Studienarbeit
Dokumentation

3D-Visualisierung von OSM-Daten in Godot

Herbstsemester 2023

Datum: 2023-12-22 15:00

Projekt Team	Fabian Freitag Josip Di Benedetto
Projekt Betreuer	Stefan F. Keller
Industriepartner	Pirmin Kalberer Sourcepole AG

Departement Informatik - IFS Institut für Software
Ostschweizer Fachhochschule OST Rapperswil-Jona

Inhaltsverzeichnis

Teil I

Abstract

Abstract

Bisher vorhandene Projekte, welche OpenStreetMap (OSM) Daten als 3D-Welt darstellen, haben aus Sicht des Industriepartners zwei Probleme: Das erste Problem ist, dass nur ein vordefinierter Bereich visualisiert wird. Es soll aber eine weltweite Abdeckung möglich sein. Das zweite Problem ist, dass diese bestehenden Projekte nicht als Videospiel existieren, sondern eher als Karte, wie Google Maps. Das Ziel dieser Arbeit ist es, eine realistische 3D-Weltkarte zu erstellen, in der man sich als Spieler frei bewegen kann, ähnlich wie in einem Open-World-Spiel. Als Datenquelle für die Geodaten wird OSM verwendet. Für die Visualisierung der Geodaten wird die freie Game-Engine Godot und die integrierte Programmiersprache GDScript verwendet. Die Geodaten werden in einem genormten, speichereffizienten Format geliefert, in sogenannten Mapbox-Vektorkacheln. Diese werden mithilfe einer externen Software namens Planetiler aus den OSM-Daten erstellt. In diesem Projekt wird eine leicht abgeänderte Version des Planetilers verwendet. Godot liest die Daten aus diesen Vektorkacheln aus und decodiert die Geometrien zur Darstellung der Objekte. Für das Auslesen der Vektorkacheln wird eine Software des Industriepartners verwendet, welche im Godot Projekt als Erweiterung integriert wird. Folgende Objekte werden momentan dargestellt: Punktobjekte wie Bäume und Sitzbänke, Polygone wie Gebäude, Linienobjekte (Pfade) wie Strassen, Gehwege und Schienen sowie Flächenobjekte wie Bodenbedeckung, Flüsse und Seen, welche ebenfalls Polygone sind.

Die Qualität der dargestellten Objekte hängt von der Datenqualität in OSM ab. Beispielsweise sieht man in grossen bekannten Städten wie New York, dass viele Gebäude verschiedene Höhen haben. In weniger bekannten Gegenden haben die meisten Gebäude keine Höhe in OSM. Die Qualität hängt ebenfalls von den Vektorkacheln ab, die vom Planetiler produziert werden. Zu beachten ist, dass die vom Planetiler generierten Vektorkacheln nicht immer alle OSM-Daten enthalten. In dieser Arbeit hat man sich nicht mit der Implementation des Planetilers befasst. Als Spieler kann man die Software starten, und man landet an einem fix definierten Startpunkt. Der Spieler kann sich überall auf der Welt hinbewegen, oder den Startpunkt selbst anpassen. Dabei wird immer ein Bereich von 4x4 Vektorkacheln dargestellt. Bewegt sich der Spieler in eine bestimmte Richtung, werden automatisch die neuen Vektorkacheln heruntergeladen und visualisiert. Der alte Bereich, von dem sich der Spieler entfernt, wird gleichzeitig aus dem Speicher entfernt.

Der Nutzen dieser Arbeit besteht darin, eine Grundlage für die Verbesserung von OSM zu schaffen. In weiterführenden Arbeiten kann dieses Projekt so erweitert werden, dass der Benutzer auf spielerische Weise Kartendaten aus der realen Welt - zum Beispiel von Mapillary oder Google Street View - mit der 3D-Welt in Godot vergleichen und fehlende Objekte oder Daten ergänzen kann, beispielsweise die Höhendaten für Gebäude. Dies hilft letztlich, OSM mit Daten anzureichern. Je mehr Daten in OSM vorhanden sind, desto realistischer kann die 3D-Welt dargestellt werden. Das Hauptziel dieser Studienarbeit bestand darin, eine Software zu entwickeln, welche die Geodaten aus OSM visualisieren kann. Dieses Ziel wurde erreicht. Die Vektorkacheln beinhalten weitere Daten, mit denen man die Visualisierung realistischer gestalten könnte. Eine weitere Möglichkeit für die detaillierte Visualisierung der OSM-Daten wäre, mithilfe eines digitalen Geländemodells die Höhen der Landschaft darzustellen.

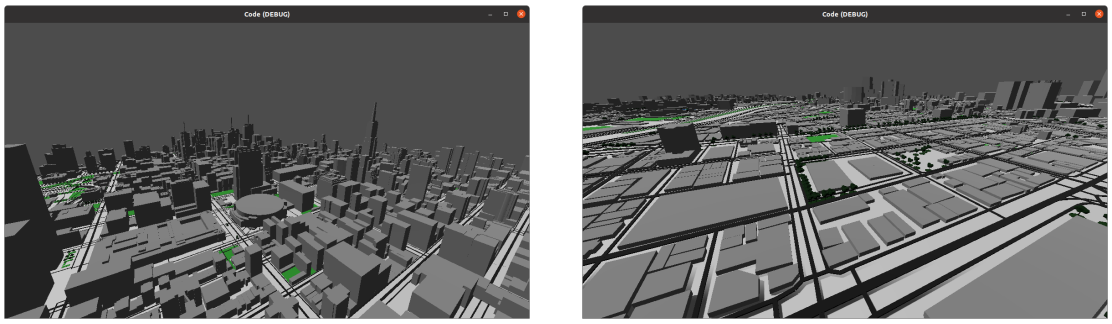
Teil II

Management Summary

Management Summary

Ziel dieser Arbeit

OpenStreetMap (OSM) ist ein Open-Source-Projekt, bei dem Nutzer verschiedene Arten von geografischen Daten erfassen können. Diese Geodaten dürfen von jedem frei verwendet werden (Open Data). OSM hilft dabei, diese Daten zu strukturieren. Das Problem ist, dass einige Orte ungenügende Daten haben, weil diese bis jetzt von niemandem erfasst wurden. In dieser Arbeit geht es darum, eine Grundlage für eine Software zu schaffen, welche Nutzern dabei hilft, OSM mit Daten zu befüllen. Das Hauptziel ist es, auf Basis von bereits vorhandenen OSM-Daten mithilfe einer Game-Engine eine 3D-Karte zu erstellen, in der sich der Benutzer frei bewegen kann, ähnlich wie in einem Open-World-Spiel. Diese Visualisierung von OSM-Daten kann mit anderen Anbietern von Geodaten, beispielsweise Mapillary oder Google Street View, verglichen werden, sodass der Benutzer direkt sieht, welche Daten in OSM nicht vorhanden sind. Ein Beispiel von Daten, welche sehr häufig nicht vorhanden sind, ist die Gebäudehöhe.



(a) Stadt mit vielen Gebäudehöhen

(b) Stadt mit wenig Gebäudehöhen

Abbildung 1: Gegenüberstellung zweier Städte mit vielen / wenigen Geodaten

In Abbildung 1a sieht man einen Ausschnitt aus Manhattan, in dem sich viele bekannte Gebäude auf kleiner Fläche befinden. Dem Bekanntheitsgrad entsprechend existieren dort für viele Gebäude die Höhendaten. In Abbildung 1b sieht man einen Ausschnitt aus einem Stadtteil in Tokio. In diesem Stadtteil gibt es sehr viele kleine Gebäude, bei denen keine Höhendaten erfasst sind. Ein Spieler, welchem die Gegend in 1b vertraut ist, kann diese OSM-Visualisierung nutzen, und mit einer anderen Quelle von Geodaten vergleichen, und die fehlenden Gebäudehöhen nachtragen. Es gibt auch noch viele andere Daten, wie beispielsweise Sitzbänke, Bäume, Gewässer etc. welche noch nicht in OSM erfasst sind.

Ergebnis

Die OSM-Daten sind in folgende Kategorien unterteilt: Polygone, Linestrings und Punkte. Um diese Daten zu visualisieren, braucht es Funktionen, welche die Vertices und Vektoren so berechnen, dass Godot damit die Geometrien bauen kann. Zusätzlich braucht es Funktionalität, welche die Metadaten der Geometrien berücksichtigt. Neben der Implementierung dieser Funktionalität braucht es auch eine Logik, welche das Nachladen von neuen Kacheln ermöglicht. Das ist notwendig, damit sich der Spieler ohne Grenzen in verschiedene Richtungen bewegen kann. Die Implementation des Prototypen - Darstellung von Gebäudeumrissen - ist gut gelungen. Auf dieser Basis konnte man relativ gut die anderen Geometrien

visualisieren. Es ist für den Spieler auch möglich, sich in alle Richtungen zu bewegen, ohne sich um das Nachladen neuer Vektorkacheln zu kümmern. Das passiert automatisch. Das Grundziel wurde erreicht. Man hat sich bei der Zielerreichung auf wichtige, wesentliche Punkte fokussiert. Die Vektorkacheln bieten viele weitere Metadaten - beispielsweise Informationen zu Dächern - mit denen sich eine realistischere 3D-Welt darstellen lässt. Aus zeitlichen Gründen wurde entschieden, auf die Implementation solcher Details zu verzichten.

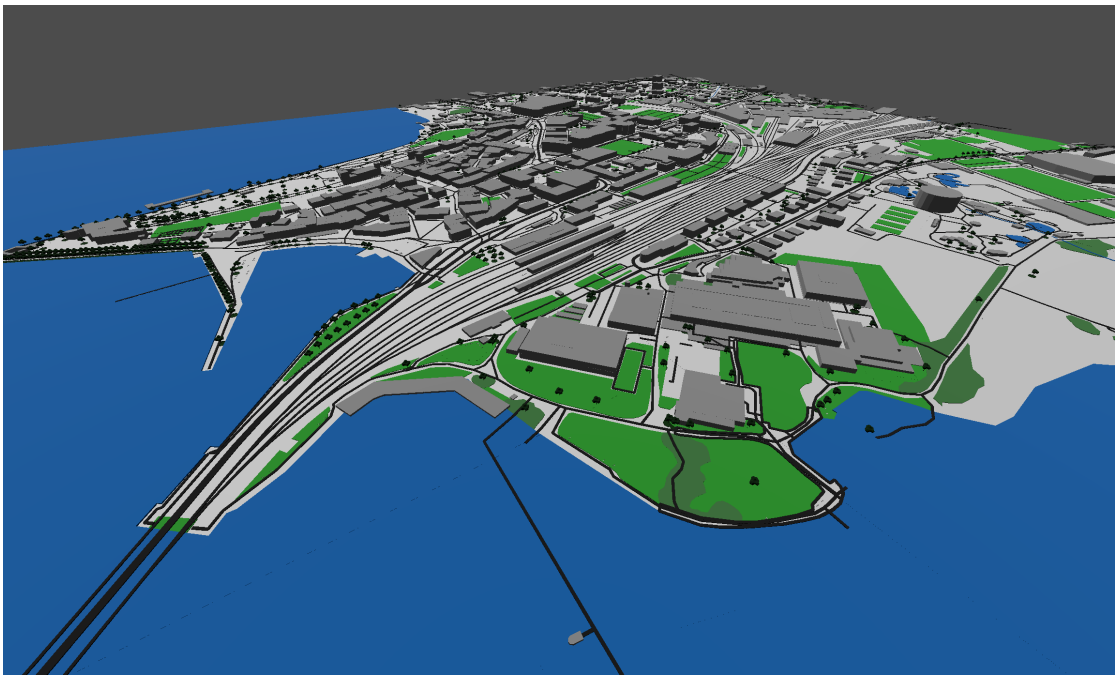


Abbildung 2: Rapperswil

Aufgetrene Probleme

Die grössten Probleme sind im Zusammenhang mit dem Datenset und der Performance aufgetreten. Zu Beginn des Projektes wurde das Datenset, welches vom Industriepartner, Sourcepole, zur Verfügung gestellt wurde, verwendet. Es stellte sich jedoch heraus, dass in diesem Datenset viele Tags, also Informationen zu den Details der darzustellenden Objekte, fehlten. Dadurch konnten zum Beispiel die Höhen von Gebäuden nicht berücksichtigt werden oder Bäume konnten nicht von Mülleimern unterschieden werden etc. Aufgrund dieser fehlenden Informationen wurde ein anderes Datenset - welches auch von StreetsGL eingesetzt wird - verwendet. Dort sind die Geodaten umfangreicher. Dieses Datenset zieht aber andere Nachteile mit sich. Beispielsweise werden in bestimmten Vektorkacheln alle Gewässer als ein Polygon - ein sogenanntes Multipolygon - gespeichert. Dieses Problem konnte behoben werden, indem das Multipolygon von einer Funktion in mehrere einzelne Polygone zerlegt wird, und diese jeweils einzeln dargestellt werden. Bei der Implementation der Visualisierung von Punkten (Points of Interest - Bäume, Mülleimer, Sitzbänke, Statuen etc.) wurde schnell ersichtlich, dass diese Objekte einen grossen Einfluss auf die Performance haben können. Zu Beginn wurde ein detailreiches Modell eines Baumes verwendet. Es stellte sich aber heraus, dass dieses Modell zu einem enormen Einbruch der Performance führte, besonders auf Geräten mit einer weniger starken Grafikkarte. Die Lösung des Problems war es, ein weniger detailreiches Modell zu verwenden. Mit einfacheren Modellen läuft die Software auch auf leistungsschwächeren Geräten flüssig. Ein weiteres

grosses Problem ist die Höhe, auf welcher sich bestimmte Objekte befinden. Spezifisch ist dies ein Problem, wenn fester Boden ein Gewässer kreuzt, zum Beispiel bei einer Brücke, oder wenn fester Boden in der Mitte eines Gewässers vorhanden ist, beispielsweise bei einer Insel. In diesen beiden Fällen sollte der feste Boden oberhalb des Wassers dargestellt werden. Wenn es sich bei dem Gewässer aber um einen Teich in einem Feld oder um einen Bach in einer Wiese handelt, sollte das Wasser oberhalb des Bodens dargestellt werden. Da keine generelle Annahme getroffen werden konnte, was oben oder unten dargestellt werden muss, konnte dieses Problem noch nicht behoben werden.

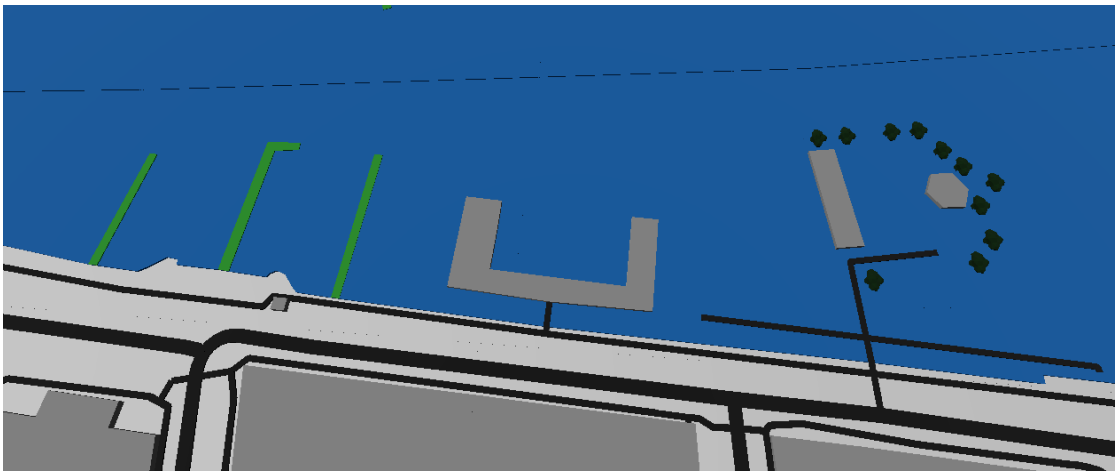


Abbildung 3: Boden wird nicht im Wasser dargestellt

Fortsetzung des Projekts

Das Projekt kann in viele Richtungen weiterentwickelt werden. Eine offensichtliche Erweiterung wäre die Implementation eines Systems zur Verbesserung der OSM-Daten, was das Schlussziel des Projektes ist. Vorher sollte aber der Detailgrad der Visualisierung ausgebaut werden. Momentan werden Gebäude als einfache Polygone dargestellt. Man könnte sie um Dächer und Fassaden (Texturen) erweitern, wodurch sie bereits deutlich realistischer aussehen würden. Ebenfalls werden noch bei weitem nicht alle Objekte dargestellt. Durch einen realistischeren Himmel mit entsprechender Beleuchtung würde die ganze Spielwelt lebendiger aussehen. Die Darstellung eines digitalen Geländemodells würde anschliessend das Gesamtbild der Spielwelt komplettieren. Ein weiterer grosser Punkt, an dem noch gearbeitet werden kann, ist das Nachladen von Vektorkacheln. Wenn der Spieler sich um eine gewisse Distanz bewegt, wird ein neuer Bereich geladen und ein altes Stück der Welt, welches nun zu weit weg ist, aus dem sichtbaren Bereich entfernt. Dieser Prozess blockiert in der aktuellen Implementation den Main-Thread für einige Sekunden. In einem weiteren Schritt könnte das Generieren der neuen Kacheln in einen Hintergrundprozess ausgelagert werden, damit sich der Spieler weiterhin bewegen kann, während die neuen Kacheln geladen werden. Ebenfalls nennenswert ist, dass das Projekt aktuell mit GDScript, der Godot internen Programmiersprache, entwickelt wurde. GDScript ist eine interpretierte Sprache, und daher weniger gut geeignet für rechenintensive Aufgaben. Dafür bietet Godot dem Programmierer die Möglichkeit, intensive Rechenaufgaben in C# oder C++ zu implementieren. Somit wäre das Visualisieren der Geometrien deutlich schneller.

