HTL - Perg
Höhere Abteilung für Informatik

Diplomarbeit

DISC – Digital Industrial Steel Campus

Projektteam: Gregor Scheuchenstuhl

Projektbetreuer: Prof. OStR Dipl.-Ing. Roland Eggetsberger

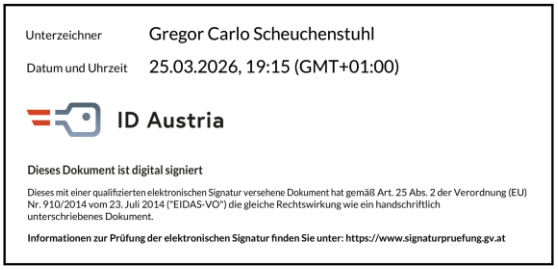
In Zusammenarbeit mit voestalpine Stahl GmbH
Betreuer Markus Wall, Abteilung TSI

Bearbeitungszeitraum: 01.08.2025 – 26.03.2026.

HTL Perg | Machlandstr. 48 | A-4320 Perg | Tel.: +43 (0) 7262 53926

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von mir angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.
Diese Versicherung umfasst auch in der Arbeit verwendete bildliche Darstellungen, Tabellen, Skizzen und Zeichnungen.
Für die Erstellung der Arbeit habe ich die generativen KI-Tools ChatGPT und Claude zu folgendem Zweck verwendet: zur Rechtschreib- und Grammatikprüfung, zur sprachlichen und stilistischen Überarbeitung der selbst verfassten Rohtexte, zur Rechercheunterstützung sowie zur Unterstützung bei der Erstellung des LaTeX-Dokuments.


Unterzeichner Gregor Carlo Scheuchenstuhl
Datum und Uhrzeit 25.03.2026, 19:15 (GMT+01:00)
ID Austria
Dieses Dokument ist digital signiert
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("EIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.
Informationen zur Prüfung der elektronischen Signatur finden Sie unter: https://www.signaturpruefung.gv.at

Gendererklärung

Im Rahmen dieser Diplomarbeit wird aus Gründen der besseren Lesbarkeit auf die gleichzeitige Verwendung der weiblichen sowie der männlichen Form verzichtet. Alle Personenbezeichnungen beziehen sich gleichermaßen auf alle Geschlechter. Diese Vereinfachung dient ausschließlich der Lesbarkeit und soll in keiner Weise Diskriminierung oder Benachteiligung bestimmter Geschlechter implizieren.

Danksagung

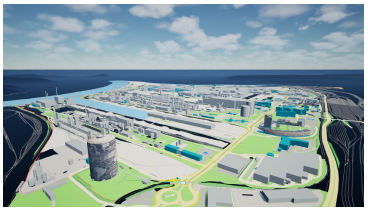
An dieser Stelle möchte ich mich herzlich bei allen Personen bedanken, die mich bei der Erstellung dieser Diplomarbeit unterstützt haben.

Ein besonderer Dank gilt Herrn Markus Wall von der voestalpine Stahl GmbH, Abteilung TSI, der mir die Möglichkeit gegeben hat, dieses Projekt durchzuführen und der mir stets mit fachlichem Rat zur Seite stand.

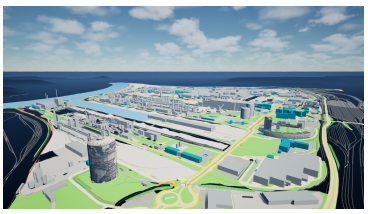
Ebenso möchte ich meinem Betreuer Herrn Prof. Roland Eggetsberger für die wertvolle Begleitung und Unterstützung während der gesamten Projektlaufzeit herzlich danken.

Abstract

The voestalpine Stahl GmbH operates one of the largest integrated steel plants in Europe at its Linz site. An interactive, three-dimensional visualisation solution for navigating the extensive works site was previously unavailable. The aim of this thesis was the development of DISC – Digital Industrial Steel Campus – an interactive 3D visualisation application based on Unreal Engine 5.6 and the ArcGIS Maps SDK for Unreal Engine. The application represents the voestalpine works site as a navigable 3D environment, integrating real GIS data consisting of a vector base map, a digital elevation model and georeferenced 3D building models. The implemented features include free 3D navigation via mouse, keyboard and gamepad, a dynamic layer panel for toggling GIS layers, a geo-bookmark system for predefined viewpoints, a real-time overview map and a structured main menu with help screens. All mandatory requirements defined by voestalpine were fully implemented. Beyond these, several optional features such as layer transparency control and a mini-map were additionally realised.



Kurzfassung



Die voestalpine Stahl GmbH betreibt am Standort Linz eines der größten integrierten Stahlwerke Europas. Eine interaktive, dreidimensionale Lösung zur Visualisierung und Navigation des weitläufigen Werksgeländes war bisher nicht vorhanden. Ziel dieser Diplomarbeit war die Entwicklung des Systems DISC – Digital Industrial Steel Campus – einer interaktiven 3D-Visualisierungsanwendung auf Basis der Unreal Engine 5.6 und des ArcGIS Maps SDK for Unreal Engine. Die Anwendung bildet das Werksgelände der voestalpine als begehbare 3D-Umgebung ab und integriert reale GIS-Daten bestehend aus einer Vektor-Basiskarte, einem digitalen Höhenmodell sowie georeferenzierten 3D-Gebäudemodellen. Die umgesetzten Funktionen umfassen eine freie 3D-Navigation per Maus, Tastatur und Gamepad, ein dynamisches Layerpanel, ein Geo-Bookmark-System, eine Echtzeit-Übersichtskarte sowie ein strukturiertes Hauptmenü mit Hilfesystem. Alle von der voestalpine definierten Muss-Funktionalitäten wurden vollständig umgesetzt. Darüber hinaus wurden mehrere optionale Funktionen wie die Transparenzsteuerung einzelner Layer und eine Echtzeit-Übersichtskarte realisiert.

Inhaltsverzeichnis

1 Einleitung

1.1 Projektbeschreibung

DISC – Digital Industrial Steel Campus – ist eine interaktive 3D-Visualisierungsanwendung für das Werksgelände der voestalpine Stahl GmbH in Linz. Die Anwendung wurde im Rahmen der Diplomarbeit an der HTL Perg, Höhere Abteilung für Informatik, als Einzelprojekt entwickelt.

Die Grundlage bilden reale GIS-Daten der voestalpine, bestehend aus einer Vektor-Basiskarte, einem digitalen Höhenmodell sowie mehreren georeferenzierten 3D-Gebäudemodellen. Diese Daten werden mithilfe des ArcGIS Maps SDK for Unreal Engine in eine lokale 3D-Szene integriert und ermöglichen eine freie, interaktive Navigation durch das Werksgelände.

1.2 Ausgangslage

Die voestalpine Stahl GmbH betreibt am Standort Linz eines der größten integrierten Stahlwerke Europas [1]. Das weitläufige Werksgelände umfasst zahlreiche Gebäude, Produktionsanlagen und Infrastruktureinrichtungen.

Für die interne Orientierung, Planung und Präsentation des Geländes wurden bisher hauptsächlich zweidimensionale Karten und klassische GIS-Werkzeuge eingesetzt. Diese Werkzeuge erlauben zwar eine grundlegende räumliche Darstellung, bieten jedoch keine Möglichkeit, das Gelände dreidimensional zu erkunden oder einzelne Gebäudemodelle interaktiv ein- und auszublenden. Eine interaktive, dreidimensionale Darstellung, die georeferenzierte Werksdaten in Echtzeit navigierbar macht, war nicht vorhanden.

Die Abteilung TSI (Technology & System Integration) der voestalpine Stahl GmbH beauftragte daher die Entwicklung eines solchen Systems im Rahmen dieser Diplomarbeit. Als Ansprechpartner auf Seiten der voestalpine fungierte Markus Wall (markus.wall@voestalpine.com).

1.3 Zielsetzung

Ziel dieser Diplomarbeit war die Entwicklung eines interaktiven, dreidimensionalen GIS-Visualisierungssystems auf Basis der Unreal Engine 5.6. Im Mittelpunkt stand die Kombination aus Geoinformationssystemen (GIS), georeferenzierten 3D-Modellen, interaktiver Benutzersteuerung sowie einer intuitiven Benutzeroberfläche.

Die konkreten Anforderungen wurden von der voestalpine in einem Anforderungsdokument definiert und in verpflichtende Muss-Funktionalitäten sowie optionale Kann-Funktionalitäten unterteilt.

1.3.1 Muss-Funktionalitäten

Folgende Funktionalitäten wurden von der voestalpine als verpflichtend definiert. Sie bilden den Kern der Anwendung und mussten vollständig umgesetzt werden:

- Einbinden der Grundkarte als Vectortile-Basiskarte (.vtpk)
- Einbinden des Elevation-Modells zur realistischen Geländedarstellung (.tpkx)
- Einbinden der 3D-Gebäudemodelle laut einer von der voestalpine bereitgestellten Exceltabelle (.slpk)
- Steuerung mit Maus und Tastatur für die freie 3D-Navigation
- Steuerung mit Gamepad als alternative Eingabemethode
- Auswahlmenü beim Start mit Moduswahl zwischen den Steuerungsarten
- Rückkehr ins Hauptmenü per ESC-Taste aus dem 3D-Level
- Layerpanel zum Ein- und Ausschalten einzelner GIS-Layer

1.3.2 Kann-Funktionalitäten

Folgende Funktionalitäten wurden von der voestalpine als optional definiert. Im Projektverlauf konnten die meisten davon umgesetzt werden:

- Help Screens mit Tastenbelegung für jeden Steuerungsmodus
- Transparenzsteuerung einzelner Layer über einen Slider im Layerpanel
- Geo-Bookmark-System zum schnellen Anspringen vordefinierter Kamerapositionen
- Echtzeit-Übersichtskarte (Mini-Map) mit Positionsanzeige und Blickrichtung

- Kamera-Geschwindigkeitssteuerung über ein Speedpanel
- Teilweise implementierter VR-Modus mit separatem Level und VR-Pawn

Zusätzlich zu den im Anforderungsdokument definierten Punkten wurden eigenständig weitere Funktionen umgesetzt:

- Dynamisches Laden der GIS-Daten über Blueprints mit relativen Pfaden, um die Anwendung auf beliebigen Systemen lauffähig zu machen
- Kontextmenü als zentrale Navigationsschnittstelle
- Gamepad-Bedienung der UI-Panels im 3D-Level (Layerpanel, Bookmarks, Speedpanel)

Das Laden der Layer aus einer externen CSV-Datei und der Thirdperson-Modus wurden nicht umgesetzt.

1.4 Überblick der Anwendung

Beim Start der Anwendung wird dem Benutzer zunächst das Hauptmenü angezeigt (Abbildung 1). Von dort aus kann zwischen den Steuerungsmodi Mouse & Keyboard, Gamepad und Virtual Reality gewählt werden. Jeder Modus zeigt vor dem Einstieg in die 3D-Welt einen eigenen Hilfescreen an, der die jeweilige Tastenbelegung erklärt (Abbildung 2).

Nach der Auswahl wird das 3D-Level geladen, in dem der Benutzer das Werksgelände frei erkunden kann. Über Tastenkürzel oder das Kontextmenü lassen sich während der Navigation weitere Funktionen wie das Layerpanel, die Bookmarks oder die Mini-Map aufrufen.

