Bachelorarbeit

Digitale Führung – Eine Augmented Reality Anwendung am Beispiel einer Museumsapp

Dennis Penno

Matrikelnummer: 21429

Fachbereich Ingenieur- und Naturwissenschaften

1. Betreuer:	Ulrich Borchert
2. Betreuer:	Prof. Dr. Sven Karol

Selbstständigkeitserklärung

Klitzschen, 25. Februar 2020 .....

Hiermit bestätige ich, Dennis P e n n o , dass ich die hier vorliegende Arbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Abstract

Die vorliegende Arbeit handelt von der Entwicklung einer Augmented Reality Applikation für mobile Endgeräte mit Unity. Diese Software ist eine Laufzeit - und Entwicklungsumgebung für Spiele oder andere Applikationen.

Auf den folgenden Seiten wird erläutert, wie sich die gewünschte Software aus verschiedenen Komponenten langsam und Stück für Stück zusammensetzt. Dabei wird der Fokus auf den zwei Hauptkomponenten, der Augmented – Reality – Komponente und der Charakter – und Indoor – Navigations – Komponente liegen.

Die Arbeit ist für alle Leser relevant, die mit der Entwicklung von mobilen Applikationen auf Unity Basis arbeiten oder sich dafür interessieren. Zudem ist sie für Leser aus dem kulturellen Bereich von Belang, die eine Hauseigene Applikation auf der Basis von Augmented Reality entwickeln lassen möchten.

Inhaltsverzeichnis

Abkürzungsverzeichnis

2D	zweidimensional
3D	dreidimensional
A-GPS	Assisted Global Positioning System
App	Application
AR	Augmented Reality
AV	Augmented Virtuality
BLE	Bluetooth Low Energy
GPS	Global Positioning System
ID	Identität
IPS	Indoor Positioning System
JDK	Java Development Kit
MR	Mixed Reality
RFID	radio-frequency identification
SDK	Software Development Kit
ToF	Time of Flight
UI	User Interface
UID	Unique Identifier
UWB	Ultra wideband
VR	Virtual Reality
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
XR	Oberbegriff und Platzhalter für AR, VR und MR

1 Einleitung

1.1 Motivation

Die durchschnittliche Anzahl von Besuchern in Museen ist über die letzten Jahre leicht angestiegen. Dies ist in der folgenden Abbildung 1 zu sehen.

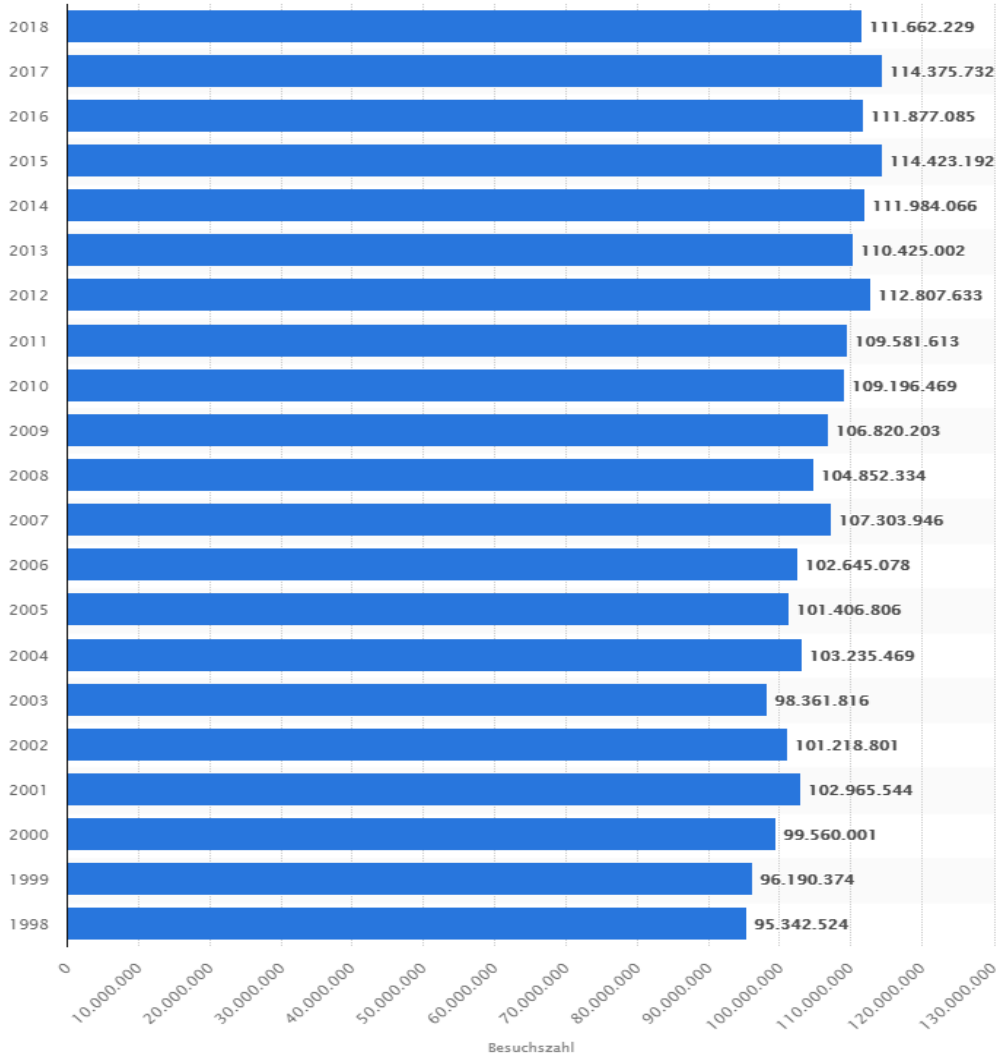


Abbildung 1: Entwicklung der Besuchszahlen in deutschen Museen von 1998 bis 2018

Quelle: [Inte01]

Um die Möglichkeiten der heutigen Zeit besser zu nutzen und um weitere Besucher ins Museum zu locken, gehen viele Museen den Schritt der Digitalisierung. Dieser Schritt erfolgt meist in Form von neuen und moderneren Webseiten, oder museumseigener Apps für verschiedene mobile Endgeräte. Mit Hilfe dieser Option steht dem Besucher vor Antritt eines Museumsbesuchs die Möglichkeit zu, sich ausgiebig zu informieren.

Zusätzlich können interaktive Apps in dem jeweiligen Museum als Unterstützung für den Museumsrundgang hinzugenommen werden. Dadurch wird auch einzelnen Personen eine Führung geboten, die dafür normalerweise nicht das nötige Budget aufbringen können und Museen, welche Personalmangel und somit nicht genügend Führungen anbieten können werden ebenso entlastet ohne dass der Besuch des Museumsgängers negativ beeinträchtig wird.

Das ganze Konzept muss nicht auf Museen beschränkt werden. Es bietet sich ebenso für Messen, Ausstellung und weitere kulturelle Angebote an. Somit existiert für den Besucher eine individuelle, aber eine interaktivere Option, um an der gewählten Veranstaltung teilzunehmen.

1.2 Ziel und Aufbau der Arbeit

Das übergeordnete Ziel dieser Abschlussarbeit ist es, zu klären, wie ein erfolgreicher und funktionstüchtiger Prototyp erstellt werden kann. Dazu wird besonders auf die Voraussetzungen eingegangen, als auch auf die erarbeiteten Konzepte für die einzelnen Komponenten des Programms. Dies umfasst die Augmented – Reality – Komponente und die Charakter – und Indoor – Navigations – Komponente.

Mit dem Hauptfokus der Arbeit auf das Thema Augmented Reality lassen sich drei Leitfragen formulieren:

- Wieso bietet sich eine Augmented Reality App für ein Museum an?
- Welche technischen Möglichkeiten eignen sich für die Implementierung von Augmented Reality?
- Kann mit Hilfe von Augmented Reality Indoor Navigation ermöglicht werden?

Diese Leitfragen legen den Fokus für einen erfolgreichen Prototypen.

Durch einen funktionstüchtigen Prototypen ergibt sich eine neue Leitfrage:

- Inwieweit ist der Prototyp mit verschiedenen mobilen Endgeräten nutzbar und welche Software Anforderungen stellt er an diese?

Um diese Fragen zu beantworten, muss zu Beginn der Arbeit die benötigte Software ausführlich analysiert werden, ob alle nötigen Anforderungen für die erfolgreiche Erstellung eines Prototypen gegeben sind.

Zu Beginn der Arbeit wird in Kapitel 2 auf die benötigten Vorkenntnisse für die Erstellung eines Prototypen eingegangen. Dieses Kapitel legt den Grundstein und die Voraussetzung für ein tieferes Verständnis für die folgenden Erläuterungen. Der Abschnitt „Grundlagen“ ist in die zwei Unterpunkte „Augmented Reality“ und „Indoor Navigation“ untergliedert. In dem Unterpunkt „Augmented Reality“ wird der Leser in die Thematik der Augmented Reality (AR) im Allgemeinen eingeführt. Vertiefend zu diesem Abschnitt werden verschiedene Möglichkeiten zur Nutzung von Augmented Reality für verschiedene mobile Endgeräte vorgestellt. Das zweite Unterkapitel führt den Leser in die „Indoor Navigation“ ein. In dem Unterpunkt „Indoor Navigation“ wird der Leser mit der Thematik Indoor Navigation und ihren Möglichkeiten vertraut gemacht.

In Kapitel 3 „Konzeptentwicklung“ werden die Konzepte für die Komponenten entwickelt. Im Unterpunkt 3.1 wird die Anforderungsanalyse behandelt. Darin wird geklärt, was das Programm können muss, um in der Praxis Verwendung zu finden. Dabei beschäftigt sich das erste Unterkapitel mit allgemeinen Informationen und verschiedenen Standards zur Erstellung eines Lastenheftes in Vorbereitung auf eine gewünschte Software. Im zweiten Unterkapitel dreht sich alles um konkrete Funktionen, die die Anwendung leisten soll. Dies sind die funktionalen Anforderungen. Danach folgen mit Unterkapitel 3 die nicht-funktionalen Anforderungen, die die Leistungen des Systems weiter vervollständigen. Als letztes wird unter diesem Unterpunkt die benötigte Hard- und Software festgelegt. In Kapitel 3.2 wird Unity als Software Ansatz behandelt. Darin wird diese vorgestellt und die Frage geklärt, warum bietet sich vor allem diese Software für die Entwicklung einer AR App an.

In Kapitel 4 „Implementierung & Test“ werden die beiden Komponenten des Prototyps erläutert und begründet. Im Unterpunkt 4.1 wird die Augmented – Reality – Komponente behandelt und im Unterpunkt 4.2 die Charakter – und Indoor – Navigations – Komponente.

Im fünften Kapitel wird ein Fazit gezogen, wie erfolgreich der Prototyp umgesetzt werden konnte. Zusätzlich behandelt dieses Kapitel den Ausblick. Dieser zeigt mögliche Verbesserungen der integrierten Funktionen oder auch zusätzliche Funktionen, welche über den Rahmen dieser Bachelorarbeit hinausgehen.

In dieser Arbeit wird auf folgende Punkte nicht eingegangen:

- Im Abschnitt „Grundlagen“ werden nur die notwendigen Kenntnisse über Indoor Navigation dargestellt. Durch Augmented Reality findet dieser Teil Anwendung in dem erstellten Programm.
- Für die Softwareentwicklung werden nur die Grundzüge erläutert. Detaillierte Betrachtungen zu den Architekturmustern und zu den Entwurfsmustern entfallen.
- Für die Darstellung der Indoor Navigation für den Nutzer wird ein Charakter verwendet. Dieser wird in vereinfachter Form und ohne ausgeprägte Animationen dargestellt.

2 Grundlagen

2.1 Augmented Reality

2.1.1 Einführung Augmented Reality

Augmented Reality (AR) beschreibt die „computergestützte Erweiterung der Realitätswahrnehmung“ [Wiki01]. Demnach ergibt sich im Deutschen der Ausdruck „Erweiterte Realität“. Zum besseren Verständnis werden zwei bekannte Beispiele betrachtet.

Beispiel 1: Im Fernsehen läuft eine Fußball-Übertragung und es erfolgt gerade ein Freistoß. Für den Zuschauer wird nun eine Entfernungsanzeige eingeblendet, ein sogenanntes virtuelles Objekt mit einer Zusatzinformation. Sollte ein Spieler bei einem Pass im Abseits stehen, dann wird meistens eine virtuelle Linie gezogen, um diese Situation dem Zuschauer besser darzustellen. Diese Linie entspricht ebenso einer Zusatzinformation, die für ein besseres Spielverständnis sorgt.

Beispiel 2: Betrachtet wird die App „Pokemon Go“ von Niantic. Dort werden wilde Pokemon mit Hilfe der Kamera des mobilen Endgeräts in die „Realität“ gebracht. Die Pokemon werden zwar nur auf dem Bildschirm angezeigt, aber durch die Verschmelzung mit dem Hintergrund wirken sie real.

Die AR ist ein Teilgebiet der Mixed Reality. In der folgenden Abbildung wird es verdeutlicht:

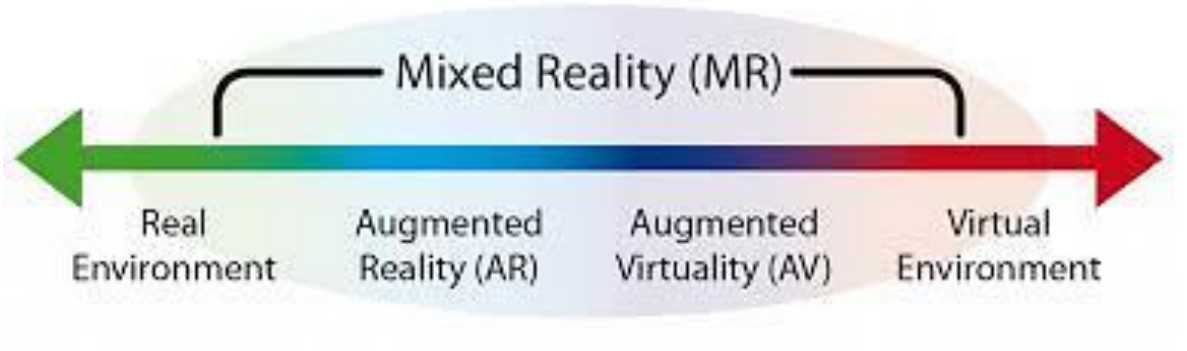


Abbildung 2: Realitäts – Virtualitäts – Kontinuum nach Milgram

Quelle: [Micr01]

Die Abbildung zeigt das Realitäts – Virtualitäts – Kontinuum nach Milgram. Die beiden äußeren Extreme bilden dabei das Gerüst seiner Behauptung. Auf der linken Seite befindet sich unsere reale Umgebung und auf der rechten Seite die virtuelle Umgebung. Die Frage, die sich nun stellt, lautet, was ist real beziehungsweise entspricht der Realität? Diese Frage alleine könnte eine ganze Bachelorarbeit in Anspruch nehmen und deswegen wird die Realität in dieser Arbeit als etwas, was mit den Sinnen des Menschen wahrgenommen werden kann[1], betrachtet. Dazu zählen die fünf Sinne Riechen, Schmecken, Fühlen, Sehen und Hören.

Bei der virtuellen Realität (VR), das andere Extrem, wird der Anwender durch Hard- und Software in eine virtuelle Umgebung „projiziert“, dafür wird meist der Sinn des Sehens angesprochen. Dies erfolgt über eine VR – Brille. Auch der Sinn des Hörens findet in der virtuellen Realität Anwendung. Dafür wird zusätzliche Hardware, meist in Form von Kopfhörern verwendet. Ein weiterer dargestellter Sinn ist der Tastsinn (Fühlen). Mit Hilfe von Controllern, die an die VR – Brille angeschlossen sind können virtuelle Objekte gegriffen werden. Dabei kann der Tastsinn aber nicht komplett angesprochen werden, denn bisher ist es noch nicht möglich mit den Controllern die Oberflächen der virtuellen Objekte wiederzugeben.

Zwischen diesen beiden Extremen befindet sich die Mixed Reality (MR). Diese unterteilt sich in AR und Augmented Virtuality (AV) und beschreibt das Verschmelzen unserer Realität mit der virtuellen Realität. Die AV und ihre Eigenschaften liegen demnach näher an der VR und die AR orientiert sich näher an der realen Umgebung. Bei der AV befindet sich der Nutzer immer noch in einer virtuellen Umgebung, aber diese Umgebung wird mit Objekten oder Personen aus der realen Umgebung erweitert. Somit existiert eine Interaktionsmöglichkeit zwischen der virtuellen und der realen Umgebung. Bei der AR ist es demnach umgekehrt. Dort befindet man sich in der Realität und nutzt Hardware, z.B. Handys oder Head-up-Displays, um zusätzliche Informationen darzustellen. Dafür wird die eingebaute Kamera verwendet, um die reale Umgebung auf dem Display abzubilden. In der verwendeten Software können außerdem Zusatzinformationen für verschiedene Objekte hinterlegt werden.

---

[1] Dies entspricht der Meinung des Autors dieser Arbeit und kann gegebenenfalls mit anderen wissenschaftlichen Betrachtungen abweichen.

Sobald die Kamera ein bestimmtes Objekt erkennt, im Museum wäre es ein bestimmtes Ausstellungsstück, können für dieses bestimmte Zusatzinformationen angezeigt werden, für die sonst kein Platz wäre.

Für AR Applikationen bieten mobile Endgeräte bisher die beste Möglichkeit, da die Hardware alle nötigen Bauteile hat. Dazu zählt nicht nur die Kamera, auch wenn diese das wichtigste ist, sondern auch andere, wie z.B. das eingebaute Gyroskop, welches wichtige Umgebungsdaten liefert.

Neben den mobilen Endgeräten gibt es auch für die AR spezielle Hardware, wie für die VR. Entwickelt wurden viele verschiedene AR – Brillen für eine leichtere und effektivere Verwendung von Augmented Reality. Dabei gab es nicht immer nur positive Verbesserungen, denn viele Forschungen auf diesem Gebiet wurden auch wieder verworfen, wie . Trotzdem gab es ein paar erfolgreiche Projekte, wie z.B. von Microsoft, die HoloLens [vgl. MicrHolo01] oder von Magic Leap, die Magic Leap One [vgl. MaLeap01]. Mit einem Preis von 3.500 $ [vgl. MicrHolo02] ist die HoloLens als Anwendungsmöglichkeit recht teuer und somit, vor allem für Museen, die viele Geräte benötigen, finanziell nicht umsetzbar.

