Bachelorarbeit

im Studiengang
B. Sc. Wirtschaftsinformatik

Chancen und Risiken beim Umstieg auf Unity für Pojekt SEE

von

Christoph Pawlowski

Erstprüfer:	Prof. Dr. Rainer Koschke
Zweitprüfer:	Prof. Dr. Andreas Breiter
Eingereicht am:	4. November 2019

Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Bremen, den 4. November 2019

Datum Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Beim Projekt SEE handelt es sich um eine Visualisierungssoftware für Software. Diese basiert auf den Erkenntnissen der Dissertation von Steinbrückner [18] und wurde im Rahmen eines Bachelorprojekts an der Universität Bremen weiterentwickelt.

Mittels dieser Visualisierungssoftware werden Softwareprojekte als sog. Software Cities dargestellt. Es gibt diverse Umsetzungen von solchen Software Cities. Der Ansatz von Steinbrückner heißt Evostreets. Bei diesem Ansatz werden die darzustellenden Bausteine, beispielsweise Klassen, an Straßen positioniert, die einzelne Pakete einer Software darstellen könnten. Die Klassen wiederum werden als metaphorische Häuser dargestellt, wobei beispielsweise die Höhe dieser Häuser eine bestimmte Metrik repräsentieren können, wie die Anzahl der Zeilen in einer Klasse.

Das Projekt SEE ist in der Unreal Engine entwickelt worden. Die Software lässt sich zurzeit ganz klassisch über die Maus und Tastatur bedienen, aber auch als Virtual Reality Anwendung mit der HTC VIVE nutzen.

In dieser Bachelorarbeit geht es um die Evaluation von Unity als potentielle Plattform für weitere Forschungen zu Software Cities an der Universität Bremen. Dabei werden sowohl funktionale als auch nicht funktionale Aspekte von Unity betrachtet.

Die Unreal Engine und Unity sind beides sog. Game Engines, bei denen es sich um Softwarepakete handelt, die zum Erstellen von Echtzeitgrafikanwendungen beispielsweise Videospielen, Simulationen oder zur Visualisierung von Daten genutzt werden. Diese beiden Game Engines unterstützen unter anderem folgende Funktionalitäten: das Rendering bzw. Zeichnen von 3D- und 2D-Objekten, die Simulation von Physik, das Abspielen von Sound. Die Funktionalitäten werden von beiden Game Engines jedoch oft sehr unterschiedlich umgesetzt und stellen daher einen erheblichen Teil der Entscheidungsfindung für die Auswahl einer Game Engine dar. Aber auch der Funktionsumfang unterscheidet sich zwischen den beiden Game Engines.

Online sind viele Vergleiche der beiden Game Engines zu finden, welche teils unterschiedliche Aspekte in den Vordergrund stellen. Es werden einige dieser Vergleiche betrachtet um einen ersten Überblick über die Ähnlichkeiten und Unterschiede der beiden Game Engines zu verschaffen. Mit Hilfe dieser Vergleiche, der Implementierung von Projekt SEE in der Unreal Engine und der Dissertation von Steinbrückner [18], werden Anforderungen an Unity hergeleitet.

Anhand dieser Anforderungen wird ein Prototyp entwickelt, welcher die grundlegenden Funktionen von Projekt SEE umsetzt, um die Umsetzbarkeit in Unity praktisch zu demonstrieren.

Zum Abschluss wird mit Hilfe der beiden Implementierungen die Frage beantwortet, ob für den weiteren Verlauf der Forschungen an Software Cities an der Universität Bremen auf Unity umgestiegen werden sollte.

2 Theoretischer Teil

Bei der Recherche von Vergleichen zwischen der Unreal Engine und Unity sind online sehr viele Informationen dazu vorhanden. Diese unterscheiden sich teilweise enorm in der Qualität und Form. So gibt es beispielsweise sehr subjektive Blogposts, welche eine der beiden Engines total bevorzugt, bis hin zu qualitativ hochwertigen, wissenschaftlichen Bachelorarbeiten, welche die beiden Engines strukturiert miteinander vergleichen.

Es werden nun die besten Vergleiche herangezogen, um ein erstes Meinungsbild zu den beiden Engines zu erhalten. Dabei stellt man fest, dass viele der Vergleiche eine ähnliche Struktur aufweisen, an der sich auch in dieser Bachelorarbeit orientiert wird. Die Struktur selbst orientiert sich an den angebotenen Funktionalitäten der beiden Engines.

Nachfolgend werden die einzelnen Features der beiden Engines miteinander verglichen und die vorhandene bzw. ggf. nicht vorhandene Relevanz jener in dieser Bachelorarbeit erläutert. Folgende Features sind ein Teil dieses Vergleichs:

- Content Pipeline
- Scripting/Programming
- Grafik
- Sprites
- User Interface
- Animation
- Physics
- Profiling
- Virtual/Augmented Reality
- Networking

2.1 Content Pipeline

Die Content Pipeline stellt die Schnittstelle zwischen einer Game Engine und verschiedenen Datenformaten dar. Es ist damit möglich Komponenten, welche mittels anderer Software erstellt wurdem, in eine Engine zu importieren und dann zu nutzen. Dabei unterscheiden sich die Unreal Engine und Unity teils enorm bei den standardmäßig unterstützen Datenformaten.

So unterstützt Unreal von Haus aus beispielsweise nur das Audioformat .wav wobei Unity die meisten gängigen Audioformate unterstützt, darunter sind unter anderem die Formate .wav, .mp3 und .ogg. [6]

Beide Game Engines besitzen die Fähigkeit vom Nutzer erweitert zu werden, um weitere Datenformate verwenden zu können. Diese Funktionalität ist eine Voraussetzung für Projekt SEE, da das verwendete Datenformat .gxl (Graph eXchange Language) von beiden Engines nicht standardmäßig unterstützt wird. Die weiteren Datenformate spielen zu Beginn von Projekt SEE bzw. der Umsetzung von U-SEE keine große Rolle, da es primär um die Darstellung der .gxl Daten geht. Dies könnte sich jedoch im weiteren Projektverlauf durchaus ändern.

2.2 Scripting/Programming

Für die Programmierung bietet die Unreal Engine zwei Systeme an. Zum einen gibt es dort das Visual Scripting namens Blueprint, welches es erlaubt mittels einer einfach gehaltenen grafischen Oberfläche Knoten miteinander zu verbinden, um so einen Graphen zu erstellen, welcher das Verhalten beschreibt. Dieser Graph wird bei dem Kompilieren der Software dann in C++ Klassen übersetzt.

Wie bereits angeschnitten, verwendet Unreal die Programmiersprache C++, welche als sehr performant gilt, jedoch auch sehr Einsteigerunfreundlich ist. Diese Einsteigerunfreundlichkeit kommt mit der Tatsache einher, dass C++ eine sehr maschinennahe Sprache ist.

In Unity gibt es kein Visual Scripting System. Die gesamte Programmierung findet hier mittels der Programmiersprache C# statt. Die Tatsache der Abwesenheit eines Visual Scripting Systems in Unity macht den Einstieg für Personen ohne Programmiererfahrung in Unity schwieriger, da man sich zusätzlich zum Einarbeiten in Unity mit dem Erlernen von Programmieren beschäftigen muss.

Für den weiteren Verlauf von Projekt SEE bzw. dem Nachfolger U-SEE sollte das Fehlen eines Visual Scripting Systems in Unity keine Hürde darstellen, da Personen, die an diesem

Projekt mitarbeiten, fast immer Programmiererfahrungen mitbringen und somit problemlos mit Unity bei der Programmierung klarkommen sollten.

2.3 Grafik

Die grafische Darstellung von 3D-Objekten ist die Kernfunktionalität einer Game Engine. Somit lässt sich mit beiden Softwarepaketen genau dies tun. Es gibt jedoch einige Unterschiede in der optischen Qualität der einzelnen Teilfeatures. Laut P. Gora und K. Leibetseder [6] besitzt Unreal die ausgereifteren Teilfunktionalitäten, wenn es um maximalen Realismus in der Darstellung von virtuellen Objekten geht.

Der Aspekt des Realismus ist für Projekt SEE und seinen Nachfolger nicht von großer Bedeutung, denn es handelt sich dabei um ein funktionales Tool, welches dazu eingesetzt werden soll Software besser zu machen. Es muss dabei nicht realitätsnah sein, um seine Aufgabe zu erfüllen.

