikipedia.org/wiki/Treemapping.

[Wikipedia, 2021c] Wikipedia (2021c). Virtual reality. https://en.wikipedia.org/wiki/Virtual_reality.

Glossar

Abstract Syntax Tree In der Informatik ist ein abstrakter Syntaxbaum (AST) oder einfach nur ein Syntaxbaum eine Baumdarstellung der abstrakten syntaktischen Struktur eines in einer formalen Sprache geschriebenen Textes (oft Source Code). Siehe https://en.wikipedia.org/wiki/Abstract_syntax_tree. 34

AI Artificial Intelligence, AI, *Englisch* für KI. 5

Android Android ist ein mobiles Betriebssystem, das auf einer modifizierten Version des Linux-Kernels und anderer Open-Source-Software basiert und hauptsächlich für mobile Touchscreen-Geräte wie Smartphones und Tablets entwickelt wurde. Siehe https://en.wikipedia.org/wiki/Android_(operating_system). 6

AR Kurzform für Augmented Reality. 4, 6, 9, 12, 35

Augmented Reality *Englisch* für Erweiterte Realität. 3–5, 9, 11, 125

Base64 Base64 ist ein Verfahren zur Kodierung von 8-Bit-Binärdaten (z.B. ausführbare Programme, ZIP-Dateien oder Bilder) in eine Zeichenfolge, die nur aus lesbaren, Codepage-unabhängigen ASCII-Zeichen besteht. Siehe https://en.wikipedia.org/wiki/Nginx oder https://www.nginx.com. 54

Beschleunigungssensor Ein Beschleunigungssensor (auch Accelerometer oder G-Sensor) ist ein Sensor, der seine Beschleunigung misst. Dies erfolgt meistens, indem die auf eine Testmasse wirkende Trägheitskraft bestimmt wird. Somit kann z.B. bestimmt werden, ob eine Geschwindigkeitszunahme oder -abnahme stattfindet. Der Beschleunigungssensor gehört zur Gruppe der Inertialsensoren. Siehe https://de.wikipedia.org/wiki/Beschleunigungssensor. 4, 5

Blender 3D-Grafiktool für Modellierung, Textuierung und Animierung. Blender: https://www.blender.org. 15, 16, 114

C4 Das C4-Modell ist eine schlanke grafische Notationstechnik zur Modellierung der Architektur von Softwaresystemen. Siehe https://en.wikipedia.org/wiki/C4_model oder https://c4model.com. 50

CDN Ein Content Delivery Network (CDN), oder auch Content Distribution Network genannt, ist ein Netz regional verteilter und über das Internet verbundener Server, mit dem Inhalte ausgeliefert werden. Ein CDN stellt skalierende Speicher- und Auslieferungskapazitäten zur Verfügung und gewährleistet auch bei großen Lastspitzen einen optimalen Datendurchsatz. Siehe https://de.wikipedia.org/wiki/Content_Delivery_Network. 13, 14, 116

Chrome Chrome als Desktop Anwendung bietet sich speziell für die Entwicklung gut an, da über eine Emulatorextension ein WebXR Device emuliert werden kann. Siehe https://caniuse.com/webxr. 7

Code Smell In der Computerprogrammierung ist ein Code Smell ein Merkmal im Quellcode, das möglicherweise auf ein tiefer liegendes Problem hinweist.[1][2] Die Bestimmung, was ein Codegeruch ist und was nicht, ist subjektiv und variiert je nach Sprache, Entwickler und Entwicklungsmethodik. Siehe https://en.wikipedia.org/wiki/Code_smell. 9

Composite Pattern Komposition von Objekten in Baumstrukturen zur Darstellung von part-whole-Hierarchien. Ermöglicht den Clients eine einheitliche Behandlung einzelner Objekte und Kompositionen von Objekten. Siehe https://en.wikipedia.org/wiki/Composite_pattern. 34, 49, 55, 60, 80

CST Ein Concrete Syntax Tree oder auch Parse Tree ist ein geordneter Baum, der die syntaktische Struktur eines Strings gemäß einer kontextfreien Grammatik darstellt. 55

Dead Code Code, der zur Laufzeit niemals ausgeführt werden kann. Siehe https://en.wikipedia.org/wiki/Dead_code. 70, 130

Degrees of freedom *Englisch* für Freiheitsgrade. 5, 126

DOF Abkürzung für Degrees of freedom. 5

ECMAScript ECMAScript ist ein JavaScript-Standard, der die Interoperabilität von Webseiten mit verschiedenen Webbrowsern gewährleisten soll. Siehe https://en.wikipedia.org/wiki/ECMAScript. 6

Edge Edge bietet WebXR Untersützung. Siehe https://docs.microsoft.com/en-us/windows/mixed-reality/develop/javascript/webxr-overview. 7

Embedded Systems Ein eingebettetes System (auch englisch „embedded system“) ist ein elektronischer Rechner oder auch Computer, der in einen technischen Kontext eingebunden ist. Dabei übernimmt der Rechner entweder Überwachungs-, Steuerungs- oder Regelfunktionen oder ist für eine Form der Daten- bzw. Signalverarbeitung zuständig, beispielsweise beim Ver- bzw. Entschlüsseln, Codieren bzw. Decodieren oder Filtern. Siehe https://en.wikipedia.org/wiki/Embedded_system. 6

Erweiterte Realität Unter erweiterter Realität (auch englisch augmented reality [...]) versteht man die computergestützte Erweiterung der Realitätswahrnehmung. Wikipedia: https://de.wikipedia.org/wiki/Erweiterte_Realit%C3%A4t. 125

Firefox Reality Von Mozilla speziell für VR entwickelter Browser mit WebXR Unterstützung. Siehe https://mixedreality.mozilla.org/firefox-reality/ und https://blog.mozvr.com/firefox-reality-10/. 7

FPS FPS bezeichnet die Anzahl der Einzelbilder, die pro Sekunde aufgenommen oder wiedergegeben werden. Siehe https://de.wikipedia.org/wiki/Bildfrequenz. 25

Framework Ein Framework (Englisch für Rahmenstruktur) ist ein Programmiergerüst, das in der Softwaretechnik, insbesondere im Rahmen der objektorientierten Softwareentwicklung sowie bei komponentenbasierten Entwicklungsansätzen, verwendet wird. Siehe https://de.wikipedia.org/wiki/Framework. 1

