

---

# D I P L O M A R B E I T

## Applied Augmented Reality in Education

### Ausgeführt im Schuljahr 2034/24 von:

Recherche zu Varianten von Knapsack-Algorithmen und Umsetzung des Knapsack-Problems als AR-Anwendungsszenario inkl. Dokumentation || Erstellen/Auswerten eines Feedbackfragebogens zur Lernunterstützung

Moritz SKREPEK 5CHIF

Design und Umsetzung der 3D-Objekte zur AR-Abbildung || Analyse der Steuerungsmöglichkeiten (Menüführung, Gesten, ...) und Erstellen der Benutzeroberfläche für die AR-Applikation mit Fokus auf UX

Dustin LAMPEL 5CHIF

Erfassen realer Objekte und kontextgerechte Überlagerung der Realität mit AR-Device || Tagging v. realen Elementen mittels QR-Codes für Tracking || Unit-Tests für d. implementierten Knapsack-Algorithmus

Seref HAYLAZ 5CHIF

Evaluierung/Auswahl Laufzeit-/Entwicklungsumgebung für Umsetzung der Applikation und Integration mit AR-Device inkl. Recherche || Konzeption/Umsetzung des Anwendungsszenarios im Bereich Netzwerktechnik

Jonas SCHODITSCH 5CHIF

### Betreuer / Betreuerin:

Mag. BEd. Reis Markus

---

Wiener Neustadt, am 15. März 2024

Abgabevermerk:

Übernommen von:

# Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Wiener Neustadt, am 15. März 2024

## Verfasser / Verfasserinnen:

Moritz SKREPEK

Dustin LAMPEL

Seref HAYLAZ

Jonas SCHODITSCH

# Inhaltsverzeichnis

<b>Eidesstattliche Erklärung</b>	i
<b>Vorwort</b>	vii
<b>Diplomarbeit Dokumentation</b>	viii
<b>Diploma Thesis Documentation</b>	x
<b>Kurzfassung</b>	xii
<b>Abstract</b>	xiii
<b>1 Einleitung</b>	1
1.1 Ausgangslage . . . . .	1
1.2 Auslöser . . . . .	1
1.3 Aufgabenstellung . . . . .	1
1.4 Team . . . . .	2
1.4.1 Aufteilung . . . . .	2
<b>2 Grundlagen</b>	3
2.1 Vorgehensmodelle . . . . .	3
2.1.1 Scrum . . . . .	3
2.1.1.1 Die drei Rollen in Scrum . . . . .	3
2.1.1.2 Scrum Meetings . . . . .	4
2.1.2 Begründung der Auswahl . . . . .	5
2.2 Projektmanagement-Tools . . . . .	6
2.2.1 GitHub . . . . .	6
2.2.2 Jira . . . . .	6
2.3 Fragebögen . . . . .	7
2.3.1 Konzeption von Fragebögen . . . . .	7
2.3.2 Planung der Fragebogenkonstruktion . . . . .	7
2.3.3 Formulierung der Fragen . . . . .	9
2.3.4 Arten von Fragen . . . . .	10
2.3.5 Struktur und Gliederung von Fragebögen . . . . .	11
2.3.6 Mögliche Verfälschung des Resultats . . . . .	11
2.3.7 Auswertung von Fragebögen . . . . .	12
2.3.8 Planung und Erstellung eines Fragebogens . . . . .	12
2.3.9 Struktur und verwendete Fragetypen . . . . .	12
2.3.10 Pilotstudio des Fragebogens . . . . .	13
2.3.11 Auswertung des erstellten Fragebogens . . . . .	13

<b>3 Produktspezifikationen</b>	<b>17</b>
3.1 Anforderungen und Spezifikationen . . . . .	17
3.1.1 Use Cases . . . . .	17
3.2 Design . . . . .	17
3.2.1 Abläufe . . . . .	17
3.3 Eingesetzte Technologien . . . . .	17
3.3.1 Microsoft HoloLens2 . . . . .	17
3.3.2 Kriterien . . . . .	18
3.3.3 Game Engine . . . . .	18
3.3.3.1 Unity . . . . .	18
3.3.3.2 Unreal Engine . . . . .	18
3.3.3.3 Game Engine Auswahl und Wechsel im Projektverlauf . . . . .	19
3.3.4 Unity foundation packages . . . . .	20
3.3.4.1 MRTK3 . . . . .	20
3.3.4.2 Microsoft OpenXR Plugin . . . . .	21
3.3.4.3 Integrationsprozess der Plugins . . . . .	21
3.3.5 Wahl des Modellierungsprogramm . . . . .	22
3.3.5.1 Wie funktioniert Blender im Allgemeinen? . . . . .	22
<b>4 Feinkonzept und Realisierung</b>	<b>24</b>
4.1 Entwicklungsumgebungen . . . . .	24
4.1.1 Visual Studio 2022 . . . . .	24
4.1.2 Unity . . . . .	24
4.1.2.1 Multidisziplinäre Unterstützung und Integration . . . . .	24
4.1.2.2 Szenengestaltung und Asset-Management . . . . .	24
4.1.2.3 Programmierung und Skripterstellung . . . . .	25
4.1.2.4 Unterstützung für Augmented Reality (AR) und Virtual Reality (VR) . . . . .	25
4.1.2.5 Erweiterte Debugging- und Profiling-Werkzeuge . . . . .	25
4.1.2.6 Aufbau einer Unity-Applikation . . . . .	25
4.1.2.7 Lebenszyklusmethoden in Unity . . . . .	25
4.1.2.8 Unity Szenen . . . . .	26
4.1.2.9 Unity Manager . . . . .	26
4.1.2.10 Unity GameObjects und Komponente . . . . .	27
4.1.2.11 Unity Prefabs . . . . .	28
4.1.3 Deployment der Anwendung . . . . .	29
4.1.3.1 Voraussetzungen für das Deployment . . . . .	29
4.1.3.2 Deployment-Prozess . . . . .	29
4.1.3.3 Erstmaliges Deployment . . . . .	31
4.2 Objektdesign . . . . .	31
4.2.1 Texturen . . . . .	32
4.2.1.1 Bild-Texturen in der 3D-Modellierung . . . . .	32
4.2.2 Mesh . . . . .	33
4.2.3 Rolle von Polygonen in einem Modell . . . . .	33
4.2.4 Optimierung der einzelnen Modelle . . . . .	34
4.2.4.1 Polygonreduktion . . . . .	34
4.2.4.2 Texturenoptimierung . . . . .	35
4.2.5 Export- und Integrationsprozess . . . . .	35
4.2.5.1 Dateiformat . . . . .	35

4.2.5.2	Koordinatensysteme . . . . .	35
4.2.6	Add-Ons und Plugins . . . . .	36
4.2.6.1	Looptools: Optimierung von Topologie und Oberflächen . .	36
4.2.6.2	Images as Planes: Effiziente Integration von Texturen . . .	37
4.2.7	Modi . . . . .	37
4.2.7.1	Object-Modus . . . . .	37
4.2.7.2	Edit-Modus . . . . .	38
4.2.7.3	Texture-Paint-Modus . . . . .	38
4.2.8	Hierarchie . . . . .	38
4.2.9	Modifier . . . . .	39
4.2.10	Modellierung von Gegenständen für das Projekt . . . . .	40
4.2.10.1	Taschenrechner . . . . .	40
4.2.10.2	Restliche Modelle . . . . .	46
4.3	Hauptmenü . . . . .	48
4.3.1	Erstentwurf . . . . .	48
4.3.1.1	Probleme beim Erstentwurf . . . . .	50
4.3.2	Finalversion des Menüs . . . . .	51
4.3.2.1	Finalversion Menü - Hauptmenü . . . . .	51
4.3.2.2	Finalversion Menü - Anwendungsszenario . . . . .	53
4.3.2.3	Setzen des Menüs . . . . .	53
4.3.3	Laden der Level . . . . .	54
4.3.4	UI/UX . . . . .	55
4.4	Nachrichtenaustausch Anwendungsscenario . . . . .	56
4.4.1	Aufbau von Nachrichtenaustausch Anwendungsscenario . . . . .	56
4.4.2	Kabelerkennung . . . . .	57
4.4.3	Start der Szene . . . . .	57
4.4.4	Foto aufnahme und verarbeitung . . . . .	59
4.4.4.1	Photocapture Klasse . . . . .	59
4.4.4.2	Unity Canvas . . . . .	59
4.4.4.3	Foto der Umgebung aufnehmen . . . . .	60
4.4.4.4	Unity Job System . . . . .	61
4.4.4.5	Vector2 Klasse . . . . .	61
4.4.4.6	Fotoverarbeitung . . . . .	61
4.4.4.7	Kabel finden . . . . .	62
4.4.5	Coroutines . . . . .	64
4.4.6	Lade Symbol . . . . .	65
4.4.7	Bewegung des Paketes . . . . .	66
4.4.7.1	ARRaycastHit Klasse . . . . .	67
4.4.7.2	Entfernung messen . . . . .	67
4.4.7.3	Erstellung und Positionierung des Paketes . . . . .	68
4.4.7.4	Bewegung des Paketes . . . . .	70
4.4.8	Anzeigen von Informationen der Nachricht . . . . .	71
4.4.9	Debugging Optionen . . . . .	72
4.4.9.1	Anzeigen der Kabelerkennung . . . . .	72
4.4.9.2	Debug Funktion von Raycasts . . . . .	72
4.4.9.3	Debug Funktion von der Kamera Benutzung . . . . .	73
4.4.10	Andere Versuche / Probleme bei der Programmierung . . . . .	73
4.4.10.1	Kabel Erkennung Version 1 . . . . .	74
4.4.11	Starten und Nutzen der Webseite . . . . .	75

4.4.11.1	Webseite starten . . . . .	75
4.4.11.2	CORS . . . . .	77
4.4.11.3	Bedienung der Webseite . . . . .	78
4.4.12	Frontend der Webseite . . . . .	78
4.4.12.1	Designprozess . . . . .	78
4.4.12.2	Verwendete Programmiersprachen . . . . .	80
4.4.12.3	Funktionen der Webseite . . . . .	81
4.4.13	Backend der Webseite . . . . .	82
4.4.13.1	Protokoll zur Datenübertragung . . . . .	83
4.4.13.2	Bearbeitung von Anfragen . . . . .	84
4.4.14	Object Tracking . . . . .	85
4.4.15	Kurvenberechnung . . . . .	85
4.5	Knapsack Problem Szenario . . . . .	85
4.5.1	Knapsack-Problem Szenario Hirarchie . . . . .	85
4.5.2	Nutzung von QR-Codes . . . . .	87
4.5.2.1	Struktur und Inhalt eines QR-Codes . . . . .	87
4.5.2.2	QR-Code-Tracking . . . . .	88
4.5.2.3	Interaktion mit QR-Codes . . . . .	91
4.5.2.4	Positionsbestimmung von QR-Codes . . . . .	92
4.5.2.5	Visualisierung von QR-Codes . . . . .	92
4.5.2.6	Spatial Graph Node . . . . .	95
4.5.2.7	Zugriff auf QR-Codes bereitstellen . . . . .	95
4.5.3	Platzieren des Inventar Prefabs . . . . .	96
4.5.3.1	Aufbau und Hirarchie des Inventar-Prefabs . . . . .	96
4.5.3.2	User Gaze in AR-Anwendungen . . . . .	97
4.5.3.3	Der Inventory Placement Controller . . . . .	98
4.5.3.4	Implementierung des User Gazes . . . . .	99
4.5.3.5	Platzierungskontrolle und ARPlane-Überwachung . . . . .	99
4.5.3.6	Platzierung des Prefabs . . . . .	101
4.5.3.7	Visualisierung des Platzierungsablaufs . . . . .	102
4.5.4	Inventarverwaltung . . . . .	102
4.5.4.1	Der Inventory Controller . . . . .	103
4.5.4.2	Begrenzungen eines Objekts . . . . .	104
4.5.4.3	Problem bei Standardbegrenzungen . . . . .	104
4.5.4.4	Erweitern der Standardbegrenzungen . . . . .	105
4.5.4.5	Erkennen hinzugefügter oder entfernter Gegenstände . . . . .	107
4.5.4.6	Zustände des Inventars aus der Sicht Benutzers . . . . .	109
4.5.5	EventManager . . . . .	111
4.5.6	Das Knapsack Problem . . . . .	113
4.5.6.1	Problemstellung . . . . .	113
4.5.6.2	Typen des Knapsack-Problems . . . . .	113
4.5.6.3	Variationen des Knapsack-Problems . . . . .	115
4.5.6.4	Ansätze zur Lösung des Knapsack-Problems . . . . .	116
4.5.6.5	Dynamischer Programmieransatz . . . . .	116
4.5.6.6	Greedy-Ansatz . . . . .	119
4.5.6.7	Anwendungen des Knapsack-Problems . . . . .	120
4.5.6.8	Auswahl des Implementierungsansatzes . . . . .	120
4.5.6.9	Auswahl der Variante und Typ des Problems . . . . .	121
4.5.6.10	Der Knapsack Solver . . . . .	121

4.5.6.11	Knapsack-Algorithmus Implementierung . . . . .	122
4.5.6.12	Berechnung des zusammengestellten Rucksacks . . . . .	125
4.5.7	Anzeigen der perfekten Lösung . . . . .	125
4.5.7.1	Aufbau und Hirarchie des best Solution Prefabs . . . . .	125
4.5.7.2	Das Best Solution Prefab . . . . .	126
4.5.7.3	Auslöser des Skripts . . . . .	127
4.5.7.4	Start des Perfect Solution Visualizer . . . . .	128
4.5.7.5	Prefab der perfekten Lösung platzieren . . . . .	129
4.5.7.6	Prefab füllen . . . . .	129
4.5.7.7	Zustand nach Knopfdruck . . . . .	130
4.6	Performance und Qualitätssicherung . . . . .	130
4.6.1	<i>Unity Test Framework</i> . . . . .	130
4.6.1.1	Test-Runner . . . . .	130
4.6.2	Testfälle . . . . .	131
4.6.2.1	Aufbau eines einzelnen Testfalls . . . . .	131
4.6.2.2	Testen von Einzelgegenständen im Rucksack . . . . .	132
4.6.2.3	Testen von mehreren Gegenständen im Rucksack . . . . .	133
4.6.2.4	Performance-Messung . . . . .	134
<b>5</b>	<b>Zusammenfassung und Abschluss</b>	<b>137</b>
5.1	Ergebnis . . . . .	137
5.2	Abnahme . . . . .	137
5.3	Zukunft . . . . .	137
<b>A</b>	<b>Mockups</b>	<b>138</b>
A.1	UI/UX . . . . .	138
A.2	Hauptmenu . . . . .	139
A.3	Nachrichtenaustausch Anwendungsszenario . . . . .	140
A.4	Knapsack-Problem Anwendungsszenario . . . . .	142
<b>B</b>	<b>Literatur</b>	<b>143</b>

# Vorwort

Die vorliegende Diplomarbeit wurde im Zuge der Reife- und Diplomprüfung im Schuljahr 2023/24 an der Höheren Technischen Bundeslehr- und Versuchsanstalt Wiener Neustadt verfasst. Die Grundlegende Idee zu dem arbeiten mit der Microsoft HoloLens2, gefördert durch das Förderungsprogramm des Land Niederösterreich "Wissenschaft trifft Schule", lieferte uns unser Betreuer Mag. BEd. Markus Reis. Das Ergebniss dieser Diplomarbeit ist eine Augmented Reality Applikation für die Verwendung innerhalb des Unterrichts und am Tag der offenen Tür.

Besonderer Dank gebührt unserem Betreuer Mag. Markus Reis für sein unerschöpfliches Engagement und seine "kompetente" Unterstützung. Weiteres möchten wir uns bei unserem Abteilungsvorstand Mag. Nadja Trauner sowie unserem Jahrgangsvorstand MSc. Wolfgang Schermann bedanken, die uns die gesamte Zeit an dieser Schule unterstützt haben.

## Diplomarbeit Dokumentation

Namen der Verfasser/innen	Skrepek Moritz Haylaz Seref Lampel Dustin Schoditsch Jonas
Jahrgang Schuljahr	5CHIF 2023 / 24
Thema der Diplomarbeit	Applied Augmented Reality in Education
Kooperationspartner	Land Niederösterreich, Abteilung Wissenschaft und Forschung

Aufgabenstellung	Erklärung und Visualisierung von zwei ausgewählten IT-Grundprinzipien mittels der Microsoft HoloLens2.
------------------	--

Realisierung	Implementiert wurde eine in Unity entwickelte Augmented Reality Applikation für die Mircosoft HoloLens2. Um ein gutes Zusammenspiel zwischen Realität und Augmented Reality zu garantieren wurde Raumerkennung verwendet. Um mit den echten Objekten zu interagieren werden QR-Codes verwendet.
--------------	---

Ergebnisse	Planung, Design, Entwicklung und Test einer funktionsfähigen AugmentedReality-Applikation auf Basis des AR-Devices HoloLens2 von Microsoft, die es ermöglicht ausgewählte technische Themenstellungen im Bereich Informatik (Visualisierung des Nachrichtenaustausches zwischen PCs, Veranschaulichung Knapsack-Problem) für den Einsatz im Unterricht sowie beim Tag der offenen Tür visuell, interaktiv und spielerisch darzustellen.
------------	---

Typische Grafik, Foto etc. (mit Erläuterung)	<p>Das vorliegende Bild stellt das Logo der AR-Applikation dar.</p>  <p><b>Applied Augmented Reality in Education</b> <b>Wissenschaft trifft Schule</b></p>
--	--

Teilnahme an Wettbewerben, Auszeichnungen	
---	--

Möglichkeiten der Einsichtnahme in die Arbeit	HTBLuVA Wiener Neustadt Dr.-Eckener-Gasse 2 A 2700 Wiener Neustadt
---	--

Approbation  (Datum, Unterschrift)	Prüfer  Mag. Markus Reis	Abteilungsvorstand  AV Mag. Nadja Trauner
--	--------------------------------	---

## Diploma Thesis Documentation

Authors	Skrepek Moritz Haylaz Seref Lampel Dustin Schoditsch Jonas
Form	5CHIF
Academic Year	2023 / 24
Topic	Applied Augmented Reality in Education
Co-operation partners	Land Niederösterreich, Abteilung Wissenschaft und Forschung

Assignment of tasks	Representation of two selected basic IT principles using the Microsoft HoloLens2.
---------------------	---

Realization	An augmented reality application for the Microsoft HoloLens2 was implemented. In order to guarantee a good interaction between reality and augmented reality, spatial recognition was used. QR codes are used to interact with the real objects.
-------------	--

Results	Planning, design, development and testing of an augmented reality application based on the AR device HoloLens2 from Microsoft, which enables selected technical topics in the field of computer science to be illustrated (visualization of a ping, illustration of the knapsack problem) for use in lessons and on the open doors day of visual, interactive and playful way.
---------	--

Illustrative graph, photo  
(incl. explanation)

This image represents the logo of the AR application.



## Applied Augmented Reality in Education Wissenschaft trifft Schule

Participation in  
competitions,  
Awards

Accessibility of diploma  
thesis  
HTBLuVA Wiener Neustadt  
Dr.-Eckener-Gasse 2  
A 2700 Wiener Neustadt

Approval  
  
(Date, Sign)

Examiner  
  
Mag. Markus Reis

Head of Department  
  
AV Mag. Nadja Trauner

# Kurzfassung

Diese Diplomarbeit widmet sich der Entwicklung einer Lernapplikation für die HTL Wiener Neustadt unter Verwendung der Unity-Plattform. Die Umsetzung erfolgte in Form einer Augmented Reality (AR) Applikation, speziell für die Microsoft HoloLens 2.

Die Applikation besteht aus drei verschiedenen Szenarien. Darunter, dass Hauptmenu, das Nachrichtenaustausch und Knapsack-Problem Anwendungsszenario,

Die Applikation ermöglicht es den Schülern, während des Tages der offenen Tür zwei wesentliche Grundprinzipien der Informatik mithilfe von Augmented Reality auf spielerische und interessante Weise zu erkunden. Dies bietet den Schülern die Möglichkeit zu erfahren, ob sie ein Interesse an solchen Themen haben. Auch wird darauf abgezielt diese Applikation im Regelunterricht zu verwenden um so diese Prinzipien / Abläufe selbst anwenden und ausprobieren zu können.

# Abstract

This thesis focuses on developing a learning application for HTL Wiener Neustadt using the Unity platform. The application is an Augmented Reality (AR) application designed for the Microsoft HoloLens 2.

The application comprises three distinct scenarios, namely the main menu, the message exchange and knapsack-problem scenario.

The application enables students to explore two fundamental principles of computer science in a playful and engaging manner using augmented reality during the open house event. This provides students with an opportunity to gauge their interest in such topics. Additionally, the aim is to integrate this application into regular classroom instruction, allowing students to apply and experiment with these principles and procedures themselves.

# Kapitel 1

## Einleitung

### 1.1 Ausgangslage

Um dem IT-Fachkräftemangel entgegenzuwirken, ist es entscheidend, die Ausbildung im MINT-Bereich attraktiver zu gestalten. Diese Diplomarbeit zielt darauf ab, einen wichtigen Beitrag zu diesem Ziel zu leisten, unterstützt durch das Förderprogramm *Wissenschaft trifft Schule* des Landes Niederösterreich. Dabei sollen exemplarische Anwendungen im Bereich Augmented Reality für die Vermittlung von Informatik-Lehrinhalten evaluiert und umgesetzt werden.

### 1.2 Auslöser

Die Besucher des *Tag der offenen Tür* erhalten durch diese Applikation eine Vorführung der neuesten Technologien, was dazu beiträgt, dass sie erkennen, dass die Schule einen sehr hohen Technologiestandard aufweist. Dies soll sicherstellen, dass sich auch in Zukunft viele interessierte Schüler für eine Ausbildung an der Abteilung für Informatik entscheiden. Darüber hinaus stärkt dies den Ruf der Schule nach außen und präsentiert sie als attraktiven Ausbildungsstandort für zukünftige Mitarbeiter vieler Unternehmen.

### 1.3 Aufgabenstellung

Die Aufgabenstellung besteht darin, zwei Anwendungszenarien zu erstellen, die jeweils ein Grundprinzip der Informatik veranschaulichen. Im ersten Szenario wird das Grundprinzip des Nachrichtenaustauschs zwischen zwei PCs dargestellt. Das Ziel hierbei ist es dem Benutzer das eigentlich unsichtbare Nachrichtenpaket zu visualisieren. Der Benutzer soll hier auf einer Website eine Nachricht eingeben können, um diese dann an den anderen Computer zu senden und zusätzlich kann auch der Inhalt dieses Pakets durch draufdrücken visualisiert werden.

Im zweiten Szenario wird das in der IT bekannte und vielseitige verwendete Optimierungsproblem des Knapsack-Problems dargestellt. Mit Hilfe von Augmented Reality wird der Rucksack als Inventar angezeigt in welchem der Benutzer alltägliche Gegenstände platzieren kann. Die Aufgabe des Benutzers besteht darin, den Rucksack bestmöglich mit den verfügbaren Gegenständen zu füllen, um den Gesamtwert den Rucksacks zu maximieren. Zusätzlich zur Lösung des Problems kann der Benutzer durch einen Knopfdruck eine der berechneten perfekten Lösungen anzeigen.

## 1.4 Team

Das Diplomarbeitsteam besteht aus:

- Moritz SKREPEK
- Seref HAYLAZ
- Dustin LAMPEL
- Jonas SCHODITSCH

### 1.4.1 Aufteilung

Die Rolle des Projektleiters der Diplomarbeit nahm Moritz SKREPEK ein, da dieser die grundlegende Idee für die Darstellung zweier Anwendungsszenarios mittels der Microsoft HoloLens2 hatte.

Die entwickelte Applikation lässt sich in das Hauptmenü, das Nachrichtenaustausch-Szenario und das Knapsack-Problem-Szenario gliedern. Die Implementierung sowie die gesamte UI/UX und das Frontend der Website übernahm Dustin LAMPEL unter Verwendung des UX-Tools-Plugins, welches sämtliche UI/UX Elemente für Mixed Reality Anwendungen bereitstellt und HTML, CSS und JavaScript. Die Umsetzung des Nachrichtenaustausch-Szenarios übernahm SCHODITSCH Jonas mittels Objekt-Tracking und Foto-Aufnahmen. Für die Implementierung des Knapsack-Problem-Szenarios waren Moritz SKREPEK und Seref HAYLAZ mittels Verwendung von Plane-detection, QR-Code Tagging und Tracking, Knapsack Algorithmus und Unitests zuständig.

# Kapitel 2

## Grundlagen

In diesem Kapitel wird das Vorgehensmodell erläutert sowie alle Tools, die für die erfolgreiche Abwicklung des Projekts benötigt werden. Außerdem wird die Konzeption der Fragebögen behandelt.

### 2.1 Vorgehensmodelle

Vor der Implementierung des Projekts wurden verschiedene methodische Ansätze für das Projektmanagement untersucht. Das Projektteam entschied sich schnell für ein agiles Vorgehensmodell, um die Dynamik der Planung und Durchführung des Projekts zu optimieren. Zu Beginn des Projekts wurde Scrum als bevorzugtes Modell ausgewählt. Im folgenden Abschnitt wird eine detaillierte Erläuterung des Vorgehensmodells präsentiert, gefolgt von einer Begründung für unsere Wahl.

→ SKREPEK

#### 2.1.1 Scrum

Scrum ist ein agiles Projektmanagement-Framework, dass sich auf die effiziente Entwicklung von Produkten und Software konzentriert. Es legt besonderen Wert auf Zusammenarbeit, Anpassungsfähigkeit und die kontinuierliche Bereitstellung funktionsfähiger Produktinkremeente innerhalb kurzer Entwicklungszyklen, die als Sprints bezeichnet werden.<sup>1</sup>

Die zuvor skizzierte Definition bietet einen prägnanten Einblick in das agile Vorgehensmodell Scrum. Die herausragenden Merkmale dieses Modells umfassen:

- Definition von drei zentralen Rollen, die im Folgenden näher erläutert werden.
- Verwaltung des Product Backlogs, der sämtliche Anforderungen enthält.
- Iterative und zeitlich definierte Entwicklung von Produkten.
- Förderung der autonomen Arbeitsweise des Teams.
- Gewährleistung der Gleichberechtigung aller Teammitglieder.

##### 2.1.1.1 Die drei Rollen in Scrum

###### Product Owner

Der Product Owner ist verantwortlich für die Pflege des Product Backlogs und vertritt dabei die fachliche Auftraggeberseite. Eine zentrale Aufgabe ist die Priorisierung der Elemente im Product Backlog, um den geschäftlichen Wert des Produkts zu maximieren und

---

<sup>1</sup>Scrum.org **What is Scrum**

die Möglichkeit für frühe Veröffentlichungen essentieller Funktionalitäten zu schaffen. Der Product Owner nimmt nach Möglichkeit an den täglichen Scrum-Meetings teil, um Einblicke zu gewinnen. Er steht dem Team für Rückfragen zur Verfügung, um einen reibungslosen Informationsaustausch zu gewährleisten.<sup>2</sup>

### Scrum Master

Der Scrum Master spielt eine zentrale Rolle im Scrum-Prozess und ist für dessen korrekte Umsetzung verantwortlich. Er fungiert als Vermittler und Unterstützer, um einen optimalen Arbeitsfortschritt zu erzielen und kontinuierliche Optimierung sicherzustellen. Ein zentrales Anliegen ist die Beseitigung von Hindernissen, um ein reibungsloses Voranschreiten des Teams zu gewährleisten. Der Scrum Master sorgt für einen effizienten Informationsfluss zwischen dem Product Owner und dem Team, moderiert Scrum-Meetings und behält die Aktualität der Scrum-Artefakte wie Product Backlog, Sprint Backlog und Burndown Charts im Blick. Darüber hinaus obliegt es seiner Verantwortung, das Team vor unbefugten Eingriffen während des Sprints zu schützen.<sup>3</sup>

### Team

Das Team besteht idealerweise aus sieben Mitgliedern und setzt sich interdisziplinär aus Entwicklern, Architekten, Testern und technischen Redakteuren zusammen. Es agiert selbstorganisiert und übernimmt die Verantwortung als eigener Leiter. Es hat die Befugnis, autonom über die Aufteilung von Anforderungen in Aufgaben zu entscheiden und diese auf die einzelnen Mitglieder zu verteilen. Dadurch entsteht der Sprint Backlog aus dem aktuellen Teil des Product Backlogs.<sup>4</sup>

#### 2.1.1.2 Scrum Meetings

Alle Anforderungen an das Produkt werden in sogenannten *User Stories* gesammelt, die vorrangig vom Product Owner im Product Backlog erstellt werden. Während eines Sprints werden die User Stories abgearbeitet. Die Projektentwicklung nach Scrum besteht aus fünf zentralen Elementen:

#### Sprint Planning Meeting

Im Sprint Planning Meeting wird das Ziel des folgenden Sprints definiert. Dabei werden die Anforderungen im Product Backlog, die in diesem Sprint umgesetzt werden sollen, in einzelne Aufgaben zerlegt und anschließend im Sprint Backlog gesammelt.<sup>5</sup>

#### Sprint

Ein Sprint ist eine Entwicklungsphase, in der eine voll funktionsfähige und potenziell veröffentlichte Software entsteht. Die Dauer eines Sprints beträgt typischerweise zwischen 1 und 4 Wochen und ist für alle Sprints gleich lang.<sup>6</sup>

<sup>2</sup>Scrum.org **What is a Product Owner**

<sup>3</sup>Scrum.org **What is a Scrum Master**

<sup>4</sup>Scrum.org **What is a developer**

<sup>5</sup>Scrum.org **What is Sprint Planning**

<sup>6</sup>Scrum.org **What is a Sprint**

## Daily Scrum

Der Daily Scrum ist ein kurzes tägliches Teammeeting im Scrum-Framework. Es dient dazu, den Fortschritt des Teams zu synchronisieren und potenzielle Hindernisse frühzeitig zu identifizieren. Während dieses Meetings informieren die Teammitglieder die anderen über den Abschluss ihrer Aufgaben seit dem letzten Treffen, diskutieren, woran sie bis zum nächsten Treffen arbeiten werden, und geben Einblicke in eventuelle aktuelle Probleme oder Herausforderungen. Dieses regelmäßige Treffen stellt sicher, dass alle Teammitglieder stets auf dem aktuellen Stand sind. Es fördert eine effektive Zusammenarbeit und ermöglicht eine zeitnahe Lösung aufkommender Probleme.<sup>7</sup>

## Sprint Review

Während des Meetings präsentiert das Entwicklungsteam den Stakeholdern die im Sprint abgeschlossenen Arbeitsergebnisse, wie zum Beispiel fertige Produktinkremente. Zu den Stakeholdern gehören typischerweise Produktbesitzer, Kunden, Führungskräfte und andere relevante Interessengruppen.<sup>8</sup>

## Sprint Retrospektive

Die Sprint Retrospektive ermöglicht es dem Scrum-Team, bestehend aus dem Entwicklungsteam, dem Scrum Master und dem Product Owner, gemeinsam den abgeschlossenen Sprint zu reflektieren und Möglichkeiten zur kontinuierlichen Verbesserung der Projekteinheit zu identifizieren.<sup>9</sup>

Durch die Elemente des agilen Ansatzes, insbesondere die iterative Natur von Scrum, kann ein optimaler Projektablauf gewährleistet werden. Die kontinuierliche Zusammenarbeit mit dem Kunden ermöglicht es, Anforderungen und Erwartungen während des gesamten Entwicklungsprozesses anzupassen und zu verfeinern. Dies führt zu höherer Kundenzufriedenheit und einem Produkt, das besser den tatsächlichen Bedürfnissen entspricht.