2.1.2 Bedeutung von Augmented Reality für Museen

Eine Augmented Reality App für ein Museum sollte nicht als eine eigenständige Sache betrachtet werden, sondern eher als unterstützendes Element. Laut Definition ist „ein Museum eine dauerhafte Einrichtung, die keinen Gewinn erzielen will, öffentlich zugänglich ist und im Dienst der Gesellschaft und deren Entwicklung steht. Sie erwirbt, bewahrt, beforscht, präsentiert und vermittelt das materielle und immaterielle Erbe der Menschheit und deren Umwelt zum Zweck von Studien, der Bildung und des Genusses.“ [Icom01]

Demnach sollen zusätzliche Angebote die Möglichkeiten für weiteres Wissen zu einzelnen Ausstellungsstücken erweitern. Der Platz für die Exponate in einem Museum ist begrenzt, damit zeitgleich auch die Möglichkeit, Wissen über das jeweilige Element zu vermitteln. Meistens werden die Daten des Objektes und die wichtigsten Informationen gemeinsam dargestellt. Zusätzliche und ausführlichere

Hinweise finden sich meist auf der museumseigenen Homepage, in Büchern oder im Internet. Jedoch ist der Zugang dazu umständlich und oft nicht direkt möglich.

Mit einer AR App für das jeweilige Museum kann dieser Schritt vereinfacht werden. Damit bietet man dem Nutzer eine leichtere und interaktive Möglichkeit für weitere Informationen bei einem Museumsbesuch. Auf dem Display eines mobilen Endgerätes können wesentlich mehr Daten zu den bestimmten Exponaten dargestellt werden. Mit einem Klick auf das Objekt (ein Anwendungsbeispiel für die App) öffnet sich ein Textfenster mit vielen zusätzlichen Informationen, die auf den Tafeln neben dem Ausstellungsstück keinen Platz gefunden haben. Hier kann man für die besonders wissenshungrigen Besucher auch gleich Links aus der App heraus für Seiten mit noch ausführlicheren Informationen hinterlegen. Somit wären alle Informationen in einem Klick erreichbar.

Eine weitere Option die die AR App bietet, ist die Anschaulichkeit der Exponate. Durch das Darstellen auf dem Display hat der Nutzer die Möglichkeit, das Objekt viel genauer und näher betrachten zu können, so dass der Glaskasten, der dem Schutz des Objektes dient, kein Hindernis mehr darstellt. Die meisten Ausstellungsstücke sind aus Sicherheitsgründen großzügig abgesperrt und somit sind auch kleine Feinheiten leicht zu übersehen. Auf dem Display kann man das Objekt vergrößern, vielleicht sogar drehen, so dass man es von allen Seiten betrachten kann. Dabei wird die Interaktion mit den Exponaten besonders hervorgehoben.

Viele starten einen Museumsbesuch und wissen dabei gar nicht, an welcher Stelle genau sie anfangen sollen. Eine Führung durch das Museum kann der Mehrheit nicht geboten werden, denn viele sind nur als Einzelpersonen oder als Familie da, aber nur größeren Gruppen wird die Möglichkeit geboten, an einer Führung teilzunehmen. Das ist eigentlich immer eine Frage der Personalverfügbarkeit. Mit Hilfe von Augmented Reality und Indoor Navigation kann man in der App jeder Person, mit einem mobilen Endgerät, die Möglichkeit bieten, an einer Führung durch das Museum teilzunehmen.

Damit steht auch der letzte Punkt dieses Unterkapitels im Fokus. Fast jede Person besitzt ein mobiles Endgerät in Form eines Smartphones. Ist die App auf vielen Geräten lauffähig, dann spart sich das Museum einige Kosten zur Beschaffung von

nötiger Hardware, aber kann trotzdem eine zusätzliche Möglichkeit für Besucher anbieten.

Eine Augmented Reality App ist somit kein Muss, aber eine sinnvolle Erweiterung für die Interaktion zwischen Museum und Besucher.

2.1.3 Augmented Reality Möglichkeiten für mobile Endgeräte

In diesem Abschnitt werden vier Möglichkeiten für die Entwicklung von AR Apps auf mobilen Endgeräten betrachtet. Manche davon laufen auf verschiedenen Betriebssystemen und sind plattformunabhängig, aber es werden auch zwei betrachtet, die nur auf ihrer jeweiligen Plattform laufen. Die vier Varianten heißen ARCore, ARKit, Vuforia und ARFoundation[2].

ARCore ist der erste Vertreter für eine Plattformabhängigkeit. Entwickelt wird ARCore von Google. Die Software läuft auf dem eigenen Betriebssystem Android. Zwar ist es auf dieses beschränkt, aber durch die hohe Nutzung von Smartphones, welche Android als Betriebssystem verwenden, ergibt sich dadurch kein Nachteil.

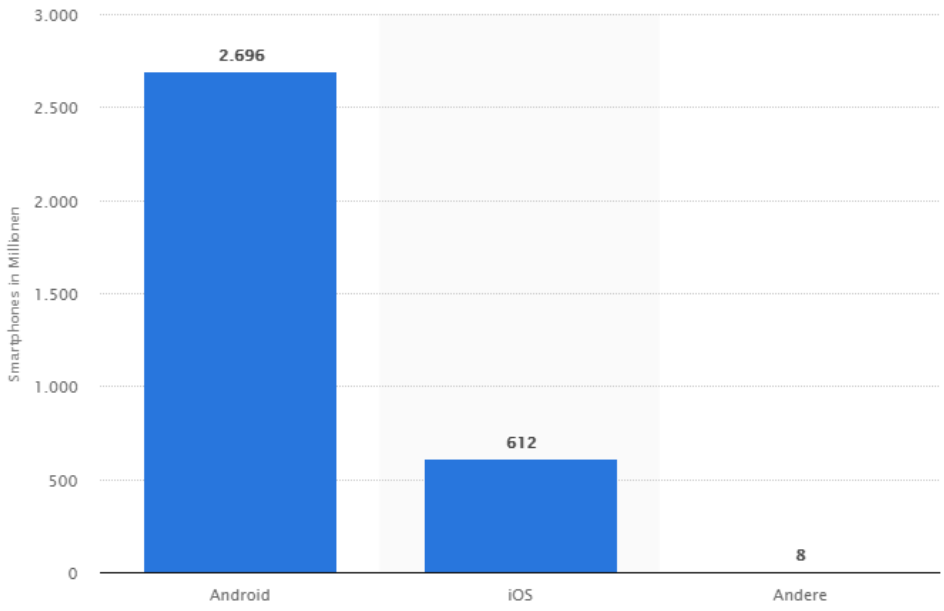


Abbildung 3: Anzahl der in Gebrauch befindlichen Smartphones weltweit nach Betriebssystem im Dezember 2017 (in Millionen)

Quelle: [Inte02]

[2] Diese vier wurden aufgrund eigener Erfahrungen bei der Programmierung mit AR ausgewählt.

Neben der Voraussetzung von Android als Betriebssystem mit fast 2,7 Milliarden mobiler Endgeräte, gibt es weitere Bedingungen für die erfolgreiche Nutzung von ARCore. Auf der Seite der Hardware benötigt das Smartphone oder Tablet eine funktionstüchtige Kamera. Das Betriebssystem Android muss mindestens Versionsnummer 7.0 haben, teilweise auf verschiedenen Geräten auch mindestens 8.0. Außerhalb von China benötigt es den Google Play Store, innerhalb von China ist es auf den dortigen Plattformen erhältlich.

Sind aber alle Anforderungen erfüllt, dann läuft es auf einer Vielzahl von Geräten. Die eigenen Google Produkte, wie das Nexus 5X, die Pixelreihe, Huawei, LG, Samsung, Sony und auch viele andere unterstützen ARCore.

Mit Hilfe der in der Hardware verbauten Teile, wie die Kamera, GPS oder das Gyroskop bietet Google mit ARCore viele verschiedene Möglichkeiten zur Nutzung von Augmented Reality. Unterstützt werden „Motion Tracking“, „Environmental understanding“, „Light estimation“, „User interaction”, „Oriented Points, „Anchors and trackables”, „Augmented Images” und „Sharing”. [vgl. Arc01]

Bei „Motion Tracking“ wird die Kamera des Smarphones oder Tablets genutzt, um visuell einzigartige Punkte festzustellen und mit diesen Punkten zu berechnen, wie sich die Position des Gerätes zu diesen Punkten verändert. Dabei kommen das GPS und das Gyroskop zum Einsatz, um Position und Richtung über einen bestimmten Zeitraum zu schätzen. Dadurch können effektiv virtuelle Elemente auf die real vorhandenen Objekte auf dem Display angezeigt werden. Dabei gilt der Rand eines realen Objekts als Rand für das virtuelle Objekt. [vgl. Arc01]

Dies spiegelt sich auch im zweiten Punkt „Environmental understanding“ wieder. Denn hier werden mit den einzelnen Punkten Cluster aus Punkten und Ebenen geformt, sogenannte „planes“. Diese Ebenen entsprechen flachen Oberflächen, wie die Wände eines Raums oder einer Tischplatte. Dabei besitzen sie feste Grenzen, um die Positionen von virtuellen Objekten genauer darzustellen. [vgl. Arc01]

„Light estimation“ beschreibt, wie der Name schon sagt, die Darstellung der Lichtverhältnisse der realen Umgebung mit Hilfe der Kamera auf dem Display, um die Realität der virtuellen Objekte besser beschreiben zu können. [vgl. Arc01]

Der Unterpunkt „User interaction“ beinhaltet die Interaktion mit den virtuellen Objekten. Der Nutzer berührt mit seinem Finger sein Display und dabei wird eine x,y Koordinate gespeichert, ausgehend von der Position des Fingers auf dem Display. Von dieser Koordinate aus wird ein imaginärer Strahl in die reale Umgebung „projiziert“. Dieser gibt alle Punkte, Ebenen oder Objekte wieder, mit denen er interagiert und liefert ebenso die Position dieser an das Endgerät. Diese Elemente können dann ausgewählt und für Interaktionen genutzt werden. [vgl. Arc01]

Unter dem Begriff „Oriented Points“ werden virtuelle Objekte beschrieben, welche an gewinkelten Oberflächen positioniert sind. Beim Zurückliefern eines bestimmten Punktes überprüft ARCore die Ausrichtung zu anderen Punkten, um so einen Winkel des virtuellen Elements bestimmen zu können. Bei Oberflächen ohne großartige Strukturen, z. B. eine weiße Wand, kann es bei der Bestimmung der einzigartigen Punkten zu Problemen kommen, da diese nicht immer zu 100 % erfolgreich erkannt werden können. [vgl. Arc01]

„Anchors and trackables“ sind dafür verantwortlich, dass virtuelle Elemente über einen längeren Zeitraum gespeichert und gemerkt werden. Dafür wird jedem Element ein Anker, also eine bestimmte Position in der realen Umgebung zugewiesen. Dadurch kann auch nach dem Verlassen des Sichtfelds der Kamera das Element wieder seiner ursprünglichen Position zugewiesen werden, sollte diese Position wieder ins Blickfeld der Kamera fallen. [vgl. Arc01]

Die vielen Punkte und Ebenen, die bestimmt werden, sind in ARCore Objekte von einem bestimmten Typ namens „trackable“. Das bedeutet, dass diese Elemente über den gesamten Zeitraum von der Anwendung der App mit ihrer Position gespeichert und hinterlegt werden können, um dadurch virtuelle Objekte besser an die reale Umgebung anpassen zu können. Um eine hohe CPU Auslastung zu minimieren, sollten nicht mehr benötigte „trackables“ gelöscht werden.

Eine weitere Möglichkeit von ARCore ist das Nutzen von 2D Bildern als Ausgangspunkt für virtuelle Objekte. Der Begriff „Augmented Images“ beschreibt daher, wie ARCore solche Bilder, wie Verpackungen von Produkten oder Filmposter als Referenzanker für hinterlegte virtuelle Elemente benutzt.

Diese Bilder können vor der Benutzung der App in eine Bilderdatenbank integriert werden, aber sie können auch während der Anwendung aktiv aufgenommen und gespeichert werden. Dafür erkennt dann ARCore automatisch die gewählten Bilder, ihre Grenzen in Form von Länge und Höhe und speichert ihre Position in der realen Umgebung. [vgl. Arc03]

Der letzte unterstütze Punkt „Sharing“ steht für das gemeinsame Nutzen einer einzigen Kamerasicht auf vielen verschiedenen mobilen Endgeräten. Dabei zeigt jedes Display die gleichen virtuellen Elemente, die in der Realität von einem der verschiedenen Geräte platziert worden ist. [vgl. Arc01]

Durch die vielen Merkmale und Möglichkeiten und durch die Masse an mobilen Endgeräten mit dem Betriebssystem Android hat ARCore für die Verwendung von Augmented Reality eine große Bandbreite von Nutzern, um aktiv AR Apps einsetzen zu können.

Das Gegenteil zu ARCore für Android bildet ARKit für iOS [vgl. Ark01]. ARKit wird von Apple für ihr eigenes Betriebssystem entwickelt und unterstützt alle hauseigenen iPhones ab SE und alle eigenen iOS Tablets. Die Voraussetzung dafür ist das iOS Betriebssystem mit mindestens Version 11.

Viele der zuvor ausführlich erklärten Merkmale für ARCore bietet auch ARKit an, teilweise unter anderer Bezeichnung, aber mit der gleichen Bedeutung. So ermöglicht ARKit das Speichern von virtuellen Objekten in der realen Umgebung für eine spätere Nutzung. Erkannt werden 2D-Bilder, 3D-Objekte und Ebenen. Auf diesen lassen sich virtuelle Elemente platzieren und diese interagieren mit der realen Umgebung, so dass beide Grenzen voneinander abhängig sind. Verschiedene iOS Geräte können für die gleiche AR App und für das gleiche Erlebnis genutzt werden. (siehe „Sharing“)

Die Anzahl der Funktionen stimmen weitestgehend überein. Sie bieten demnach nur eine optimale Version für das jeweilige Betriebssystem Android oder iOS.

Die dritte Variante läuft sowohl auf Android als auch auf iOS unter dem Namen Vuforia. Vuforia ist ein SDK für die Entwicklung von AR Apps von der Firma PTC Inc. Der große Unterschied zu ARCore oder ARKit liegt hinter der Anwendung von Vuforia Apps. Hier wird nicht die komplette reale Umgebung bestimmt und genutzt, sondern nur bestimmte Objekte aus der Realität. Dies können 2D-Bilder oder 3D-Objekte sein, welche entweder mit oder ohne Marker in einer Bilderdatenbank in der App gespeichert werden. Bei einem Bild mit Marker wird das virtuelle Objekt sofort projiziert, sobald die Kamera das Bild, also den Marker, ausreichend im Blickfeld hat. Dieses virtuelle Element kann dann mit verschiedenen Interaktionen versehen werden, so dass der Nutzer es beispielsweise drehen oder vergrößern kann. Ein reales Objekt ohne Marker funktioniert ähnlich, nur ist das Platzieren von virtuellen Elementen freier und nicht so sehr eingeschränkt, wie bei der Variante mit Marker.

Abbildung 4: Eine Vuforia AR-App im Einsatz

Quelle: [Arvuf02]

Durch die Möglichkeiten, die Vuforia bietet, vor allem in der Kombination mit der Microsoft HoloLens, ist Vuforia besonders im industriellen Sektor bei mechanischen Detailarbeiten beliebt. Der größte Unterschied zu ARCore und ARKit ist jedoch der Kostenpunkt. Während ARCore und ARKit kostenlos nutzbar sind, wird für die Erstellung und Nutzung einer Vuforia App einmalig ein Preis von 499$ gefordert. [vgl. Arvuf03]

Die letzte bekanntere Möglichkeit zur Entwicklung von AR Apps bietet AR Foundation. Diese Variante wird von Unity Technologies für ihre eigene 3D-Entwicklungsumgebung mit dem Namen „Unity“ entwickelt. Da Unity eine eigene Engine verwendet und somit Apps für sowohl Android als auch iOS anbieten kann, bietet AR Foundation eine gute Brücke zur Programmierung für beide Plattformen. Mit Hilfe des ARKit XR Plugin‘s oder des ARCore XR Plugin’s können die jeweiligen Plattformen mit ihren besonderen AR Funktionen angesprochen werden.

Jedoch befindet sich AR Foundation noch in einer recht frühen Phase der Entwicklung und kann damit nicht alle möglichen Funktionen nutzen, die ARCore auf Android bzw. ARKit auf iOS anbieten. Dennoch ergibt sich damit eine gute Schnittstelle für die beiden verschiedenen Betriebssysteme. [vgl. Arfnd01]

2.2 Indoor Navigation

„Der Begriff Indoor Navigation bezeichnet eine Navigation innerhalb von Gebäuden.“ [InNav01] Bekannt ist der Ausdruck auch unter den Bezeichnungen „Indoor positioning system“ (IPS) oder Indoor Routing. In der deutschen Übersetzung findet es sich als innerräumliche Navigation wieder. Geht man nach dem englischen Begriff, dann ist das Gegenteil zur Indoor Navigation die Outdoor Navigation. Auch bekannt als „Global positioning system“ (GPS). Dieses System nutzt Satelliten aus dem Weltraum für die genaue Bestimmung einer Position. Dafür werden Signale von mindestens vier verschiedenen Satelliten an das GPS Gerät des Benutzers gesendet, um so eine eindeutige Standortbestimmung zu ermöglichen. [vgl. Gps01] Je mehr Satelliten für die Bestimmung herangezogen werden, desto genauer wird die Position ermittelt. So wird die hohe Ungenauigkeit durch viele verschiedene Fehlerquellen, wie Satellitenfehler oder atmosphärische Störungen abgeschwächt, damit eine Genauigkeit von bis zu 10 Metern erzielt werden kann.

Diese Art der Ortung benötigt direkten Sichtkontakt zu den bestimmenden Satelliten, daher kann GPS für Indoor Navigation nicht ohne weiteres genutzt werden. Google bietet mit „Google Indoor Maps“ eine Möglichkeit an, um weiterhin auf GPS aufzubauen, aber hier werden zusätzlich WLAN-Netzwerke der Umgebung mit herangezogen. Bezeichnet werden sie als Assisted GPS (A-GPS).