Abbildung 1: Hauptmenü der DISC-Anwendung

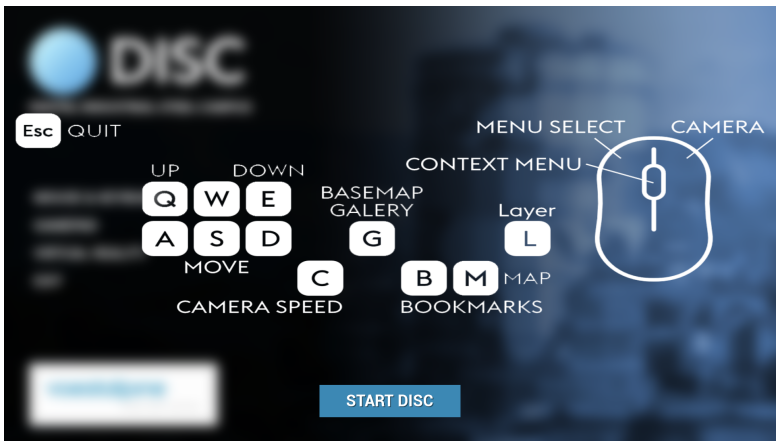


Abbildung 2: Hilfescreen für Maus- und Tastatursteuerung

1.5 Abgrenzung

Der Schwerpunkt der Arbeit lag auf der technischen Implementierung der GIS-Integration, der Systemarchitektur sowie der Entwicklung eines stabilen Steuerungs- und UI-Systems. Folgende Punkte waren bewusst nicht Bestandteil der Arbeit:

- Erstellung eigener 3D-Modelle – sämtliche Modelle wurden von der voestalpine als fertige Scene Layer Packages bereitgestellt
- Bearbeitung der ursprünglichen GIS-Rohdaten – die Daten wurden unverändert übernommen und ausschließlich in die Unreal-Engine-Szene integriert
- Produktiver Einsatz im Unternehmensumfeld – die Anwendung ist ein Prototyp und wurde nicht für den laufenden Betrieb ausgelegt

Der VR-Modus wurde begonnen und grundlegende Strukturen wie ein eigenes VR-Level und ein VR-Pawn wurden erstellt. Aufgrund technischer Einschränkungen beim Input-Rebinding und dem Neuladen der ArcGIS Shader konnte der VR-Modus jedoch nicht vollständig finalisiert werden.

2 Technologien und Grundlagen

2.1 Unreal Engine 5.6

Die Unreal Engine ist eine Grafik- und Entwicklungsplattform von Epic Games, die erstmals 1998 veröffentlicht wurde. Ursprünglich für die Unreal-Spieleserie entwickelt, wird sie heute weit über den Spielebereich hinaus eingesetzt – unter anderem in der Film- und Fernsehindustrie, in Architekturvisualisierungen und in industriellen Anwendungen [2].

Für das Projekt DISC fiel die Wahl auf die Unreal Engine 5.6, da sie mehrere Eigenschaften vereint, die für eine GIS-basierte 3D-Visualisierung notwendig sind. Die Engine bietet leistungsfähiges Echtzeit-Rendering, das auch bei komplexen 3D-Szenen mit vielen Gebäudemodellen eine flüssige Darstellung ermöglicht. Durch die native Unterstützung für C++ konnte die Kernlogik der Anwendung performant und typsicher implementiert werden. Gleichzeitig erlaubt das Blueprint-System eine visuelle Programmierung, die sich besonders für UI-Prototyping und die schnelle Umsetzung interaktiver Elemente eignet. Die Unterstützung externer Plugins war essenziell, da das gesamte GIS-System über das ArcGIS Maps SDK als Plugin eingebunden wurde. Zudem verfügt die Engine über eine integrierte VR-Unterstützung, die für den teilweise umgesetzten VR-Modus genutzt wurde.

2.1.1 C++ und Blueprints

Das Projekt kombiniert zwei Entwicklungsansätze, die in der Unreal Engine parallel eingesetzt werden können. C++ wurde für alle Bereiche verwendet, in denen Performanz, Typsicherheit und direkte Kontrolle über das System wichtig waren. Konkret umfasst das die Klasse `AFlyPawn` für die gesamte Steuerungslogik, die Input-Verarbeitung über das Enhanced Input System, die Levelwechsel zwischen Hauptmenü und 3D-Welt sowie das zentrale UI-Management, das dafür sorgt, dass nie mehrere Fenster gleichzeitig geöffnet sind.

Blueprints kamen überall dort zum Einsatz, wo visuelle Gestaltung und schnelle Iteration im Vordergrund standen. Das betrifft das Layout sämtlicher Widget Blueprints wie `WBP_LayerPanel`, `WBP_Bookmarks` und `WBP_MainMenu`, die dynamischen Listen für Layer und Bookmarks sowie die visuelle Gestaltung der Benutzeroberfläche.

Diese Trennung hat sich im Projektverlauf als sinnvoll erwiesen. Änderungen an der Oberfläche – etwa das Hinzufügen eines neuen Buttons oder das Anpassen eines Layouts – waren ohne Neukompilierung möglich. Gleichzeitig blieb die Kernlogik stabil und unabhängig von visuellen Anpassungen.

2.1.2 Enhanced Input System

Für die Eingabeverarbeitung wurde das Enhanced Input System der Unreal Engine verwendet, das seit Version 5.0 das ältere Input System ablöst [3]. Im Vergleich zum klassischen System bietet es eine saubere Trennung zwischen der Definition einer Aktion und der konkreten Tastenbelegung.

Eine *Input Action* beschreibt dabei nur, *was* passieren soll – zum Beispiel "Vorwärts bewegen" oder "Layerpanel öffnen". Ein *Mapping Context* legt dann fest, *welche Taste* diese Aktion auslöst. Im Projekt wurden zwei Mapping Contexts angelegt: `IMC_KeyboardMouse` für Maus und Tastatur sowie `IMC_Gamepad` für die Gamepad-Steuerung. Beim Start der Anwendung wird abhängig vom gewählten Steuerungsmodus der passende Mapping Context geladen. Dadurch konnte dieselbe Spiellogik mit völlig unterschiedlichen Eingabegeräten genutzt werden, ohne den Code der Aktionen selbst ändern zu müssen.

2.2 ArcGIS Maps SDK for Unreal Engine

Das ArcGIS Maps SDK for Unreal Engine ist ein von Esri bereitgestelltes Plugin, das die direkte Integration von geografischen Daten in eine Unreal Engine-Szene ermöglicht [4]. Es bildet die zentrale Schnittstelle zwischen den GIS-Daten der voestalpine und der 3D-Darstellung in der Anwendung.

Über das Plugin können verschiedene Typen von GIS-Daten eingebunden werden:

- **Vector Tile Layer** – vektorbasierte Basiskarten im Format `.vtpk`, die das Grundlayout des Geländes darstellen
- **Tiled Elevation Layer** – Höhenmodelle im Format `.tpkx`, die die Geländeoberfläche dreidimensional abbilden
- **3D Object Scene Layer** – einzelne georeferenzierte 3D-Objekte im Format `.slpk`, etwa Brücken oder Industrieanlagen

- **Building Scene Layer** – hierarchisch strukturierte Gebäudedaten mit internen Sub-Layern

Das Plugin sorgt dafür, dass alle Daten anhand ihres Koordinatensystems automatisch korrekt positioniert werden, sofern die Spatial Reference im ArcGIS Map Actor richtig konfiguriert ist. Ohne diese korrekte Konfiguration würden die einzelnen Layer gegeneinander verschoben dargestellt werden – ein Problem, das während der Entwicklung mehrfach auftrat und in Kapitel 4 näher beschrieben wird.

2.3 GIS-Grundlagen

2.3.1 Koordinatensysteme

Für die korrekte Positionierung der GIS-Daten in einer 3D-Szene muss ein passendes Koordinatensystem definiert werden. Koordinatensysteme legen fest, wie geografische Positionen auf der Erdoberfläche in numerische Werte umgerechnet werden.

Die von der voestalpine bereitgestellten Daten lagen im österreichischen Landeskoordinatensystem *MGI / Austria GK Central*, das über die sogenannte WKID (Well Known ID) **31255** eindeutig identifiziert wird [5]. In diesem System werden Koordinaten als Rechtswert (X), Hochwert (Y) und Höhe (Z) in Metern angegeben. Die Verwendung dieses spezifischen Systems war notwendig, da die voestalpine ihre Geodaten intern in diesem Format pflegt und eine Umrechnung in ein anderes System zu Ungenauigkeiten bei der Positionierung der Gebäudemodelle geführt hätte.

2.3.2 GIS-Begriffe

Im Kontext dieses Projekts werden mehrere GIS-spezifische Begriffe verwendet, die an dieser Stelle kurz erläutert werden:

Layer Eine Repräsentation von Geodaten, die in der Szene ein- und ausgeblendet werden kann. Jeder Layer entspricht einem bestimmten Datensatz, beispielsweise einem einzelnen Gebäudemodell oder der Basiskarte.

Lokale Szene Eine 3D-Karte mit ebener Basisfläche, die sich für kleinflächige Darstellungen eignet. Im Gegensatz zur globalen Szene, deren Basisfläche eine Kugel ist, wurde für das voestalpine-Werksgelände eine lokale Szene gewählt, da das Gelände nur wenige Quadratkilometer umfasst.

Elevationsmodell Ein Höhenmodell, das die Geländeoberfläche definiert. Es wird als Rasterdatei gespeichert, bei der jeder Pixel statt eines Farbwerts einen Höhenwert enthält. Der Bezug erfolgt nach GHA (Gebrauchshöhen Adria) zum Meeresspiegel der Adria am Pegel Triest.

VTPK Vector Tile Package – ein Dateiformat für offline nutzbare vektorbasierte Kartendaten, das im Projekt für die Basiskarte verwendet wurde.

SLPK Scene Layer Package – ein Dateiformat für georeferenzierte 3D-Modelle, das für sämtliche Gebäude- und Infrastrukturmodelle zum Einsatz kam.

2.3.3 Layer-Typen

Im Projekt wurden zwei Typen von Scene Layern verwendet, deren Unterscheidung für die korrekte Darstellung der Modelle entscheidend war.

Der *3D Object Scene Layer* eignet sich für einzelne, freistehende Objekte wie Brücken, Gebäude und Industrieelemente. Jedes Objekt wird als eigenständiges 3D-Modell geladen und kann unabhängig ein- und ausgeblendet werden.

Der *Building Scene Layer* hingegen enthält hierarchisch strukturierte Gebäudedaten mit internen Sub-Layern. Das bedeutet, dass ein einzelner Layer mehrere Unterebenen enthalten kann – beispielsweise Stockwerke oder Gebäudeteile. Dieser Typ wurde für komplexere Gebäudestrukturen verwendet.

Die korrekte Auswahl des Layer-Typs war essenziell. Während der Entwicklung kam es vor, dass Modelle beim falschen Typ nicht dargestellt wurden oder Fehlermeldungen im Output Log der Unreal Engine erschienen. Die Identifikation des richtigen Typs erforderte teilweise manuelles Testen, da die Dokumentation der bereitgestellten Daten nicht immer eindeutig war.

2.4 Entwicklungsumgebung

2.4.1 Visual Studio 2022

Für die C++-Entwicklung wurde Visual Studio 2022 als IDE eingesetzt. Die Unreal Engine erfordert bestimmte Komponenten, die über den Visual Studio Installer installiert werden müssen: die Workload *Desktopentwicklung mit C++*, den MSVC v143 Compiler sowie das Windows 10/11 SDK.

Ohne diese Komponenten kann die Unreal Engine kein C++-Projekt kompilieren. Nach der Installation wurde das Unreal-Projekt automatisch mit Visual Studio verknüpft, sodass Änderungen am C++-Code direkt über Visual Studio gebaut werden konnten. Die Kompilierung erfolgte standardmäßig über Rechtsklick auf das Projekt `voestUE` und *Build* mit der Konfiguration *Development Editor* auf der Plattform *Win64*.

2.4.2 Projektstruktur

Das Unreal-Projekt wurde modular aufgebaut, um eine klare Trennung zwischen Logik, Benutzeroberfläche, Eingabeverarbeitung und Datenintegration zu gewährleisten. Im Content-Verzeichnis wurden folgende Hauptbereiche definiert:

- `Blueprints` – sämtliche Widget Blueprints für UI-Elemente wie Hauptmenü, Layerpanel und Bookmarks
- `Input` – die definierten Input Actions und die beiden Mapping Contexts für Tastatur und Gamepad
- `GIS_Data` – die eingebundenen geografischen Datengrundlagen (VTPK, TPKX und SLPK-Dateien)
- `Level` – die drei Level der Anwendung: `MainMenu` für das Startmenü, `voestMap` für die 3D-Welt und `voestMap_VR` für den VR-Modus

Die C++-Logik befindet sich im Verzeichnis `Source/voestUE/` und umfasst unter anderem die Klassen `AFlyPawn` für die Steuerungslogik, `UMainMenuWidget` für die Menüsteuerung sowie die verschiedenen GameMode-Klassen, die festlegen, welcher Pawn und welche UI in welchem Level verwendet werden.

3 Projektplanung

3.1 Projektumfeld

3.1.1 Projektteam

Das Projekt war ursprünglich als Zweier-Projekt angelegt. Nach dem Ausscheiden des zweiten Teammitglieds im Herbst 2025 wurde es als Einzelprojekt fortgeführt. Sämtliche Aufgabenbereiche – von der Projektplanung über die technische Implementierung bis zur Dokumentation – wurden von Gregor Scheuchenstuhl alleine übernommen.