Darüber hinaus bietet regelmäßige Kommunikation und Transparenz innerhalb des Teams und mit den Stakeholdern die Möglichkeit, Missverständnisse und Probleme frühzeitig zu erkennen und anzugehen. So kann das Team schnell auf Veränderungen reagieren und den Kurs des Projekts entsprechend anpassen. Dadurch können potenzielle Risiken minimiert und die Effizienz sowie die Qualität der Arbeit verbessert werden.

### 2.1.2 Begründung der Auswahl

Die Applikation *Applied Augmented Reality in Education* besteht aus drei verschiedenen Szenarien. Das Team, bestehend aus vier Schülern, übernahm jeweils einen Teilbereich oder arbeitete in Subteams an einem dieser Szenarien. Dabei erhielten sie Unterstützung vom betreuenden Lehrer, der stets für Fragen zur Verfügung stand und häufig beratend tätig war.

Als Vorgehensmodell wählte das Team das agile Modell Scrum. Die Scrum-Richtlinien konnten leicht eingehalten werden, da sich das Team täglich in der Schule traf und auch außerhalb der Schule privat in Kontakt stand. Diese regelmäßige Kommunikation ermöglichte es dem Team, Änderungen, Probleme und andere Angelegenheiten leicht zu kommunizieren und zu besprechen.

---

<sup>7</sup>Scrum.org What is a Daily Scrum

<sup>8</sup>Scrum.org What is a Sprint Review

<sup>9</sup>Scrum.org What is a Sprint Retrospective

Am Ende jedes Sprints wurden die erreichten Ergebnisse mit dem Betreuer besprochen und Neuerungen vorgestellt. Im Rahmen der Sprintreviews wurden Feedback zu den Ergebnissen gesammelt und neue Ansichten sowie Denkweisen vom Betreuer eingebracht und integriert.

Die Sprint Retrospektive ermöglichte den Schülern, einen größeren Mehrwert aus der Projektentwicklung zu ziehen, da sie neben der Anwendung des Scrum-Prozesses auch ihre Fähigkeiten in den einzelnen Bereichen verbessern konnten. Hierbei diskutierten und reflektierten sie positive und negative Aspekte.

## 2.2 Projektmanagement-Tools

Um einen positiven Verlauf des Projekts zu ermöglichen, benötigt man unterstützende Tools für das Projektmanagement sowie die Verwaltung von Sourcecode.

### 2.2.1 GitHub

Als Repository für die Source Code Dateien wurde Git in Verbindung mit seiner Webanwendung verwendet. Bei Projektbeginn galt es, die Entscheidung zu treffen, welche Technologie und welcher Anbieter für das Versionskontrollsystem am besten geeignet waren. Andere namhafte Anbieter solcher Plattformen für Verwaltungssysteme sind:

- GitLab
- SourceForge
- BitBucket

Die Wahl von GitHub war durch mehrere Aspekte begründet. Zum einen stellt GitHub eine kostenfreie Lösung dar, die es ermöglicht, ein privates Projekt mit mehreren Mitgliedern ohne Kosten anzulegen. Im Gegensatz dazu bieten einige Plattformen lediglich eine begrenzte Anzahl von Mitgliedschaften in kostenfreien Projekten an. Die Registrierung erforderte lediglich einen Account.

Darüber hinaus zeichnet sich GitHub durch eine benutzerfreundliche Oberfläche, eine breite Unterstützung für verschiedene Programmiersprachen und eine aktive Entwicklergemeinschaft aus. Dies erleichtert die Zusammenarbeit und den Informationsaustausch im Projektteam.

### 2.2.2 Jira

Jira wurde als Verwaltungstool für die Vorgänge im Projekt in Verbindung mit seiner Webanwendung verwendet. Auch hier stand zu Projektbeginn die Frage im Raum, welche Technologie und welcher Anbieter für das Aufgabenmanagement am besten geeignet waren. Neben Jira existieren noch weitere namhafte Anbieter solcher Tools wie beispielsweise:

- VivifyScrum<sup>10</sup>
- Trello<sup>11</sup>

Die Entscheidung für Jira basierte auf mehreren Überlegungen. Zum einen bietet Jira eine kostenfreie Lösung, die es ermöglicht, ein SCRUM Board mit mehreren Mitgliedern kostenfrei anzulegen. Ein weiterer entscheidender Faktor war die direkte Verbindung zu dem GitHub-Repository und die Möglichkeit, neue Branches und Commits direkt in Jira zu erstellen.

---

<sup>10</sup>VivifyScrum-Website <https://app.vivifyscrum.com>

<sup>11</sup>Trello-Website <https://Trello.com>

Darüber hinaus bietet Jira eine umfassende Funktionalität für das Projektmanagement, einschließlich der Verfolgung von Aufgaben, der Planung von Sprints und der Erstellung von Berichten. Diese Features ermöglichen es dem Projektteam, den Fortschritt genau zu überwachen und eventuelle Herausforderungen frühzeitig zu identifizieren und anzugehen.

## 2.3 Fragebögen

→ SKREPEK

Um die erstellte Applikation anhand von User-Feedback zu verbessern wurde in dieser Diplomarbeit ein Fragebogen erstellt und eine Umfrage durchgeführt.

### 2.3.1 Konzeption von Fragebögen

Bei jeder Umfrage werden Informationen von Personen oder Personengruppen zu der allgemeinen Umsetzung und dem Verständnis der Applikation gesammelt. Diese werden im Anschluss ausgewertet und interpretiert. Wichtig ist hier den Zweck jeder Umfrage genau zu definieren. Durch präzise und detaillierte Zielsetzungen ist es später dann möglich, den Erfolg der Umfrage zu garantieren.

### 2.3.2 Planung der Fragebogenkonstruktion

Die sorgfältige Konzeption und Gestaltung eines Fragebogens sind grundlegende Schritte, die bei der Planung einer Erhebung unternommen werden. Durch eine sorgfältige Planung wird sichergestellt, dass relevante Daten erhoben werden und die spätere Auswertung erleichtert wird. Aus diesem Grund müssen bereits im Vorfeld verschiedene Entscheidungen getroffen und Definitionen festgelegt werden:

#### 1. Inhalt

Die Auswahl der Inhalte ist für die Qualität der erhobenen Daten entscheidend. Es sollte erwogen werden, bestehende Fragebögen wiederzuverwenden und gegebenenfalls an die spezifischen Anforderungen der Erhebung anzupassen. Um Missverständnisse zu vermeiden, sollten die Fragen klar und prägnant formuliert sein. Die Verwendung validierter Fragebögen kann bei der Gewährleistung der Vergleichbarkeit mit anderen Studien hilfreich sein.

#### 2. Umfang

Ein wichtiger Faktor, der je nach Forschungsziel abgewogen werden muss, ist die Länge des Fragebogens. Das Ziel sollte ein ausgewogenes Verhältnis zwischen der Tiefe der Informationen und der Aufrechterhaltung der Teilnahme der Teilnehmer sein. Ein zu umfangreicher Fragebogen kann dazu führen, dass die Befragten ermüden und die Qualität der Antworten beeinträchtigt wird.

#### 3. Ablauf und zeitlicher Rahmen

Die Entscheidung bezüglich des Ablaufs und des Zeitrahmens der Befragung beeinflusst die Art der Datensammlung. Die Wahl zwischen einer postalischen und einer elektronischen Befragung wirkt sich auf die Antwortzeit und die Effizienz der Datenerhebung aus. Bei elektronischen Erhebungen liegen die Ergebnisse oft schneller vor, während bei postalischen Erhebungen längere Antwortzeiten möglich sind.

#### 4. Zielgruppe

Die Definition der Zielgruppe spielt eine wichtige Rolle für die Repräsentativität der Ergebnisse. Die Entscheidung für eine Vollerhebung oder eine Stichprobe ist eine Frage der zur Verfügung stehenden Ressourcen und der spezifischen Forschungsziele.

Eine Vollerhebung kann eine umfangreiche Datenbasis liefern, während eine Stichprobenerhebung insbesondere bei großen Zielgruppen effizienter sein kann.

### 5. Fragetypen und Antwortskalen

Die Qualität der erhobenen Daten wird durch die Wahl der Art der Fragen und die Wahl der Antwortskalen beeinflusst. Geschlossene Fragen mit vorgegebenen Antwortmöglichkeiten erleichtern die quantitative Analyse, während offene Fragen die Möglichkeit bieten, qualitative Einsichten zu gewinnen. Die Wahl der Antwortskalen, unabhängig davon, ob es sich um Likert-Skalen oder numerische Bewertungen handelt, sollte auf die spezifischen Forschungsziele abgestimmt sein.

### 6. Ethik und Datenschutz

Es ist von entscheidender Bedeutung, ethische Aspekte zu berücksichtigen, wie die Anonymität der Teilnehmer zu wahren und sensible Informationen zu schützen. Der Fragebogen sollte so konzipiert sein, dass die Integrität der Teilnehmer geschützt wird und keine unangebrachten persönlichen Informationen gesammelt werden.

### 7. Pilotstudie

Es empfiehlt sich, vor der endgültigen Implementierung des Bogens eine Pilotstudie durchzuführen. Im Rahmen dieser Testphase können mögliche Probleme, Unklarheiten oder Missverständnisse bei der Formulierung der Fragen erkannt und behoben werden. Das Feedback der Testteilnehmer trägt dazu bei, den Fragebogen noch weiter zu optimieren.

Grundsätzlich werden die *quantitative* und die *qualitative Forschung* unterschieden, wobei Fragebögen zur *quantitativen Forschung* aufgrund der besseren Vergleichbarkeit leichter zu interpretieren sind.

Bei der Quantifizierung schließt man von der Stichprobe auf die Grundgesamtheit N, wie in Abbildung 2.1 gezeigt. Es ist wichtig, dass die ausgewählte Stichprobe repräsentativ ist. Das bedeutet, dass die Personen, die an der Stichprobe teilnehmen, die gleichen Voraussetzungen haben wie die Personen, die tatsächlich in der Grundgesamtheit vorkommen. Es handelt sich hierbei um numerische Daten, die erhoben werden und für die eine Auswertung vorgenommen wird. Von Bedeutung ist in diesem Zusammenhang das Verhältnis zwischen der untersuchten Stichprobe und der Grundgesamtheit (Population). In diesem Zusammenhang ist das Verhältnis zwischen der untersuchten Auswahl und der Population von Bedeutung. Alle Merkmale der Personen in der Auswahl müssen mit den Merkmalen der Personen in der Population übereinstimmen, zum Beispiel Alter und Geschlecht.<sup>12</sup>

---

<sup>12</sup>Vgl. Mayer, **Interview und schriftliche Befragung**, S. 57 ff.



Abbildung 2.1: Zusammenhang zwischen der *Grundgesamtheit* und *Stichprobe*<sup>13</sup>

Die qualitative Forschung ist eine Methode zur Auswertung von Daten, die ausschließlich über Sprache (verbal) übermittelt werden. Diese Methode eignet sich vor allem zur näheren Beschreibung und Analyse von subjektiven Wahrnehmungen, persönlichen Einstellungen, Motiven und Meinungen der Befragten.<sup>14</sup>

Am sinnvollsten ist eine Kombination von *qualitativen und quantitativen Ergebnissen*. Nach einer quantitativen Befragung sollte eine stichprobenartige qualitative Befragung durchgeführt werden, um die Ergebnisse besser interpretieren zu können.

Unabhängig davon, ob es sich um qualitative oder quantitative Forschung handelt, sind die drei Gütekriterien *Objektivität, Zuverlässigkeit und Validität* zu erfüllen.

Sie dienen, den Forschungsprozess zu steuern und zu kontrollieren. Die Validität ist ein Maß für die Brauchbarkeit der Methode und bezieht sich auf die tatsächliche Fähigkeit zur Messung des gewünschten Wertes. Das Ergebnis ist umso zuverlässiger, je klarer die Fragen formuliert sind. Entscheidend für die Validität der Analyse ist die Objektivität der Messung. Dabei ist sowohl die Durchführungsobjektivität des Befragers als auch die Auswertungs- und Interpretationsobjektivität des Analytikers zu beachten.<sup>15</sup>

Die drei Gütekriterien stehen in einem wechselseitigen Zusammenhang. Nur, wenn die Objektivität gegeben ist, kann die Reliabilität (Zuverlässigkeit) gewährleistet werden. Ist die Reliabilität gering, kann die Validität nur mit einer gewissen Unsicherheit vorhergesagt werden.<sup>16</sup>

### 2.3.3 Formulierung der Fragen

Um eine erfolgreiche Umfrage effizient durchführen zu können, ist eine sorgfältige Vorbereitung erforderlich. Die Erkenntnis, dass Umfragen nur bestimmte Aspekte eines Themenbereichs abdecken können, ist von entscheidender Bedeutung. Aus diesem Grund ist eine sorgfältige und präzise Definition dieser Aspekte erforderlich. Besonderes Augenmerk ist darauf zu richten, dass die Fragen klar formuliert sind.

Die zentrale Priorität bei der Formulierung der Fragen ist die Verständlichkeit und Eindeutigkeit. Die folgenden Formulierungsrichtlinien sollten unbedingt beachtet werden:

- Verwendung von einfachem Vokabular ohne Verwendung von Fachausdrücken, Fremdwörtern oder Ausdrücken aus anderen Sprachen.

<sup>13</sup>Vgl. Brell, **Statistik von Null auf Hundert**, S. 122

<sup>14</sup>Vgl. Mayer, **Interview und schriftliche Befragung**, S. 36

<sup>15</sup>Vgl. Mayer, **Interview und schriftliche Befragung**, S. 54 ff., S. 88.

<sup>16</sup>Vgl. Bühner, **Einführung in die Test- und Fragebogenkonstruktion**, S. 33 f.

- Die Fragen prägnant formulieren.
- Belastende Begriffe wie *Ehrlichkeit* vermeiden.
- Hypothetische Formulierungen ausschließen.
- Fokussierung auf ein bestimmtes Thema für jede einzelne Frage.
- Vermeidung von Überforderung durch die Bereitstellung einer angemessenen Menge an Informationen pro Frage.
- Doppelte Verneinungen vermeiden.<sup>17</sup>

Die genannten Kriterien sind besonders wichtig bei schriftlichen Befragungen. Um sicherzustellen, dass die Ergebnisse nicht verfälscht werden, darf der Interviewer keine zusätzlichen Fragen stellen oder die bereits gestellten Fragen ändern.

Direkte Fragen sind geeignet, um Fakten und Wünsche zu ermitteln, während formulierte Aussagen oder Feststellungen eher dazu dienen, die Bewertung durch die Befragten in Erfahrung zu bringen. Diese Techniken werden hauptsächlich zur Erfassung von Einstellungen, Wahrnehmungen und Meinungen eingesetzt.

#### 2.3.4 Arten von Fragen

In Abhängigkeit von den Anforderungen der jeweiligen Evaluation können sowohl offene als auch geschlossene Fragen gestellt werden.

Bei offenen Fragen handelt es sich um Fragen, bei denen keine Antwortmöglichkeiten vorgegeben werden. Im Anschluss an die Frage sollte ausreichend Platz für die Beantwortung der Frage gelassen werden. Dieser Fragetyp sollte in den folgenden Fällen verwendet werden:

- Wenn die Anzahl der Antwortmöglichkeiten unbekannt ist.
- Wenn die Formulierung der Antwort des Auskunftspflichtigen für die Auswertung von Bedeutung ist.
- Wenn das Ziel der Erhebung darin besteht, die Unwissenheit und Meinung zu ermitteln.

Im Gegensatz zu offenen Fragen gibt es bei geschlossenen Fragen vordefinierte Antwortmöglichkeiten. Die Teilnehmerinnen und Teilnehmer wählen ihre Antworten aus einer vorgegebenen Liste aus oder entscheiden sich zwischen den vorgegebenen Optionen. Es gibt verschiedene Szenarien, in denen der Einsatz von geschlossenen Fragen sinnvoll ist:

- Wenn die Anzahl der möglichen Antwortalternativen begrenzt und bekannt ist.
- Bei Umfragen, die quantitative Daten für eine statistische Auswertung erfordern.
- Wenn die Standardisierung der Antworten wichtig ist, um eine konsistente Analyse zu ermöglichen.<sup>18</sup>

In Bezug auf die geschlossenen Fragen ist es noch wichtig anzumerken, dass es im Wesentlichen drei Möglichkeiten für die Benennung oder Kennzeichnung gibt:

##### 1. Numerische Benennung

Die numerische Benennung ist ein klassisches Notationssystem mit semantischer Bedeutung. Jede Note oder Zahl ist eindeutig einer sprachlichen Formulierung zugeordnet. Der Abstand zwischen den einzelnen Noten ist dabei gleich groß.

---

<sup>17</sup>Vgl. Mayer, **Interview und schriftliche Befragung**, S. 89.

<sup>18</sup>Vgl. Scholl, **Die Befragung**, S. 157.

## 2. Kennzeichnung durch Formen

Eine Möglichkeit, geschlossene Fragen zu kennzeichnen, besteht darin, bestimmte Formen wie Kreise, Kästchen oder grafische Skalen (Symbole) zu verwenden.

## 3. Sprachliche Benennung

Die Benennung von geschlossenen Fragen erfolgt durch klare sprachliche Ausdrücke oder Texte, welche die verschiedenen Antwortoptionen definieren.<sup>19</sup>

### 2.3.5 Struktur und Gliederung von Fragebögen

Die Struktur und Gliederung eines Fragebogens spielen eine entscheidende Rolle bei der Erhebung von Daten. Ein gut durchdachter Aufbau gewährleistet nicht nur eine klare und präzise Erfassung der benötigten Informationen, sondern erleichtert auch die Analyse der Ergebnisse. Bei der Gestaltung eines Fragebogens sollten mehrere wichtige *Schlüsselemente* berücksichtigt werden.

*Fragearten* sind ein zentraler Aspekt bei der Strukturierung von Fragebögen. Die geschickte Kombination von *geschlossenen* und *offenen* Fragen ermöglicht es, eine umfassende Datenerhebung durchzuführen und einen tieferen Einblick zu gewinnen.

Die *Reihenfolge* der Fragen sollte einer sinnvollen *Sequenz* und *Logik* folgen. Der Fragebogen sollte mit allgemeinen und weniger sensiblen Fragen beginnen, um das Vertrauen der Teilnehmer zu gewinnen. Danach sollten spezifischere und möglicherweise persönlichere Fragen gestellt werden.

*Klarheit* ist entscheidend. Klare Anweisungen, eine gut lesbare Schrift und genügend Leerraum tragen dazu bei, Missverständnisse zu vermeiden. Sie ermutigen die Teilnehmer, präzise Antworten zu geben.

Vor dem endgültigen Einsatz des Fragebogens empfiehlt sich die Erprobung des Fragebogens im Rahmen von *Pilotstudien*. Dadurch können mögliche Probleme in Bezug auf *Verständlichkeit*, *Länge* und *Schwierigkeitsgrad* der Fragen identifiziert werden, bevor der endgültige Fragebogen an die Zielgruppe verteilt wird.

Die Beachtung dieser Grundsätze bei der Strukturierung und Gliederung von Fragebögen trägt dazu bei, zuverlässige und aussagekräftige Daten für die Analyse zu gewinnen.

### 2.3.6 Mögliche Verfälschung des Resultats

Die Zuverlässigkeit von Umfrageergebnissen kann durch verschiedene Arten der Verfälschung beeinträchtigt werden. Zwei häufige Verfälschungsarten sind:

- **Simulation:** Teilnehmer neigen dazu, ihre Antworten absichtlich zu verfälschen, um ein bestimmtes Bild von sich selbst zu vermitteln. Dies kann dazu führen, dass die gegebenen Antworten nicht mit den tatsächlichen Meinungen oder Verhaltensweisen übereinstimmen und somit zu einer Verfälschung der Daten führen.
- **Dissimulation:** Es handelt sich hierbei um die bewusste Verzerrung von Informationen durch Teilnehmer mit dem Ziel, bestimmte Aspekte zu verschleiern oder zu verheimlichen. Diese Verzerrung kann dazu führen, dass die gewonnenen Daten nicht der Realität entsprechen und somit die Verlässlichkeit der Ergebnisse der Erhebung in Frage gestellt wird.<sup>20</sup>

Eine Vielzahl von Faktoren kann Verfälschungen auslösen. Gesellschaftliche Normen üben oft Druck auf den Einzelnen aus, sich selbst in einem positiven Licht darzustellen,

---

<sup>19</sup>Vgl. Scholl, **Die Befragung**, S. 164 ff.

<sup>20</sup>Vgl. Bühner, **Einführung in die Test und Fragebogenkonstruktion**, S. 56.

was zu einem Verhalten führen kann, das eine Simulation darstellt. Auf der anderen Seite kann die Furcht vor sozialen Konsequenzen oder persönlichen Nachteilen dazu führen, dass Individuen Aspekte ihrer selbst zurückhalten oder verbergen.<sup>21</sup>

Ein umfassendes Verständnis der Ursachen von Simulation und Dissimulation ist entscheidend für die Entwicklung wirksamer Strategien, mit denen diese Verfälschungen in der Umfrageforschung minimiert werden können.

### 2.3.7 Auswertung von Fragebögen

Die Hauptintention einer Fragebogenerhebung besteht darin, eine homogene Vergleichbarkeit der individuellen Antworten der Befragten zu gewährleisten. Dies ermöglicht eine fundierte statistische Auswertung. Eine unabdingbare Voraussetzung, um aus den erhobenen Fragebogendaten inhaltlich sinnvolle Aussagen ableiten zu können, ist die Umrechnung der qualitativen Antworten in quantitative Werte. Diese Umrechnung erfolgt insbesondere bei computergestützten Verfahren automatisch.

Offene Fragen erfordern eine inhaltliche Auswertung, die in der Regel in Form einer Häufigkeitsanalyse erfolgt, bei der ähnliche Antworten zu Kategorien zusammengefasst werden. Auf diese Weise ist es möglich, die Anzahl der Befragten zu ermitteln, die eine vergleichbare Aussage getroffen haben.

Die Auswertung geschlossener Fragen ist im Allgemeinen mit geringerem Aufwand verbunden, da die Antwortmöglichkeiten bereits vorgegeben sind und jede einzelne Antwort zu einer dieser vorgegebenen Möglichkeiten passen muss.

Für eine angemessene grafische Darstellung der analysierten Daten sind Kreis-, Balken- und Säulendiagramme besonders geeignet. Während Balken- und Säulendiagramme die absoluten Häufigkeiten der Antworten visualisieren und damit Unterschiede in der Anzahl der Antworten deutlich machen, eignen sich Kreisdiagramme besonders, um relative Häufigkeiten, ausgedrückt in Prozent, darzustellen.

### 2.3.8 Planung und Erstellung eines Fragebogens

Im Verlauf dieser Diplomarbeit wurde ein Fragebogen gemäß den erläuterten Regeln und Grundlagen entwickelt. Der Fragebogen soll am *Tag der offenen Tür* an der HTBLuVA Wiener Neustadt in der Abteilung Informatik am 01.12.2024 durchgeführt werden. In der Planungsphase wurde das allgemeine Layout des Fragebogens festgelegt. Anschließend wurden im Projektteam Fragen zusammengestellt, die zur Weiterentwicklung der Applikation beitragen sollen. Der Fragebogen wurde mithilfe der Markup-Sprache *LaTeX* erstellt.

### 2.3.9 Struktur und verwendete Fragetypen

Eine klare und logische Struktur des Fragebogens sowie die Auswahl geeigneter Fragetypen spielen eine entscheidende Rolle, um es dem Befragten zu erleichtern, die Fragen zu beantworten, und seine Motivation aufrechtzuerhalten. Der Fragebogen beginnt mit einer kurzen Einleitung und Begrüßung, um dem Befragten den Zweck des Fragebogens und den Grund für seine Durchführung zu vermitteln.

Bei der Auswahl der Fragetypen wurde bewusst entschieden, ausschließlich geschlossene Fragen zu verwenden. Diese Entscheidung basierte auf zwei Hauptüberlegungen. Erstens sollte die begrenzte Zeit am Tag der offenen Tür effizient genutzt werden, und es galt, die Besucher nicht zu lange aufzuhalten. Daher wurde es als taktisch sinnvoll erachtet, keine offenen Fragen einzubeziehen. Zweitens erleichtern geschlossene Fragen dem Projektteam die

<sup>21</sup>Vgl. Bühner, **Einführung in die Test und Fragebogenkonstruktion**, S. 59.

Umwandlung quantitativer Befragungsergebnisse in qualitative Werte. Außerdem ermöglichte diese Entscheidung dem Projektteam, spezifische Bereiche einzugrenzen, in denen bereits bekannt war, dass Verbesserungsbedarf besteht.

### 2.3.10 Pilotstudio des Fragebogens

Um Missverständnisse oder Unklarheiten bezüglich der gestellten Fragen zu vermeiden, wurde vor der endgültigen Durchführung der Befragung ein Pilotversuch in einer Abschlussklasse der HTBLuVA Wiener Neustadt, Abteilung Informatik, durchgeführt. Die Ergebnisse dieser Pilotstudie zeigten, dass die gestellten Fragen verständlich waren und ohne größere Probleme beantwortet werden konnten.

### 2.3.11 Auswertung des erstellten Fragebogens

Um die Ergebnisse detailliert zu präsentieren, wird in diesem Abschnitt ein realer Fragebogen verwendet, der während des *Tags der offenen Tür* der HTBLuVA Wiener Neustadt, Abteilung Informatik eingesetzt wurde. Die Fragen werden zunächst aufgeführt, gefolgt von einer Analyse der Ergebnisse. Für diese Analyse wird das Tool *Excel* verwendet, um die *qualitativen* Antworten in *quantitative* Werte umzurechnen.

Diese Vorgehensweise ermöglicht es, die gesammelten Daten konkret zu veranschaulichen und deren Bedeutung für das Entwicklerteam besser verständlich zu machen.

Während des Tages der offenen Tür im Schuljahr 2023/2024 (25 Personen) und in einer Abschlussklasse 5CHIF 2023 / 24 (14 Personen) einer Schule mit Matura wurde eine Umfrage durchgeführt, um die Nutzermeinungen zur aktuellen Anwendung zu erfassen. Das Ziel dieser Umfrage bestand darin zu ermitteln, ob die zugrunde liegenden ausgewählten Konzepte der Informatik verstanden wurden und ob potenzielle Verbesserungsmöglichkeiten vorliegen.

Anschließend wurden sieben Personen (Stichprobe) der Abschlussklasse befragt, um potenzielle Verbesserungsvorschläge zu ermitteln. Das Ziel dieser Umfragen bestand darin, Personen zu befragen, die bereits über grundlegendes IT-Wissen verfügen und dadurch besser beurteilen können, welche Aspekte verbessert werden können.

Der durchgeführte Fragebogen bestand aus fünf geschlossenen Fragen, wobei bei zwei Fragen die Möglichkeit bestand, mehrere Antworten auszuwählen. Die Entscheidung, ausschließlich geschlossene Fragen zu verwenden, wurde getroffen, um das Feedback für das Projektteam leichter auswertbar zu machen und um die Umsetzung von Verbesserungen im Nachhinein durch diese quantitative Befragung zu erleichtern. Die Fragen samt Antwortmöglichkeiten und die zugehörige Auswertung sind wie folgt:

- **1. Frage:** Wie leicht war es für Sie, dass Prinzip des *Nachrichten Austauschs* zwischen zwei PCs in der Applikation zu verstehen?

#### Antwortmöglichkeiten:

Sehr schwer(5), Schwer(4), Mittel(3), Leicht(2), Sehr leicht(1)

#### Auswertung:



Abbildung 2.2: Auswertung Frage 1

- **2. Frage:** Konnten Sie das Knapsack-Problem in der Applikation ohne Probleme nachvollziehen?

**Antwortmöglichkeiten:**

Nein nicht wirklich(3), Ja mit einigen Schwierigkeiten(2), Ja problemlos(1)

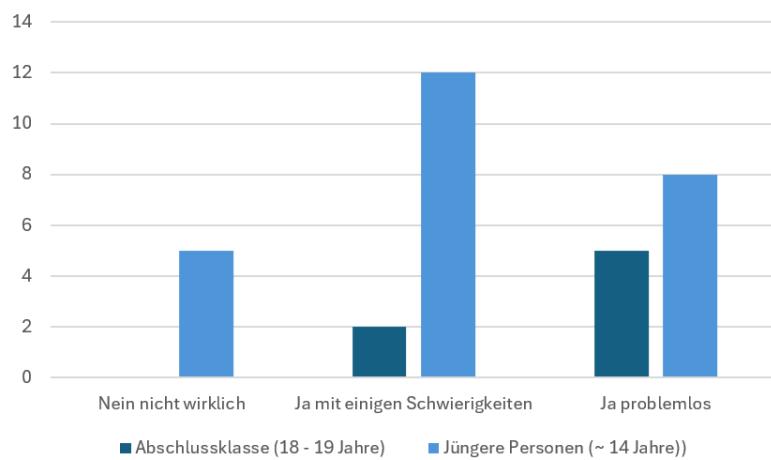
**Auswertung:**

Abbildung 2.3: Ergebnis Frage 2

- **3. Frage:** Wie sehr hat die AR-Technologie Ihnen dabei geholfen die Informatik Prinzipien zu verstehen?