Jedoch hat auch dieses System seine Vor- und Nachteile. Je mehr Sender vorhanden sind, desto genauer wird das Ergebnis der Standortermittlung werden. Eine Genauigkeit von bis zu 10 Metern soll erreicht werden. Doch die Praxis sieht anders aus, denn zum einen lässt die Genauigkeit sehr zu wünschen übrig, zum anderen gibt es bei Gebäuden mit mehreren Etagen ein Problem, das richtige Stockwerk anzuzeigen. [vgl. InNav02]

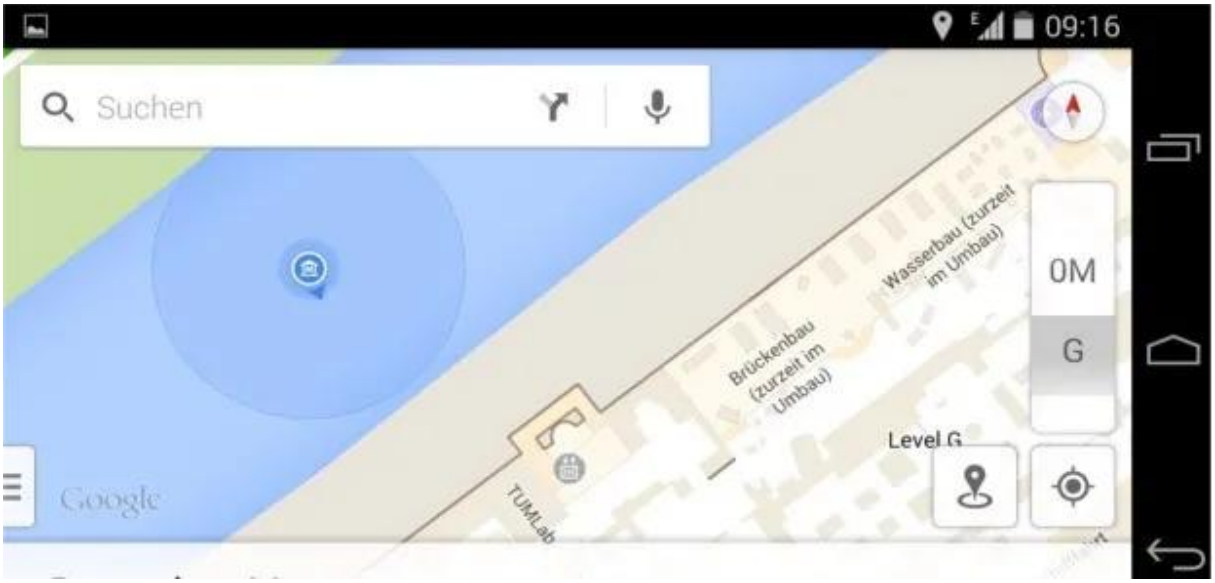


Abbildung 5: Standortbestimmung mit Google Indoor Maps für

das Deutsche Museum in München

Quelle: [InNav02]

Die Abbildung 5 zeigt das Genauigkeitsproblem von Google Indoor Maps. Hier wird der Besucher, der sich eigentlich im Gebäude des Deutschen Museums befinden sollte, außerhalb in der Isar platziert.

Die Verwendung von WLAN-Netzwerken ist aber nur eine von vielen Möglichkeiten für die Nutzung von Indoor Navigation. Hierfür kommen auch die zuvor genannten IPS mit verschiedenen anderen Ortungsmethoden zur Anwendung. Als Ortungsmethoden stehen Bluetooth Low Energy (BLE), Ansätze und Lösungen auf Basis von Ultra-wideband (UWB) oder passivem RFID zur Verfügung. [vgl. InNav03]

Für die Bluetooth Ortungsmethode BLE werden so genannte Beacon verwendet. Beacon sind ortsfeste Navigationsfunkstellen.

Bei „Bluetooth Low Energy handelt es sich um einen universellen Funkstandard mit einem geringen Stromverbrauch, sodass eine einfache Verbindung von beliebigen Produkten mit einem Smartphone oder Tablet gewährleistet ist.“ [Ble01]

Damit ergibt sich eine kostengünstige Möglichkeit zum Aufbau von Wireless Personal Area Networks (WPAN) in verschiedenen Bereichen, wie z.B. Gesundheit, Medizin oder Autoelektronik. Dieser Standard existiert in den Versionen 4.0, 4.1 und 4.2. BLE wurde als erstes von Nokia entwickelt und genutzt und später mit dem klassischen Bluetooth zusammengeführt. Zwischenzeitlich wurde es auch als „Bluetooth Smart“ oder „Bluetooth Smart Ready“ bezeichnet. Viele der heutigen mobilen Endgeräte verwenden diese Art von Standard. Jedoch legen wenige Hersteller ihre Übertragungsprotokolle offen, dadurch werden Verbindungen von beliebig vielen Geräten stark eingeschränkt und lassen sich daher meist nicht miteinander verbinden. Dieses Problem kann aber mit einer App für das mobile Endgerät, das mit den anderen Geräten kommuniziert, gelöst werden. [vgl. Ble02]

Als Sender für diese Technik werden Beacon genutzt. Diese schicken in regelmäßigen Zeitabständen Bluetooth-Signale an den Empfänger, z.B ein Smartphone. Diese Signale enthalten relevante Daten für die Ortsbestimmung. Ein Beacon ist ein passives Bauelement, denn es kann nur Signale senden, aber nicht empfangen. Ein einziges Signal enthält drei Informationszustände:

- Eine einzigartige und eindeutige Kennung, auch bezeichnet als **Unique Identifier** (UID), bestehend aus Zahlen und Buchstaben und von einer Gesamtgröße von 16 Byte.
- Einen **Major-Wert**, der die Funktion hat, Signalregionen oder Use-Cases zu definieren. Hierbei liegt der Fokus auf dem generellen Ort des Signals, d. h., ein Gebäude einer Firma in Berlin würde einen anderen Major-Wert senden, als ein Gebäude der gleichen Firma in Hamburg.
- Einen **Minor-Wert**, der das Gebiet des Major-Werts in verschiedene Subregionen oder Use-Cases unterteilen kann. Für das Beispiel der Firma können einzelne Abteilungen, wie die Software-Abteilung, die Hardware-Abteilung oder die Geschäftsebene als verschiedene Abteilungen erkannt werden.

Sowohl der Major- als auch der Minor-Wert sind jeweils 2 Byte groß. Jedes Signal sendet daher maximal 20 Byte für eine genaue Positionsbestimmung. Für die Verarbeitung der Signale wird die installierte App herangezogen. Diese nimmt die Signale vom Sender auf und misst den Abstand vom Sender zum Empfänger. Für eine genaue Positionsbestimmung im dreidimensionalen Raum werden mindestens vier Beacon benötigt. Beim GPS sind auch die Signale von mindestens vier Satelliten zur genauen Standortbestimmung nötig. Die Berechnung der Abstände erfolgt nach dem Laterations-Prinzip [vgl. LatPri01] und ist auf einen Nahbereich ausgelegt.

Die Position des Empfängers zu den einzelnen Beacon wird über vier Stufen angegeben:

- Stufe 1: Immidiate (weniger als 50 cm)
- Stufe 2: Near (bis drei Meter)
- Stufe 3: Far (mehr als zehn Meter)
- Stufe 4: Unknown (Unbekannt)

Manche Beacon können Signale bis zu einer Reichweite von 450 Meter senden. Jedoch wird die Reichweite durch Hindernisse stark verkürzt. Als Hindernisse gelten Menschen, Mauern, aber auch Einrichtungsgegenstände.

Sind die Positionen für bestimmte Stufen erreicht, dann kann die App weitere Prozesse in Gang setzen. Sobald man sich in einem Museum in der Nähe eines Ausstellungsgegenstandes befindet, kann sich dann in der App ein Fenster mit all den relevanten Informationen zum Ausstellungsstück öffnen. Dadurch wird eine Indoor Navigation mit zusätzlichen Informationen für bestimmte Objekte erstellt. [vgl. Ble03]

„Ultra-wideband (UWB) ist eine Nahbereichs-Funktechnik, die extrem große Frequenzbereiche mit einer Bandbreite von mindestens 500 MHz nutzt.“ [Uwb01] Eingesetzt wird es durch die hohe Genauigkeit von 10 bis 30 Zentimetern, die diese Technik erlaubt. Zusätzlich funktioniert UWB mit einer niedrigen Sendeleistung von 0,5 mW und -41,3 dBm/MHz, sodass keine bereits verfügbaren Frequenzbereiche gestört werden.

Der genaue Frequenzbereich liegt zwischen 3,1 und 10,6 GHz. Die Signale von UWB Sendern können nicht von allen Geräten empfangen werden. Dafür werden spezielle UWB-Empfänger benötigt.

Für den Einsatz dieser Technik existieren zwei mögliche Verfahren. Zum einen das clientbasierte und zum anderen das serverseitige Verfahren, um Asset Tracking zu betreiben. Mit Hilfe von drei verwendeten Anchors wird durch ein Laufzeitverfahren (Time of Flight, ToF) die Position bestimmt. Berechnet wird die Position dabei mit einer Lichtlaufzeit zwischen den einzelnen Anchors und den Assets. Dabei hat jedes Asset seinen eigenen angebrachten Tag, der an die Anchors übermittelt wird. Die erhobenen Daten gelangen als Nächstes zu einem Server und werden dort verarbeitet. Als Daten zählen die ID, ToF und ein timestamp. Nach der Auswertung können die Daten auf ein beliebiges mobiles Endgerät weitergeleitet werden. [vgl. Uwb01]

Durch die speziellen Anforderungen und hohen Kosten von Spezialhardware ist UWB trotz seiner Vorteile nicht sonderlich weit verbreitet. Trotzdem findet es Einsatz, vor allen in großen Fabrikanlagen, wie z.B. der Automobilfertigung.

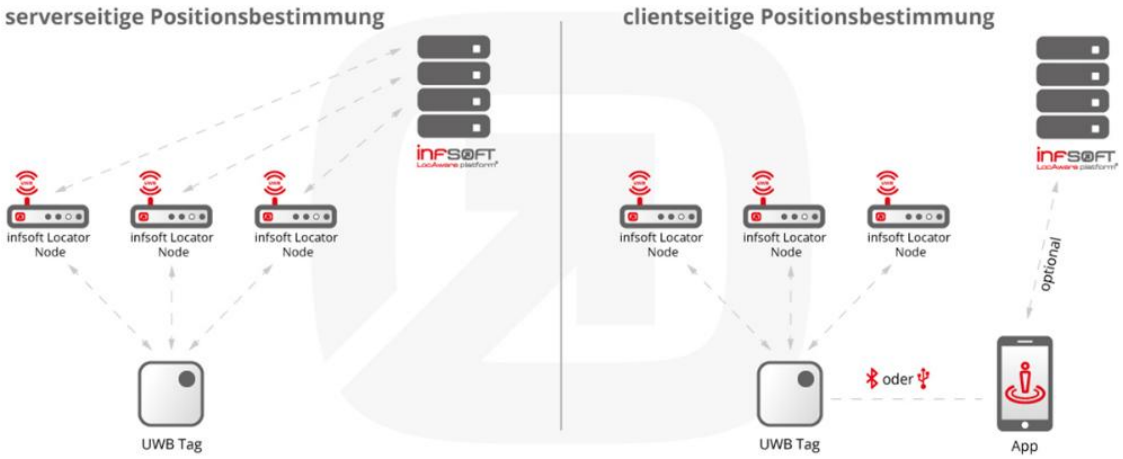


Abbildung 6: Positionsbestimmung bei UWB

Quelle: [Uwb01]

Als weitere Möglichkeit für die Indoor Navigation existiert radio-frequency identification (RFID). Dieses System verwendet Funkwellen zur drahtlosen Übertragung. Gesendet wird die eindeutige ID eines Objektes, z.B. eine Seriennummer. Das entspricht einer punktuellen Ortung, da die Reichweite von RFID-Systemen mit weniger als einem Meter sehr begrenzt ist.

Jedes bestimmbare Objekt hat seinen eigenen Transponder. Mit einem Lesegerät wird jeder Transponder angesprochen. Ebenso gilt das Lesegerät als Stromquelle für die Transponder und versorgt diese über Funkwellen mit Energie. Bezeichnet wird dies als Remote Coupling. Danach werden die ID und weitere Daten erfasst und an einen Server für die Auswertung gesendet. [vgl. Rfid01]

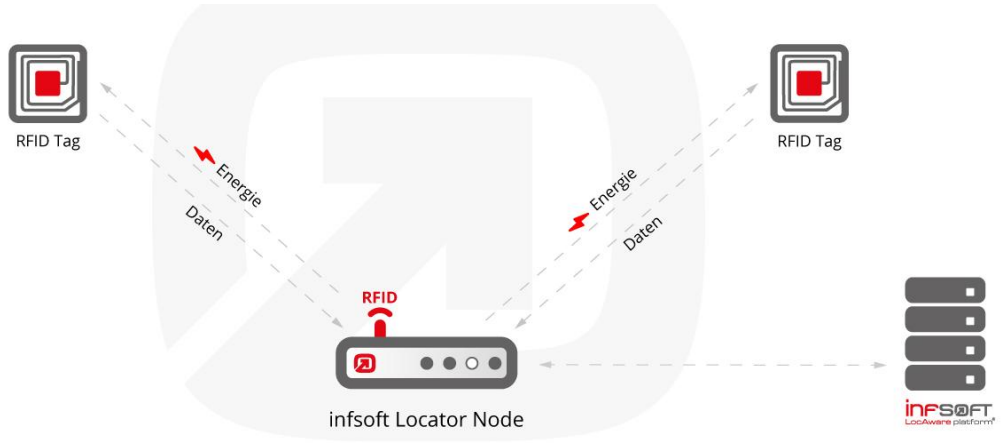


Abbildung 7: Objektidentifikation mit RFID

Quelle: [Rfid01]

Obwohl dieses System seine Vorteile (geringe Kosten und Wartungsfreundlichkeit) hat, besitzt es durch die geringe Reichweite keine besonders großen Aufgabenbereiche. Um das zu kompensieren wird dieses System meist in der Kombination mit einem weiteren System, wie z.B. UWB angewendet.

3 Konzeptentwicklung

3.1 Anforderungsanalyse

3.1.1 Allgemeines

Für jedes zu entwickelnde Softwareprojekt muss vor der Implementierung eine ausführliche Analyse getätigt werden. In dieser Analyse werden alle Aspekte des Projekts erfasst und wiedergegeben. Dies geschieht meist in der Form eines Lasten- und Pflichtenhefts. Dabei wird das Lastenheft von dem Auftragsgeber erstellt und das Pflichtenheft von der beauftragten Firma. Das Lastenheft entspricht daher einem Dokument und beinhaltet alle verschiedenen Anforderungen an das Softwaresystem, die vom Auftraggeber gestellt werden. Zu diesen zählt der Ist- und Soll-Zustand des Projekts. Der Ist-Zustand beschreibt die Analyse und Ermittlung eines aktuellen Problems. Dieses Problem soll in einem Softwareprojekt gezielt umgesetzt werden. Dabei spielt der Soll-Zustand eine wichtige Rolle, da hier das gewünschte Softwaresystem näher beschrieben wird und alle weiteren Anforderungen an das System geklärt werden.

Vor der Anforderungsanalyse kommt daher der Schritt der Anforderungserhebung. Hierzu zählen der Ist- und Soll-Zustand. Durch verschiedene Methoden werden alle benötigten Informationen gesammelt. Die Datenerhebung kann in Form von Interviews oder Fragebögen mit den Mitarbeitern oder durch bloße Beobachtung erfolgen. Zusätzlich existiert die Berichtsmethode, in der ein Mitarbeiter eine Thematik schriftlich erläutert, oder eine Inventurmethode, in der schriftliche Unterlagen studiert und so alle nötigen und möglichen Informationen gesammelt werden. Sollten all diese verschiedenen Methoden trotzdem kein zufrieden stellendes Ergebnis aufweisen, besteht immer noch die Möglichkeit eines Brainstorming-Meetings, um weitere Ideen zu erörtern.

Besonders wichtig ist die Genauigkeit und Verständlichkeit der gesammelten Informationen. Die Sicht auf den Ist-Zustand sollte konsistent sein. Für das System sollte eine klare Zielvorstellung vorhanden sein, die jedoch nicht zu komplex ist, damit eine Umsetzung realisierbar bleibt. Die festgelegten Ziele und Anforderung sollten nur in besonderen Ausnahmefällen verändert werden.

Weiterhin muss die Anforderungsbeschreibung eine gewisse Qualität haben. Wichtig ist dabei die eindeutige und genaue Formulierung der verschiedenen Anforderungen. Fachbegriffe sollten vermieden oder zusätzlich ausführlich erklärt werden, damit es zwischen dem Auftraggeber und der beauftragten Firma zu keinen Unklarheiten und Missverständnissen kommt. [vgl. NormIE01]

Nach der Anforderungserhebung folgt die Anforderungsanalyse und darauf folgt die Anforderungsbeschreibung. Diese erfolgt in einem weiteren Dokument und in diesem werden die gesammelten und geprüften Anforderungen in einer einheitlichen Form, wie z.B. als Anwendungsfall, wiedergegeben und beschrieben. Dieser gesamte Inhalt entspricht dem Pflichtenheft.

Sowohl das Lasten- als auch das Pflichtenheft werden nach dem Standard für Anforderungsdokumente erstellt. Dies entspricht der Norm IEEE 830-98. [vgl. Ieee01] Für die Erstellung eines Anforderungsdokuments schlägt diese Norm drei Kapitel vor.

- Eine Einleitung, die wichtige Meta-Informationen zum Dokument beinhaltet. Darunter zählen der Zweck, Verweise und der Aufbau des Dokuments.
- Eine Allgemeine Beschreibung der Software, in der die wichtigsten Produktfunktionen beschrieben werden. Zusätzlich werden Angaben zu Einschränkungen des Lösungsraums gebracht.
- Eine Erläuterung von spezifischen Anforderungen, die die funktionalen und nicht-funktionalen Anforderungen umfassen. Zusätzlich werden in diesem Kapitel Qualitätsanforderungen, Anforderungen an die Performanz, externe Schnittstellen und sonstiges niedergefasst. [vgl. NormIE01]

Eine ausführlichere Betrachtung liefert Abbildung 8 auf der folgenden Seite 22:

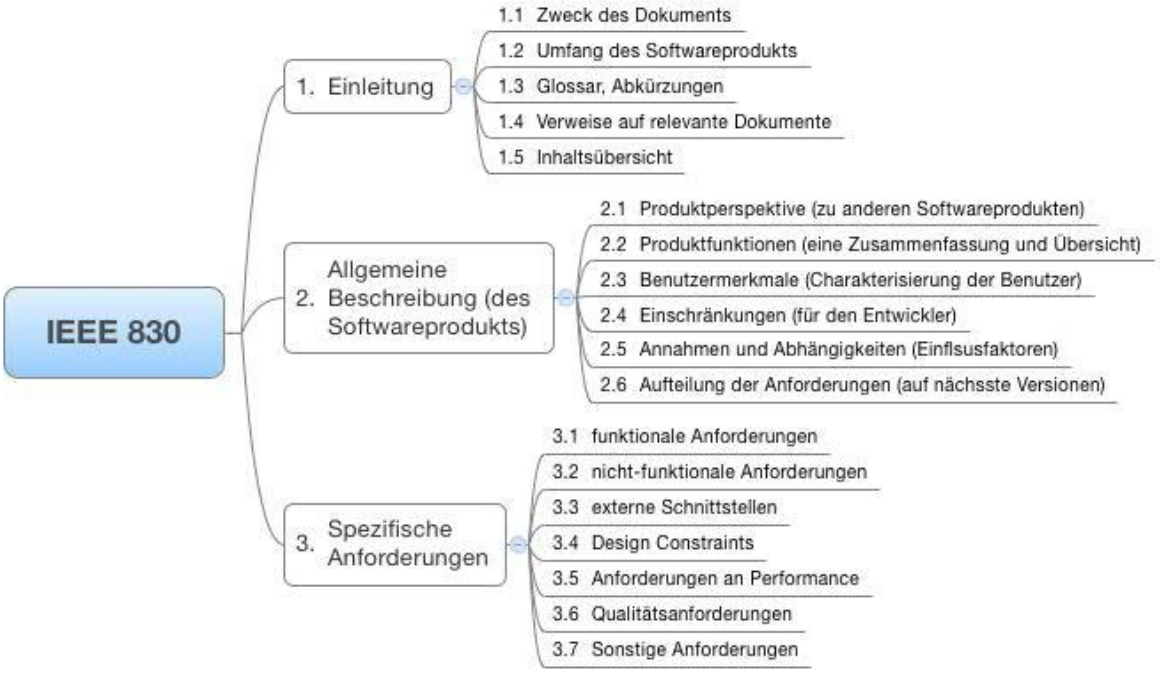


Abbildung 8: Detaillierte Kapitelübersicht nach Norm IEEE 830

Quelle: [NormIE01]

Mit Hilfe des Lastenhefts erfolgt die Anforderungsanalyse, die Vorarbeit für das Pflichtenheft. Hier werden die gesammelten Anforderungen klassifiziert und bewertet. Dies könnte nach Kosten- und Nutzen-Aspekten geschehen. Zusätzlich werden sie verglichen und geprüft. Sind die Anforderungen konsistent und vollständig? Solche Fragen werden in der Anforderungsanalyse betrachtet, da diese beiden Aspekte für die Erstellung des Pflichtenheftes unabdingbar sind.