3.1.2 Auftraggeber

Auftraggeber des Projekts war die voestalpine Stahl GmbH, konkret die Abteilung TSI (Technology & System Integration) am Standort Linz. Als Ansprechpartner fungierte Markus Wall, der die Anforderungen definierte, die GIS-Daten bereitstellte und während der gesamten Projektlaufzeit für fachliche Rückfragen zur Verfügung stand.

3.1.3 Betreuung

Die schulische Betreuung der Diplomarbeit übernahm Prof. Roland Eggetsberger an der HTL Perg.

3.2 Projektorganisation

Die Zusammenarbeit mit der voestalpine erfolgte über regelmäßige Abstimmungstermine. Nahezu jeden Montag fand ein Online-Meeting über Microsoft Teams mit Markus Wall statt, in dem der aktuelle Projektfortschritt besprochen, offene Fragen geklärt und die nächsten Arbeitsschritte abgestimmt wurden.

Ein dediziertes Projektmanagement-Tool wurde nicht eingesetzt. Die Aufgabenplanung erfolgte auf Basis der im Anforderungsdokument definierten Muss- und Kann-Funktionalitäten, die als

Leitfaden für die Priorisierung der Arbeitspakete dienten. Durch die wöchentlichen Meetings war eine kontinuierliche Rückkopplung mit dem Auftraggeber sichergestellt.

3.3 Projektstrukturplan

Der Projektstrukturplan (Abbildung 3) zeigt die hierarchische Gliederung des Projekts in seine wesentlichen Arbeitspakete. Die Struktur orientiert sich an den im Anforderungsdokument der voestalpine definierten Funktionalitäten.

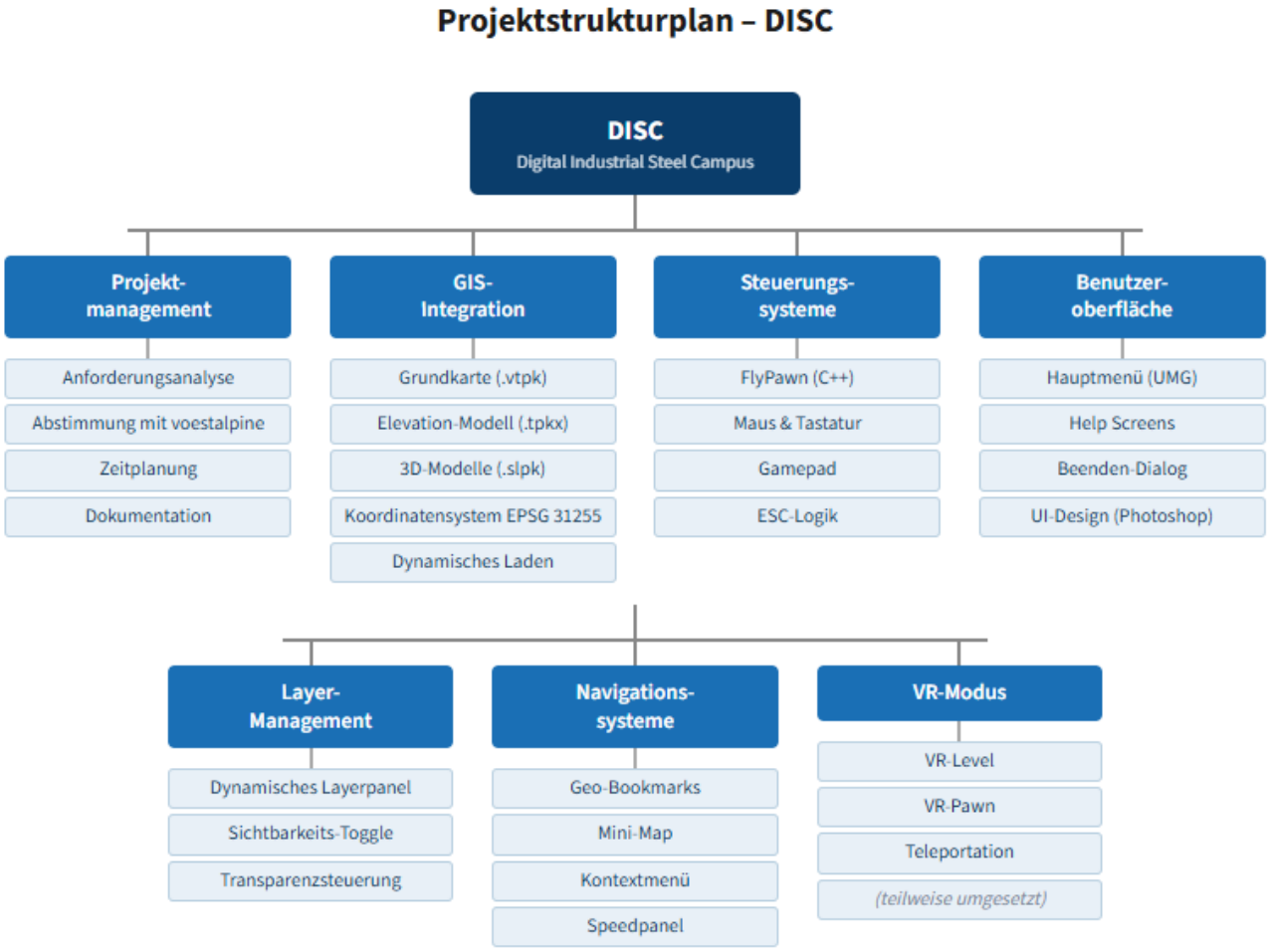


Abbildung 3: Projektstrukturplan von DISC

3.4 Zeitplan

Der Projektzeitraum erstreckte sich von August 2025 bis März 2026. Tabelle 1 zeigt die wesentlichen Projektphasen und deren zeitliche Zuordnung. Die Meilensteine orientierten sich an den Muss-Funktionalitäten des Anforderungsdokuments, wobei die Kann-Funktionalitäten gegen Ende der Entwicklungsphase umgesetzt wurden.

Zeitraum	**Meilenstein**
August 2025	Projektstart, Einrichtung der Entwicklungsumgebung (UE 5.6, ArcGIS Plugin, VS 2022), Einbinden der Grundkarte und des Elevation-Modells
September 2025	Einbinden der 3D-Gebäudemodelle, Konfiguration des Koordinatensystems (EPSG 31255), Umstellung auf dynamisches Laden über Blueprints
Oktober 2025	FlyPawn-Implementierung, Steuerungssysteme (Maus/Tastatur und Gamepad), Hauptmenü mit Moduswahl
Nov. – Dez. 2025	VR-Modus: Erstellung des VR-Levels, VR-Pawn mit Motion-Controller-Komponenten, Teleportationsansatz, Input-Versuche
Jänner 2026	Abschluss VR-Modus (Teilimplementierung), ESC-Logik, Help Screens, Beenden-Dialog, Überarbeitung des UI-Designs
Februar 2026	Layerpanel mit Transparenzsteuerung, Bookmark-System, Mini-Map, Kontextmenü, Speedpanel, Gamepad-UI-Integration, Testing, Abschluss der technischen Entwicklung
März 2026	Dokumentation und Fertigstellung der Diplomarbeit

Tabelle 1: Projektzeitplan

Eine besondere Herausforderung im Zeitplan war der VR-Modus, der über einen Zeitraum von rund zwei Monaten bearbeitet wurde. Trotz erheblichem Aufwand konnte er aufgrund technischer Einschränkungen beim Input-Rebinding und dem Neuladen der ArcGIS-Shader nicht vollständig fertiggestellt werden (siehe Abschnitt 5.2).

Die verbleibenden Kann-Funktionalitäten – darunter das Layerpanel, das Bookmark-System, die Mini-Map und das Kontextmenü – wurden in einer intensiven Arbeitsphase während der Semesterferien im Februar 2026 umgesetzt.

4 Implementierung

4.1 Entwicklungsumgebung einrichten

4.1.1 Installation Unreal Engine

Die Entwicklung erfolgte mit Unreal Engine 5.6, die über den Epic Games Launcher installiert wurde. Nach der Installation wurde ein neues Projekt auf Basis des *Blank*-Templates erstellt, da keine vorgefertigte Spiellogik benötigt wurde. Als Projekttyp wurde zunächst Blueprint gewählt, weil sich damit schnell erste UI-Prototypen bauen ließen. C++ wurde erst im nächsten Schritt ergänzt, als klar war, welche Teile der Anwendung eine performantere Implementierung erforderten.

4.1.2 Integration des ArcGIS Plugins

Das ArcGIS Maps SDK for Unreal Engine wurde manuell in das Projekt eingebunden. Dafür wurde die Engine zunächst geschlossen, der entpackte Plugin-Ordner `ArcGISMapsSDK` in das Verzeichnis `UnrealProjekt\Plugins\` kopiert und das Projekt anschließend wieder geöffnet. Die Engine erkennt Plugins in diesem Ordner automatisch und kompiliert sie beim Start mit. Nach diesem Vorgang stand das Plugin im Editor zur Verfügung und der `ArcGIS Maps SDK Mode` konnte in der Toolbar aufgerufen werden.

4.1.3 Visual Studio und C++ Komponenten

Für die C++-Entwicklung wurde Visual Studio 2022 installiert. Im Visual Studio Installer wurden drei Komponenten hinzugefügt, die die Unreal Engine zwingend benötigt: die Workload *Desktopentwicklung mit C++*, der MSVC v143 Compiler sowie das Windows 10/11 SDK. Ohne diese Komponenten lässt sich kein Unreal-C++-Projekt kompilieren.

Die Kompilierung erfolgte standardmäßig über einen Rechtsklick auf das Projekt `voestUE` und *Build* mit der Konfiguration *Development Editor* auf der Plattform *Win64*. Nach dem Hinzufügen der ersten C++-Klasse wurde das Projekt automatisch zu einem hybriden Blueprint/C++-Projekt erweitert, was regelmäßige Kompilierung über Visual Studio notwendig machte.

4.1.4 Live Coding Problematik

Zu Beginn der Entwicklung wurde das Unreal-Feature *Live Coding* verwendet, das C++-Änderungen während der Laufzeit des Editors kompilieren kann. Der Vorteil liegt darin, dass der Editor nicht geschlossen und das Projekt nicht vollständig neu gebaut werden muss. In der Praxis führte Live Coding jedoch nach der Integration des ArcGIS Plugins zu erheblichen Problemen. Der Editor stürzte regelmäßig ab, Modulreferenzen gingen nach dem Hot-Reload verloren und die ArcGIS-Shader wurden fehlerhaft dargestellt. Nach mehreren solcher Vorfälle wurde Live Coding vollständig deaktiviert und alle Änderungen ausschließlich über den regulären Build-Prozess in Visual Studio kompiliert. Das kostete zwar bei jeder Änderung etwas mehr Zeit, brachte aber eine wesentlich stabilere Entwicklungsumgebung.

4.2 GIS-Datenintegration

4.2.1 Koordinatensystem

Bevor die eigentlichen GIS-Daten eingebunden werden konnten, musste zunächst das Koordinatensystem korrekt konfiguriert werden. Im *Details Panel* des `ArcGIS Map Actors` wurde dazu *Enable Manual Selection* aktiviert und im Feld *Horizontal Coordinate System* der Wert **31255** eingetragen. Dieser Wert entspricht dem österreichischen Landeskoordinatensystem *MGI / Austria GK Central* (EPSG: 31255), in dem die voestalpine ihre Geodaten pflegt. Zusätzlich wurde der *Map Type* auf *Local* gesetzt, da das Werksgelände nur wenige Quadratkilometer umfasst und eine ebene Basisfläche ausreicht.

Ohne diese Konfiguration hätten die einzelnen Layer nicht korrekt übereinandergelegen. In einem frühen Entwicklungsstand, als die Spatial Reference noch auf dem Standardwert stand, erschienen die 3D-Modelle um mehrere hundert Meter verschoben – ein Problem, das erst durch manuelles Testen verschiedener EPSG-Codes gelöst werden konnte.

4.2.2 Einbinden der Basiskarte

Die Basiskarte wurde von der voestalpine als Vector Tile Package (`.vtpk`) bereitgestellt und lokal im Projektverzeichnis unter `GIS_Data` abgelegt. In der ersten Implementierungsphase wurde sie über den *ArcGIS Maps SDK Mode* eingebunden: Unter dem Reiter *Basemap* wurde der Typ *Vector Tile Layer* ausgewählt und der absolute Dateipfad zur `.vtpk`-Datei angegeben. Die Basiskarte bildet die zweidimensionale Grundlage der Szene und zeigt Straßen, Wege und Gebäudeumrisse auf dem Werksgelände.