2.4 Sprites

Das Sprite System stammt aus den Anfängen der Computerspiele, welches für die Erstellung von 2D-Spielen entwickelt wurde. Es kann somit als Gegenstück zu den heutigen 3D-Engines angesehen werden. Auch heute noch sind 2D-Spiele sehr gefragt, aus diesem Grund besitzen sowohl die Unreal Engine als auch Unity die nötigen Funktionalitäten und Tools, um 2D- Spiele bzw. 2D-Grafikanwendungen zu erstellen.

Auch diese Funktionalität und deren Umfang kann für dieses Projekt vernachlässigt werden, da es sich hierbei um eine dreidimensionale Anwendung handelt. Für die Erstellung einer zweidimensionalen Benutzeroberfläche bieten beide Engines gesonderte Tools und Funktionalitäten, welche im nächsten Abschnitt betrachtet werden.

2.5 User Interface

So ziemlich jede 3D-Anwendung benötigt die Funktionalität unabhängig von den dreidimensionalen Objekten, zusätzliche Informationen auf dem Bildschirm des Nutzers darzustellen. Diese wird als Benutzeroberfläche bzw. User Interface bezeichnet, denn über dieses User Interface können beispielsweise Einstellungen vorgenommen werden.

Beide Engines besitzen die Möglichkeit User Interfaces umzusetzen. Hierbei ist das Blueprint System von Unreal sehr hilfreich, um schnell und unkompliziert ein User Interface zu erstellen.

Die Erstellung eines User Interfaces in Unity ist etwas komplizierter in der Handhabung als es bei Unreal der Fall ist, da die User Interface Elemente Teil von 3D-Objekten sind und somit nicht an einer zentralen Stelle verwaltet werden können, wie es in der Unreal Engine der Fall ist. [17]

Wie bereits in Abschnitt 2.2 erwähnt sollte das Fehlen eines Visual Scripting Systems in Unity kein entscheidendes Kriterium sein, um sich für eine der beiden Engines zu entscheiden, die an Projekt SEE arbeitenden Personen haben alle Programmiererfahrungen.

2.6 Animation

Die Animation von Objekten in Computerspielen ist eine gängige Praxis der Spielproduzenten und somit eine klare Anforderung an Game Engines solche Tools zur Verfügung zu stellen. Diese grundlegende Anforderung wird auch von beiden Game Engines erfüllt.

Wobei Unreal auch hier mehr Möglichkeiten bietet als Unity jedoch zum Nachteil, dass diese Funktionalität nicht so leicht zugänglich ist wie in Unity. [6]

Die Funktionalität für Animationen ist für dieses Projekt zunächst nicht relevant, da es für die reine Funktionalität keine Vorteile bringt, jedoch kann es durchaus im weiteren Projektverlauf interessant werden, wenn es beispielsweise darum gehen soll diese Software kommerziell zu vertreiben und es somit ansprechender zu gestalten.

2.7 Physics

Für die physikalischen Eigenschaften und Interaktionen von Objekten kommt oft eine Physik-Engine zum Einsatz. Hierbei greifen sowohl Unreal als auch Unity auf die quelloffene 3D Physik Engine PhysX von Nvidia zurück.

Im Bereich der dreidimensionalen Physiksimulation in Spielen sind beide Engines gleich auf, da beide auf der gleichen zugrundeliegenden Physik Engine arbeiten. Für die zweidimensionale Physiksimulation verwendet Unity hingegen eine für zweidimensionale Physik optimierte Physik Engine namens Box2D. Auch bei Box2D handelt es sich um eine quelloffene Software.

Genau wie beim Unterpunkt 2.6 Animation, ist die Physiksimulation eine Funktionalität, welche für die reinen Forschungsarbeiten zunächst von geringer Relevanz ist.

2.8 Profiling

Profiling ermöglicht die Performance eines Spiels genauer zu untersuchen um so ggf. Performanceprobleme zu finden und diese zu beseitigen.

Beide Engines stellen einen Profiler bereit, mit dem die Performance von einzelnen Szenen genauer untersucht werden kann. Zuvor war der Unity Profiler nur in der Pro Version von Unity enthalten. Seit Unity 5 von 2015 ist dieser nun bereits in der kostenlosen Version von Unity enthalten. Zusätzlich bietet der Unreal Profiler im Vergleich zu Unity mehr Funktionen. [6]

Bei der Erstellung von Software ist das Finden von Fehlern im Code ein sehr großer Teil. Bei der Suche nach der Ursache von Problemen in Software hilft klassischerweise ein Debugger, mit dem es möglich ist den Programmablauf Schritt für Schritt zu verfolgen. Der Profiler hingegen hilft den Entwicklern dabei Fehler zu erkennen bzw. auf Codeteile einzugrenzen. Es handelt sich also um ein sehr hilfreiches Tool für die Entwicklung und somit auch für die Forschung an Software Cities.

Außerdem kann der Profiler beim Vergleich der Performance von Projekt SEE und seiner Portierung in Unity verwendet werden, um ggf. vorhandene Unterschiede in der Performance zu begründen.

2.9 Virtual/Augmented Reality

Die weitreichenden Forschungen und Entwicklungen in Richtung Virtual und Augmented Reality lassen auch die Hersteller von Game Engines nicht außen vor. Bei Virtual Reality setzt sich der Spieler eine mit Monitoren versehene Brille auf. Bei Augmented Reality wird im Gegensatz zu Virtual Reality versucht die bereits vorhandene Umgebung um digitale Elemente zu ergänzen.

Sowohl Unreal als auch Unity bieten die Möglichkeit an Virtual und Augmented Reality zu nutzen. Für Augmented Reality können in beiden Engines eine Magic Leap oder HoloLens 1/2 verwendet werden. Des Weiteren kann auf die Handheld Augmented Reality Frameworks ARCore und ARKit zurückgegriffen werden. Unreal kann jedoch nur mit ARKit 2 genutzt werden wohingegen Unity ARKit 3 unterstützt. Außerdem ist es mit Unity möglich eine bestehende Mobile App mittels Unity um Augmented Reality zu erweitern, ohne das diese von Grund auf in Unity entwickelt worden sein muss.

Grundsätzlich bieten beide Game Engines eine ähnliche Unterstützung für unterschiedliche Augmented und Virtual Reality Plattformen. Die Unterstützung für die HoloLens 2 ist bei

Unity jedoch ausgereifter, da die Unterstützung der HoloLens 2 in Unreal erst seit Ende Mai 2019 in einer Beta Version veröffentlicht wurde. [26]

Für die weiteren Forschungen an Software Cities ist es geplant die HoloLens 2 zu unterstützen. Somit stellt Unity eine gute Alternative zur jetzt eingesetzten Unreal Engine. Eine Umsetzung von Virtual bzw. Augmented Reality wird in dieser Bachelorarbeit nicht durchgeführt.

2.10 Networking

Viele Spiele nutzen heutzutage das Internet um das Spielen mit mehreren Spielern zu ermöglichen. Auch für diese Anforderung an Spiele bieten Unreal und Unity geeignete Tools, um dies zu vereinfachen.

Beide Game Engines greifen dabei auf das Server-Client-Modell zurück, wobei der Server alle relevanten Änderungen an die Clients verteilt. Auch hierbei ist wieder das Visual Scripting System von Unreal vertreten und ermöglicht es so schnell einen Prototypen zu erstellen. Generell ist es jedoch in beiden Engines möglich die gleichen Ergebnisse zu erreichen.

Für die hier angestrebte Machbarkeitsstudie ist die Verwendung von Netzwerkkommunikation nicht erforderlich, jedoch ist die langfristige Vision, dass es möglich sein soll die erzeugten Software Cities über weite Entfernungen mit mehreren Personen gemeinsam zu betrachten.

2.11 Fazit

Durch reine Recherche ist klar zu erkennen, dass grundsätzlich Spiele und Grafikanwendungen in beiden Engines realisiert werden können. Jedoch setzen beide Game Engines andere Schwerpunkte.

So versucht die Unreal Engine frn mächtigeren Funktionalitätsumfang zu bieten und somit dem Entwickler mehr Möglichkeiten zu schaffen. Dies hat oft eine steilere Lernkurve zur Folge.

Unity auf der anderen Seite versucht die zur Verfügung stehenden Tools einfach zu halten, um einen möglichst leichten Einstieg zu gewähren.

Es sind somit theoretisch alle nötigen Funktionalitäten vorhanden, welche für eine Portierung von Projekt SEE in Unity benötigt werden. Die konkreten Möglichkeiten für einzelne Anforderungen an Unity werden im nächsten Abschnitt im Detail erläutert.