**Antwortmöglichkeiten:**

Sehr negativ(5), Negativ(4), Neutral(3), Positiv(2), Sehr positiv(1)

**Auswertung:**

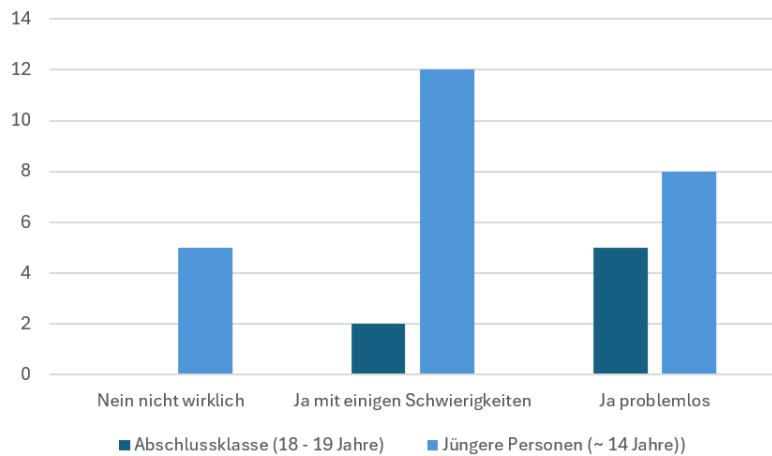


Abbildung 2.4: Ergebnis Frage 3

- **4. Frage:** Welche der folgenden Aspekte haben Ihnen am meisten gefallen? (Mehr als eine Auswahl möglich)

**Antwortmöglichkeiten:**

Lernfördernd, Benutzerfreundlichkeit, Kreativität, Innovativ, Interaktiv

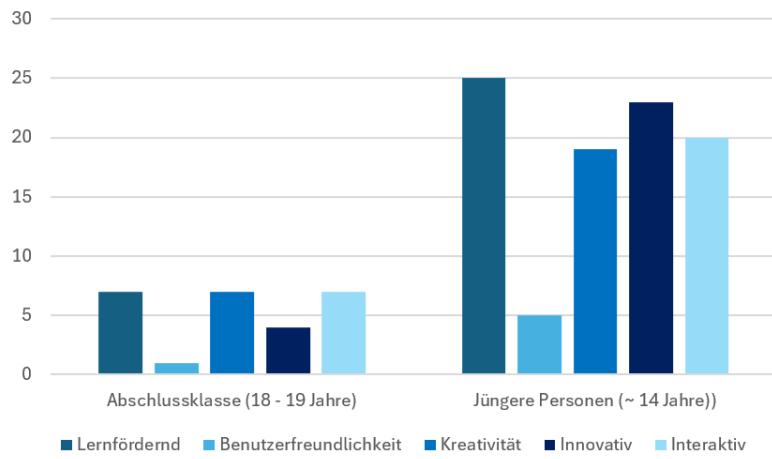
**Auswertung:**

Abbildung 2.5: Ergebnis Frage 4

- **5. Frage:** Welche der folgenden Aspekte sind noch verbesserungswürdig? (Mehr als eine Auswahl möglich)

**Antwortmöglichkeiten:**

Tips / Anweisungen, Leistungsfähigkeit, Inhalt, Stabilität, Übersichtlichkeit

**Auswertung:**

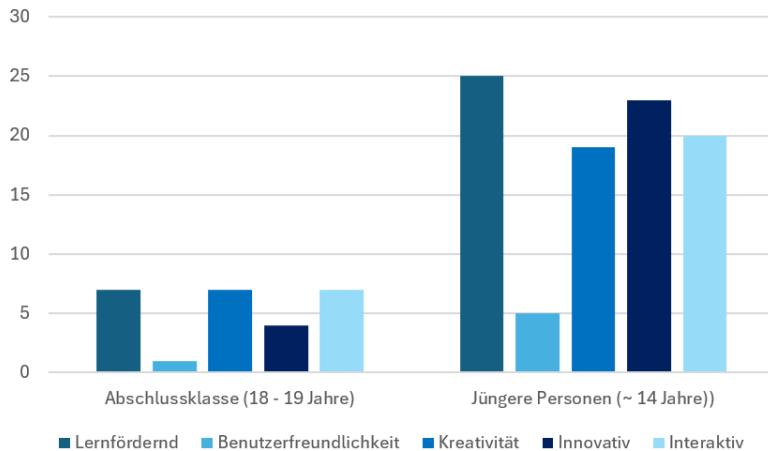


Abbildung 2.6: Ergebnis Frage 5

Anhand der Auswertungen des Fragebogens wird deutlich, dass die Antworten der befragten Personen am Tag der offenen Tür und von den Personen der Abschlussklasse stark voneinander abweichen. Dies wird besonders bei den letzten beiden Fragen, die sich auf Aspekte, die dem Benutzer gefallen, und Verbesserungsmöglichkeiten beziehen, deutlich. Hier ist nämlich klar zu erkennen, dass die Personen der Abschlussklasse im Vergleich zu den am Tag der offenen Tür Befragten Person signifikant mehr Angaben dazu gemacht haben. Dies lässt sich darauf zurückführen, dass sie über deutlich mehr Erfahrung im Bereich der Entwicklung von Applikationen verfügen und generell ein vertieftes Verständnis der Informatik besitzen. Ein erfreuliches Ergebnis ist jedoch, dass ein Großteil sowohl der am Tag der offenen Tür befragten als auch der in der Abschlussklasse befragten Personen die vermittelten Prinzipien der Informatik gut verstanden hat und auch, dass der Einsatz von AR-Technologie dabei geholfen hat diese zu verdeutlichen.

Zu erwähnen ist, dass bei der Auswertung insbesondere bei der vorletzten Frage festgestellt wurde, dass die Applikation als nicht benutzerfreundlich bewertet wurde. Dieses Ergebnis war jedoch erwartet, da zum Zeitpunkt der Befragung das Projekt noch in einem sehr frühen Entwicklungsstadium war und daher einige essenzielle Funktionen und Erweiterungen wie Anweisungen und Tipps, Performance und Inhalt fehlten.

Darüber hinaus hat diese Befragung und Auswertung wichtige Ergebnisse geliefert, die dazu beigetragen haben, viele Aspekte der Applikation maßgeblich zu erweitern und zu verbessern.

# Kapitel 3

## Produktspezifikationen

Dieses Kapitel behandelt die Planung und Spezifikation des Projekts. Weiteres wird die verwendete Technologieauswahl begründet und mit Alternativlösungen verglichen.

### 3.1 Anforderungen und Spezifikationen

Hier kommt noch was über Anforderungen und Spezifikationen

#### 3.1.1 Use Cases

Hier kommt noch was über Use Cases

### 3.2 Design

Hier kommt noch was über das Design

#### 3.2.1 Abläufe

Hier werden die Abläufe der drei Szenarien modelliert und eingefügt

### 3.3 Eingesetzte Technologien

#### 3.3.1 Microsoft HoloLens2

→ SKREPEK

Zentrale Plattform des Projekts ist die Augmented-Reality-Brille "HoloLens 2" von Microsoft. Dabei handelt es sich um eine bahnbrechende Augmented-Reality-Brille, die eine revolutionäre Verbindung zwischen der digitalen und der physischen Welt schafft. Mit einer Reihe hochentwickelter Sensoren, Kameras und einem hochauflösenden Display eröffnet die HoloLens 2 neue Horizonte für interaktive und immersive AR-Erlebnisse.

Die HoloLens 2 bietet eine beeindruckende Immersion, die es dem Nutzer ermöglicht, digitale Objekte nahtlos in seine Umgebung zu integrieren. Dank des fortschrittlichen Hand- und Blickverfolgungssystems können Benutzer mit den holografischen Elementen interagieren, als wären sie Teil ihrer realen Welt. Dies ermöglicht eine breite Palette von Anwendungen, von Unterhaltung über Bildung bis hin zu industriellen Anwendungen.

### 3.3.2 Kriterien

Bei der Auswahl der Technologien, die für die Entwicklung der geplanten Anwendung eingesetzt werden, war es besonders wichtig, dass diese über eine gute Dokumentation und eine große Entwicklergemeinschaft verfügen, falls Fragen auftauchen, für die nicht direkt im Internet eine Lösung gefunden werden kann. Außerdem sollten sie einfach zu bedienen sein und vor allem eine performante Nutzung der Anwendung gewährleisten.

### 3.3.3 Game Engine

Um einen reibungslosen Verlauf des Projekts zu gewährleisten, ist die sorgfältige Auswahl der richtigen *Game Engine* von entscheidender Bedeutung. Die Game Engine fungiert als fundamentale Plattform für die Entwicklung und Erstellung von Videospielen, indem sie eine umfassende Palette von Werkzeugen, Bibliotheken und Funktionen bereitstellt, die Entwicklern hilft, Spiele zu konzipieren, umzusetzen und zu optimieren. In diesem Abschnitt werden zwei potenzielle Game Engines, die für das Projekt in Betracht gezogen wurden, eingehend untersucht und anschließend die Gründe für die getroffene Auswahl erläutert.

#### 3.3.3.1 Unity

Unity ist eine Game Engine, die von Entwicklern weltweit für die Erstellung von 2D- und 3D-Spielen genutzt wird. Sie unterstützt verschiedene Plattformen wie PC, Konsolen, Mobilgeräte und AR/VR-Geräte. Unity bietet Entwicklern eine umfangreiche Sammlung von Werkzeugen und Ressourcen, um Spiele schnell zu prototypen und zu entwickeln.

Ein herausragendes Merkmal von Unity ist der Asset Store. Hier können Entwickler Assets wie 3D-Modelle, Texturen, Sounds und Plugins kaufen oder verkaufen. Dadurch können sie ihre Projekte mit hochwertigen Inhalten erweitern und verbessern, ohne alles von Grund auf neu erstellen zu müssen. Unity bietet außerdem eine starke Community-Unterstützung mit Foren, Tutorials und Schulungen, was besonders für neue Entwickler hilfreich ist.

Die Programmierung in Unity erfolgt hauptsächlich durch die Verwendung von C-Sharp. Diese ist sowohl für erfahrene Entwickler als auch für Anfänger zugänglich. Unity ist aufgrund der Kombination von benutzerfreundlichen Werkzeugen, einer großen Community und einer breiten Plattformunterstützung eine beliebte Wahl für Indie-Entwickler sowie für große Studios.<sup>1</sup>

#### 3.3.3.2 Unreal Engine

Die Unreal Engine ist eine leistungsstarke Game Engine, die für ihre hochwertigen Grafiken und fortgeschrittenen Funktionen bekannt ist. Sie wird häufig für die Entwicklung von AAA-Titeln sowie für hochwertige VR-Erfahrungen verwendet. Die Engine bietet branchenführende Grafikfunktionen wie fortschrittliche Beleuchtung, Partikelsysteme, Physiksimulationen und Echtzeit-Rendering.

Eine bemerkenswerte Funktion der Unreal Engine ist das Blueprints-System (siehe folgender Absatz). Es ermöglicht Entwicklern, Spiele und interaktive Inhalte ohne herkömmlichen Programmiercode zu erstellen. Dadurch wird die Engine besonders zugänglich für Künstler und Designer, die möglicherweise keine tiefen Programmierkenntnisse haben.

Die Unreal Engine bietet eine umfangreiche Sammlung von vorgefertigten Assets und Werkzeugen sowie einen integrierten Marketplace, auf dem Entwickler zusätzliche Inhalte

---

<sup>1</sup>Unity Website, <https://unity.com/>

erwerben können. Sie unterstützt eine Vielzahl von Plattformen, darunter PC, Konsolen, Mobilgeräte und VR-Headsets.

Insgesamt ist die Unreal Engine eine leistungsstarke Wahl für Entwickler, die hochwertige Grafiken und fortschrittliche Funktionen in ihren Spielen und Anwendungen benötigen. Sie wird häufig von größeren Studios genutzt, die Zugang zu hochwertigen Tools und Support benötigen.<sup>2</sup>

### Blueprint Visual Scripting

Das Blueprint Scripting System ist eine leistungsstarke visuelle Programmierumgebung innerhalb der Unreal Engine. Es ermöglicht Entwicklern, komplexe Logik und Interaktionen ohne traditionelle Programmierkenntnisse zu erstellen. Das System basiert auf dem Konzept von *Blueprints*, die visuelle Darstellungen von Logik und Funktionen sind.

Die Verwendung von Blueprints bietet eine Reihe von Vorteilen. Einerseits ermöglicht es Künstlern und Designern ohne tiefgreifende Programmiererfahrung, komplexe Interaktionen und Spielmechaniken zu implementieren. Durch das Drag-and-Drop-Interface können Benutzer Funktionsblöcke miteinander verbinden und so den Fluss der Spiellogik steuern.

Andererseits erleichtert das Blueprint-System die Iteration und Prototypisierung. Da Änderungen visuell vorgenommen werden können, können Entwickler schnell experimentieren und Anpassungen vornehmen, um das gewünschte Verhalten zu erreichen. Dies beschleunigt den Entwicklungsprozess und ermöglicht es Teams, flexibel auf Feedback und sich ändernde Anforderungen zu reagieren.

Das Blueprint Scripting System bietet zudem eine hohe Flexibilität und Erweiterbarkeit. Entwickler können benutzerdefinierte Blueprints erstellen und sie in anderen Projekten wiederverwenden, was die Effizienz erhöht und die Entwicklung beschleunigt.<sup>3</sup>

#### 3.3.3.3 Game Engine Auswahl und Wechsel im Projektverlauf

Bei Projektbeginn wurde nach umfassender Recherche und Evaluierung verschiedener Optionen entschieden, die Unreal Engine als primäre Entwicklungsumgebung für dieses Projekt zu verwenden. Diese Entscheidung wurde aufgrund mehrerer überzeugender Faktoren getroffen, darunter insbesondere das beliebte Blueprint Visual Scripting, das die Entwicklung von interaktiven Inhalten erleichtert und auch für Personen ohne umfangreiche Programmierkenntnisse zugänglich macht. Zusätzlich zu diesem herausragenden Merkmal wurden weitere Aspekte berücksichtigt, die die Unreal Engine zu einer attraktiven Wahl machten, wie beispielsweise ihre leistungsstarken Grafikfunktionen, die branchenweit anerkannt sind, sowie ihre Unterstützung für die Entwicklung hochwertiger VR-Erfahrungen und AAA-Titel.

#### Herausforderungen im ersten Monat des Entwicklungsprozesses

Trotz der zu Beginn getroffenen Entscheidung für die Unreal Engine traten im Verlauf des ersten Monats des Entwicklungsprozesses bestimmte Herausforderungen auf. Diese Herausforderungen können als unerwartete Schwierigkeiten betrachtet werden, die während der Umsetzung des Projekts auftraten und zweifel an der ursprünglichen Entscheidung mit sich brachte. Im Einzelnen wurden folgende Herausforderungen identifiziert:

1. **Mangelhafte Dokumentation für AR-Entwicklung in der Unreal Engine:**  
Die unzureichende Dokumentation für die Entwicklung von Augmented Reality (AR)-

---

<sup>2</sup>Unreal Engine Website, <https://www.unrealengine.com/>

<sup>3</sup>Unreal Engine Dokumentation, **Blueprint Visual Scripting**

Anwendungen in der Unreal Engine erwies sich als erhebliche Hürde. Fehlende detaillierte Anleitungen und Referenzen für AR-spezifische Funktionen behinderten die effiziente Integration von AR-Elementen.

2. **Begrenzte Verfügbarkeit von AR-spezifischen Online-Tutorials:** Ein Mangel an Online-Tutorials, die sich speziell mit der Entwicklung von AR-Anwendungen in der Unreal Engine befassten, führte zu einer beträchtlichen Lernkurve für das Entwicklerteam und verzögerte den Implementierungsprozess von AR-spezifischen Features.
3. **Komplexität der AR-Entwicklung in der Unreal Engine:** Die Unreal Engine erwies sich als sehr anspruchsvoller für Neueinsteiger in Bezug auf die Umsetzung von AR-spezifischen Funktionen. Die Notwendigkeit, komplexe Skripte zu erstellen und vielfältige Einstellungen anzupassen, führte zu einem erhöhten Zeitaufwand für die Umsetzung von AR-Elementen.
4. **Eingeschränkte Community-Unterstützung für AR-Entwicklung:** Im Vergleich zu Unity war die Community-Unterstützung für die AR-Entwicklung in der Unreal Engine begrenzt. Die Verfügbarkeit von Ratschlägen und Lösungen für spezifische AR-Herausforderungen war eingeschränkt, was die Eigenständigkeit bei der Lösung von Problemen beeinträchtigte.

Diese Herausforderungen bildeten die Grundlage für die strategische Entscheidung des Projektteams, von Unreal Engine zu Unity zu wechseln. Der Wechsel ermöglichte eine effizientere und zielführende Entwicklung der AR-Applikation, gestützt durch Unity's umfassende Unterstützung, detaillierte Dokumentation und breite Community-Ressourcen.

### 3.3.4 Unity foundation packages

Um Augmented Reality (AR)-Applikationen in Unity erfolgreich zu entwickeln, sind bestimmte *Foundation Packages* erforderlich. Diese Pakete müssen in Unity importiert werden, um grundlegende Funktionalitäten bereitzustellen, die für die erfolgreiche Umsetzung einer AR-Applikation notwendig sind. Im Folgenden werden die beiden wesentlichen Pakete näher erläutert, die für die Funktionalität einer solchen Applikation unerlässlich sind.

#### 3.3.4.1 MRTK3

Das Mixed Reality Toolkit 3 (MRTK3), welches sowohl für Unity als auch in der Unreal Engine anwendbar ist, ist ein leistungsstarkes Open-Source-Entwicklungstoolkit, das von Microsoft entwickelt und gepflegt wird. Es ist speziell darauf ausgerichtet, die Entwicklung von Mixed-Reality-Anwendungen zu erleichtern, indem es eine umfassende Sammlung von Komponenten, Skripten und Assets bereitstellt. MRTK3 bietet Entwicklern eine Vielzahl von Funktionen und Werkzeugen, die speziell für die Erstellung von Anwendungen für Augmented Reality (AR), Virtual Reality (VR) und Mixed Reality (MR) optimiert sind.

Das Toolkit umfasst eine breite Palette von Funktionen, darunter Interaktionselemente wie Hand- und Gestenerfassung, räumliches Mapping, Physiksimulationen für Objekte in der realen Welt, Benutzerschnittstellen-Design-Werkzeuge und vieles mehr. MRTK3 ist plattformübergreifend und unterstützt verschiedene AR-/VR-Headsets sowie andere Mixed-Reality-Geräte.

Die Bedeutung von MRTK3 für die Entwicklung von AR-Applikationen liegt in seiner Fähigkeit, Entwicklern eine solide Grundlage und eine Vielzahl von vordefinierten Komponenten und Tools zu bieten, die den Entwicklungsprozess beschleunigen und vereinfachen. Durch die Verwendung von MRTK3 können Entwickler komplexe AR-Anwendungen

schneller prototypisieren und implementieren, da sie auf eine umfangreiche Bibliothek von Funktionen zugreifen können, die speziell für AR-Szenarien optimiert sind. Dies erhöht die Effizienz der Entwicklung und ermöglicht es den Entwicklern, sich auf die Gestaltung und Umsetzung innovativer AR-Erfahrungen zu konzentrieren, ohne sich um die Grundlagen der AR-Entwicklung kümmern zu müssen.<sup>4</sup>

### 3.3.4.2 Microsoft OpenXR Plugin

Das Microsoft OpenXR Plugin stellt eine bedeutende Komponente im Kontext der Entwicklung von Augmented Reality (AR)-Applikationen innerhalb der Entwicklungsumgebung dar. Entwickelt und bereitgestellt von Microsoft, ermöglicht dieses Plugin die nahtlose Integration des OpenXR-Standards in Unity. OpenXR, initiiert durch die Khronos Group, fungiert als branchenweiter Standard zur Vereinheitlichung der XR-Anwendungsentwicklung, wobei XR für Extended Reality steht und sowohl Augmented Reality (AR), Virtual Reality (VR) als auch Mixed Reality (MR) umfasst.

Das Microsoft OpenXR Plugin erleichtert Entwicklern die Arbeit innerhalb von Unity, indem es eine reibungslose Integration des OpenXR-Standards ermöglicht. Hierdurch erhalten Entwickler Zugang zu einer breiten Palette von Funktionen und Möglichkeiten, die durch den OpenXR-Standard standardisiert sind, inklusive plattformübergreifender Kompatibilität sowie optimierter Leistung für XR-Anwendungen.

Die Relevanz des Microsoft OpenXR Plugins für die AR-Entwicklung liegt in seiner Fähigkeit, eine konsistente Entwicklungsumgebung für XR-Anwendungen innerhalb von Unity zu schaffen. Durch die Nutzung dieses Plugins können AR-Entwickler die Vorteile des OpenXR-Standards voll ausschöpfen, einschließlich verbesserte Kompatibilität, Leistung und Zukunftssicherheit ihrer Anwendungen.

Zusätzlich erlaubt das Plugin den Entwicklern den Zugriff auf spezifische Funktionen und Optimierungen, die von Microsoft speziell für die AR-Entwicklung entwickelt wurden. Dies beinhaltet beispielsweise Features zur Verbesserung der AR-Umgebungserkennung, Handhabung von Eingaben sowie Optimierung der Grafikleistung für AR-Anwendungen.<sup>5</sup>

### 3.3.4.3 Integrationsprozess der Plugins

Zur Integration der genannten Plugins in den Unity Editor, um diese verwenden zu können, wird das externe Tool namens *Mixed Reality Feature Tool* von Microsoft verwendet. Dieses Tool fungiert als umfassende Sammlung von Plugins und Erweiterungen im Bereich der Augmented und Virtual Reality Entwicklung für Unity. Neben den erwähnten Plugins umfasst es weitere wichtige Erweiterungen, darunter:

- Azure Mixed Reality Services
- Experimental
- Mixed Reality Toolkit
- Plattform Support
- Spatial Audio
- World Locking Tools
- Weitere Funktionen (Other features)

Um die Integration durchzuführen, müssen innerhalb des *Plattform Support* Bereichs des *Mixed Reality Feature Tools* sowohl das *Mixed Reality OpenXR Plugin* als auch das

---

<sup>4</sup>Microsoft Dokumentation **Mixed Reality Toolkit 3**

<sup>5</sup>Khronos Group **OpenXR Plugin**

gesamte MRTK3-Paket ausgewählt werden. Nach der Auswahl dieser Pakete ist es erforderlich, sie zu validieren und anschließend in das Unity-Projekt zu importieren.

Dieser Prozess gewährleistet eine erfolgreiche Integration der benötigten Plugins und Erweiterungen, die für die Entwicklung von Augmented Reality-Anwendungen in Unity von entscheidender Bedeutung sind.

### 3.3.5 Wahl des Modellierungsprogramm

→ LAMPEL

Durch Unity und die Open-Source-lastige Modellierungscommunity gibt es verschiedene Möglichkeiten, um an die 3D-Modelle zu gelangen, die für die Anwendung der Szenarien benötigt werden.

Eine Möglichkeit besteht darin, auf weit verbreitete 3D-Modelle auf sogenannten *Asset Stores* zurückzugreifen, wie zum Beispiel dem von Unity selbst. Dort können sie entweder gekauft oder kostenlos heruntergeladen werden. Die verschiedenen Angebote unterscheiden sich in der Texturqualität und dem Modellierungsaufwand.<sup>6</sup>

Eine weitere Möglichkeit besteht darin, die Modelle selbst in einem Modellierungsprogramm zu entwerfen. Diese Variante ist zwar zeitintensiver und aufwendiger, führt jedoch zu maßgeschneiderten Modellen, die den Wünschen und Anforderungen des Projekts entsprechen. Es gibt verschiedene Programme, die die Anforderungen erfüllen würden. Beispielsweise würden Blender<sup>7</sup>, 3ds Max<sup>8</sup> von Autodesk oder Cinema 4D<sup>9</sup> von Maxon in Frage kommen. Diese Programme unterscheiden sich nur hinsichtlich der Bedienung, aber der eigentliche Zweck ist gleich.

Nach Absprachen innerhalb des Teams wurde beschlossen, das Modellierungsprogramm Blender zu benutzen, da eines der Teammitglieder bereits gewisses Wissen in der Bedienung und Modellierung mit diesem Programm angehäuft hat. Nicht nur das war der ausschlaggebende Punkt, sondern auch die große Benutzergemeinschaft in Blender mit zahlreichen Tutorials und Hilfestellungen für auftretende Herausforderungen.

Obwohl diese Variante einen höheren Zeitaufwand und zusätzliche Komplexität des Projekts mit sich bringt, erhält man am Ende eine Sammlung von Objekten, die vollständig den Anforderungen und Wünschen entspricht. Zudem ist dies die kostengünstigste Methode, da der Kauf von Modellen oder einer kostenpflichtigen Modellierungsplattform vermieden werden konnte, um hohe Kosten zu vermeiden.

#### 3.3.5.1 Wie funktioniert Blender im Allgemeinen?

Die Applikation Blender ist sehr komplex. Daher werden in der folgenden Beschreibung nur die Schlüsselaspekte und die Funktionalität von Blender für unseren speziellen Anwendungsbereich hervorgehoben.

- **Benutzeroberfläche und Interaktion**

Die Benutzeroberfläche von Blender ist hoch anpassbar, obwohl sie komplex gestaltet ist. Sie enthält 3D-Modelle, Ansichten, Fenster und Panels. Benutzer interagieren mit Objekten und Werkzeugen über Maus- und Tastaturbefehle. Die Effizienz der Arbeit und die Modellierungsdauer hängen stark von der Erfahrung des Benutzers ab, insbesondere von der Nutzung von Shortcuts und Hotkeys.<sup>10</sup>

---

<sup>6</sup>Unity Asset Stores

<sup>7</sup>Blender Blender Allgemein

<sup>8</sup>Autodesk 3DS Max

<sup>9</sup>Maxon Cinema 4D

<sup>10</sup>Blender BenutzeroberflÄd'che

- **3D-Modellierung**

Blender ermöglicht die Erstellung von 3D-Modellen durch die Verwendung von Primitiven wie Würfeln, Kugeln, Flächen und Kurven. Diese können bearbeitet und modifiziert werden, um komplexe Formen zu erstellen. Modellierungswerkzeuge wie Extrusion, Verschiebung, Skalierung und Rotation stehen zur Verfügung.<sup>11</sup>

- **Materialien und Texturen**

Um realistische Oberflächen zu erzeugen, können Materialien erstellt und Texturen auf Objekte angewendet werden. Blender ermöglicht die Feinanpassung von Materialeigenschaften wie Diffusreflexion, Glanz, Transparenz und Emission.<sup>12</sup>

- **Gemeinschaft und Ressourcen**

Blender hat eine engagierte Benutzergemeinschaft, die umfassende Dokumentation, Tutorials und Foren bereitstellt. Diese Ressourcen erleichtern die Einarbeitung und Problemlösung.

Blender wird im gesamten Projekt eingesetzt, angefangen beim Entwurf und der Erstellung des Hauptmenüs bis hin zur Gestaltung jedes einzelnen Objekts. Es wird hauptsächlich für die digitale Modellierung der wichtigsten täglichen Gegenstände von Schülern verwendet. Das Ziel besteht darin, am Ende eine umfangreiche Sammlung von Objekten zu haben, um den Benutzern eine vielfältige und zahlreiche Auswahl an verschiedenen Modellen zu bieten.

---

<sup>11</sup>Blender **Toolbar**

<sup>12</sup>Blender **Materials**

# Kapitel 4

## Feinkonzept und Realisierung

### 4.1 Entwicklungsumgebungen

#### 4.1.1 Visual Studio 2022

Visual Studio 2022 ist eine integrierte Entwicklungsumgebung (IDE) von Microsoft, die speziell für die Entwicklung von Softwareanwendungen, Webanwendungen und Desktop-Anwendungen konzipiert ist. Es handelt sich um eine umfangreiche Entwicklungsumgebung, die von Entwicklern weltweit für eine breite Palette von Anwendungsfällen eingesetzt wird.

#### 4.1.2 Unity

Der Unity-Editor, entwickelt von Unity Technologies, fungiert als umfassende integrierte Entwicklungsumgebung (IDE) und zentrale Arbeitsumgebung für die Konzeption und Umsetzung von 2D-, 3D-, Augmented Reality (AR) und Virtual Reality (VR) Anwendungen und Spielen. Als Kernelement der Unity-Plattform spielt der Editor eine entscheidende Rolle in der Entwicklung von Projekten, die auf Unity-Technologien basieren.

Die Funktionalität des Unity-Editors erstreckt sich über verschiedene Aspekte der Softwareentwicklung, angefangen bei der visuellen Gestaltung von Szenen und Spielwelten bis hin zur Implementierung komplexer Logik und Interaktionen. Die folgenden Abschnitte vertiefen die Schlüsselmerkmale und Funktionen des Unity-Editors, die ihn zu einem essenziellen Werkzeug für Entwickler machen.

##### 4.1.2.1 Multidisziplinäre Unterstützung und Integration

Der Unity-Editor zeichnet sich durch seine multidisziplinäre Unterstützung aus, die Entwicklern ermöglicht, kollaborativ an Projekten zu arbeiten. Künstler, Entwickler und Designer können innerhalb derselben Umgebung zusammenarbeiten, wodurch ein nahtloser Austausch von Assets, Szenen und Ressourcen ermöglicht wird. Die Integration von Grafik-, Physik- und Audio-Engines erleichtert die Schaffung immersiver und ansprechender digitaler Umgebungen.

##### 4.1.2.2 Szenengestaltung und Asset-Management

Ein zentrales Merkmal des Unity-Editors ist die intuitive Szenengestaltung, die es Entwicklern ermöglicht, 2D- und 3D-Szenen durch Drag-and-Drop-Operationen zu erstellen und anzupassen. Das Asset-Management ermöglicht eine effiziente Organisation von Ressourcen

wie Modelle, Texturen und Audio-Dateien. Hierbei kommt dem Editor eine Schlüsselrolle in der Strukturierung und Verwaltung umfangreicher Projekte zu.

#### 4.1.2.3 Programmierung und Skripterstellung

Der Unity-Editor integriert leistungsstarke Programmierfunktionen, die Entwicklern erlauben, Skripte in C-Sharp oder JavaScript zu verfassen. Die Implementierung von Logik, Interaktionen und Funktionalitäten erfolgt durch die Integration von Skripten in Game-Objects und Szenen. Die Echtzeitansicht von Codeänderungen unterstützt einen iterativen Entwicklungsprozess.

#### 4.1.2.4 Unterstützung für Augmented Reality (AR) und Virtual Reality (VR)

Der Unity-Editor ist essenziell für die Entwicklung von AR- und VR-Anwendungen. Durch die Integration von AR Foundation und XR Interaction Toolkit bietet der Editor leistungsstarke Werkzeuge zur Erstellung immersiver Erlebnisse. Die Möglichkeit, Szenen in Echtzeit in AR- und VR-Geräten zu überprüfen, unterstützt Entwickler bei der Feinabstimmung und Optimierung ihrer Projekte.

#### 4.1.2.5 Erweiterte Debugging- und Profiling-Werkzeuge

Der Unity-Editor stellt umfassende Debugging- und Profiling-Werkzeuge zur Verfügung, um die Leistung und Funktionalität von Anwendungen zu optimieren. Durch Echtzeit-Inspektion, Fehlerverfolgung und Ressourcenüberwachung unterstützt der Editor Entwickler bei der Identifizierung und Behebung von Problemen, um eine reibungslose Ausführung der Anwendungen sicherzustellen.

#### 4.1.2.6 Aufbau einer Unity-Applikation

Die Struktur einer Unity-Applikation ist entscheidend für eine effektive Entwicklung und Organisation von 3D-Anwendungen und Spielen. Eine typische Unity-Anwendung besteht aus verschiedenen Schlüsselementen, darunter Szenen, GameObjects, Komponenten, Skripte und Assets. Diese werden koordiniert durch die Hauptkomponente der Anwendung, die sogenannte "GameManager" oder "MainScene". In diesem Abschnitt werden die grundlegenden Bausteine einer Unity-Anwendung sowie bewährte Praktiken für die Strukturierung und Verwaltung dieser Elemente beleuchtet.

#### 4.1.2.7 Lebenszyklusmethoden in Unity

Die Entwicklung von Augmented Reality (AR)-Applikationen in Unity erfordert ein tiefgreifendes Verständnis der Lebenszyklusmethoden, die in MonoBehaviour-Klassen implementiert werden können. Diese Methoden regeln den Fluss der Programmlogik und ermöglichen Entwicklern, spezifische Aktionen zu bestimmten Zeitpunkten im Lebenszyklus einer Anwendung auszuführen.

- **Awake():** Die `Awake()`-Methode wird aufgerufen, wenn das Skript erstellt wird. Dies geschieht vor anderen Initialisierungsmethoden wie `Start()`. Sie eignet sich für die Durchführung von Initialisierungen, bei denen auf andere Skriptkomponenten oder Ressourcen zugegriffen werden soll. Der Hauptzweck besteht darin, die Ressourcen für das Skript vorzubereiten.
- **Start():** Die `Start()`-Methode wird vor dem ersten Frame aufgerufen und bietet die Möglichkeit, Initialisierungsaufgaben durchzuführen. Im Gegensatz zu `Awake()`

garantiert `Start()` die vollständige Initialisierung aller GameObjects in der Szene. Entwickler nutzen diese Methode oft für Konfigurationen und Vorbereitungen, die spezifisch für die Startphase der Anwendung sind.