4.2.3 Einbinden des Elevation-Modells

Zur dreidimensionalen Darstellung des Geländes wurde ein Elevation-Modell als Tile Package (`.tpkx`) eingebunden. Dieses Höhenmodell definiert die Geländeoberfläche und sorgt dafür, dass Gebäude nicht auf einer flachen Ebene stehen, sondern dem realen Höhenverlauf des Werksgeländes folgen. Im *ArcGIS Maps SDK Mode* wurde unter dem Reiter *Elevation* eine neue Elevation Source vom Typ *ArcGIS Image Elevation Source* hinzugefügt und der Dateipfad zur `.tpkx`-Datei angegeben.

4.2.4 Einbinden der 3D-Modelle

Die 3D-Gebäudemodelle lagen als Scene Layer Packages (`.slpk`) vor. Die voestalpine stellte eine Excel-Tabelle bereit, in der jedes Modell mit seinem Dateinamen und dem zugehörigen Layer-Typ aufgelistet war. Je nach Datentyp musste zwischen zwei Layer-Typen unterschieden werden: *3D Object Scene Layer* für einzelne freistehende Objekte wie Brücken, Waggons und Industrieelemente, sowie *Building Scene Layer* für komplexere Gebäudestrukturen mit internen Sub-Layern.

Die korrekte Unterscheidung war in der Praxis nicht immer offensichtlich. Bei einigen Modellen funktionierte ausschließlich der Building Scene Layer, obwohl die Dokumentation keinen Hinweis darauf gab. Eine fehlerhafte Typauswahl äußerte sich darin, dass das Modell entweder überhaupt nicht in der Szene erschien oder Fehlermeldungen im *Output Log* der Unreal Engine ausgegeben wurden. Die Identifikation des richtigen Typs erforderte deshalb bei mehreren Modellen manuelles Durchprobieren beider Optionen.

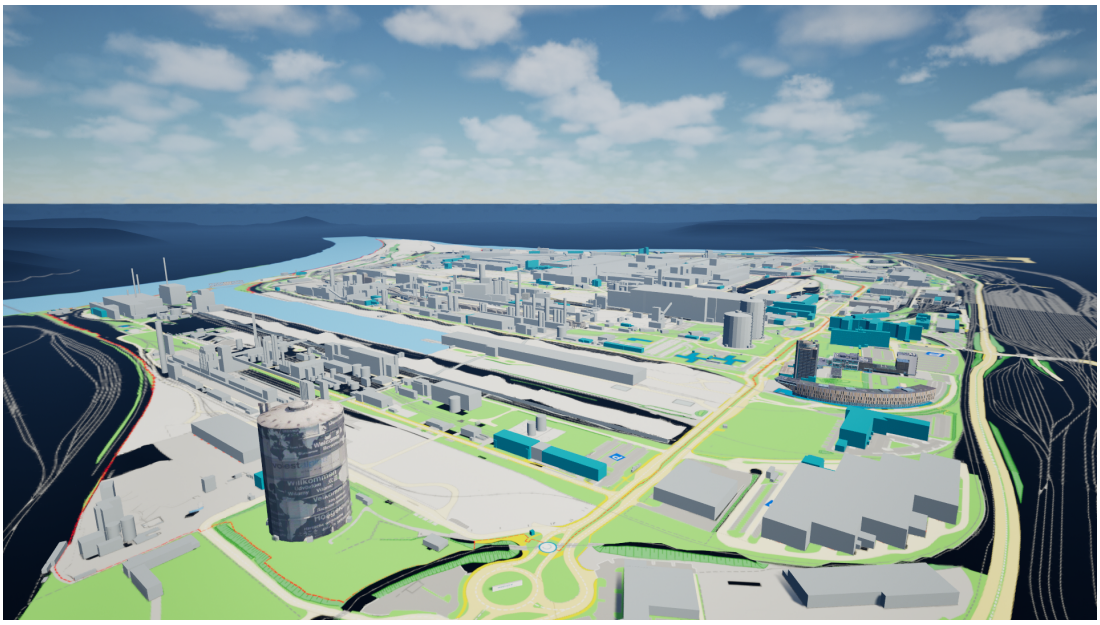


Abbildung 4: 3D-Ansicht des voestalpine Werksgeländes mit integrierten GIS-Daten

4.2.5 Dynamisches Laden über Blueprints

In der ersten Implementierungsphase wurden sämtliche GIS-Daten mit absoluten Dateipfaden eingebunden – also mit Pfaden wie `C:\Users\Gregor\...`. Das funktionierte auf dem Entwicklungsrechner problemlos, führte aber dazu, dass die Anwendung auf jedem anderen System die Layer nicht finden konnte. Beim Erstellen einer ausführbaren Datei (Packaging) waren die GIS-Daten schlicht nicht auffindbar.

Um die Anwendung portabel zu machen, wurde die gesamte Layer-Integration auf Blueprint-Basis umgestellt. Im Level Blueprint von `voestMap` wird beim *Event BeginPlay* ein *Sequence*-Node verwendet, der für jeden Layer nacheinander folgende Schritte ausführt:

1. *Get Project Content Directory* ermittelt den Projektpfad zur Laufzeit
2. Ein *Append*-Node hängt den relativen Pfad zur jeweiligen Datei an (z.B. `GIS_Data/Beschilderung.slpk`)
3. Abhängig vom Datentyp wird entweder *Create Arc GIS 3D Object Scene Layer* oder *Create Arc GIS Vector Tile Layer* aufgerufen
4. Der erstellte Layer wird über *Get Map* und *Get Layers* zur `ArcGISLayerCollection` hinzugefügt

Für das Elevation-Modell wurde ein analoger Ablauf implementiert, wobei stattdessen *Create Arc GIS Image Elevation Source* verwendet und die Source über *Set Elevation* dem Map Actor zugewiesen wurde. Durch diese Umstellung funktioniert die Anwendung auf beliebigen Windows-Systemen, solange die GIS-Daten im Content-Verzeichnis des Projekts liegen.

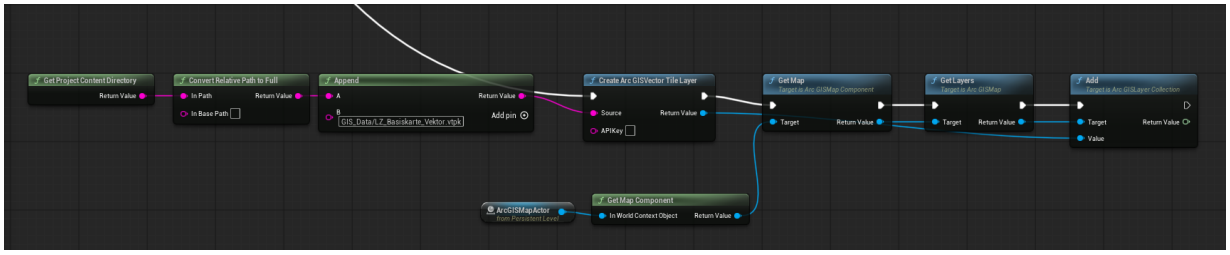


Abbildung 5: Level Blueprint: Dynamisches Laden der Basiskarte über relativen Pfad

4.3 Steuerungssysteme

4.3.1 FlyPawn

Die Steuerungslogik der gesamten Anwendung ist zentral in der C++-Klasse `AFlyPawn` implementiert. Der FlyPawn stellt die Bewegungseinheit dar, über die der Benutzer die 3D-Welt erkundet.

Er fungiert gleichzeitig als zentrale Steuerinstanz für Bewegungslogik, Eingabeverarbeitung und UI-Zustandsmanagement.

Der Pawn basiert auf drei Komponenten, die im Konstruktor erstellt werden:

- `USceneComponent` als Root-Komponente
- `UCameraComponent` für die freie 3D-Kamera, die als Child an die Root-Komponente angehängt wird
- `UFloatingPawnMovement` für die Bewegungslogik, die eine reibungslose Flugbewegung ohne Schwerkraft ermöglicht

Im Konstruktor werden außerdem zwei Eigenschaften gesetzt, die für die Kamerasteuerung wichtig sind:

Listing 1: Konstruktor-Konfiguration des FlyPawn

```
1 bUseControllerRotationYaw = true;
2 bUseControllerRotationPitch = true;
```

Durch diese beiden Zeilen wird die Rotation des Player Controllers direkt auf den Pawn übertragen. Das bedeutet, dass Mausbewegungen oder Gamepad-Stick-Eingaben die Blickrichtung der Kamera unmittelbar verändern, ohne dass eine eigene Rotationslogik geschrieben werden muss.

Die eigentliche Bewegung erfolgt vektororientiert. Für jede Bewegungsrichtung wird der entsprechende Richtungsvektor des Actors mit dem Eingabewert multipliziert und als Bewegungsimpuls an die `UFloatingPawnMovement`-Komponente übergeben:

Listing 2: Vektororientierte Bewegungslogik

```
1 // Vorwaerts/Rueckwaerts
2 AddMovementInput(GetActorForwardVector(), Value);
3 // Seitwaerts
4 AddMovementInput(GetActorRightVector(), Value);
5 // Vertikal (Hoch/Runter)
6 AddMovementInput(FVector::UpVector, Value);
```

Durch diesen Ansatz ist die Bewegung vollständig richtungsunabhängig: Egal in welche Richtung die Kamera blickt, die Bewegung erfolgt immer relativ zur aktuellen Blickrichtung. Das entspricht einem klassischen Fly-Modus, wie er in 3D-Editoren üblich ist, und eignet sich gut für die Exploration eines Werksgeländes aus der Vogelperspektive.

4.3.2 Maus- und Tastatursteuerung

Für die Eingabeverarbeitung wurde das Enhanced Input System verwendet. Im Projekt wurden insgesamt elf Input Actions definiert, die verschiedene Aspekte der Steuerung abdecken:

- `IA_MoveForward` und `IA_MoveRight` für die horizontale Bewegung (Tasten W/A/S/D)
- `IA_UpDown` für die vertikale Bewegung (Tasten Q und E)
- `IA_Look` für die Kameradrehung über die Maus-Achsen
- `IA_LookEnable` als Gate-Mechanismus, der an die rechte Maustaste gebunden ist
- `IA_Speed` für die Kamera-Geschwindigkeitsanpassung (Taste C)
- `IA_Esc` für die Rückkehr ins Hauptmenü
- `IA_ToggleLayers`, `IA_Bookmarks`, `IA_Map` und `IA_ContextMenu` für die jeweiligen UI-Elemente

Sämtliche Actions sind im Mapping Context `IMC_KeyboardMouse` zusammengefasst. Eine Besonderheit ist die Kameradrehung: Sie wird nur verarbeitet, wenn gleichzeitig die rechte Maustaste gehalten wird. Dafür arbeiten `IA_LookEnable` und `IA_Look` zusammen. Erst wenn `IA_LookEnable` den Wert `true` liefert, werden die Maus-Achsenwerte aus `IA_Look` an die Controller-Rotation weitergegeben. Dieser Mechanismus verhindert, dass sich die Kamera dreht, wenn der Benutzer eigentlich nur mit der Maus auf UI-Elemente klicken möchte.

4.3.3 Gamepad-Steuerung

Neben Maus und Tastatur wurde eine vollständige Gamepad-Unterstützung implementiert. Dafür existiert ein separater Mapping Context `IMC_Gamepad`, der dieselben Input Actions wie der Tastatur-Context verwendet, sie aber auf Gamepad-Buttons und -Sticks mappt. Die Umschaltung zwischen den beiden Contexts erfolgt dynamisch über die `GameInstance`.

Die `GameInstance` ist eine Klasse der Unreal Engine, die über den gesamten Lebenszyklus der Anwendung bestehen bleibt – auch bei Levelwechseln. Sie wurde genutzt, um das gewählte Steuerungsschema (Keyboard oder Gamepad) zu speichern. Beim Spawn des FlyPawns in der `BeginPlay`-Funktion werden zunächst alle bestehenden Mappings entfernt und anschließend abhängig vom gespeicherten Steuerungsschema entweder `IMC_Gamepad` oder `IMC_KeyboardMouse` geladen. Dadurch kann der Benutzer im Hauptmenü seinen bevorzugten Modus wählen, und diese Wahl bleibt auch nach dem Wechsel in die 3D-Welt erhalten.