Freiheitsgrade Freiheitsgrade (englisch Degrees of Freedom, auch DOF) bezeichnet die Anzahl Möglichkeiten, mit denen sich ein starres Objekt im dreidimensionalen Raum bewegen kann. Insgesamt gibt es sechs Freiheitsgrade, die jede mögliche Bewegung eines Objekts beschreiben: 3 für die Rotationsbewegungen, 3 für die Translationsbewegungen. 5, 126

Fully Dressed Ein Use Case im Fully Dressed Format beschreibt detailliert alle Schritte und Variationen. Siehe http://www.csci.csusb.edu/dick/cs375/fullydressed.html. 39

GNU GPL Die GNU General Public License ist eine Protective License (Coplyleft). Siehe https://www.gnu.org/licenses/gpl-3.0.de.html. 15

GNU LGPL Die GNU Lesser General Public License erlaubt den Entwicklern und Firmen das Verwenden und Einbinden von LGPL-Software in eigene (sogar proprietäre) Software, ohne durch ein starkes Copyleft gezwungen zu sein, den Quellcode der eigenen Software-Teile offenzulegen. Lediglich das Ändern der LGPL-Software-Teile muss Endnutzern ermöglicht werden: Deshalb werden im Falle von proprietärer Software die LGPL-Teile meist in Form einer dynamischen Programmbibliothek (z. B. DLL) verwendet, um so die notwendige Trennung zwischen proprietären und quelloffenen LGPL-Teilen zu ermöglichen. Siehe https://de.wikipedia.org/wiki/GNU_Lesser_General_Public_License oder https://www.gnu.org/licenses/lgpl-3.0.de.html. 14

GPU Eine *graphics processing unit* (GPU) ist ein spezialisierter elektronischer Schaltkreis, der den Speicher schnell manipulieren und verändern kann, um die Erstellung von Bildern in einem Bildpuffer für die Ausgabe an ein Anzeigegerät zu beschleunigen. GPUs werden in eingebetteten Systemen, Handys Siehe https://en.wikipedia.org/wiki/Graphics_processing_unit. 7

Gyroskop Ein Gyroskop ist ein Gerät, das zur Messung oder Aufrechterhaltung der Orientierung und Winkelgeschwindigkeit verwendet wird. Siehe https://en.wikipedia.org/wiki/Gyroscope. 4

haptisches Feedback Haptische Technologie, auch bekannt als kinästhetische Kommunikation oder 3D-Touch, bezieht sich auf jede Technologie, die durch die Anwendung von Kräften, Vibrationen oder Bewegungen auf den Benutzer ein Gefühl der Berührung erzeugen kann. Siehe https://en.wikipedia.org/wiki/Haptic_technology. 5, 8

HTML5 HTML5 ist eine Auszeichnungssprache, die für die Strukturierung und Darstellung von Inhalten im World Wide Web verwendet wird. Es ist die fünfte und letzte große HTML-Version, die eine Empfehlung des World Wide Web Consortium (W3C) ist. Die aktuelle Spezifikation ist als HTML Living Standard bekannt. Sie wird von der Web Hypertext Application Technology Working Group (WHATWG) gepflegt, einem Konsortium der wichtigsten Browserhersteller (Apple, Google, Mozilla und Microsoft). Siehe https://en.wikipedia.org/wiki/HTML5. 6

Immersion Die Wahrnehmung, physisch in einer nicht-physischen Welt anwesend zu sein. Siehe https://www.lifewire.com/what-is-an-immersive-experience-4588346. 4, 5

Intertiale Messeinheit Eine inertiale Messeinheit (englisch inertial measurement unit, IMU) ist eine räumliche Kombination mehrerer Inertialsensoren wie Beschleunigungssensoren und Drehratensensoren. Sie stellt die sensorische Messeinheit eines Trägheitsnavigationssystems (englisch Inertial Navigation System, INS) dar. Siehe https://de.wikipedia.org/wiki/Inertiale_Messeinheit. 5

Java Java ist eine high-level, klassenbasierte, objektorientierte Programmiersprache, die so konzipiert ist, dass sie so wenig Implementierungsabhängigkeiten wie möglich aufweist. Siehe https://en.wikipedia.org/wiki/Java_(programming_language). 10

Khronos Siehe Khronos Group. 6

Khronos Group Die Khronos Group ist ein im Jahr 2000 gegründetes Industriekonsortium, das sich für die Erstellung und Verwaltung von offenen Standards im Multimedia-Bereich auf einer Vielzahl von Plattformen und Geräten einsetzt. Zu den über 100 Mitgliedern zählen unter anderem AMD, Intel, NVIDIA, SGI, Apple, Microsoft, Google sowie Oracle. Siehe https://de.wikipedia.org/wiki/Khronos_Group. 6, 128

KI Künstliche Intelligenz (KI), auch artifizielle Intelligenz (AI bzw. A. I.), englisch artificial intelligence (AI bzw. A. I.) ist ein Teilgebiet der Informatik, das sich mit der Automatisierung intelligenten Verhaltens und dem maschinellen Lernen befasst. 125

Latenz Verzögerungszeit, in unterschiedlichen Zusammenhängen auch Reaktionszeit oder Latenz(zeit) genannt, ist der Zeitraum zwischen einer Aktion oder einem Ereignis und dem Eintreten einer Reaktion. Siehe https://de.wikipedia.org/wiki/Verzögerungszeit. 4

Mesh In 3D-Computergrafiken ist ein Mesh eine Sammlung von Vertices, Edges und Faces, die die Form eines polyedrischen Objekts definieren. Siehe https://en.wikipedia.org/wiki/Polygon_mesh. 105, 106, 109–111, 114, 118

Mixed Reality Unter Mixed reality, Vermischte Realität oder Gemischte Realität werden Umgebungen oder Systeme zusammengefasst, die die natürliche Wahrnehmung eines Nutzers mit einer künstlichen (computererzeugten) Wahrnehmung vermischen. Neben der hauptsächlich computererzeugten virtuellen Realität sind dies insbesondere Systeme der erweiterten Realität und der erweiterten Virtualität. 4, 5, 11