3 Anforderungsanalyse Prototyp

Kommen wir nun zu den konkreten Anforderungen an Unity, welche benötigt werden um Projekt SEE in Unity umzusetzen. Für Projekt SEE können fünf primäre Anforderungen definiert werden. Zum einen muss es möglich sein die mittels der Axivion Suite generierten GXL Dateien einzulesen, um mit diesen Informationen die Software City zu generieren. Die bis dahin nur als Datenstruktur vorhandene Software City muss dynamisch per Software im dreidimensionalem virtuellen Raum gezeichnet werden können. Außerdem soll es möglich sein Änderungen an den Konfigurationsparametern sowohl für die Generierung als auch für das Zeichnen der Software Cities vorgenommen werden können. Des Weiteren sollte es möglich sein Splines zu erstellen und darzustellen, eine tatsächliche Umsetzung dieser Funktionalität wird in dieser Bachelorarbeit jedoch auf Grund des Umfangs nicht angestrebt. Zu guter Letzt möchte man die Möglichkeit haben die Software City mittels einer Virtual Reality Brille zu betrachten.

3.1 Import von GXL Dateien

Die Struktur der in Projekt SEE dargestellten Software Cities basiert auf GXL Dateien. Diese Dateien werden von der Axivion Suite generiert. Sie stellen einen gerichteten Graphen dar, wobei die Knoten beispielsweise Klassen einer Software repräsentieren und die Kanten repräsentieren die Relationen zwischen den Klassen und anderen Elementen des Graphen.

Das GXL Dateiformat basiert dabei auf dem XML Format. Da Unity die Programmiersprache C# verwendet und die C#-Standardbibliothek unterstützt wird, kann das Standard XML Parsing von C# genutzt werden. Für eine leichtere Handhabung der XML Daten im Code wird dabei zusätzlich das .NET Konzept LINQ genutzt. [11]

Bevor jedoch die Standardbibliothek genutzt werden kann, müssen die Daten aus der Datei in das Programm übertragen werden. Hierfür stellt Unity eine Schnittstelle namens ScriptedImporter zur Verfügung. [21] Dies erlaubt die Datei mittels Drag and Drop in den Unity Editor zu schieben, um daraufhin ein Unity Asset daraus zu machen. Daraufhin sollte dieses erzeugte Asset wie für Unity üblich genutzt werden können.

3.2 Konfigurationsparameter für Software Cities

Nach dem nun die benötigten Daten aus den GXL Dateien gelesen wurden und somit in Unity nutzbar sind, wird eine Möglichkeit benötigt Konfigurationsparameter für die Generierung der Software Cities benötigt. In der Unreal Variante von Projekt SEE können diese

Konfigurationsparameter über die Editorbenutzeroberfläche angepasst werden. Hierfür kann man in der Unreal Engine Variablen von C++ Strukturen direkt im Editor anzeigen lassen und Änderungen vornehmen. [5]

Auch in Unity ist es möglich Variablen direkt im Editor anzeigen zu lassen. Hierfür müssen die gewünschten Felder der C# Klassen als öffentlich deklariert werden. Daraufhin werden diese im Unity Editor dargestellt.

Die Parameter für die Konfiguration der Knoten und Kanten basiert in der Unreal Variante auf der Datenstruktur TMap, dabei handelt es sich um eine Schlüssel-Wert-Datenstruktur. In C# heißen solche Datenstrukturen Dictionaries. Diese lassen sich in Unity nicht ohne weiteres im Editor anzeigen, da dieser Datentyp von Unity nicht standardmäßig serialisiert werden kann. Aus diesem Grund wird auf eine von Unity serialisierbare Implementierung gesetzt, welche auf GitHub verfügbar ist. [2]

3.3 Dynamische Generierung von 3D Objekten

Da es sich bei den in GXL Dateien enthaltenden Daten um Graphen handelt, müssen diese in Kombination mit Konfigurationsparametern zu 3D Objekten berechnet werden und danach auch gezeichnet werden. Sowohl Unreal als auch Unity besitzen Schnittstellen zur Berechnung und Repräsentation von 3D Objekten. Diese Schnittstellen unterscheiden sich jedoch zwischen den beiden Engines. Unter anderem unterscheiden sich die beiden Interpretationen der Koordinatensysteme. Die Interpretation der einzelnen positiven Komponenten eines Vektors kann der folgenden Tabelle 1 entnommen werden. [14]

Vektorkomponente	Unreal	Unity
X	vorwärts	rechts
Y	rechts	oben
Z	oben	vorwärts

Tabelle 1: Unterschiede im Koordinatensystem

Aufgrund der unterschiedlichen Koordinatensysteme wird der Anteil zur Generierung der Software City mit Ausnahme des Anteils zum Zeichnen der 3D Objekte im Unreal Koordinatensystem stattfinden. Dies hat den Vorteil, dass zunächst die Implementierungen der Algorithmen zur Positionierung der einzelnen Elemente der Software City aus der Unreal Variante von Projekt SEE weitgehend unverändert übernommen werden können. Die Änderungen an den Algorithmen wären so nur syntaktisch um sie in C# nutzen zu können.

3.4 Virtual Reality

Für Projekt SEE kann in der Unreal Variante eine Virtual Reality Brille dazu genutzt werden sich durch die Software Cities zu bewegen. In Unreal wurde dabei das RuneBerg VR Plugin genutzt.

In Unity ist die Implementierung der Virtual Reality Funktionalität mit Unity eigenen Mitteln möglich. Hier muss lediglich eine Virtual Reality Brille angeschlossen werden und die Unterstützung für Vritual Reality eingeschaltet werden. Daraufhin sollte die Kamera von nun an auf die Bilder für eine Virtual Reality Brille rendern. Zudem sollten darauf hin die Bewegungen der Brille auf die Kamera in Unity übertragen werden.

Der einzige manuelle Programmieraufwand, der unternommen werden muss, ist die Implementierung für die vollständige Bewegung im dreidimensionalem Raum. Denn die Nutzer einer Virtual Reality Brille sind in deren räumlicher Bewegung durch Vorgaben des Raums, in dem sie sich befinden, eingeschränkt. Hierfür werden die Controller der Virtual Reality Brille genutzt.

4 Konzeptionelle Sicht

Um die Algorithmen für die Berechnung der Positionen der einzelnen Elemente der Software City möglichst mit geringen Änderungen übernehmen zu können, setzt dies ähnliche oder gar gleiche Datenstrukturen in C# voraus. Aus diesem Grund werden zunächst die Datenstrukturen von Projekt SEE aus Unreal weitestgehend übernommen. Sämtliche von Projekt SEE genutzten bzw. definierten Datenstrukturen sind dabei in der Header-Datei SCOOPStructs.h enthalten.

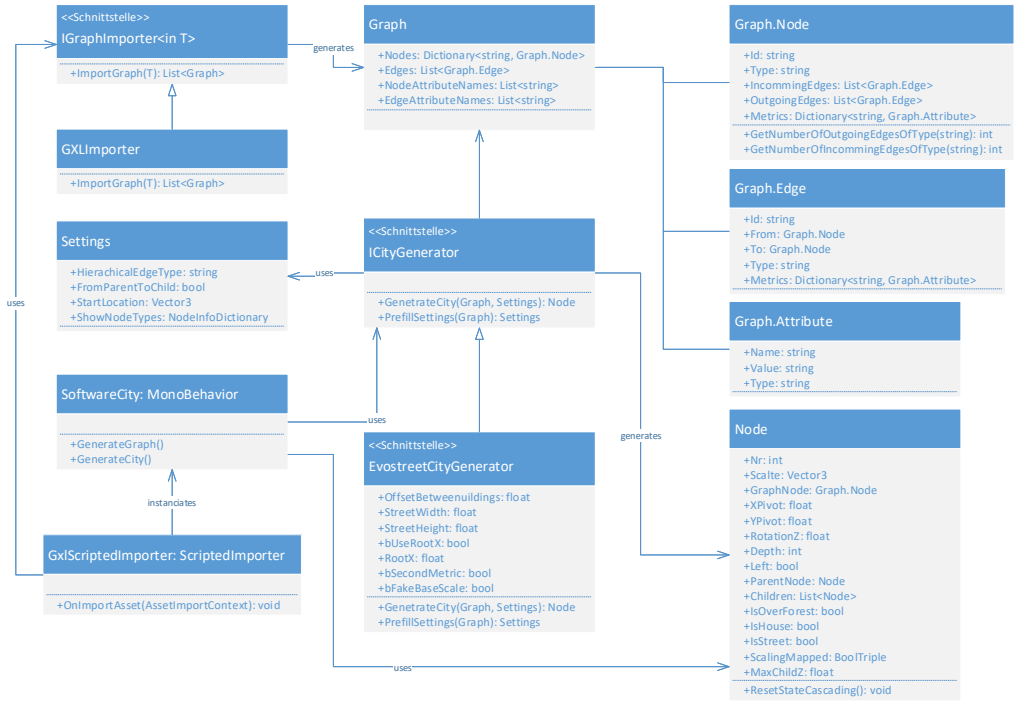

Abbildung 1: Konzeptionelle Sicht

4.1 GXL Import

Die Vorgehensweise für die Erstellung der Konzeptionellen Sicht folgt dabei dem Bottom-Up- Prinzip. Somit wird zu Beginn über den Import der Daten aus den GXL Dateien nachgedacht. Hieraus entstand die Klasse Graph mit sämtlichen inneren Klassen in Unity. Diese stellt die in den GXL Dateien vorhandenen Daten dar und besitzt lediglich Abhängigkeiten zur C#- Standardbibliothek. Außerdem entspricht es weitestgehend der Semantik der Datenstrukturen aus Unreal.