Ein zusätzliches Problem bei der Gamepad-Steuerung betraf die UI-Navigation. Während die Maus automatisch den Fokus auf Buttons und Slider setzen kann, muss bei Gamepad-Eingabe der Fokus explizit an das jeweilige Widget übergeben werden. Beim Öffnen eines UI-Elements wie dem Layerpanel oder den Bookmarks musste daher ein *Set User Focus* auf den ersten interaktiven Button des Widgets ausgeführt werden, damit die Gamepad-Navigation funktionierte.

4.3.4 ESC-Logik

Die ESC-Taste erfüllt eine kontextabhängige Funktion. Befindet sich der Benutzer im 3D-Level, wird über `UGameplayStatics::GetCurrentLevelName()` zunächst geprüft, ob das aktuelle Level nicht bereits das Hauptmenü ist. Ist das Level ungleich `MainMenu`, wird per `UGameplayStatics::OpenLevel()` das Hauptmenü geladen. Vor dem Levelwechsel wird der *InputMode* auf *UIOnly* gesetzt und der Mauszeiger aktiviert, damit der Benutzer im Hauptmenü sofort mit der Maus navigieren kann. Im Hauptmenü selbst hat die ESC-Taste bewusst keine Funktion zugewiesen.

4.4 Benutzeroberfläche

4.4.1 Hauptmenü

Das Hauptmenü wurde mit dem Unreal Motion Graphics (UMG) System als Fullscreen-Widget realisiert. Das zentrale Widget `WBP_MainMenu` steuert alle Aspekte des Startbildschirms: die Auswahl der Steuerungsart, die Anzeige der Hilfescreens, den Start des 3D-Levels und das Beenden der Anwendung.

Die Widget-Hierarchie ist in mehrere logisch getrennte Bereiche unterteilt. Das `Panel_MainMenu` enthält eine `VerticalBox` mit den Buttons für die Steuerungsauswahl (Mouse & Keyboard, Gamepad, VR, Exit). Dahinter liegt ein `BackgroundVoest`-Element mit einem industriellen Hintergrundbild und Gradient-Overlay. Separate Elemente für `Blur_Overlay`, `QuitDialog` und die Help-Images können über `SetVisibility` unabhängig voneinander ein- und ausgeblendet werden.

Das Menü ist bewusst minimalistisch gehalten. Dunkle Farbtöne und wenige UI-Elemente sollen den industriellen Charakter der Anwendung widerspiegeln.

4.4.2 Trennung von C++ und Blueprint

Ein zentrales Architekturprinzip war die konsequente Trennung zwischen Systemlogik in C++ und UI-Logik in Blueprint. C++ übernimmt die Steuerungsarchitektur im FlyPawn, die Levelwechsel sowie das ControlScheme-Management über die `GameInstance`. Blueprint übernimmt die Button-Events, Sichtbarkeitsumschaltungen, die Overlay-Logik und die Dialogverwaltung.

Wenn der Benutzer im Hauptmenü auf eine Steuerungsart klickt, laufen im Blueprint folgende Schritte ab:

1. Ein Cast zur `VoestGameInstance` wird ausgeführt
2. Über `SetControlScheme` wird der gewählte Modus gespeichert (z.B. *KeyboardMouse* oder *Gamepad*)
3. Der zugehörige Help-Screen wird sichtbar geschaltet und das `Blur_Help`-Overlay eingeblendet
4. Ein Start-Button wird aktiviert, über den der Benutzer das 3D-Level laden kann

Der eigentliche Levelwechsel passiert erst beim Klick auf "Start DISC". Diese Trennung stellt sicher, dass der Benutzer vor dem Einstieg in die 3D-Welt die Steuerungsanleitung sehen kann.

4.4.3 Help Screens

Für jede Steuerungsart wurde ein eigener Help-Screen erstellt, der die jeweilige Tastenbelegung visuell darstellt. Die drei Screens (`Img_HelpMouse`, `Img_HelpGamepad`, `Img_HelpVR`) sind als Bilder direkt im `WBP_MainMenu` eingebettet und werden über Sichtbarkeitssteuerung aktiviert. Das hat den Vorteil, dass kein zusätzliches Widget geladen werden muss und die Performance-Kosten minimal bleiben.

4.4.4 Beenden-Dialog

Beim Klick auf EXIT wird kein sofortiger Programmabbruch ausgeführt. Stattdessen wird zunächst das `Blur_Overlay` sichtbar geschaltet, das den Hintergrund abdunkelt und visuell in den Hintergrund rückt. Darüber erscheint der `QuitDialog` mit dem Text "Do you really want to leave the application?" und den Buttons *Yes* und *No*. Ein Klick auf *Yes* ruft `QuitGame()` auf, ein Klick auf *No* blendet Overlay und Dialog wieder aus. Diese Implementierung verhindert

unbeabsichtigtes Beenden der Anwendung – gerade bei Präsentationen vor Kunden ein wichtiger Aspekt.

4.5 Layer-Management-System

4.5.1 Dynamisches Layerpanel

Zur interaktiven Steuerung der GIS-Datenebenen wurde ein vollständig dynamisches Layer-Management-System entwickelt. Das Ziel war ein wartungsfreies Panel, das automatisch alle im ArcGIS Map Actor vorhandenen Layer erkennt, ohne dass Layernamen manuell gepflegt werden müssen.

Das System besteht aus zwei Widget Blueprints, die zusammenarbeiten: `WBP_LayerPanel` fungiert als Container und enthält eine `VerticalBox` namens `VB_Layers`. `WBP_LayerRow` repräsentiert eine einzelne Zeile mit dem Layernamen, einem Sichtbarkeits-Toggle und einem Transparenz-Slider.

Beim Öffnen des Panels läuft im *Event Construct* folgende Logik ab (Abbildung 6):

1. *Clear Children* auf `VB_Layers` – das verhindert doppelte Einträge bei erneutem Öffnen
2. *Get All Actors Of Class* holt eine Referenz auf den `ArcGISMapActor`
3. Über *Get Component By Class* wird die `ArcGISMapComponent` abgerufen
4. *Get Map* und *Get Layers* liefern die `ArcGISLayerCollection`
5. *Get Size* ermittelt die Anzahl der Layer
6. Eine *For Loop* von 0 bis Size - 1 iteriert über alle Layer (das -1 ist wichtig, um einen Index-Out-of-Bounds zu vermeiden)
7. Für jeden Index wird über *At(Index)* das konkrete `ArcGISLayer`-Objekt geholt
8. *Create Widget* erstellt eine Instanz von `WBP_LayerRow` und übergibt dabei das Layer-Objekt
9. *Add Child* fügt die neue Zeile zur `VerticalBox` hinzu

Das Ergebnis: Alle Layer erscheinen automatisch im Panel (Abbildung 7). Wird im Unreal Editor oder im Level Blueprint ein neuer Layer hinzugefügt, taucht er beim nächsten Öffnen des Panels sofort auf – ganz ohne Anpassung am Widget-Code.

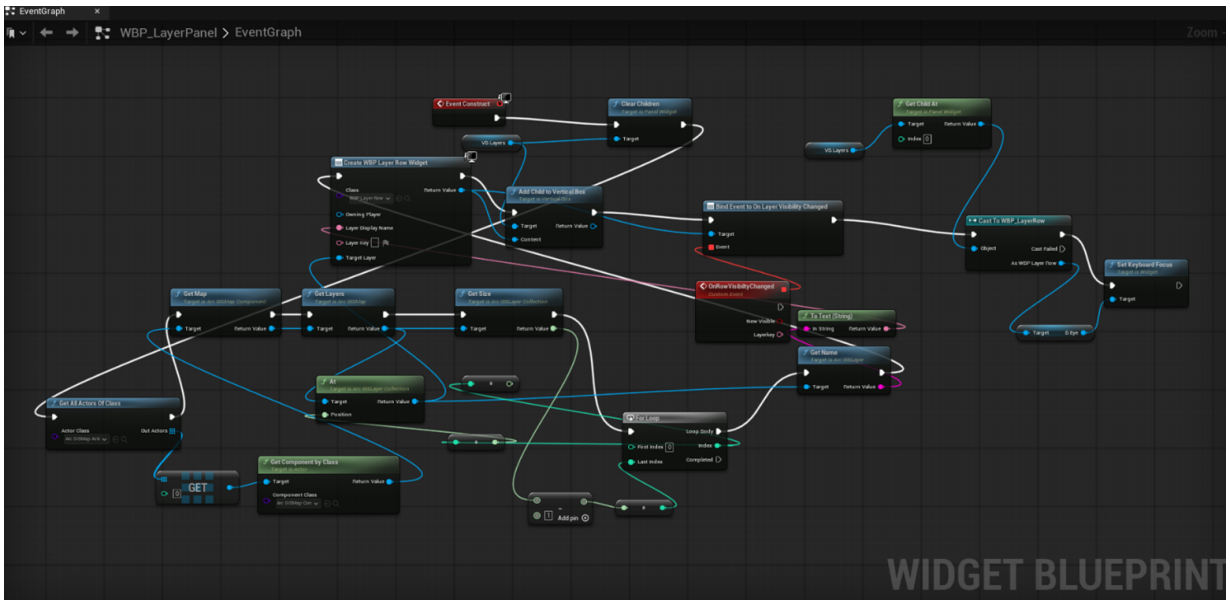


Abbildung 6: Blueprint-Logik des Layerpanels: Automatisches Auslesen und Erstellen der Layer-Zeilen

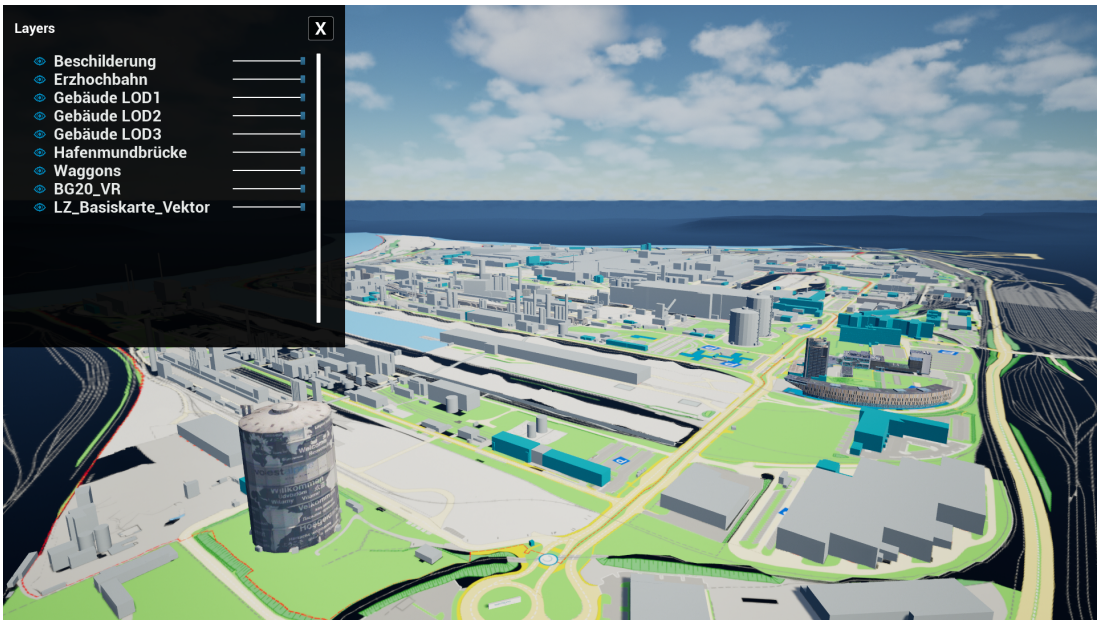


Abbildung 7: Dynamisches Layerpanel mit Transparenzsteuerung

4.5.2 Objekt-Referenz-Injection

Ein zentrales Architekturprinzip des Layerpanels ist die direkte Übergabe von Objekt-Referenzen an die einzelnen Zeilen. In `WBP_LayerRow` existiert die Variable `TargetLayer` vom Typ *ArcGISLayer* mit der Option *Expose on Spawn* aktiviert. Beim Erstellen jeder Zeile wird das konkrete ArcGISLayer-Objekt direkt übergeben.