- **Update():** Die `Update()`-Methode ist von entscheidender Bedeutung, da sie in jedem Frame aufgerufen wird. Hier kann kontinuierliche Logik ausgeführt werden, wie etwa die Aktualisierung von Animationen, die Verarbeitung von Benutzereingaben oder die Anpassung von Positionen basierend auf der Zeit. Es ist wichtig zu beachten, dass `Update()` häufig aufgerufen wird und daher effizient implementiert werden sollte.
- **LateUpdate():** Ähnlich wie `Update()`, wird aber nachdem alle `Update()`-Methoden aufgerufen wurden. Dies ist besonders nützlich, wenn Anpassungen oder Berechnungen vorgenommen werden müssen, nachdem andere GameObjects und Skripte bereits ihre `Update()`-Logik abgeschlossen haben. Beispielsweise eignet sich `LateUpdate()` gut für Kamera-Anpassungen, bei denen die Position anderer GameObjects bereits aktualisiert wurde.
- **OnEnable() und OnDisable():** Die `OnEnable()`-Methode wird aufgerufen, wenn ein Skript aktiviert wird, während `OnDisable()` aufgerufen wird, wenn es deaktiviert wird. Diese Methoden bieten die Möglichkeit, spezifische Aktionen auszuführen, wenn ein Skript seine Ausführung aufnimmt oder beendet. Entwickler können diese nutzen, um Ressourcen zu laden oder freizugeben, Abonnements auf Ereignisse einzurichten oder abzubrechen, oder um andere vorbereitende oder aufräumende Maßnahmen durchzuführen.

#### 4.1.2.8 Unity Szenen

Unity-Szenen bilden das grundlegende Gerüst für die Gestaltung von Inhalten in der Unity-Entwicklungsumgebung. Sie stellen Assets dar, die alle oder einen Teil eines Spiels oder einer Anwendung enthalten. Szenen bieten eine strukturierte Möglichkeit, verschiedene Elemente wie *Umgebungen*, *Charaktere*, *Hindernisse*, *Dekorationen* und *Benutzeroberflächen* zu organisieren und miteinander zu verknüpfen.

Ein wichtiger Aspekt von Unity-Szenen ist ihre Flexibilität. In einem Projekt können beliebig viele Szenen erstellt werden, um die Organisation und Entwicklung des Spiels zu erleichtern. Durch das modulare Konzept von Szenen können Entwickler einzelne Teile des Spiels separat bearbeiten und optimieren, was die Zusammenarbeit im Team und die Wartung des Projekts vereinfacht.

Unity-Szenen dienen nicht nur der Darstellung von Inhalten, sondern auch der Steuerung des Spielablaufs. Durch die gezielte Aktivierung und Deaktivierung von Szenen können verschiedene Abschnitte des Spiels geladen und entladen werden, was die Leistung und Ressourcennutzung optimiert.

Insgesamt bieten Unity-Szenen eine leistungsstarke und flexible Möglichkeit, Spiele und Anwendungen zu strukturieren, zu organisieren und zu verwalten. Sie bilden das Grundgerüst für die Entwicklung von Inhalten in Unity und ermöglichen es Entwicklern, ihre Visionen zu verwirklichen und ansprechende Spielerlebnisse zu schaffen.<sup>1</sup>

#### 4.1.2.9 Unity Manager

Die präzise und immersive Umsetzung von Augmented-Reality-(AR-)Applikationen erfordert den Einsatz spezialisierter Manager, die grundlegende Funktionen bereitstellen, die für die erfolgreiche Umsetzung verschiedener Szenarien unerlässlich sind. In dieser Appli-

---

<sup>1</sup>Unity Dokumentation **Scenes**

kation werden zwei Manager aus der breiten Palette von Unity bereitgestellten Managern verwendet. Diese sind die folgenden:

- **ARPlaneManager<sup>2</sup>:** Der ARPlaneManager ist ein bedeutender Bestandteil von Unity's Augmented Reality (AR)-Entwicklungsumgebung. Als Teil des Unity-eigenen *Mixed Reality Toolkit*<sup>3</sup> bietet der ARPlaneManager essenzielle Funktionen zur nahtlosen Integration von AR-Elementen in die reale Umgebung. Seine Hauptaufgaben umfassen die automatische Erkennung von *horizontalen* und *vertikalen* Flächen in der Umgebung des Benutzers, was die *präzise Platzierung* virtueller Objekte auf diesen Flächen ermöglicht. Diese Flächen können verschiedene Strukturen wie *Böden*, *Tische* oder andere *flache Oberflächen* umfassen. Nach der Erkennung *überwacht* der ARPlaneManager *kontinuierlich* die Bewegungen der Flächen in Echtzeit, was essenziell ist, um die *Stabilität* virtueller Inhalte auf den realen Flächen zu gewährleisten. Erkannte Flächen können durch *Texturmarkierungen* visuell hervorgehoben werden, um dem Benutzer die Grenzen dieser Flächen deutlicher zu zeigen und die Integration von virtuellen Objekten zu verbessern. Zudem erleichtert der ARPlaneManager das Platzieren virtueller 3D-Objekte in der realen Welt, indem er eine Referenz für die Position und Ausrichtung der erkannten Flächen bereitstellt.
- **ARRaycastManager<sup>3</sup>:** Der ARRaycastManager in Unity ist eine wichtige Komponente für die Entwicklung von Augmented Reality (AR)-Anwendungen. Er ermöglicht es, *Raycasts* von einem *festgelegten Ursprungspunkt* aus durchzuführen, um *Kollisionen* oder *Treffer* mit Objekten in der AR-Umgebung zu erkennen. Diese Funktionalität ist entscheidend für die genaue Platzierung virtueller 3D-Objekte in der realen Welt, basierend auf den Interaktionen des Benutzers. Der ARRaycastManager bietet somit eine grundlegende Funktionalität zur nahtlosen Integration von virtuellen Elementen in die physische Umgebung.

Die erfolgreiche Umsetzung der funktionalen Anforderungen in den spezifischen Augmented-Reality-(AR)-Anwendungsszenarien des *Knapsack Problems* sowie des *Pings* hängt maßgeblich von der Integration und Anwendung der zwei genannten Manager, für die Schaffung einer qualitativ *hochwertigen*, *präzisen* und *immersiven* Benutzererfahrung.

Im Kontext des *Knapsack Problem* Anwendungsszenarios spielt der ARPlaneManager eine zentrale Rolle. Durch die Markierung von horizontalen Flächen in der Benutzerumgebung garantiert dieser eine präzise Platzierung des virtuellen Inventar-Objekts und gewährleistet dadurch eine stabile Integration in die reale Umgebung.

Im Kontext des Anwendungsszenarios des *Ping* spielen beide Manager eine wichtige Rolle. Der ARPlaneManager erkennt die ARPlanes in der Umgebung und der ARRaycast-Manager erkennt anhand eines Rays, auf welchem ARPlane das reale Kabel, über das das Ping-Paket visualisiert wird, liegt.

Insgesamt sind diese Manager wichtige Ressourcen, da sie die technische Umsetzbarkeit und Effektivität von AR-Anwendungen maßgeblich beeinflussen. Durch ihre integrierte Anwendung wird eine nahtlose Verschmelzung von virtuellen und physischen Elementen realisiert, was eine immersive und präzise AR-Benutzererfahrung sowohl in dem Knapsack-Problem als auch auf dem Ping Anwendungsszenarios gewährleistet.

#### 4.1.2.10 Unity GameObjects und Komponente

Die Konzeption und Verwaltung von GameObjects stellt einen essenziellen Bestandteil der Entwicklungsumgebung von Unity dar. Ein *GameObject* repräsentiert in dieser Umgebung

---

<sup>2</sup>Unity Dokumentation **PlaneManager**

<sup>3</sup>Unity Dokumentation **RaycastManager**

jede *Entität* innerhalb eines *digitalen Szenarios*, sei es ein *Charakter*, eine *Umgebungskomponente* oder ein *Effekt*. Diese grundlegenden Objekte agieren als *Behälter für Komponenten*, welche die *Funktionalität* und das *Verhalten* definieren.

Im Kontext von Unity bilden GameObjects die grundlegenden Bausteine einer Szene. Sie sind abstrakte Entitäten, die allein nicht aktiv handeln können, sondern erst durch das *Hinzufügen* von *Komponenten* zu funktionalen Einheiten werden. Die Zuweisung von *Eigenschaften* und *Verhalten* erfolgt durch das Anbringen spezifischer Komponenten an ein GameObject. Beispielsweise kann einem GameObject, das das Konzept einer Lichtquelle repräsentiert, eine *Lichtkomponente* zugewiesen werden.

Komponenten in Unity dienen dazu, die Eigenschaften und das Verhalten von GameObjects zu definieren. Sie können beispielsweise einer Lichtquelle die Fähigkeit verleihen, Licht zu emittieren, oder einem Charakter die Möglichkeit geben, sich zu bewegen und mit seiner Umgebung zu interagieren. Die Flexibilität von Unity zeigt sich in der Vielfalt der verfügbaren Komponenten, die sowohl vordefiniert als auch maßgeschneidert sein können. Entwickler können mithilfe der Unity Scripting API eigene Komponenten erstellen, um spezifische Verhaltensweisen zu implementieren und die Funktionalität ihrer GameObjects zu erweitern.

Insgesamt bilden GameObjects und deren Komponenten das Rückgrat der Entwicklung von Spielen und interaktiven Anwendungen in Unity. Ihr Verständnis und ihre effektive Verwaltung sind entscheidend für die erfolgreiche Umsetzung digitaler Szenarien und tragen maßgeblich zur Entwicklung innovativer und ansprechender Spielerlebnisse bei.<sup>4</sup>.

#### 4.1.2.11 Unity Prefabs

Unity bietet mit *Prefabs* eine äußerst praktische Funktion zur Erstellung und Wiederverwendung von Game-Objekten. Prefabs sind modulare, wiederverwendbare Elemente, die als Blaupausen für die Konstruktion von Game-Objekten dienen. Sie ermöglichen Entwicklern, spezifische Objekte oder Objektstrukturen zu erstellen und zu katalogisieren. Diese Muster können dann in unterschiedlichen Szenarien oder Projekten repliziert werden, was eine effiziente Wiederverwendung und Konsistenz über verschiedene Projektumgebungen hinweg ermöglicht.

→  
HAYLAZ

Prefabs bieten zahlreiche Vorteile für die Gestaltung und Entwicklung von Projekten. Ihr Hauptvorteil liegt in ihrer Fähigkeit, eine effiziente und konsistente Gestaltung von Projekten zu ermöglichen. Sie erleichtern die Wiederverwendung von Designelementen und tragen so zur Effizienz und Konsistenz des Designprozesses bei.

In komplexen Projekten, die eine Vielzahl von Objekten wie Charaktere, Umgebungen und visuelle Effekte beinhalten, erweisen sich Prefabs als äußerst nützlich. Sie dienen als wiederverwendbare Vorlagen, die es ermöglichen, einmal erstellte Elemente zu speichern und in unterschiedlichen Szenarien wiederzuverwenden. Dies eliminiert die Notwendigkeit, jedes Mal, wenn eine neue Szene erstellt werden muss, von Grund auf neu zu beginnen. Darüber hinaus trägt die Verwendung von Prefabs dazu bei, eine konsistente Designästhetik über das gesamte Projekt hinweg zu gewährleisten. <sup>56</sup>

Ein weiterer bedeutsamer Vorteil von Prefabs besteht in ihrer Fähigkeit, eine unkomplizierte Aktualisierung und Iteration von Projektobjekten zu ermöglichen. Bei notwendigen Modifikationen an einem spezifischen Objekt kann das zugehörige Prefab bearbeitet werden. Diese Modifikationen werden dann automatisch auf alle Instanzen dieses Prefabs angewendet, die in der Szene implementiert sind. Dies optimiert den Workflow und vereinfacht

<sup>4</sup>Unity Dokumentation **GameObjects**

<sup>5</sup>Unity-Dokumentation, **Prefabs**

<sup>6</sup>Unity-Dokumentation, **Prefabs2**

den Prozess erheblich.

Innerhalb des Projekts wird die Prefab-Technologie in folgenden Bereichen eingesetzt:

- **3D-Modelle ??:** Die dreidimensionalen Modelle sind als einzelne Prefabs konzipiert. Dies ermöglicht uns, häufige Änderungen am Aussehen der Modelle vorzunehmen, ohne sie einzeln ersetzen zu müssen. Beispiele für die Verwendung dieser Modelle sind QR-Codes, Menü-Objekte und das *Nachrichtenpaket*, welches ein Modell in einem Prefab ist.
- **QR-Codes ??:** In unserer Szene arbeiten wir gleichzeitig mit mehreren QR-Codes. Durch die Verwendung der Prefab-Technologie können wir diese mit minimalem Aufwand häufig instanziieren. Auch nach der Instanziierung ist die Arbeit mit diesen virtualisierten QR-Codes vereinfacht.
- **Nachrichteninformationen ??:** Im zweiten Anwendungsszenario finden *Prefabs* weiterhin Anwendung. Bei der Versendung von Nachrichten müssen verschiedene Informationen angezeigt werden. Der Zugriff auf diese Informationen ist einfacher, wenn sie in einem Prefab verpackt sind.

Zusammenfassend lässt sich sagen, dass Prefabs ein unverzichtbares Werkzeug für die effektive Projektentwicklung in Unity sind. Sie ermöglichen es Entwicklern, Zeit zu sparen, die Konsistenz ihres Projekts zu gewährleisten und den Prozess der Aktualisierung und Iteration von Projektobjekten zu optimieren. Daher sind sie ein wesentlicher Bestandteil jeder effektiven und effizienten Projektentwicklungsumgebung.

#### 4.1.3 Deployment der Anwendung

Die Entwicklung unserer Anwendung findet größtenteils auf unseren Laptops statt. Jedoch ist es auf diesen Geräten nur begrenzt möglich, die AR-Funktionalitäten zu testen. Um die Anwendung vollständig auf einem AR-fähigen Gerät zu überprüfen, muss sie auf dieses geladen werden. In diesem Abschnitt wird der genaue Prozess beschrieben, wann und wie die Anwendung auf ein AR-fähiges Gerät deployt wird.

→  
HAYLAZ

##### 4.1.3.1 Voraussetzungen für das Deployment

Damit das Deployment der Anwendung auf einem AR-fähigen Gerät erfolgreich verläuft, müssen bestimmte Voraussetzungen erfüllt sein:

- **Kompliiertes Unity-Projekt:** Alle erforderlichen Dateien und Ressourcen der Anwendung werden während des Build-Prozesses zu einem ausführbaren Unity-Paket kompiliert.
- **Netzwerkverbindung:** Es ist notwendig, dass sowohl das AR-fähige Gerät als auch der Computer, auf dem der Build durchgeführt wurde, sich im selben Netzwerk befinden. Nur so kann das Unity-Paket über das Netzwerk auf das AR-fähige Gerät übertragen werden.
- **Authentifizierung:** Der Computer, der sich mit der HoloLens verbinden und die Anwendung deployen möchte, muss von der Brille authentifiziert werden. Dies gewährleistet, dass nur autorisierte Geräte Zugriff auf die Brille haben und die Übertragung sicher erfolgt.

##### 4.1.3.2 Deployment-Prozess

Die Entwicklung und Bereitstellung einer Anwendung für die HoloLens 2 unter Verwendung von Unity und der Universal Windows Platform (UWP) ist ein strukturierter Prozess, der

mehrere Schritte umfasst, beginnend mit der Kompilierung des Unity-Projekts. Dieser Prozess findet auf dem Computer statt, auf dem sich das Unity-Projekt befindet, und kann über das Tastenkürzel **Strg + Umschalt + B** aufgerufen werden, um die Build-Einstellungen in Unity zu öffnen.

In den Build-Einstellungen wird die Zielplattform festgelegt, auf die die Anwendung bereitgestellt werden soll. Für die HoloLens 2 ist die Universal Windows Platform (UWP) die geeignete Zielplattform aufgrund ihrer Kompatibilität mit diesem Gerät. Die UWP ermöglicht die Entwicklung von Anwendungen, die auf verschiedenen Windows 10/11-Geräten einschließlich der HoloLens 2 ausgeführt werden können.

Es gibt verschiedene Build-Einstellungen, wie in Abbildung 4.1 dargestellt. Besonders relevant sind dabei die Einstellungen für die *Architektur* und die *Build-Konfiguration*. Die Architektur wird entsprechend der HoloLens 2 auf ARM 64-Bit festgelegt. Die Build-Konfiguration sollte auf *Release* eingestellt sein. Die übrigen Optionen sollten entsprechend den Vorgaben in der Abbildung beibehalten werden. Diese Einstellungen sind wichtig, um in den nächsten Schritten eine funktionierende Übertragung zu garantieren.

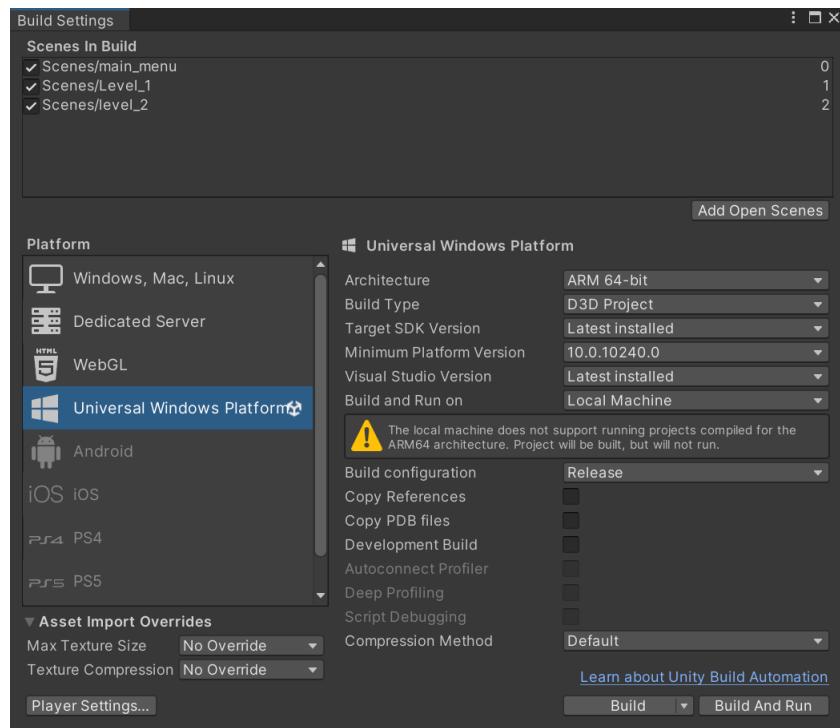


Abbildung 4.1: Build-Einstellungen in Unity

Nach einem erfolgreichen Build erhalten Sie einen Ordner mit den benötigten Dateien für das Deployment. Darin befindet sich unter anderem eine Visual Studio Solution (*VS Solution*), die Informationen zur Struktur des Projekts enthält und Build-Konfigurationen verwaltet.<sup>7</sup>

Der nächste Schritt erfolgt in dieser Visual Studio Solution, in unserem Fall *AARIE.sln*. Nach dem Öffnen der Datei müssen die Konfiguration auf *Release* und die Plattform auf *ARM64* eingestellt werden. Diese zwei Einstellungen sind auf der *Toolbar* zu finden. Außerdem muss die IP-Adresse des AR-fähigen Geräts in den Projekteigenschaften unter Debugging gesetzt werden. Die IP-Adresse der HoloLens 2 kann auf der Brille in den Ein-

<sup>7</sup>Visual Studio Solution, **VS-Solution**

stellungen unter Netzwerk gefunden werden. In den Projekteigenschaften ist noch folgende Einstellung zu treffen:

Der Authentication Mode *Universal (Unencrypted Protocol)* wird gewählt, um eine reibungslose Kommunikation zwischen dem Computer und der HoloLens 2 zu ermöglichen, insbesondere während des Deployments der Anwendung. Dieser Modus erlaubt eine unverschlüsselte Kommunikation zwischen den Geräten, was in diesem speziellen Anwendungsszenario aus verschiedenen Gründen vorteilhaft ist. Zum einen vereinfacht die Verwendung eines unverschlüsselten Protokolls die Konfiguration und ermöglicht eine schnellere Einrichtung der Verbindung zwischen dem Entwicklungscomputer und der HoloLens 2. Da dies ein Entwicklungs- und Testumfeld ist, wo Sicherheit weniger prioritär ist als Effizienz und Schnelligkeit, wird diese einfachere Konfiguration bevorzugt. Darüber hinaus minimiert die Verwendung eines unverschlüsselten Protokolls das Risiko von Verbindungsproblemen und -verzögerungen, die bei verschlüsselten Kommunikationsprotokollen auftreten können, insbesondere in lokalen Netzwerken oder Umgebungen, in denen die Infrastruktur möglicherweise nicht optimal konfiguriert ist. In einer Entwicklungs- und Testumgebung, in der die Anwendung iterativ entwickelt und getestet wird, steht die Effizienz im Vordergrund. Die Verwendung eines unverschlüsselten Protokolls erleichtert den Entwicklungsprozess, da sie weniger Overhead und Komplexität mit sich bringt. Es ist jedoch wichtig zu beachten, dass in produktiven Umgebungen, in denen Sicherheit eine höhere Priorität hat, verschlüsselte Kommunikationsprotokolle verwendet werden sollten, um die Vertraulichkeit und Integrität der übertragenen Daten zu gewährleisten.<sup>8</sup>

Nachdem alle Einstellungen vorgenommen wurden, kann die

Anwendung auf die HoloLens 2 deployt werden. Dazu muss die Solution mit *Start without Debugging* gestartet werden. Nach einem längeren Ladevorgang wird die Anwendung auf der HoloLens 2 geladen und gestartet.

Falls die Anwendung beendet wird, kann sie auf der HoloLens 2 unter *Start → Alle Apps → AARIE* erneut gestartet werden. Falls Änderungen an der Anwendung auf den Computern vorgenommen wurden, muss der gesamte Prozess erneut durchgeführt werden. Es ist wichtig zu beachten, dass die Anwendung nicht automatisch aktualisiert wird.

#### 4.1.3.3 Erstmaliges Deployment

Beim erstmaligen Deployment auf die HoloLens kann es zu Problemen kommen, die durch die Authentifizierung der HoloLens verursacht werden. In diesem Fall muss der "neue" Computer sich auf der HoloLens erneut authentifizieren. Dazu werden Sie nach dem Starten des Deployments aufgefordert. Auf dem Computer wird ein Authentifizierungscode angezeigt, der auf der HoloLens unter *Einstellungen → Update & Sicherheit → Für Entwickler → Koppeln* eingegeben wird. Nachdem die Authentifizierung erfolgreich abgeschlossen wurde, kann das Deployment erneut gestartet werden.

## 4.2 Objektdesign

Die Bedeutung des Objektdesigns nimmt in der vorliegenden Arbeit einen zentralen Stellenwert ein, wie bereits im Abschnitt ?? erläutert wurde. Da sämtliche Modelle und Gegenstände eigenständig erstellt werden sollen, ist ein fundiertes Verständnis der grundlegenden Konzepte und Prozesse des Objektdesigns unerlässlich. Im nachfolgenden Abschnitt werden daher alle relevanten Begriffe erläutert, die beim Modellieren auftreten, und die einzelnen Schritte des Modellierungsprozesses werden in möglichst detaillierter Form durchgegangen.

→ LAM-  
PEL

---

<sup>8</sup>Universal Unencrypted Protocol, **VS-Protocol**

Das Objektdesign umfasst nicht nur die Schaffung von ästhetisch ansprechenden und funktionalen 3D-Modellen, sondern auch die Optimierung dieser Modelle, um ein optimales Benutzererlebnis sicherzustellen. Dies beinhaltet Aspekte wie die Effizienz der Modelle in Bezug auf Rechenleistung und Speicherplatz, die Benutzerfreundlichkeit sowie die visuelle und funktionale Qualität der Modelle.

### 4.2.1 Texturen

Texturen sind ein wichtiger Bestandteil der Modellierung und verbessern die visuelle Erscheinung von Objekten, um sie realistischer wirken zu lassen. Sie beschreiben die Oberflächenbeschaffenheit eines Objekts, einschließlich Eigenschaften wie Farbe, Rauigkeit, Reflexionsvermögen, Glanz, Lichtdurchlässigkeit und mehr. In Blender werden Texturen typischerweise als Grafiken oder Muster verwendet, die auf die Oberfläche eines 3D-Modells projiziert werden. Im Projekt wurde viel mit *Bild-Texturen* gearbeitet, um möglichst effiziente, realistische und ästhetisch ansprechende Modelle zu erzeugen.

#### 4.2.1.1 Bild-Texturen in der 3D-Modellierung

Im folgenden Abschnitt wird erläutert, was unter dem Begriff **Bild-Textur** zu verstehen ist und wie die Umsetzung und Arbeit mit Bild-Texturen im Kontext von Blender funktioniert.

Diese Texturen können als Bilddateien importiert oder innerhalb des Programms selbst erstellt werden.

Die Verwendung von Bild-Texturen ermöglicht es Benutzern, eine Vielzahl von Oberflächeneffekten zu erzeugen, wie beispielsweise Holzmaserungen, Metalltexturen, Stoffmuster und mehr. Durch die Verwendung einer geeigneten Textur kann die visuelle Qualität eines Modells erheblich verbessert werden.

Im Blender-Programm können Bild-Texturen mithilfe des *Shader-Editors* auf das Material des 3D-Objekts angewendet werden. Hierfür wird eine *Image-Texture-Node* erstellt, welche den Farbwert der Textur an das Material übergibt. Es können verschiedene Einstellungen vorgenommen werden, um die Art und Weise der Texturprojektion zu steuern. Zum Beispiel kann die Projektionsart (Box, Flat, Sphere usw.) festgelegt werden, um anzugeben, wie die Textur auf die Oberfläche des Modells angewendet wird. Weitere Einstellungen können die Skalierung der Textur, die Wiederholungsmuster, die Mischmodi und andere Textureigenschaften umfassen.<sup>9</sup>

#### Einschub Shader-Editor

Dieser Editor hat in Blender eine spezielle Funktion. Hier können mithilfe von sogenannten *Nodes* Eigenschaften für Texturen und Materialien vergeben werden. Der Editor wurde im Projekt hauptsächlich für die Anwendung von Bild-Texturen genutzt.

---

<sup>9</sup>Blender **Texturen**

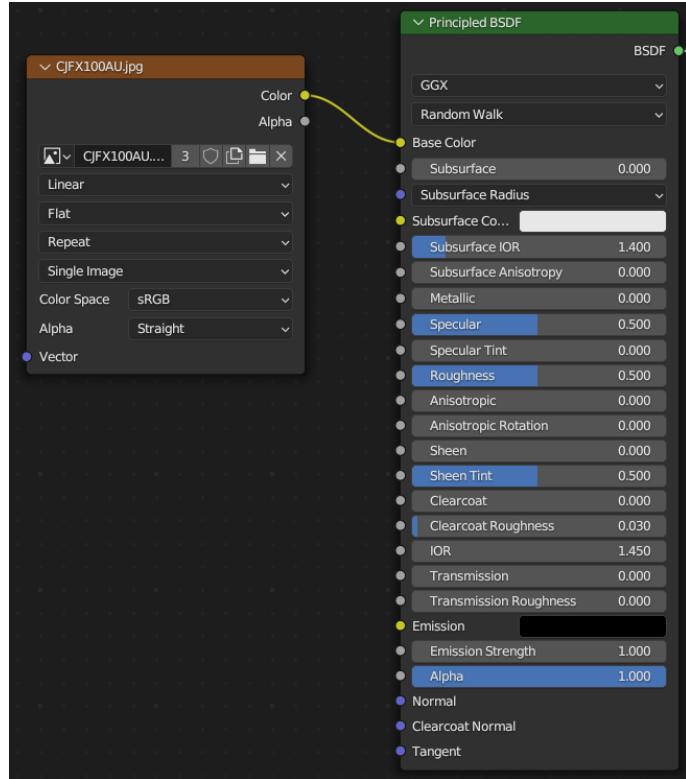


Abbildung 4.2: Beispielhafte Ansicht auf Bild-Textur-Nodes im Shader-Editor

In Abbildung 4.2 ist zu erkennen, wie die Bild-Textur-Node den Farbwert an das Material weitergibt. Als Beispiel wurde hier der Taschenrechner verwendet, der im Laufe des Kapitels noch näher erläutert wird. Die Abbildung zeigt die verschiedenen Konfigurationsmöglichkeiten, die bereits im Abschnitt 4.2.1.1 erläutert wurden.

#### 4.2.2 Mesh

In der 3D-Grafik bezeichnet ein Mesh eine Sammlung von Eckpunkten, Kanten und Flächen, die die Geometrie eines 3D-Objekts definieren. Es bildet die Grundlage für die Darstellung von Objekten in einer virtuellen Umgebung. Ein Mesh kann aus einer beliebigen Anzahl von Polygonen bestehen, die die äußere Form und Struktur des Objekts definieren.

Die Rolle eines Meshes bei der Modellierung besteht darin, die Form, Oberflächenstruktur und Details eines Objekts zu definieren. Durch die Anordnung und Verbindung der Vertices, Edges und Faces können komplexe geometrische Formen erzeugt werden. Polygone ermöglichen dabei die Darstellung von Objekten mit unterschiedlichen Detailgenauigkeiten. Je mehr Polygone ein Mesh hat, desto detaillierter und realistischer kann das Modell dargestellt werden. Allerdings führt dies auch zu einem höheren Bedarf an Rechenleistung und Speicherplatz.

Meshes werden in verschiedenen Bereichen der 3D-Grafik verwendet, wie zum Beispiel der Modellierung von Objekten, der Animation und der Visualisierung. Sie dienen als Grundlage für die Erstellung von digitalen Szenen.

#### 4.2.3 Rolle von Polygonen in einem Modell

Polygone sind grundlegende Bausteine eines Meshes und spielen eine wichtige Rolle bei der Darstellung von 3D-Modellen. Sie bestehen aus einer Reihe von Eckpunkten (Verti-

ces), Kanten und Flächen, die diese Punkte miteinander verbinden. In einem 3D-Modell definieren Polygone die Form, die Oberflächenstruktur und die Details eines Objekts.<sup>10</sup>

Durch die Anordnung und Verbindung von Polygone entstehen komplexe Strukturen, die eine visuelle Darstellung von Objekten ermöglichen. Je mehr Polygone ein Modell hat, desto detaillierter und realistischer kann es dargestellt werden. Allerdings führt dies auch zu einem höheren Bedarf an Rechenleistung und Speicherplatz. Polygone sind somit ein wesentlicher Bestandteil eines Meshes und tragen maßgeblich zur visuellen Qualität und Komplexität eines 3D-Modells bei.

#### 4.2.4 Optimierung der einzelnen Modelle

Der Entwurf von 3D-Objekten mit Blender erfordert ein systematisches Vorgehen, insbesondere im Zusammenhang mit der Optimierung dieser. Im Folgenden werden spezifische Aspekte dieses Prozesses beleuchtet, darunter die Polygonreduktion und die Texturoptimierung.