Die Anforderungen an ein Anwendungssystem teilen sich grundsätzlich in zwei Kategorien ein. Auf der einen Seite stehen die *funktionalen* Anforderungen und auf der anderen Seite die *nicht-funktionalen* Anforderungen.

3.1.2 Funktionale Anforderungen

Funktionale Anforderungen beschreiben den Kern der Anwendungssoftware. Sie beziehen sich auf das, was das System leisten soll. Dabei besitzt jede einzelne Anforderung ihre eigene funktionale Einheit, den Anwendungsfall. Dieser wird in der Informatik auch als Use Case bezeichnet. Dabei dokumentieren Use Cases „die Funktionalität eines vorhandenen oder geplanten Systems mit einfachen Modellen.“ [Ucase01]

Es existieren zwei Arten von Use Cases. Zum Einen gibt es das Black-Box Use Case und zum Anderen das White-Box Use Case. Dabei zeigt das Black-Box **was** ein System leisten soll, also wie es von außen betrachtet aussieht, aber nicht, **wie** es dies schaffen soll. Trotzdem werden vorhandene Schnittstellen und Beziehungen zu anderen Systemen dargestellt. Ein White-Box Use Case zeigt dementsprechend eher, wie eine gewollte Funktionalität eines Systems mit verschiedenen Klassen, Schnittstellen und anderen Komponenten am besten dargestellt wird.

Der Prototyp dieser Arbeit besitzt drei funktionale Anforderungen an das System. Als erste Anforderung existiert eine Menü Funktion. Dargestellt wird dies in Abbildung 9.

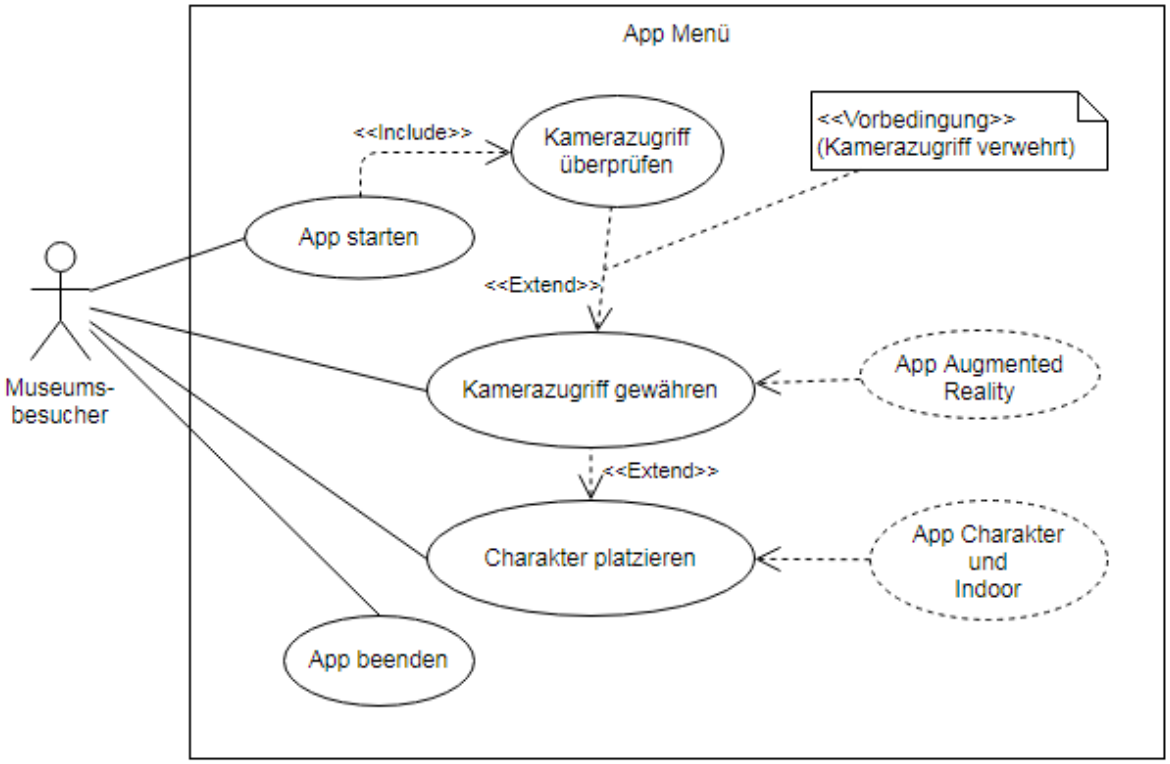


Abbildung 9: Use Case Diagramm für das Menü der App

Diese Abbildung zeigt ein typisches Use Case Diagramm. Das System bildet dabei das Rechteck mit all seinen enthaltenen Elementen. In diesem Fall wird das System mit „App Menü“ betitelt. Links neben dem System befindet sich der Akteur. Dieser interagiert und greift auf das System zu. Verwendet wird dieses System von einem Museumsbesucher. Dieser hat Zugriff auf verschiedene Anwendungsfälle, also Use Cases. Im Fall der Abbildung sind es vier. Er kann die App starten, aber sie auch wieder beenden. Zusätzlich kann er den Kamerazugriff gewähren, sollte die Kameranutzung nicht schon erlaubt sein. Dies wird beim Start der App überprüft. Dies geschieht mit einem „include“ nach dem Use Case „App starten“ zu einem Use Case mit dem Titel „Kamerazugriff überprüfen“. Mit „extend“ wird eine Antwort an den Anwendungsfall „Kamerazugriff gewähren“ gesendet. Jedoch geschieht dies nur unter einer bestimmten Vorbedingung. In dem Fall der Abbildung kann der Nutzer den Anwendungsfall „Kamerazugriff gewähren“ nur verwenden, wenn vorher der Kamerazugriff verwehrt wurde. Ist der Zugriff auf die Kamera erlaubt, dann kann das System nun auf das System „App Augmented Reality“ zugreifen und damit arbeiten. Zusätzlich kann der Museumsbesucher nun einen Charakter platzieren. Sollte er dies wollen, dann wird in ein anderes System, das „App Charakter und Indoor“ System gewechselt.

Der Rahmen bildet in diesem Beispiel die Systemgrenze und Akteure interagieren von außerhalb mit dem System. In diesem Beispiel ist es nur ein Akteur, aber es muss nicht immer nur ein Akteur bleiben. Sollte ein weiterer Anwendungsfall, wie z.B ein RunTime Update für die App hinzugefügt werden, dann könnte mit dem Entwickler ein weiterer Akteur ganz einfach in das System hinzugefügt werden.

Die anderen beiden Systeme sind in den folgenden beiden Abbildung 10 und 11 dargestellt.

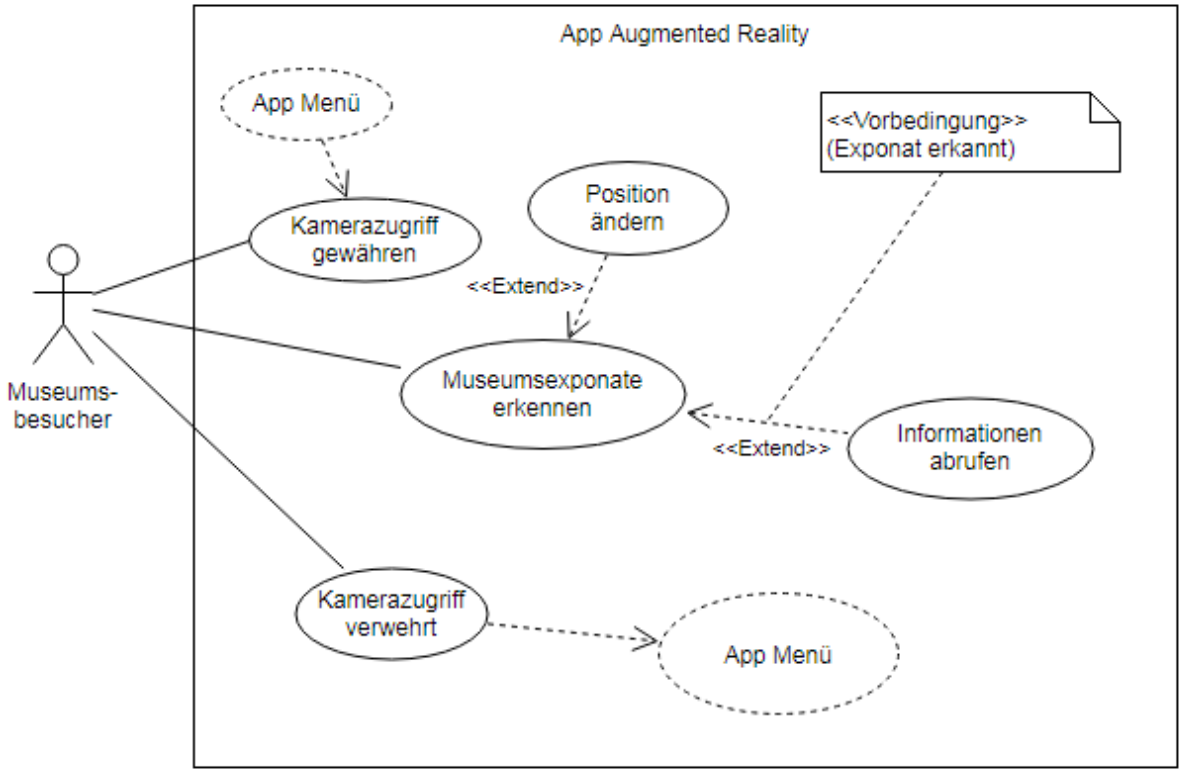


Abbildung 10: Use Case Diagramm für die Augmented-Reality-Komponente der App

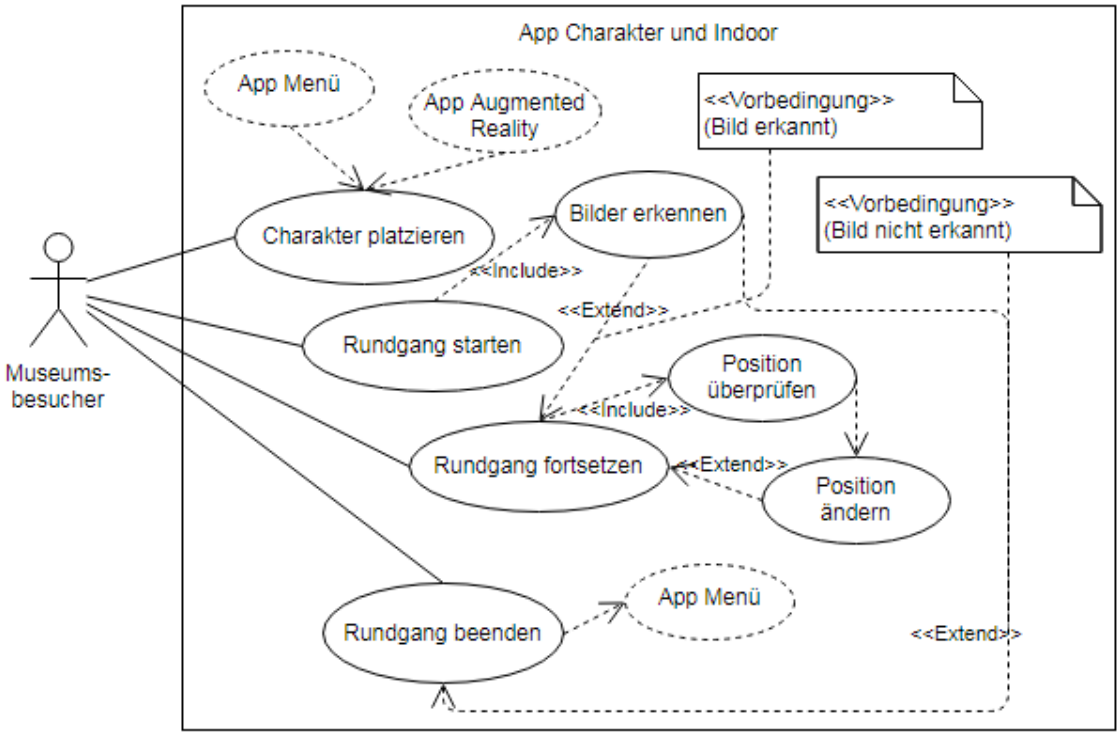


Abbildung 11: Use Case Diagramm für die Charakter- und Indoor-Navigations-Komponente der App

Alle drei Systeme interagieren miteinander. Dabei bildet „App Menü“ das Hauptsystem und „App Augmented Reality“ und „App Charakter und Indoor“ die Untersysteme, in die der Nutzer durch Use Cases aus dem Hauptsystem zugreifen kann. Sobald der Kamerazugriff gewährt wurde, aktiviert sich das System „App Augmented Reality“. Dieses kann Museumsexponate erkennen, wenn diese sich im Sichtfeld der Kamera befinden. Das geschieht durch die Positionsänderung des mobilen Endgeräts. Unter der Vorbedingung, dass ein Exponat erkannt wurde, lassen sich zusätzliche Informationen abrufen. Sollte im Laufe der Nutzung der App der Kamerazugriff verwehrt werden, dann wird dieses System geschlossen und der Nutzer wird in das Ausgangssystem „App Menü“ zurückgebracht. Als zweites Untersystem lässt sich „App Charakter und Indoor“ durch den Anwendungsfall „Charakter platzieren“ erreichen. Dieses System benötigt zur vollständigen Nutzung beide anderen Systeme. Dabei gilt „App Menü“ als Startpunkt, um in das Untersystem zu gelangen. Mit Hilfe von „App Augmented Reality“ interagiert das „App Charakter und Indoor“ System mit der Umgebung. Zusätzlich dazu verwendet der Rundgang die Positionen der einzelnen Bilder bzw. Exponate. Diese müssen vor dem Starten des Rundgangs erkannt werden. Deswegen werden zu Beginn des Rundgangs die Liste der Bilder überprüft, ob alle Bilder erkannt worden. Ist diese Vorbedingung erfüllt, kann der Museumsrundgang erfolgreich fortgesetzt werden. Sollten die Bilder nicht erkannt werden, dann muss der Rundgang beendet werden und der Museumsbesucher gelangt automatisch in das Hauptsystem. Wurden die Bilder erfolgreich erkannt, dann werden die Positionen dieser gespeichert und mit dem Charakter auf seinem Weg zwischen den verschiedenen Exponaten verglichen und ausgetauscht.

Für eine vollständige und erfolgreiche Nutzung der Endsoftware sind alle drei Systeme nötig. Die App kann nicht durch das Hauptsystem „App Menü“ allein gesteuert werden. Die Interaktionen und Verschachtelungen zwischen den drei Systemen ermöglichen den reibungslosen Ablauf der Software.

3.1.3 Nicht-funktionale Anforderungen

Im Gegensatz zu den funktionalen Anforderungen, die betrachten, was das System zu leisten hat, betrachten die nicht-funktionalen Anforderungen das System unter dem Gesichtspunkt, wie es diese Leistungen erbringen soll.

Dargestellt sind mögliche Funktionalitäten unter der Norm ISO 9126. [vgl. NormIso01] Dort sind die Oberbegriffe Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Wartbarkeit und Übertragbarkeit als Vergleichspunkte genannt.

Die **Funktionalität** handelt von der Richtigkeit der Software bzw. der Systeme. Betrachtet wird zudem die Konformität mit verschiedenen Normen, die eingehalten werden müssen. Ein weiterer Punkt ist die Interoperabilität, also der reibungslose Ablauf innerhalb mehrerer Systeme. Zusätzlich werden hier Sicherheiten definiert und eingehalten. [vgl. NormIso01]

Der Unterpunkt **Zuverlässigkeit** betrachtet Funktionalitäten des Systems, wie die Fehlertoleranz, die Robustheit bei Störungen oder die Wiederherstellbarkeit bei Ausfällen. [vgl. NormIso01]

Bei der **Benutzbarkeit** wird das System unter dem Gesichtspunkt von projektunabhängigen Personen betrachtet, also Personen, die nicht an der Entwicklung der Software beteiligt waren. Hier zählen die Verständlichkeit, die Bedienbarkeit und die Erlernbarkeit der Software. Dazu spielt die Attraktivität der App eine wichtige Rolle. Ist die Software übersichtlich und ansprechend gehalten, dann wird sie vielleicht schon automatisch von den Museumsbesuchern genutzt. [vgl. NormIso01]

Der Ressourcenverbrauch ist ein wichtiger Gesichtspunkt unter dem Unterpunkt **Effizienz**. Damit die Software auf vielen Endgeräten laufen kann, müssen die Anforderungen an die Hardware soweit minimiert werden, wie es möglich ist. [vgl. NormIso01]

Die Software muss mit ihrem Code eine gewisse Stabilität liefern. Der Code muss demnach ordentlich geschrieben sein, damit im Nachhinein noch Änderungen erfolgen können. Die Software soll später ebenso analysierbar wie zu Beginn der Erstellung sein. So ist sie auch später immer wieder auf Fehler prüfbar. All diese Punkte finden sich unter dem Begriff der **Wartbarkeit** wieder. [vgl. NormIso01]

Als letzter Punkt wird die **Übertragbarkeit** betrachtet. Hier wird getestet, wie erfolgreich sich die App installieren lässt. Oder ob bei der Installation irgendwelche Fehler auftreten bzw. auftreten können. Ist die Software im Nachhinein noch änderbar? Können neue Module auch später noch hinzugefügt werden? Solche und weitere Fragen werden auch unter dem Gesichtspunkt der Übertragbarkeit erörtert. [vgl. NormIso01]

Im Vergleich zu den funktionalen Anforderungen sind die nicht-funktionalen Anforderungen sehr verallgemeinert gehalten. Trotzdem sind sie nicht unwichtiger. Die funktionalen Anforderungen beziehen sich mehr auf die inneren Systeme und ihre Funktionalität zueinander. Also das, was die Software leisten soll. Trotz der allgemeinen Aspekte haben die nicht-funktionalen Anforderungen einen großen Einfluss auf die fertige Software. Denn hier wird nicht nur darauf geachtet, ob bestimmte Normen eingehalten werden, sondern unter diesen Aspekten wird die Software mit der Hardware in bestmöglichster Form kompatibel gemacht.