Des Weiteren werden die Daten, wie bereits im vorherigen Abschnitt 3.1 erwähnt, von einer Implementierung von ScriptedImporter, einer Unity Schnittstelle, an eine Instanz der Klasse GXLImporter übergeben und darauf hin ein Graph Objekt erzeugt. Dies hat den Vorteil, dass die Datenstruktur unabhängig von Unity genutzt werden kann.

4.2 Konfigurationsparameter Darstellung

Auch für die Konfiguration der Software City wird versucht die Semantik der Datenstruktur aus der Unreal Variante beizubehalten, um auch hier die dazugehörigen Algorithmen weitestgehend unangetastet zu lassen.

Die größte Besonderheit in der Klasse Settings, welche die Konfigurationsparameter hält, ist das Feld ShowNodeTypes. Das Feld ist von dem Typ NodeInfoDictionary, welches nichts anderes ist als ein für die Unity Serialisierung optimiertes C# Dictionary. Zusätzlich darf es sich bei der Klasse um keine Klasse mit generischen Typen handeln, denn Unity kann mit generischen Datentypen grundsätzlich nur umgehen, wenn es sich dabei um den Typen List handelt. Aus diesem Grund erweitert die Klasse NodeInfoDictionary die serialisierbare Dictionary und definiert die benötigten generischen Typen.

4.3 Berechnung und Darstellung der Software Cities

Wie bereits im Abschnitt 3.3 erwähnt nutzen die Unreal Engine und Unity unterschiedliche Interpretationen der einzelnen Vektorkomponenten. Um diesem Unterschied entgegen zu wirken, ist es durchaus sinnvoll das grundlegende Konzept der Software daran anzupassen und das Unreal Koordinatensystem zu nutzen, bis es für die Darstellung in Unity in das Unity Koordinatensystem übertragen werden muss. Das nötige Fundament ist bereits für die erfolgreiche Nutzung des Unreal Koordinatensystems geschaffen worden, denn die verwendeten Datenstrukturen sind den aus der Unreal Variante von Projekt SEE semantisch weitestgehend gleich.

Aus diesem Anlass ist die Idee für die Schnittstelle ICityGenerator entstanden. Diese Schnittstelle stellt den Programmteil dar, welcher dafür zuständig ist aus einem Graphen in Kombination mit den Konfigurationsparametern eine Software City Struktur in Form eines logischen Baums zu erzeugen. Der Baum besteht dabei aus der Klasse Node, welche entweder eine Straße oder ein Haus der Software City repräsentieren. Die erste und hier im Projekt verwendete Implementierung der Schnittstelle ICityGenerator wird dabei das Unreal Koordinatensystem für die Berechnungen benutzen. Dies erlaubt es, wie bereits dutzende Male erwähnt, die Algorithmen aus Unreal ohne große Änderungen zu übernehmen. Es muss lediglich der Syntax an die Programmiersprache C# angepasst werden und eventuelle kleine

semantische Änderungen vorgenommen werden, um ggf. die Lesbarkeit zu erhöhen oder um eventuelle Feature Unterschiede zwischen C++ und C# auszugleichen.

Für das Rendern der Software City in der richtigen Orientierung ist es unerlässlich, dass sämtliche Vektoren in das Unity Koordinatensystem transformiert werden (siehe Tabelle 1). Hierfür wird wieder eine eigene Klasse erstellt, um zum einen die für Unity spezifische Implementierung für das Rendern der Software City von der Engine unabhängigen Generierung der Software City zu trennen. Zum anderen ermöglicht diese Vorgehensweise einen späteren Austausch der Implementierung der Software City Generierung und des Renderns dieser, ohne dabei die bestehenden Implementierungen ändern zu müssen, wenn beispielsweise eine Implementierung ohne das Unreal Koordinatensystem gewünscht ist.

Die für diese genannten Punkte verwendete Klasse heißt SoftwareCity und erweitert die Unity Klasse MonoBehavior. Diese Klasse stellt den Eintrittspunkt für das Rendern und Verhalten von sog. GameObjects dar und wird als Component eines GameObjects bezeichnet. [20] Dabei können Components einem GameObject beliebig hinzugefügt werden, welches die eben erwähnte Austauschbarkeit ermöglicht und fördert.

Die Vorgehensweise des Component SoftwareCity ist wie folgt: sie nutzt einen GXLImporter um einen Graphen zu erzeugen, welcher daraufhin einer Implementierung von ICityGenerator übergeben wird, in diesem Fall EvostreeCityGenerator. Diese erzeugt darauf hin einen Baum aus Nodes, welcher eine Software City repräsentiert. Dieser Baum wird dann von dem Component SoftwareCity in das Unity Koordinatensystem übersetzt und daraufhin werden alle Nodes entsprechend ihres Typs, ob Haus oder Straße, als GameObject erzeugt und gerendert.

4.4 Virtual Reality

Die Klasse für die Steuerung ist nicht in Abbildung 1 enthalten, da diese unabhängig von dem Rest betrachtet werden kann. Wie bereits im Abschnitt 3.4 erwähnt ist die Umsetzung einer Virtual Reality Funktionalität mit Unity eigenen Mitteln möglich. Man braucht nur eine Virtual Reality Brille anzuschließen und die nötige Unterstützung in Unity aktivieren.

Lediglich die vollständige Bewegung im dreidimensionalen Raum muss gesondert implementiert werden. Es ist nötig für die Kamera in Unity ein Eltern GameObject zu definieren, da die Unterstützung einer Virtual Reality Brille in Unity impliziert, dass die Kamera Position und Rotation an die Virtual Reality Brille gekoppelt werden und nicht manuell im Code geändert werden kann. Dieses Eltern GameObject kann problemlos im dreidimensionalen Raum transformiert werden und man bekommt somit einen indirekten Einfluss auf die Transformation der Kamera.

5 Vergleich Prototyp und Projekt SEE

Für die folgenden Vergleiche zwischen dem Prototypen in Unity oder der Implementierung von Projekt SEE in Unreal wird ein Computer mit folgender Spezifikation verwendet:

- CPU: Intel Core i7 4790K
- GPU: Nvidia GTX 970
- RAM: 16 GB 1600 MHz

In den nachfolgenden Abschnitten wird zunächst der genutzte Testaufbau beschrieben, um einen zuverlässigen und wiederholbaren Vergleich zu gewährleisten. Im Anschluss werden die Ergebnisse der Tests bzw. Vergleiche dargestellt.

Für die Konfigurationsparameter der Software City Generierung werden folgende Werte genutzt:

- Genutzte GXL Datei: java2rfg-ohne_DOC.gxl
- Anzuzeigende Node Types: Class
- Scale Mapping für Node Type Class: X=1; Y=1; Z=Source.Region.Length
- Scale Mapping Scaling für Node Type Class: X=1; Y=1; Z=0.1
- Anzuzeigende Edge Types: Keine

5.1 Bilder pro Sekunde

Um die Performance der beiden Implementierungen von Software Cities zu vergleichen, wird zunächst die klassische Performance-Metrik der Bilder pro Sekunde betrachtet. Hierfür werden die Anzahl der Bilder pro Sekunde sowohl im jeweiligen Editor als auch in der Release Variante gemessen.