##### 4.2.4.1 Polygonreduktion

Die Polygonreduktion ist eine Technik, die darauf abzielt, die Anzahl der Polygone in einem 3D-Modell zu reduzieren, um die Belastung der Hardware zu verringern, insbesondere bei Echtzeitanwendungen wie Computerspielen oder Simulationen. Eine effiziente Polygonreduktion ermöglicht eine bessere Leistung und eine schnellere Darstellung der Modelle auf verschiedenen Plattformen.<sup>11</sup>

In der Computergrafik steht die Anzahl der Polygone eines Modells in direktem Zusammenhang mit dem benötigten Speicherplatz und der Rechenleistung. Je mehr Polygone ein Modell hat, desto mehr Daten müssen verarbeitet und gerendert werden, was zu höheren Anforderungen an die Hardware führt. Durch eine Reduzierung der Polygone können diese Ressourcen effizienter genutzt werden, ohne dass die visuelle Qualität des Modells wesentlich beeinträchtigt wird.

Werkzeuge wie der *Decimate Modifier* in Blender bieten die Möglichkeit, die Anzahl der Polygone automatisch zu reduzieren und gleichzeitig visuelle Artefakte zu minimieren. Artefakte in Bezug auf 3D-Modellierung sind unerwünschte visuelle Effekte oder Fehler, die während des Modellierungs- oder Renderingprozesses auftreten können.

Dennoch ist es ratsam, bereits während des Modellierungsprozesses darauf zu achten, keine unnötigen zusätzlichen Unterteilungen zu erzeugen, um eine optimale Ausgangsbasis für die Polygonreduktion zu schaffen. Eine Möglichkeit hierfür ist die regelmäßige Nutzung der *Merge*-Funktion, welche es ermöglicht, einzelne Punkte anhand ihres Abstandes zu kombinieren und einen einzelnen Punkt daraus zu erstellen. Wenn dieser Vorgang auf das gesamte Modell angewendet wird, verhindert er außerdem ungewollte Kopien von Eckpunkten durch beispielsweise unbeabsichtigte und abgebrochene *Extrusion*.

Die Polygonreduktion ist ein wichtiger Bestandteil der 3D-Modellierung und -Optimierung, da sie dazu beiträgt, die Leistung von Anwendungen zu verbessern und die Benutzererfahrung zu optimieren. Durch eine sorgfältige Anwendung dieser Technik können qualitativ hochwertige 3D-Modelle erstellt werden, die sowohl ästhetisch ansprechend als auch effizient zu verarbeiten sind.

---

<sup>10</sup>CGIFurniture What are Polygons

<sup>11</sup>All3DP How to reduce Polygons

#### 4.2.4.2 Texturenoptimierung

Die Optimierung von Texturen ist ein wesentlicher Bestandteil der Gestaltung von 3D-Modellen für Augmented Reality (AR)-Anwendungen, da sie einen direkten Einfluss auf die Benutzererfahrung haben. Bei der Auswahl geeigneter Texturen muss der Auflösung besondere Aufmerksamkeit geschenkt werden, insbesondere im Hinblick auf die begrenzte Leistungsfähigkeit von AR-Geräten wie der HoloLens 2.

Während des Texturierungsprozesses wurde stark auf die Leistung der HoloLens 2 und die Bilder pro Sekunde geachtet, um zu verhindern das zu hochauflösende Texturen, das Benutzererlebnis beeinträchtigen. Wurden Einbußen festgestellt, wurden entsprechende Anpassungen an den Texturen vorgenommen. Dies beinhaltete entweder die Suche nach alternativen Texturen oder die Komprimierung der vorhandenen Texturen, um eine geringere Auflösung zu erreichen.

Die Texturoptimierung ist ein iterativer Prozess, der eine ausgewogene Berücksichtigung der visuellen Qualität und der Leistungsfähigkeit der AR-Plattform erfordert. Durch die gezielte Optimierung von Texturen können AR-Anwendungen erstellt werden, die eine ansprechende visuelle Darstellung bieten und gleichzeitig ein flüssiges und immersives Benutzererlebnis gewährleisten.

#### 4.2.5 Export- und Integrationsprozess

In diesem Abschnitt wird der Export- und Integrationsprozess für 3D-Modelle beschrieben, beginnend mit der Auswahl des geeigneten Dateiformats und der Berücksichtigung des Koordinatensystems.

Der Export- und Integrationsprozess ist ein entscheidender Schritt bei der Übertragung von 3D-Modellen aus der Konstruktions- oder Modellierungssoftware in eine Zielumgebung, sei es eine Spiele-Engine, eine Virtual-Reality-Plattform oder eine AR-Anwendung. Dieser Prozess umfasst mehrere Schritte, um sicherzustellen, dass das Modell korrekt dargestellt und funktional in die Zielumgebung integriert wird.

##### 4.2.5.1 Dateiformat

Als Dateiformat für den Export von 3D-Modellen in diesem Projekt wurde das von Autodesk entwickelte Filmbox-Format (FBX) gewählt. FBX wurde aufgrund seiner weit verbreiteten Unterstützung und seiner Fähigkeit, umfassende Informationen über geometrische Formen, Materialien, Animationen und andere Szenendaten zu speichern, ausgewählt.

FBX ist ein proprietäres Format, das speziell für den Austausch von 3D-Inhalten zwischen verschiedenen Anwendungen entwickelt wurde. Es bietet eine hierarchische Struktur, die es ermöglicht, komplexe Modelle (zum Beispiel von Blender) zu organisieren und zu übertragen. Diese Struktur umfasst Knoten, die verschiedene Elemente der 3D-Szene repräsentieren, wie Geometrie, Materialien, Animationen, Kamerassen und Lichtquellen.

Durch die Verwendung von FBX können 3D-Modelle nahtlos zwischen verschiedenen Softwareanwendungen und Plattformen ausgetauscht werden, was die Zusammenarbeit und Integration in Projekten wie diesem, das Unity als Engine verwendet, erleichtert. Die Wahl des FBX-Formats bietet somit eine solide Grundlage für einen effizienten Workflow und eine erfolgreiche Umsetzung des Projekts.

##### 4.2.5.2 Koordinatensysteme

Die Berücksichtigung und korrekte Handhabung von dem Koordinatensystem während des Modellierungsprozesses ist ein entscheidender Aspekt für die nahtlose Integration von

3D-Modellen in verschiedene Anwendungen. In diesem Projekt wurden spezifische Maßnahmen ergriffen, um potenzielle Probleme im Zusammenhang mit Koordinatensystemen zu minimieren.

Während der Modellierung wurden alle Objekte im Koordinatenursprung platziert, um sicherzustellen, dass beim Export der Modelle nach Unity keine Komplikationen hinsichtlich Platzierung und Ausrichtung auftreten. Diese Vorgehensweise trägt dazu bei, mögliche Diskrepanzen zwischen den Koordinatensystemen der Modellierungssoftware und der Zielplattform zu vermeiden, was den Integrationsprozess vereinfacht und beschleunigt.

Darüber hinaus wurden alle Modelle in der gleichen Ausrichtung modelliert, um zusätzliche Anpassungen in Unity zu vermeiden. Diese konsistente Orientierung stellt sicher, dass alle Modelle bereits in einer standardisierten Orientierung vorliegen, was die Notwendigkeit weiterer manueller Eingriffe minimiert und einen reibungsloseren Arbeitsablauf gewährleistet.

Für den Fall, dass Modelle dennoch mit einer falschen Rotation exportiert werden, wurden entsprechende Korrekturen direkt im Unity Inspector vorgenommen. Diese Nachjustierung ermöglicht es, eventuelle Fehler in der Ausrichtung der Modelle schnell und effizient zu beheben, ohne den Modellierungsprozess zu unterbrechen oder zusätzlichen Aufwand zu verursachen.

Insgesamt zeigt die Berücksichtigung von dem ausgewählten Koordinatensystem während des gesamten Workflows einen proaktiven Ansatz bei der Modellierung und Integration von 3D-Objekten. Die Umsetzung dieser Maßnahmen wird die Konsistenz und Effizienz des Projekts verbessern und gleichzeitig mögliche Komplikationen im Zusammenhang mit Koordinatensystemen effektiv vermeiden.

## 4.2.6 Add-Ons und Plugins

Während der Modellierungsphase wurden verschiedene Add-ons und Plug-ins genutzt, um die Modellierung effizienter zu gestalten und die Funktionalität von Blender zu erweitern. Im folgenden Abschnitt werden die wichtigsten Add-ons und Plug-ins erläutert, die zur Unterstützung und Verbesserung der Modellierungserfahrung eingesetzt wurden.

### 4.2.6.1 Looptools: Optimierung von Topologie und Oberflächen

Das Add-on *Looptools*<sup>12</sup> für Blender stellt eine wichtige Ergänzung für die Flächenmodellierung und Topologieoptimierung dar. Insbesondere bei der Umwandlung von rechteckigen oder quadratischen Flächen in kreisförmige Flächen erweist sich dieses Werkzeug als äußerst nützlich. Im *Stromverteiler-Modell* müssen beispielsweise die einzelnen runden Steckbuchsen aus der eckigen Basis heraus modelliert werden.

Als Hauptwerkzeug wurde das so genannte *Circle*-Werkzeug verwendet, das speziell zur Lösung des oben genannten Problems entwickelt wurde. Mit dem *Circle*-Werkzeug können rechteckige Flächen unter Berücksichtigung der Topologie und der Anzahl der Polygone nahtlos in kreisförmige Flächen umgewandelt werden.

Das *Circle*-Werkzeug bietet verschiedene Einstellungsmöglichkeiten, um die extrahierte Figur optimal für den jeweiligen Zweck zu konfigurieren. Die wichtigsten Konfigurationspunkte sind

- **Best Fit:** Dieses Werkzeug berechnet einen Kreis mit Hilfe einer nichtlinearen Methode der kleinsten Quadrate. Dadurch wird sichergestellt, dass der berechnete Kreis optimal zu den ausgewählten Eckpunkten passt.

---

<sup>12</sup>Blender **LoopTools**

- **Fit Inside:** Mit dieser Option wird der Kreis so berechnet, dass kein Eckpunkt vom Kreismittelpunkt entfernt wird. Dies ist besonders nützlich, wenn die Topologie des umgebenden Mesh erhalten bleiben soll.

Zusätzlich ermöglicht das *Circle*-Werkzeug die Anpassung weiterer Konfigurationsparameter wie Radius und Regularität, um die extrahierte Figur feiner abzustimmen und besser an individuelle Anforderungen anzupassen.

Insgesamt trägt das *Lootools* Add-On dazu bei, den Modellierungsprozess effizienter zu gestalten und die Qualität der resultierenden Modelle zu verbessern, indem es eine Reihe präziser und flexibler Werkzeuge für die Topologie- und Flächenoptimierung bereitstellt.

Dieses Add-on wurde konkret bei den folgenden Modellen eingesetzt:

- **Handy-Modell:** Für die Modellierung der einzelnen Kameras sowie der runden Ein- und Ausgänge
- **Stromverteiler-Modell:** Zur Einarbeitung der einzelnen Steckbuchsen wurde bereits zu Beginn erwähnt.
- **USB-Stick-Modell:** Um den Übergang von der Halterung zum eigentlichen USB-Stick zu modellieren.

#### 4.2.6.2 Images as Planes: Effiziente Integration von Texturen

Das Add-on *Images as Planes*<sup>13</sup> spielt eine entscheidende Rolle in der Modellierungspraxis, insbesondere im Bereich der realistischen Modellierung.

Das Hauptanwendungsgebiet dieses Add-ons in dem Projekt liegt in der Möglichkeit, Vorschaubilder für die Modellierung hinter Objekten zu platzieren, um eine realistischere und präzisere Modellierung zu ermöglichen. Diese Funktion bietet die Möglichkeit, reale Bilder als Referenz in Blender-Szenen zu integrieren und als Hintergrund für die Modellierung zu verwenden. Durch die direkte Integration von Bildern in die Arbeitsumgebung können feine Details und Proportionen besser beurteilt und reproduziert werden, was zu einer verbesserten Qualität der Modelle führt.

Die Verwendung von *Images as Planes* trägt somit wesentlich zur Erhöhung der Genauigkeit und Realitätsnähe der Modellierung bei, indem sie eine effiziente Möglichkeit bietet, reale Referenzen in den Modellierungsprozess zu integrieren. Diese Funktion ist besonders nützlich bei der Modellierung von Objekten, die auf realen Vorbildern basieren, da sie es dem Modellierer ermöglicht, direkt aus Bildern heraus zu arbeiten und so einen höheren Detaillierungsgrad zu erreichen.

Dieses Add-on wurde in allen modellierten Objekten des Projekts eingesetzt, um anhand von Referenzbildern anschaulichere Modelle zu entwerfen.

#### 4.2.7 Modi

In Blender stehen verschiedene Modi<sup>14</sup> zur Verfügung, mit denen unterschiedliche Aspekte eines Objekts bearbeitet und manipuliert werden können. Diese verschiedenen Modi in Blender bieten eine umfassende Palette an Werkzeugen und Funktionen, mit denen die Benutzer ihre 3D-Modelle auf vielfältige Weise bearbeiten und verfeinern können.

##### 4.2.7.1 Object-Modus

Der Object-Modus ist der Standardmodus in Blender und steht für alle Arten von Objekten zur Verfügung. In diesem Modus können grundlegende Transformationen wie Posi-

---

<sup>13</sup>Blender **Images as Planes**

<sup>14</sup>Blender **Modi**

tionierung, Rotation und Skalierung sowie Duplizierung und andere Objekteigenschaften bearbeitet werden.

#### 4.2.7.2 Edit-Modus

Der Edit-Modus ist ein spezialisierter Modus zum Bearbeiten der Form eines Objekts. Hier können mithilfe verschiedener Werkzeuge und Kontrollpunkte einzelne Knoten/(Eck-)Punkte, Kanten und Flächen des Objekts bearbeitet werden. Dies ermöglicht detaillierte Manipulationen und Anpassungen an der Geometrie des Objekts, was insbesondere für die Modellierung von entscheidender Bedeutung ist.<sup>15</sup>

#### 4.2.7.3 Texture-Paint-Modus

Der Texture-Paint-Modus ist ein spezieller Modus, der es ermöglicht, Texturen direkt auf das Mesh<sup>16</sup> eines Objekts zu malen. Dieser Modus ist ausschließlich auf das Bearbeiten von Meshes beschränkt und bietet eine intuitive Möglichkeit, Texturen im 3D-Viewport zu zeichnen und zu bearbeiten. Durch die direkte Malerei auf dem Objekt können komplexe Texturen und Oberflächeneffekte einfach erstellt und angepasst werden.<sup>17</sup>

### 4.2.8 Hierarchie

In der 3D-Modellierung bezieht sich Hierarchie auf die strukturierte Organisation von Objekten innerhalb einer Szene oder eines Modells. Diese Hierarchie wird oft durch eine Baumstruktur dargestellt, in der übergeordnete Objekte untergeordnete Objekte enthalten können. Die Hierarchie spielt eine entscheidende Rolle bei der Verwaltung und Manipulation von Objekten sowie bei der Definition von Beziehungen zwischen ihnen.

- **Eltern-Kind-Beziehungen:** Übergeordnete Objekte werden oft als Eltern bezeichnet und enthalten untergeordnete Objekte, die als Kinder bezeichnet werden. Diese Beziehung ermöglicht es, Transformationen wie Verschieben, Drehen und Skalieren auf das übergeordnete Objekt anzuwenden, welche dann auf seine untergeordneten Objekte übertragen werden. Diese Technik ist besonders nützlich für komplexe Strukturen wie Roboterglieder oder hierarchische Modelle.
- **Gruppierung:** Objekte können hierarchisch gruppiert werden, um sie logisch zu organisieren und ihre Handhabung zu erleichtern. Durch die Gruppierung ist es möglich, mehrere Objekte gleichzeitig auszuwählen, zu verschieben oder zu bearbeiten, ohne jedes einzelne Objekt separat manipulieren zu müssen.

Die Hierarchie ist ein grundlegendes Konzept in der 3D-Modellierung. Sie ermöglicht eine effiziente Organisation und Manipulation von Objekten. Durch die kluge Nutzung von Hierarchien können komplexe Modelle erstellt und verwaltet werden. Dies führt zu einer effizienteren Arbeitsweise und einer verbesserten Qualität der Ergebnisse.

Im Projekt wurde dies beispielsweise verwendet, um im Taschenrechner-Modell die vielen Buttons von der Basisform logisch abzutrennen oder beim Handy-Modell die Basisform von den einzelnen Kameras und der Kameraauflagefläche abzutrennen.

---

<sup>15</sup>Blender Vertices

<sup>16</sup>nähere Erklärung siehe Abschnitt 4.2.2

<sup>17</sup>Blender Mesh

### 4.2.9 Modifier

*Modifier* sind Werkzeuge oder Operationen, die auf Objekte in der 3D-Modellierung angewendet werden, um ihr Aussehen oder Verhalten zu verändern, ohne die zugrunde liegende Geometrie dauerhaft zu ändern. Sie ermöglichen es den Modellierenden, komplexe Effekte zu erzielen, ohne manuell jeden einzelnen Aspekt des Modells zu bearbeiten. Modifier sind ein wichtiger Bestandteil vieler 3D-Modellierungssoftware und bieten eine Vielzahl von Funktionen zur Verbesserung des Modellierungsprozesses. Die Verwendung von Modifern bringt mehrere Vorteile mit sich:

- **Non-destructive Bearbeitung:** Modifier werden auf das Modell angewendet, ohne die ursprüngliche Geometrie zu verändern. Dies ermöglicht es, Änderungen vorzunehmen und bei Bedarf zum ursprünglichen Zustand zurückzukehren. Als gutes Beispiel kann hier der *Subdivision-Modifier* herangezogen werden, welcher das gesamte Modell grundlegend einmal zerteilt, sodass man doppelt soviele Unterteilungen und feinere Details hat, bei bedarf kann man die Zerteilung ausmachen oder noch mehr zerteilen. Wenn man händisch ohne Modifier alles subdividen würde, wäre es nicht möglich/ sehr schwer möglich das rückgängig zu machen
- **Effizienzsteigerung:** Durch die Verwendung von Modifern können komplexe Effekte und Veränderungen mit weniger manuellem Aufwand erreicht werden. Dies führt zu einem effizienteren Modellierungsprozess und spart Zeit und Ressourcen. Wenn man das ganze Modell abrunden (*Bevel-Modifier*) oder feiner/detailreicher (*Subdivision-Modifier*) gestalten möchte ist es deutlich zeitintensiver alle kanten und flächen einzeln händisch zu bearbeiten.
- **Experimentierfreude:** Da Modifier nicht-destruktiv sind, können sie leicht hinzugefügt, angepasst oder entfernt werden. Dies ermutigt dazu, verschiedene Optionen auszuprobieren und kreativ zu experimentieren, ohne Angst vor irreversiblen Änderungen haben zu müssen.

Beispiele für Modifier sind Subdivision Surface, Mirror, Bevel und Array<sup>18</sup>. Jeder Modifier hat spezifische Anwendungsfälle und ermöglicht es den Modellierenden, eine Vielzahl von Effekten zu erzielen, von der Glättung von Kanten bis zur Erstellung von komplexen Wiederholungsmustern.

Modifier sind ein wichtiger Bestandteil der 3D-Modellierung. Sie bieten eine flexible und leistungsstarke Möglichkeit, Objekte zu bearbeiten und zu verbessern, ohne die Integrität des Modells zu beeinträchtigen. Die Verwendung von Modifern kann den Modellierungsprozess rationalisieren und die Kreativität der Modellierenden fördern.

#### Subdivision Surface Modifier

Dieser Modifier glättet die Oberfläche eines Modells, indem er die Anzahl der Polygon-subdivisionen erhöht. Dadurch entstehen weichere Kanten und eine insgesamt glattere Oberfläche. Der Subdivision Surface Modifier wird oft verwendet, um organische Formen zu erstellen oder die Details eines Modells zu verfeinern. Er kann auch helfen, die Qualität von Meshes zu verbessern, indem er unregelmäßige Geometrien ausgleicht und sie besser für das Rendern und die Animation geeignet macht.

#### Mirror Modifier

Der Mirror Modifier spiegelt die Geometrie eines Objekts entlang einer oder mehrerer Achsen. Dadurch können symmetrische Modelle erstellt werden, indem nur eine Hälfte mo-

---

<sup>18</sup>nähtere Erklärung im Abschnitt 4.2.10.1

delliert werden muss. Der Modifier erzeugt automatisch die gespiegelte Seite des Objekts, wodurch Zeit und Aufwand gespart werden. Der Mirror Modifier wird häufig bei der Modellierung von symmetrischen Objekten wie Menschen, Fahrzeugen und Gebäuden eingesetzt. Im Projekt wurde er beispielsweise für das Modell des Taschenrechners eingesetzt, um die Basisform gleichmäßig auf beiden Seiten zu formen.

### Bevel Modifier

Der Bevel Modifier fügt abgerundete Kanten zu den Ecken eines Objekts hinzu. Er erzeugt eine Fase oder eine abgeschrägte Kante, indem er die Kanten des Modells verändert und ihnen eine gewisse Breite verleiht. Dies hilft, harte Kanten zu glätten und das Aussehen des Modells zu verbessern. Der Bevel Modifier wird oft verwendet, um realistischere und ansprechendere Oberflächen zu erzeugen, insbesondere bei der Modellierung von mechanischen Teilen, Möbeln und Architektur.

## 4.2.10 Modellierung von Gegenständen für das Projekt

Dieser Abschnitt beschreibt die für das Projekt in Blender modellierten Gegenstände sowie die dabei aufgetretenen Herausforderungen.

Zu Beginn des Projekts musste eine wichtige Entscheidung getroffen werden, nämlich wie viele Objekte modelliert werden sollten. Es war wichtig, eine ausgewogene Anzahl zu wählen, die das Spiel nicht überladen, aber dennoch eine Herausforderung für die Spieler darstellen soll. Nach eingehenden Recherchen und internen Abstimmungen einigte sich das Team auf 11 Gegenstände. Diese sollten alltägliche Gegenstände eines Schülers der HTBLuVA darstellen und den Spielern einen Einblick in den Schulalltag bieten.

### 4.2.10.1 Taschenrechner

Um den Modellierungsprozess am Beispiel des Taschenrechners zu veranschaulichen, werden spezifische Schritte und Überlegungen während des Prozesses erläutert. Der Prozess begann mit der Idee, den Taschenrechner als Teil des Projekts zu integrieren. Anschließend wurde der Modellierungsumfang und die Art des Taschenrechners festgelegt. Dabei diente das Casio FX-991 ESPLUS Modell als Vorlage für das zu erstellende Modell. Als Referenzbild wurde nach kurzer Recherche ein geeignetes Bild mit hoher Auflösung aus dem Internet gefunden und ausgewählt.

### Ausgangslage

Im Modellierungsprogramm Blender war das Standardprojekt als Startpunkt geeignet. Es wurde bereits geringfügig angepasst. Die Standardkamera und Lichtquelle, die für das Rendering innerhalb von Blender verwendet werden, wurden entfernt, da sie für das finale Projekt in Unity nicht benötigt werden und unnötige Komplikationen verursachen könnten. Je nach Anforderung des Modells wird eine geeignete Grundform wie beispielsweise ein Würfel, eine Fläche oder ein Zylinder erstellt, um darauf aufbauend mit der eigentlichen Modellierung zu beginnen.

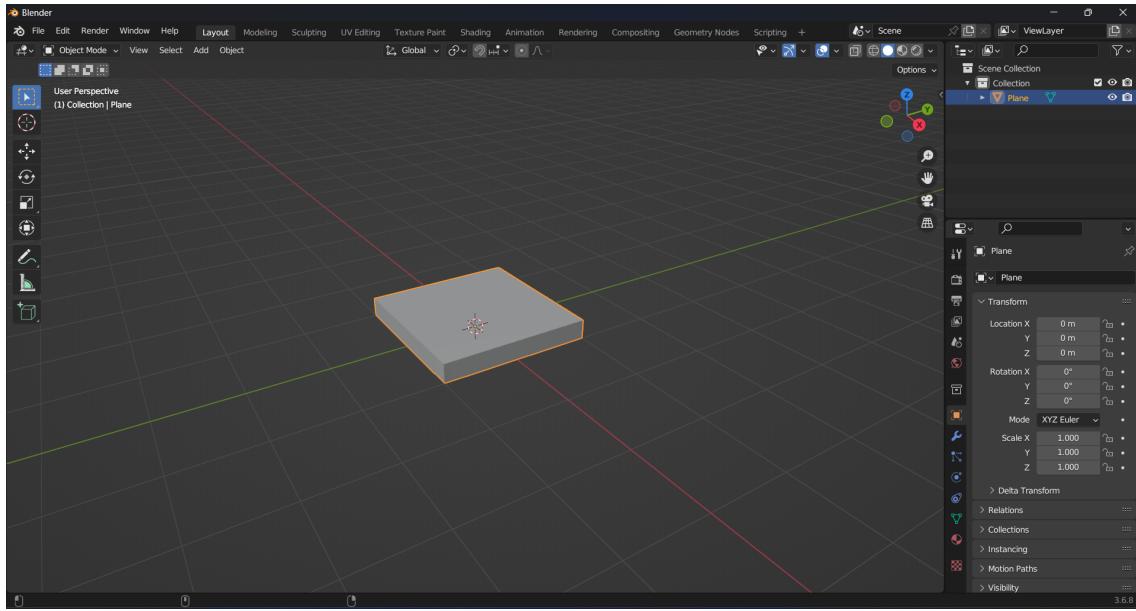


Abbildung 4.3: Ausgangslage innerhalb von Blender

Durch diesen Prozess der Initiierung wurde die Grundlage für die folgenden Schritte der Modellierung gelegt, die in weiteren Abschnitten detaillierter beschrieben werden.

### Erste Schritte nach der Erstellung

Nach Auswahl eines geeigneten Referenzbildes für das Taschenrechnermodell wurde dieses mithilfe des Add-Ons *Images as Planes* in Blender eingefügt. Das Bild dient als Leitfaden für die Modellierung und wurde unterhalb des bereits vorhandenen Objekts platziert. Um eventuelle Unregelmäßigkeiten im Design zu vermeiden, wurde der Mirror Modifier aufgrund der symmetrischen Form des Taschenrechners angewendet. Durch diese Maßnahme werden alle Modellierungsaktionen, die auf der linken Seite durchgeführt werden, automatisch auf die rechte Seite gespiegelt. Dadurch wird die Effizienz und Genauigkeit des Modellierungsprozesses erhöht.

### Extrahieren als Modellierungswerkzeug

Extrahieren<sup>19</sup> ist ein Werkzeug im Edit-Modus von Blender. Es dient dazu, ausgewählte Punkte zu duplizieren und zu verschieben, während die ursprünglichen Punkte der Bearbeitungslinie erhalten bleiben. Für die Verwendung des Extrahieren-Werkzeugs ist ein Verständnis der grundlegenden Konzepte des Edit-Modus in Blender sowie eine präzise Auswahl der zu extrahierenden Punkte erforderlich, um die gewünschten Modifikationen am Modell vorzunehmen. Während des Modellierungsprozesses ist es von entscheidender Bedeutung, unbeabsichtigte Extraktionen zu vermeiden, da diese dazu führen können, dass duplizierte Vertices über bereits vorhandenen liegen. Dies kann nicht nur die weitere Bearbeitung des Modells beeinträchtigen, sondern auch zu einer unnötigen Zunahme der Anzahl der Vertices führen.

---

<sup>19</sup>Blender Extrahieren

### Weiterführung beim Taschenrechnermodell

Im Anschluss begann die Modellierung durch Extrudieren eines Eckpunkts der Fläche, um grob der Form des Referenzbildes zu folgen.

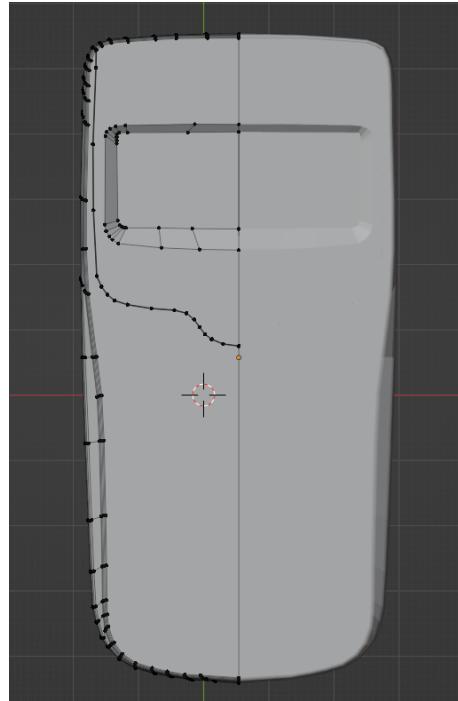


Abbildung 4.4: Taschenrechner mit Buttons in Blender

Nach Abschluss dieser groben Modellierungsschritte entstand das Grundgerüst des Taschenrechnermodells, wie in Abbildung 4.4 dargestellt. Die verschiedenen Eckpunkte (Vertices) werden im Edit-Modus von Blender als schwarze Punkte dargestellt. Durch den Mirror Modifier wird die Bearbeitung auf der linken Seite des Modells automatisch auf die rechte Seite gespiegelt, was eine symmetrische Form gewährleistet. Um dem Modell mehr Tiefe zu verleihen, wurde Extrudieren verwendet, um aus der ebenen Fläche eine dreidimensionale Struktur zu formen.

Als nächster Schritt steht die Modellierung der Knöpfe und anderer Funktionen des Taschenrechners an, um das Modell weiter zu verfeinern und seinem realen Gegenstück näherzukommen. Dazu wird der *Array-Modifier* verwendet, da dieser es ermöglicht, ein Button-Modell zu entwerfen und dieses dann zu vervielfältigen. Dadurch wird die Modellierungszeit enorm reduziert und die gleichmäßige Platzierung der Buttons verbessert.

#### Einschub Array-Modifier

Der Array-Modifier in Blender ermöglicht die Erstellung einer Reihe von Kopien eines Basisobjekts. Jede Kopie wird dabei auf eine vordefinierte Weise von der vorherigen Kopie versetzt. Der Modifier bietet eine Vielzahl von Optionen zur Anpassung der Positionierung und Ausrichtung der Kopien. Die grundlegenden Funktionen des Array-Modifiers umfassen die Möglichkeit, Vertices in benachbarten Kopien zusammenzuführen, insbesondere wenn sie nahe beieinander liegen. Dadurch können nahtlose und konsistente Verbindungen zwischen den einzelnen Kopien hergestellt werden. Eine wichtige Option des Array-Modifiers ist der sogenannte *Fit Type*, der angibt, wie das Objekt dupliziert werden soll. Dies kann

entlang einer Kurve, mit einer passenden Länge oder einer festgelegten Anzahl von Kopien erfolgen. Darüber hinaus kann eine relative oder feste Verschiebung entlang der x-, y- oder z-Achse definiert werden, um die Positionierung der Kopien weiter anzupassen.