3.1.4 Festlegung von Hard- und Software

Vor der Erstellung von Software müssen verschiedene Möglichkeiten betrachtet und ausgewertet werden. Dies ist zum Teil innerhalb dieser Arbeit schon geschehen. Im Unterkapitel 2.1.3 (Seite 9) ist mit der Abbildung 3 die unterschiedliche Nutzung von verschiedenen mobilen Betriebssystemen gezeigt worden. Da diese Software auf möglichst vielen Endgeräten laufen soll und vor allem auf den eigenen Geräten der Museumsbesucher, muss die breite Masse angesprochen werden. Deshalb wird die App für Android entwickelt. Eine weitere Veröffentlichung auf iOS Geräten fällt vorerst weg. Durch die Eingrenzung des Betriebssystems der Endhardware kann nun eine Entscheidung für andere Komponenten getroffen werden. Für Augmented Reality kann demnach ARCore verwendet werden. ARCore wird von Google eigens für Android entwickelt und bietet sich deswegen hervorragend an. Für die Implementierung von Indoor Navigation wird die integrierte Bilderkennung von ARCore und die Speicherung der Positionen im Raum herangezogen. Das Betriebssystem der Endhardware muss demnach eine Versionsnummer haben, die ARCore unterstützt, zusätzlich müssen Hardwarekomponente vorhanden sein, die

von ARCore für die räumliche Orientierung genutzt werden. Um alle verschiedenen Komponenten in einer App zu vereinen, wird eine Software namens Unity verwendet. Unity bietet die Möglichkeiten und vor allem die Schnittstellen, um eine App mit Augmented Reality und Indoor Navigation zu erstellen. Im folgenden Unterkapitel 3.2 wird Unity als Software Ansatz noch einmal genauer betrachtet.

3.2 Unity als Software Ansatz

Unity ist eine 3D-Entwicklungsumgebung mit einer eigenen Engine. Damit ermöglicht diese Software die Erstellung von einfachen bis komplexen Programmen. Die erstellten Anwendungen können sowohl im 2D-Design als auch im 3D-Design konstruiert werden. Die interaktive Benutzeroberfläche von Unity bietet viele verschiedene Optionen für die Auswahl von Objekten und ermöglicht eine einfache Verwendung dieser. Mit Hilfe von C# können die verschiedenen Elemente mit Skripts verbunden werden.

Die Benutzeroberfläche ist in Abbildung 11 zur besseren Veranschaulichung mit einer Version des Prototypens dargestellt.

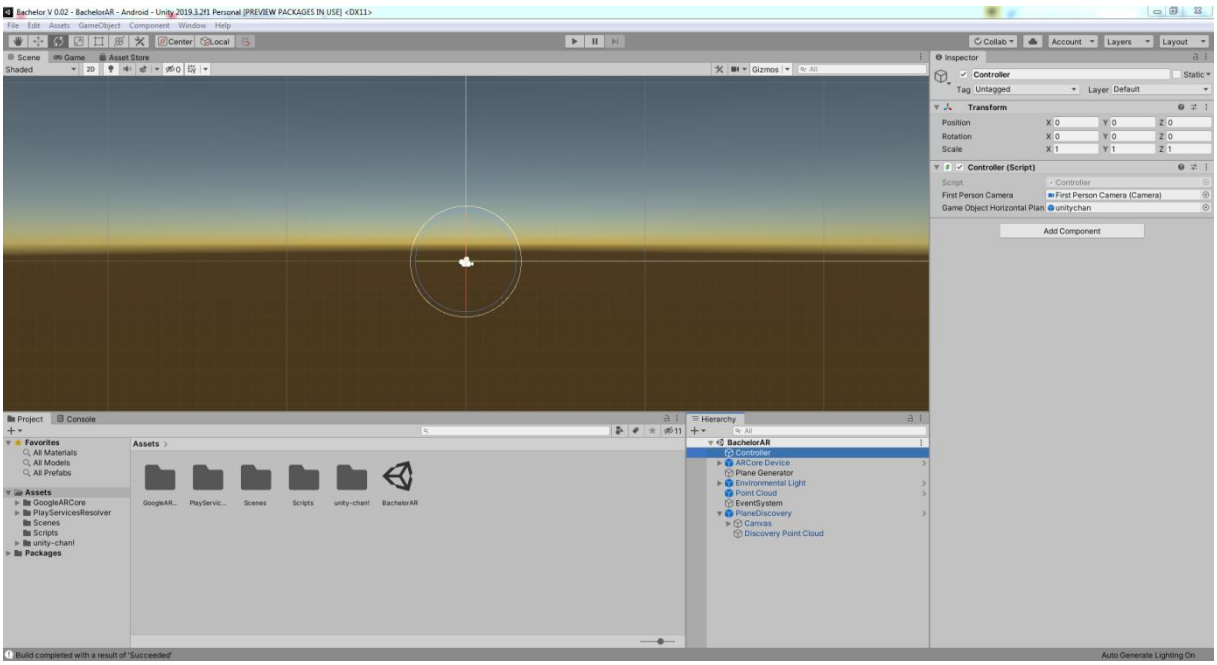


Abbildung 12: Grafische Benutzeroberfläche in Unity

Für eine bessere Übersicht soll die Benutzeroberfläche mit einem schematischen Bild erklärt werden. Die folgende Abbildung 12 zeigt den schematischen Aufbau der Benutzeroberfläche.

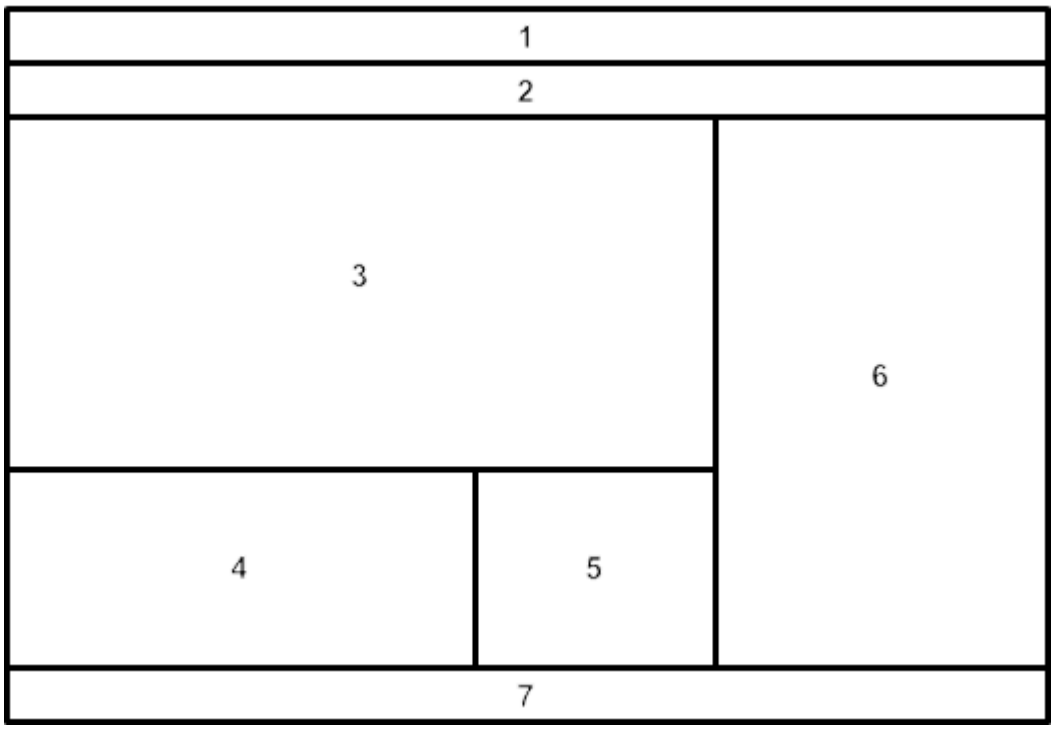


Abbildung 13: Schematischer Aufbau der Benutzeroberfläche

Die Benutzeroberfläche wurde in sieben Abschnitte gegliedert: Menüleiste, Bedienleiste, Szenenoberfläche mit Asset Store, Projektordner mit Meldungsanzeige, Hierarchie, Inspektor und Statusanzeige.

Die Menüleiste (1) besteht aus den Buttons „File“, „Edit“, „Asset“, „GameObject“, „Component“, „Window“, „Help“ und den Manipulatoren (Minimieren, Vollbild, Schließen). Zusätzlich gibt die Statusleiste das derzeitige Projekt, die Szene, die Entwicklungsplattform und die verwendete Version an. Unter jedem Button verstecken sich viele weitere Befehle. Mit dem Button „File“ können neue Szenen erstellt und gespeichert oder vorhandene geöffnet werden. Hier ist es ebenso möglich ein neues Projekt zu erstellen oder ein vorhandenes zu öffnen. Weiterhin kann hier das Programm „gebaut“ werden. Mit Hilfe der „Built Eigenschaften“ können die Endszenen der App, sowie die gewünschte Zielplattform gewählt werden. Hinzu kommen viele weitere Einstellungen, die die Zielplattform näher betrachten. Dies wird in Abbildung 13 dargestellt.

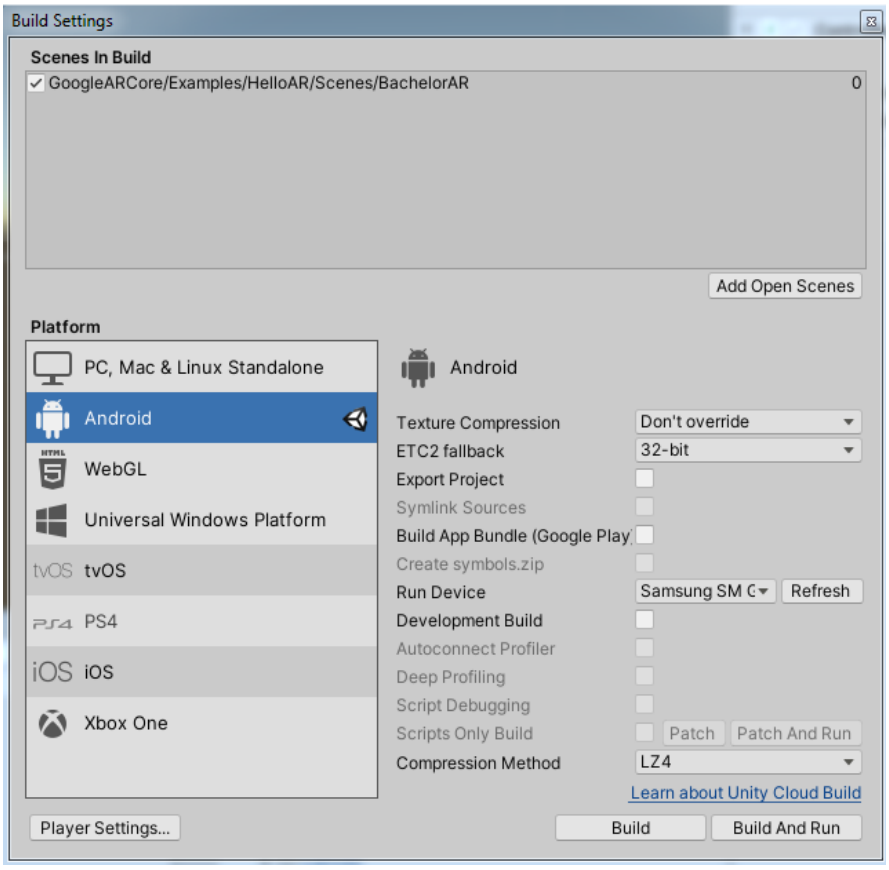


Abbildung 14: Built Eigenschaften in Unity

Der Button „Edit“ ermöglicht mit vielen verschiedenen Optionen, die in der Szene benutzten Elemente zu bearbeiten. Dort finden sich vor allem Standardbefehle, wie Auswählen, Ausschneiden, Kopieren oder Einfügen wieder. Mit „Assets“ lassen sich bestimmte vorhandene Pakete importieren und exportieren. Assets sind zusätzliche Schnittstellenpakete, die in das Projekt integriert werden können. Durch diese können neue Elemente und Objekte, die standardmäßig nicht in dem Projekt sind, verwendet werden. Diese werden über verschiedene Webseiten heruntergeladen und dann hinzugefügt. Dies kann auch über den Asset Store, der von Unity angeboten wird geschehen. Über „GameObject“ können in die Szene neue Objekte, wie z.B. ein Quader, Zylinder sowie Effekte, wie Licht eingefügt werden. Neue Eigenschaften erhält das gerade erstellte Element über den Button „Component“. Unter „Window“ gibt es weitere Fenster zum Feintuning der Objekte. Dazu zählt unter anderem der „Animator“ zur Entwicklung von Animation bei beweglichen Körpern. Zuletzt gibt es unter „Help“ eine Anleitung zur Nutzung von verschiedenen Befehlen, um die Eigenschaften und die Komponenten im Skript zu nutzen.

In der Bedienleiste (2) stehen dem Benutzer verschiedene Steuerelemente zur Verfügung. Hiermit kann jedes Element per Hand bewegt, rotiert oder skaliert werden. Des Weiteren gibt es einen Start-, Pause- und Stoppknopf zum Testen der entwickelten Anwendung und zusätzlich kann auf den eigenen Account, auf verschiedene Layouts der Benutzeroberfläche oder Kooperationsmöglichkeiten mit anderen Unity-Nutzern zugegriffen werden.

Die Szenenoberfläche mit Asset Store (3) unterteilt sich in mehrere Reiter. Dazu zählen die Szene und der Asset Store. In der Szene werden benutzte Elemente der jeweiligen ausgewählten Szene angezeigt und das in entweder 2D oder 3D, je nachdem wie der Benutzer es in dem Moment benötigt. Über den Reiter Asset Store[3] können zusätzliche Assets heruntergeladen und hinzugefügt werden.

Der Projektordner mit Meldungsanzeige (4) ist ebenso in verschiedene Reiter aufgeteilt. Dazu zählen der Projektreiter und der Konsolenreiter, der auch als Meldungsanzeige bezeichnet wird. In dem Projektordner werden alle verschiedenen Assets, die verwendeten Szenen und Skripts, sowie weitere Zusatzpakete geordnet und angezeigt. In der Konsole werden nach dem kompilieren des Programmes mögliche Fehler oder Probleme aufgezeigt.

Mit der Hierarchie (5) wird eine genauere Szenenbetrachtung ermöglicht. Hier wird jede verwendete Szene, in der vom Benutzer festgelegten Reihenfolge, angegeben. Zusätzlich ordnen sich alle verwendeten Objekte im Top-Down-Verfahren unter. Dabei haben höher gestellte Elemente eine höhere Priorität und Wichtigkeit als Elemente, die sich weiter unten in der Kette befinden.

Mit Hilfe von dem Inspektor (6) können Bausteine aus der Hierarchie ausgewählt und näher betrachtet werden. Hier ist jede Eigenschaft des Elements aufgelistet und kann statisch bearbeitet werden. Zusätzlich werden hier Skripts zu einzelnen Objekten verlinkt, in dem diese einfach bei einem ausgewählten Element in den Inspektor gezogen werden. Jedes Skript kann danach noch ein- oder ausgeschaltet werden.

---

[3]siehe Anhang 1.

Die Statusanzeige (7) gibt nach dem Kompilieren an, ob das Programm erfolgreich ausgeführt werden kann oder nicht.

4 Implementierung und Test

4.1 Die Augmented – Reality – Komponente

Bevor Augmented Reality in Unity erfolgreich genutzt werden kann, müssen ein paar Bedingungen erfüllt sein. Die Software Unity muss mindestens die Version **2017.3.34f1** sein oder spätere. Für den Prototyp wird Version **2019.3.2f1** verwendet, womit keine Probleme entstehen sollten. Durch die neuere Version werden dafür in Unity für das Projekt zwei weitere Pakete benötigt. Zum Einen **Multiplayer HLAPI** und zum Anderen **XR Legacy Input Helper**. Beide können über den Package Manager von „Window“ aus der Menüleiste in das Projekt importiert werden. Als Nächstes wird mit der **ARCore SDK** von mindestens Version 1.15.0 ein weiteres Asset importiert. Eine besondere JDK wird nicht benötigt, diese kann gleich beim Installieren der aktuellen Unity Version mit auf den Computer gespielt werden. Für die Android ARCore Entwicklung fehlt nun nur noch eine **Android SDK** von mindestens Version **7.0**. Diese installiert man mit dem Programm Android Studio und dem dazugehörigen SDK Manager. [vgl. Arc02]

Nachdem alle Anforderungen erfüllt sind, kann mit der Implementierung gestartet werden. Um ARCore auf dem Android Smartphone nutzen zu können, muss in Unity erst einmal ein Grundgerüst geschaffen werden. Die folgende Abbildung 14 verdeutlicht die Grundbausteine für eine Augmented Reality fähige App.

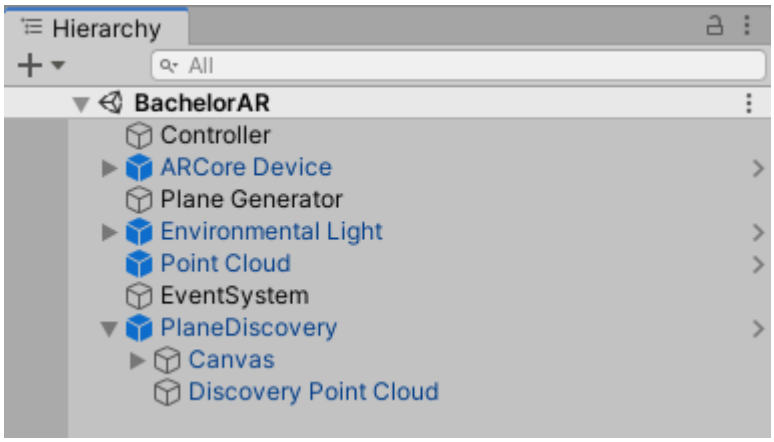


Abbildung 15: Grundbausteine zur Nutzung von Augmented Reality

In der Abbildung 14 ist die Hierarchie der „BachelorAR“ Szene zu sehen. Dieser Szene untergeordnet sind verschiedene GameObjects. Alle dort gezeigten Objekte sind nötig, um Augmented Reality auf dem Android Smartphone zu ermöglichen. Wenn die Abbildung genauer betrachtet wird, dann werden zwei unterschiedliche Arten von Elementen erkannt. Hier als weiß und blau dargestellt. Weiß bilden dabei neu erstellte GameObjects und blaue sind Prefabs, sogenannte „fertige Bausteine“. Diese besitzen schon bestimmte Eigenschaften oder zugewiesene Skripte. Solche Prefabs können aus dem vorher hineingezogenen ARCore Asset verwendet werden.