Bei der Messung in den jeweiligen Editoren werden die Bilder pro Sekunde zum einen über die Editor eigene Statistik Anzeige abgelesen als auch über den dazugehörigen Profiler. Für die Messung in der Release Variante wird das externe Programm FRAPS genutzt. Dieses Programm besitzt sowohl eine kostenpflichtige als auch eine kostenlose Version. Die Funktion zum Messen der Bilder pro Sekunde ist in beiden Versionen enthalten, somit ist die kostenlose Version vollkommen ausreichend.

Des Weiteren sind sämtliche Konfigurationsparameter, welche für die Darstellung der Software City entscheidend sind, gleich. Um möglichst vergleichbare Ergebnisse zu erhalten,

wird außerdem eine ähnliche Perspektive der Kamera eingestellt und die Ergebnisse werden bei stiller Kamera abgelesen.

5.1.1 Unity Editor Ergebnis

Bei Unity ist die Messung über die integrierte Anzeige der Bilder pro Sekunde sehr einfach. Sie muss lediglich aktiviert werden und man bekommt das Ergebnis von ca. 1100-1300 Bildern pro Sekunde zu sehen. Die meiste Zeit bewegt sich die Zahl jedoch im Bereich 1200.

Nutzt man hingegen den Profiler von Unity erhält man ein Ergebnis von rund 800-850 Bildern pro Sekunde. Dieses Ergebnis lässt sich damit erklären, dass der Profiler zusätzliche Zeit benötigt um seine Informationen zu erhalten.

5.1.2 Unreal Editor Ergebnis

Die Messung der Bilder pro Sekunde im Unreal Editor ist prinzipiell ähnlich wie bei Unity. Es muss nur die Option aktiviert werden, dass die Bilder pro Sekunde angezeigt werden sollen. Das Ergebnis wirkt jedoch mit 120 Bildern pro Sekunde zunächst ernüchternd. Nach einer kurzen Recherche bei Google findet man heraus, dass der Unreal Editor eine Begrenzung der Bilder pro Sekunde standardmäßig aktiviert hat. Diese Begrenzung kann jedoch in den Einstellungen von Unreal angepasst werden, sodass es diese nicht mehr geben sollte.

Nach dem die Einstellungen für die Begrenzung übernommen wurden, kann man feststellen, dass die Anzahl der Bilder pro Sekunde durchaus gestiegen ist. Sie beträgt nun ca. 200 Bilder pro Sekunde.

5.1.3 Fazit Editor Ergebnisse

Auch nach längerer Recherche habe ich keine Möglichkeit gefunden diese Anzahl zu erhöhen. Es kann also sein, dass es durch den technischen Aufbau des Unreal Editors nicht möglich ist eine höhere Anzahl an Bildern pro Sekunde zu erreichen. Somit ist es jedoch auch nicht möglich mit diesen Ergebnissen ein Vergleich zu ziehen. Es kann lediglich festgestellt werden, dass beide Engines eine ausreichende Anzahl an Bildern pro Sekunde liefert.

5.1.4 Unity Release Ergebnis

Für das Erstellen einer Release Version in Unity ist es nötig V-Sync auszustellen, um eine Begrenzung der Anzahl der Bilder pro Sekunde auszuschließen. Außerdem ist es nötig Klassen, welche Abhängigkeiten zu Editorfunktionalitäten haben, in einen Unterordner namens Editor zu verschieben. Sämtliche Klassen und Dateien, welche sich in diesem Ordner befinden, werden beim Erstellen einer Release Version ignoriert.

Das Ergebnis liegt bei ca. 1340-1350 Bildern pro Sekunde. Wie bereits an dem Zahlenbereich zu erkennen ist, liegen die Zahlen über dem im Unity Editor. Außerdem gibt es eine geringere Schwankung bei der Release Variante. Dies ist damit zu erklären, dass im Editor noch die weitere grafische Oberfläche berechnet und gerendert werden muss.

5.1.5 Unreal Release Ergebnis

Die Erstellung einer verpackten Release Version in Unreal ist aufgrund von genutzten Abhängigkeiten des Editors in Projekt SEE nicht möglich. Diese Erkenntnis stammt von Studenten, welche sich mit dem Projekt SEE in einem Bachelorprojekt beschäftigt haben. [27]

Dabei handelt es sich prinzipiell um den gleichen Fehler, welcher beim Erstellen der Unity Release Version aufgetreten ist. Dieser konnte jedoch schnell beseitigt werden, da die Abhängigkeiten zum Editor in der Unity Variante sich auf zwei Klassen beschränkten.

Bei der Unreal Variante von Projekt SEE sind diese Abhängigkeiten jedoch in vielen Teilen des Codes vorhanden und können nicht ohne weiteres entfernt werden. Die Studenten aus dem Bachelorprojekt haben bereits unterschiedliche Ansätze ausprobiert. Diese waren leider nicht erfolgreich. Ein weiterer Lösungsvorschlag der Studenten ist die Neuimplementierung des Multiplayers ohne editorspezifische Funktionalitäten. Dieses wurde aufgrund von mangelnder Zeit von den Studenten nicht umgesetzt. Aufgrund des Umfangs ist es auch kein Bestandteil dieser Bachelorarbeit.

Es ist somit mit der jetzigen Implementierung von Projekt SEE in Unreal nicht möglich eine aussagekräftige Anzahl an Bildern pro Sekunde zu messen.

5.2 Anwendung von Quelltextänderungen

Bei der Entwicklung von Software ist es unerlässlich, dass man die vorgenommenen Änderungen anwenden kann und die Software zum Testen ausführen kann. Die Zeit, welche

erforderlich ist, dass Änderungen übernommen werden, ist ein nicht zu vernachlässigender Aspekt. Der Entwickler verliert Zeit während Änderungen angewendet werden.

Aus diesem Grund wird die Zeit verglichen, welche nötig ist um Änderungen am Quelltext in den jeweiligen Game Engines Unreal und Unity anzuwenden. Hierfür wird eine Änderung am Quelltext vorgenommen und daraufhin die Zeit gemessen wie lange es braucht, bis diese in der jeweiligen Engine zu sehen sind bzw. angewandt wurden. Die Messung wird dabei drei Mal mittels einer Stoppuhr ausgeführt, um eine verlässliche Aussage treffen zu können.

5.2.1 Unity Ergebnis

Unity wendet die Quelltextänderungen an, wenn man diese speichert und daraufhin in das Editorfenster zurück wechselt. Dann wird der Cursor eine kurze Zeit zum Ladesymbol und der Unity Editor ist in dieser Zeit nicht nutzbar. Die Ergebnisse der Messungen stellen somit die Zeit zwischen dem Wechseln in den Unity Editor und dem Zeitpunkt ab dem der Editor wieder nutzbar ist dar. Dabei wurde eine durchschnittliche Zeit von ca. 2053 ms erreicht. Das Ergebnis der Messung ist durch die Reaktionszeit beim händischen Zeitmessen mit der Stoppuhr etwas höher als bei einer automatischen Messung.

Dieses Ergebnis kann mittels sog. Assembly Definitions in Unity verbessert werden. Der jetzige Stand ohne Assembly Definitions hat zur Folge, dass bei einer Quelltextänderung alle Scripts neu kompiliert werden müssen. Mit Assembly Definitions ist es möglich Scripts in unterschiedliche Module - Assemblies - zu unterteilen, was den Vorteil hat, dass bei Quelltextänderungen nur Scripts neu kompiliert werden müssen, welche von der Änderung direkt bzw. über Abhängigkeiten betroffen sind. [19]

In Unity spielt es dabei keine Rolle ob die Änderung am Quelltext den Editor betrifft oder nicht, die Zeit bis die Änderung angewendet wird bleibt die gleiche.

Durchlauf	Zeit in ms
1	2290
2	1940
3	1930

Tabelle 2: Unity Performance Quelltextänderung

5.2.2 Unreal Ergebnis

In Unreal werden die Änderungen nicht automatisch übernommen. Damit Quelltextänderungen in Unreal angewandt werden, muss das Kompilieren der Änderungen im Unreal

Editor manuell gestartet werden. Hierfür gibt es eine Schaltfläche 'Compile' im Unreal Editor. Die Zeit muss hierbei nicht mit einer Stoppuhr gemessen werden, da es Logmeldungen mit Zeitstempeln zum Kompiliervorgang gibt. Das Ergebnis der Messung liegt bei durchschnittlich ca. 1605 ms.

Durchlauf	Zeit in ms
1	1622
2	1617
3	1575

Tabelle 3: Unreal Performance Quelltextänderung

5.3 Performance GXL Parsing

Bevor eine Software City generiert werden kann, müssen die GXL Dateien importiert werden. Dies findet pro GXL Datei einmal statt, wenn die Daten initial hinzugefügt werden oder wenn Änderungen am Import Code vorgenommen werden.