### Weiterführung beim Taschenrechnermodell

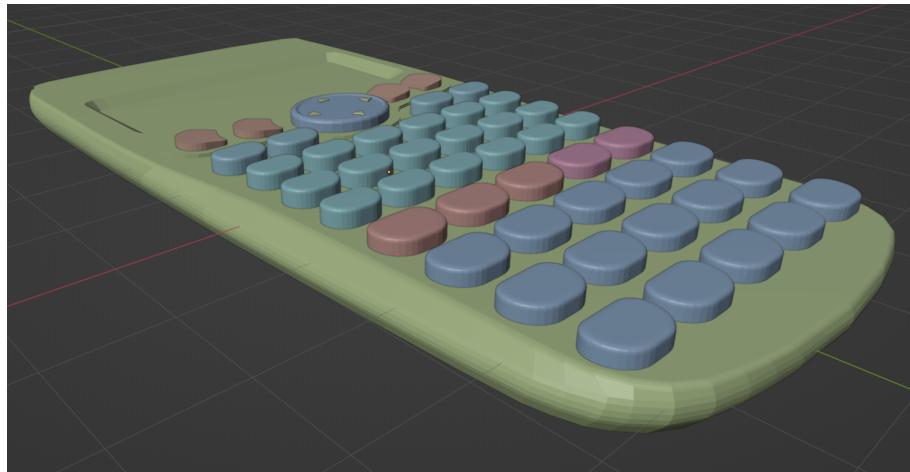


Abbildung 4.5: Taschenrechner mit Buttons in Blender

Abbildung 4.5 zeigt das Taschenrechnermodell in Blender mit den neu modellierten Buttons. Die dargestellten Farben dienen lediglich als visuelle Hilfestellung im Blender-Editor und stellen keine inhärenten Eigenschaften des Modells dar. Jedes neu erstellte Objekt wird automatisch mit einer zufälligen Farbe versehen, um eine einfachere Unterscheidung innerhalb der Szene zu ermöglichen.

Einige der Buttons im Modell wurden mithilfe des Array-Modifiers vervielfältigt. Dies gilt insbesondere für Buttons, die im Referenzbild die gleiche Größe, Funktion und Farbe aufweisen. Durch diese Technik ist es möglich, wiederkehrende Elemente effizient zu modellieren, indem eine einzelne Vorlage kopiert und entsprechend der gewünschten Anordnung angepasst wird.

Der aktuelle Stand des Modells lässt darauf schließen, dass die Modellierung größtenteils abgeschlossen ist. Der nächste Schritt besteht darin, dem Modell Farben oder Texturen hinzuzufügen, um ein realistischeres Aussehen zu erzielen. Dies kann durch Anwendung von Materialien und Texturen im Renderprozess erreicht werden, um dem Modell visuelle Tiefe und Detailtreue zu verleihen.

Die Fortführung des Modellierungsprozesses umfasst somit die Umsetzung der texturierten Oberfläche sowie mögliche Feinanpassungen, um das Modell weiter zu verbessern und an die Referenzvorlage anzupassen.

### Hierarchie des fertiggestellten Modells

In Blender wird die Hierarchie eines Modells als Baumstruktur dargestellt. Dabei ermöglichen verschiedene Unterteilungen und Kategorien eine übersichtliche Organisation. Die Struktur wird in Abbildung 4.7 veranschaulicht. Das weiße Box-Symbol steht für eine Collection (Ordner).

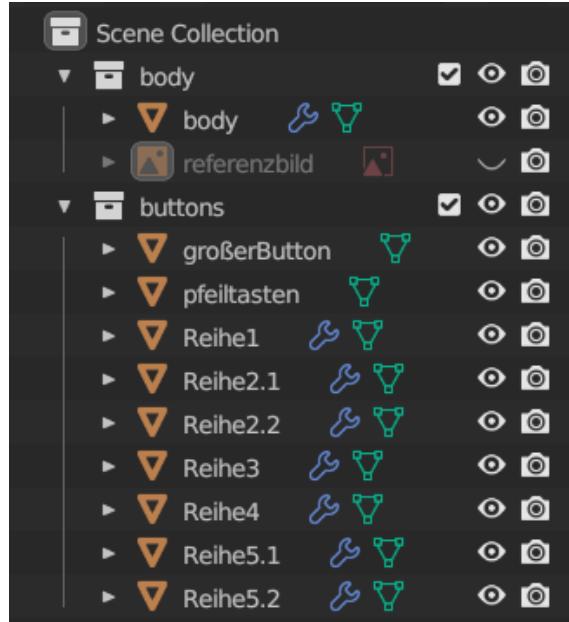


Abbildung 4.6: Hierarchie des fertigen Taschenrechnermodells

- Die **Scene Collection** ist die oberste Ebene, die das gesamte Modell enthält.
- Die **Body Collection** enthält das Hauptmodell des Taschenrechners, also die grundlegende Form.
- Die **Buttons Collection** umfasst alle Buttons, die auf dem Taschenrechner abgebildet sind.
- Die untergeordnete Struktur mit den orangefarbenen Dreiecken repräsentiert die eigentlichen Objekte innerhalb der jeweiligen Collections.
- Das **Referenzbild** ist ein importiertes Bild, das beispielsweise als Vorlage für das Modellieren verwendet wird.

In Abbildung 4.7 ist zu erkennen, dass ein Objekt ausgegraunt ist. Dies bedeutet in dem Fall, dass das Objekt ausgeblendet wurde, da das Hauptmodell bereits fertiggestellt wurde und das Referenzbild nicht mehr benötigt wird. Das Referenzbild kann jedoch bei Bedarf aktiviert werden, beispielsweise zu Hilfe- oder Veranschaulichungszwecken.

Die hierarchische Darstellung der Elemente erleichtert die Organisation und Bearbeitung des Modells. Eine klare Strukturierung der verschiedenen Komponenten ermöglicht eine effiziente Arbeitsweise und hilft, den Überblick über die einzelnen Teile zu behalten.

### Textur und Farbe des Taschenrechnermodells

Für die Texturierung und Farbgebung des Modells diente das Referenzbild zur Orientierung. Das *Eye-Dropper*-Werkzeug in Blender ermöglichte es, Farben gezielt aus dem Referenzbild auszuwählen und auf das entsprechende Modell anzuwenden. Durch Klicken auf einen bestimmten Bereich des Referenzbildes wurde die Farbe erfasst und dann auf den entsprechenden Bereich des Modells übertragen.

Dieser Prozess wurde für jedes Element des Modells durchgeführt, um eine genaue Anpassung an die Farben und Texturen des Referenzbildes zu erreichen. Die Verwendung des *Eye-Dropper*-Werkzeugs ermöglichte eine präzise und konsistente Umsetzung der Farbgebung und trug dazu bei, dass das Modell möglichst authentisch und realitätsnah aussieht. Um noch bessere und realitätsnähtere Textur zu gewährleisten, wurde ein Vorgang Namens

*UV-Mapping* durchgeführt.

### UV-Mapping

*UV-Mapping* ist ein wichtiger Schritt im Texturierungsprozess von 3D-Modellen. Dabei werden Texturen auf dreidimensionale Objekte projiziert. Die Bezeichnungen  $U$  und  $V$  stehen dabei für die beiden Koordinatenachsen im zweidimensionalen Raum. In Blender gibt es dafür einen eigenen Editor mit einer spezifischen Benutzeroberfläche.

Während der Modellierung des Taschenrechnermodells wurde *UV-Mapping* genutzt, um Texturen detailgetreu von einem Referenzbild auf die einzelnen Knöpfe und Tasten des Modells zu übertragen. Dieser Prozess ermöglicht es, die Beschriftungen und andere Details automatisch von der Textur abzuleiten, anstatt sie manuell für jedes Element erstellen zu müssen.

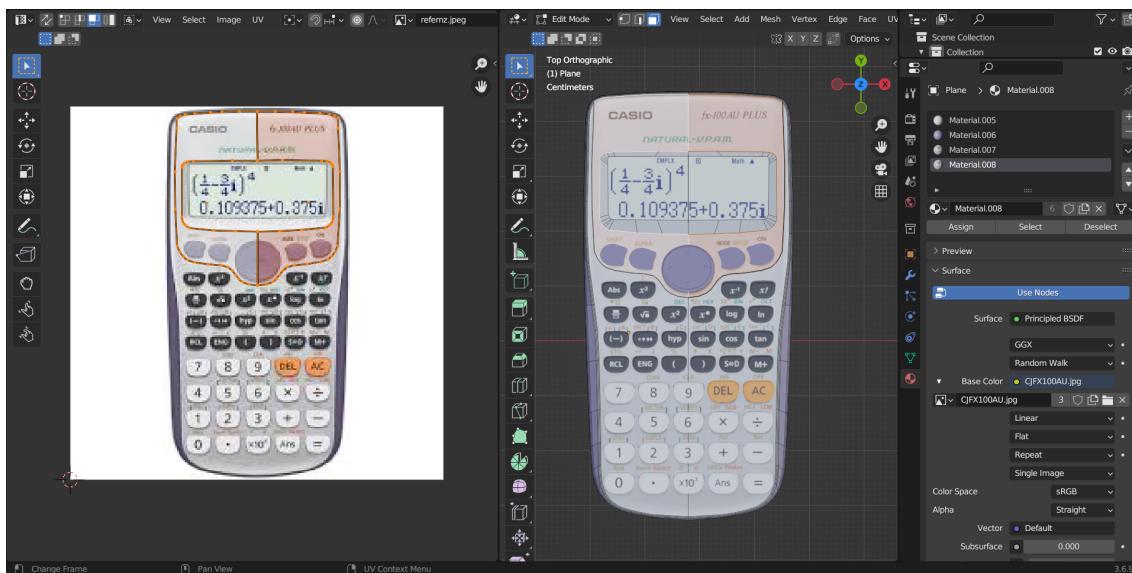


Abbildung 4.7: Ansicht des UV-Editing Editors in Blender

Die Abbildung zeigt den *UV-Editing-Editor* in Blender. Mit diesem Editor können die UV-Maps der 3D-Modelle bearbeitet werden, indem die 3D-Geometrie auf eine zweidimensionale Ebene projiziert wird, um die Positionierung der Texturen zu steuern.

Als Beispiel wurde der obere Teil des Taschenrechners rund um den Bildschirm ausgewählt. Zunächst wird dem Bereich ein neues Material zugewiesen, das die Basisfarbe des Referenzbildes enthält. Anschließend wird das Referenzbild als Texturhintergrund im Editor ausgewählt, um es anzuzeigen.

Der nächste Schritt besteht darin, die ausgewählten Flächen des Modells aufzuteilen, denn die Erstauswahl in Blender legt die einzelnen Flächen übereinander. Dies geschieht durch Auswahl der Flächen im Editor und Anwendung der *UV-Unwrap*-Funktion in Blender.

Nach dem *UV-Unwrappen* werden die Flächen im Editor positioniert, um die Texturen und Beschriftungen korrekt auf das Modell zu projizieren, ohne dass sie abgeschnitten oder verzerrt wirken. Dieser Prozess ist zeitaufwendig, da er für jedes einzelne Element des Modells wiederholt werden muss. Dabei muss auch auf korrekte Skalierungen geachtet werden, um eine einheitliche Darstellung zu gewährleisten.

*UV-Mapping* ist somit ein entscheidender Schritt bei der Erstellung realistischer 3D-Modelle, da es eine präzise Texturierung und Gestaltung ermöglicht.

#### 4.2.10.2 Restliche Modelle



##### Ping-Paket-Modell

Das Ping-Paket-Modell wurde speziell für didaktische Zwecke auf das erste Anwendungsszenario konzipiert, um den Benutzern das grundlegende Konzept eines Daten-Pings zu veranschaulichen. Es dient dazu, die Übertragung von Daten zwischen zwei Endpunkten, beispielsweise Laptops, zu visualisieren. Das Modell besteht im Wesentlichen aus einem simplen Quader, ergänzt durch Details wie Paketklebeband, um eine realistische Darstellung zu erreichen. Bei der Texturierung wurde ein Amazon-Paket als Referenz verwendet.

##### Laptop-Modell

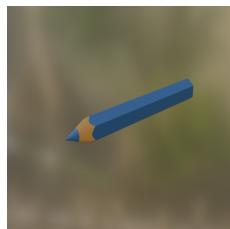
Der Laptop stellt einen unverzichtbaren Gegenstand im zweiten Anwendungsszenario dar und ist ein essentielles Werkzeug für Schüler ab der 3. Klasse. Bei der Modellierung wurden zusätzliche Details wie USB-Ports und der Laptopständer berücksichtigt. Um den Aufwand zu minimieren, wurde der Laptop im geschlossenen Zustand modelliert, wodurch die Komplexität von Tastatur und Bildschirm vermieden wurde. Die Referenz für das Modell lieferte ein reales Laptop-Exemplar eines Projektmitglieds.

##### Router-Modell

Die Modellierung des Routers war eine iterative Aufgabe, die mehrere Versionen erforderte, um ein realistisches und ansprechendes Modell zu erzielen. Das finale Modell repräsentiert einen Gaming-Router mit vielen Anschlüssen und Bedienelementen. Aufgrund der Vielzahl an Details war die Modellierung sehr anspruchsvoll und zeitaufwendig.

##### Maus-Modell

Die Modellierung der Maus war eine Standardaufgabe, bei der der Mirror Modifier effektiv eingesetzt wurde, um die symmetrische Natur des Objekts zu betonen. Eine einfache Büromaus diente als Referenz für das Modell.



##### Notizblock-Modell

Der Block ist ein wichtiger Gegenstand im schulischen Umfeld bis zur 3. Klasse, wenn noch keine Laptops verwendet werden. Während der Designphase ermöglichte der Array

Modifier eine einfache Gestaltung der spiralförmigen Halterung der Blätter.

### **Stift-Modell**

Der Stift wurde einfach modelliert und erhielt eine unkomplizierte Texturierung mit einfachen Farben. Ein Buntstift diente als Referenz für das Modell.

### **Kopfhörer-Modell**

Die Modellierung der Kopfhörer erforderte aufgrund ihrer geschwungenen Form und der texturierten Oberfläche besondere Aufmerksamkeit. Das Modell durchlief zwei Phasen: zunächst eine Annäherung an das Referenzbild eines Gaming-Kopfhörers und dann eine Verfeinerung, einschließlich einer aufwendigen Lederstrukturtextur.

### **Dose-Modell**

Die Modellierung der Dose eines Energydrinks begann mit einem einfachen Zylinder, der dann anhand eines Referenzbildes geformt wurde. Besonderes Augenmerk wurde auf die Textur gelegt, wobei ein bekanntes Produktlabel als Referenz diente.



### USB-Stick-Modell

Der USB-Stick wurde mit einer integrierten Halterung für zusätzliche Komplexität modelliert. Die Texturierung betonte einen metallischen Effekt, insbesondere für die Halterung und das vordere Teil des USB-Sticks.

### Stromverteiler-Modell

Der Stromverteiler wurde mit fünf Steckplätzen und einem Ein/Aus-Schalter modelliert, wobei der Fokus auf einfacher Funktionalität lag. Die Texturierung orientierte sich an einfachen weißen Steckerleisten.

### Handy-Modell

Das Handymodell orientierte sich an modernen iPhones von Apple und erforderte aufgrund seiner zahlreichen Details wie Kameras, Tasten und Mikrofoneinlässe eine sorgfältige Modellierung und Texturierung, um einen realistischen Eindruck zu erzeugen.

## 4.3 Hauptmenü

Im folgenden Abschnitt werden die Entwicklungsphasen beim Entwurf des Menüs erläutert. Es wird erklärt, welche Probleme dabei auftreten und worauf bei der Benutzeroberfläche (UI) und der Benutzererfahrung (UX) geachtet werden muss. Es wird aufgeteilt in die Konzeption und Planung der Benutzeroberfläche bis hin zur finalen Umsetzung. Die iterative Entwicklung des Menüs durchlief mehrere Phasen, bei denen Entscheidungen zum Design des Layouts, des Farbschemas und der Interaktionselemente getroffen wurden. Während der Entwicklung wurden regelmäßig Überprüfungen und Anpassungen durchgeführt, um sicherzustellen, dass das Menü den gestellten Anforderungen entspricht und eine optimale Benutzererfahrung bietet. Dabei wurden auch Rückmeldungen und Tests von Nutzern in die Iterationsschleife einbezogen, um mögliche Verbesserungspotenziale zu identifizieren und zu berücksichtigen. Schließlich wurde das Menü in seiner endgültigen Version implementiert. Dabei wurde besonderes Augenmerk auf Funktionalität, Benutzerfreundlichkeit und Ästhetik gelegt. Durch diesen iterativen Entwicklungsprozess wurde sichergestellt, dass das Menü den Anforderungen entspricht und einen positiven Beitrag zur Gesamterfahrung des Spiels leistet.

### 4.3.1 Erstentwurf

Ursprünglich war geplant, das UI/UX-System mit einem sogenannten **Nahmenü** zu realisieren. Dieses folgt dem Nutzer in seiner Nähe und besteht in unserem Fall, aus drei primären Schaltflächen sowie einem Pin-Button. Das Nahmenü ist etwa auf Hüfthöhe des Benutzers positioniert und zeichnet sich durch eine vereinfachte Struktur aus, die eine intuitive Bedienung gewährleistet. Die Gestaltung des Menüs hat zum Ziel, Verwirrung zu

vermeiden und dem Nutzer ohne umfassende Vorabinformationen klar zu machen, wie er die Anwendung verwenden kann.

## Nahmenü

Innerhalb von Unity bezeichnet der Begriff *Nahmenü* eine vordefinierte Konstruktion, die mit bestimmten Skripten ausgestattet ist, um sicherzustellen, dass das Menü dem Benutzer in alle Richtungen folgt. Es werden von Unity bereitgestellte Skripte verwendet, die es ermöglichen, das Menü dynamisch an die Position des Nutzers anzupassen.

Im Erstentwurf wurde das gesamte Menü ausschließlich innerhalb der Unity-Umgebung konstruiert. Dabei wurden sämtliche vorhandenen Schaltflächen und Funktionen aus den nativen Ressourcen von Unity zusammengesetzt. Dieser Ansatz führte zu einer konsistenten visuellen Gestaltung des Menüs. Allerdings waren die Möglichkeiten zur kreativen Gestaltung und Anpassung im Rahmen unseres individuellen Projekts stark begrenzt. Durch die ausschließliche Verwendung der internen Ressourcen von Unity wurden gewisse Einschränkungen hinsichtlich der Flexibilität und Individualisierungsmöglichkeiten des Menüs in Kauf genommen.

Obwohl diese Herangehensweise zu einem homogenen Erscheinungsbild des Menüs führte, war es wichtig, eine Balance zwischen visueller Einheitlichkeit und der Notwendigkeit individueller Anpassungsmöglichkeiten zu finden. Diese Herausforderung verdeutlicht die Bedeutung einer flexiblen und erweiterbaren Architektur für die Benutzeroberfläche, um den Anforderungen und Zielen des Projekts gerecht zu werden.



Abbildung 4.8: Darstellung des Erstentwurfes im Unity Editor

Die Abbildung 4.8 zeigt den ersten Entwurf des Menüs, das eine wichtige Schnittstelle für den Nutzer darstellt, um zwischen verschiedenen Szenarien zu navigieren. Die Struktur des Menüs bleibt unabhängig vom jeweiligen Spiellevel konstant, was zur Simplifizierung und Klarheit beiträgt. Die primären Schaltflächen auf der linken Seite dienen der Initiierung des Ladevorgangs für die verschiedenen Spielszenarien. Hervorzuheben ist der Pin-Button in Kombination mit dem darunterliegenden weißen Balken. Diese beiden Funktionen gemeinsam ermöglichen es dem Benutzer, das Menü an gewünschten Positionen zu verankern und bietet somit Flexibilität bei der Positionierung entsprechend individueller Präferenzen. Ein solcher Einsatz ist von Vorteil, wenn externe Objekte die Nutzbarkeit des Menüs beeinträchtigen könnten, wie zum Beispiel im Szenario des Platznehmens an einem Tisch, wo das Menü ungünstigerweise mit dem Tisch kollidieren könnte und somit die Nutzererfahrung beeinträchtigt werden würde.

#### 4.3.1.1 Probleme beim Erstentwurf

##### Problem 1 - Fehlende Dokumentation

Das Hauptziel bei dem Konzept des ersten Menüs, bestand darin, ein benutzerfreundliches Menü zu gestalten, das alle erforderlichen Funktionen enthält und dennoch übersichtlich ist. Nach mehreren Iterationen und Tests mit Testpersonen, die nicht mit dem Projekt vertraut waren, stellte sich heraus, dass der ursprüngliche Entwurf erhebliche Mängel bei der Dokumentation und Erklärung der verschiedenen Anwendungsszenarien und ihrer jeweiligen Aufgaben aufweist. Es wurde bemängelt, dass Benutzer lediglich zwischen den einzelnen Leveln wechseln können, ohne klare Informationen darüber zu erhalten, worum es in den einzelnen Leveln geht und welche Aufgaben diese beinhalten.

Diese Feststellung verdeutlicht eine wesentliche Lücke in der Benutzerführung und information innerhalb des Menüsystems. Die unzureichende Dokumentation der Level und ihrer Ziele führt zu einer fehlenden Orientierung für die Benutzer und kann sich negativ auf ihre Erfahrung auswirken. Das Menü sollte nicht nur als Navigationswerkzeug dienen, sondern auch als Informationsquelle, die dem Benutzer eine klare Vorstellung über den Spielverlauf und die zu erreichen Ziele vermittelt.

Die Identifizierung dieses Problems betont die Wichtigkeit einer umfassenden Benutzerbeobachtung und -evaluation während des Designprozesses. Dadurch wird sichergestellt, dass das entwickelte Benutzeroberflächensystem den Bedürfnissen und Erwartungen der Zielgruppe entspricht. Eine zielgerichtete Überarbeitung des Menüs ist erforderlich, um die fehlende Dokumentation der Anwendungsszenarien und ihrer Aufgaben zu adressieren und somit die Benutzererfahrung zu verbessern.

Bei Testdurchführungen ohne Entwickler aus dem Projektteam kam es wiederholt zu Unklarheiten bei den Benutzern, insbesondere während des ersten Durchlaufs, bezüglich der zugewiesenen Aufgaben. Während der Nutzung der HoloLens-Anwendung gab es vermehrt Anfragen seitens der Benutzer an das Projektteam aufgrund von Unklarheiten. Dies lässt vermuten, dass die Anwesenheit von Entwicklern im Projektteam einen signifikanten Einfluss auf die Benutzererfahrung und Effektivität der HoloLens-Anwendung hat.

##### Problem 2 - Falsche Wahl des Menütyps

Während des Testprozesses stellte sich heraus, dass das Navigationsmenü oft eine Behinderung darstellte, insbesondere für Entwickler, die Funktionen wiederholt testeten und Anwendungsszenarien neu luden. Das ständige Verschieben des Menüs zur Seite erwies sich als zeitraubend und störte den Arbeitsfluss erheblich. Die ursprüngliche Intention, das Navigationsmenü zur Erleichterung der Interaktion zwischen Benutzer und Spielumgebung einzuführen, erwies sich somit als kontraproduktiv.

Es ist wichtig, die Auswirkungen von Benutzeroberflächenelementen auf den Entwicklungsprozess zu berücksichtigen, um ein reibungsloses Testen und Experimentieren zu gewährleisten. Eine effiziente und produktive Arbeitsweise des Entwicklerteams hängt davon ab. Die Notwendigkeit einer kontinuierlichen Evaluation und Optimierung von UI/UX-Komponenten während des gesamten Entwicklungszyklus wird durch dieses Problem verdeutlicht.

Dieses Problem verdeutlicht, dass der Erstentwurf und die ursprüngliche Konzeption hauptsächlich auf die Innovation und Neuheit des Menüdesigns abzielten, anstatt den Fokus auf die funktionale Nützlichkeit zu legen. Der einfache und kompakte Menütyp erwies sich als unzureichend, um eine umfassende und übersichtliche Dokumentation der Anwendungsszenarien zu integrieren. Folglich war eine Migration zu einem alternativen Menüansatz erforderlich. Diese Beobachtung verdeutlicht die Bedeutung einer ausgewogene-

nen Gestaltung von Benutzeroberflächen in Mixed-Reality-Systemen. Es ist wichtig, dass nicht nur ästhetische Innovationen verfolgt werden, sondern auch die tatsächliche Benutzererfahrung und -effizienz in Betracht gezogen wird. Die Reflexion über diesen Prozess bietet Einsichten in die iterative Entwicklung von Benutzerschnittstellen und die Anpassung an die Anforderungen und Rückmeldungen der Benutzer.

### 4.3.2 Finalversion des Menüs

Die endgültige Version des Menüs wurde entwickelt, um die Mängel des ursprünglichen Entwurfs anzugehen und idealerweise vollständig zu beheben. Dieses neue Konzept wurde durch eine umfassende Analyse populärer AR/VR-Spiele wie zum Beispiel *Beat Saber* geleitet, um die Gründe für die verbesserte Benutzererfahrung in diesen Kontexten zu untersuchen. Dabei wurden spezifische Merkmale und Designentscheidungen identifiziert, die einen signifikanten Einfluss auf die Zufriedenheit der Anwender haben.

Basierend auf vorangegangenen Recherchen wurde entschieden, das neue Menü in zwei klar definierte Bereiche zu unterteilen. Dies gewährleistet eine deutliche Unterscheidung des Hauptmenüs sowie eine klare Orientierung bezüglich der Navigation. Insbesondere innerhalb der Anwendungsszenarien wurde ein minimalistisches und unaufdringliches Menü implementiert, um den Fokus der Benutzer uneingeschränkt auf die auszuführenden Tätigkeiten zu lenken. Diese Designstrategie hat zum Ziel, die kognitive Belastung der Benutzer zu reduzieren und eine nahtlose Integration der Menüfunktionalitäten in den jeweiligen Nutzungskontext zu ermöglichen.

Das Ergebnis ist ein ansprechendes und benutzerorientiertes Menükonzept, das die Gesamterfahrung des Spiels verbessert und den Erwartungen der Zielgruppe gerecht wird.

#### 4.3.2.1 Finalversion Menü - Hauptmenü

Auf dieser Grundlage wurde das Design des Menüsystems vollständig überarbeitet, wobei der erste Schritt bereits die vollständige Entfernung der Idee des Nahmenüs beinhaltete.

Anstelle des Nahmenüs wurden vorgefertigte Objekte in das Hauptmenü integriert, welche zuvor mit Blender modelliert wurden. Die Entscheidung, vorgefertigte Objekte aus Blender zu importieren, bietet eine breite Palette an Gestaltungsmöglichkeiten und ermöglicht es, das Menü visuell ansprechend und einladend zu gestalten. Außerdem trägt die Integration dieser Objekte dazu bei, das Menü intuitiver und zugänglicher zu machen, da sie dem Benutzer klare Anhaltspunkte und Handlungsmöglichkeiten bieten.

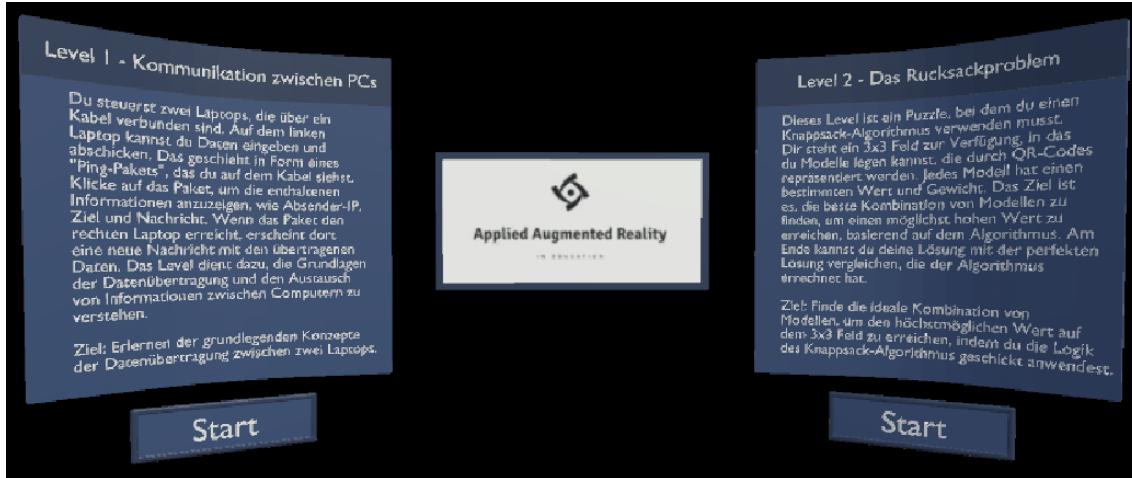


Abbildung 4.9: Darstellung des finalen Menüs im Unity Editor

Die in der Abbildung 4.9 dargestellten Objekte bestehen aus zwei separaten Tafeln, die durch unser Diplomarbeitslogo voneinander getrennt sind. Jede Tafel enthält den Titel des Anwendungsszenarios sowie einen vorgefertigten Startbutton. In Unity wurde ein unsichtbarer, aber dennoch klickbarer Button mit exakt denselben Dimensionen wie das Modell des Startbuttons an der entsprechenden Stelle platziert. Das Skript *SceneChange.cs*, welches für die Szenenwechsel-Funktionalität verantwortlich ist, ist diesem Button angehängt.

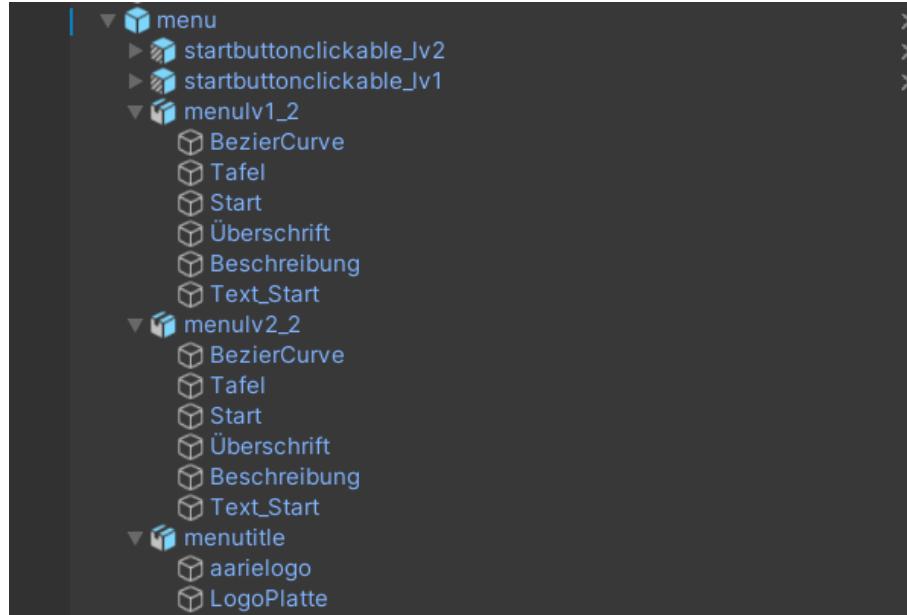


Abbildung 4.10: Darstellung der Hierarchie des finalen Menüs im Unity Editor

In der Abbildung 4.10 ist die Hierarchie des Hauptmenüs im Unity Editor zu sehen. Dem Prefab **menu** sind dabei 5 einzelne Objekte zugewiesen. Die Prefabs **startbuttonclickable\_lv1/2** sind die in Unity erstellten unsichtbaren Buttons, die dafür zuständig sind den vormodellierten Buttons eine Funktion zu geben. Darunter zu sehen sind die 3 verschiedenen Assets, welche aus Blender importiert wurden. Diese bestehen jeweils aus den verschiedenen Einzelteilen, welche in Unity als Gameobjects dargestellt werden. Die **BezierCurve** ist eine spezielle Art von Kurve welche in Blender für die leichte Wölbung

der Tafeln genutzt wurde.