Das Controller GameObject hält die Szene zusammen und verknüpft alle anderen Objekte miteinander. Deswegen werden zuerst die anderen Elemente betrachtet und als letztes der Controller, der alles zusammen fügt.

Das erste Prefab der Liste bildet das Element **ARCore Device**. Dieses hat ein untergeordnetes GameObject mit dem Titel „First Person Camera“. Diese hat mehrere Eigenschaften, um aktiv die Kamera des Handys verwenden zu können. Dafür hat es einmal die Kamera Eigenschaft bekommen, so kann es eine Kamera nutzen und dort können verschiedenen Optionen angepasst werden. Hinzu kommt eine von ARCore gestellte Eigenschaft mit dem Titel „Tracked Pose Driver“. Diese gibt an, **womit** getrackt werden soll, also das Device, auf dem die App laufen soll, **was** für eine Kameraeinstellung verwendet wird, also eine farbenunterstützte Kamera und **wann** getrackt werden soll, sprich ob das Handy bewegt oder gedreht wird. Sowie welcher Update Typ verfolgt werden soll. In dem Prototyp soll immer aktualisiert werden, solange die Anwendung läuft. Abbildung 15 zeigt, wie das Ganze im Inspektor in Unity aussieht.

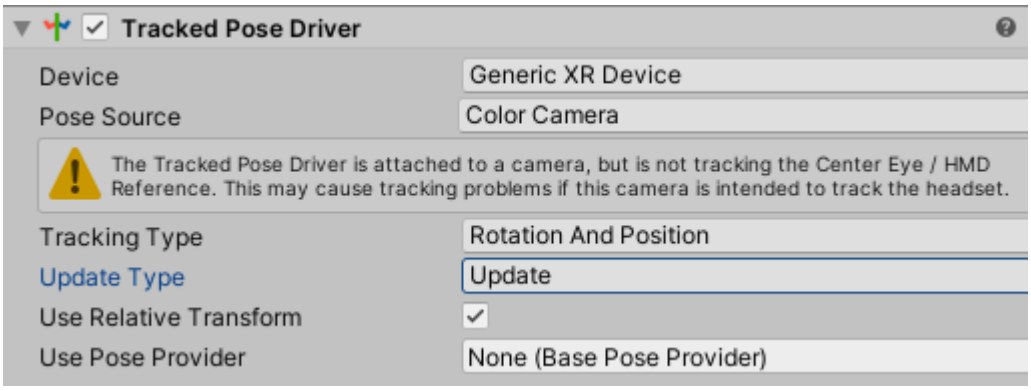


Abbildung 16: First Person Camera Eigenschaft „Tracked Pose Driver“

Dem ARCore Device ist zusätzlich ein Skript zugewiesen, um verschiedene Fehlermeldungen ausgeben zu können, sollten bestimmte Probleme mit der Kamera existieren. So können z.B. keine Ebenen erkannt werden, wenn keine Kamera auf der Rückseite des Smartphones verfügbar ist und es kann keine Gesichtserkennung erfolgen, wenn keine Kamera oberhalb des Displays existiert. In diesem Prototyp spielt die Gesichtserkennung aber keine Rolle.

Als Nächstes folgt das „**Plane Generator**“ GameObject. Dieses besitzt nur ein einfaches Skript, welches im folgenden Listing 1 betrachtet wird.

```
1  public class DetectedPlaneGenerator : MonoBehaviour
2      {
3
4          public GameObject DetectedPlanePrefab;
5
6          private List<DetectedPlane> m_NewPlanes = new List<DetectedPlane>();
7
8          public void Update()
9          {
10             // Check that motion tracking is tracking.
11             if (Session.Status != SessionStatus.Tracking)
12             {
13                 return;
14             }
15
16             Session.GetTrackables<DetectedPlane>(m_NewPlanes, TrackableQueryFilter.New);
17             for (int i = 0; i < m_NewPlanes.Count; i++)
18             {
19                 GameObject planeObject =
20                     Instantiate(DetectedPlanePrefab, Vector3.zero, Quaternion.identity, transform);
21                 planeObject.GetComponent<DetectedPlaneVisualizer>().Initialize(m_NewPlanes[i]);
22             }
23         }
24     }
```

Listing 1: Auszug aus der „DetectedPlaneGenerator“ Klasse

In diesem Codebeispiel wird die „DetectedPlaneGenerator“ Klasse betrachtet. Für diese existiert ein öffentliches GameObject, welches über den Inspektor zugewiesen wird und eine private Liste an m_NewPlanes, also eine Liste aus allen gefundenen Ebenen. Dem öffentlichen GameObject wird ein Prefab hinzugefügt, aus dem sich später die angezeigte Ebene zusammensetzt. In der Update Funktion wird zu aller erst geprüft, ob überhaupt getrackt wird. Danach wird geschaut, ob die Ebene schon fokussiert wurde und dann wird für alle neuen Ebenen ein neues GameObject angelegt, mit den jeweiligen Größen der getrackten Ebene. Dazu zählen die Ausbreitungsrichtungen in X und Y und auf welcher Koordinate Z sich die neue Ebene befindet. Diese Funktion wird im Laufe der Anwendung immer und immer wieder aufgerufen, sollte eine neue Ebene gefunden werden.

Als weiteres GameObject folgt das „**Environmental Light**“ GameObject Prefab. Dieses reagiert auf die verschiedenen Lichtverhältnisse, welche durch das Tageslicht entstehen können. So können Ebenen besser, oder schlechter, oder gar nicht erkannt werden. Zusätzlich können den projizierten Objekten Schatten gegeben werden, womit diese noch realer wirken.

Das nächste Prefab „**Point Cloud**“ bildet die Ausgangsbasis für die Erstellung, sowie für die visuelle Darstellung der zukünftigen Ebenen, die mit Hilfe des Plane Generator’s in einer Liste gespeichert werden. Diesem Element sind neben einem Skript zwei weitere Eigenschaften zugewiesen. Zum Einen ist da der „Mesh Filter“, der die Darstellungsart des Objekts auf dem Display angibt. In diesem Fall entspricht es dem Körper eines Würfels. Zum Anderen gibt es den „Mesh Renderer“, der die Beschaffenheit des angezeigten Objekts näher beschreibt. Dazu zählt die Größe, mit der diese Punkte auf dem Screen dargestellt werden sollen, aber auch die Interaktionen zu anderen Umgebungsvariablen, wie zudem Licht. So sollen die erstellten Punkte Schatten werfen können. Zum besseren Verständnis werden zwei verschiedene Prozeduren aus dem Skript der „PointCloud Visualizer“ Klasse betrachtet.

```
1   public void Start()
2       {
3           m_MeshRenderer = GetComponent<MeshRenderer>();
4           m_Mesh = GetComponent<MeshFilter>().mesh;
5           if (m_Mesh == null)
6           {
7               m_Mesh = new Mesh();
8           }
9           m_Mesh.Clear();
10
11          m_CachedColor = PointColor;
12
13          m_ScreenWidthId = Shader.PropertyToID("_ScreenWidth");
14          m_ScreenHeightId = Shader.PropertyToID("_ScreenHeight");
15          m_ColorId = Shader.PropertyToID("_Color");
16
17          m_PropertyBlock = new MaterialPropertyBlock();
18          m_MeshRenderer.GetPropertyBlock(m_PropertyBlock);
19          m_PropertyBlock.SetColor(m_ColorId, m_CachedColor);
20          m_MeshRenderer.SetPropertyBlock(m_PropertyBlock);
21
22          m_CachedPoints = new LinkedList<PointInfo>();
23      }
```

Listing 2: Start Funktion als Auszug aus der „PointCloud Visualizer“ Klasse

Listing 2 zeigt die „Start“ Funktion der Klasse. Zu aller erst werden mit dem Befehl „GetComponent“ die beiden zuvor beschriebenen Eigenschaften in das Skript aufgenommen und für Berechnungen zugänglich gemacht. Danach wird mit der if-Schleife geprüft, ob die gerade erstellte Variable „m_Mesh“ vom Typ Mesh leer ist. Sollte sie leer sein, dann wird sie als neue Mesh Prozedur behandelt. Folglich wird sie als Nächstes geleert und für die zukünftige Nutzung verfügbar gemacht. Die folgende Variable „m_CachedColor“ beschreibt die Farbe der auf dem Display dargestellten Punkte und wird im Inspektor ausgewählt. „m_ScreenWidthId“, „m_ScreenHeightId“ und „m_ColorId“ sind die genauen Daten für die Erstellung eines Schattens zu den einzelnen Punkten der „Point Cloud“ in Abhängigkeit von der Displaygröße und der zuvor eingestellten Farbe. Der nächste Block an Codezeilen von Zeile 17 bis Zeile 20 beschreibt bestimmte Einstellungen, wie z.B. das Material und die Farbe für den dargestellten Würfel. Mit dem Befehl „get“ werden die gewünschten Eigenschaften geladen und mit „set“ auf das Objekt angewendet. Die letzte Variable „m_CachedPoints“ dieser Prozedur wird als eine verbundene Liste von vielen Punkten erstellt. Dabei sind Punkte miteinander verknüpft, die ein und dieselbe Ebene repräsentieren. Dies geschieht durch verschiedene Parameter, wie z.B. die Y-Koordinate für horizontale oder die X-Koordinate für vertikale Ebenen.

Im folgenden Listing 3 wird die „Update“ Prozedur gezeigt.

```
1   public void Update()
2       {
3           if (Session.Status != SessionStatus.Tracking)
4           {
5               _ClearCachedPoints();
6               return;
7           }
8           if (Screen.currentResolution.height != m_CachedResolution.height
9               || Screen.currentResolution.width != m_CachedResolution.width)
10          {
11              _UpdateResolution();
12          }
13          if (m_CachedColor != PointColor)
14          {
15              _UpdateColor();
16          }
17          if (EnablePopAnimation)
18          {
19              _AddPointsIncrementallyToCache();
20              _UpdatePointSize();
21          }
22          else
23          {
24              _AddAllPointsToCache();
25          }
26
27          _UpdateMesh();
28      }
```

Listing 3: Update Funktion als Auszug aus der „PointCloud Visualizer“ Klasse

Die „Update“ Funktion ist eine im Hintergrund ablaufende Prozedur, die die verschiedenen if-Schleifen immer und immer wieder durchläuft und davon ausgehend das Gitternetzt[4] der PointCloud überarbeitet. Die erste if-Bedingung prüft, ob ARCore derzeit am tracken ist. Sollte nicht getrackt werden, dann werden alle gespeicherten Punkte gelöscht und die „Update“ Funktion wird nicht mehr ausgeführt. Als Nächstes werden die derzeitigen Displaygrößen, also die Höhe und Breite, mit den gespeicherten Displaywerten verglichen. Sollten diese unterschiedlich zueinander sein, dann werden die gespeicherten Werte mit den neuen überschrieben. Hierbei zählen die im Display angezeigten Objekte in Abhängigkeit zur realen Umgebung und nicht die Festwerte des Bildschirms des Handys. Des Weiteren wird die derzeitige Farbe der neu erstellten Punkte mit der im Inspektor gewählten Farbe verglichen und notfalls geändert, falls diese nicht übereinstimmen sollten. Die letzte if-Schleife prüft, ob beim Auftauchen neuer Punkte eine Animation gespielt werden muss. Das trifft auf den Prototypen nicht zu, deshalb wird der else-Zweig genommen. In diesem werden alle derzeitigen auf dem Display angezeigten Punkte mit all ihren Daten in einem Zwischenspeicher gelagert.

Das folgende „**EventSystem**“ Objekt wird von Unity automatisch erstellt sobald in die verwendete Szene ein UI-GameObject integriert wird. Zu UI-Elementen zählen unter anderem Text- und Bildelemente oder auch Buttons. Das System regelt verschiedene Interaktionen des Nutzers mit solchen Objekten. Vor allem, wenn dieser auf dem Display eines dieser Komponenten berührt. Dann reagiert die Anwendung darauf, dass dieses bestimmte Element ausgewählt wird. Jedoch muss der Programmierer selbst festlegen, was danach passieren soll. Als Beispiel wird ein Button betrachtet. Dieser Button wird von einem User betätigt und das Programm erkennt automatisch, dass dieser eine Knopf gedrückt wurde. Nun passiert aber noch nichts, denn der Entwickler muss in seinem eigenen Code erst festlegen, was passieren soll, sobald der Button gedrückt wurde. Z.B könnte mit Hilfe dieses Buttons Szene 1 zu Szene 2 wechseln.

---

[4]Der deutsche Begriff für das englische Wort mesh.

Als letztes Prefab ist „Plane Discovery“ dafür zuständig, dass der Nutzer weiß, dass nach Ebenen gesucht wird. Dort kommt mit Hilfe der Canvas[5] Textelemente und Buttons zum Einsatz, falls nach einer bestimmten Zeit immer noch keine geeignete Ebene gefunden wurde. Dadurch bekommt der User eine Hilfestellung, falls er nicht weiß, wie er Augmented Reality verwenden soll.

Da nun alle anderen Elemente betrachtet wurden, erfolgt als letztes das **Controller** GameObject. Diesem wird das eigens entwickelte Skript zugeordnet. Dieser Code bildet das Grundgerüst des Programms und sorgt dafür, dass es ordnungsgemäß gestartet und ausgeführt wird. Dazu werden mit Listing 4, Listing 5 und Listing 6 Auszüge aus der „Controller“ Klasse gezeigt.

```
1   public class Controller : MonoBehaviour
2       {
3           public Camera FirstPersonCamera;
4
5           private bool m_IsQuitting = false;
6
7           public void Update()
8           {
9               _UpdateApplicationLifecycle();
10
11              Touch touch;
12              if (Input.touchCount < 1 || (touch = Input.GetTouch(0)).phase != TouchPhase.Began)
13              {
14                  return;
15              }
16              if (EventSystem.current.IsPointerOverGameObject(touch.fingerId))
17              {
18                  return;
19              }
20
21              TrackableHit hit;
22              TrackableHitFlags raycastFilter = TrackableHitFlags.PlaneWithinPolygon |
23                  TrackableHitFlags.FeaturePointWithSurfaceNormal;
24
```

Listing 4: Update Funktion als Auszug aus der „Controller“ Klasse

In Listing 4 ist ein Teil der Variablen der Controller Klasse und ein Auszug aus der Update Funktion gezeigt. Vor einigen Seiten wurde das Prefab „ARCore Device“ und das untergeordnete GameObject „First Person Camera“ angesprochen. Dieses Objekt wird in Zeile 3 als Kamera der Anwendung festgelegt. Dazu wird das Element aus der Hierarchie in das Feld im Inspektor gezogen. Somit wird eine Verknüpfung geschaffen. Die private boolsche Variable wird erst in Listing 5 benötigt und erläutert. Für die „Controller“ Klasse gibt es keine Start Prozedur.

---

[5] Die Canvas ist eine Fläche, die sich an die Screengröße anpasst und auf die alle UI-Elemente kommen.

Die Augmented Reality Umgebung soll, solange die Anwendung geöffnet ist, die ganze Zeit laufen. Daraus folgt, dass die Hauptfunktion der Klasse eine „Update“ Funktion ist. Diese läuft im Hintergrund immer und immer wieder, so lange der Fokus des Smartphones auf der App liegt. Der erste Befehl in Zeile 9 ist eine Verknüpfung zu einer weiteren Prozedur. Diese wird ebenfalls in Listing 5 näher ausgeführt. Danach werden ab Zeile 11 verschiedene Abfragen behandelt, wenn der Nutzer sein Display berührt. Es wird eine neue lokale Variable „touch“ vom Typ Touch erstellt. Mit dieser wird in den nächsten beiden if-Schleifen gearbeitet. Hat der User noch nicht sein Display berührt, dann wird die erste Schleife durchlaufen und die „Update“ Funktion für diesen Durchlauf beendet. Die Bedingung dafür lautet, dass die Anzahl der Berührungen kleiner als 1 ist und dass die Touchphase[6] der touch Variablen an der nullten Stelle ungleich der beginnenden Touchphase ist. Eine Touchphase hat begonnen, sobald eine Eingabe in Form einer Berührung stattgefunden hat. Die zweite Schleife handelt davon, dass der Nutzer schon auf das Display getippt hat, aber er über einem UI-Element war, wodurch die „Update“ Funktion für diesen Durchlauf ebenso beendet wird. Denn UI-Objekte werden über das Eventsystem geregelt.

Mit Zeile 21 wird eine weitere lokale Variable erstellt. Diese mit dem Namen „hit“ ist vom Typ TrackableHit. Hier wird ein imaginärer Strahl, ausgehend vom User touch auf dem Display, in die Augmented Reality Umgebung projiziert, um mit irgendeinem Element auf dem Weg reagieren zu können. Diese Elemente können Ebenen sein, spezielle Punkte der Ebene oder Objekte, die in der Augmented Reality Umgebung vom Nutzer platziert wurden.

---

[6]Die Touchphase bezeichnet eine längere Berührung des Displays.

```
private void _UpdateApplicationLifecycle()
    {
        if (Input.GetKey(KeyCode.Escape))
        {
            Application.Quit();
        }
        if (Session.Status != SessionStatus.Tracking)
        {
            Screen.sleepTimeout = SleepTimeout.SystemSetting;
        }
        else
        {
            Screen.sleepTimeout = SleepTimeout.NeverSleep;
        }
        if (m_IsQuitting)
        {
            return;
        }
        if (Session.Status == SessionStatus.ErrorPermissionNotGranted)
        {
            _ShowAndroidToastMessage("Kamerazugriff wird für das Benutzen der App benötigt.");
            m_IsQuitting = true;
            Invoke("_DoQuit", 0.5f);
        }
        else if (Session.Status.IsError())
        {
            _ShowAndroidToastMessage(
                "ARCore encountered a problem connecting.  Please start the app again.");
            m_IsQuitting = true;
            Invoke("_DoQuit", 0.5f);
        }
    }
```

Listing 5: _UpdateApplicationLifecycle Funktion als Auszug aus der „Controller“ Klasse

Die in Listing 4 erwähnte „_UpdateApplicationLifecycle“ Prozedur wird in Listing 5 gezeigt. In dieser Funktion werden alle Möglichkeiten durchgegangen, wie die App geschlossen werden könnte. Mit der ersten if-Schleife wird geprüft, ob der „zurück“ Button des Handys betätigt wurde. Wenn ja, dann wird die Systemfunktion „Application.Quit“ ausgeführt. Diese beendet die Anwendung. Mit dem nächsten if-Block wird getestet, ob gerade getrackt wird. Sollte dies nicht der Fall sein, dann wechselt das Display in den Schlafmodus, immer in Abhängigkeit von der zeitlichen Systemeinstellung. Andernfalls soll das Display nie in den Schlafmodus wechseln. In Listing 4 wurde die Variable „m_IsQuitting“ im Code gezeigt. Hier in Listing 5 dient sie als eine Wahrheitsabfrage. Ist die Variable auf true geändert worden, dann wird die immer und immer wieder ausgeführte „_UpdateApplicationLifecycle“ Funktion beendet, damit so die Anwendung ordnungsgemäß geschlossen werden kann. Als letztes werden noch zwei allgemeine Probleme betrachtet.