Das hat eine wichtige Konsequenz: Jede Zeile weiß exakt, welches Layer sie steuert. Es sind keine String-Vergleiche nötig, um den richtigen Layer anhand seines Namens zu finden, und es gibt keinen globalen Zugriff auf eine Layer-Liste. Wenn der Benutzer den Sichtbarkeits-Toggle betätigt, wird *Set Is Visible* direkt auf dem `TargetLayer`-Objekt aufgerufen. Das ist im Grunde objektorientiertes Arbeiten in Blueprint – jede Zeile hält eine Referenz auf ihr Datenobjekt und manipuliert es direkt.

4.5.3 Transparenzsteuerung

Das Layerpanel wurde um eine stufenlose Transparenzsteuerung erweitert, mit der einzelne Layer halbtransparent dargestellt werden können. In `WBP_LayerRow` wurde dazu neben dem Sichtbarkeits-Toggle ein Slider (`slider_transp`) integriert. Um ein sauberes Layout zu erreichen, bei dem alle Slider unabhängig von der Länge des Layernamens exakt auf derselben X-Position beginnen und enden, wurde der Textblock in eine `SizeBox` mit fester *Width Override* gesetzt und der Slider auf *Fill* konfiguriert.

Die Blueprint-Logik hinter dem Slider ist kompakt(Abbildung 8): Beim Event *On Value Changed* wird der Float-Wert des Sliders (zwischen 0.0 und 1.0) direkt an *Set Opacity* auf dem `TargetLayer`-Objekt übergeben. Da die Referenz direkt auf das ArcGISLayer zeigt, wird die Transparenz sofort in der 3D-Szene sichtbar – ohne Verzögerung, ohne Neuladen der Szene und ohne Neuberechnung der Layer-Positionen.

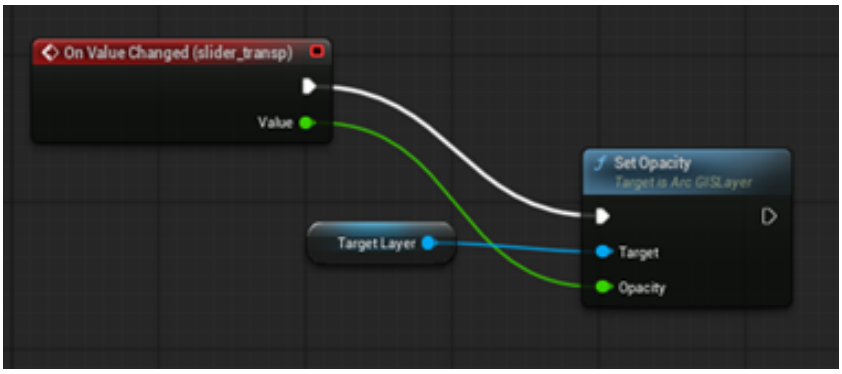


Abbildung 8: Blueprint-Logik: Stufenlose Transparenzsteuerung über Slider und Set Opacity

4.6 Navigationssysteme

4.6.1 Geo-Bookmarks

Das Geo-Bookmark-System ermöglicht es, per Knopfdruck an vordefinierte Positionen im digitalen Zwilling zu springen. Dabei wird nicht nur die Kameraposition (X, Y, Z) wiederhergestellt, sondern auch die Blickrichtung (Rotation), sodass der Benutzer exakt den gleichen Ausschnitt sieht wie beim Erstellen des Bookmarks.

Das System basiert auf zwei Widget Blueprints. `WBP_BookmarkRow` repräsentiert eine einzelne Zeile in der Liste und enthält einen Button (`Btn_Jump`) mit einem Textblock (`Txt_Name`). Die beiden Variablen `BookmarkName` (Typ *Text*) und `TargetTransform` (Typ *Transform*) sind als *Instance Editable* und *Expose on Spawn* markiert. Beim Klick auf den Button wird über *Get Owning Player Pawn* der FlyPawn geholt und `SetActorTransform` mit den gespeicherten Koordinaten aufgerufen – die Kamera springt sofort an den Zielort.

`WBP_Bookmarks` ist das Hauptfenster, das die Liste der Bookmarks dynamisch generiert. Die Daten werden in einer `Map`-Variable namens `MyBookmarks` gespeichert, wobei ein *String* als Key den Namen des Orts enthält (z.B. “Hochofen” oder “Stahlwelt”) und ein *Transform* als Value die Position und Rotation speichert. Beim Öffnen des Fensters werden alle Keys ausgelesen, per *ForEachLoop* iteriert und für jeden Eintrag eine `WBP_BookmarkRow` erstellt.

Das Hinzufügen neuer Bookmarks erfordert keine Codeänderung. Im Unreal Editor fliegt man mit dem Pawn an die gewünschte Stelle, liest die Transform-Werte aus dem Details Panel ab, öffnet `WBP_Bookmarks` und fügt unter `MyBookmarks` ein neues Element mit Name und Transform hinzu. Nach dem Kompilieren ist der neue Ort sofort in der Anwendung verfügbar (Abbildung 9).

Die Blueprint-Logik von `WBP_Bookmarks` (Abbildung 10) zeigt den Ablauf der dynamischen Listenerstellung. In `WBP_BookmarkRow` (Abbildung 11) wird beim Klick auf den Button der Kamerasprung über `SetActorTransform` ausgelöst.

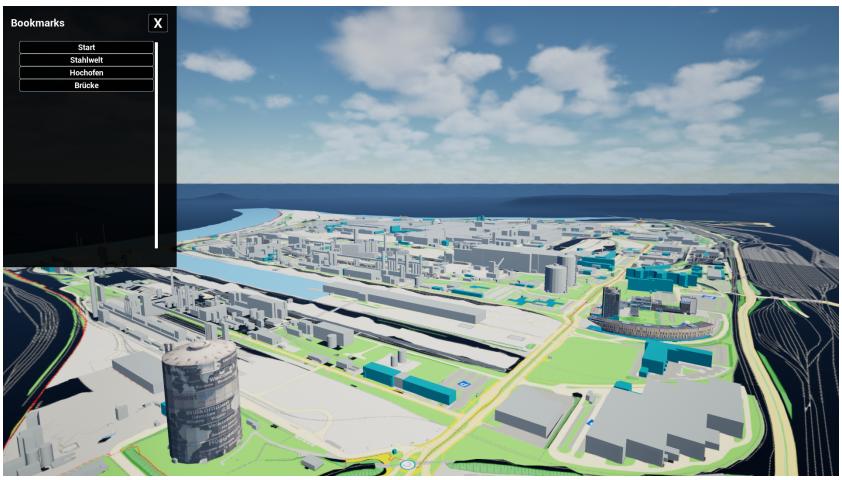


Abbildung 9: Geo-Bookmark-System mit vordefinierten Standpunkten

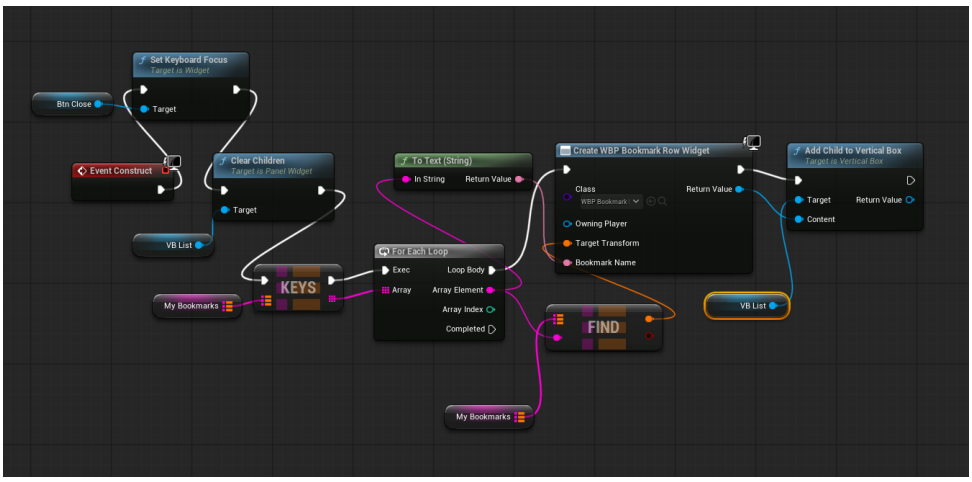


Abbildung 10: Blueprint-Logik von WBP_Bookmarks: Dynamische Listenerstellung

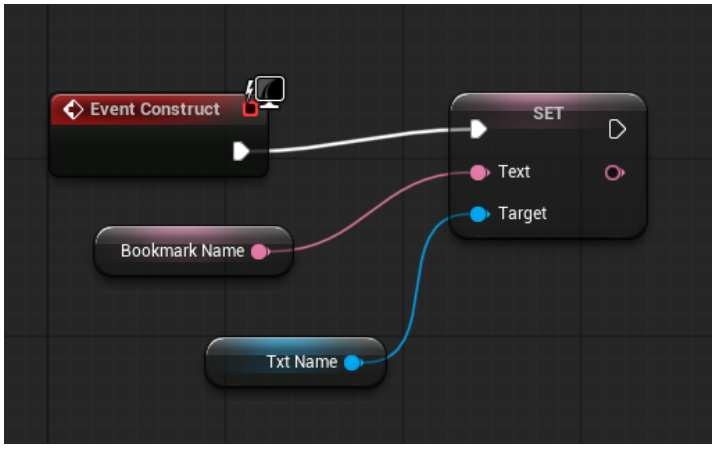


Abbildung 11: Blueprint-Logik von WBP_BookmarkRow: Kamerasprung

4.6.2 Mini-Map

Die Mini-Map stellt dem Benutzer eine Echtzeit-Orientierungshilfe zur Verfügung. Auf einer statischen 2D-Übersichtskarte (`Overview.jpg`, 800×750 Pixel) bewegt sich ein roter Pfeil synchron zur Bewegung des FlyPawns in der 3D-Welt und dreht sich entsprechend der aktuellen Blickrichtung.

Die größte technische Herausforderung bei der Umsetzung war die Transformation der 3D-Weltkoordinaten in 2D-Bildschirmkoordinaten. Da die Unreal Engine ein linkshändiges Koordinatensystem verwendet (X = Nord/Süd, Y = Ost/West), die Bildschirmdarstellung aber Y als vertikale Achse nutzt, mussten die Achsen getauscht werden. Im *Event Tick* des Widget Blueprints `WBP_Map` werden in jedem Frame zwei Berechnungen durchgeführt (Abbildung 12):

Für die Positionszuordnung wird *MapRangeClamped* verwendet, um die Welt-X-Koordinate (Nord/Süd) auf die Widget-Y-Achse und die Welt-Y-Koordinate (Ost/West) auf die Widget-X-Achse zu interpolieren. Die Grenzwerte dafür wurden über ein manuelles Kalibrierungsverfahren ermittelt: Im Editor wurden die vier Ecken des Kartenbildes in der 3D-Szene angeflogen und die exakten World-Locations über `PrintString`-Debugging ausgelesen.

Für die Rotationskorrektur wird der Yaw-Winkel des Pawns mit einem Offset von 90 Grad addiert, da das verwendete Pfeil-Icon im Rohzustand nach rechts zeigt, während der 0-Grad-Winkel in Unreal nach Norden definiert ist. Abbildung 13 zeigt das Ergebnis in der laufenden Anwendung.