#### 4.3.2.2 Finalversion Menü - Anwendungsszenario

Im Gegensatz zum Erstentwurf, der ein einheitliches Menü für alle Anwendungsszenarien vorsah, weist diese Version eine Differenzierung auf. Die Entscheidung, das Menü innerhalb der einzelnen Szenarien anzupassen, wurde getroffen, um die volle Aufmerksamkeit des Benutzers auf das jeweilige Level und die darin ausgeführten Aktivitäten zu lenken. Aus diesem Grund wurde ebenfalls das Nahmenü aus den Szenarien entfernt und durch ein simples Handmenü ersetzt.



Abbildung 4.11: Darstellung des finalen Menüs im Anwendungsszenario im Unity Editor

Wie in der Abbildung 4.11 gezeigt, besteht dieses Menü ausschließlich aus einem Zurück-Button, der den Benutzer zum Hauptmenü zurückführt. Das Handmenü wurde so konzipiert, dass es nur dann sichtbar ist, wenn der Benutzer es benötigt oder aktivieren möchte. Der Zurückbutton wird erst sichtbar, wenn der Benutzer seine linke Hand umdreht und auf die Handfläche schaut.

Diese Funktionalität hat zum Ziel, die Benutzerinteraktion intuitiver und kontextbezogener zu gestalten. Hierfür wird Bewegungserkennung und Blickrichtungserkennung genutzt, um das Handmenü nur dann einzublenden, wenn der Benutzer aktiv nach einem Navigationsmittel sucht. Dadurch wird die visuelle Ablenkung minimiert und die Benutzererfahrung optimiert, indem nur relevante Informationen und Interaktionselemente präsentiert werden, wenn sie benötigt werden.

#### 4.3.2.3 Setzen des Menüs

Beim Erstentwurf gab es keine Probleme beim Wechseln zwischen den Szenen beziehungsweise dem Laden in das Hauptmenü, jedoch muss man in der neuen Version darauf achten das Menü richtig zu setzen. Dafür muss man die Modelle im richtigen Moment und an der richtigen Stelle in der richtige Szene selbstständig instanzieren, deshalb wurde extra für diesen Fall das *MenuSpawn.cs* Skript geschrieben.

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
```

```

4
5 public class SpawnPrefab : MonoBehaviour
6 {
7     public GameObject menu;
8
9     void Start()
10    {
11         SpawnPrefabMenu();
12    }
13
14     void SpawnPrefabMenu()
15    {
16         Vector3 cameraPosition = Camera.main.transform.position;
17         Vector3 spawnPosition = new Vector3(cameraPosition.x, cameraPosition.y,
18         cameraPosition.z + 1f);
19         Instantiate(menu, spawnPosition, Quaternion.identity);
20         menu.SetActive(true);
21    }
21 }
```

Codeabschnitt 4.1: Menueinstanzierung bei Neuladen.

Beim Start der Anwendung wird die Methode *SpawnPrefabMenu()* ausgeführt. Diese Methode ist für die Positionierung des Menüs verantwortlich. Die im Codeabschnitt 4.1 ersichtlichen Positionsvektoren sind einerseits dafür zuständig, die aktuelle Kameraposition zu ermitteln, aber auch eine neue Spawnposition zu setzen, die um 1 auf der z-Achse versetzt ist, so dass das Menü vor dem Benutzer erscheint. Der Funktionsaufruf *Instantiate* ist dafür verantwortlich, dass das Menü an der jeweiligen Spawnposition gesetzt wird. Anschließend wird das zuvor versteckte Menü mit *menu.SetActive(true)* aktiviert, so dass der Benutzer es sehen und mit ihm interagieren kann.

### 4.3.3 Laden der Level

Um den Buttons eine Funktion zuzuweisen, wird ein Skript benötigt, das aktiviert wird, sobald ein Button gedrückt wird. Im vorliegenden Fall ist das Skript *SceneChange.cs* für den Wechsel zwischen den verschiedenen Anwendungsszenarien verantwortlich. Wenn einer der Buttons im Hauptmenü gedrückt wird, wird dieses Skript ausgeführt. Die Spielszene wird im *Unity Inspector* festgelegt. Der Code greift auf die Variable zu, die den Namen der Szene enthält, wie sie zuvor im Inspector benannt wurde.

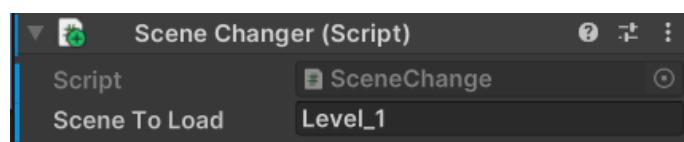


Abbildung 4.12: Darstellung der SceneToLoad Variable, anhand des Level1 Buttons.

Für den linken Button im Hauptmenü ist beispielsweise die Spielszene *Level1* vorgesehen, während für den rechten Button die Szene *Level2* zugewiesen ist. Durch diese Konfiguration im Inspector wird die Flexibilität gewährleistet, Szenen dynamisch anzupassen, ohne dass Änderungen am eigentlichen Skript vorgenommen werden müssen. Dies ermöglicht eine effiziente Verwaltung und Anpassung der Spielinhalte während des Entwicklungsprozesses.

```

1 using UnityEngine;
2 using UnityEngine.SceneManagement;
```

```

3
4 public class SceneChanger : MonoBehaviour
5 {
6     public string sceneToLoad;
7
8     public void ChangeScene()
9     {
10         if(SceneManager.GetActiveScene().name != sceneToLoad)
11         {
12             PlayerPrefs.DeleteAll();
13             SceneManager.LoadScene(sceneToLoad, LoadSceneMode.Single);
14         }
15     }
16 }
```

Codeabschnitt 4.2: Auf Knopfdruck Szene wechseln.

Das vorliegende Skript hat eine vergleichsweise einfache Struktur. Die Variable `public string sceneToLoad` dient als Empfänger für den Namen der zu ladenden Szene, welcher im Unity-Inspector individuell für jede Szene festgelegt werden kann. Um den Szenenwechsel zu initiieren, wird die Methode `ChangeScene()` aufgerufen. Zunächst wird in dieser Methode geprüft, ob die aktive Szene, die durch `SceneManager.GetActiveScene()` ermittelt wird, mit der zu ladenden Szene übereinstimmt. Falls dies nicht der Fall ist, werden alle gespeicherten PlayerPrefs mittels `PlayerPrefs.DeleteAll()` gelöscht. Dieser Schritt ist von entscheidender Bedeutung, um potenzielle Komplikationen im Zusammenhang mit dem HoloLens-Cache während des Szenenwechsels zu vermeiden.

Nachdem die Prefabs bereinigt wurden, wird die definierte Szene mit der Methode `SceneManager.LoadScene()` geladen. Dabei wird die Ladeoption `LoadSceneMode.Single` verwendet, um sicherzustellen, dass nur die angegebene Szene aktiv geladen wird und keine anderen Szenen im Hintergrund verbleiben. Diese präzise Steuerung des Ladevorgangs zielt darauf ab, die Performance und Stabilität der Anwendung auf der HoloLens-Plattform zu optimieren und potenzielle Konflikte zwischen Szeneninhalten zu vermeiden.

#### 4.3.4 UI/UX

Die Gestaltung der Benutzeroberfläche und die damit verbundene Benutzererfahrung sind wesentliche Aspekte bei der Konzeption und Entwicklung von Software. Eine bewährte Strategie besteht darin, sich an populären und gleichgerichteten Anwendungen zu orientieren, insbesondere im Bereich der Augmented- oder Virtual Reality. Es wird darauf geachtet, komplexe und verwirrende Menüstrukturen zu vermeiden und eine konsistente Farbpalette über die gesamte Benutzeroberfläche hinweg zu verwenden.

Diese Herangehensweise basiert auf der Erkenntnis, dass erfolgreiche Anwendungen in ähnlichen Domänen oft bewährte Designprinzipien und Interaktionsmuster aufweisen, die sich positiv auf die Benutzerakzeptanz und -zufriedenheit auswirken können. Durch die Anpassung an gängige Standards und Trends in der Branche kann das Risiko von Benutzerirritationen und -ablehnungen minimiert werden. Gleichzeitig wird eine vertraute und angenehme Benutzererfahrung gefördert.

Es ist wichtig zu betonen, dass eine solche Inspiration von bestehenden Anwendungen nicht als direkte Kopie verstanden werden sollte, sondern vielmehr als Quelle für Anregungen und bewährte Praktiken, die in einem eigenständigen und angepassten Kontext implementiert werden können. Auf diese Weise kann die Effektivität und Attraktivität der Benutzeroberfläche optimiert werden, um die Bedürfnisse und Erwartungen der Zielgruppe bestmöglich zu erfüllen.

## 4.4 Nachrichtenaustausch Anwendungsscenario

In diesem Szenario wird das IT-Grundprinzip des Nachrichtenaustausches illustriert. Zwei PCs sind in einer realen Umgebung durch ein rotes Kabel miteinander verbunden, wobei auf jedem PC dieselbe Website geöffnet ist. Der Benutzer trägt eine HoloLens 2 und erhält die Aufforderung, das rote Kabel zu betrachten. Nach einem Countdown wird ein Foto der Umgebung aufgenommen und anschließend verarbeitet. Dabei werden Raycasts eingesetzt, um die exakte Position des roten Kabels in der physischen Welt zu bestimmen. Sobald die Position des Kabels erkannt wurde, kann der Benutzer über die Website auf den beiden PCs eine Nachricht senden. Diese Nachricht wird dann in Form eines Pakets für den Hololens 2 Benutzer visualisiert. Der Benutzer hat außerdem die Möglichkeit, auf das Paket zu drücken, um die Informationen wie Absender und Nachricht einzusehen.

(→)  
Schoditsch

### 4.4.1 Aufbau von Nachrichtenaustausch Anwendungsscenario

(→) Scho-  
ditsch

In diesem Abschnitt wird detailliert beschrieben, wie die Unity-Szene aufgebaut ist, welche Rollen und Funktionen die einzelnen GameObjects haben und wie sie miteinander interagieren, um die gewünschte Funktionalität der Szene zu erreichen.

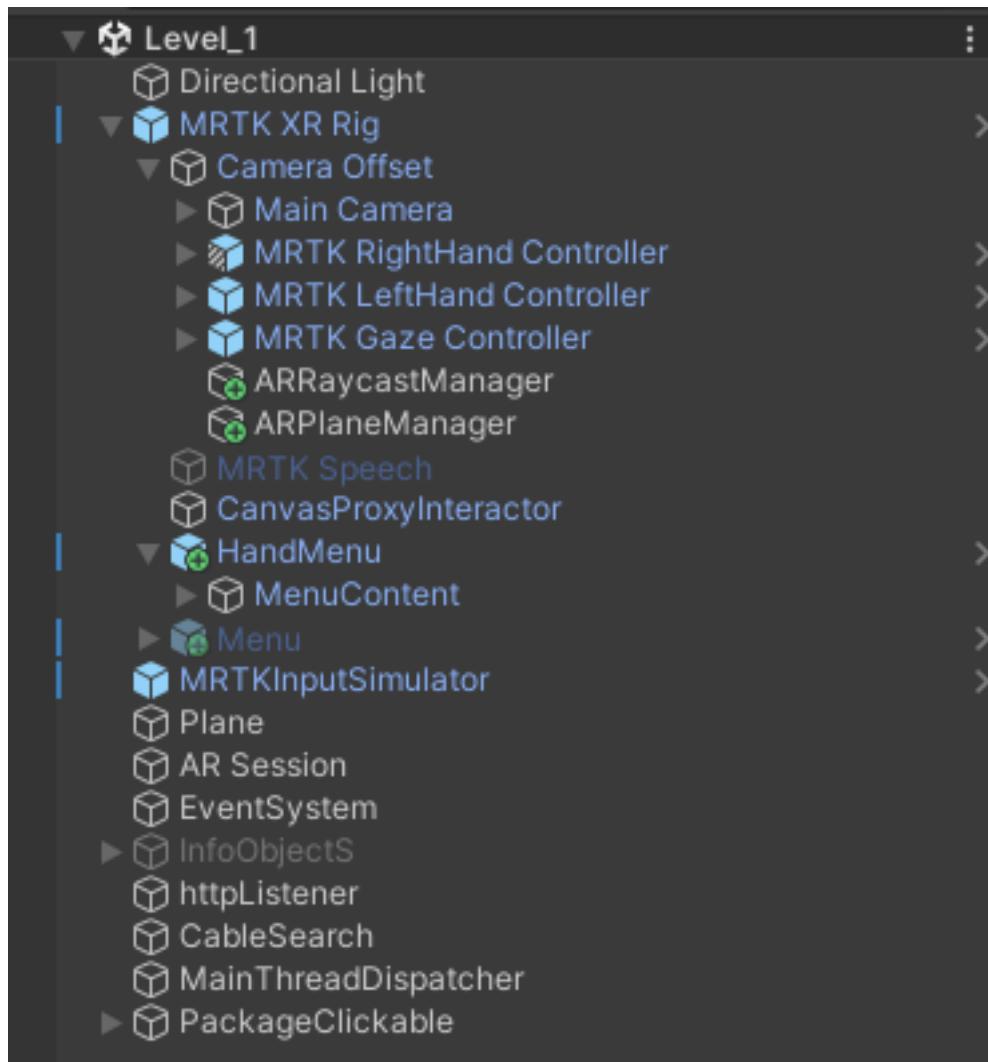


Abbildung 4.13: Hierarchie des Nachrichtenaustausch Levels im Unity Editor.

- **Level-1:** Die Scene in der alle Unity-Objekte enthalten sind.
- **UserInfo:** Gameobject welches den Hololens Benutzer einen Countdown angibt und von welchem aus daraufhin alle anderen Scripts ausgeführt werden.
- **LoadingCircle:** Während gewisse Prozesse von der Maschine durchgeführt werden welche länger brauchen könnten, wird dieses *Gameobjekt* angezeigt um den Benutzer zu informieren, dass das Programm Berechnungen ausführt. Das Objekt beinhaltet ein *Canvas* mit dem *User Interface*.
- **Plane:** Ist ein *GameObject* welches den Code beinhaltet und ausführt. Kann auch zum Debugging genutzt werden, da es die erkannten roten Pixel visuell darstellt.
- **InfoObjectS:** Beinhaltet den Text welcher angezeigt wird sobald auf das erstellte digitale Paket gedrückt wird wird.
- **httpListener:** Verarbeitet die eingehenden http anfragen der Website.
- **CableSearch:** Beinhaltet den Start der Unity-Szene und zeigt den Benutzer Informationen an
- **MainThreadDispatcher:** Da ein Paar Funktionen wie zum Beispiel das Initialisieren eines Objekt nur im HauptThread erlaubt ist um fehler zu vermeiden wird ein *MainThreadDispatcher* gebraucht. Das Skript innerhalb diesen *Gameobjekts* sorgt dann dafür das sobald der Hauptthread verfügbar ist dieser den Prozess durchführt
- **PackageClickable:** Ist ein Objekt welches die Textur des Paketes beinhaltet und zwei Code Skripte beinhaltet. Eines dieser Code Skripte ist ein *BoxCollide*: welcher von Unity zur Verfügung gestellt wird. Sowie ein Button Skript von Mixed Reality Toolkit Contributors

#### 4.4.2 Kabelerkennung

Um das Übertragung der Nachricht über das Kabel zu gewährleisten, muss zunächst die Position des Kabels in der umgebenden genau bestimmt werden. Dies geschieht durch die Aufnahme eines Bildes mit der Kamera der Hololens 2. Während dieser Aufnahmearaufgabe ist es wichtig, dass der Benutzer seinen Blick auf das jeweilige Kabel richtet. Nur so ist eine exakte Orientierung des Kabelverlaufs möglich.

(→) Schöditsch

#### 4.4.3 Start der Szene

Zu Beginn der Szene wird von den Gameobjekt *CableSearch* welches das Skript *Cable Search* beinhaltet gestartet.

(→) Schöditsch

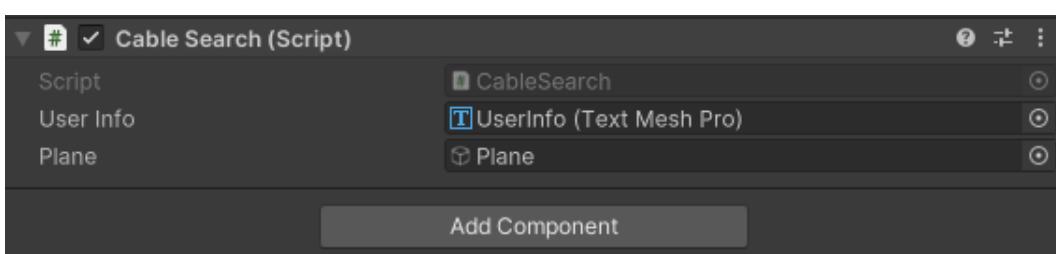


Abbildung 4.14: Komponenten des *CableSearch* Gameobjekts

Dem Skript wird zuerst ein *Text Mesh Pro* übergeben, dieser Text wird innerhalb von der Unity Szene anzeigt. Sowohl wird ein weiteres *Gameobjekt* mit den Namen *Plane* (welches ebenfalls in der Szene existiert) fürs Debuggen (Siehe Kapitel 4.4.9.1) und ausführen vom weiteren Code übergeben.

```

1 void Start() {
2     userInfo.text = "Schauen Sie auf das Rote Kabel";
3     StartCoroutine(DelayedStart());
4 }
5
6 private IEnumerator DelayedStart() {
7     yield return new WaitForSeconds(2.0f);
8     canStartScript = true;
9 }
```

Codeabschnitt 4.3: Start des *CableSearch* Skripts

Beim Start des Skripts wird die Funktion *Start* aufgerufen. Diese modifiziert den Text von *userInfo*, welcher dem Benutzer angezeigt wird. Nach dieser Änderung wird eine *Coroutine*, (siehe Kapitel 4.4.5) gestartet und an einen neuen Thread namens *DelayedStart* übergeben. Die Funktion *DelayedStart* stellt sicher, dass der Benutzer die Nachricht zwei Sekunden lang lesen kann, bevor sie durch einen Timer ersetzt wird. Dieser Timer wird in der *Update*-Funktion von Unity verarbeitet, um eine zeitgesteuerte Aktualisierung zu gewährleisten.

```

1 void Update() {
2     /* Ueberprueft, ob das Skript gestartet werden kann und ob das Skript noch schon
   ausgefuellt wurde*/
3     if (canStartScript && !objectPlaced) {
4         /* Ueberprueft ob noch Zeit uebrig ist */
5         if (timeRemaining > 1f) {
6             /* Verringert die verbleibende Zeit basierend auf der verstrichenen Zeit
   seit dem letzten Update */
7             timeRemaining -= Time.deltaTime;
8             /* Berechnet die verstrichene Zeit in Sekunden */
9             int elapsedTime = Mathf.FloorToInt(timeRemaining % 60);
10            /* Aktualisiert den Text auf dem Bildschirm mit der verbleibenden Zeit */
11            userInfo.text = elapsedTime.ToString();
12        }
13    else {
14        /* Setzt die verbleibende Zeit auf 1 Sekunde, um zu verhindern, dass sie
   negativ wird */
15        timeRemaining = 1f;
16        /* Leert den Text auf dem Bildschirm */
17        userInfo.text = "";
18        /* wird auf true gesetzt um zu verhindern das weiter ueberpueft wird
   objectPlaced = true;
19        /* fuert den weiteren code aus */
20        startPictureTaking();
21    }
22 }
23 }
24 }
```

Codeabschnitt 4.4: Update des *CableSearch* Skripts

Nach Ablauf der Wartezeit von zwei Sekunden wird eine Zeitmessung gestartet. Dabei wird zunächst geprüft, ob die Restzeit *timeRemaining* (die außerhalb der Funktion als Gleitkommazahl von sechs definiert ist), größer als eins ist. Ist diese Bedingung erfüllt, wird die Restzeit durch Abziehen der verstrichenen Zeit zwischen den *Updates* angepasst. Anschließend wird die verbleibende Zeit in Sekunden umgerechnet und dem Benutzer angezeigt. Nach Ablauf der fünf Sekunden Ladezeit wird die verbleibende Zeit auf eins gesetzt, um Fehler zu vermeiden. Danach wird der Text geleert und der boolesche Wert *objectPlaced* auf *true* gesetzt, um sicherzustellen, dass die Update-Funktion nicht weiter ausgeführt wird. Schließlich wird innerhalb der Funktion *startPictureTaking* das nächste Skript gestartet,

um den Prozess fortzusetzen.

#### 4.4.4 Foto aufnahme und verarbeitung

Dieses Kapitel beschäftigt sich mit dem aufnehmen und verarbeiten von dem Foto des ditsch Roten Kabels

##### 4.4.4.1 Photocapture Klasse

Die *PhotoCapture* Klasse benutzt die *PhotoCapture* API, um ein Foto mit der Kamera des Gerätes zu machen. Damit dies möglich ist, muss man Unity die Erlaubnis für die *Webcam* und das *Microphone* geben. Dies gibt dann die Erlaubnis das wenn zum Beispiel ein Laptop verwendet wird. Dieser über Unity code Photos/Videos sowie Sprachaufnahmen über die Kamera und Mikrophone aufnehmen kann. Dies trifft dann natürlich auch auf die Hololens 2 zu. Dies kann unter *Edit* ⇒ *Project Settings* ⇒ *Player* ⇒ *Settings for Universal Windows Platform Rightarrow Publishing Settings* ⇒ *Capabilities* gemacht werden.

Die wichtigsten Funktionen von *PhotoCapture* sind:

- **PhotoCapture.CreateAsync()**: Erstellt asynchron eine Instanz eines PhotoCapture Objekts welches benutzt werden kann um ein Photo zu schießen.
- **PhotoCapture.StartPhotoModeAsync(CameraParameter)**: Dafür werden die Parameter der Kamera benötigt, die zur *CameraParameter*-Klasse instanziert werden.
- **PhotoCapture.TakePhotoAsync()**: Macht das Foto.
- **PhotoCaptureFrame.UploadImageDataToTexture**: Kopiert das unverarbeitete Foto innerhalb einer variable.
- **PhotoCaptureObject.StopPhotoModeAsync()**: Beendet die Instanz des Photocapture und ruft danach die Funktion innerhalb der Klammer auf.

##### 4.4.4.2 Unity Canvas

Ein Canvas stellt einen Bereich dar, in dem alle Benutzeroberflächen (kurz UI) enthalten sind. Die Hierarchie von so einen UI besteht aus dem *GameObject* als sogenannten *parent* weleches als *child* die *UI* besitzt. Diese *UI* kann auch als weiteres child ein *Image* besitzen. Das Canvas wird als Rechteck dargestellt, was es erleichtert, die genaue Position des Objekts zu bestimmen.

Ein Canvas verfügt auch über verschiedene Rendermodi:

- **Screen Space-Overlay**: Dieser Modus ermöglicht es UI-Elementen, über der Szene auf dem Bildschirm gerendert zu werden. Das bedeutet, dass das UI-Element unabhängig von dem, was in der Szene passiert, über der Szene gerendert wird.
- **Screen Space-Camera**: Dieser Modus funktioniert ähnlich wie *Screen Space-Overlay*, mit dem Unterschied, dass das UI-Element vor der zugewiesenen Kamera platziert wird.
- **World Space**: Dieser Rendermodus wird in diesem Projekt am häufigsten verwendet. In diesem Modus kann der Canvas im Gegensatz zu den anderen Modi verschoben werden, genau wie alle anderen Objekte in der Szene.<sup>20</sup>

#### 4.4.4.3 Foto der Umgebung aufnehmen

Das Aufnehmen des Fotos wird mit Hilfe der *PhotoCapture*-Klasse (siehe Kapitel 4.4.4.1) aufgenommen wird. Um zu gewährleisten das es funktioniert und mit der Kamera ein qualitativ gutes Foto zum verarbeiten gemacht wird müssen Einstellungen der für die Kameraparameter eingestellt werden.

(→) Schöditsch

```

1 public void takingPicture() {
2     // ueberprueft, ob bereits ein Bild aufgenommen wird, um zu verhindern, dass die
3     // Methode erneut aufgerufen wird
4     if (!takingNewPicture) {
5         // Aktiviert das LoadingSymbol-Spielobjekt, um anzusehen, dass ein Bild
6         // aufgenommen wird
7         loadingCircle.SetActive(true);
8         // Holt sich das Canvas-Objekt innerhalb des LoadingSymbol-Spielobjekts
9         loadingCircleCanvas = loadingCircle.GetComponentInChildren<Canvas>();
10        // Startet die Ladeanimation im LoadingCircle
11        loadingCircleCanvas.GetComponent<LoadingCircle>().StartLoading();
12        // Erstellt asynchron eine Fotoaufnahme
13        PhotoCapture.CreateAsync(false, delegate (PhotoCapture captureObject) {
14            // ueberprueft, ob die Fotoaufnahme erfolgreich erstellt wurde
15            if (captureObject != null) {
16                // Speichert die Fotoaufnahme in einer Instanzvariablen
17                photoCaptureObject = captureObject;
18                // Konfiguriert die Kameraparameter fuer die Aufnahme
19                CameraParameters cameraParameters = new CameraParameters();
20                cameraParameters.hologramOpacity = 0.0f;
21                // Setzt die Auflösung der Kamera
22                cameraParameters.cameraResolutionWidth = cameraResolution.width;
23                cameraParameters.cameraResolutionHeight = cameraResolution.height;
24                // Setzt das Pixelformat fuer die Aufnahme
25                cameraParameters.pixelFormat = CapturePixelFormat.BGRA32;
26                try {
27                    // Startet asynchron den Fotoaufnahmemodus
28                    photoCaptureObject.StartPhotoModeAsync(cameraParameters, delegate
29                    (PhotoCapture.PhotoCaptureResult result) {
30                        // Nimmt ein Foto auf und speichert es im Arbeitsspeicher
31                        photoCaptureObject.TakePhotoAsync(onCapturedPhotoToMemory);
32                    });
33                }
34            });
}

```

Codeabschnitt 4.5: Bild einstellung und aufnahme

Zunächst wird in diesem Code mit dem Boolean *takingNewPicture* geprüft, ob bereits ein Bild aufgenommen wurde. Daraufhin wird sofort das Lade-Symbol (siehe Kapitel 4.4.6) gestartet und angezeigt. Anschließend wird eine asynchrone Methode *PhotoCapture.CreateAsync()* aufgerufen, um ein *Photo-Capture* zu erstellen. Diese Methode erstellt ein *PhotoCapture-Objekt*, das für die Aufnahme von Fotos verwendet wird. Die Methode akzeptiert den Parameter *false*, der angibt, dass kein Audio-Capture erstellt werden soll. Wenn die Aufnahme erfolgreich war, wird das PhotoCapture-Objekt gespeichert und die Kameraparameter für die Aufnahme konfiguriert. Anschließend wird der Photo-Capture-Modus mit *StartPhotoModeAsync* gestartet, und wenn dieser erfolgreich gestartet wurde, wird das Photo aufgenommen und mit *TakePhotoAsync* im Speicher abgelegt.

#### 4.4.4.4 Unity Job System

(→) Schöditsch

Das Job System von Unity bietet eine effiziente Möglichkeit zur Implementierung von Multithreading in Anwendungen, wodurch die Anwendung in der Lage ist, alle verfügbaren CPU-Kerne optimal zu nutzen. Im Gegensatz zu einem sequentiellen Durchlauf des Codes durch einen einzigen Thread, dem sogenannten Hauptthread, ermöglicht das Job System die parallele Verarbeitung von Codeabschnitten durch separate Arbeits-Threads. Durch die Nutzung mehrerer Arbeits-Threads können Aufgaben effizienter verteilt und gleichzeitig bearbeitet werden. Dies trägt wesentlich zur Verbesserung der Effizienz und der Gesamtdauer der Ausführung des Programms bei. Eine wichtige Eigenschaft des Job Systems besteht darin, sicherzustellen, dass nur so viele Threads verwendet werden, wie von den verfügbaren CPU-Kernen unterstützt werden.

Ein weiteres wichtiges Konzept, das im Job System implementiert ist, ist das sogenannte "Work Stealing". Dabei handelt es sich um eine Planungsstrategie für Threads, bei der ein Thread, der seine aktuellen Aufgaben abgeschlossen hat, zusätzliche Aufgaben von anderen Arbeits-Threads übernimmt und sie in seine eigene Warteschlange einfügt. Dies ermöglicht eine dynamische und effiziente Verteilung von Aufgaben zwischen den Threads und trägt dazu bei, die Last gleichmäßig auf alle verfügbaren Ressourcen zu verteilen.

Um das Schreiben von Multithread-Code zu erleichtern, verfügt das Job-System über ein Sicherheitssystem, das sicherstellt, dass potenzielle Wettlaufbedingungen eine sogenannte *race condition* zu verhindern und somit Fehler vermieden werden können. Um sicherzustellen, dass keine Wettlaufbedingungen auftreten können, erhält jeder Thread eine isolierte Kopie der Daten, die er verarbeiten kann.<sup>21</sup>

#### 4.4.4.5 Vector2 Klasse

(→) Schöditsch

Unity bietet dem Benutzer eine umfassende Funktionalität zur Darstellung und Manipulation von 2D-Positionen, -Richtungen und -Geschwindigkeiten innerhalb deren *Engine*. Mit Hilfe der *Vektor2D* Klasse von Unity lassen sich Bewegung, Interaktion und Objekte exakt steuern und simulieren. Diese Klasse besitzt die folgenden Parameter:

- *x* und *y*: x Koordinate von einem 2 Dimensionalen Koordinatensystem
- *magnitude*: Besitzt die Größe des 2D Vektors. Die Größe wird des Vektors wird mithilfe von dieser Formel berechnet  $\sqrt{x^2 + y^2}$
- *normalized*: Normalisierung bedeutet, den Vektor auf eine Länge von 1 zu bringen, während seine Richtung beibehalten wird.

#### 4.4.4.6 Fotoverarbeitung

(→) Schöditsch

Nachdem ein Foto aufgenommen wurde, erfolgt die Datenverarbeitung mittels einer *Texture2D*-Klasse innerhalb des *Unity-Job-Systems* in paralleler Ausführung. Jeder Thread des Systems ist dafür zuständig, eine festgelegte Anzahl von Pixeln im Bild zu überprüfen, um festzustellen, ob diese die Eigenschaft Rot aufweisen. Die Ergebnisse dieser Überprüfung werden in einem zweidimensionalen Boolean-Array gespeichert, wobei die x- und y-Koordinaten jeweils einer Dimension zugeordnet sind. Jeder Pixel im Array wird entsprechend seines Farbzustands mit dem booleschen Wert *true* gekennzeichnet, falls er rot ist, andernfalls wird ihm der Wert *false* zugewiesen. Dieser Prozess wird durch das *Job-System* ausgeführt, wobei jeder einzelne Job die eine gewisse Anzahl an Pixeln überprüft.