Problem Nummer eins:

Beim Starten der App wird der Nutzer gefragt, ob er den Kamerazugriff erlauben möchte. Da der Prototyp eine Augmented Reality Anwendung ist, wird die Kamera essentiell benötigt. Sollte jedoch die Kamera nicht zur Verfügung gestellt worden

sein, dann wird eine Toast[7] Nachricht mit der Information, dass die Kamera benötigt wird, gezeigt. Dies geschieht im Falle des Prototypens durch das Aufrufen der Funktion „_ShowAndroidToastMessage“ an die aus dieser Prozedur eine Nachricht vom Typ „string“ übergeben wird. Zusätzlich wird die vorher erwähnte Variable auf true gesetzt. Außerdem wird eine neue Funktion aufgerufen, die das Programm beendet.

Problem Nummer zwei:

Die zweite Variante tritt in Kraft, sobald der Kamerazugriff gewährt wurde und prüft, ob die Kamera oder andere benötigte Teile der Hardware oder die Software ordnungsgemäß laufen. Sollte dabei ein Fehler auftreten, dann wird ebenso die Variable „m_IsQuitting“ auf wahr gesetzt und die Funktion aufgerufen, die das Programm beendet.

In Listing 6 wird gezeigt, wie eine Toast Nachricht mit Hilfe von Unity für das Android Gerät erstellt und angezeigt wird.

```
1   private void _ShowAndroidToastMessage(string message)
2       {
3           AndroidJavaClass unityPlayer = new AndroidJavaClass("com.unity3d.player.UnityPlayer");
4           AndroidJavaObject unityActivity =
5               unityPlayer.GetStatic<AndroidJavaObject>("currentActivity");
6
7           if (unityActivity != null)
8           {
9               AndroidJavaClass toastClass = new AndroidJavaClass("android.widget.Toast");
10              unityActivity.Call("runOnUiThread", new AndroidJavaRunnable(() =>
11              {
12                  AndroidJavaObject toastObject =
13                      toastClass.CallStatic<AndroidJavaObject>(
14                          "makeText", unityActivity, message, 0);
15                  toastObject.Call("show");
16              }));
17          }
18      }
```

Listing 6: _ShowAndroidToastMessage Funktion als Auszug aus der „Controller“ Klasse

Die Funktion wird mit der übergebenen Zeichenkette aufgerufen. Danach wird eine neue AndroidJavaClass für den unityPlayer erstellt. Die derzeitige Szene wird der gerade erstellten Klasse zugewiesen, um Toast-Anzeigen für das Display zu ermöglichen. Ist eine Szene derzeit aktiv, dann wird ein neues Toast-Objekt erstellt und auf Abruf gehalten.

---

[7]Toast ist ein Pop-up-Fenster mit einer Nachricht

Ist ein Text, welcher angezeigt werden soll, verfügbar, dann wird das Toast-Objekt aufgerufen und auf dem Display gezeigt. Sind alle benötigten GameObjects mit ihren jeweiligen Skripts und Einstellungen erfolgreich implementiert wurden, dann sollte die App erfolgreich starten und ausgeführt werden. Die folgende Abbildung 16 zeigt die Reaktion der Anwendung auf die Umgebung.

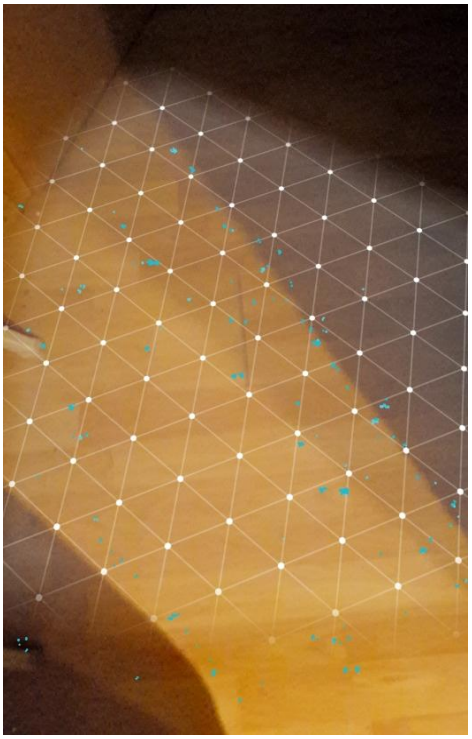


Abbildung 17: Augmented Reality Erkennung auf dem Display des Handys

Sind noch nicht ausreichend Information zusammengekommen, um eine Ebene zu bilden, dann werden in der Anwendung die Punkte schwarz beziehungsweise grau dargestellt. Sobald ausreichend Daten gesammelt sind, wird eine Ebene erzeugt und die Farbe der Punkte ändert sich zu blau, um eine Zugehörigkeit zu verdeutlichen. Die erstellte Fläche wird mit einem Gitternetz aus vielen Dreiecken dargestellt, wie in Abbildung 16 zu erkennen ist.

Nachdem die Augmented Reality Umgebung läuft und erkannt wird, handelt der nächste Schritt von der Bilderdatenbank und der Bilderkennung. Diese sorgt dafür, dass der Charakter weiß, wo er sich hinzubewegen hat. Dafür werden einige Bilder vom Format .png oder .jpeg herausgesucht und in das Unityprojekt integriert. Diese werden danach ausgewählt und mit Hilfe des Google ARCore Package wird eine neue AugmentedImageDatabase erstellt.

So eine Datenbank kann bis zu 1000 verschiedene Bilder aufnehmen. Für weitere Bilder kann jederzeit eine neue erstellt werden, jedoch kann immer nur eine aktiv sein. ARCore unterstützt ebenso das Hinzufügen neuer Bilder in die Datenbank, während die Anwendung zur Erkennung von Bildern läuft. [vgl. Arc03] In diesem Prototyp sind benutzte Bilder schon vor dem kompilieren der App eingefügt wurden.

Das folgende Listing 7 zeigt den erweiterten Code der „Controller“ Klasse.

```
1   public AugmentedImageVisualizer AugmentedImageVisualizerPrefab;
2
3   private Dictionary<int, AugmentedImageVisualizer> m_Visualizers
4       = new Dictionary<int, AugmentedImageVisualizer>();
5
6   private List<AugmentedImage> m_TempAugmentedImages = new List<AugmentedImage>();
7
8
9   public void Update()
10      {
11          Session.GetTrackables<AugmentedImage>(m_TempAugmentedImages, TrackableQueryFilter.Updated);
12
13          foreach (var image in m_TempAugmentedImages)
14          {
15              AugmentedImageVisualizer visualizer = null;
16              m_Visualizers.TryGetValue(image.DatabaseIndex, out visualizer);
17              if (image.TrackingState == TrackingState.Tracking && visualizer == null)
18              {
19
20                  Anchor anchor = image.CreateAnchor(image.CenterPose);
21                  visualizer = (AugmentedImageVisualizer)Instantiate(AugmentedImageVisualizerPrefab, anchor.transform);
22                  visualizer.Image = image;
23                  m_Visualizers.Add(image.DatabaseIndex, visualizer);
24              }
25              else if (image.TrackingState == TrackingState.Stopped && visualizer != null)
26              {
27                  m_Visualizers.Remove(image.DatabaseIndex);
28                  GameObject.Destroy(visualizer.gameObject);
29              }
30
31          }
```

Listing 7: erweiterte Update Funktion als Auszug aus der „Controller“ Klasse

Für die Verdeutlichung der Darstellung in der Anwendung wird ein Rahmen um erkannte Bilder gezogen. Dieser wird mit Hilfe eines Prefabs erstellt, sobald ein Bild aus der Datenbank erfolgreich erkannt wurde. Dieser vorgefertigte Baustein wird mit Zeile 1 integriert. Über den Inspektor wird das vorgefertigte Element an das Skript verlinkt. Die Positionsanzeigen der einzelnen Rahmen erfolgt in der Art eines privaten Lexikons. Dieses wird in Zeile 3 und 4 für alle „m_Visualizers“ mit einer festen Id und den dazugehörigen Positionen im Raum als Variablenwerte erstellt. Als Nächstes wird eine private Liste vom Typ „AugmentedImage“ für alle „m_TempAugmentedImages“ initiiert. Dort werden alle erkannten Bilder zur Verwendung hinein gespeichert. Erkannt werden diese mit dem Befehl „GetTrackables“ für alle Elemente des Typs „AugmentedImage“ aus Zeile 11. In der danach folgenden for-Schleife wird für jedes erkannte Bild ein passender Rahmen erzeugt.

Zunächst wird ein Anzeiger mit einem Wert von null[8] erstellt und jedem Bild einen Anzeiger zugewiesen beziehungsweise gefragt, ob schon einer mit einem anderen Wert als null vorhanden ist. Danach wird mit der if-Schleife gefragt, ob das Bild gerade getrackt wird und der Anzeiger noch nicht vorhanden ist. Sollten beide Bedingungen erfüllt sein, dann wird für den Mittelpunkt des Bildes ein Anker in Abhängigkeit von der realen Umgebung erstellt, um das Bild weiterhin tracken zu können, auch wenn die Kamera schon weiter bewegt wurde. Dies erfolgt mit Hilfe des „CreateAnchor“ Befehls in Zeile 20. Danach wird mit den Daten des Ankers das Prefab für den Rahmen initialisiert, an die Größe des Bildes angepasst und hinzugefügt, so dass es die äußeren vier Ecken umfasst. Wenn jedoch eine der beiden Vorbedingungen nicht wahr ist, dann geht die Schleife in den „else“ Teil über und prüft eine neue Bedingung. Dafür muss sowohl das Bild nicht mehr getrackt werden, als auch ein Anzeiger für das Bild vorhanden sein. Sollte beides zutreffen, wird die Umgebung aufgeräumt. Ein Anzeiger für das Bild wird entfernt und das dargestellte Objekt wird mit dem Befehl „Destroy“ aus Zeile 28 endgültig gelöscht.

Wie ein erkanntes Bild mit Rahmen aussieht, wird mit Abbildung 17 gezeigt.

Abbildung 18: Im Prototypen erkanntes Bild

[8]Hier ist der leere Wert eine Variable gemeint. Nicht der Zahlenwert null.

4.2 Die Charakter und Indoor Navigations Komponente

Die wichtigen Aspekte der Augmented Reality sind erfolgreich implementiert wurden. Zur Vervollständigung des Prototypens fehlt noch der Charakter, der den Nutzer von Bild zu Bild führt und dabei wichtige Informationen zu den einzelnen Objekten gibt. Dafür wurde ein Charakter aus dem Asset Store ausgewählt.

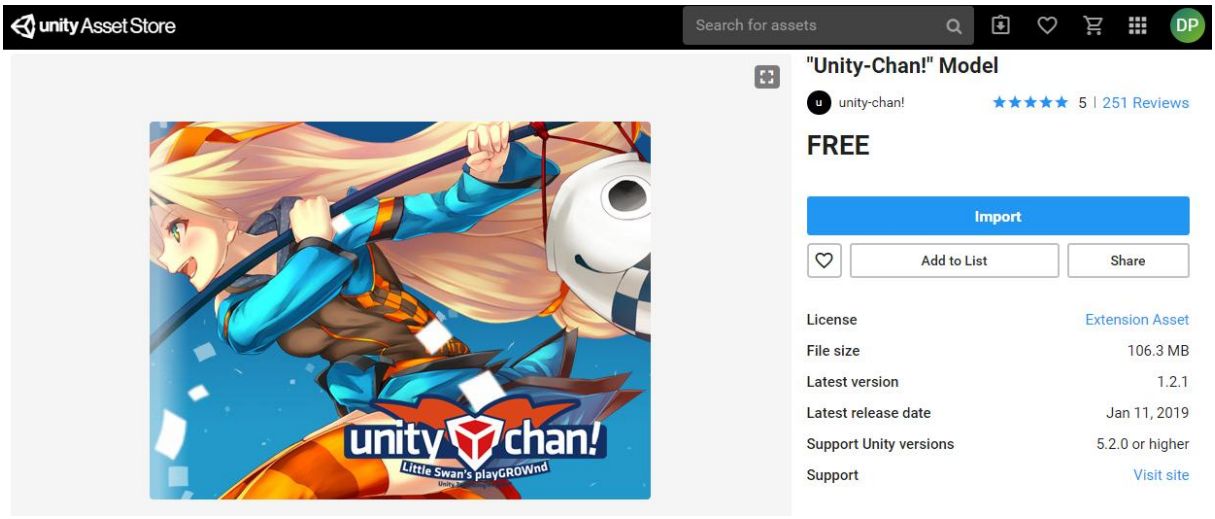


Abbildung 19: Unity-chan Package im Asset Store

In Abbildung 18 ist der Charakter für den Prototypen dargestellt. Dieser ist „Unity-Chan“, ein Charaktermodel zur kostenlosen Nutzung von der japanischen Unityabteilung. Mit Hilfe des „Import“ Buttons kann dieses Package während der Nutzung von Unity in das derzeitige Projekt integriert und somit sofort verwendet werden. Für die Nutzung existiert in dem Ordner ein Prefab mit ihrem Charakter und all ihren Einstellungen und Möglichkeiten. Wie diese „Figur“ in der Anwendung effektiv integriert werden kann wird mit Hilfe der folgenden Listing 8[9] und 9 erläutert.

[9]siehe Anhang 2

```
45                  var gameObject = Instantiate(prefab, hit.Pose.position, hit.Pose.rotation)
46
47                  gameObject.transform.localScale -= new Vector3(6/8, 6/8, 6/8);
48
49                  Vector3 cameraPositionSameY = FirstPersonCamera.transform.position;
50                      cameraPositionSameY.y = hit.Pose.position.y;
51
52                  gameObject.transform.LookAt(cameraPositionSameY,
53                      gameObject.transform.up);
54
55                  var anchor = hit.Trackable.CreateAnchor(hit.Pose);
56
57                  gameObject.transform.parent = anchor.transform;
58
59                  Characterexist = true;
60              }else
61              {
62                  return;
63              }
64          }
65      }
```

Listing 9: Fortsetzung von Listing 8

Listing 8 und Listing 9 sind weitere Auszüge aus der „Controller“ Klasse. Zuerst wird ein neues öffentliches GameObject erstellt. Dieses entspricht dem Prefab für den Charakter. In Unity wird das Prefab Objekt für Unity-chan in den Inspektor der Klasse in dieses neu erstellte Feld gezogen und wird dadurch mit der Szene und dem Programm verknüpft. Eine boolsche Variable mit dem Titel „Characterexist“ prüft, ob Unity-chan schon einmal erstellt wurde. Für die Anwendung soll nur ein Charakter existieren, so dass der Nutzer nur einen erstellen darf. Beim Hinzufügen eines neuen Prefabs in der Augmented Reality Umgebung muss dieses Objekt gedreht werden, damit es Richtung Kamera schaut. Dies geschieht mit der privaten konstanten Gleitkommazahl „k_PrefabRotation“. Als Nächstes wird die in Listing 4[10] angesprochene Variable „hit“ vom Typ „TrackableHit“ verwendet, beziehungsweise den Strahl, der in die Augmented Reality Umgebung projiziert wird, ausgehend von der Nutzer Eingabe auf das Display. Mit der Abfrage, ob so ein Strahl existiert, werden die Endpositionen des Strahls bestimmt. Dies entspricht einer X- und einer Y-Position. Mit der Bedingung aus Zeile 17 wird geprüft, ob mit dem Strahl eine Ebene erfolgreich getroffen wurde, aber ob die Ebene von unterhalb berührt wurde. Sollte dem so sein, dann besteht kein Grund für weitere Berechnungen und es wird ein Hilfetext ausgegeben, dass die Rückseite der aktiven Ebene getroffen wurde. Interaktionen mit Ebenen können immer nur **auf** und niemals hinter diesen stattfinden. Danach wird geprüft, ob schon ein Charakter in der Umgebung existiert.

---

[10]Seite 39

Wenn ja, dann soll nichts weiter geschehen und die Eingabe des Nutzers auf eine Ebene wird ignoriert. Sollte die Variable „Characterexist“ falsch sein, dann wird versucht ein neues Charakterprefab an der Endposition des Strahls zu erstellen. Dabei wird ein neues GameObject mit dem Titel „prefab“ erzeugt. Dieses wird in Abhängigkeit von der getroffenen Ebene beschrieben. In Zeile 27 wird überprüft, ob der Strahl eine Ebene getroffen hat, die die Anwendung schon als Ebene erkannt hat. Ist dies der Fall, dann wird kontrolliert von welcher Art diese Ebene ist. Sollte es eine vertikale Ebene sein, dann soll Unity-chan nicht erscheinen. Nur bei einer horizontalen soll sie erstellt werden. Dafür wird das vorher erzeugte GameObject mit dem Prefab aus dem Inspektor beschrieben.

Mit Listing 9 wird der Prozess fortgesetzt. Hier wird ein neues GameObject vom Typ „gameObject“ mit den Eigenschaften von dem Prefab, der Position und der Rotation des Strahls, erstellt. Als Nächstes wird in Zeile 47 die Größe von Unity-chan herunter skaliert, da die normale Größe dieses Charakters zu gewaltig wäre. Mit Zeile 49 wird eine Vektor 3 Variable namens „cameraPositionSameY“ mit der y-Koordinate der Smartphonekamera beschrieben. Dadurch kann mit dem Befehl „LookAt“ aus Zeile 52 dafür gesorgt werden, dass Unity-chan beim Erstellen in die Kamera schaut. Als letztes wird noch ein Anker für die derzeitige statische Position des Charakters erstellt und dieses Objekt wird dem Anker als Kind hinzugefügt, so dass eine Verbindung zwischen diesen beiden entsteht. Zusätzlich wird die Abfragevariable „Characterexist“ auf wahr gesetzt.

Die folgende Abbildung 19 auf der nächsten Seite zeigt den erstellten Charakter in der Augmented Reality Umgebung.