Für die Messung der Geschwindigkeit für den Import der GXL Dateien werden Logmeldungen geschrieben, einmal zu Beginn des Imports und einmal zum Ende des Imports. Die zeitliche Differenz bildet dabei die Dauer, welche benötigt wird um die GXL Dateien vollständig zu importieren. Beim Import wird die zum Projekt java2rfg dazugehörige GXL Datei verwendet. Zudem wird angenommen, dass der Log Befehl keinen nennenswerten Einfluss auf das Ergebnis hat. Die GXL Datei ist außerdem auf einer SSD gespeichert, um eine minimale Dauer beim Lesen der Datei zu erreichen. Die Messung wird dabei drei Mal ausgeführt, um eine verlässliche Aussage treffen zu können.

5.3.1 Unity Ergebnis

Die Zeitstempel der Logmeldungen in Unity ist auf Sekunden beschränkt. Daher wurde Code in die für den GXL Import verantwortliche Klasse eingebaut, um Messungen im Millisekunden Bereich anzeigen zu können. Wie in der Tabelle 4 zu sehen, bewegt sich das Ergebnis bei ca. 700 ms für den Import von GXL Dateien.

Im Vergleich zu der Unreal Variante von Projekt SEE ist es in Unity möglich die GXL Dateien mit dem DOCTYPE Xml Tag zu parsen.

Durchlauf	Zeit in ms
1	659
2	708
3	703

Tabelle 4: Unity Performance GXL Parsing

5.3.2 Unreal Ergebnis

In der Unreal Engine ist es möglich die Zeitstempel der Logmeldungen mit einer Millisekunden Auflösung zu konfigurieren. Dies ist bequem über den Editor möglich. Somit muss für die Messung kein weiterer Code eingebaut werden. Die Ergebnisse sind wie in Tabelle 5 zusehen rund doppelt so hoch die bei Unity.

Durchlauf	Zeit in ms
1	1401
2	1257
3	1361

Tabelle 5: Unreal Performance GXL Parsing

5.4 Performance Software City Generierung

Aufgrund der Tatsache, dass die Algorithmen für die Generierung der Software City sehr ähnlich bis identisch sind, ist der Vergleich der Dauer für die Generierung einer Software City sehr aussagekräftig. Diese Messung ist ein guter Indikator für den Umgang und die effiziente Ausnutzung der CPU von den jeweiligen Game Engines, da hierbei keine Daten von einem langsamen, nicht flüchtigen Speichermedium gelesen werden müssen. Die Berechnung basiert allein auf Daten, welche bereits im Arbeitsspeicher vorhanden sind. Des Weiteren werden bei diesem Test keine Daten an die GPU des Systems übertragen, da die Berechnung nicht im sog. Render-Loop stattfindet.

Wie auch beim vorherigen Testaufbau im Abschnitt 5.3 für die Performancemessung des GXL Parsings wird bei der Performancemessung für die Generierung einer Software City auf Logausgaben gesetzt, da dies zuverlässig und wiederholbar ausgeführt werden kann und vor allem genauer ist als die Zeit von Hand zu messen.

5.4.1 Unity Ergebnis

Auch bei dieser Messung muss in Unity für die Genauigkeit im Millisekunden Bereich Code in der dafür zuständigen Klasse hinzugefügt werden. Daraufhin kann im Unity Editor auf

den Play Knopf gedrückt werden, damit die Software City Generierung beginnt. Dabei ist es unerheblich, ob es der erste oder ein wiederholter Durchlauf ist, denn es werden keine Daten zwischen Durchläufen zwischengespeichert. Der Mittelwert der Messungen in Unity liegt bei ca. 617 Millisekunden. Die Ergebnisse der Durchläufe können in der Tabelle 6 betrachtet werden.

Durchlauf	Zeit in ms
1	516
2	726
3	608

Tabelle 6: Unity Performance City Generierung

5.4.2 Unreal Ergebnis

Wie bereits im Abschnitt 5.3.2 beschrieben, sind die Zeitstempel der Logmeldungen genau genug, um ein Ergebnis im Millisekunden Bereich zu erhalten. Aufgrund der Tatsache, dass die Generierung der Software City Hierarchie zwischen mehreren Durchläufen zwischengespeichert wird, wurde das Objekt zwischen den Durchläufen komplett aus dem Unreal Editor entfernt, um einen gleichen Ausgangspunkt für die Messungen zu haben. Das Ergebnis von durchschnittlich 290 Millisekunden ist somit bei der Hälfte der Zeit, welche Unity für die Generierung einer Software City benötigt. Für die Ergebnisse sämtlicher Durchläufe siehe Tabelle 7.

Durchlauf	Zeit in ms
1	258
2	339
3	273

Tabelle 7: Unreal Performance City Generierung

5.5 Dokumentation

Bei Game Engines handelt es sich um funktionsreiche und durchaus komplexe Software. Damit ein Entwickler mit solch einer Software arbeiten kann, wird ein Leitfaden bzw. Dokumentation für die Nutzung der jeweiligen Game Engines benötigt. Bei diesem Vergleich handelt es sich um ein nicht funktionales, qualitatives Kriterium.

Es wird unter anderem die Verfügbarkeit von Dokumentation für die jeweiligen Game Engines, als auch die Qualität und Vollständigkeit dieser verglichen.

5.5.1 Unity Ergebnis

Für die Unity Engine gibt es drei unterschiedliche Arten von Dokumentation, welche man als Entwickler nutzen kann. Es gibt ein Benutzerhandbuch, eine API Referenz und videobasierte Lektionen. Bei allen drei Dokumentationsformen handelt es sich um Informationen, welche direkt vom Unity Hersteller kommen.

Das Unity Benutzerhandbuch beschreibt unter anderem die Editorbenutzeroberfläche und die Konzepte von Unity. Die Konzeptbeschreibung beinhaltet dabei oft sowohl die konzeptionelle Beschreibung als auch Beispielcode für die Implementierung und Anwendung dieser Konzepte. Es gibt zudem Verweise zu den genutzten und ggf. weiteren interessanten Klassen in der API Referenzbeschreibung gegeben.

Wie bereits erwähnt ist für Unity eine API Referenzbeschreibung verfügbar. Sie beschreibt die von Unity zur Verfügung stehenden Schnittstellen. Dies ist sehr umfangreich gestaltet. Sämtliche Klassen und Methoden beschreiben mindestens mit einem kurzen Satz was diese Klassen und Methoden machen. Bei den zentralen und oft genutzten Klassen ist die Beschreibung dann umfangreicher. Es wird beispielsweise beschrieben, wofür die Klassen genutzt werden können und es ist Beispielcode vorhanden. Dieser Beispielcode ist oft ein anderer als aus dem Benutzerhandbuch. Des Weiteren gibt es auch Verweise aus der API Referenzbeschreibung zum Benutzerhandbuch, wenn dort Konzepte aus dem Benutzerhandbuch aufgegriffen werden.

Die dritte Kategorie der Dokumentation von Unity bilden die videobasierten Lektionen, sog. Tutorials. Für diese Tutorials hat Unity Technologies eine eigene Plattform geschaffen. Die vielen Tutorials sind von Unity Technologies selbst erstellt worden. Es sind jedoch auch Tutorials von anderen Personen zu finden. Bei den Tutorials handelt es sich um Beispiel-Spiele und die praktische Beschreibung von Konzepten in Unity. Die Qualität der Videos wirkt zudem sehr professionell.

Alle drei Arten von Dokumentationen ergänzen sich gegenseitig, sodass der Einstieg in Unity sehr vereinfacht wird.

5.5.2 Unreal Ergebnis

Epic Games stellt für die Unreal Engine zwei Arten von Dokumentationen zur Verfügung. Für die Unreal Engines gibt es ein Benutzerhandbuch und eine API Referenzbeschreibung.

Das Benutzerhandbuch für Unreal ist ähnlich aufgebaut wie das von Unity. Es gibt eine Beschreibung der Benutzeroberfläche und der Konzepte bzw. Funktionalitäten, welche Unreal

zu bieten hat. Neben der rein textuellen Beschreibung der Konzepte gibt es auch Beispielcode, welche die Umsetzung der Konzepte darstellen. Es gibt jedoch meist keine Verweise auf die API Referenzbeschreibung.