Teil III

Produkt Dokumentation

Kapitel 1

Einführung

1.1 Industriepartner

Herr Pirmin Kalberer von der Sourcepole AG [1] ist der Industriepartner bei diesem Projekt. Sourcepole entwickelt kundenspezifische Lösungen im Bereich Geoinformatik auf der Basis von Open-Source-Software Komponenten. Die Sourcepole AG stellt die Geoinformationen für die ganze Schweiz zur Verfügung. Diese werden in Form von Vektorkacheln bereitgestellt. Dabei handelt es sich um ein effizientes Format für die Strukturierung von vielen Geoinformationen. Des Weiteren stellt die Sourcepole AG eine entsprechende Software zur Verfügung, welche Funktionalitäten für das Auslesen der Vektorkacheln bereitstellt.

1.2 Problembeschreibung

Die Visualisierung von Geoinformationen ist eine komplexe Herausforderung. Ziel ist es, Geodaten aus OpenStreetMap in 3D abzubilden. OpenStreetMap sammelt und strukturiert Geodaten, welche frei verwendet werden können. Dabei bietet OpenStreetMap auch die Möglichkeit, für bestimmte Objekte, insbesondere Gebäude, 3D-Daten zu hinterlegen. Mithilfe dieser Daten ist es möglich, eine Software zu bauen, welche diese Daten visualisieren kann. Das ist grundsätzlich die Idee dieses Projekts. Der Benutzer hat so die Möglichkeit, seine Umgebung wie in einem Open-World-Spiel auszukundschaften.

1.3 Vision

Die Idee dieser Arbeit besteht darin, eine Software zu entwickeln, welche OpenStreetMap-Daten in einer realistischen 3D-Umgebung darstellt. Dazu wird die freie Game-Engine Godot verwendet. Die Geodaten befinden sich in Vektorkacheln, und sollen automatisch ausgelesen und dargestellt werden können. Während der Laufzeit sollen die Vektorkacheln effizient visualisiert werden.

1.4 Ziele

Die Hauptziele dieses Projekts sind folgende:

- Darstellung von Gebäuden, Pfaden (Strassen, Gehwege und Gleise) und bestimmten Punkten
- Visualisierung der Umgebung durch dateibasiertes Laden der Vektorkacheln
- Benutzer kann sich frei in der visualisierten Umgebung bewegen
- Umgebung soll aufgrund der Spielerposition visualisiert werden

Kapitel 2

Stand der Technik

Im folgenden Kapitel wird beschrieben, welche Technologien und Standards für diese Arbeit in Betracht gezogen wurden, welche Vor- und Nachteile die einzelnen Technologien und Standards mit sich bringen und warum die schlussendliche Auswahl getroffen wurde.

2.1 OpenStreetMap

OpenStreetMap (OSM) [2] ist eine Open-Source-Software, welches Geodaten sammelt, die durch jeden Benutzer erfasst werden können. Gleichzeitig können diese Daten, welche von OSM aufbereitet und strukturiert werden, von jedem verwendet werden.

2.2 Game-Engine

Eine Game-Engine ist eine Software-Plattform, die verwendet wird, um Videospiele zu entwickeln. Sie stellt verschiedene Werkzeuge zur Verfügung, welche die Entwicklung erleichtern sollen. Die Werkzeuge helfen zum Beispiel dabei, die Spielwelt zu erstellen, Grafiken zu rendern oder die Physik zu simulieren.

2.2.1 Godot

Godot [3] ist eine Open-Source Game-Engine [4]. Sie wurde entwickelt, um die Entwicklung von 2D- und 3D-Spielen und interaktiven Anwendungen zu erleichtern. Die erste Version wurde 2007 von Juan Linietsky veröffentlicht. 2014 wurde dann die erste offizielle Open-Source-Version veröffentlicht. Stand Oktober 2023 ist Godot 4.2.1 die stabile Version. Aktuell wird Godot vor allem von Indie-Studios eingesetzt.

Vorteile

- Kenntnis beim Industriepartner: Der Industriepartner dieser Studienarbeit hat bereits Erfahrung mit Godot und kann bei Problemen behilflich sein.
- Open-Source: Godot ist eine Open-Source-Software und darf frei verwendet werden. Falls tatsächlich nötig, könnten Godot-Funktionalitäten auf die Anforderungen dieser Arbeit angepasst werden.
- GDScript: Godot hat eine eigene integrierte Skriptsprache, welche GDScript heisst [5]. GDScript wird in Godot üblicherweise verwendet, um die Spiellogik zu implementieren. Die Sprache ist darauf ausgelegt, möglichst effizient auf der Godot-Engine zu laufen und hat eine API, welche den Zugriff auf Godot-spezifische Funktionalitäten erlaubt. Es handelt sich um eine dynamisch typisierte Sprache mit einfacher Syntax, welche an die von Python angelehnt ist.

Nachteile

- Alter: Godot ist eine relativ junge Game-Engine, daher kommen die grossen Nachteile: Godot ist weniger bekannt als die Alternativen, daher ist die Gemeinschaft kleiner und die verfügbaren Hilfen im Internet sind beschränkter. Ebenfalls sind deutlich weniger Bibliotheken und Erweiterungen verfügbar.
- GDScript: Während GDScript Vorteile hat, gibt es auch einen Nachteil, welcher sich auf diese Arbeit auswirken könnte. Es handelt sich um eine interpretierte Skriptsprache, welche für Rechenintensive Aufgaben weniger geeignet ist als beispielsweise C# oder C++.

2.2.2 Unreal Engine

Unreal Engine [6] ist eine der grössten Game-Engines. Sie wurde von Epic Games entwickelt und wird von Epic Games gewartet. Unreal Engine hat sich einen hervorragenden Ruf erarbeitet und wird neben der Spieleindustrie auch in anderen Bereichen wie Architekturvisualisierung, Filmproduktion und Simulation eingesetzt. Einige bekannte Spiele, die mit Unreal Engine entwickelt wurden, sind zum Beispiel Fortnite, PlayerUnknown's Battlegrounds und die Call of Duty Serie.

Vorteile

- Leistung: Mit Unreal Engine können hoch performante Spiele mit ausgezeichneter Grafik entwickelt werden.
- Gemeinschaft: Durch den grossen Bekanntheitsgrad von Unreal Engine ist eine sehr grosse Gemeinschaft von Entwicklern vorhanden, die bei vielen Problemen helfen können und ausserdem ist eine grosse Anzahl von Bibliotheken und Erweiterungen verfügbar.

Nachteile

- Lernkurve: Aufgrund der hohen Komplexität hat Unreal Engine eine sehr steile Lernkurve, weshalb sie für diese Studienarbeit aufgrund der begrenzten Zeit nicht geeignet ist.
- Lizenz: Für Projekte, welche keinen Gewinn produzieren, ist die Verwendung von Unreal Engine kostenfrei und kann auch für Open-Source-Projekte verwendet werden. Für Projekte, die einen Gewinn erzielen, fallen Gebühren an.

2.2.3 Unity

Unity [7] ist neben Unreal Engine eine der am weitesten verbreiteten Game-Engines. Unity wurde von Unity Technologies entwickelt, 2005 erstmals veröffentlicht und seither regelmässig weiterentwickelt. Einige nennenswerte Spiele, die mit Unity entwickelt wurden sind zum Beispiel Pokémon GO, Hearthstone und Cities Skylines. Im September 2023 geriet Unity in Verruf, nachdem sie ihr Kostenmodell in eine Richtung geändert haben, die für Entwickler und End-User stark von Nachteil ist [8].

Vorteile

- Asset Store: Unity bietet einen sehr umfangreichen Asset Store mit tausenden frei verwendbaren Erweiterungen und Texturen.
- Benutzerfreundliche Oberfläche: Unity bietet eine intuitive, benutzerfreundliche Oberfläche.

Nachteile

- Leistungsoptimierung: Unity kann bei rechenintensiven Spielen Leistungsprobleme aufweisen, die aufwändig optimiert werden müssen.
- Dokumentation: Die umfangreiche Dokumentation ist teilweise unvollständig oder veraltet.

2.3 Mapbox Vektorkacheln

Mapbox Vectorkacheln [9] sind ein Datenformat, welches verwendet wird, um Raumdaten wie Punkte, Linien und Polygone zu speichern. Dadurch, dass sie nur die Geometrie speichern, und nicht die Bilder einer Karte, sind sie verhältnismässig klein.

2.3.1 Vektorkachel Schema

Es gibt verschiedene Schemas für Vektorkacheln. Für dieses Projekt wird das gleiche Schema verwendet, welches auch StreetsGL einsetzt, um eine 3D-Karte zu generieren. Dabei handelt es sich um eine Implementation der Mapbox Vector Tile Spezifikation [10]. Die Vektorkacheln werden mit einer modifizierten Version des Planetilers [11] generiert. Planetiler [12] ist ein Werkzeug, mit dem man Vektorkacheln aus geografischen Quellen wie OpenStreetMap generieren kann. Die generierten Vektorkacheln basieren auf OpenMapTiles [13], welches eine konkrete Implementation der Mapbox Vector Tile Spezifikation ist. Die Dateiendung einer Kachel ist .mvt, und der Inhalt ist im Protobuf Format serialisiert. Das Protobuf Format [14] (Protocol Buffers) wurde von Google entwickelt und wird verwendet, um strukturierte Daten zu serialisieren. Es ist sprachunabhängig und glänzt vor Allem dadurch, dass es sehr kompakt und schnell ist. Eine Kachel ist in sogenannten Layers unterteilt. Einige wichtige Layers sind:

- `buildings`
- `common`
- `highways`
- `water`
- `point`
- `natural`

Die Layers beinhalten Geodaten und weitere Metadaten. Die Geodaten eines Layers befinden sich in einem Feld, welches Features genannt wird. Features müssen folgende Felder beinhalten:

- geometry: Beinhaltet die Geometrien (Vertices)
- type: `UNKNOWN`, `POINT`, `LINESTRING`, `POLYGON`
- tags: Spezifische Metadaten, beispielsweise Gebäudehöhe
- id: Einzigartige Identifizierung (OpenStreetMap ID)

Die ersten beiden Felder, geometry und type, sind zwingend erforderlich, die anderen beiden sind optional. Die Geometrien verwenden eine Codierung, welche sich Zigzag [15] Codierung nennt, damit nur positive Ganzzahlen gespeichert werden. Die Idee dieser Codierung ist es, die Geodaten möglichst speichereffizient in den Vektorkacheln zu speichern, weil die Daten nach der Codierung eine variable Länge haben, und keine fixe mehr.

2.4 StreetsGL

Streets GL [16] ist ein Webbasierte 3D-Karte, welche Open Street Map Daten visualisieren kann. Das Projekt ist in Typescript geschrieben, dennoch können einige Punkte aus dem Quellcode für dieses Projekt als Vorlage dienen.

Kapitel 3

Resultate

3.1 Zielerreichung

Zu Beginn jedes Sprints werden Ziele festgelegt, welche vor Abschluss des Sprints abgeschlossen sein sollten. In diesem Kapitel werden diese Ziele erläutert. Dabei handelt es sich um produktspezifische Ziele, welche nicht die projektadministrativen Ziele abdecken.

3.1.1 Implementation eines Prototypen

Die Implementation eines einfachen Prototypen hatte den Zweck, ein Grundverständnis für die Funktionalität der Game-Engine und die Vektorkacheln zu entwickeln. Basierend auf diesem Prototypen können leichtere Entscheidungen bezüglich Implementation, Testing und Architektur getroffen werden. Man hat sich mit der Funktionalität der Tools in der ersten Woche schnell zurechtgefunden, sodass sich die Implementation des Prototypen als einfach erwies. Man hat sich dazu entschieden, einfache Polygone (Gebäude) einer Vektorkachel auszulesen, und diese mithilfe GDScript darzustellen. Dieses Ziel wurde schnell und effizient erreicht. Detaillierte Informationen zur Vorgehensweise werden im Unterkapitel 6.1.1 beschrieben.

3.1.2 Darstellung der Pfade

Man hat angenommen, dass die Darstellung der Strassen und Gehwege ähnlich ist wie die Darstellung der Gebäudeumrisse. Es erschien logisch, dass eine Strasse einen Startpunkt hat, und dann jeweils nur noch Punkte, die vom Startpunkt aus miteinander verbunden werden müssen. Bei der Analyse der Geometrien in den Vektorkacheln fiel auf, dass einige Strassen mehrere Startpunkte hatten. Nach genauer Analyse und einigen Vergleichen von Strassendaten mit OpenStreetMap stellte sich heraus, dass eine Strasse mehrere Startpunkte haben kann, weil sie an einem bestimmten Punkt unterbrochen wird, beispielsweise durch einen Kreisel, und an einem anderen Punkt weitergeht. Durch das benötigte man eine Woche länger, um die Visualisierung der Strassen zu implementieren. Das Ziel wurde erreicht, aber später, als geplant. Detaillierte Informationen zur Darstellung der Strassen finden sich im Unterkapitel 6.1.2.

3.1.3 Darstellung der Gebäudehöhe

Die Implementation der Gebäudehöhe erwies sich als relativ einfach. Die Höhe ist in den Tags der Vektorkacheln entweder in Anzahl Stockwerken, oder in Anzahl Metern angegeben. Zu Beginn des Projekts wurde die Darstellung der Höhe bereits implementiert. Aufgrund des Problems mit dem unvollständigen Datenset, welches in 8.1.1 beschrieben wird, musste die entsprechende Funktion angepasst werden, weil im alten Datenset die Höhen alle in einer Einheit angegeben waren. Für die Erreichung dieses Ziels musste beim neuen Datenset jeweils geprüft werden, in welcher Eineheit (Stockwerke oder Meter) die Höhe codiert war, und diese in eine Einheit umrechnen. Man hat sich für Meter entschieden, wobei ein Stockwerk 2.75 Meter hoch ist.