NGINX Nginx ist ein Webserver, der auch als Reverse Proxy, Load Balancer, Mail Proxy und HTTP Cache verwendet werden kann. Siehe https://en.wikipedia.org/wiki/Nginx oder https://www.nginx.com. 51, 73

npm Node Package Manager. Siehe https://www.npmjs.com. 13, 14, 55, 76, 112, 116

Observer Pattern Das Observer Pattern ist ein Design Pattern, bei dem ein Objekt, das Subject genannt wird, eine Liste seiner Observer unterhält und diese automatisch über Zustandsänderungen benachrichtigt, normalerweise durch den Aufruf einer ihrer Methoden. Siehe https://en.wikipedia.org/wiki/Nginx oder https://www.nginx.com. 59

Oculus Browser Von Oculus speziell für ihre Headsets entwickelter VR Browser mit WebXR Unterstützung. Siehe https://www.oculus.com/experiences/quest/1916519981771802/?locale=en_US und https://developer.oculus.com/webxr/. 7

Oculus Quest Oculus Quest ist ein Virtual Reality-Headset, das von Oculus VR, einer Tochter von Facebook, entwickelt wurde. Es handelt sich dabei um ein Standalone-Gerät. Siehe https://de.wikipedia.org/wiki/Oculus_Quest. 1

omnidirektional Funkwellen in alle Richtungen gleich gut empfangen oder senden. Siehe https://www.merriam-webster.com/dictionary/omnidirectional. 107

OpenGL OpenGL (Open Graphics Library) ist eine Spezifikation einer plattform- und programmiersprachenübergreifenden API zur Entwicklung von 2D- und 3D-Computergrafikanwendungen. Siehe https://en.wikipedia.org/wiki/OpenGL_ES. 6, 11

OpenGL ES Open Graphics Library for Embedded Systems (kurz OpenGL ES geschrieben) ist eine Spezifikation für eine plattform- und sprachenunabhängige Programmierschnittstelle zur Entwicklung von 3D-Computergrafik für Embedded Systems. Siehe https://en.wikipedia.org/wiki/OpenGL_ES. 6, 7

PBR Physically Based Rendering (PBR), auch Physically Based Shading, ist eine Technik zur Erzeugung fotorealistischer 3D-Grafiken. Siehe https://de.wikipedia.org/wiki/Physically_Based_Rendering. 119

Progressive Web App Eine Progressive Web App (PWA) ist eine Website, die zahlreiche Merkmale besitzt, die bislang nativen Apps vorbehalten waren. Sie kann daher auch als Symbiose aus einer responsiven Webseite und einer App beschrieben werden. Wikipedia: https://de.wikipedia.org/wiki/Progressive_Web_App. 12

Shadow Acne Shadow Acne ist ein Problem, das durch die begrenzte Auflösung der Shadow Map verursacht wird. Siehe https://digitalrune.github.io/DigitalRune-Documentation/html/3f4d959e-9c98-4a97-8d85-7a73c26145d7.htm. 107

Single Page Application Als Single-Page-Webanwendung (englisch single-page application, kurz SPA) oder Einzelseiten-Webanwendung wird eine Webanwendung bezeichnet, die aus einem einzigen HTML-Dokument besteht und deren Inhalte dynamisch nachgeladen werden. Wikipedia: https://de.wikipedia.org/wiki/Single-Page-Webanwendung. 12, 51

State Pattern Das State Pattern ist ein behavioral Design Pattern, das es einem Objekt ermöglicht, sein Verhalten zu ändern, wenn sich sein interner Zustand ändert. Siehe https://en.wikipedia.org/wiki/State_pattern. 53

TOF-Kamera TOF-Kameras sind 3D-Kamerasysteme, die mit dem Laufzeitverfahren (englisch: time of flight, TOF, auch ToF) Distanzen messen (https://de.wikipedia.org/wiki/TOF-Kamera). Das resultierende Bild nennt sich *Range Image* hat Pixelwerte, die der jeweiligen Entfernung entsprechen (https://en.wikipedia.org/wiki/Range_imaging). 5

Translation Eine Translation (auch reine Translation, lineare Bewegung) ist eine Bewegung, bei der alle Punkte eines physikalischen Systems, z. B. eines starren Körpers, dieselbe Verschiebung erfahren. Siehe https://de.wikipedia.org/wiki/Translation_(Physik). 5

Tree Shaking Tree Shaking ist eine Optimierungstechnik zur Beseitigung von Dead Code. Im Gegensatz zu den traditionellen Techniken zur Eliminierung von Dead Code, wie sie bei Minimizern üblich sind, werden bei Tree Shaking ungenutzte Funktionen aus dem gesamten Bundle eliminiert, indem am Einstiegspunkt begonnen wird und nur Funktionen einbezogen werden, die ausgeführt werden. Siehe https://en.wikipedia.org/wiki/Tree_shaking. 70, 119

UI User Interface. 37

Umgebungslichtsensor Ein Umgebungslichtsensor (englisch ambient light sensor, abgekürzt ALS) hat die Aufgabe das Umgebungslicht zu messen. Siehe https://de.wikipedia.org/wiki/Umgebungslichtsensor. 5

UML Die Unified Modeling Language, kurz UML, ist eine grafische Modellierungssprache zur Spezifikation, Konstruktion, Dokumentation und Visualisierung von Software-Teilen und anderen Systemen. Siehe https://de.wikipedia.org/wiki/Unified_Modeling_Language. 74

Virtual Reality *Englisch* für Virtuelle Realität. 1, 3–5, 7, 11, 35

Virtuelle Realität Als virtuelle Realität, kurz VR, wird die Darstellung und gleichzeitige Wahrnehmung der Wirklichkeit und ihrer physikalischen Eigenschaften in einer in Echtzeit computergenerierten, interaktiven virtuellen Umgebung bezeichnet. Wikipedia: https://de.wikipedia.org/wiki/Virtuelle_Realit%C3%A4t. 1, 130

Visitor Pattern GOF Definition: Stellt ein Operation dar, die an den Elementen einer Objektstruktur durchzuführen ist. Ermöglicht es, eine neue Operation zu definieren, ohne die Klassen der Elemente zu ändern, auf der sie operiert. Siehe https://en.wikipedia.org/wiki/Visitor_pattern. 34, 55

Vive HTC Vive ist ein Virtual Reality-Headset, das von HTC in Kooperation mit Valve produziert wird. Siehe https://de.wikipedia.org/wiki/HTC_Vive. 1

VR Kurzform für Virtuelle Realität. 4, 6, 7, 9, 12

W3C Das World Wide Web Consortium (W3C) ist die wichtigste internationale Standardisierungsorganisation für das World Wide Web. Siehe https://en.wikipedia.org/wiki/World_Wide_Web_Consortium. 6