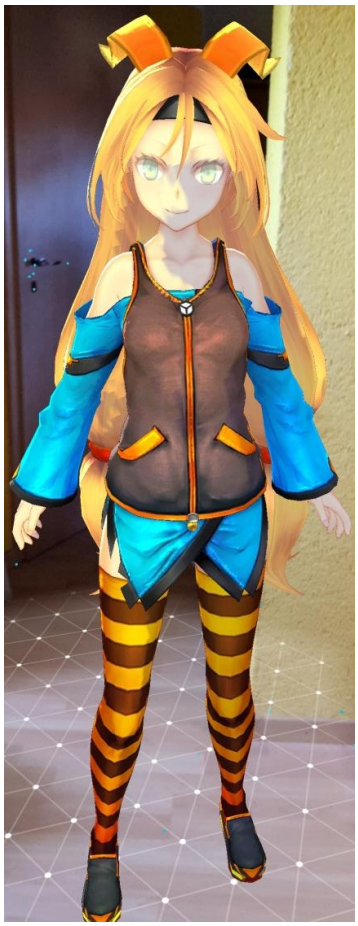


Abbildung 19:Unity-chan im Prototypen

Nachdem der Charakter erfolgreich erstellt wurde, soll er mit dem Nutzer kommunizieren, von Bild zu Bild laufen und Informationen zu den Bildern erzählen. Davor muss jedes benötigte Bild nach dem Starten der Anwendung einmal getrackt werden, so dass der Anker und die Position in der realen Umgebung zur Verfügung stehen. Danach muss dafür gesorgt werden, dass Unity-chan immer in Richtung Kamera schaut, wenn sie an einer festen Stelle steht. Dafür wird folgendes Listing 10 betrachtet.

```
1   private void _UpdateCharacter()
2       {
3
4           if (character == null && Characterexist)
5           {
6               character = GameObject.Find("characterObject");
7           }
8           else if (Characterexist && (moving = false))
9           {
10              character.transform.rotation = Quaternion.RotateTowards(character.transform.rotation,
11                  FirstPersonCamera.transform.rotation, 5f);
12          }
13
14      }
```

Listing 10: _UpdateCharacter Funktion aus der „Controller“ Klasse

Hier wird eine neue Unterfunktion zur „Update“ Prozedur in der „Controller“ Klasse hinzugefügt. Diese trägt den Namen „_UpdateCharacter“. In dieser Funktion wird ein neues GameObject angesprochen und geprüft, ob dieses leer ist. Zusätzlich wird abgefragt, ob der Charakter schon erschienen ist. Sollte beides zutreffen, dann wird das leere Objekt mit dem Befehl „GameObject.Find“ mit dem zuvor erstellten Charakter beschrieben. Nun kann damit gearbeitet werden kann. Die nächste Abfrage prüft, ob der Charakter existiert und sich gerade nicht bewegt. Wenn das zutreffen sollte, dann richtet er sich zur Kamera aus. Dies geschieht mit dem Befehl aus Zeile 10 und 11. Dort wird Unity-chan gedreht, sobald der Winkel zwischen ihrem Blickfeld und dem der Kamera mehr als 5° beträgt.

Mit einem Fingertipp auf den Charakter soll er sich zu dem ersten Bild bewegen, sich neben dieses stellen und per Textfeld etwas dazu erzählen. Mit jedem weiteren Fingertipp läuft Unity-chan zu einem nächsten Bild, bis alle Bilder aufgesucht wurden. Sie verschwindet nach kurzer Zeit, wenn ein letzter Fingertipp erfolgte. Diese Befehle sind in Listing 11 und Listing 12 noch einmal zusammengefasst.

```
1   {
2       if (hit.Trackable is character)
3       {
4           if (characterHit == 0)
5           {
6               imageCount = m_TempAugmentedImages.Count;
7
8               GameObject text0 = new GameObject("startText");
9               text0.transform.SetParent(this.transform);
10
11              Text imagetext0 = text0.AddComponent<Text>();
12              imagetext0.text = "Willkommen zur heutigen Führung. Ich bin Unity-chan und " +
13                                "werde dich durch das Museum begleiten";
14          }
15          if (characterHit == 1)
16          {
17              GameObject.Destroy(imagetext0);
18              character.transform.position = Vector3.MoveTowards(character.transform.position,
19                  m_TempAugmentedImages[0].CenterPose.position, 1.0f);
20
21              yield return new WaitForSeconds(5);
22
23              if (character.transform.position == m_TempAugmentedImages[0].CenterPose.position)
24              {
25                  GameObject text1 = new GameObject("imagetextErde");
26                  text1.transform.SetParent(this.transform);
27
28                  Text imagetext1 = text1.AddComponent<Text>();
29                  imagetext1.text = "Dies ist eine Aufnahme der Erde aus dem All.";
30              }
31          }
32
33          if (characterHit == 2)
34          {
49
50          if (characterHit == 3)
51          {
```

Listing 11: Auszug der Charakterbewegung aus der „Controller“ Klasse

Listing 11 ist eine Fortsetzung der „Update“ Funktion aus der „Controller“ Klasse. Nachdem der Charakter erstellt wurde, weil eine Ebene durch den Strahl getroffen wurde, wird jetzt geprüft, ob der Strahl Unity-chan berührt. Wenn er sie erfasst hat, dann werden verschiedene Abfragen getätigt. Dafür wird ein Vergleichswert namens „characterHit“ herangezogen. Dieser wird bei jedem Fingertipp auf den Charakter um eins erhöht. Zu sehen ist dies in Listing 12. Bei dem ersten Fingertipp ist die Variable noch auf null und damit wird die Führung zu den einzelnen Bildern begonnen. In Zeile 6 wird zu allererst gezählt, wie viele Bilder erkannt wurden. Das wird der Vergleichswert zum Beenden der Führung. Als Nächstes wird ein Textfeld erstellt, damit Unity-chan mit dem Nutzer kommunizieren kann. Diesem wird ein bestimmter Text in Abhängigkeit von der Variablen „characterHit“ zugewiesen, wie in Zeile 12 zu erkennen ist. Dieses Textelement wird zerstört, wenn sie durch Tippen zum nächsten Bild geschickt wird. Das dauert fünf Sekunden. Dann erfolgt die Abfrage, ob sie an der gewünschten Position angekommen ist. Wenn ja, öffnet sich ein neues Textfenster mit den Informationen zu diesem Bild. Der geschilderte Vorgang wiederholt sich, bis alle Bilder durchlaufen sind. Listing 12 zeigt, was nach dem letzten Bild passiert.

```
66
67          if (characterHit > imageCount)
68          {
69              GameObject.Destroy(imagetext3);
70
71              GameObject text4 = new GameObject("endText");
72              text4.transform.SetParent(this.transform);
73
74              Text imagetext4 = text4.AddComponent<Text>();
75              imagetext4.text = "Hiermit ist die Führung beendet!";
76
77              yield return new WaitForSeconds(5);
78
79
80              GameObject.Destroy(character);
81              Characterexist = false;
82              characterHit = -1;
83          }
84
85          characterHit++;
86      }
87  }
```

Listing 12: Fortsetzung von Listing 11

Dieses Listing entspricht der Fortsetzung von Listing 11. Hier wird zuerst eine Abfrage gestellt, ob alle Bilder betrachtet wurden. Da nach jedem Bild die „characterHit“ Variable um eins erhöht wurde, das ist in Zeile 85 zu sehen, wird nun geprüft, ob diese größer ist, als die Anzahl aller Bilder. Ist dies der Fall, dann wird der vorherige Text wieder gelöscht, damit eine Abschlussnachricht gezeigt werden kann. Dieser beendet die Führung. Fünf Sekunden später wird Unity-chan verschwinden, da ihr GameObject gelöscht wurde. Die Variable, dass sie existiert, wird auf falsch zurückgesetzt, wodurch jederzeit ein neuer Charakter platziert werden kann, um so eine neue Führung durch die verschiedenen Bilder zu starten.

5 Fazit und Ausblick

Das Ergebnis der vorliegenden Arbeit ist ein funktionstüchtiger Prototyp. Dieser besitzt alle nötigen Funktionen, um die gestellten Leitfragen positiv zu beantworten. Folgende Betrachtungen lassen sich zusammenfassend sagen:

Eine Augmented Reality Anwendung ist für Museen eine kostengünstige Alternative, um neue interaktive Möglichkeiten für Besucher zu schaffen. Dies wird durch die Tatsache bekräftigt, dass heutzutage fast jeder ein eigenes Smartphone besitzt. Dadurch kann sich ein Museum Anschaffungskosten für neue Hardware fast komplett sparen und alles in die Entwicklung dieser App investieren.

Die Augmented Reality Umgebung auf Basis von ARCore lässt sich einfach und für viele heutige mobile Endgeräte bereitstellen. Vorrausgesetzt wird ein Android Betriebssystem, die benötigte Softwareversion und eine Rückseitenkamera. Die Verfügbarkeit der Anwendung für andere Betriebssysteme ist ebenso möglich, jedoch mit wesentlich größerem Aufwand verbunden.

Durch die vielen verschiedenen Einstellungsmöglichkeiten bietet Unity eine nahezu perfekte Plattform für die Entwicklung des Prototypen. Hier können eine große Anzahl von Schnittstellen einfach und leicht miteinander verknüpft werden. Zusätzlich unterstützt ARCore Unity mit einem Asset, welches für dieses Programm entwickelte wurde. Dieses erleichtert die Erstellung der App sehr.

Im Test hat sich gezeigt, dass nur eine Verwendung von AR tatsächlich in der Lage ist, auch Indoor Navigation auf kleinster Basis zu ermöglichen. Der Charakter läuft von Objekt zu Objekt. Im Rahmen dieser Arbeit dienen Bilder als Beispielobjekte. Eine Ausstellung über mehrere Stockwerke gestaltet sich mit diesem Programm schwierig, da der Charakter nicht ohne Weiteres seine Ebene verändern kann. Hinzu kommt, dass nur Positionen für bis zu 20 Objekte vorher gespeichert werden können. [vgl. Arc03] Dadurch sinkt die Nutzbarkeit und Relevanz des Prototypen. Trotzdem ist er einsatzfähig und erfüllt die erwünschten Funktionen. Damit der Prototyp in Zukunft als hilfreiche Anwendung für kulturelle Einrichtungen, wie Museen, verwendet werden kann, sind einige weitere Implementierungen notwendig. Um möglichst viele Ausstellungsstücke anlaufen zu können, müssen Augmented Reality und Indoor Navigation strikt von einander getrennt werden. Mit der zusätzlichen Nutzung von Beacons kann dafür gesorgt werden, dass mehr als nur 20 Objekte von dem Charakter gezeigt werden können. Dadurch vervielfältigen sich die Möglichkeiten einer solchen digitalen Führung für Museumsbesucher. Zusätzlich verbessert sich die Beweglichkeit und die Reaktionsmöglichkeiten des Charakters, wenn er sich anhand von Wänden des Gebäudes oder an Beacons orientieren kann. Als weitere Steigerung können die Informationen für verschiedene Bilder aus dem Code gelöst und in einer zentralen Datenbank gespeichert werden. Dadurch wird die Übersichtlichkeit der Daten verbessert, sowie die Möglichkeit diese anzupassen, hinzuzufügen oder zu löschen leichter realisierbar. Ansonsten müsste ständig die Anwendung mit den veränderten Daten neu aufgespielt werden. So muss nur eine einzelne Datenbank regelmäßig gepflegt werden.

Zusätzlich kann die Anwendung mit einem VR-Modul für Zuhause erweitert werden. Dies würde dem Nutzer der App ermöglichen, sich vor einem Museumsantritt schon über ausgewählte Museumsstücke zu informieren. Dadurch kann spielerisch das Interesse für einen Museumsbesuch gesteigert werden.

Abbildungsverzeichnis

Listingsverzeichnis

Literaturverzeichnis

Arc01 - https://developers.google.com/ar/discover/concepts

Aufgerufen am 24.11.2019

Arc02 - https://developers.google.com/ar/develop/unity/quickstart-android

Aufgerufen am 24.11.2019

Arc03 - https://developers.google.com/ar/develop/c/augmented-images

Aufgerufen am 24.11.2019

Arfnd01 - https://unity.com/de/unity/features/arfoundation

Aufgerufen am 25.11.2019

Arfnd02 - https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@1.0/manual/index.html

Aufgerufen am 25.11.2019

Ark01 - https://developer.apple.com/augmented-reality/

Aufgerufen am 25.11.2019

Arminds01 - https://www.augmented-minds.com/de/erweiterte-realitaet/was-ist-augmented-reality/

Aufgerufen am 23.11.2019

Arvuf01 - https://developer.vuforia.com/

Aufgerufen am 25.11.2019

Arvuf02 - https://www.ptc.com/en/products/augmented-reality

Aufgerufen am 25.11.2019

Arvuf03 - https://developer.vuforia.com/vui/pricing

Aufgerufen am 25.11.2019

Ble01 - https://www.gruenderszene.de/lexikon/begriffe/bluetooth-low-energy?interstitial

Aufgerufen am 03.12.2019

Ble02 - https://www.elektronik-kompendium.de/sites/kom/1805171.htm
Aufgerufen am 03.12.2019

Ble03 - https://de.ryte.com/wiki/Beacon
Aufgerufen am 03.12.2019

Gps01 - https://www.magicmaps.de/produktinfo/gps-grundlagen/wie-funktioniert-gps.html
Aufgerufen am 03.12.2019

Icom01 - http://www.icom-deutschland.de/schwerpunkte-museumsdefinition.php
Aufgerufen am 20.11.2019

Ieee01 - http://www.cse.msu.edu/~cse870/IEEEXplore-SRS-template.pdf
Aufgerufen am 20.02.2020

InNav01 - https://www.gruenderszene.de/lexikon/begriffe/indoor-navigation?interstitial
Aufgerufen am 01.12.2019

InNav02 - https://www.pcwelt.de/ratgeber/Indoor-Navigation_im_Praxistest_zeigt_Licht_und_Schatten-Google_Indoor_Maps-8976212.html
Aufgerufen am 01.12.2019

InNav03 - https://www.infsoft.com/de/indoor-positionsbestimmung
Aufgerufen am 01.12.2019

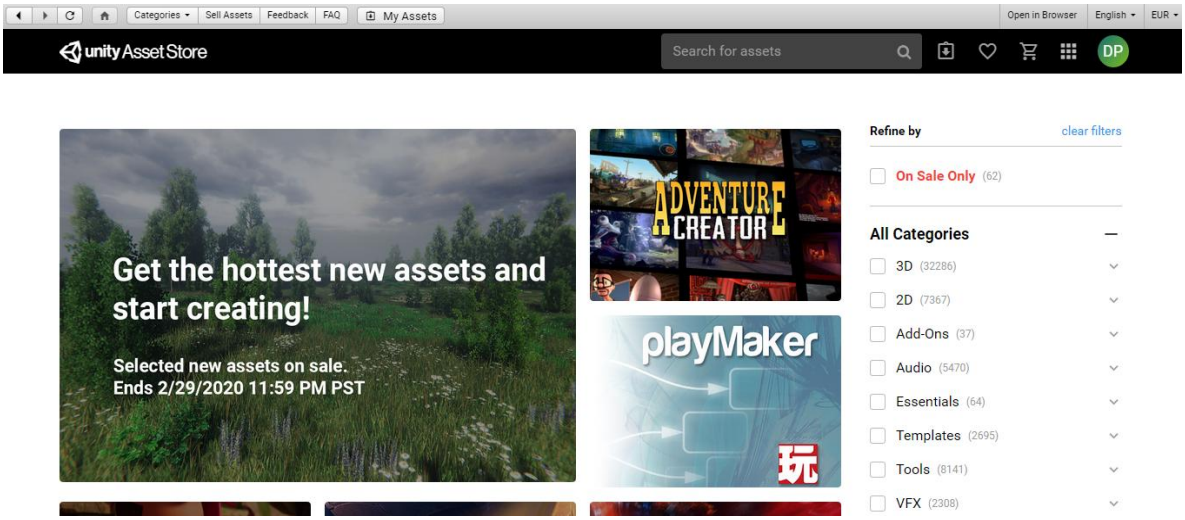
Inte01 - https://de.statista.com/statistik/daten/studie/37304/umfrage/besuchszahlen-in-deutschen-museen-seit-1998/
Aufgerufen am 26.02.2020

Inte02 - https://de.statista.com/statistik/daten/studie/246004/umfrage/weltweiter-bestand-an-smartphones-nach-betriebssystem/
Aufgerufen am 22.11.2019

LatPri01 - https://de.wikipedia.org/wiki/Lateration
Aufgerufen am 04.12.2019

MaLeap01 - https://www.magicleap.com/
Aufgerufen am 12.12.2019
Micr01 - https://www.mixed-reality-training.com/uebersicht-microsoft-mixed-reality/
Aufgerufen am 23.11.2019
MicrHolo01 - https://www.microsoft.com/en-us/hololens
Aufgerufen am 12.12.2019
MicrHolo02 - https://www.microsoft.com/en-us/hololens/buy
Aufgerufen am 12.12.2019
NormIE01 - https://www.johner-institut.de/blog/tag/software-anforderungen/
Aufgerufen am 06.12.2019
NormIso01 - https://www.johner-institut.de/blog/iec-62304-medizinische-software/iso-9126-und-iso-25010/
Aufgerufen am 06.12.2019
Rfid01 - https://www.infsoft.com/de/technologie/sensorik/rfid
Aufgerufen am 03.12.2019
Ucase01 - https://t2informatik.de/wissen-kompakt/use-case/
Aufgerufen am 15.12.2019
Uwb01 - https://www.ultrawideband.de/de/technologie.php
Aufgerufen am 03.12.2019
Wiki01 - https://de.wikipedia.org/wiki/Erweiterte_Realität
Aufgerufen am 20.10.2019

Anhang



Anhang 1: Der Asset Store

```
1   public class Controller : MonoBehaviour
2       {
3           public GameObject GameObjectHorizontalPlanePrefab;
4
5           private bool Characterexist = false;
6
7           private const float k_PrefabRotation = 180.0f;
8
9           public void Update()
10          {
11              TrackableHit hit;
12              TrackableHitFlags raycastFilter = TrackableHitFlags.PlaneWithinPolygon |
13                  TrackableHitFlags.FeaturePointWithSurfaceNormal;
14
15              if (Frame.Raycast(touch.position.x, touch.position.y, raycastFilter, out hit))
16              {
17                  if ((hit.Trackable is DetectedPlane) &&
18                      Vector3.Dot(FirstPersonCamera.transform.position - hit.Pose.position,
19                          hit.Pose.rotation * Vector3.up) < 0)
20                  {
21                      Debug.Log("Hit at back of the current DetectedPlane");
22                  }
23                  else if (Characterexist == false)
24                  {
25
26                      GameObject prefab;
27                      if (hit.Trackable is DetectedPlane)
28                      {
29                          DetectedPlane detectedPlane = hit.Trackable as DetectedPlane;
30                          if (detectedPlane.PlaneType == DetectedPlaneType.Vertical)
31                          {
32                              prefab = null;
33                          }
34                          else
35                          {
36
37                              prefab = GameObjectHorizontalPlanePrefab;
38                          }
39                      }
40                      else
41                      {
42                          prefab = null;
43                      }
```

Anhang 2: Listing 8: Auszug zur Charaktererstellung aus der „Controller“ Klasse