3.1.4 Darstellung von Punkten

Die Implementation der Punkte konnte mit wenig Aufwand realisiert werden. Es gibt einen Layer, welcher `POINT` heisst. Darin sind spezielle Punkte erfasst, welche nur eine Koordinate haben. Das können beispielsweise Bäume, Sitzbänke oder Mülleimer sein. Man hat sich hier dazu entschieden, zuerst die Bäume in der 3D-Welt abzubilden. Grund dafür ist, dass man bei Bäumen nicht die Himmelsrichtung beachten muss, wie beispielsweise bei Sitzbänken. Ausserdem gibt es im Internet viele frei verfügbare 3D-Modelle von Bäumen. Detaillierte Details zur Implementation der Punkte finden sich im Unkterkapitel 6.1.4. Des Weiteren lassen sich Punkte nicht vernünftig auf Geräten darstellen, welche eine schwache Grafikkarte haben. Details zu diesem Problem werden im Unterkapitel 8.1.5

3.1.5 Darstellung der Gewässer

Hauptgewässer sind Flüsse und Seen. Diese konnten zu Beginn des Projekts relativ einfach impementiert werden, da es sich wie bei den Gebäudeumrissen um Polygone handelt. Im weiteren Projektverlauf musste diese Implementation wieder Rückgängig gemacht werden, weil mit einem neuen Datenset gearbeitet wurde, siehe 8.1.1. Beim neuen Datenset befinden sich sowohl Flüsse als auch Seen im gleichen Layer `WATER`. Nachdem die Visualisierung der Gewässer mit dem neuen Datenset re-implementiert wurde, sah es so aus, als ob diese korrekt dargestellt wurden. Im weiteren Projektverlauf fiel auf, dass bestimmte Gewässer und Flussteile nicht visualisiert wurden. Der Grund dafür war, das Flüsse als sogenannte Multipolygone codiert waren. Auch bestimmte Zusammenhängende Gewässer wie Teiche oder Abwasserreinigungsanlagen waren als solche Multipolygone codiert. Detaillierte Informationen zu diesem Problem finden sich im Unterkapitel 8.1.3. Um dieses Ziel zu erreichen, mussten im späteren Projektverlauf die Funktionen für die Berechnung und Darstellung von Polygonen so angepasst werden, dass auch Multipolygone als Input akzeptiert und korrekt ausgegeben wurden. Dieses Ziel wurde also erreicht, aber später als geplant.

3.1.6 Darstellung der Breite von Pfaden

Die Breite der Pfade ist nicht in den Vektorkacheln angegeben. Hier handelt es sich lediglich um Linestrings. Um dieses Ziel zu erreichen, wurden für verschiedene Pfade (Autobahn, Gehweg, Strasse, Gleis etc.) fixe Breiten definiert. Diese werden beim Visualisieren der Pfade an die entsprechende Linestring-Funktion mitgegeben. Dieses Ziel konnte planmässig erreicht werden.

3.1.7 Implementation einfacher Texturen

Ein Ziel, welches nicht zu den Hauptzielen dieser Arbeit gehörte, war die Implementation von Texturen. Darauf wurde aus zeitlichen Gründen verzichtet. Ausserdem wurde dieses Ziel niedrig priorisiert. Zu Beginn von Sprint 6 wurde festgelegt, dass man Texturen nur implementiert, wenn dafür genug Zeit bleibt. Dieses Ziel wurde somit nicht erreicht.

3.1.8 Nachladen mehrerer Vektorkacheln

Das automatische Nachladen der Vektorkacheln erwies sich als grosse Herausforderung. Die Idee war, dass sich der Spieler frei bewegt, und irgendwann einen Schwellenwert überschreitet, und damit das Nachladen einer neuen Kachelreihe auslöst. Um die Implementation zu vereinfachen, hat man hier zunächst versucht einige Vektorkacheln gleichzeitig zu visualisieren, ohne das weitere Vektorkacheln

automatisch nachgeladen werden. Das wurde erreicht, indem man Versuchsweise die Länge / Breite einer Vektorkachel ermittelte. Die Funktionen für die Berechnung der Vektoren konnten problemlos erweitert werden, sodass Sie die Offset-Werte als Argumente entgegen nehmen. In einem weiteren Schritt konnte man mithilfe der `_process(delta)` Funktion, welche von Godot zur Verfügung gestellt wird, und pro Frame aufgerufen wird, die Berechnungslogik für das automatische Nachladen neuer Vektorkacheln implementieren. Die Implementation war etwas aufwändig und Zeitintensiv, aber das Ziel konnte planmässig erreicht werden. Detaillierte Informationen finden sich in Kapitel 6.1.6

Teil IV

Projekt Dokumentation

Kapitel 4

Anforderungen

4.1 Funktionale Anforderungen

ID	FR-1
Title	Darstellung von Gebäuden und Strassen
Beschreibung	Minimal: Gebäude und Strassen aus Vector Tiles visualisieren
Priorität	Hoch

Tabelle 4.1: FR-1

ID	FR-2
Title	Dateibasiertes Laden der Daten
Beschreibung	Vector Tiles sollen vom lokalen Dateisystem gelesen werden
Priorität	Hoch

Tabelle 4.2: FR-2

ID	FR-3
Title	Evaluation von Terrain-Rendering
Beschreibung	Minimal: Verständnis für die Problematik erarbeiten
Priorität	Mittel

Tabelle 4.3: FR-3

ID	FR-4
Title	Darstellung von Bodenbedeckung
Beschreibung	Seen, Flüsse, Wiesen, Wälder und Bäume sollen aus den Daten aus den Vector Tiles dargestellt werden
Priorität	Mittel

Tabelle 4.4: FR-4

ID	FR-5
Title	Schöne Darstellung
Beschreibung	Schöne Darstellung von Himmel, Sonne, Reflektionen und Schatten
Priorität	Niedrig

Tabelle 4.5: FR-5

ID	FR-6
Title	Darstellung von Labels
Beschreibung	Labels wie Strassennamen, Adressen oder Ortsnamen sollen dargestellt werden
Priorität	Mittel

Tabelle 4.6: FR-6

ID	FR-7
Title	Kombination von 3D Darstellung und Terrain-Rendering
Beschreibung	Die 3D Darstellung der NFRs 1, 4 und 6 sollen auf dem gerenderten Terrain dargestellt werden
Priorität	Niedrig

Tabelle 4.7: FR-7

4.2 Nicht funktionale Anforderungen

ID	NFR-1
Title	Performance
Beschreibung	Die Applikation soll im Normalen zustand mit mindestens 30 FPS laufen (Normalzustand ist nicht während dem Laden von weiteren Tiles)
Überprüfbarkeit	Tests mit FPS Counter
Priorität	Hoch

Tabelle 4.8: NFR-1

ID	NFR-2
Title	Usability
Beschreibung	Benutzer soll keine Dokumentation lesen müssen, um sich in der Spielwelt zurechtzufinden
Überprüfbarkeit	Mindestens 5 Benutzer die Applikation verwenden lassen, 4 davon sollen sagen, dass sie einfach zu verwenden ist
Priorität	Mittel

Tabelle 4.9: NFR-2

4.3 Stretch Goals

ID	SG-1
Title	VR Anwendung
Beschreibung	Ansicht und Navigation in der Spielwelt mit VR Brille
Priorität	-

Tabelle 4.10: SG-1

ID	SG-2
Title	Integration von Mapillary Bildern
Beschreibung	Die Mapillary Bilder der aktuellen Position sollen dem Nutzer dargestellt werden
Priorität	-

Tabelle 4.11: SG-2

ID	SG-3
Title	Integration von Street-Complete o.ä.
Beschreibung	In Kombination mit SG-2 soll die Funktion von Street-Complete oder etwas ähnlichem erstellt werden
Priorität	-

Tabelle 4.12: SG-3

4.4 Status der Anforderungen

4.4.1 Funktionale Anforderungen

ID	Status	Anmerkungen
FR-1	i.O.	
FR-2	i.O.	Daten werden via HTTP in das Filesystem geladen
FR-3	i.O.	Nur dokumentiert
FR-4	i.O.	
FR-5	n.i.O.	Grundlegende Beleuchtung
FR-6	n.i.O.	Labels werden nicht dargestellt
FR-7	n.i.O.	Möglicher Implementationsansatz wurde dokumentiert

Tabelle 4.13: Status der FR

4.4.2 Nicht funktionale Anforderungen

ID	Status	Anmerkungen
NFR-1	i.O.	Mit Godot internem counter auf 3 Maschinen getestet
NFR-2	n.A.	Da kein SSpielërstellt wurde, nicht testbar

Tabelle 4.14: Status der NFR

4.4.3 Status der Stretch Goals

Die Stretch Goals wurden nicht bearbeitet.

ID	Status	Anmerkungen
SG-1	n.i.O.	
SG-2	n.i.O.	
SG-3	n.i.O.	

Tabelle 4.15: Status der SG

Kapitel 5

Architecture

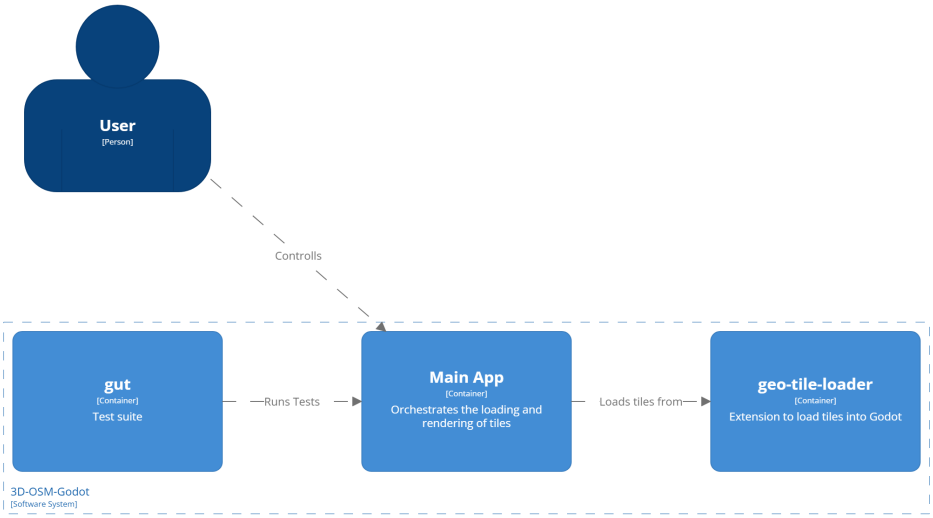
Dieses Kapitel Zeigt die Architektur des Projektes anhand des C4 Models.

5.1 Context

[System Context] 3D-OSM-Godot
Dienstag, 14. November 2023 um 11:03 Mitteleuropäische Normalzeit

Abbildung 5.1: C4 Model Context Diagramm

5.2 Container



[Container] 3D-OSM-Godot
Dienstag, 14. November 2023 um 11:04 Mitteleuropäische Normalzeit

Abbildung 5.2: C4 Model Container Diagramm

5.3 Components

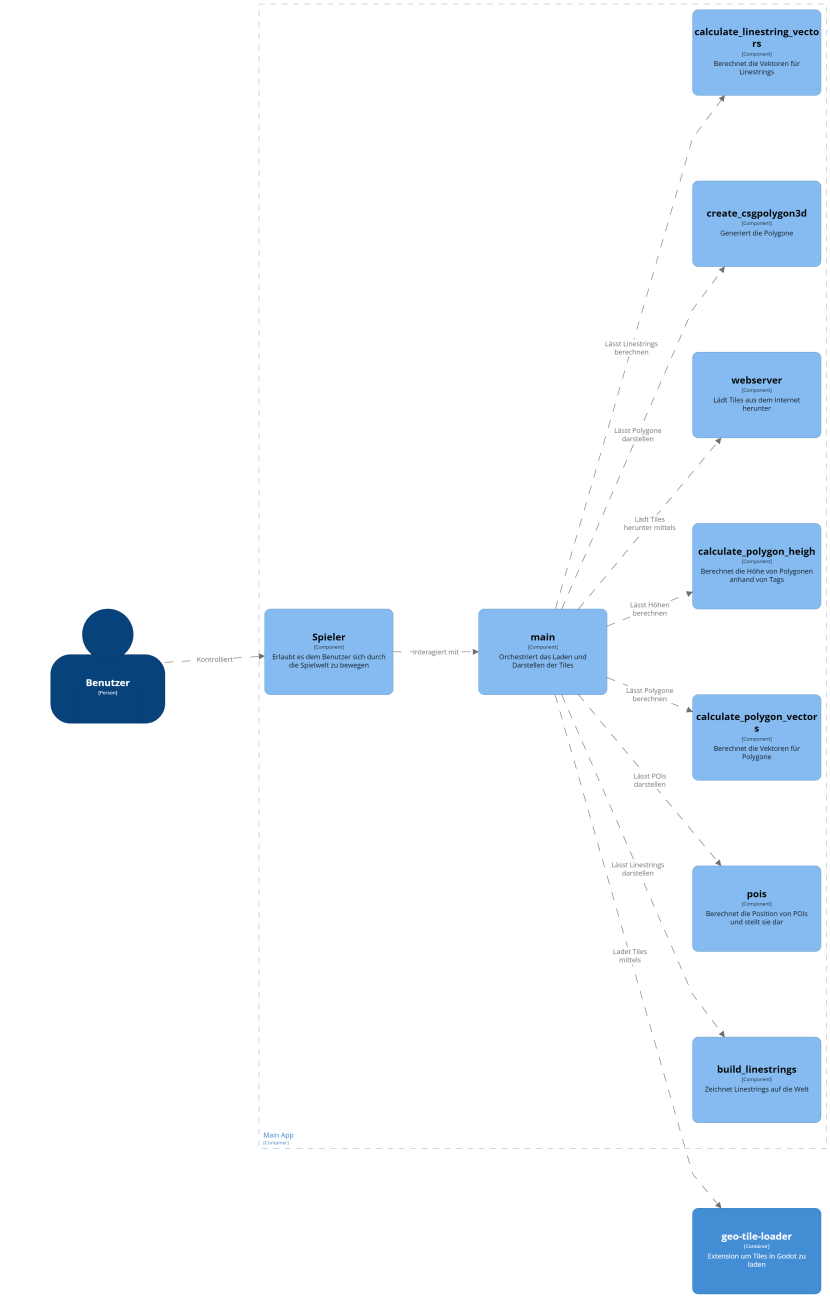


Abbildung 5.3: C4 Model Component Diagramm

Kapitel 6

Implementation

6.1 Implementation

In diesem Kapitel wird die Implementation der Funktionalität beschrieben.

6.1.1 Darstellung der Gebäudeumrisse

Die Gebäude haben einen eigenen Layer in den Vektorkacheln. Die Layer haben wiederum sogenannte Features, welche unter Anderem ein Feld mit dem Namen "geometry"beinhalten. In diesem geometry-Feld befinden sich die Punkte für alle Gebäude. Dabei hat jedes Gebäude einen Startpunkt, und die restlichen Punkte werden jeweils zum Startpunkt addiert. Es gibt auch selten Gebäude, welche Mutlipolygone sind, und somit mehrere Startpunkte haben. Die Darstellung solcher Gebäude ist sehr schwer. Genaue Informationen können dem Kapitel 8.1.4 entnommen werden.

```
name: "buildings";
features: [
  0: {
    id: 55086548;
    tags: [
    type: ENUM::3;
    geometry: [
      0: 9,          point at
      1: 98832,      x0
      2: 36432,      y0
      3: 26,         draw line to
      4: 2008,       x1
      5: 4359,       y1
      6: 2520,       x2
      7: 1160,       y2
      8: 2007,       x3
      9: 4352,       y3
      10: 15         close polygon
    ];
  }
];
```

Abbildung 6.1: Codierte Geometrien in Vektorkachel

Die Abbildung 6.1 zeigt die codierten Geometrien. Die Punkte x0 und y0 bilden den Startpunkt und beschreiben die Entfernung vom Nullpunkt der aktuellen Vektorkachel aus. Somit bilden x0 und y0 den ersten Vertex P1 des Gebäudes. Der zweite Vertex ergibt sich aus der Addition von (x0 / y0) und (x1 / y1) und bildet damit den Vertex P2. Der dritte Vertex P3 ergibt sich aus der Summe von P2 und (x2 / y2). Die Punkte 9, 26 und 15 sind keine Koordinaten, sondern Instruktionen. Diese werden bei der Datenaufbereitung herausgefiltert. Die Aufbereitung der Daten besteht aus dem Decodieren der Rohdaten, und dem Aufsummieren der Punkte. Zu Beginn des Projekts wurde die Decodierung der Daten direkt in GDScript implementiert. Im Verlauf des Projekts wurde ein neuer Reader für die Vektorkacheln bereitgestellt, welches sich um das decodieren kümmert. Godot erhält am Ende der Datenaufbereitung (von einer einzigen Vektorkachel) ein Array, welches wiederum Vektor Arrays von allen Gebäuden der aktuellen Kachel enthält.