WebGL WebGL ist eine JavaScript-API zum Rendern interaktiver 2D- und 3D-Grafiken in jedem kompatiblen Webbrowser ohne die Verwendung von Plug-ins. WebGL ist vollständig in andere Webstandards integriert und ermöglicht die GPU-beschleunigte Nutzung von Physik, Bildverarbeitung und Effekten als Teil der Webseiten-Leinwand. Siehe https://en.wikipedia.org/wiki/WebGL. 6, 7, 11, 13–15

WebVR WebVR wurde durch die WebXR Device API ersetzt, die eine breitere Unterstützung, mehr Funktionen und eine bessere Leistung bietet und sowohl VR als auch AR unterstützt. Siehe https://webvr.info. 12

WebXR WebXR ist eine Gruppe von Standards, die zusammen verwendet werden, um das Rendern von 3D-Szenen auf Hardware zu unterstützen, die für die Darstellung virtueller Welten (Virtual Reality, VR) oder für das Hinzufügen grafischer Bilder zur realen Welt (Augmented Reality, AR) entwickelt wurde. Siehe https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API. 5–7, 11, 12, 14, 15, 17, 19, 20, 36, 73, 104, 114–116

WebXR Device API Die WebXR Device API implementiert den Kern des WebXR-Funktionssatzes, verwaltet die Auswahl der Ausgabegeräte, rendert die 3D-Szene auf dem gewählten Gerät und verwaltet die Bewegungsvektoren der Eingabesteuerungen Input Controllers. Siehe https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API. 6, 73, 114

Appendix

Anhang A

Verifikations- und Validationstests

In diesem Anhang sind die Testergebnisse ausführlich dokumentiert. Die erste Tabelle bezieht sich auf die Testcoverage der Businesslogik durch Unit- und Integrationtests. Die nachfolgenden Tabellen beinhalten die Resultate der Systemtests und anderen nichtfunktionalen Tests.

Test Coverage Unit- und Integrationtests

Insgesamt wurden 118 Unit- und Integrationtests geschrieben welche über 35 Test Suites aufgeteilt sind.

Tabelle A.1: Test Coverage Unit- und Integrationtests

File	% **Stmts**	% **Branch**	% **Funcs**	% **Lines**
src	0	100	100	0
index.ts	0	100	100	0
src/businessLogic/ cityLayoutBuilder	86.66	100	75	89.28
cityLayoutBuildStrategy.ts	100	100	100	100
cityLayoutBuilder.ts	85.18	100	75	88
src/businessLogic/ cityLayoutBuilder/strategies	96.66	89.53	96.87	96.56
cityLayoutSquarifiedTreemap Strategy.ts	95.65	89.47	95.45	95.55
cityLayoutTreemapStrategy.ts	98.55	87.5	100	98.48
strategies.ts	100	100	100	100
src/businessLogic/ cityLayout-Builder/strategies/areas	100	100	100	100
area.ts	100	100	100	100
squarifiedArea.ts	100	100	100	100
src/businessLogic/ codeAnalyzer	90.24	75.86	85	91.25
javaAnalyzer.ts	80.64	66.66	75	82.75
metricsCollector.ts	96.07	76.92	91.66	96.07
src/dataLayer/githubProxy	86.31	69.56	90	87.09
githubApi.ts	85.71	70	100	85.71
javaFileCollector.ts	97.05	85.71	100	97.05
projectDataProxy.ts	77.5	50	77.77	78.94

src/dataLayer/model/ configuration	100	100	100	100
appConfig.ts	100	100	100	100
enums.ts	100	100	100	100
src/dataLayer/model/ projectData	93.18	80	93.75	93.18
githubInformation.ts	100	100	100	100
projectData.ts	91.89	80	91.66	91.89
src/dataLayer/model/ projectData/layout	70	62.5	45	70
cityComponent.ts	81.25	100	28.57	81.25
dimension.ts	100	100	100	100
position.ts	100	100	100	100
quarter.ts	53.84	0	40	53.84
src/dataLayer/model/ projectData/layout/cityItems	74.6	55.55	65	73.77
building.ts	60	12.5	55.55	57.14
buildingFrontYard.ts	91.66	85.71	100	91.66
cityItem.ts	81.81	100	33.33	81.81
street.ts	90	100	75	90
src/dataLayer/model/ projectData/processed	79.03	82.35	57.14	87.03
javaPackage.ts	74.5	72.72	52	83.72
processedComponent.ts	100	100	100	100
src/dataLayer/model/ projectData/processed/javaItems	92.3	92.3	86.66	92.3
javaClass.ts	100	100	100	100
javaEnum.ts	100	100	100	100
javaInterface.ts	100	100	100	100
javaItem.ts	83.33	100	63.63	83.33
metrics.ts	92.85	87.5	100	92.85
src/dataLayer/model/ projectData/unprocessed	70.21	100	60.86	73.33
unprocessedComponent.ts	100	100	100	100
unprocessedDirectory.ts	71.42	100	57.14	78.94
unprocessedFile.ts	52.94	100	57.14	52.94
src/utils/errorhandling	75	100	62.5	75
Errors.ts	75	100	62.5	75
All files	**86.86**	**81.06**	**77.36**	**87.79**

System Tests

Der letzte Systemtest wurde lokal mit dem Tag *system_test* durchgeführt.

Durchführung:

1. `npm start`

2. im Browser `https://localhost:8080` eingeben

UC1: Übersicht über Softwareprojekt erhalten

Tabelle A.2: Systemtest: GitHub Daten angeben

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
User hat ein öffentliches GitHub Repository mit Java-Files	1. User gibt GitHub-Name und -Repository an 2. User klickt "Check Repository"-Button 3. User klickt "Build City"-Button	GitHub Repository kann geladen werden
User gibt privates Repositoy an, oder hat Typo in den Angaben	1. User gibt fehlerhafte Angaben oder privates GitHub-Repository an 2. User klickt "Check Repository"-Button	User wird informiert, dass kein Repository gefunden wurde
User gibt Repositoy an, welches keine Java-Files beinhaltet	1. User gibt GitHub-Name und -Repository an 2. User klickt "Check Repository"-Button 3. User klickt "Build City"-Button	User wird informiert, dass im Repository keine Java-Files gefunden wurden
User gibt Repositoy an, mit Java-Files, die keinen validen Java-Code beinhalten	1. User gibt GitHub-Name und -Repository an 2. User klickt "Check Repository"-Button 3. User klickt "Build City"-Button	User wird informiert, welche Files aufgrund falschem Inhalt ignoriert wurden
User gibt ein beliebiges Repositoy an	1. User gibt GitHub-Name und -Repository an 2. User klickt "Check Repository"-Button	User wird mit einer Fehlermeldung informiert, falls die GitHub REST API nicht mehr verfügbar ist