Die API Referenzbeschreibung ist in drei Teile aufgeteilt und beschreibt die Schnittstellen für das Blueprint Scripting, Python und C++. Dabei gibt es teilweise Unterschiede in der Darstellung und Vollständigkeit der Referenzbeschreibungen. So sind die Blueprint und C++ Referenzbeschreibungen Teil des Benutzerhandbuchs und für die Python Dokumentation wird man auf eine unabhängige Seite weitergeleitet. Des Weiteren ist die Blueprint Scripting Dokumentation am besten beschrieben. Der Großteil der Schnittstellen des Blueprint Scriptings ist beschrieben. Für die Python und C++ Schnittstellen ist dies nicht der Fall. Dort sind viele Schnittstellen wenig bis gar nicht beschrieben, sodass es schwierig ist ggf. gesuchte Schnittstellen zu finden bzw. gefundene und benötigte Schnittstellen zu verstehen und zu nutzen.

Grundsätzlich ist eine Dokumentation für die Unreal Engine vorhanden. Sie ist jedoch zum Teil unvollständig bzw. weist eine geringe Qualität auf.

5.6 Community

Die Community rund um eine Game Engine kann durchaus ein Wettbewerbsvorteil sein. Denn mit einer großen und aktiven Community ist es Entwicklern möglich Informationen und Erfahrungen mit anderen Entwicklern auszutauschen, um beispielsweise oft auftretende Probleme schnell lösen zu können. Die Community in Form von Foren oder Wikis kann somit als weitere Sammlung von Dokumentation und Information über eine Game Engine angesehen werden, denn es werden beispielsweise Lösungen für Probleme, welche bei der Nutzung einer Game Engine auftauchen können in den Foren bzw. Wikis archiviert und können über eine Suchmaschine gefunden werden.

5.6.1 Unity Ergebnis

Rund um Unity gibt es viele Nutzer, welche sehr aktiv in der Community unterwegs sind und gerne Hilfe leisten. Von den Unity Entwicklern stehen zwei Plattformen für die Nutzer zum Austausch von Informationen und Erfahrung zur Verfügung.

Zum einen das Unity Forum, welches es erlaubt über Fragen, Anregungen und ggf. Verbesserungsvorschläge zu diskutieren. Es ist sehr gut und übersichtlich strukturiert, sodass man sich dort leicht zurecht findet. Auch im Verlauf dieser Bachelorarbeit habe ich diese Informationsquelle genutzt, um beispielsweise den Fehler der Editor Abhängigkeiten beim

Erstellen einer Release Version zu beheben. Auch viele Vergleiche, welche online verfügbar sind, loben die um Unity vorhandene Community sehr. [12] [13] [25]

Des Weiteren stellen die Unity Entwickler eine an Stackoverflow erinnernde Plattform namens Answers zur Verfügung. Hier können Nutzer von Unity Fragen über die Nutzung und auftauchende Fehler stellen. Es kann außerdem als FAQ Sammlung angesehen werden, da viele Fragen, die Nutzer haben, oft schon gestellt wurden und die Antworten auf diese Fragen offen einsehbar sind. Die Fragen und Antworten reichen teilweise mehrere Jahre zurück; viele der Antworten, welche im Rahmen dieser Bachelorarbeit recherchiert wurden, waren durchaus noch aktuell.

Grundsätzlich ist es möglich für die Mehrheit der Fragen die man hat Antworten entweder über das Forum oder Unitys Answers zu finden. Sollte man jedoch keine Antwort auf eine Frage finden, kann man diese problemlos auf den genannten Plattformen stellen und erhält mit hoher Wahrscheinlichkeit eine Antwort.

Außerdem gibt es von der Community erstellte Tutorials und andere Lernhilfen, um den Umgang mit Unity möglichst schnell und einfach zu erlernen.

5.6.2 Unreal Ergebnis

Auch Epic Games, die Entwickler von der Unreal Engine, stellen ihren Nutzern Möglichkeiten zum Austausch zur Verfügung. Insgesamt gibt es vier von Epic Games zur Verfügung gestellte Plattformen.

Wie auch für Unity gibt es ein Forum rund um die Unreal Engine. Dieses besitzt eine ähnliche Struktur wie das Unity Forum. Es ist in Themengebiete, Features und Zielplattformen strukturiert und macht einen guten, wenn aber auch älteren Eindruck. Schaut man sich einige Kategorien an, so kann man eine Tendenz erkennen, dass die Beiträge auf dem Unreal Forum wesentlich weniger Aufrufe und Antworten haben. Dies bestätigt zunächst die Aussagen der anderen Vergleiche zwischen Unity und Unreal, dass die Community rund um Unity größer und aktiver ist als um Unreal herum.

Auch für die Unreal Engine gibt es einen Stackoverflow Ableger namens AnswerHub. Auf dieser Plattform kann man, wie der Name suggeriert, Antworten auf seine Fragen rund um Unreal bekommen bzw. bereits beantwortete Fragen durchsuchen.

Des Weiteren gibt es von den Entwicklern von Unreal eine Wiki, welche von den Nutzern eingesehen und bearbeitet werden kann. Zum Zeitpunkt dieser Bachelorarbeit arbeitet Epic Games an einer neuen Wiki, weshalb einige Bereiche des Wiki nur einen Verweis auf die bald erscheinende neue Wiki enthalten. Da es sich bei der Wiki größtenteils um Inhalte

handelt, welche von der Community erstellt worden sind und es keine Qualitätskontrolle seitens der Entwickler gibt, ist die Qualität der Beiträge sehr unterschiedlich. Es gibt Beiträge, welche jeden Schritt für ein bestimmtes Ziel beschreiben bzw. erklären und eine sehr gute Qualität aufweisen. Jedoch existieren auch Beiträge, welche nur eine Überschrift besitzen und Code Beispiele ohne jegliche Erklärung.

Aufgrund der Tatsache, dass der Quelltext der Unreal Engine grundsätzlich auf GitHub verfügbar ist, kann die Unreal Engine Projektseite auf GitHub als Community Plattform angesehen werden. Dort können Nutzer beispielsweise potenzielle Fehler mit der Unreal Engine melden und sogar Änderungsvorschläge machen oder Fehler selber versuchen zu beheben.

Zusammenfassend bietet Epic Games unterschiedliche Plattformen für die Nutzer der Unreal Engine an um sich auszutauschen. Aufgrund der kleineren Community, habe ich festgestellt, dass einige Fragen, welche ich im Verlauf dieser Bachelorarbeit hatte, nicht zu finden waren oder die Antworten für eine ältere Version der Unreal Engine gedacht waren und mir somit nicht weitergeholfen haben.

6 Fazit

Zu Beginn dieser Bachelorarbeit war eine Recherche der wesentlichen Funktionsweise von Unity nötig, um bei der Einarbeitung in den Projekt SEE Code in Unreal Parallelen zu Unity ziehen zu können und einen Plan für den Unity Prototypen entwickeln zu können.

Nachdem das Verständnis des Unreal Codes vorhanden war, war es mit dem Wissen der benötigten Funktionen möglich eine grobe Architektur für die Implementierung in Unity zu erstellen. Mit dieser groben Architektur wurde daraufhin eine Grundlage geschaffen, um die Algorithmen für die Generierung des Software City weitestgehend übernehmen zu können. Das Verstehen des Unreal Codes und Erstellen einer Architektur für den Unity Prototypen inklusive technischer Recherchen, war der größte Aufwand im Verlauf der Implementierung einer Unity Variante von Projekt SEE.

Nach der Schaffung einer Datenstrukturgrundlage und des Imports von GXL Dateien waren die Algorithmen für die Generierung der Software Cities an der Reihe. Diese konnten innerhalb von wenigen Tagen nach C# übernommen werden, da größtenteils nur der Syntax angepasst werden musste. Die Korrektheit der Algorithmen konnte mittels der gleichen Logmeldungen verifiziert werden.

Anschließend musste die Software City, welche zu diesem Zeitpunkt nur als Baum im Arbeitsspeicher existierte, gerendert werden. Aufgrund des zu diesem Zeitpunkt fehlenden Wissens, dass sich das Koordinatensystem zwischen Unity und Unreal unterscheidet, hat dieser Teil der Implementierung den zweitgrößten Anteil des Aufwands ausgemacht. Infolge der zusätzlichen Unerfahrenheit mit Unity, bestand oft der Gedanke, die Implementierung bzw. das Verständnis für die Schnittstellen zum Rendern wäre falsch gewesen. Zusätzlich wurde versucht die Struktur der Konfigurationsparameter gleich zu halten.