6.1.2 Darstellung von Strassen und Schienen

Im Layer `TRANSPORTATION` finden sich die Strassen, Gehwege und Schienen, die in den Kacheln eingetragen sind, als Features. Die Geometrien dieser Features können gleich erreicht werden wie die der Gebäude, allerdings unterscheidet sich die Codierung.

Abbildung 6.2: Beispiel einer unterbrochenen Strasse

Während ein Gebäude ein Polygon mit einem Startpunkt und beliebig vielen Eckpunkten ist, ist eine Strasse ein Linestring mit mindestens einem Startpunkt und mindestens einem Endpunkt. Da eine Strasse aber unterteilt sein kann, zum Beispiel wenn sie für eine kurze Strecke einer anderen Strasse folgt, wie zum Beispiel die Bachtobelstrasse in Zürich in Abbildung 6.2, kann es sein, dass eine Strasse mehr als einen Startpunkt hat.

```
name: "transportation";
features: [
  0: {
    id: 191696077;
    tags: [
    type: ENUM::2;
    geometry: [
      0: 9,          jump to
      1: 55960,      x0
      2: 127224,     y0
      3: 10,         line to
      4: 10599,      x1
      5: 2688,       y1
      6: 911,        x2
      7: 256,        y2
      8: 9,          jump to
      9: 47920,      x3
      10: 43527,     y3
      11: 10,        line to
      12: 463,       x4
      13: 447,       y4
    ];
  },
```

Abbildung 6.3: Codierung von Linestrings

Die Abbildung 6.3 zeigt ein Beispiel der Codierung der Geometrie von Linestrings. Gleich wie bei Polygonen wird zuerst zum Startpunkt (x0 / y0) navigiert. Von diesem Punkt aus wird eine Linie zu beliebig vielen Punkten gezogen, in diesem Beispiel zuerst zum Punkt (x1 / y1) und dann zum Punkt (x2 / y2). Da eine Strasse aber aufgeteilt sein kann, kann es nun sein, dass ein Sprung zu einem Punkt, hier (x3 / y3) nötig ist, für den aber keine Linie gezeichnet werden soll. Es ist zu beachten, dass die Koordinaten der Punkte, wie auch bei den Polygonen, die Distanz

zum letzten Punkt ist, bzw. für den Startpunkt relativ zum Nullpunkt der Kachel.

6.1.3 Darstellung von Gewässern

Der Layer `WATER` beinhaltet Flüsse und Seen. Flüsse sind gleichzeitig Linestrings und Polygone, Seen hingegen sind nur Polygone. In beiden Fällen werden sie gleich Codiert wie Gebäude und Pfade und entsprechend gleich Dargestellt. In der Datenstruktur können sie über die Funktion `geomType()` unterschieden werden.

6.1.4 Darstellung von Points of Interest

Der Layer `POINTS` beinhaltet die Points of Interest. Das sind zum Beispiel einzelne grosse Bäume, Sitzbänke, Mülleimer oder Statuen. In der Datenstruktur haben diese POIs genau eine Koordinate, welche relativ zum Nullpunkt ist und angibt, wo in der Kachel das Objekt angezeigt werden soll. POIs werden in diesem Projekt mit einer sogenannten Multimesh Instanz [17] dargestellt. Diese Instanzen sind speziell darauf ausgelegt, das gleiche Objekt, wie zum Beispiel einen Baum, hunderte oder sogar tausende mal mit möglichst kleinem Einfluss auf die Performance darzustellen, indem sie die Grafikkarte optimal nutzen. Dies führt dazu, dass eine Grafikkarte nötig ist, um das Projekt sinnvoll zu nutzen. Der Multimesh Instanz wird eine Liste von Punkten und ein 3D Modell übergeben, die Instanz stellt dann an jedem dieser Punkte das Modell dar.

6.1.5 Darstellung des Bodens

Damit die Strukturen nicht einfach in der Luft schweben, wurde jeder Kachel ein Boden hinzugefügt. Dieser boden ist nichts weiteres als ein Quadratisches Polygon welches als Seitenlänge die Seitenlänge einer ganzen Kachel hat.

6.1.6 Nachladen mehrerer Kacheln

Ein sehr wichtiger Punkt der Arbeit war, dass nicht nur ein ganz bestimmter, definierter Bereich vorgerendert wird, sondern eine globale Abdeckung gewährleistet werden kann und dass der Benutzer sich im laufenden Programm durch die Welt bewegen kann. Es wäre unrealistisch zu erwarten, dass die ganze Welt gerendert werden kann. Deshalb musste ein Weg gefunden werden, um einen verschiebbaren Bereich um den Spieler herum zu visualisieren. Es wurde experimentell herausgefunden, dass eine Vektorkachel eine Fläche mit einer Seitenlänge von ca. 655 Metern abdeckt. Anhand der aktuellen Koordinaten des Spielers kann herausgefunden werden, wie weit entfernt er sich vom Rand befindet. Dementsprechend kann ermittelt werden, ob ein neuer Bereich geladen werden muss. Ist dies der Fall, werden vier HTTPRequest Nodes erzeugt. Diese Nodes laden die gewünschte Vektorkachel von Tileserver herunter und speichern sie lokal. Ist der Download abgeschlossen, wird eine Funktion aufgerufen, die eine neue Node für die Vektorkachel erzeugt und anschliessend Schritt für Schritt die einzelnen Objekte generiert und diese der Node als Child-Nodes übergibt. Dass für jede Vektorkachel eine eigene Node erzeugt wird, ist deshalb wichtig, weil so alte Vektorkacheln, welche sich nicht mehr in dem Bereich befinden, der visualisiert werden soll, entfernt werden können. Entfernte Nodes werden durch den Garbage-Collector von Godot aus dem Arbeitsspeicher gelöscht.

Kapitel 7

Qualität

In diesem Kapitel wird beschrieben, welche Schritte unternommen wurden, um die Qualität des Projekts sicherzustellen.

7.1 Unit Tests

Unit Tests werden verwendet, um die Funktionalität von individuellen Komponenten sicherzustellen. Das sind vor Allem Funktionen, welche Berechnungen durchführen. Diese Funktionen können sehr einfach getestet werden, weil bei der Berechnung von Vektoren immer eine bestimmte Ausgabe erwartet werden kann.

7.1.1 Umfang

Es existieren drei Arten von Objekten:

- Polygone - Einzelpolygone und Multipolygone
- Linestrings - Einzelne Linestrings und Multilinestrings
- Punkte - Einzelne Koordinatenpunkte

Die Unit Tests decken alle Berechnungsfunktionen für diese Objekte ab. Zusätzlich gibt es einige Tests für die Erweiterung, welche verwendet wird, um Vektorkacheln auszulesen. So sieht man als Programmierer gleich, ob der Fehler bei der Erweiterung für das Auslesen der Vektorkacheln liegt, oder woanders.

7.1.2 GUT - Godot Unit Testing

Godot selbst bietet keine Möglichkeit, wie man die Funktionalität der eigenen Software testen kann. Weil aber das Bedürfnis dafür vorhanden ist, existieren einige externe Bibliotheken, welche Unit Tests unterstützen. Dabei hat sich GUT [18] - Godot Unit Testing - als de facto Standard durchgesetzt. Diese Bibliothek wird in diesem Projekt verwendet.

7.2 Linter

Der Code des Projektes wurde mithilfe des Linters `gdlint` aus dem GDScript Toolkit [19] formatiert. Der Linter überprüft jede Datei, auf die er angewendet wird zweimal. Im ersten Durchgang überprüft er nur den GDScript Syntax, im zweiten Schritt überprüft er verschiedene vordefinierte Regeln. Es wurden die Standardregeln des Linters verwendet.

7.3 Formatter

Das GDScript Toolkit bietet ebenfalls einen Formatter. Dieser wurde wie der Linter auf jedes Code File im Projekt angewendet. Dadurch konnten bereits die meisten Linter Probleme behoben werden, da oft das Problem nur war, dass eine Zeile zu lange war. Auch für den Formatter wurden die Standardeinstellungen verwendet.

7.4 Cyclomatic Complexity

Ebenfalls aus dem GDToolkit wir der Cyclomatic Complexity Calculator verwendet. Die Cyclomatic Complexity ist eine Messung, wie komplex eine Funktion ist [20]. Ein tiefer Wert ist dabei erwünschenswert. Wie in der Abbildung 7.2 zu sehen ist, hat die schlimmste Funktion eine Bewertung von C mit 17 Punkten. Dies bedeutet, dass die gesamte Applikation höchstens moderat komplex ist.

CC score	Rank	Risk
1 - 5	A	low - simple block
6 - 10	B	low - well structured and stable block
11 - 20	C	moderate - slightly complex block
21 - 30	D	more than moderate - more complex block
31 - 40	E	high - complex block, alarming
41+	F	very high - error-prone, unstable block

Abbildung 7.1: Bewertung von Cyclomatic Complexity [21]

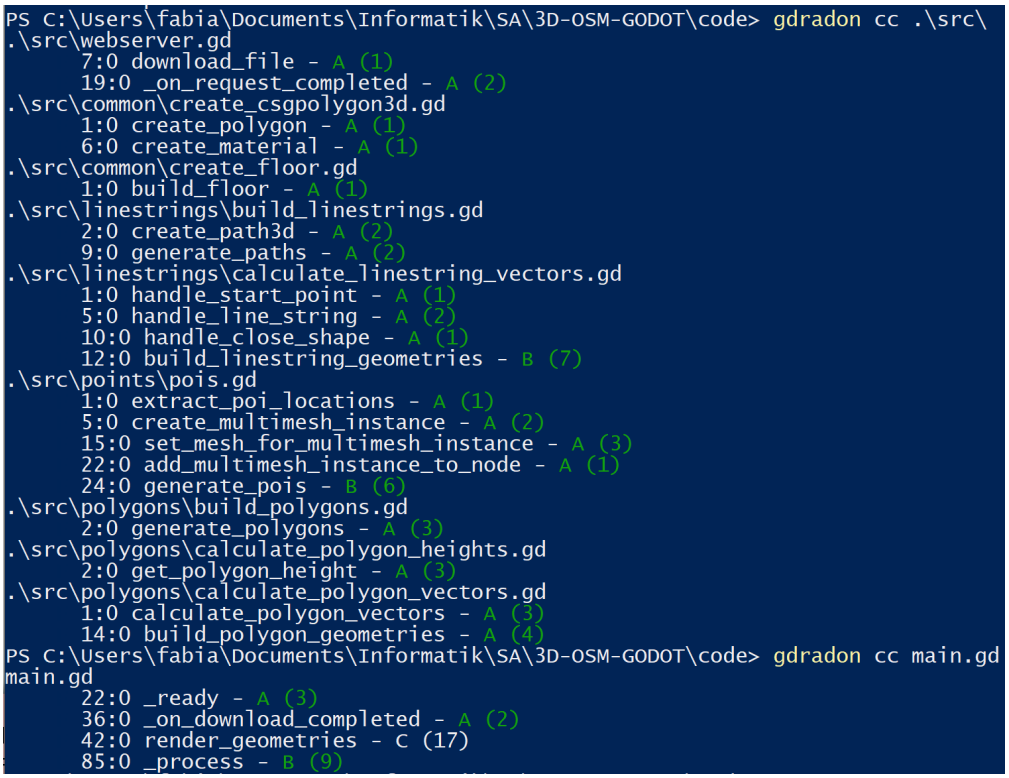


Abbildung 7.2: Bewertung von Cyclomatic Complexity des Codes

Kapitel 8

Probleme und Ausblick

8.1 Aufgetretene Probleme

Hier werden Probleme beschrieben, welche den Verlauf der Umsetzung dieses Projekts erheblich beeinflusst haben.

8.1.1 Fehlerhaftes Datenset

Zu Beginn des Projekts arbeitete man mit den Vektorkacheln, welche vom Industriepartner zur Verfügung gestellt wurden [22]. Während Sprint 3, bei dem es darum ging, Gebäudehöhe, Bäume und Gewässer darzustellen, ist aufgefallen, dass die Tags in den Layern nicht den Tags entsprechen, welche auf OpenStreetMap angezeigt wurden. Eine kurze Überprüfung durch den Industriepartner ergab, dass das entsprechende Datenset unvollständig war. Um das Problem zu lösen, stieg man auf ein anderes Datenset um. Dabei entschied man sich die gleichen Vektorkacheln zu verwenden, welche auch StreetsGL verwendet. Diese können direkt unter `https://tiles.streets.gl/vector/16/x/y` abgerufen werden. Die Parameter x und y stellen die Nummer der Vektorkachel dar. Diese Anpassung erforderte allerdings, dass die Kacheln zuerst über GDScript heruntergeladen und lokal abgelegt werden mussten. Zu diesem Zweck wurde ein entsprechendes Script aufgesetzt. Der Industriepartner versicherte, diese Vektorkacheln auf einem einem eigenen Server zur Verfügung zu stellen, sodass man nicht von einer externen Quelle abhängig ist, über die man selbst keine Kontrolle hat.

8.1.2 Strassen unvollständig visualisiert

Während Sprint 3 ist aufgefallen, dass die Strassen zwar korrekt, aber nicht ganz vollständig visualisiert wurden. Die Implementation für die Berechnung der Strassen war richtig, aber die Anwendung der von GDScript zur Verfügung gestellten range-Funktion war falsch:

```
range(b: int, n: int, s: int)
```

Die Funktion beginnt bei b, inkrementiert um s Schritte und endet vor n. Der Parameter b ist inklusiv während Parameter n exklusiv ist. Dieses Detail wurde bei der Implementation ausser Acht gelassen. In der Annahme, dass n inklusiv ist, wurde

```
for i in range(0, feature_geometry.size() - 1, 1):
```

verwendet, anstatt

```
for i in range(0, feature_geometry.size(), 1):
```

Das führte dazu, dass das letzte Element nicht beachtet wurde und somit die Strassen unvollständig visualisiert wurden.

8.1.3 Flüsse werden nicht vollständig dargestellt

Während der Entwicklung wurde übersehen, dass es neben Polygonen - ein Startpunkt gefolgt von beliebig vielen Vertices - auch sogenannte Multipolygone gibt.

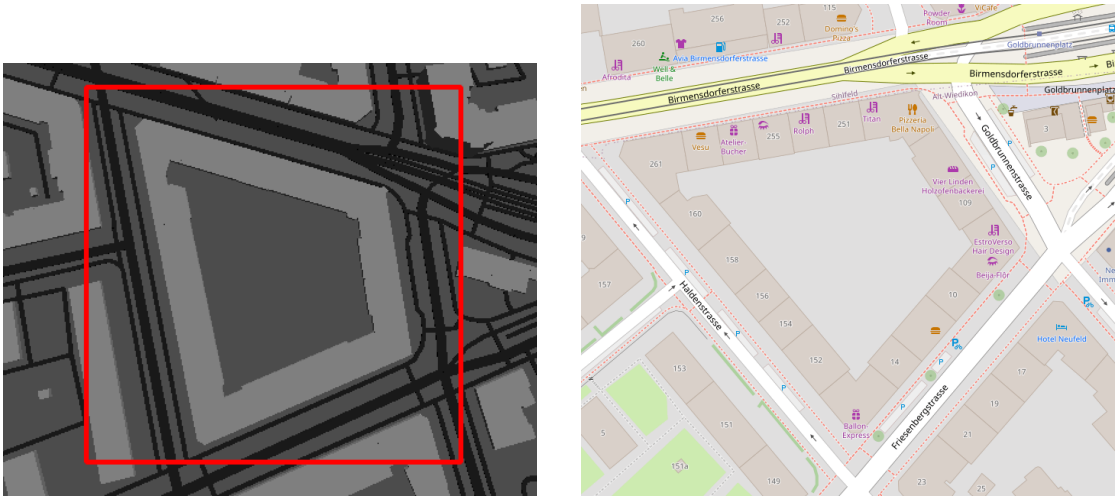
Der Grund, weshalb das übersehen wurde ist, dass die meisten Polygone Gebäude sind, und nur einen Startpunkt haben. Es gibt aber in seltenen Fällen Polygone, welche aus mehreren Polygonen bestehen. Beispielsweise Gebäude mit einem Innenhof, oder Zusammenhängende gewässer etc. Flüsse sind in den Vektorkacheln als Multipolygone codiert. Da die Funktion zur Aufbereitung von Polygonen nicht auf Multipolygone ausgelegt war, wurde nur das erste Unterpolygon dargestellt.