Tabelle A.3: Systemtest: Stadtlayout erstellen

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
GitHub Daten müssen erfolgreich geladen und geparst sein	1. User bestätigt Repository mit einem Klick auf den "Build City"-Button	Stadtlayout wurde erstellt und ist bereit für das Rendering

Tabelle A.4: Systemtest: Stadt besichtigen

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
Stadlayout ist erstellt und die Stadt gerendert	1. User klickt ”Enter VR”-Button	User is im Immersive-Mode. User kann sich mit Joystick teleportieren oder drehen und das Menu ein- & ausblenden

Tabelle A.5: Systemtest: Eigenschaften anzeigen

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
Die Stadt ist gerendert und der User im Immersive-Mode	1. User wählt ein Gebäude aus	Ein Fenster mit den Eigenschaften des Gebäudes wird eingeblendet
	1. User wählt mehrere Gebäude aus	Für jedes Gebäude wird ein Fenster eingeblendet
	1. User wählt ein oder mehrere Gebäude erneut aus (abwählen)	Für das abgewählte Gebäude wird das entsprechende Fenster ausgeblendet

Tabelle A.6: Systemtest: Suche nach Komponenten

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
Die Stadt ist gerendert und der User im Immersive-Mode	1. User öffnet das Menu (A-Knopf) 2. User wechselt in das Menu ”Search” 3. User gibt existierenden Namen einer Klasse, eines Interfaces, oder Enums ein 4. User drückt den ”Search”-Button	Gesuchtes Element wird gehighlighted und die Eigenschaften werden in einem neuen Fenster angezeigt
	1. User öffnet das Menu (A-Knopf) 2. User wechselt in das Menu ”Search” 3. User gibt inexistenten Namen einer Klasse, eines Interfaces, oder Enums ein 4. User drückt den ”Search”-Button	User wird informiert, dass das gesuchte Objekt nicht gefunden wurde

Tabelle A.7: Systemtest: Anderes Projekt begutachten

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
Die Stadt ist gerendert und der User im Immersive-Mode	1. User öffnet das Menu (A-Knopf) 2. User wechselt in das Menu "Settings" 3. User klickt "Exit VR"-Button	Der User verlässt den Immersive-Mode und hat wieder den Browser vor sich. Wenn der User erneut auf den "Enter VR"-Button klickt kommt er in den selben Immersive-State wie er ihn zuvor verlassen hat
	1. User drückt den Home-Button des Controllers 2. User klickt den "Quit"-Button	

Tabelle A.8: Systemtest: Hinweis zur Performance

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
Der User befindet sich auf der Website	1. User scrollt runter in das Menu *How it works*	User findet den Hinweis, dass sich bei schwächerer Hardware und grösseren Projekten die Performance verschlechtern kann

UC2: Nach Eigenschaften filtern können

Tabelle A.9: Systemtest: Nach Metriken filtern

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
Die Stadt ist gerendert und der User im Immersive-Mode	1. User öffnet das Menu (mit A-Knopf) 2. User wechselt in das Menu "Filter" 3. User setzt Metriken und Min/Max wie gewünscht 4. User bestätigt mit "Filter Metrics"-Button	Stadt wird neu gerendert. Es werden nur noch die gewünschten Gebäude angezeigt
	1. User öffnet das Menu (mit A-Knopf) 2. User wechselt in das Menu "Filter" 3. User wählt aus, ob er Klassen/Interfaces/Enums sehen möchte 4. User bestätigt mit "Filter Metrics"-Button	
User hat bereits ein- oder mehrmals den Filter angepasst	1. User öffnet das Menu (mit A-Knopf) 2. User wechselt in das Menu "Filter" 3. User setzt Metriken und Min/Max wie gewünscht 4. User wählt aus, ob er Klassen/Interfaces/Enums sehen möchte 5. User bestätigt mit "Filter Metrics"-Button	

Tabelle A.10: Systemtest: Schwellwertüberschreitende Gebäude markieren

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
Stadlayout ist erstellt und die Stadt gerendert	1. User klickt ”Enter VR”-Button	Stadt ist gerendert. Alle Gebäude, welche einen definierten Schwellwert überschreiten, werden markiert
Die Stadt ist gerendert und der User im Immersive-Mode	1. User öffnet das Menu (mit A-Knopf) 2. User wechselt in das Menu ”Settings” 3. User setzt Schwellwerte wie gewünscht 4. User bestätigt mit ”Customize”-Button	Stadt wird neu gerendert. Alle Gebäude, welche einen Schwellwert überschreiten, werden markiert

UC3: Styling der Stadt oder Umgebung anpassen

Tabelle A.11: Systemtest: Tageszeit einstellen

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
Die Stadt ist gerendert und der User im Immersive-Mode	1. User öffnet das Menu (mit A-Knopf) 2. User wechselt in das Menu ”View” 3. User klickt den Button in der Zeile ”Time” 4. User passt Zeit mit seinem Controller an 5. User bestätigt Zeit mit erneutem klick auf den Button	Die Zeit wurde angepasst, die Umgebung (Licht & Schatten) entsprechend verändert

Tabelle A.12: Systemtest: Stadtbild verändern

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
Die Stadt ist gerendert und der User im Immersive-Mode	1. User öffnet das Menu (mit A-Knopf) 2. User wechselt in das Menu "View" 3. User klickt den Button in der Zeile "Enhanced City"	Die Stadt wurde neu gerendert, es werden richtige Gebäude und nicht mehr nur Blöcke angezeigt. Bäume, Fahrzeuge und weitere Objekte schmücken die Stadt
Der User hat das Stadtbild bereits "erweitert/enhanced"	1. User öffnet das Menu (mit A-Knopf) 2. User wechselt in das Menu "View" 3. User klickt den Button in der Zeile "Enhanced City"	Die Stadt wurde neu gerendert, es werden nur noch Blöcke als Gebäude angezeigt. Bäume, Fahrzeuge und weitere Objekte schmücken die Stadt nicht mehr