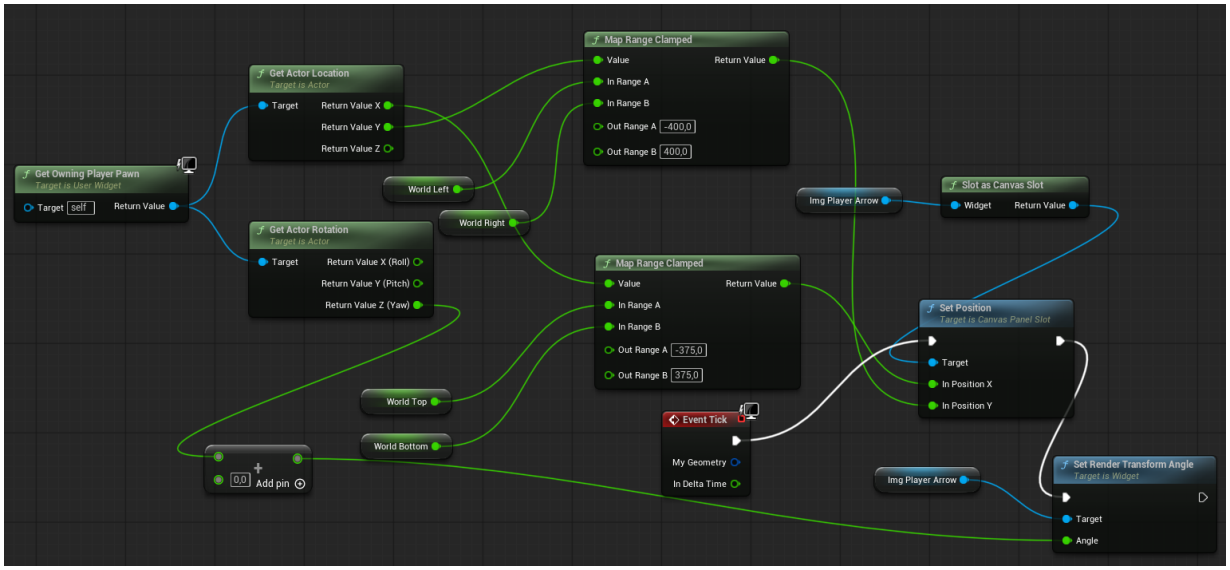


Abbildung 12: Blueprint-Logik der Mini-Map: Koordinatentransformation und Rotationskorrektur im Event Tick

Abbildung 13: Echtzeit-Übersichtskarte (Mini-Map)

4.6.3 Kontextmenü

Das Kontextmenü (Abbildung 14) dient als zentrale Navigationsschnittstelle und wird per mittlere Maustaste oder über den Gamepad aufgerufen. Es ermöglicht den Zugriff auf alle UI-Module der Anwendung: Layerpanel (Taste L), Bookmarks (Taste B), Kamera-Geschwindigkeit (Taste C), Mini-Map (Taste M) sowie das Beenden der Anwendung.

Das Widget `WBP_ContextMenu` ist als *Vertical Box* mit fünf Buttons aufgebaut. Jeder Button führt im Blueprint dieselbe Struktur aus: Über *Get Owning Player Pawn* und einen Cast zu `BP_FlyPawn` wird eine `BlueprintCallable`-Funktion im Pawn aufgerufen, beispielsweise `ToggleLayerPanelUI()`, `ToggleBookmarksUI()` oder `ToggleMapUI()`.

Das Kontextmenü selbst enthält bewusst keine Fensterlogik. Die gesamte Zustandsverwaltung – also welches Fenster gerade offen ist, ob andere Fenster geschlossen werden müssen und ob der InputMode umgeschaltet wird – liegt ausschließlich im C++-Code des FlyPawns. Die Toggle-Funktionen folgen dabei alle demselben Muster:

Listing 3: Exklusive Fensterlogik am Beispiel ToggleBookmarksUI

```
1  void AFlyPawn::ToggleBookmarksUI()
2  {
3      if (BookmarkWidget && BookmarkWidget->IsInViewport())
4      {
5          BookmarkWidget->RemoveFromParent();
6          SetGameInputMode(false);
7      }
8      else
9      {
10         // Alle anderen Fenster schliessen
11         if (LayerPanelWidget && LayerPanelWidget->IsInViewport())
12             LayerPanelWidget->RemoveFromParent();
13         if (SpeedPanelWidget && SpeedPanelWidget->IsInViewport())
```

```
14              SpeedPanelWidget->RemoveFromParent();
15
16          if (!BookmarkWidget)
17              BookmarkWidget = CreateWidget<UUserWidget>(
18                  PC, BookmarkWidgetClass);
19          BookmarkWidget->AddToViewport(40);
20          SetGameInputMode(true);
21      }
22  }
```

Durch diese exklusive Fensterlogik ist sichergestellt, dass nie mehrere Panels gleichzeitig sichtbar sind. Das verhindert UI-Überlagerungen und sorgt für einen konsistenten Zustand der Eingabeverarbeitung.

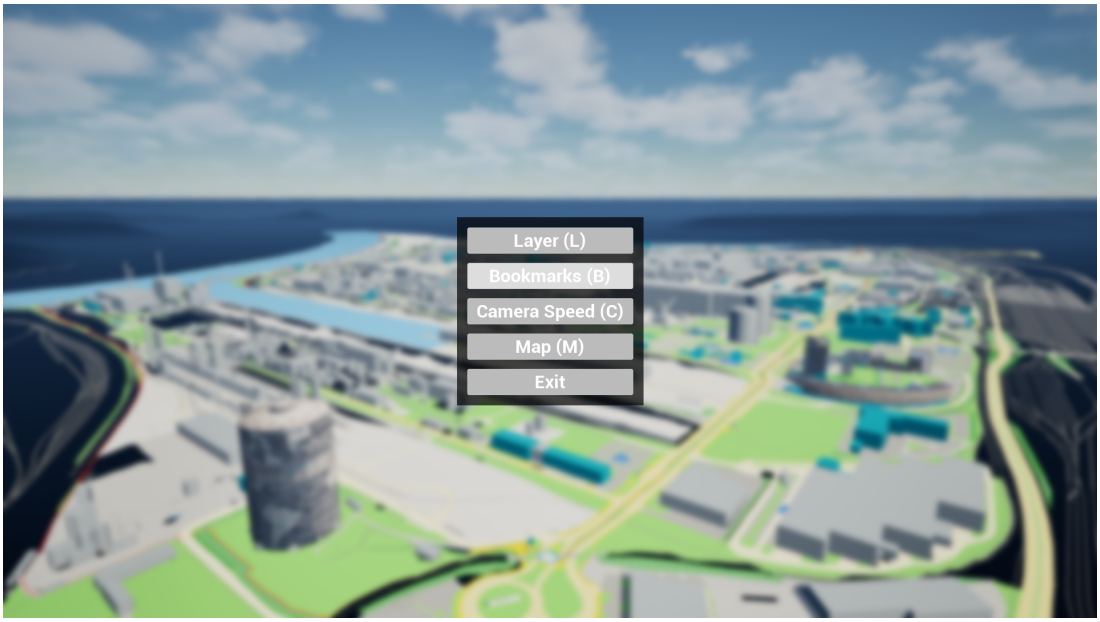


Abbildung 14: Kontextmenü als zentrale Navigationsschnittstelle

4.6.4 Kamera-Geschwindigkeitssteuerung

Um dem Benutzer die Anpassung der Fluggeschwindigkeit zur Laufzeit zu ermöglichen, wurde ein Speedpanel implementiert. Das Widget `WBP_CameraSpeed` enthält einen Slider mit Werten zwischen 0 und 1 sowie eine Min/Max-Beschriftung.

Im Blueprint wird beim *Event Construct* über *Get Owning Player Pawn* und einen Cast zu `BP_FlyPawn` eine Referenz auf den Pawn gespeichert. Bei jeder Wertänderung des Sliders wird die C++-Funktion `SetSpeedRatio()` auf dem Pawn aufgerufen. Im FlyPawn wird die tatsächliche Bewegungsgeschwindigkeit dann als Produkt aus `BaseSpeed` und dem übergebenen `SpeedRatio` berechnet.

Als UX-Detail wechselt die Slider-Farbe dynamisch je nachdem, ob der Slider aktuell den Tastatur-Fokus hat. Dafür wird im *Event Tick* des Widgets *HasKeyboardFocus* abgefragt und die

Farbe entsprechend angepasst – ein visuelles Feedback, das bei Gamepad-Steuerung besonders hilfreich ist.

4.6.5 VR-Modus

Ein VR-Modus wurde im Rahmen des Projekts begonnen, konnte aber nicht vollständig fertiggestellt werden. Umgesetzt wurden ein separates VR-Level (`voestMap_VR`), ein eigener VR-Pawn mit Motion-Controller-Komponenten sowie ein Teleportationsansatz für die Fortbewegung.

Die Probleme, die zur Nicht-Fertigstellung führten, waren technischer Natur. Das Enhanced Input System ließ sich für VR-Controller nicht ohne Weiteres umkonfigurieren, da die bestehenden Mapping Contexts auf Maus/Tastatur und Gamepad ausgelegt waren. Beim Wechsel vom Hauptmenü ins VR-Level ging der Input-Fokus verloren, und die ArcGIS-Shader mussten nach dem Levelwechsel neu kompiliert werden, was zu langen Ladezeiten und gelegentlichen Darstellungsfehlern führte.

Trotz dieser Einschränkungen bildet die erstellte Infrastruktur eine solide Grundlage für eine spätere vollständige VR-Umsetzung. Das separate Level und der VR-Pawn sind funktionsfähig, und die GIS-Daten werden korrekt geladen – lediglich die Eingabeverarbeitung und das Shader-Handling müssten überarbeitet werden.

5 Ergebnis

5.1 Umgesetzte Funktionalitäten

Im Folgenden wird dargestellt, welche der definierten Anforderungen im Rahmen des Projekts umgesetzt wurden. Die Anforderungen waren in verpflichtende Muss-Funktionalitäten und optionale Kann-Funktionalitäten unterteilt, wobei beide Kategorien von der voestalpine in einem Anforderungsdokument definiert wurden.

5.1.1 Muss-Funktionalitäten

Alle acht von der voestalpine definierten Muss-Funktionalitäten wurden vollständig implementiert und im Rahmen von Testläufen auf dem Entwicklungsrechner überprüft. Tabelle 2 gibt einen Überblick über den jeweiligen Status.

Funktionalität	**Status**
Einbinden der Grundkarte (.vtpk)	Umgesetzt
Einbinden des Elevation-Modells (.tpkx)	Umgesetzt
Einbinden der 3D-Modelle (.slpk)	Umgesetzt
Steuerung mit Maus und Tastatur	Umgesetzt
Steuerung mit Gamepad	Umgesetzt
Auswahlmenü beim Start	Umgesetzt
Rückkehr ins Menü per ESC	Umgesetzt
Layerpanel zum Ein-/Ausschalten	Umgesetzt

Tabelle 2: Übersicht der umgesetzten Muss-Funktionalitäten

Die GIS-Daten (Basiskarte, Elevation-Modell und sämtliche 3D-Gebäudemodelle) werden korrekt im Koordinatensystem EPSG 31255 positioniert und können über das Layerpanel einzeln gesteuert werden. Beide Steuerungsmodi – Maus/Tastatur und Gamepad – ermöglichen eine flüssige Navigation durch die 3D-Szene. Das Auswahlmenü beim Start erlaubt die Wahl des Steuerungsmodus und zeigt den zugehörigen Hilfescreen an, bevor die 3D-Welt geladen wird.

5.1.2 Kann-Funktionalitäten

Über die verpflichtenden Anforderungen hinaus wurden sieben der neun im Anforderungsdokument definierten Kann-Funktionalitäten umgesetzt. Zusätzlich wurden drei weitere Funktionen eigenständig ergänzt. Der VR-Modus wurde begonnen, konnte aber aus den in Abschnitt 5.2 beschriebenen Gründen nicht vollständig fertiggestellt werden.

Funktionalität	**Status**
Aus dem Anforderungsdokument:	
Help Screens (Tastenbelegung)	Umgesetzt
Transparenzsteuerung der Layer	Umgesetzt
Geo-Bookmark-System	Umgesetzt
Echtzeit-Übersichtskarte (Mini-Map)	Umgesetzt
Kamera-Geschwindigkeitssteuerung	Umgesetzt
VR-Modus	Teilweise umgesetzt
Laden der Layer aus CSV-Datei	Nicht umgesetzt
Thirdperson-Modus	Nicht umgesetzt
Zusätzlich umgesetzt:	
Dynamisches Laden (relative Pfade)	Umgesetzt
Kontextmenü	Umgesetzt
Gamepad-Bedienung der UI-Panels	Umgesetzt

Tabelle 3: Übersicht der umgesetzten Kann-Funktionalitäten

Das Laden der Layer aus einer externen CSV-Datei und der Thirdperson-Modus wurden nicht realisiert. Stattdessen wurde das dynamische Laden der GIS-Daten über Blueprints mit relativen Pfaden umgesetzt, was die Anwendung von absoluten Dateipfaden unabhängig macht und auf beliebigen Windows-Systemen lauffähig ist.

5.2 Nicht vollständig umgesetzte Punkte

Der VR-Modus wurde begonnen, konnte jedoch aufgrund technischer Einschränkungen nicht vollständig finalisiert werden. Die grundlegende Infrastruktur ist vorhanden: Ein separates VR-Level (`voestMap_VR`) wurde angelegt, ein VR-Pawn mit Motion-Controller-Komponenten erstellt und ein Teleportationsansatz für die Fortbewegung implementiert.