Abbildung 8.1: Unvollständige Darstellung von Flüssen

In Abbildung 8.1 sieht man beim Fluss, dass nur der erste Teil angezeigt wird. Beim Beginn der neuen Kachel (rechter rand des roten Rechtecks) werden alle Elemente neu dargestellt. Hier sieht man, wie wieder nur der erste Teil des Multipolygons dargestellt wird. Man beachte auch, dass dieses Problem nicht nur auf Flüsse zutrifft. Die drei markierten Kreise, welche zur ARA Werdhölzli gehören, sind als zusammenhängendes Gewässer codiert, und nicht als einzelne. Daher wird nur der erste Kreis als Gewässer visualisiert. Das Problem konnte behoben werden, indem man bei den jeweiligen Funktionen, welche die Polygone berechnen, einen for-Loop eingebaut hat, damit Multipolygone in einzelne Polygone aufgeteilt werden.

8.1.4 Gebäude als Multipolygone werden nicht korrekt visualisiert

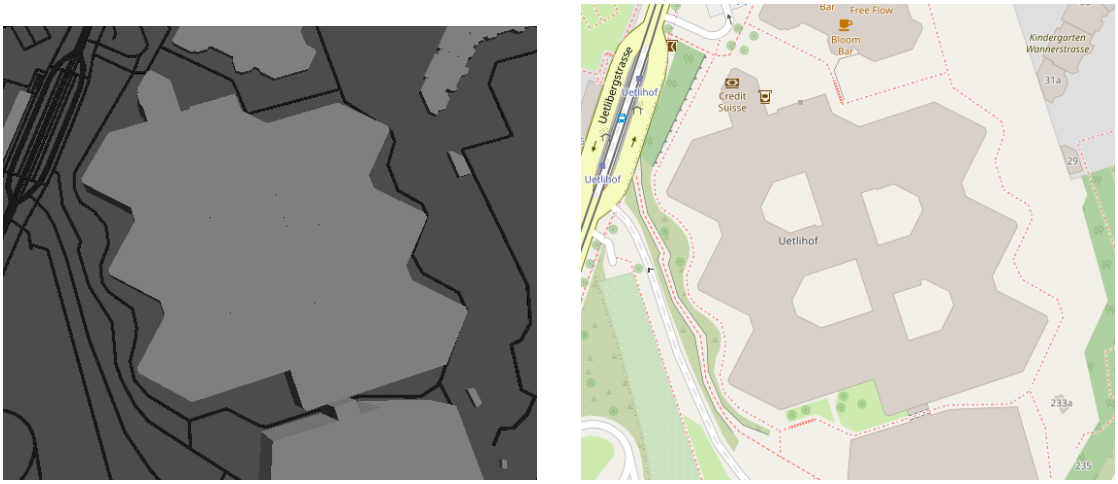
Es gibt viele Gebäude mit einem Innenhof. Auf den ersten Blick werden diese korrekt dargestellt. Das funktioniert aber nicht bei Gebäuden, welche als Multipolygon codiert sind.



(a) Gebäude mit Innenhof (b) Gebäude mit Innenhof auf OSM

Abbildung 8.2: Gegenüberstellung Gebäude mit Innenhof Godot / OpenStreetMap

In der Abbildung 8.2a sieht man ein Gebäude mit einem Innenhof. Wie man In Abbildung 8.2b erkennen kann, handelt es sich hier um ein Gebäude, welches aus mehreren einzelnen Polygonen zusammengesetzt ist. Es ist also nicht ein einzelnes Gebäude, welches als Multipolygon codiert ist. Deshalb wird der Innenhof auch korrekt als Hohlraum visualisiert. Bei einem einzelnen Gebäude, welches einen Innenhof hat, funktioniert das nicht.



(a) Multipolygon mit Innenhof Godot (b) Multipolygon mit Innenhof OSM

Abbildung 8.3: Gegenüberstellung Multipolygon mit Innenhof Godot / OpenStreetMap

In der Abbildung 8.3 sieht man das Credit Suisse Gebäude an der Uetlibergstrasse 231 in Zürich. Es ist als Multipolygon codiert. Das äussere Polygon stellt das Gebäude dar, und die inneren vier Polygone stellen die Innenhöfe dar. Die äusseren und inneren Polygone werden ganz normal visualisiert, wobei die inneren Polygone in das äussere eingebettet werden. Dadurch wirkt das Gebäude wie ein solider block.

8.1.5 POI Elemente beeinträchtigen Leistung des Programms

Neben Polygonen und Linestrings gibt es Punkte. Diese werden in OpenStreetMap (Points Of Interest) POI genannt. Ein POI besteht aus einer Koordinate. Um diese zu

visualisieren, benötigt es ein entsprechendes 3D-Modell. Detailierte Informationen zur Implementation finden sich in Kapitel 6.1.4. Das Problem, welches hier auftreten kann, ist, dass das 3D-Modell zu detailreich ist, und dadurch sehr rechenintesiv wird. Bei Geräten, welche eine schwache Grafikkarte haben, wird die Leistung des Systems so stark beeinträchtigt, dass die Software nicht richtig verwendet werden kann. Ein Möglicher Ansatz für die Minderung dieses Problems findet sich in Kapitel 8.2.5.

8.1.6 Höhe von Gewässer und Land ist gleich

An diversen Stellen gibt es sowohl Wasser als auch eine Bodenbedeckung. Das Problem ist, dass keine eindeutige Entscheidung getroffen werden kann, welcher Layer oberhalb des Anderen dargestellt werden soll. Dieses Problem ist zum Beispiel bei Brücken besonders gut zu sehen. In der Abbildung 8.4 sieht man, dass die Quaibrücke am Zürich Bellevue unter Wasser dargestellt wird. Wird das Wasser immer oberhalb der Bodenbedeckung angezeigt, gehen zum Beispiel Brücken unter, wird aber die Bodenbedeckung oberhalb des Wassers dargestellt, gehen kleinere Gewässer unter.

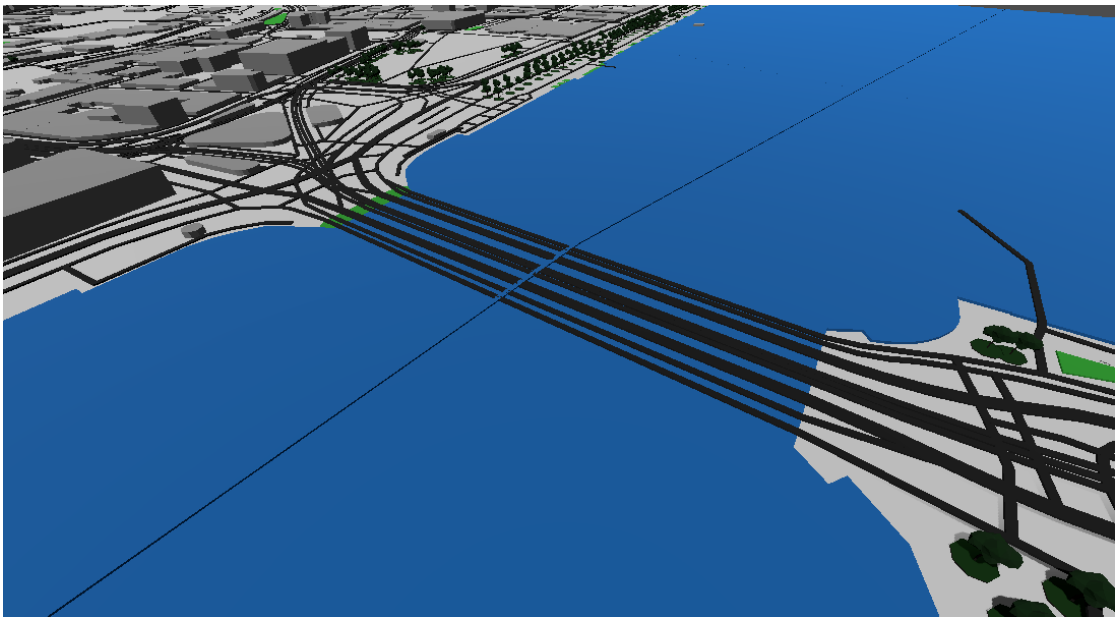
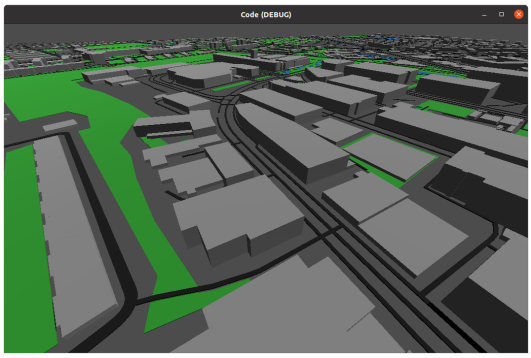


Abbildung 8.4: Brücke wird unter dem Wasser dargestellt

8.2 Ausblick - Mögliche Weiterentwicklung

8.2.1 Erweiterte Beleuchtung implementieren

Aktuell existiert nur eine Basisbeleuchtung- und Schattierung in der Applikation. Dazu wird das von Godot zur Verfügung gestellte DirectionalLight3D verwendet. In der Abbildung 8.5 sieht man den Unterschied mit und ohne Basisbeleuchtung.



(a) Umgebung mit DirectionalLight3D

(b) Umgebung ohne DirectionalLight3D

Abbildung 8.5: Gegenüberstellung Anwesenheit / Abwesenheit von DirectionalLight3D

Diese Lichtquelle verteilt das Licht gleichmässig in alle Richtungen, und simuliert den Lichteffekt der Sonne. Hier könnte man noch den Himmel und die Sonne darstellen. Um die Beleuchtung authentischer zu gestalten, kann man die Lichtintensität bearbeiten, und das Sonnenlicht aus verschiedenen Himmelsrichtungen strahlen lassen. So entsteht ein realistischer Schatteneffekt.

8.2.2 Nicht blockierendes Laden neuer Kacheln

Aktuell bewegt sich der Spieler permanent über einer 4x4 grossen Kachelfläche. Der Spieler kann sich in alle Richtungen bewegen. Wenn sich der Spieler Beispielsweise in eine bestimmte Richtung bewegt und dabei einen Schwellenwert überschreitet, werden automatisch die nachfolgenden vier Kacheln geladen. Während die Objekte in den neu heruntergeladenen Kacheln ausgelesen und visualisiert werden, ist der Main-Thread blockiert, und der Spieler kann sich nicht bewegen. Dieser Prozess dauert - je nach Bandbreite, Hardware und Informationsgehalt der Vektorkacheln - bis zu einigen Sekunden. Ideal wäre es, wenn man einen Mechanismus implementieren würde, bei dem sich der Spieler permanent frei bewegen könnte, ohne das der Hauptthread blockiert wird.

8.2.3 Gleichzeitiges Herunterladen mehrerer Kacheln implementieren

Wenn sich der Spieler in eine bestimmte Richtung bewegt, und damit das Laden neuer Kacheln auslöst, werden diese zuerst heruntergeladen, und erst dann ausgelesen und visualisiert. Bewegt sich der Spieler in die gleiche Richtung zurück, werden die gleichen Kacheln erneut heruntergeladen, und die bestehenden Kacheln werden überschrieben. Man könnte hier einen Mechanismus implementieren, bei dem überprüft wird, ob eine Kachel heruntergeladen wurde. Wenn das der Fall ist, kann man diese als Node3D-Knoten wieder in den Baum einfügen und visualisieren. So könnte man auch einen Mechanismus implementieren, bei dem bereits beim Programmstart die nötigen Kacheln für ein grösseres Feld heruntergeladen werden, sodass diese nur noch visualisiert werden müssen.

8.2.4 Gebäude als Multipolygone korrekt visualisieren

Wie in 8.1.4 beschrieben, werden Multipolygone nicht immer korrekt dargestellt. Ein gutes Beispiel sind Gebäude, welche als solche codiert sind und einen Innenhof haben. Aktuell werden Multipolygone in einzelne Polygone zerteilt, und diese werden wie normale Polygone visualisiert. Zuerst wird das äussere Polygon, welches

das Gebäude darstellt, visualisiert. Die inneren Polygone stellen den Hohlraum eines Gebäudes dar. Es stellt sich heraus, dass es ziemlich anspruchsvoll ist, in Godot ein Polygon mit einem Loch zu erstellen, weil Godot vom Konzept her keine Polygone mit Löchern unterstützt. Es wäre eine deutliche Verbesserung, wenn solche Gebäude korrekt visualisiert werden würden. Ein möglicher Lösungsansatz wird in 8.3.4 beschrieben.

8.2.5 POI Elemente durch weniger Rechenintensive Elemente ersetzen

Aktuell werden POI-Elemente wie Bäume und Sitzbänke als 3D-Modell visualisiert. Dazu wird in Godot jeweils eine sogenannte MultiMeshInstance3D verwendet. Bei der Verwendung von MultiMeshInstance3D sollte man genau darauf achten, welches 3D-Modell gewählt wird. Die Grafikintesität solcher Objekte hängt stark vom Detailgrad des 3D-Modells ab.

Abbildung 8.6: Vergleich 3D-Modelle für Baum

In der Abbildung 8.6 sieht man auf der linken Seite das 3D-Modell für den Baum, der zu Beginn der Implementation verwendet wurde. Dieser ist hat sehr viele Details, beispielsweise die die einzelnen Blätter, die sich in Form, Farbe und Textur voneinander unterscheiden. Damit konnte die Software nicht vernünftig auf Geräten ausgeführt werden, welche eine leistungsschwache Grafikkarte haben. Auf der rechten Seite in Abbildung 8.6 sieht man das 3D-Modell für die Bäume, welches aktuell verwendet wird. Damit läuft die Software auch auf schwacher Hardware flüssig.

8.2.6 Digitales Geländemodell

Eine mögliche Erweiterung der Arbeit ist die Einbindung von Terrain Höhendaten. Besonders in einem bergigen Land wie der Schweiz kann das Landschaftsbild nicht realistisch dargestellt werden, wenn alle Gebäude und Strukturen auf einer flachen Ebene gebaut werden.

8.2.7 Verbesserung der Perfomance

Aktuell ist die ganze Software in Godot-eigenen Programmiersprache GDScript implementiert. Einzige Ausnahme ist das Modul für das Auslesen der Vektorkacheln, welches in Rust implementiert ist. Je nach Informationsgehalt einer Vektorkachel kann es lange dauern, bis diese ausgelesen und visualisiert wurden. Da es sich bei

GDScript um eine interpretierte Programmiersprache handelt, hat man hier bezüglich der Perfomance einen Nachteil. Bevor das Modul für das Auslesen der Vektorkacheln in Rust umgeschrieben wurde, war es in GDScript implementiert. Man konnte einen deutlichen Perfomance Unterschied zwischen diesen beiden Varianten feststellen. Ein möglicher Lösungsansatz, wie man das Visualisieren der Geometrien verbessern könnte, wird in Unterkapitel 8.3.6 beschrieben.

8.3 Prozess für mögliche Weiterentwicklung

8.3.1 Erweiterte Beleuchtung implementieren

Momentan wird die Umgebung in der Applikation von einer fixen Lichtquelle beleuchtet. Eine bessere Option wäre es, die Sonne und den Himmel darzustellen. Das ist ein wenig aufwändiger zu implementieren, aber Godot nimmt hier bereits sehr viel Arbeit ab. So hätte man auch keinen grauen Hintergrund mehr, womit die ganze Umgebung authentischer wirken würde. In einem weiteren Schritt kann man noch die Anpassung des Lichts an die Uhrzeit implementieren, um einen Tag-Nacht-Zyklus zu ermöglichen. So könnte man beispielsweise die Nacht visualisieren und Strassenbeleuchtungen, die als POI codiert sind, leuchten lassen. Zudem würde sich der Schatten der aktuellen Uhrzeit, also der aktuellen Position der Sonne, anpassen. Diese kleinen Details lassen die 3D-Welt sehr realistisch aussehen.