Abschließend ist die Machbarkeit für Projekt SEE in Unity gegeben. Sämtliche Anforderungen, welche für die Umsetzung von Projekt SEE benötigt werden, sind in Unity vorhanden und können erfolgreich genutzt werden, um einen Prototypen zu realisieren. Die Performance der beiden Implementierungen ist in den meisten Punkten wie erwartet ähnlich. Die Tatsache, dass es nicht möglich war die Anzahl an Bildern pro Sekunde in Unreal zu erfassen, macht den Vergleich der Leistung des Endprodukts schwierig. Weiterhin fiel mir die Einarbeitung in Unity um ein vielfaches leichter als in Unreal, was dadurch bedingt war, dass ausführlichere und bessere Dokumentation über Unity vorhanden ist.

Schließlich stellt Unity eine gute Alternative als Plattform für die weitere Forschung an Software Cities dar.

7 Ausblick

Es können noch viele Verbesserungen und Optimierungen an der Implementierung des Unity Prototypen vorgenommen werden. Dies schließt unter anderem das Starten für die Generierung der Software City ein. Es ist durchaus denkbar, dass die Generierung der Meshes für die Software City über einen gesonderten Knopf im Editor gestartet werden kann und somit persistent bleibt.

Ferner können die weiteren Funktionalitäten aus der Unreal Variante implementiert werden, wie beispielsweise das Anzeigen von sog. Districts, der grafischen Gruppierung von zusammengehörigen Häusern.

Auch ein Multiplayer ist mit der Unity Engine mit Unity eigenen Mitteln umsetzbar. Hierfür sind viele Informationen in den Unity Dokumentationen vorhanden.

Literatur

[1] 1runeberg. *GitHub - 1runeberg/RunebergVRPlugin*. URL: https://github.com/1runeberg/RunebergVRPlugin (besucht am 17. 10. 2019).

[2] azixMcAze. *GitHub - azixMcAze/Unity-SerializableDictionary*. URL: https://github.com/azixMcAze/Unity-SerializableDictionary (besucht am 16. 10. 2019).

[3] Rune Berg. *Virtual Reality Pawn and Components Plugin by Rune Berg in Code Plugins - UE4 Marketplace*. 13. Okt. 2016. URL: https://www.unrealengine.com/marketplace/en-US/slug/vr-pawn-components-plugin (besucht am 17. 10. 2019).

[4] Paul E. Dickson u. a. „An Experience-based Comparison of Unity and Unreal for a Stand-alone 3D Game Development Course“. In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '17*. the 2017 ACM Conference. Bologna, Italy: ACM Press, 2017, S. 70–75. ISBN: 978-1-4503-4704-4. DOI: 10.1145/3059009.3059013. URL: http://dl.acm.org/citation.cfm?doid=3059009.3059013 (besucht am 20. 06. 2019).

[5] Epic Games. *UPROPERTY - Epic Wiki*. URL: https://wiki.unrealengine.com/UPROPERTY (besucht am 16. 10. 2019).

[6] Przemyslaw Gora und Lukas Leibetseder. *Unreal vs Unity - Ein Vergleich zwischen zwei modernen Spiele-Engines*. 25. Okt. 2016.

[7] grbury. *Entwicklung - Mixed Reality*. URL: https://docs.microsoft.com/de-de/windows/mixed-reality/development (besucht am 13. 10. 2019).

[8] Christian Grohganz. *Die meisten Entwickler nutzen Unity – Geld macht man aber mit Unreal*. VR Nerds. 6. März 2017. URL: https://www.vrnerds.de/die-meisten-entwickler-nutzen-unity-geld-macht-man-aber-mit-unreal/ (besucht am 20. 06. 2019).

[9] Jean-Luc Lugrin u. a. „CaveUDK: a VR game engine middleware“. In: *Proceedings of the 18th ACM symposium on Virtual reality software and technology - VRST '12*. the 18th ACM symposium. Toronto, Ontario, Canada: ACM Press, 2012, S. 137. ISBN: 978-1-4503-1469-5. DOI: 10.1145/2407336.2407363. URL: http://dl.acm.org/citation.cfm?doid=2407336.2407363 (besucht am 20. 06. 2019).

[10] Raka Mahesa. *Unity and Unreal comparison*. Packt Hub. 26. Jan. 2018. URL: https://hub.packtpub.com/unity-and-unreal-comparison/ (besucht am 20. 06. 2019).

[11] Microsoft. *Übersicht über LINQ to XML (C#) | Microsoft Docs*. 30. Okt. 2018. URL: https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/concepts/linq/linq-to-xml-overview (besucht am 15. 10. 2019).

[12] Sirawat Pitaksarit. *Objectively comparing Unity and Unreal Engine*. Game Torrahod. 16. Jan. 2019. URL: https://gametorrahod.com/objectively-comparing-unity-and-unreal-engine/ (besucht am 20. 06. 2019).

[13] PONTYPANTS. *Unity VS Unreal - Which Engine Should You Choose?* Sunday Sundae. 8. Aug. 2018. URL: https://sundaysundae.co/unity-vs-unreal/ (besucht am 20. 06. 2019).

[14] Giuseppe Portelli. *World Coordinate Systems in 3ds Max, Unity and Unreal Engine | A Clockwork Berry*. 2. Aug. 2015. URL: http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/ (besucht am 16. 10. 2019).

[15] Rama. *How to Change FPS Cap , Near Clip, Editor & Game - Epic Wiki*. URL: https://wiki.unrealengine.com/How_to_Change_FPS_Cap_,_Near_Clip,_Editor_%26_Game (besucht am 23. 10. 2019).

[16] Carsten Seifert und Jan Wislaug. *Spiele entwickeln mit Unity 5: 2D- und 3D-Games mit Unity und C# für Desktop, Web & Mobile*. 3., aktualisierte Auflage. OCLC: 985979231. München: Hanser, 2017. 650 S. ISBN: 978-3-446-45368-5 978-3-446-45197-1. URL: http://www.hanser-elibrary.com/doi/book/10.3139/9783446453685.

[17] Antonín Šmíd. *Comparison of Unity and Unreal Engine*. Mai 2017.

[18] Frank Steinbrückner. „Consistent Software Cities“. In: (), S. 244.

[19] Unity Technologies. *Unity - Manual : Assembly Definitions*. URL: https://docs.unity3d.com/Manual/ScriptCompilationAssemblyDefinitionFiles.html (besucht am 03. 11. 2019).

[20] Unity Technologies. *Unity - Manual : GameObject*. URL: https://docs.unity3d.com/Manual/class-GameObject.html (besucht am 03. 11. 2019).

[21] Unity Technologies. *Unity - Manual: Scripted Importers*. URL: https://docs.unity3d.com/Manual/ScriptedImporters.html (besucht am 16. 10. 2019).

[22] Unity Technologies. *Unity - Manual: VR overview*. URL: https://docs.unity3d.com/Manual/VROverview.html (besucht am 17. 10. 2019).

[23] Unity Technologies. *Unity - Scripting API: MonoBehaviour*. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html (besucht am 20. 10. 2019).

[24] Thinkwik. *CryEngine vs Unreal vs Unity: Select the Best Game Engine*. Medium. 20. Apr. 2018. URL: https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e (besucht am 20. 06. 2019).

[25] *Unity vs Unreal: Ultimate Game Engine Showdown*. The Ultimate Resource for Video Game Design. 9. Mai 2019. URL: https://www.gamedesigning.org/engines/unity-vs-unreal/ (besucht am 20. 06. 2019).

[26] Ryan Vance. *Unreal Engine 4 support for HoloLens 2 released in beta*. Unreal Engine. 31. Mai 2019. URL: https://www.unrealengine.com/en-US/blog/unreal-engine-4-support-for-hololens-2-released-in-early-access (besucht am 13. 10. 2019).

[27] Nico Weiser u. a. *Projekt SEE Projektbericht*.

[28] Richard Wettel. *CodeCity*. 22. Mai 2017. URL: https://wettel.github.io/codecity.html (besucht am 11. 07. 2019).

[29] Richard Wettel, Michele Lanza und Romain Robbes. „Software systems as cities: a controlled experiment“. In: *Proceeding of the 33rd international conference on Software engineering - ICSE '11*. Proceeding of the 33rd international conference. Waikiki, Honolulu, HI, USA: ACM Press, 2011, S. 551. ISBN: 978-1-4503-0445-0. DOI: 10.1145/1985793.1985868. URL: http://portal.acm.org/citation.cfm?doid=1985793.1985868 (besucht am 11. 07. 2019).