Die Probleme, die zur Nicht-Fertigstellung führten, lagen in drei Bereichen. Das Enhanced Input System ließ sich für VR-Controller nicht ohne Weiteres umkonfigurieren, da die bestehenden Mapping Contexts auf Maus/Tastatur und Gamepad ausgelegt waren. Beim Wechsel vom

Hauptmenü ins VR-Level ging der Input-Fokus verloren. Und die ArcGIS-Shader mussten nach dem Levelwechsel neu kompiliert werden, was zu langen Ladezeiten und gelegentlichen Darstellungsfehlern führte.

Eine vollständige Umsetzung wäre im Rahmen einer Erweiterung möglich, da die Grundstruktur bereits steht und lediglich die Eingabeverarbeitung sowie das Shader-Handling überarbeitet werden müssten.

5.3 Technische Architektur im Überblick

Die finale Systemarchitektur von DISC ist in fünf klar getrennte Ebenen unterteilt, die jeweils eine spezifische Aufgabe übernehmen:

Datenebene Die GIS-Daten und 3D-Modelle der voestalpine bilden die unterste Schicht. Sie liegen als lokale Dateien in den Formaten VTPK, TPKX und SLPK im Content-Verzeichnis des Projekts.

Engine-Ebene Die Unreal Engine 5.6 übernimmt das Echtzeit-Rendering und die Szenenverwaltung. Das ArcGIS Maps SDK ist als Plugin integriert und sorgt für die korrekte Positionierung und Darstellung der GIS-Daten.

Logikebene Die C++-Klassen `AFlyPawn`, die verschiedenen GameModes sowie die `GameInstance` bilden die Steuerungslogik. Hier werden Bewegung, Eingabeverarbeitung, Levelwechsel und die exklusive Fensterlogik verwaltet.

UI-Ebene Sämtliche Widget Blueprints – MainMenu, LayerPanel, Bookmarks, Mini-Map, Speedpanel und Kontextmenü – sind auf dieser Ebene angesiedelt. Sie kommunizieren über `BlueprintCallable`-Funktionen mit der Logikebene.

Eingabeebene Das Enhanced Input System mit seinen separaten Mapping Contexts für Tastatur und Gamepad bildet die oberste Schicht. Es sorgt dafür, dass dieselbe Spiellogik mit unterschiedlichen Eingabegeräten genutzt werden kann.

Diese Schichtarchitektur ermöglicht es, einzelne Ebenen unabhängig voneinander zu erweitern oder anzupassen. Ein neuer Layer kann auf der Datenebene hinzugefügt werden, ohne die Logik- oder UI-Ebene zu verändern. Umgekehrt könnte ein neues UI-Element ergänzt werden, ohne die GIS-Integration zu berühren.

5.4 Erweiterungsmöglichkeiten

Die modulare Architektur von DISC bietet mehrere Ansatzpunkte für zukünftige Erweiterungen:

- **Vollständige Fertigstellung des VR-Modus** – die bestehende Infrastruktur (VR-Level, VR-Pawn) müsste um eine überarbeitete Eingabeverarbeitung für VR-Controller ergänzt werden
- **Externe Layer-Konfiguration** – statt die Layerliste im Level Blueprint fest zu verdrahten, könnte sie aus einer externen CSV- oder JSON-Datei geladen werden, was das Hinzufügen neuer Layer ohne Änderung am Blueprint ermöglichen würde
- **Thirdperson-Modus** – neben dem bestehenden Fly-Modus könnte ein Modus mit steuerbarer Figur implementiert werden, der eine Begehung des Werksgeländes auf Bodenniveau erlaubt
- **Benutzerdefinierte Geo-Bookmarks** – die Bookmark-Daten könnten zur Laufzeit vom Benutzer erstellt und in einer lokalen Datei gespeichert werden, statt sie nur über den Editor pflegen zu können
- **Datenbankanbindung** – die GIS-Daten könnten über eine Datenbankverbindung dynamisch aus einer zentralen Quelle geladen werden, statt lokal im Projektverzeichnis zu liegen

6 Resümee

6.1 Zusammenfassung

Ziel dieser Diplomarbeit war die Entwicklung von DISC – Digital Industrial Steel Campus – einer interaktiven 3D-Visualisierung des voestalpine-Werksgeländes in Linz. Die Anwendung wurde auf Basis der Unreal Engine 5.6 und des ArcGIS Maps SDK for Unreal Engine realisiert und verarbeitet reale GIS-Daten der voestalpine, bestehend aus einer Vektor-Basiskarte, einem Elevation-Modell und mehreren georeferenzierten 3D-Gebäudemodellen.

Sämtliche acht Muss-Funktionalitäten, die von der voestalpine im Anforderungsdokument definiert wurden, konnten vollständig umgesetzt werden. Von den neun optionalen Kann-Funktionalitäten wurden sieben realisiert, darunter die Help Screens, das Layerpanel mit Transparenzsteuerung, das Geo-Bookmark-System und die Echtzeit-Mini-Map. Zusätzlich wurden ein Kontextmenü, das dynamische Laden über relative Pfade und die Gamepad-Bedienung der UI-Panels eigenständig ergänzt. Das Laden der Layerliste aus einer externen CSV-Datei und der Thirdperson-Modus wurden nicht umgesetzt. Der VR-Modus wurde begonnen, konnte aber aus technischen Gründen nicht fertiggestellt werden.

Die technisch anspruchsvollste Aufgabe im gesamten Projekt war die korrekte Integration der GIS-Daten in die Unreal-Engine-Szene. Das Koordinatensystem EPSG 31255 musste manuell konfiguriert werden, die Layer-Typen (3D Object Scene Layer vs. Building Scene Layer) erforderten teilweise manuelles Durchprobieren, und die Umstellung von absoluten auf relative Dateipfade war notwendig, damit die Anwendung auch außerhalb des Entwicklungsrechners funktioniert.

6.2 Technische Herausforderungen

Während der Entwicklung traten mehrere Probleme auf, für die jeweils eine Lösung gefunden werden musste.

Live Coding und ArcGIS Plugin Zu Beginn des Projekts wurde Live Coding genutzt, um C++-Änderungen schneller testen zu können. Nach der Integration des ArcGIS Plugins

kam es jedoch regelmäßig zu Editor-Abstürzen und fehlerhaften Shader-Darstellungen. Die Ursache lag vermutlich in Konflikten zwischen dem Hot-Reload-Mechanismus und den ArcGIS-Modulen. Als Lösung wurde Live Coding vollständig deaktiviert und ausschließlich über den klassischen Build-Prozess in Visual Studio kompiliert.

Absolute Dateipfade In der ersten Phase wurden alle GIS-Daten mit absoluten Pfaden eingebunden. Das fiel erst auf, als die Anwendung auf einem anderen Rechner getestet wurde und keiner der Layer geladen werden konnte. Die Umstellung auf relative Pfade über *Get Project Content Directory* im Level Blueprint war aufwendig, da jeder einzelne Layer umgebaut werden musste, hat aber das Problem dauerhaft gelöst.

Gamepad-Fokus in der UI Bei der Gamepad-Steuerung zeigte sich ein Problem, das bei Maus und Tastatur nicht auftritt: Widgets erhielten nach dem Öffnen keinen automatischen Fokus. Ohne Fokus reagierte die Gamepad-Navigation nicht auf die Buttons im Layerpanel oder den Bookmarks. Die Lösung bestand darin, beim Öffnen jedes UI-Elements explizit *Set User Focus* auf den ersten Button aufzurufen.

Layer-Typ-Identifikation Einige der bereitgestellten 3D-Modelle ließen sich nur als Building Scene Layer korrekt darstellen, nicht als 3D Object Scene Layer. Die Dokumentation der voestalpine gab darüber keinen eindeutigen Aufschluss. In der Praxis mussten bei mehreren Modellen beide Typen ausprobiert werden, bis die Darstellung funktionierte.

6.3 Persönliches Resümee

Vor diesem Projekt hatte ich weder mit der Unreal Engine noch mit GIS-Systemen gearbeitet. Der Einstieg war entsprechend steil – die Unreal Engine ist ein komplexes System mit einer steilen Lernkurve, und das ArcGIS Plugin kam mit einer eigenen Terminologie und eigenen Konzepten, die erst verstanden werden mussten.

Zu Beginn war mein Plan, möglichst alles in C++ zu schreiben. Das hat für den FlyPawn und die zentrale Steuerungslogik auch gut funktioniert. Bei der Umsetzung der Benutzeroberfläche bin ich aber an Grenzen gestoßen – das Layout von Widgets, dynamische Listen und Sichtbarkeitssteuerungen waren in C++ deutlich aufwendiger als nötig. Der Wechsel zu Blueprints für die gesamte UI-Logik war dann ein Wendepunkt. Die visuelle Darstellung der Logik als Knotengraph machte es wesentlich einfacher, Abläufe nachzuvollziehen und Fehler zu finden. Im Rückblick war diese Aufteilung – C++ für die Kernlogik im FlyPawn, Blueprints für alles was mit Oberfläche und Interaktion zu tun hat – die richtige Entscheidung.

Ein Aspekt, den ich unterschätzt hatte, war der Zeitaufwand für die GIS-Datenintegration. Auf dem Papier klingt es einfach: Dateien ins Projekt legen, Koordinatensystem einstellen, Layer einbinden. In der Praxis hat allein die korrekte Konfiguration des Koordinatensystems und das Testen der verschiedenen Layer-Typen mehrere Arbeitstage gekostet.

Das Projekt hat mir gezeigt, dass Game-Engines wie die Unreal Engine weit über Spiele hinaus eingesetzt werden können. Die Vorstellung, ein komplettes Werksgelände dreidimensional begehbar zu machen und dabei echte Geodaten zu verwenden, hat mich von Anfang an motiviert. Die Zusammenarbeit mit der voestalpine und die Arbeit mit realen Unternehmensdaten war eine Erfahrung, die im normalen Schulalltag so nicht möglich wäre.

6.4 Ausblick

DISC bildet in seiner jetzigen Form einen funktionsfähigen Prototypen. Für einen produktiven Einsatz bei der voestalpine wären weitere Schritte notwendig.

Am naheliegendsten wäre die Fertigstellung des VR-Modus. Die Infrastruktur mit eigenem Level und VR-Pawn steht bereits, und die GIS-Daten werden auch im VR-Level korrekt geladen. Was fehlt, ist eine saubere Eingabeverarbeitung für VR-Controller und eine Lösung für das Shader-Neuladen beim Levelwechsel.

Darüber hinaus könnte die Anwendung um eine Datenbankanbindung erweitert werden, sodass die Layerliste nicht mehr im Level Blueprint fest verdrahtet ist, sondern dynamisch aus einer zentralen Quelle geladen wird. Das würde es der voestalpine ermöglichen, neue Modelle hinzuzufügen, ohne das Unreal-Projekt selbst anfassen zu müssen.

Langfristig könnte DISC als Grundlage für einen digitalen Zwilling des Werksgeländes dienen. In Kombination mit Echtzeit-Sensordaten, Wartungsinformationen oder Produktionsdaten wäre eine Anwendung denkbar, die über die reine Visualisierung hinausgeht und als operatives Werkzeug für Planung und Instandhaltung eingesetzt werden könnte.

Literaturverzeichnis

[1] voestalpine Stahl GmbH, „voestalpine Standort Linz," 2024, letzter Zugriff am 15.03.2025. Online verfügbar: https://www.voestalpine.com/stahl/

[2] Epic Games, „Unreal Engine," 2024, letzter Zugriff am 15.03.2025. Online verfügbar: https://www.unrealengine.com

[3] ——, „Enhanced Input System," 2024, letzter Zugriff am 15.03.2025. Online verfügbar: https://dev.epicgames.com/documentation/en-us/unreal-engine/enhanced-input-in-unreal-engine

[4] Esri, „ArcGIS Maps SDK for Unreal Engine," 2024, letzter Zugriff am 15.03.2025. Online verfügbar: https://developers.arcgis.com/unreal-engine/

[5] ——, „Coordinate systems, map projections, and transformations," 2024, letzter Zugriff am 15.03.2025. Online verfügbar: https://developers.arcgis.com/documentation/spatial-references/

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang

A Aufgabenverteilung

Das Projekt DISC wurde als Einzelprojekt durchgeführt. Sämtliche Aufgabenbereiche – von der Projektplanung über die technische Implementierung bis zur Dokumentation – wurden von Gregor Scheuchenstuhl übernommen.