8.3.2 Nicht blockierendes Laden neuer Kacheln

In der aktuellen Version blockiert der Aufbau neuer Kacheln den Main-Thread. Diese Blockierung könnte umgangen werden, indem die Visualisierung neuer Kacheln in einen Hintergrundthread ausgelagert wird, sofern die Hardware dies zulässt. GDScript bietet die Möglichkeit, Multithreading zu implementieren. Dazu müssten allerdings die Funktionen, welche für die Visualisierung angepasst werden, damit sie überhaupt aus dem Hintergrundthread ausgeführt werden können. Diese Anpassungen wären relativ aufwändig, würden sich allerdings lohnen, damit der Spieler nicht unterbrochen wird.

8.3.3 Gleichzeitiges Herunterladen mehrerer Kacheln implementieren

Die Idee ist, dass sich der Spieler immer über einem Feld mit einer bestimmten Grösse bewegt. Aktuell ist es ein 4x4 grosses Feld. Wenn sich der Spieler in eine Richtung bewegt, wird die nächste Kachelreihe heruntergeladen, und in einem weiteren Schritt ausgelesen und visualisiert.

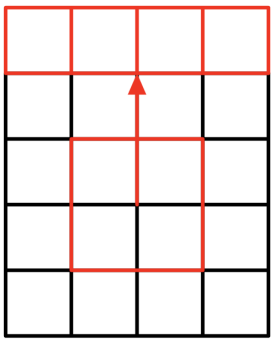


Abbildung 8.7: Laden einer neuen Kachelreihe

In der Abbildung 8.7 sieht man, wie der Spieler sich in die angezeigte Pfeilrichtung bewegt, und dabei den Schwellenwert - das rote Quadrat - überschreitet, und somit das Laden der neuen Kachelreihe auslöst (rot markiert). Man könnte bereits weitere umgliegende Kacheln herunterladen, sodass diese beim überschreiten des Schwellenwerts nur noch visualisiert werden müssen.

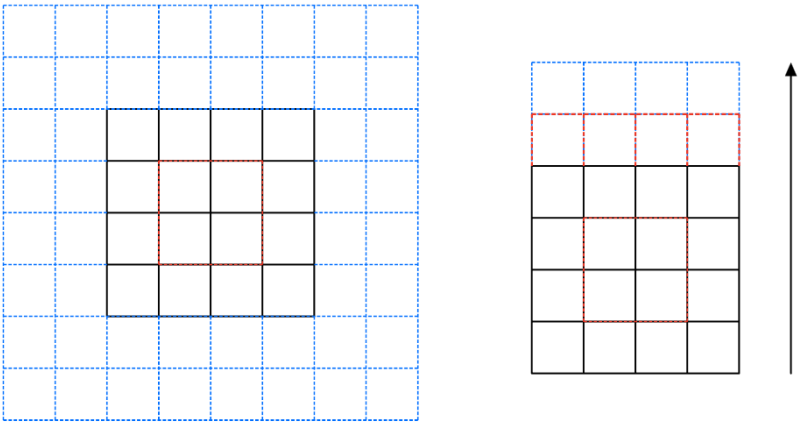


Abbildung 8.8: Herunterladen umliegender Kacheln

In Abbildung 8.8 sieht man auf der linken Seite das 4x4 grosse Feld, welches für den Spieler sichtbar ist. Man könnte einen Mechanismus implementieren, dass bereits die umliegenden Kacheln - blau markiert - heruntergeladen werden, ohne dass der Spieler den rot markierten Schwellenwert überschreitet. In Abbildung 8.8 auf der rechten Seite sieht man das gleiche, was in Abbildung 8.7 passiert. Der Unterschied ist, dass bereits die nächste Kachelreihe heruntergeladen wird, auch wenn der Schwellenwert noch nicht überschriiten wurde. Das Ziel ist also, die Kacheln vor dem Rendering herunterzuladen. Das kann relativ einfach implementiert werden. Damit bereits heruntergeladene Kacheln nicht erneut heruntergeladen werden, beispielsweise wenn sich der Spieler in die gleiche Richtung zurück bewegt, kann man eine einfache Liste verwenden, und jeweils nach dem Herunterladen einer neuen Vektorkachel den Dateinamen in der Liste speichern. Ist der Name einer bestimmten Vektorkachel in der Liste vorhanden, überspringt man den Download-Prozess.

8.3.4 Gebäude als Multipolygone korrekt visualisieren

In 8.2.4 wird beschrieben, weshalb die darstellung von Gebäuden, welche ein Multipolygon sind, eine Herausforderung ist. Die inneren Polygone stellen den Holhraum eines Gebäudes dar. Wenn das äussere Polygon - das Gebäude - visualisiert wird, werden die Vertices des Gebäudes miteinander verbunden, und der Bereich innerhalb der Verbindungslinieren wird ausgefüllt. Anschliessend wird das innere Polygon - der Innenhof / Hohlraum - erstellt und in das äussere Polygon gesetzt, sodass sich das äussere und ineere Polygon schneiden.

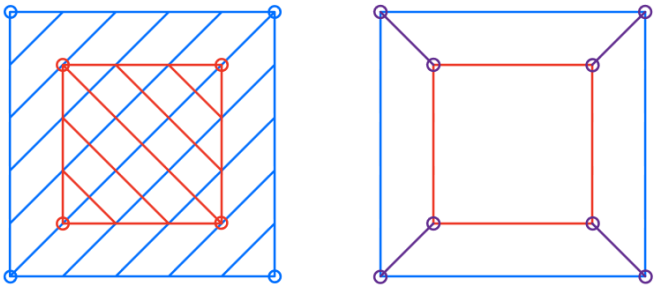


Abbildung 8.9: Polygon mit Loch

Die Abbildung 8.9 auf der linken Seite zeigt die aktuelle Implemantation. Das blaue Quadrat stellt das Gebäude dar. Die blauen Vertices werden miteinander verbunden, und der Bereich innerhalb der Verbindungslinien wird ausgefüllt. Dieser Bereich ist blau schraffiert. Das kleine rote Quadrat stellt den Hohlraum dar. Die roten Vertices werden miteinander verbunden, und anschliessend wird der Raum dazwischen ausgefüllt (rot schraffiert). Man sieht, wie das innere Polygon in das äussere Polygon eingebettet wird. Zu Beginn überlegte man sich, dass man das `visibility`-Attribut des inneren Polygons auf `false` setzen könnte. Dieser Ansatz funktioniert aber nicht, weil dann immer noch der blaue Bereich, der in Abbildung 8.9 links blau schraffiert ist, übrig bleibt, und somit kein Hohlraum sichtbar ist. Ein möglicher, aber anspruchsvoller Lösungsansatz, wird in der Abbildung Abbildung 8.9 auf der rechten Seite gezeigt. Man verbindet die Vertices des äusseren Polygons direkt mit den Vertices des inneren Polygons. In der Grafik rechts sind diese Vertices als violette Punkte dargestellt. Somit hat man vier trapezförmige Polygone, welche zusammen aussehen wie ein einzelnes Polygon mit einem Hohlraum.

8.3.5 Digitales Geländemodell

Es könnten zum Beispiel die Digital LiDAR Models (DTMs) von “Sonny” [23] verwendet werden. Sonny hat bereits aus verschiedenen Quellen Höhendaten für Europa zusammengetragen und stellt diese für jeglichen Zweck zur Verfügung. Seine Daten können sowohl als .hgt Files oder als GeoTIFF Files verwendet werden.
Eine andere Datenquelle könnte das Jet Propulsion Laboratory der NASA sein [24]. Die NASA hat mit der Shuttle Radar Topography Mission Höhendaten für fast die ganze Welt gesammelt und diese 2015 der Öffentlichkeit zur Verfügung gestellt.
Für beide Quellen müssen einige Punkte beachtet werden. Für alle verfügbaren Daten wird ein Reader benötigt. Der aktuell im Projekt verwendete Reader kann nur MVT Files lesen, welche keine Höhendaten beinhalten. Die gelesenen Höhendaten müssen mit den Daten aus den MVTs zusammengeführt werden, was darum ein Problem darstellen wird, weil ein Tile aus den Datenquellen nicht die gleiche Fläche beschreiben wird. Das bedeutet, dass eine der Datenquellen “aufgeschnitten” werden muss. Ein weiteres Problem wird irgendwann darin bestehen, dass nicht für die gesamte Welt Höhendaten vorhanden sind. Die Daten von Sonny umfassen nur Europa. Die Daten der NASA decken fast die ganze Welt ab, allerdings nicht für die Nördlichsten Breitengrade oder Antarktika. Ebenfalls wäre es von Vorteil die Daten ähnlich wie bei den MVTs “on demand” von einem Webserver zu laden, damit nicht die Daten der ganzen Welt lokal gespeichert werden müssen.

8.3.6 Verbesserung der Perfomance

Godot bietet eine API für C#. So kann man als Skriptsprache C# verwenden, anstatt GDScript. Es ist auch möglich, nur Perfomance-kritische Teile der Software in C# zu implementieren, und diese Funktionen über die C#-API von Godot über GDScript aufzurufen. Das gleiche funktioniert auch umgekehrt [25]. Da C# eine kompilierte Sprache ist, lässt sich so eine Verbesserung der Perfomance bei der Visualisierung von Geometrien erreichen. Es ist auch möglich, teile der Software in C++ oder Rust zu implementieren. Godot bietet keine API für C++ oder Rust an, so wie bei C#. Man kann aber Erweiterungen / Module in C++ oder Rust schreiben, und diese als dynamische Programmbibliothek kompilieren. Diese Bibliothek kann in Godot mit GDNative [26] genutzt werden. Bei GDNative handelt es sich um eine Godot-spezifische Technologie, welche es der Game-Engine erlaubt, zur Laufzeit mit solchen Bibliotheken zu interagieren.

Kapitel 9

Projektplanung

9.1 Projektplanungsmethode

Für die Langzeitplanung wird das Rational Unified Process (RUP) Modell verwendet, für die Kurzzeitplanung werden Sprints erstellt, ähnlich wie bei SCRUM. Das RUP-Modell definiert die Meilensteine. Die Meilensteine werden in Iterationen (Sprints) abgearbeitet, wobei eine Iteration bei diesem Projekt zwei Wochen dauert. Diese Projektplanungsmethode wurde gewählt, da die Projektmitarbeiter (Josip Di Benedetto, Fabian Freitag) bereits in vorherigen Projekten nach dieser Methode gearbeitet haben und bereits positive Erfahrungen damit gemacht haben. Zusätzlich hat man den Vorteil, dass man bei der Planung der Iterationen viel Spielraum hat. Hier hat man sich für eine agile Arbeitsmethode entschieden, damit die Planung der Iterationen angepasst werden kann, sobald sich Anforderungen oder andere Kriterien ändern. Die agile Planung der Iterationen ermöglicht es, besser auf das Feedback des Projektbetreuers oder auf das Feedback des Industriepartners eingehen zu können.

9.2 Involvierte Personen und Verantwortlichkeiten

Name	Rolle - Verantwortlichkeit
Josip Di Benedetto	Entwickler - Implementation der Funktionalität und Dokumentation.
Fabian Freitag	Entwickler - Implementation der Funktionalität und Dokumentation.
Prof. Stefan F. Keller	Betreuer - Unterstützung für das Projekt, Bewertung des Projekts.
Primin Kalberer	Industriepartner Sourcepole AG - Bereitstellung des Readers für Vektorkacheln und Bereitstellung Tileset CH, Kommunikation der Anforderungen.

Tabelle 9.1: Rollen und Verantwortlichkeiten

9.3 Prozesse

Beschreibung	Termin	Zweck
Sprint planning	Jeden zweiten Montag	Planung und Festlegung der Ziele für kommende Iteration.
Meeting mit Stefan Keller	Jeden Montag	Betreuer über Projektstand, Fortschritte und Probleme informieren. Feedback entgegen nehmen.
Meeting mit Pirmin Kalberer	Jeden Donnerstag	Industriepartner über Produktstatus informieren, Tipps und Feedback entegegen nehmen, Probleme besprechen
Sprint review	Jeden zweiten Sonntag	Prüfen, ob Ziele der aktuellen Iteration erreicht wurden.

Tabelle 9.2: Prozesse

9.4 Langzeitplanung

Das ganze Projekt wird gemäss RUP in vier Phasen unterteilt:

Phase	Start	Ende
Inception	2023-09-18	2023-10-01
Elaboration	2023-10-02	2023-10-15
Construction	2023-10-16	2023-12-10
Transition	2023-12-11	2023-12-22

Tabelle 9.3: Phasen

9.4.1 Inception Phase - Sprint 1

In der Inception Phase geht es darum, die Aufgabe zu verstehen. Ziel ist es, grobe Anfoderungen zu sammen, welche in der Elaboration Phase verfeinert werden. Das Verständnis der Aufgabenstellung zusammen mit einem groben Überblick der Anfoderungen hilft dabei, den Umfang des Projekts festzulegen, und das Projekt zu planen. Daraus ergeben sich folgende Ziele:

- Definition der Aufgabenstellung
- Einarbeitung in Technologien
- Wichtigste Anforderungen sammeln (grob)
- Projektumfang festlegen (grob)
- Erstellung des Projektplans

9.4.2 Elaboration Phase - Sprint 2

In dieser Phase ist eine grobe Vision des Projekts vorhanden, und der Umfang konnte auf das Wesentliche eingeschränkt werden. Hier werden die genauen Anfoderungen definiert und mögliche Risiken identifiziert. Am Ende dieser Phase sollte ein Prototyp vorhanden sein, welcher den Entwicklern eine solide Basis für die Construction Phase liefert. Folgende Ziele sollten erreicht werden:

- Prototyp entwickeln (Darstellung der Gebäudeumrisse)
- Verfeinerung der Aufgabenstellung und Vision
- Spezifische Requirement Analyse (FR und NFR)
- Identifikation und Minderung von Risiken
- Domainanalyse erstellen (falls notwendig)

9.4.3 Construction Phase - Sprint 3, 4, 5 und 6

In dieser Phase ist ein Prototyp vorhanden, welcher min. die Gebäudeumrisse basierend auf den Geometrien der ausgelesenen Vektorkachel zeichnen kann. Das Projektteam hat ein grundlegendes Verständnis der verwendeten Game Engine (Godot) und des Vektorkachel-Schemas. Damit ist es jetzt möglich, Architekturentscheidungen zu treffen, und basierend darauf die restliche Funktionalität zu implementieren. Im ersten Sprint der Construction Phase (Sprint 3) sollen die Strassen dargestellt werden. Ziel ist es, dass diese zusammen mit den Gebäuden erkennbar dargestellt werden. Als Basis wird der bestehende Prototyp verwendet. Werden Strassen und Gebäude korrekt visualisiert, wird damit begonnen, die Details zu implementieren. Der zweite Sprint der Construction Phase (Sprint 4) befasst sich mit der Visualisierung von Gebäudedetails, und der Darstellung von Punkten (bspw. Bäume) und Gewässern. Ziel dieses Sprints ist es, mindestens die Gebäudehöhe, Seen, Flüsse und Bäume darzustellen. Ziel von Sprint 5 ist es, das automatische Nachladen mehrerer Vektorkacheln zu implementieren. Damit wäre die Basisimplementation abgeschlossen und die minimalen Ziele wurden erfüllt. Schätzungen zufolge sollten die eben genannten Ziele bis zum 2023-11-26 erreicht werden können. Der letzte Sprint der Construction Phase (Sprint 6) befasst sich mit der Ausarbeitung der Details. Hier geht es darum, Texturen zu implementieren, und die korrekte Breite für Strassen und Flüsse darzustellen. Falls dieses Ziel deutlich vor Ende des Sprints erreicht werden sollte, befassen sich die Projektmitarbeiter mit der Implementation der Höhendaten. Ziel ist es nicht, diese zu implementieren, sondern zu verstehen, wie bei einer Implementation vorgegangen werden könnte. Auch sollte eine experimentelle Implementation der Höhendaten durchgeführt werden.

9.4.4 Transition Phase - Sprint 7

In dieser Phase soll das Projekt zu Ende gebracht werden. Dazu gehört es, letzte kleine Programmierarbeiten fertigzustellen, die Dokumentation für die Abgabe vorzubereiten und sie schlussendlich abzugeben.