UC6: Sich in der Stadt forbewegen können (optional)

Tabelle A.13: Systemtest: Ansicht auf die Stadt ändern

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
Die Stadt ist gerendert, der User im Immersive-Mode und sieht die Stadt in der Vogelperspektive	1. User öffnet das Menu (mit A-Knopf) 2. User wechselt in das Menu "View" 3. User klickt den Button in der Zeile "Visit City" 4. Der User wählt den Punkt in der Stadt wo er hinteleportiert werden möchte	Der User ist nun in der Stadt (Froschperspektive)
	1. User öffnet das Menu (mit A-Knopf) 2. User wechselt in das Menu "View" 3. User klickt den Button in der Zeile "Visit City" 4. Der User wählt ein Gebäude in der Stadt wo er hinteleportiert werden möchte	Der User ist nicht teleportiert und wird benachrichtigt, dass dieser Ort nicht betretbar ist
	1. User öffnet das Menu (mit A-Knopf) 2. User wechselt in das Menu "View" 3. User klickt den Button in der Zeile "Visit City" 4. Der User wählt einen Punkt ausserhalb der Stadt wo er hinteleportiert werden möchte	
Die Stadt ist gerendert, der User im Immersive-Mode und steht in der Stadt (Froschperspektive)	1. User öffnet das Menu (mit A-Knopf) 2. User wechselt in das Menu "View" 3. User klickt den Button in der Zeile "Leave City"	Der User ist nun wieder über Stadt (Vogelperspektive)

Tabelle A.14: Systemtest: In der Stadt fortbewegen

Voraussetzungen	**Ablauf**	**Test erfüllt wenn:**
Die Stadt ist gerendert, der User im Immersive-Mode und steht in der Stadt (Froschperspektive)	1. User bewegt den Joystick eines Controllers nach links oder rechts	Die Ansicht des Users auf die Stadt rotiert gemäss seiner Eingabe
	1. User bewegt den Joystick eines Controllers nach vorne	Dem User wird ein Symbol auf dem Punkt angezeit wo er hinteleportiert wird, falls er den Joystick loslässt. Wenn er ihn loslässt wird er auf diesen Punkt bewegt
	1. User bewegt den Joystick eines Controllers nach hinten	Der User bewegt sich ein Stück nach hinten
Die Stadt ist gerendert, der User im Immersive-Mode und schwebt über der Stadt (Vogelperspektive)	1. User bewegt den Joystick eines Controllers nach links oder rechts	Die Ansicht des Users auf die Stadt rotiert gemäss seiner Eingabe
	1. User bewegt den Joystick eines Controllers nach vorne oder hinten	Es passiert nichts

Protokoll

Tabelle A.15: Systemtest: Systemtestprotokoll

Tests	Implementiert	Fehler/Unsicherheit	Status
UC1: GitHub Daten angeben	ja	keine	erfolgreich
UC1: Stadtlayout erstellen	ja	keine	erfolgreich
UC1: Stadt besichtigen	ja	keine	erfolgreich
UC1: Eigenschaften anzeigen	ja	keine	erfolgreich
UC1: Nach Komponenten suchen	ja	keine	erfolgreich
UC1: Stadtteile ausblenden	nein	-	-
UC1: Anderes Projekt begutachten	ja	keine	erfolgreich
UC1: Hinweis zur Performance	ja	Der Hinweis erscheint nur auf der Webseite. Schöner wäre es, wenn bei der Benutzer bei der Eingabe von grösseren Projekten auf die Performance hingewiesen wird.	erfolgreich
UC2: Nach Metriken filtern	ja	keine	erfolgreich
UC2: Schwellwert überschreitende Gebäude markieren	ja	keine	erfolgreich
UC3: Tageszeit einstellen	ja	keine	erfolgreich
UC3: Stadtbild verändern	ja	Falls das Stadtbild zu schnell verändert wird, können sich die Gebäude überlappen.	erfolgreich
UC3: Styling anpassen (Farben der Gebäude)	nein	-	-
UC6: Ansicht auf die Stadt ändern	ja	keine	erfolgreich
UC6: In der Stadt fortbewegen	ja	keine	erfolgreich

Bekannte Einschränkungen

Tabelle A.16: Systemtest: Bekannte Einschränkungen

Use Case	Einschränkung
UC1	Einzelne Gebäude oder ganze Stadtteile können nicht ausgeblendet werden (ausser durch Filter).
	Die Suche erfolgt ausschliesslich über den Gebäudenamen und der Nutzer muss den Namen exakt eintippen.
UC2	Dem User wird nicht genügend gut vermittelt, ob und wo sein Code gut oder schlecht ist. Die Schwellwerte können nach Belieben angepasst werden, sind somit kein Anhaltspunkt für Codequalität.
UC3	Der User kann die Farben der Stadt nicht nach seinen eigenen Wünschen anpassen.

Nichtfunktionale Tests

Alle Nichtfunktionalen Anforderungen (Kapitel 4.2), die nicht bereits getestet wurden, werden hier noch verifiziert. Die Testresultate beziehen sich auf die Beschreibungen im Kapitel 4.2.

Reliability

Tabelle A.17: Test: Reliability

Nr	NFR	Beschreibung	Status
04	Recoverability	Da der Server von sich aus einmal abgestürzt ist, musste kein Absturz simuliert werden, um diese Anforderung zu testen. Nachdem der Absturz aufgefallen war, konnte der Server innerhalb von 10 Minuten wieder zum laufen gebracht werden, da nur die Docker Instanz wieder gestartet werden musste.	Erfolgreich

Performance Efficiency

Tabelle A.18: Test: Performance Efficiency

Nr	NFR	Beschreibung	Status
05	Time Behavior	Das Referenzprojekt[1] konnte beim Performancetest im Schnitt über 5 Messungen in ca. 40s geladen, analysiert und gerendert werden. Zu beachten ist, dass der Zeitaufwändigste Teil das Abfragen der Daten ist und der Test mit einer relativ guten Verbindung von 300Mbps durchgeführt wurde.	Erfolgreich
07	Capacity	Der Test wurde mit einer Oculus Quest 2 und dem Oculus Browser durchgeführt. Das Referenzprojekt[2] mit 67 Gebäuden konnte ohne Schatten und mit den primitiven Gebäuden mit etwa 20 FPS dargestellt werden. Mit den Blender Gebäuden waren es noch ca. 10 FPS. Mit Blender Gebäuden und eingeschaltenem Schatten waren es noch ca. 7 FPS. Der grafische Rechenaufwand wurde somit komplett unterschätzt.	Nicht erfolgreich