9.5 Meilensteine

Phase	Ziel	Bis
Inception	Aufgabenstellung definieren, Einarbeitung in Technologien	2023-10-01
Elaboration	Prototyp - Gebäudeumrisse als Polygon zeichnen	2023-10-15
Construction	Strassen darstellen	2023-10-29
Construction	Details ausarbeiten - Gebäudehöhe, Bäume und Gewässer darstellen	2023-11-12
Construction	Laden mehrerer Kacheln implementieren	2023-11-26
Construction	Details ausarbeiten - Texturen, Breite für Strassen und Flüsse	2023-12-10
Transition	Dokumentation fertigstellen	2023-12-22

Tabelle 9.4: Meilensteine

9.6 Risikoanalyse

Beschreibung	Wahrscheinlichkeit	Auswirkung	Verminderung
Performance Probleme beim Laden grosser Kacheln	Niedrig	Verminderte Leistung, Benutzererfahrung beeinträchtig	Verbesserung des Tile Readers durch Industriepartner
Schwierigkeiten bei der Interpretation der Daten (bspw. komplexe Codierung der Geometrien)	Mittel	Verzögerung bei der Entwicklung, da mehr Zeit für Schulung	Detailliertes Befassen mit Vektorkacheln, programmatisches Analysieren der Geodaten
Ineffiziente Nutzung der Tools und daraus resultierende Designfehler	Niedrig	Schlechte Effizienz der Software (des Endprodukts)	Gründliche Einarbeitung in Godot, bei Fragen Doku oder Industriepartner kontaktieren
Fehlerhaftes Datenset	Niedrig	Verzögerung bei der Implementation der Funktionalität	Verwendung eines neuen Datensets
Perfomance Probleme durch Grafikintensives rendering von Objekten	Hoch	Schlechte Benutzererfahrung für Geräte ohne Grafikkarte	Verwendung von weniger Grafikintensiven Objekten
Perfomance Probleme durch HTTP-basiertes Laden der StreetsGL Vektorkacheln	Mittel	Schlechte Benutzererfahrung bei geringer Bandbreite	Implementation eines Buffering Mechanismus

Tabelle 9.5: Risiken

9.6.1 Updates Risikoanalyse

- 2023-10-14: Eintrittswahrscheinlichkeit für Risiko "Performance Probleme beim Laden grosser Kacheln"wurde von "Hochäuf "Niedrig"gestuft. Industriepartner hat neuen Reader für Vektorkacheln bereitgestellt.
- 2023-10-20: Eintrittswahrscheinlichkeit für Risiko SSchwierigkeiten bei der Interpretation der Daten"wurde von Hoch auf mittel gestuft. Die Geodaten für Gebäude machen Sinn und konnten entsprechend dargestellt werden. Geodaten für Strassen sind noch nicht ganz klar und müssen weiter analysiert werden.
- 2023-11-21: Neues Risiko "Perfomance Probleme durch Grafikintensives rendering von Objekten"hinzugefügt. Aktuell wird für die Darstellung von Bäumen ein detailreiches 3D-Objekt verwendet. Bei der Implementation einer Kachel fiel der Fehler nicht auf, weil durch die geringe Anzahl Bäume keine Perfomance Einbusse festgestellt wurde. Erst nachdem mehrere Kacheln mit

vielen Bäumen dargestellt wurden, fiel auf, dass die Software unbrauchbar ist, wenn man eine schwache Grafikkarte hatte.

- 2023-11-21: Eintrittswahrscheinlichkeit für Risiko Ïneffiziente Nutzung der Tools und daraus resultierende Designfehler"von Mittel auf "Niedrig"gestuft. Man ist mittlerweile sehr zuversichtlich bei der Arbeit mit den vorhandenen Tools.
- 2023-11-21: Risiok "Fehlerhaftes Datenset"nachträglich hinzugefügt. Die Tags in den Vektorkacheln waren unvollständig. Man verwendet jetzt provisorisch die Vektorkacheln von StreetsGL, daher ist die Eintrittswahrscheinlichkeit "Niedrig".
- 2023-11-21: Risiko "Perfomance Probleme durch HTTP-basiertes Laden der StreetsGL Vektorkacheln"hinzugefügt. Aufgrund des unvollständigen Datensets wurde diese Lösung implementiert. Da wir die Kacheln nicht lokal haben, müssen diese heruntergeladen werden, bevor Sie dargestellt werden. Da die Implementation eines Buffering Mechanismus als Anspruchsvoll angesehen wird, ist die Eintrittswahrscheinlichkeit "Mittel".

Kapitel 10

Zeiterfassung

10.1 Sprint 1

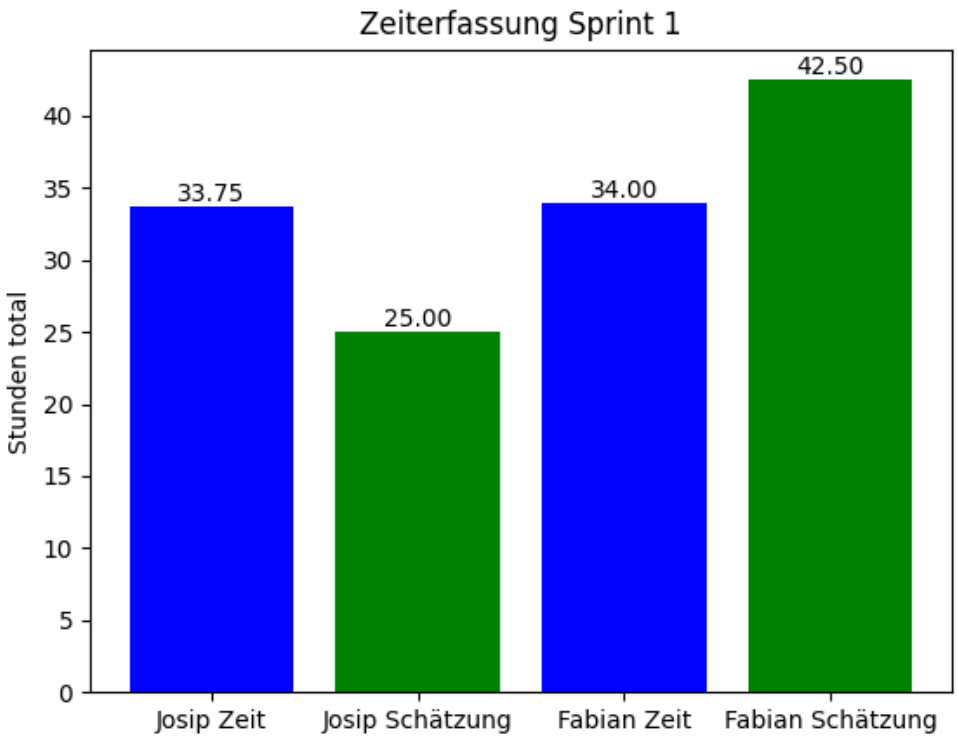


Abbildung 10.1: Aufgewendete und geschätzte Zeit Sprint 1

10.2 Sprint 2

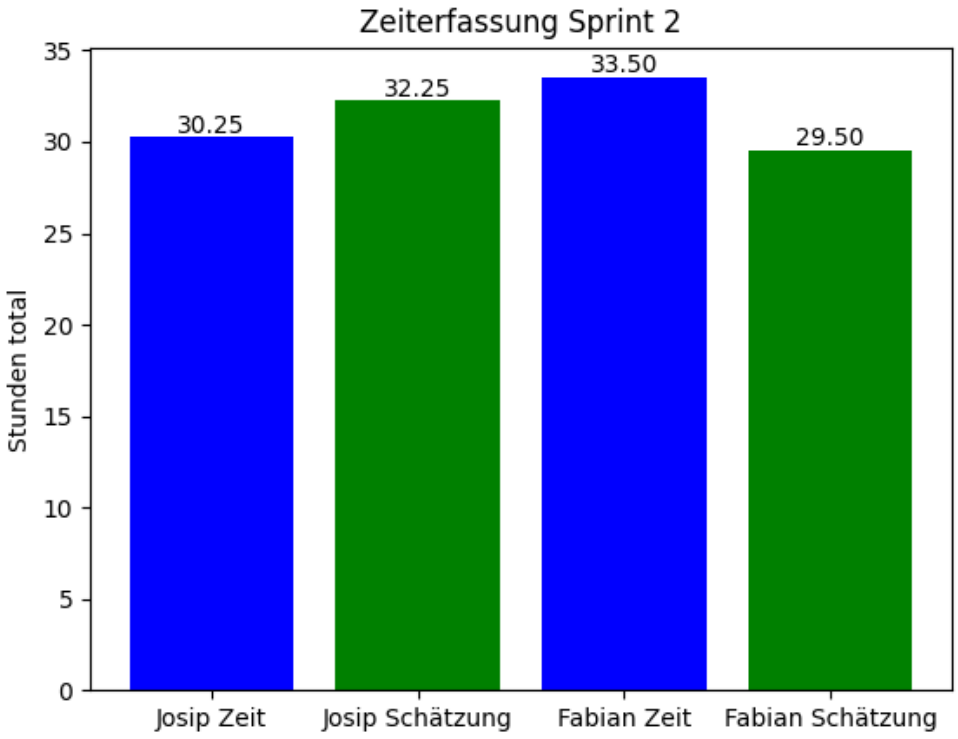


Abbildung 10.2: Aufgewendete und geschätzte Zeit Sprint 2

10.3 Sprint 3

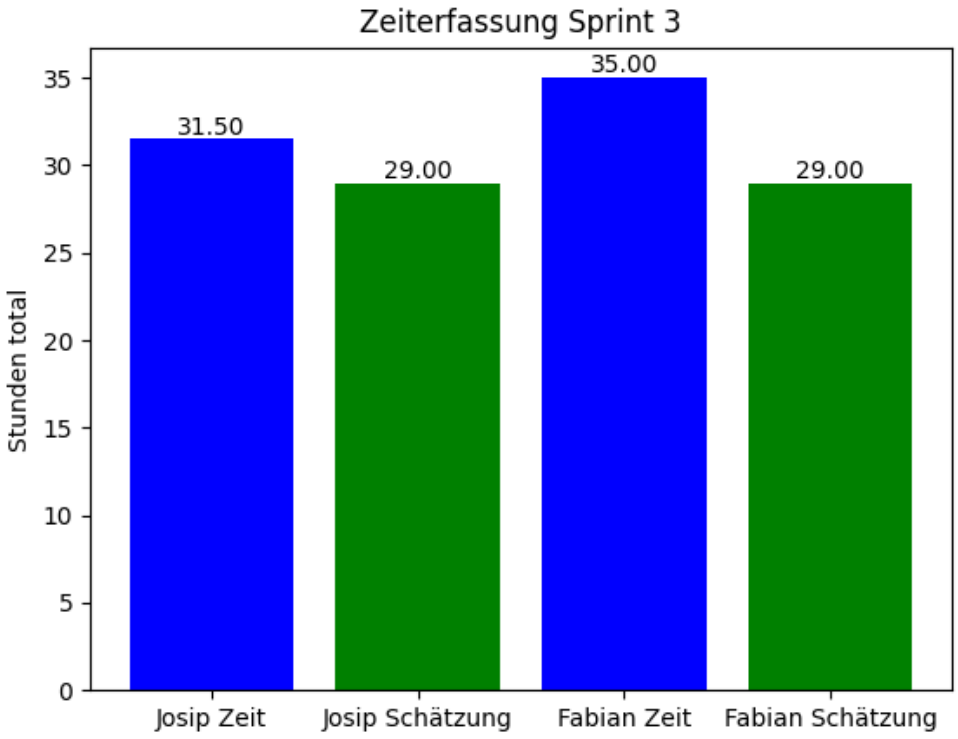


Abbildung 10.3: Aufgewendete und geschätzte Zeit Sprint 3

10.4 Sprint 4

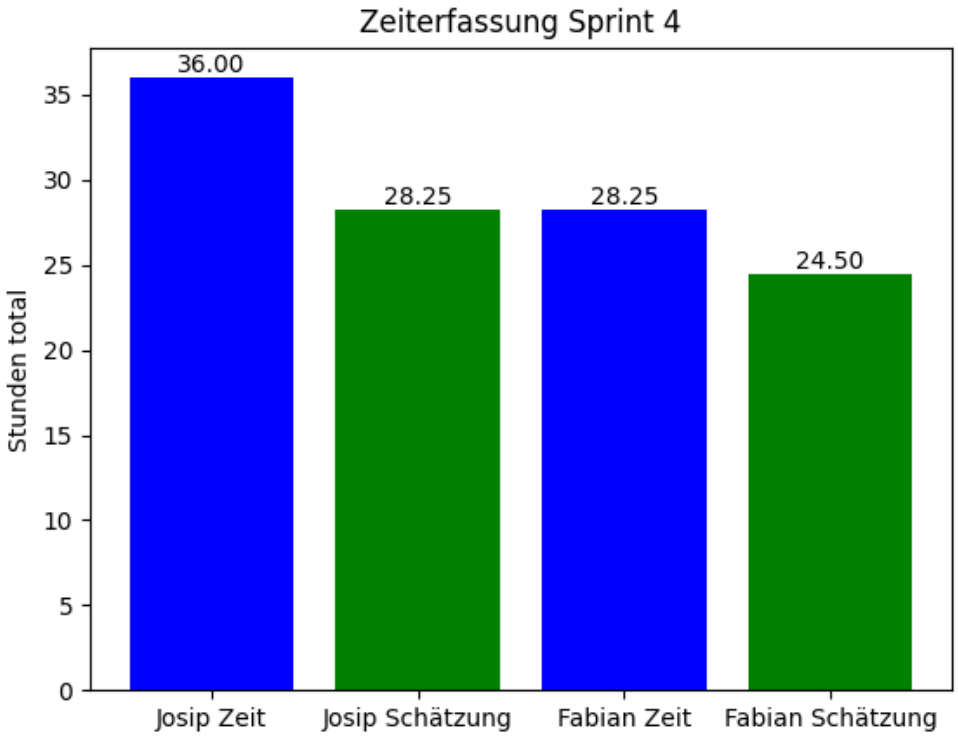


Abbildung 10.4: Aufgewendete und geschätzte Zeit Sprint 4

10.5 Sprint 5

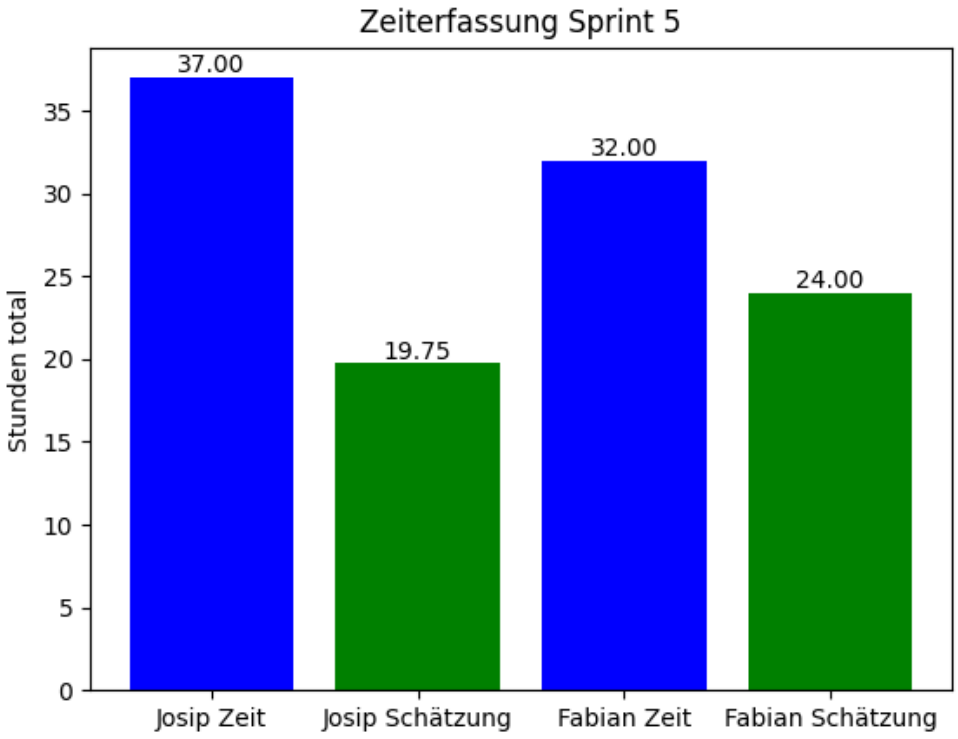


Abbildung 10.5: Aufgewendete und geschätzte Zeit Sprint 5

10.6 Sprint 6

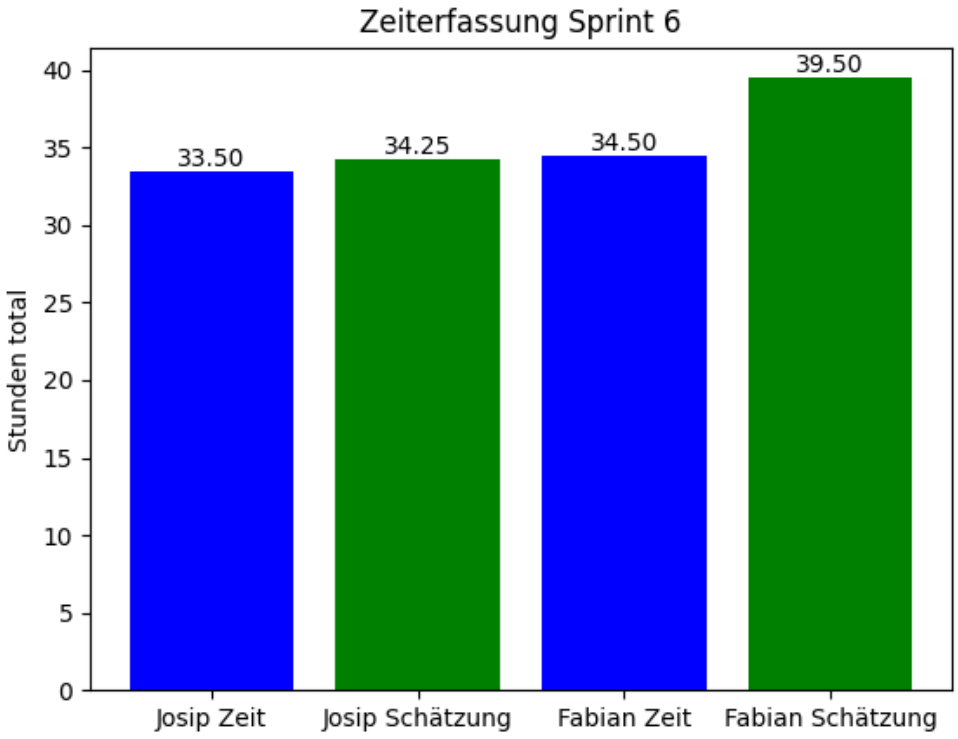


Abbildung 10.6: Aufgewendete und geschätzte Zeit Sprint 6

10.7 Sprint 7

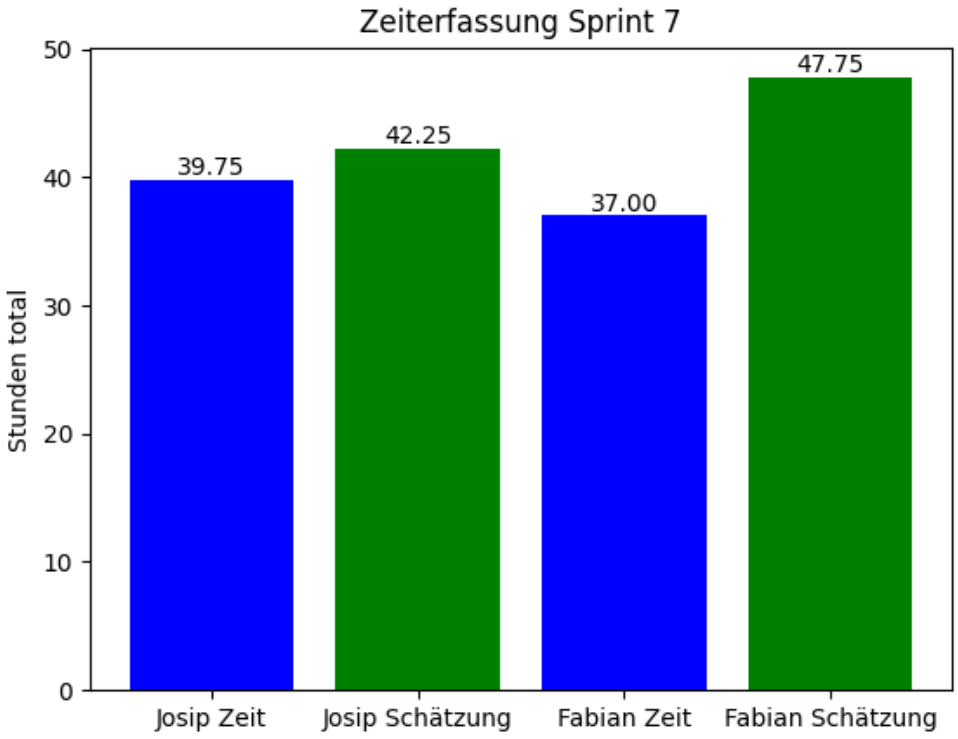


Abbildung 10.7: Aufgewendete und geschätzte Zeit Sprint 7

Kapitel 11

Persönliche Berichte

11.1 Fabian Freitag

Das Thema OSM-Daten in einer Game Engine darzustellen war für mich komplett neu. Ich habe noch quasi gar nicht mit OSM-Daten gearbeitet und ich habe noch nie ein Spiel in einer Game Engine programmiert. Besonders der zweite Teil hat mich persönlich sehr interessiert. Ich fand es eine spannende Herausforderung herauszufinden, wie wir die Daten so aufbereiten können, dass wir sie in Godot darstellen können. Ich war mir zu Beginn der Arbeit nicht wirklich bewusst, wie viel Aufwand es sein wird, dieses Projekt zu verwirklichen. Mit dem Endprodukt bin ich zufrieden. Ich hätte Anfangs etwas mehr erhofft, allerdings hat sich im Verlaufe des Projektes gezeigt, dass viel mehr nicht realistisch gewesen wäre. Die Zusammenarbeit mit Josip hat gut funktioniert. Wir kannten uns bereits aus dem SE Project und wussten daher, worauf wir uns einlassen, wenn wir die SA zusammen machen. Wir haben uns jeweils einen Tag in der Woche reserviert, an dem wir uns getroffen haben, um an dem Projekt zu arbeiten. Für mich war dieser Tag jeweils sehr wertvoll, da wir so direkt allfällige Probleme miteinander besprechen konnten.

11.2 Josip Di Benedetto

Zu Beginn war es für uns schwer, den Umfang der Arbeit festzulegen. Das Thema OpenStreetMap, Godot und Mapbox Vektorkacheln war für uns neu. Unser Industriepartner konnte uns gut dabei helfen, die Aufgabenstellung und den Umfang zu definieren. Meiner Meinung nach waren diese angemessen für eine Studienarbeit. Ich selbst bin zufrieden mit dem Endprodukt, obwohl ich gerne mehr Zeit zur Verfügung gehabt hätte, um einige kleine Baustellen zu beseitigen, welche aktuell noch offen sind. Mit dem Abschluss dieser Studienarbeit können wir mit gutem Gewissen eine weitere Erfahrung zum Thema Softwareentwicklung und agiles Arbeiten zu unserem Wissen hinzufügen. Ich finde auch, dass es ein spannendes Projekt war, und dass es auch einem sinnvollen Zweck dient. Bereits jetzt kann man bestimmte Objekte in der 3D-Welt in Godot mit einem anderen Kartenanbieter vergleichen, und sieht beispielsweise, wo überall Gebäudehöhen fehlen, sodass man diese in OpenStreetMap nachtragen kann. Das Projekt hat diesbezüglich noch viel Potenzial.

Kapitel 12

Anhang

12.1 Eigenständigkeitserklärung

OST
Ostschweizer
Fachhochschule

Erklärung zur Urheberschaft

Erklärung	Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich trage die Verantwortung für die Qualität des Textes sowie die Auswahl aller Inhalte und habe sichergestellt, dass Informationen und Argumente mit geeigneten wissenschaftlichen Quellen belegt bzw. gestützt werden. Die aus fremden Quellen übernommenen Texte, Gedankengänge, Konzepte, Grafiken usw. in meinen Ausführungen habe ich als solche eindeutig gekennzeichnet und mit vollständigen Verweisen auf die jeweilige Quelle versehen. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.
KI-Einsatz ohne Kennzeichnungspflicht	Ich bin mir bewusst, dass die Nutzung maschinell generierter Texte keine Garantie für die Qualität von Inhalten und Text gewährleistet. Ich versichere daher, dass ich mich textgenerierender KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Ich verantworte die Übernahme jeglicher von mir verwendeter maschinell generierter Textpassagen vollumfänglich selbst. Ich versichere, dass ich keine KI-Schreibwerkzeuge verwendet habe, deren Nutzung der Prüfer / die Prüferin explizit schriftlich ausgeschlossen hat.
Verstoss	Mir ist bekannt, dass ein Verstoss gegen die genannten Punkte prüfungsrechtliche Konsequenzen haben und dazu führen kann, dass die Prüfungsleistung mit “nicht ausreichend” bzw. „nicht bestanden“ bewertet wird.
Ort/Datum	Rapperswil, 22. Dezember 2023
Unterschrift Verfasser/ Verfasserin	Fabian Freitag Josip di Benedetto

12.2 Nutzungsrecht

OST
Ostschweizer
Fachhochschule

Vereinbarung

Ohne anderslautende Vereinbarungen stehen die Schutzrechte und das Know-how an der Studienarbeit oder Bachelorarbeit (nachfolgend ‚Arbeit' genannt) und an der in diesem Rahmen geschaffenen Güter, wie Software, sowohl dem Rechtsträger der OST Ostschweizer Fachhochschule, dem für die Arbeit verantwortlichen Professoren sowie dem Verfasser der Arbeit resp. Entwickler der in diesem Rahmen geschaffenen Güter, wie Software, zu.

Die genannten Parteien übertragen sich gegenseitig nicht exklusiv, jedoch unentgeltlich, weltweit, sachlich und zeitlich unbeschränkt die jeweiligen Schutzrechte und das Know-how an der Arbeit und an der in diesem Rahmen geschaffenen Güter, wie Software, einschliesslich dem Recht zur Weiterübertragung, ab. Entsprechend steht es jeder Partei zu, sämtliche Schutzrechte an der Arbeit resp. an der in diesem Rahmen geschaffenen Güter, wie Software, beliebig weltweit, zeitlich und sachlich unbeschränkt zu verwerten. Darunter fällt namentlich aber nicht abschliessend das Recht zur Lizenzierung in jeder Art, Umfang und Form, das Recht zur Bearbeitung und damit zur Nutzung z. B. der Software oder Komponenten hiervon als Grundlage eines neuen schutzfähigen Guts. Die Parteien erklären sich gegenseitig den Verzicht auf Namensnennung bei der Verwertung der Schutzrechte und des Know-how durch eine oder mehrere Parteien gemeinsam und stimmen namentlich zu, dass jede Partei allein unter ihrem eigenen Namen die Schutzrechte resp. das Know-how verwertet. Die vorliegende gegenseitige unentgeltliche Übertragung der Schutzrechte resp. des Know-how bezieht sich auch auf Verwertungsarten, welche heute noch nicht bekannt sind.

Rapperswil, den 22.12.23 .....
Die Studenten

Rapperswil, den 22.12.23 .....
Der Betreuer / die Betreuerin der Studienarbeit

12.3 Einverständniserklärung EPrint

OST
Ostschweizer
Fachhochschule

Einverständniserklärung Publikation auf eprints.ost.ch

☒ SA
○ BA

Titel der Arbeit: Serious 3D-VR-Game mit realistischen Stadtwelten

Team: Fabian Freitag, Josip di Benedetto

Betreuer: Stefan F. Keller

Wir sind mit der Publikation unserer Arbeit auf eprints.ost.ch einverstanden, sofern für diese Arbeit keine Geheimhaltungsvereinbarung unterzeichnet wurde.
Nach Bekanntgabe der Note haben wir die Möglichkeit innert 14 Tagen Einsprache zu erheben und das Einverständnis zur Publikation der Arbeit auf eprints.ost.ch zurückzuziehen. In diesem Falle wird nur der Abstract publiziert.

Rapperswil, 22.12.2023

Name(n) Unterschrift(en)

Fabian Freitag

Josip di Benedetto

Literaturverzeichnis

[1] Sourcepole AG, Linux and Open Source Solutions, Adresse: Weberstrasse 4, 8004 Zürich. https://www.sourcepole.com. Accessed at 2023-12-22.

[2] https://www.openstreetmap.org/. Accessed at 2023-12-22.

[3] https://godotengine.org/. Authors: Juan Linietsky, Ariel Manzur und Mitwirkende. Accessed at 2023-12-22.

[4] https://godotengine.org/license/. Authors: Juan Linietsky, Ariel Manzur und Mitwirkende. Accessed at 2023-12-22.

[5] https://docs.godotengine.org/de/4.x/tutorials/scripting/gdscript/gdscript_basics.html. Authors: Juan Linietsky, Ariel Manzur und Godot Community. Accessed at 2023-12-22.

[6] https://www.unrealengine.com/en-US. Authors: Epic Games. Accessed at 2023-12-22.

[7] https://unity.com/de. Authors: Unity Technologies. Accessed at 2023-12-22.

[8] https://techcrunch.com/2023/09/18/unity-reportedly-backtracking-on-new-fees-after-developers-revolt/. Authors: Devin Coldewey. Accessed at 2023-12-22.

[9] https://docs.mapbox.com/help/glossary/vector-tiles/. Accessed at 2023-12-22.

[10] https://github.com/mapbox/vector-tile-spec. Authors: Vladimir Agafonkin, John Firebaugh, Eric Fischer, Konstantin Käfer, Charlie Loyd, Tom MacWright, Artem Pavlenko, Dane Springmeyer, Blake Thompson. Version 2.1 (2016-01-19). Accessed at 2023-12-22.

[11] https://github.com/strandedkitty/planetiler. Authors: StrandedKitty. Accessed at 2023-12-22.

[12] https://github.com/onthegomap/planetiler. Authors: planetiler. Version 0.7.0 (2023-10-02). Accessed at 2023-12-22.

[13] https://openmaptiles.org/schema/. Authors: OMT community. Accessed at 2023-12-22.

[14] https://protobuf.dev/. Authors: Google. Accessed at 2023-12-22.

[15] https://en.wikipedia.org/wiki/Variable-length_quantity#Zigzag_encoding. Authors: Google. Accessed at 2023-12-22.

[16] https://github.com/StrandedKitty/streets-gl. Authors: StrandedKitty. Accessed at 2023-12-22.

[17] https://docs.godotengine.org/en/stable/tutorials/3d/using_multi_mesh_instance.html. Authors: Juan Linietsky, Ariel Manzur und Mitwirkende. Accessed at 2023-12-22.

[18] https://github.com/bitwes/Gut. Author: Bitwes und Mitwirkende. Accessed 2023-12-22.

[19] https://github.com/Scony/godot-gdscript-toolkit. Author: Scony. Accessed at 2023-12-22.

[22] https://pkg.sourcepole.ch/switzerland.zip. Authors: Pirmin Kalberer. Accessed at 2023-12-22.

[23] https://sonny.4lima.de/. Author: Sonny. Accessed at 2023-12-22.

[24] https://www2.jpl.nasa.gov/srtm/. Author: NASA JPL. Accessed at 2023-12-22.

[20] https://en.wikipedia.org/wiki/Cyclomatic_complexity. Author: A J Sobey. Accessed at 2023-12-22.

[21] https://radon.readthedocs.io/en/latest/commandline.html#the-cc-command. Author: Michele Lacchia. Accessed at 2023-12-22.

[25] https://docs.godotengine.org/en/stable/tutorials/scripting/cross_language_scripting.html. Author: Juan Linietsky, Ariel Manzur und Mitwirkende. Accessed at 2023-12-22.

[26] https://docs.godotengine.org/en/3.5/tutorials/scripting/gdnative/what_is_gdnative.html. Author: Juan Linietsky, Ariel Manzur und Mitwirkende. Accessed at 2023-12-22.