Compatibility

Tabelle A.19: Test: Compatibility

Nr	NFR	Beschreibung	Status
08	Interoperability	Die Applikation wurde auf dem Oculus Browser und Firefox Reality getestet. Die Applikation funktioniert auf beiden Browsern, wobei die Performance auf Firefox Reality viel schlechter war.	Erfolgreich

Maintainability

Tabelle A.20: Test: Maintainability

Nr	NFR	Beschreibung	Status
12	Modularity	Damit Parser und Layout Builder ersetzt werden können, ist im Kapitel Softwarearchitektur beschrieben, wie die Übergabe der Daten zwischen den Komponenten aussieht.	Erfolgreich

[1] https://github.com/amir650/BlackWidow-Chess/tree/1a8716160dc188842377b2d3ea486856c344da10
[2] https://github.com/amir650/BlackWidow-Chess/tree/1a8716160dc188842377b2d3ea486856c344da10

13	Modifiability	Die Test Coverage der Business Logik beträgt über 80% und besteht aus Unit und Integration Tests.	Erfolgreich
14	Testability	Alle Merge Requests wurden erst dann akzeptiert, wenn die Pipeline erfolgreich durchgelaufen ist. Die Pipeline hat folgende Schritte durchgeführt: Linter durchlaufen lassen, Zirkuläre Abhängigkeiten überprüft, Tests ausgeführt, Test Coverage überprüft.	Erfolgreich

Portability

Tabelle A.21: Test: Portability

Nr	NFR	Beschreibung	Status
15	Installability	Das Projekt konnte erfolgreich auf einer Linux Umgebung deployt werden. Für die Projektübergabe wurde ausserdem eine Installationsanleitung erstellt.	Erfolgreich

Anhang B

Lizenzen und externe Ressourcen

B.1 Verwendete Lizenzen

Nachfolgend werden die Lizenzen aller verwendeten Libraries und Frameworks aufgelistet. Verwendet wurden die Lizenzen: MIT, Apache-2.0, ISC und BSD-2-Clause.

Tabelle B.1: Verwendete Lizenzen

Package	Lizenz	Repository
@babylonjs/core	Apache-2.0	github.com/BabylonJS/Babylon.js
@babylonjs/loaders	Apache-2.0	github.com/BabylonJS/Babylon.js
@babylonjs/gui	Apache-2.0	github.com/BabylonJS/Babylon.js
@babylonjs/materials	Apache-2.0	github.com/BabylonJS/Babylon.js
typescript	Apache-2.0	github.com/Microsoft/TypeScript
Jest	MIT	github.com/facebook/jest
ts-jest	MIT	github.com/kulshekhar/ts-jest
eslint	MIT	github.com/eslint/eslint
eslint-config-google	Apache-2.0	github.com/google/eslint-config-google
eslint-config-standard	MIT	https://github.com/standard/eslint-config-standard
eslint-plugin-import	MIT	https://github.com/import-js/eslint-plugin-import
eslint-plugin-node	MIT	https://github.com/mysticatea/eslint-plugin-node
eslint-plugin-promise	ISC	https://github.com/xjamundx/eslint-plugin-promise

@typescript-eslint/eslint-plugin	MIT	https://www.npmjs.com/package/@typescript-eslint/eslint-plugin
@typescript-eslint/parser	BSD-2-Clause	https://github.com/typescript-eslint/typescript-eslint
madge	MIT	github.com/pahen/madge
webpack	MIT	github.com/webpack/webpack
webpack-cli	MIT	github.com/webpack/webpack-cli
webpack-dev-server	MIT	https://github.com/webpack/webpack-dev-server
webpack-merge	MIT	github.com/survivejs/webpack-merge
copy-webpack-plugin	MIT	https://github.com/webpack-contrib/copy-webpack-plugin
clean-webpack-plugin	MIT	https://github.com/johnagan/clean-webpack-plugin
css-loader	MIT	github.com/webpack-contrib/css-loader
file-loader	MIT	github.com/webpack-contrib/file-loader
html-webpack-plugin	MIT	https://github.com/jantimon/html-webpack-plugin
source-map-loader	MIT	https://github.com/webpack-contrib/source-map-loader
style-loader	MIT	https://github.com/webpack-contrib/style-loader
ts-loader	MIT	github.com/TypeStrong/ts-loader
java-parser	Apache-2.0	https://github.com/jhipster/prettier-java/tree/main

B.2 3D-Modelle

Für eine belebtere Stadt wurden einige 3D-Modelle verwendet, die nicht von uns erstellt wurden. Zum Teil wurden diese mit Blender angepasst.

- cgtrader für die Gebäudemodelle
- TurboSquid für diverse Fahrzeuge

B.3 Icons

Für das GUI wurden Icons von flaticon übernommen:

Anhang C

Bilder

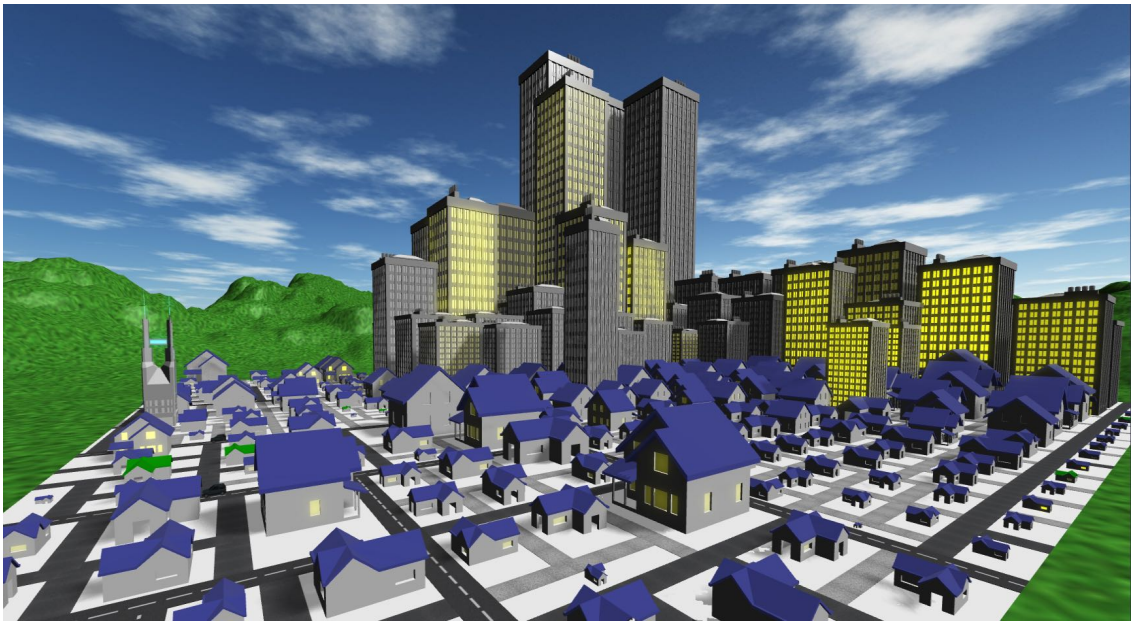


Abbildung C.1: Beliebige Stadt

Abbildung C.2: Beliebige Stadt bei Sonnenaufgang

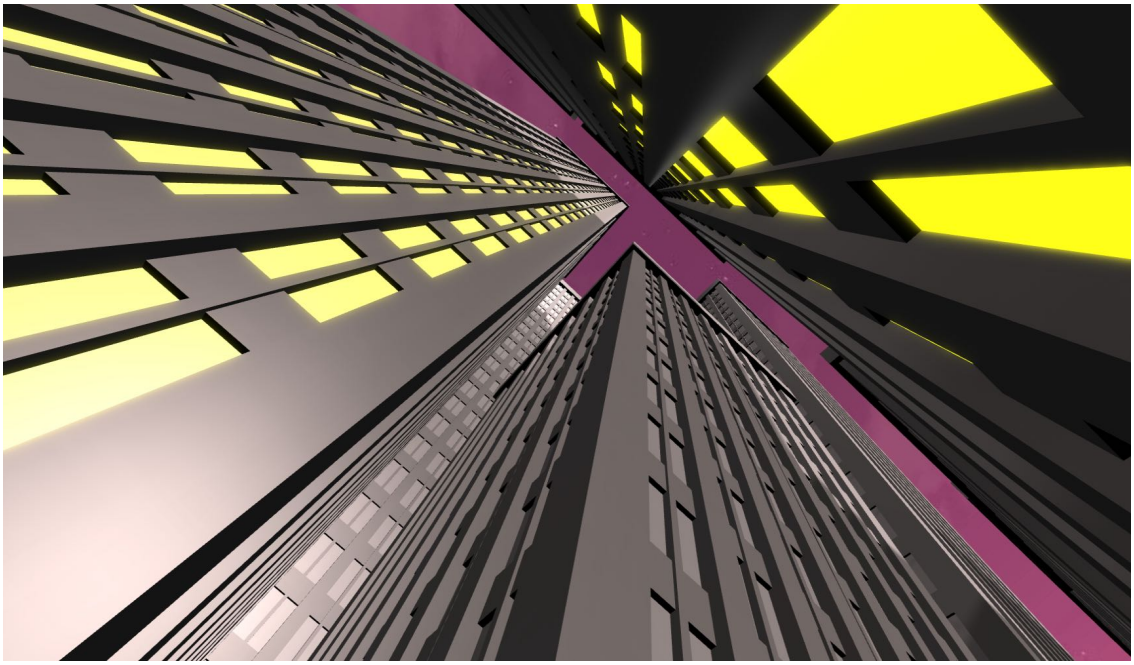


Abbildung C.3: Wolkenkratzer bei Sonnenaufgang

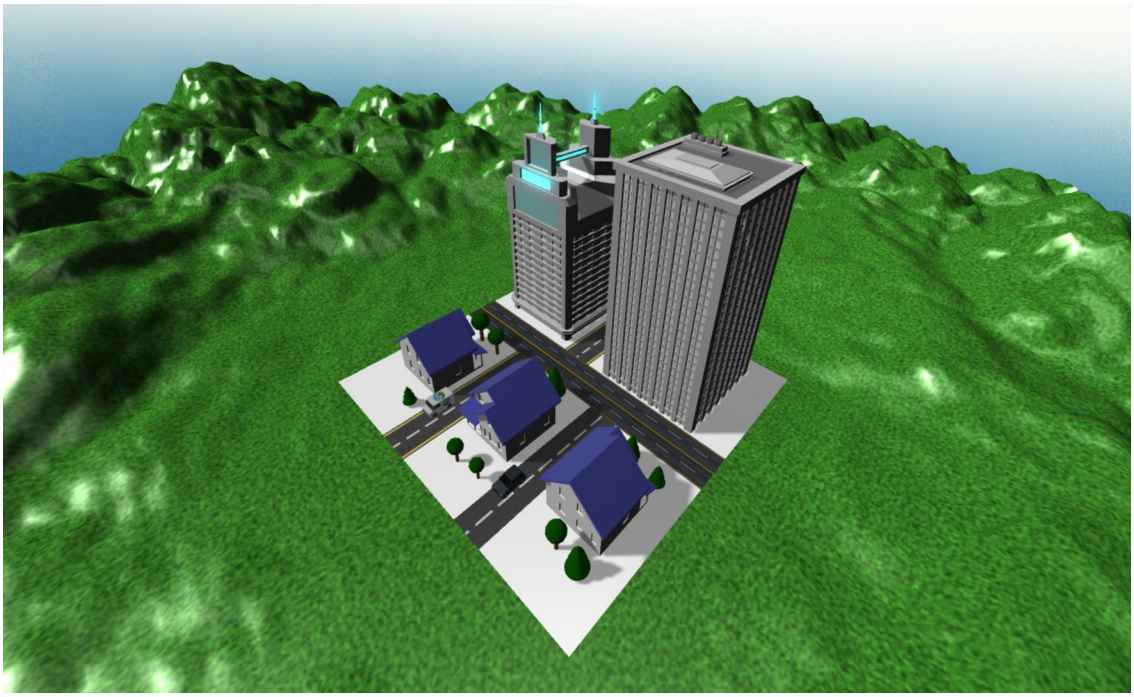


Abbildung C.4: Kleine Stadt