```
1 struct PixelJob : IJobParallelFor {
```

<sup>21</sup>Unity Job System

```

2     public NativeArray<Color> pixels;
3     public NativeArray<Vector2> redPixelPositions;
4     public int textureWidth;
5     public int batchSize;
6
7     public void Execute(int batchIndex) {
8         int startPixelIndex = batchIndex * batchSize;
9         int endPixelIndex = Mathf.Min((batchIndex + 1)
10             * batchSize, pixels.Length);
11
12        for (int pixelIndex = startPixelIndex;
13            pixelIndex < endPixelIndex; pixelIndex++) {
14            int x = pixelIndex % textureWidth;
15            int y = pixelIndex / textureWidth;
16
17            if (pixels[pixelIndex].r > 0.7f &&
18                pixels[pixelIndex].g < 0.5f && pixels[pixelIndex].b < 0.5f) {
19                pixels[pixelIndex] = Color.red;
20                redPixelPositions[pixelIndex] = new Vector2Int(x, y);
21            }
22            else {
23                pixels[pixelIndex] = Color.black;
24            }
25        }
26    }
27 }
```

Codeabschnitt 4.6: Rote Pixel suche

- **NativeArray:** bieten verbesserte Leistung im Vergleich zu standardmäßige Arrays, da diese direkt auf native Speicherbereiche zugreifen können, was besonders wichtig ist, wenn große Datenmengen verarbeitet werden müssen und die Ausführung parallelisiert werden soll.
- **pixels:** pixels ist ein *NativeArray* welcher die aktuell noch unbearbeiteten Pixels beinhaltet.
- **batchSize:** Die *batchSize* beinhaltet die Anzahl von Pixel die jeder *Job* durchgeht und überprüft.
- **texturewidth:** Wird vor dem Aufruf der *Jobs* berechnet um die Koordinaten des roten Pixels herauszufinden.
- **batchIndex:** Nachdem der Prozess unter mehreren *Jobs* aufgeteilt wird und alle *Jobs* von an den selben *NativeArray pixels* arbeiten. Bekommt jeder *Job* einen *batchIndex* mit, dieser zeigt an wo die Arbeit dieses *Jobs* innerhalb des *NativeArray* beginnt.
- **startPixelIndex:** nachdem jeder Arbeiter Thread seine eigenen Batches Besitz muss der Index für diese berechnet werde.
- **redPixelPositions:** Wird vor der Initialisierung in der Klasse mit der selben Länge wie Pixels erstellt.

#### 4.4.4.7 Kabel finden

(→) Scho-

Nachdem das Foto verarbeitet wurde wird nach roten Pixel in dem Bild gesucht, um danach die noch fehlende Entfernung des Kabels zu messen. Mit dieser hat man dann die genaue Position des roten Kabels in der Echten Welt errechnet. Um dies zu erreichen wird nach roten Objekten Roten Pixeln im Bild gesucht. Da es aber unsicher ist, ob diese roten Pixel zu einem Kabel gehören oder nicht, wird eine Methode angewendet, um die längste

ditsch

zusammenhängende Reihe von roten Pixeln zu ermitteln. Es kann auch dazu kommen, das rote Pixel nicht erkannt werden, welche zum roten Kabel gehören. Aus diesem Grund wird ein Toleranzbereich festgelegt. Es wird geschaut ob in einen bestimmten Bereich rote Pixel existieren von welchen aus dann weiter geschaut wird. Dadurch wird eine präzisere Erkennung und Zuordnung von relevanten roten Pixeln im Kontext des zu identifizierenden Kabels ermöglicht. Die methodische Vorgehensweise zur Berechnung gestaltet sich wie folgt:

```

1 List<Vector2> SearchForRedPixels(int startX, int startY) {
2     /* Ist die momentane Line an der er Schaut ob sie einen weiteren Punkt findet
3      */
4     List<Vector2> currentLine = new List<Vector2>();
5     bool hasFound = false;
6     /* foundX und foundY sind die momentare x und y positionen des Roten pixels
7      von den man ausgeht */
8     int foundX = startX, foundY = startY;
9     /* heightY wird verwendet um zu ueberpruefen ob die gepruefte laenge nicht die
10    exestierente uebertrifft */
11    int heightY = redPixels.GetLength(1);
12    /* widthX wird verwendet um zu ueberpruefen ob die gepruefte breite nicht die
13    exestierente uebertrifft */
14    int widthX = redPixels.GetLength(0);
15
16    do {
17        hasFound = false;
18        /* Damit die funktion nicht in eine unendlich schleife bei den darueber
19        und darunter ueberpruefen kommt wird die Position von redPixels auf false gesetzt
20        */
21        redPixels[foundX, foundY] = false;
22        currentLine.Add(new Vector2(foundX, foundY));
23        /* ueberprueft ob 25 pixel "rechts" von den Pixel von den man ausgehen
24        mindestens ein roter vorhanden ist */
25        for (int i = 1; i < 25 && foundX + i < widthX; i++) {
26            if (redPixels[foundX + i, foundY]) {
27                foundX = foundX + i;
28                hasFound = true;
29            }
30        }
31
32        /* ueberprueft ob 10 pixel "unter" von den Pixel von den man ausgehen
33        mindestens ein roter vorhanden ist */
34        for (int i = 1; i < 10 && !hasFound && foundY - i > 0; i++) {
35            hasFound = redPixels[foundX, foundY - i];
36            if (hasFound) {
37                foundY = foundY - i;
38            }
39        }
40        /* ueberprueft ob 10 pixel "ueber" von den Pixel von den man ausgehen
41        ueberprueft ein roter vorhanden ist */
42        for (int i = 1; i < 10 && !hasFound && foundY + i < heightY; i++) {
43            hasFound = redPixels[foundX, foundY + i];
44            if (hasFound) {
45                foundY = foundY + i;
46            }
47        }
48    } while (hasFound);
49    return currentLine;
50}

```

Codeabschnitt 4.7: Kabel suche

Die Funktion beginnt mit der Erstellung einer Liste von Vektoren namens *currentLine*, die dazu dient, die Positionen der gefundenen Pixel zu speichern. Zu Beginn wird festgelegt, dass kein rotes Pixel gefunden wurde (*hasFound = false*). Die x- und y-Positionen des Startpixels werden ebenfalls initialisiert (*foundX = startX*, *foundY = startY*). Zudem werden Variablen für die Höhe (*heightY*) und Breite (*widthX*) des Arrays *redPixels* festgelegt, um sicherzustellen, dass die Überprüfungen nicht außerhalb des gültigen Bereichs liegen. Der Hauptteil der Funktion ist eine Schleife, die so lange ausgeführt wird, bis kein rotes Pixel mehr gefunden wird. Innerhalb dieser Schleife wird zuerst das aktuelle Pixel als bereits überprüft markiert und seiner Position zur Liste *currentLine* hinzugefügt. Anschließend wird nach roten Pixeln in den nächsten 25 Pixeln nach rechts gesucht. Dabei wird überprüft, ob das Pixel in *redPixels* an der Position (*foundX + i*, *foundY*) den Wert *true* hat, wobei *i* von 1 bis 24 läuft. Wenn ein rotes Pixel gefunden wird, wird dessen Position aktualisiert und die Schleife erneut durchlaufen. Falls kein rotes Pixel in den nächsten 25 Pixeln nach rechts gefunden wird, wird nach unten und dann nach oben gesucht. Dabei wird jeweils überprüft, ob in den nächsten 10 Pixeln in der entsprechenden Richtung ein rotes Pixel vorhanden ist. Wenn eins gefunden wird, wird dessen Position aktualisiert und die Schleife erneut durchlaufen. Diese Schleife setzt sich solange fort, bis kein rotes Pixel mehr gefunden wird (*hasFound = false*). Schließlich wird die Liste *currentLine*, die die Positionen der gefundenen Pixel enthält, zurückgegeben.

#### 4.4.5 Coroutines

*Coroutines* werden immer noch mit den Mainthread ausgeführt, erlaube es den für *Coroutines* verwendeten Code über mehrere Frames aufzuteilen. *Coroutines* werden deswegen hauptsächlich in Methoden verwendet welche eine Abfolge von Ereignissen über zeit darstellt.

(→) Schöditsch

Zum Darstellen von *Coroutines* ist dieses kurze Script welches das Objekt an den das angebunden ist in eine Richtung bewegt.

```

1 public class CoroutineExample : MonoBehaviour {
2     // Das Zielobjekt, zu dem das Objekt bewegt werden soll
3     public Transform target;
4     // Die Geschwindigkeit, mit der das Objekt bewegt wird
5     public float moveSpeed = 1f;
6     // Die Anzahl der Schritte fñr die Bewegung
7     public int numSteps = 10;
8
9     void Start() {
10         // Starte die Coroutine fñr die Bewegung
11         StartCoroutine(MoveToTarget());
12     }
13
14     IEnumerator MoveToTarget() {
15         // Die Startposition des Objekts
16         Vector3 startPos = transform.position;
17         // Die Zielposition des Objekts
18         Vector3 endPos = target.position;
19         // Iteriere ueber jeden Schritt der Bewegung
20         for (int i = 0; i <= numSteps; i++) {
21             // Berechne den Fortschritt der Bewegung (Wert zwischen 0 und 1)
22             float t = (float)i / numSteps;
23             // Bewege das Objekt entsprechend dem aktuellen Fortschritt
24             transform.position = Vector3.Lerp(startPos, endPos, t);
25             // Wartet fuer eine bestimmte Zeit, bevor der nñchste Schritt ausgefuehrt
26             wird
yield return new WaitForSeconds(moveSpeed / numSteps);
}

```

```

27     }
28 }
29 }
```

Codeabschnitt 4.8: Coroutine Beispiel

Wenn dieses Skript zu einem *GameObject* hinzugefügt wird, bewegt sich das Objekt mit der angegebenen Geschwindigkeit. Diese Geschwindigkeit gibt die variable *moveSpeed* an, welches wenn nichts angegeben wird, standardmäßig auf eins gesetzt wird. Es wird auch angegeben in wie vielen Schritten (also Frame-Passagen) der Vorgang stattfinden soll, dies übernimmt die variable *numSteps*. Dann soll auch noch die Position, wo das Objekt am Ende ankommen soll als *Transform* mit dem Namen *target* angegeben werden. Die *Transform* Klasse welche speichert die Position, Rotation und Größe und lässt diese auch bearbeiten. Beim Start des Skripts wird sofort die *Couroutine* mit der von Unity zur Verfügung gestellten *StartCoroutine* mit der entsprechenden *IEnumerator* Funktion innerhalb der Klammern aufgerufen. Die Funktion *MoveToTarget* muss mit der Klasse *IEnumerator* geschrieben werden, da wenn es ohne der *IEnumerator* Klasse nicht möglich wäre, die Methode *MoveToTarget* als Coroutine zu verwenden und mit *StartCoroutine* aufzurufen. *Coroutine*-Funktionen müssen eine *IEnumerator*-Rückgabe haben, damit Unity sie als asynchrone Task ausführen kann. Durch die Verwendung von *IEnumerator* können innerhalb von *Coroutine* *yield*-Anweisungen verwendet werden, um zu steuern, wann und wie die Ausführung unterbrochen und fortgesetzt wird. In diesem Beispiel wird *yield return new WaitForSeconds(moveSpeed / numSteps)* verwendet, um zwischen den einzelnen Schritten der Bewegung eine Pause einzulegen, bevor der nächste Schritt im nächsten Frame ausgeführt wird. Dies ermöglicht eine fließende und kontrollierte Bewegung des Objekts.

Falls man aus irgendeinen Grund so eine Coroutine beenden will kann man auch *StopCoroutine* oder *StopAllCoroutines* verwenden. Als Beispiel wird eine sehr kleine Erweiterung von den Code Abschnitt 4.8.

```

1 public void stopFunctionCoroutine() {
2     StopCoroutine(MoveToTarget)
3 }
4
5 public void stopAllCurrentCoroutine() {
6     StopAllCoroutines()
7 }
```

Codeabschnitt 4.9: Coroutine beenden

- *StopCoroutine*: Sorgt dafür das nur die *IEnumerator* Funktion welche für die *Coroutine* verwendet wurde beendet wird.
- *StopAllCoroutines*: Sorgt dafür das alle Funktion welche für alle laufenden *Coroutine* verwendet wurde beendet werden.

#### 4.4.6 Lade Symbol

Das Lade Symbol ist eine visuelle Darstellung, die den Benutzer darüber informiert, dass Berechnungen oder Prozesse im Hintergrund ausgeführt werden. In erster Linie wird dieses Symbol während der Kabelsuche verwendet. Diese Animation des Lade Symbols wird durch entsprechenden Programmcode gesteuert, um dem Benutzer eine klare und konsistente Rückmeldung zu geben, dass der Prozess läuft. Im Design der Interaktion spielt das Ladesymbol eine wichtige Rolle, um den Benutzer über den aktuellen Status der Anwendung zu informieren.

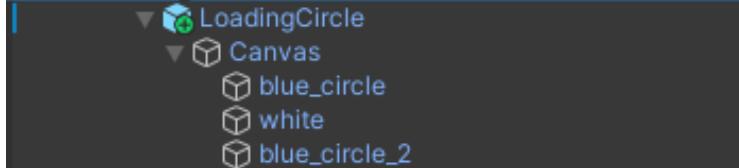


Abbildung 4.15: Hierarchie Lade Kreises im Unity Editor

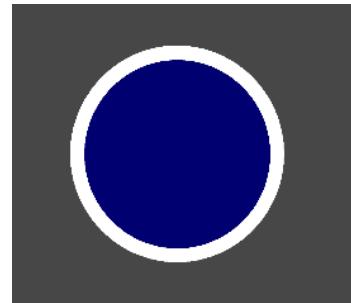


Abbildung 4.16: Lade Symbol

```

1 IEnumerator LoadingCoroutine() {
2     while (isRunnig) {
3         if (img.fillAmount >= 1f) {
4             img.fillAmount = 0f;
5         }
6         else {
7             img.fillAmount += fillSpeed;
8         }
9         yield return new WaitForSeconds(0.1f);
10    }
11 }
```

Codeabschnitt 4.10: Code zum ausführen des lade Symbols

Um eine Funktion in einem parallelen Thread auszuführen, während der Hauptthread gleichzeitig Berechnungen durchführt, ist es erforderlich, eine Funktion in einen separaten Thread auszulagern. Dies wird in den Fall durch die Verwendung von Asynchronität erreicht. Dafür gibt Unity die Funktion Coroutine-Routinen zur Verfügung. Bei der Initialisierung dieser Funktion wird die Methode *StartCoroutine(LoadingCoroutine())* aufgerufen. Durch diesen Aufruf wird sichergestellt, dass die Funktion nebenbei und asynchron zum Code davor(main thread) ausgeführt wird. Zum anzeigen von den Lade Symbol wird ein Objekt *img* verwendet, dieses *img* beinhaltet ein Bild eines Weißen Kreises (äußerer Kreis von Abbildung (4.16)) und stellt den Fortschritt während der Berechnung dar.

#### 4.4.7 Bewegung des Paketes

Um das Paket über das Kabel gleiten zu lassen, muss zuerst der Abstand gemessen werden. Sobald diese gemessen wurde, sind alle wichtigen Punkte berechnet, über die das Paket gleiten soll. Dann muss das Objekt, welches das Paket darstellt, initialisiert und am Anfang des Kabels positioniert werden. Danach wird sich das Paket über die Punkte bewegt.

(→) Schöditsch

#### Definition eines Rays

Ein Ray ist ein abstraktes Konzept in der Computergrafik und Physiksimulation, das einen unendlich langen, geraden Strahl repräsentiert. Dieser Strahl wird durch einen Ausgangspunkt definiert, der üblicherweise als *Ursprung* bezeichnet wird. Im Kontext von dreidimensionalen Szenen und Visualisierungen entspricht der Ursprung oft der Position einer *virtuellen Kamera* oder eines *Blickpunkts*. Der Ray erstreckt sich dann in eine bestimmte Richtung, die durch Vektor definiert wird. Diese Richtung kann durch verschiedene Methoden festgelegt werden, abhängig von der Anwendung, in der der Ray verwendet wird. Im

(→) Schöditsch

Falle der Bildschirmkoordinaten kann die Richtung beispielsweise durch Blick des Benutzers bestimmt werden. In der Praxis wird der Ray häufig dazu verwendet, um *Kollisionen mit Objekten in einer Szene zu erkennen* oder *um Lichtstrahlen für die Beleuchtungsrechnung zu simulieren*. Durch das Schießen eines Rays in eine Szene und das Überprüfen auf *Kollisionen* mit den vorhandenen Objekten kann festgestellt werden, ob der Ray ein Objekt trifft und wenn ja, an welcher *Stelle* und unter welchem *Winkel*.

#### 4.4.7.1 ARRaycastHit Klasse

Die *ARRaycastHit* Klasse in Unity wird verwendet, um die Ergebnisse eines *Raycasts* in Augmented Reality (AR) Anwendungen zu erhalten. Ein Raycast ist eine Technik, bei der ein imaginärer Strahl (Ray) von einem Startpunkt aus in eine bestimmte Richtung gesendet wird, um festzustellen, ob er auf eine Oberfläche auftrifft, und, wenn dies der Fall ist, Informationen über den Auftreffpunkt zurückzugeben. Diese Informationen werden dann von der *ARRaycastHit*-Klasse aufgefangen und zur Verfügung gestellt. Die *ARRaycastHit*-Klasse enthält mehrere Methoden, die es ermöglichen, auf die Details des Treffpunkts zuzugreifen und entsprechende Aktionen in einer AR-Anwendung auszuführen. Dazu gehören Informationen wie die Position des Trefferpunktes, die Entfernung zum Startpunkt des Strahls. Durch die Verwendung der *ARRaycastHit*-Klasse ist es möglich, präzise Interaktionen in AR-Szenen zu implementieren, wie z.B. das Platzieren virtueller Objekte auf realen Oberflächen. Darüber hinaus ermöglicht die *ARRaycastHit*-Klasse das Hinzufügen von physikalischen Eigenschaften in AR-Anwendungen, wodurch z. B. Kollisionen integriert werden können. Diese Klasse besitzt die folgenden Parameter:

- *hit*: *hit* ist eine *XRRaycastHit*-Klasse welche die unverarbeiteten Daten von den *hit* beinhaltet.
- *distance*: Enthält die Entfernung vom Ursprung des Strahls bis zu dem Punkt, an dem er getroffen wurde.
- *hitType*: Trefferart, die angibt, ob der Strahl einen Punkt (Feature Point) oder ein Objekt (Trackable) getroffen hat.

#### 4.4.7.2 Entfernung messen

Da man noch die z-Achse, also die Tiefe, ermitteln muss, um das visuelle Paket angemessen auf dem (physischen) zu platzieren und bewegen zu können, werden im späteren Verlauf RayCasts<sup>22</sup> verwendet. Aufgrund der Ressourcenintensität, die mit dem Versenden eines Raycasts für jede Position verbunden wäre, um die z-Achse für jede x/y Koordinate zu erhalten, ist eine Optimierung notwendig. Zur Verbesserung der Effizienz werden die wichtigsten Positionen berechnet. Diese wichtigen Punkte sind der Anfangspunkt des Kabels (am ersten PC), der Mittelpunkt des Kabels, der Endpunkt des Kabels (am zweiten PC), sowie zehn andere Punkte welche gleichmäßig über das gesamte Kabel verteilt sind. Diese Punkte werden in einer globalen *List<Vector3>* Variable namens *redPixelCoordinates* gespeichert um auf alle 3 Koordinaten (x/y/z) zugreifen zu können. Diese Liste wird durch Iterieren und für jeden Punkt wird ein Raycast ausgeführt.

```
1 foreach (Vector2 redPixel in redPixelCoordinates) {
2     positionVirtualObject(redPixel, targetTexture)
3 }
```

Codeabschnitt 4.11: Iteration durch die Liste der Raycastpunkte

---

<sup>22</sup>Unity Raycast

- *positionVirtualObject* ist die Funktion welche die folgenden Berechnungen für Raycasts errechnet und den Raycast ausführt.

Um Raycasts erfolgreich zu nutzen, greift Unity auf Bildschirmkoordinaten zurück. Jedoch liegen uns momentan nur die Koordinaten vor, die der Skalierung des Bildes entsprechen. Daher müssen diese in ein Format umgerechnet werden, das es ermöglicht, die Raycasts an die richtigen Positionen zu senden. Diese Umrechnung erfolgt im Code wie folgt:

```

1 Vector2 screenCoordinates = new Vector2(
2     redPixel.x * Camera.main.pixelWidth / image.width,
3     redPixel.y * Camera.main.pixelHeight / image.height
4 );

```

Codeabschnitt 4.12: Koordinaten Umrechnung

- **redPixel.x** und **redPixel.y** sind die Positionen welche in *screenCoordinates* umgerechnet werden soll.
- **Camera.main.pixelWidth** und **Camera.main.pixelHeight** sind die Breite und Höhe des Bildschirms, auf dem die Kameraansicht dargestellt wird.
- **image.width** und **image.height** sind die gesamte Breite und Höhe des verarbeiteten Bildes vom roten Kabel in Pixels.

Sobald dann die *screenCoordinates* berechnet wurden, wird der Raycast auf die Position im Bildschirm aufgerufen werden. Dafür ist dieser Code teil von *positionVirtualObject* zuständig:

```

1 List<ARRaycastHit> hits = new List<ARRaycastHit>();
2 if (raycastManager.Raycast(screenCoordinates, hits, TrackableType.Planes)) {
3     for (int i = 0; i < hits.Count; i++) {
4         if (hits[i].hitType != TrackableType.PlaneWithinInfinity) {
5             if (hits[i] != null) {
6                 cablePositinos.Add(new Vector3(hits[i].pose.position.x, hits[i].pose.
7                     position.y + 0.05f, hits[i].pose.position.z));
8                 break;
9             }
10        }
11    }

```

Codeabschnitt 4.13: Raycast schießen

Es wird eine Liste von ARRaycastHit Objekten erstellt, die den Namen *hits* trägt. In dieser Liste werden nach dem Ausführen einer AR-Raycast-Operation die Treffer gespeichert. Die Funktion Raycast erhält die *screenCoordinates*, die *hits*-Liste sowie die zu suchenden *TrackableType.Planes*. Wenn die Funktion mindestens einen Treffer erhält, werden alle Treffer überprüft. Diese Treffer werden dann daraufhin überprüft, ob ihr Typ nicht *TrackableType.PlaneWithinInfinity* ist, um sicherzustellen, dass der Raycast nicht ins Unendliche geht. Wenn ein Objekt getroffen wurde, wird geprüft, ob es Werte besitzt. Wenn alle diese Voraussetzungen erfüllt sind, wird es mit einer um 5 cm erhöhten Höhe (damit das Paket nicht innerhalb von Kabel gleitet sondern leicht darüber) zu *cablePositions* hinzugefügt. *cablePositions* ist die Position, die das Paket später durchqueren wird.

#### 4.4.7.3 Erstellung und Positionierung des Paketes

Nach der Bestimmung der kritischen Positionen mittels Raycast hat der Benutzer die Möglichkeit, über die Website eine Nachricht zu übermitteln. Die Übermittlung dieser

(→) Scho-

ditsch

Nachricht löst ein entsprechendes Ereignis aus, das zur Initialisierung und zum Absenden des Datenpakets führt.

Bei der *Start* Funktion welche von Unity zur Verfügung gestellt wird. Führt diese Zeile aus:

```
1 EventManager.OnMessageReceived += editAndSendPackage;
```

Codeabschnitt 4.14: Binden an der Methode

Diese Zeile sorgt dafür das beim Start der Unity *Scene* *editAndSendPackage* an dem Event *OnMessageReceived* gebunden wird. Das bedeutet das wenn das Event *OnMessageReceived* erhalten wird, wird die Funktion *editAndSendPackage* ausgelöst. Die Funktion *editAndSendPackage* initialisiert das visuelle Paket am Anfang des Kabels falls es noch nicht erstellt wurde.

```
1 public void editAndSendPackage(string username, string message) {
2     if (isAtEnd) {
3         isAtEnd = false;
4         moveToCounter = 1;
5         moveFromCounter = 0;
6     }
7     MainThreadDispatcher.Instance().Enqueue(() => {
8         if (isObjectInstantiated == false) {
9             instantiatedObject = Instantiate(virtualObject, cablePositinos[0],
10             Quaternion.identity);
11             isObjectInstantiated = true;
12         }
13         else {
14             Destroy(instantiatedObject);
15             instantiatedObject = Instantiate(virtualObject, cablePositinos[0],
16             Quaternion.identity);
17         }
18         shouldMove = !shouldMove;
19         EditTextOfInformationObject editTextOfInformationObject = new
20         EditTextOfInformationObject(infoTextS);
21         editTextOfInformationObject.GetTextMeshProFromChild();
22         editTextOfInformationObject.EditText(username, message);
23         EventManager.SendMsg(username, message);
24     });
25 }
```

Codeabschnitt 4.15: Initialisierung und Bearbeitung des Paketes

Im Rahmen der Methode wird zunächst geprüft, ob ein Paket erstellt wurde und ob es sich bereits am Ende des vorgesehenen Weges befindet. Sollte dies der Fall sein, wird der Zähler für die Bewegung des Paket zurückgesetzt. Aufgrund der Tatsache, dass der *EventManager*, der die Funktion aufruft, nicht im Hauptthread läuft und die Initialisierung des Paket sowie die Funktion *EventManager.SendMsg* nur im Hauptthread ausgeführt werden darf, ist die Verwendung des *MainThreadDispatcher* erforderlich. Dieser stellt sicher, dass dieser Codeabschnitt nicht von einem Nebenthread, sondern vom Hauptthread ausgeführt wird. Dies ist notwendig, um sicherzustellen, dass die Unity-Szene synchronisiert ist, wie von Unity vorgegeben. Innerhalb des *MainThreadDispatcher* wird, wenn das Objekt noch nicht erstellt wurde, dieses am Anfang des Kabels erstellt. Wenn das Paket jedoch bereits erstellt wurde, wird es zerstört und wieder auf die Startposition zurückgesetzt. Die Klasse *EditTextOfInformationObject* ist dafür verantwortlich, das *TextMeshPro* des Gameobjects zu verwenden, um die korrekten Daten im Text anzuzeigen.

#### 4.4.7.4 Bewegung des Paketes

(→) Scho-

Die Bewegung des Paketes läuft so ab, dass das Paket nacheinander die Punkte durchläuft, welche davor berechnet wurden. Die Bewegung zwischen diesen Positionen wird innerhalb von Unity über die Funktion *Update* ausgeführt. Diese Funktion wird bei jedem Frame aufgerufen, um eine kontinuierliche Aktualisierung der Position des Pakets zu gewährleisten und somit eine nahtlose Bewegung darzustellen zu erzielen.

```

1 void Update() {
2     if (shouldMove) {
3         if (moveToCounter < cablePositinos.Count) {
4             instantiatedObject.transform.Translate((cablePositinos[moveToCounter] -
5                                         cablePositinos[moveFromCounter]) * 0.0075f);
6             if (Vector3.Distance(instantiatedObject.transform.position, cablePositinos
7 [moveToCounter]) < 0.00375f) {
8                 instantiatedObject.transform.position = cablePositinos[moveToCounter];
9                 moveToCounter++;
10                moveFromCounter++;
11                if (Vector3.Distance(instantiatedObject.transform.position,
12                                     cablePositinos[cablePositinos.Count - 1]) < 0.00375f) {
13                    isAtEnd = true;
14                }
15            }
16        }
17    }
18 }
```

Codeabschnitt 4.16: Update Funktion

Dieses Skript beinhaltet drei Globale Variablen welche von mehreren Funktionen gebraucht werden

- *shouldmove* ist ein Boolischer Wert (welcher wahr oder falsch sein kann.) Dieser wird verwendet um zu überprüfen ,ob das Objekt initialisiert wurde und sich bewegen sollte.
- *moveFromCounter* ist ein Integer Wert, welcher mit zählt, wie oft das Paket schon von einer Position aus weg bewegt hat. Dieser Zähler wird als Index für die Liste *cablePositions* verwendet um die momentane Position des Paketes zu finden.
- *moveToCounter* ist genauso wie *moveFromCounter* ein Integer, welcher ebenfalls als Index für die Liste *cablePositions* benutzt wird um die Position zu finden wo das Paket als nächstes hin soll.

Nachdem geprüft wurde, ob sich das Objekt bewegen soll, wird weiterhin überprüft, ob die Nachricht bereits am Ende angekommen ist. Dies wird durch den Vergleich zwischen der Variable *moveToCounter* und der Anzahl der Positionen in *cablePositions* festgestellt. Diese Überprüfung ist erforderlich, da der Index einer Liste bei null beginnt, und eine Fortsetzung ohne diese Kontrolle zu einer möglichen Null-Exception führen würde. So lange *moveToCounter* kleiner ist als die Anzahl der Positionen, wird die Bewegung mittels *instantiatedObject.transform.Translate* ausgeführt. Die Bewegung selbst wird durch die Differenz der nächsten Position, zu der das Objekt bewegt werden soll, und der aktuellen Position, von der das Paket sich weg bewegt, definiert. Diese Translation wird mit einem Faktor multipliziert, in diesem Fall mit *0.0075f*, um die Geschwindigkeit des Objekts anzupassen. Die Bewegung erfolgt mit einer Geschwindigkeit von 0.0075 Metern pro Frame(, da die Einheiten direkt in Metern angegeben sind). Nach jeder Bewegung wird überprüft,

ob das Paket bereits den Endpunkt erreicht hat. Dieser Überprüfungsbereich umfasst die Hälfte der Geschwindigkeit, mit der sich das Objekt bewegt. Diese Wahl erfolgt, damit das Objekt nicht versehentlich über den Endpunkt hinausgeht, da es sich mit einer festen Geschwindigkeit von 0.0075 bewegt und somit mindestens um die Hälfte, also 0.00375, überprüft werden muss. Sobald sich das Objekt innerhalb dieses Bereichs befindet, wird seine Position festgelegt, um zu verhindern, dass das Paket über das Ziel hinausläuft. Anschließend wird der Bewegungszähler erhöht, und falls dies nicht der letzte Durchlauf war, wird der Vorgang wiederholt.

#### 4.4.8 Anzeigen von Informationen der Nachricht

Der Benutzer kann Mithilfe der Website welche auf beiden Laptops (links und rechts) offen ist eine Nachricht innerhalb des Paketes senden. Diese Informationen werden dann angezeigt wenn der Benutzer auf das virtuelle Paket klickt. Dies wird mithilfe des Gameobjekt *InfoObject* gemacht welches Voreinstellungen für den Hintergrund und Text besitzt falls es zu Fehlern kommt.

(→) Schöditsch

Um den Text des Objekts zu ändern muss auf das *Text Mesh Pro* Gameobjekt welches ein unter Objekt des *InfoObject* zugegriffen werden. Dies passiert durch diesen code

```

1 public class EditTextOfInformationObject : MonoBehaviour
2 {
3     // Referenz auf das Elternspielobjekt, von dem aus das Kindspielobjekt gefunden
4     // werden soll
5     public GameObject parentGameObject;
6     // Referenz auf das TextMeshPro-Objekt, das bearbeitet werden soll
7     TextMeshPro textMeshPro;
8
9     // Methode zum Abrufen des TextMeshPro-Objekts vom Kindspielobjekt
10    public void GetTextMeshProFromChild() {
11        // Ueberpruefen, ob das Elternspielobjekt vorhanden ist
12        if (parentGameObject != null)
13        {
14            // Das Kindspielobjekt mit dem Namen InformationTextS im Elternspielobjekt
15            // finden
16            GameObject childGameObject = parentGameObject.transform.Find("InformationTextS").gameObject;
17            // Ueberpruefen, ob das Kindspielobjekt gefunden wurde
18            if (childGameObject != null) {
19                // Das TextMeshPro-Komponente vom Kindspielobjekt abrufen
20                TextMeshPro tmp = childGameObject.GetComponent<TextMeshPro>();
21                // Ueberpruefen, ob das TextMeshPro-Komponente vorhanden ist
22                if (tmp != null) {
23                    // Das TextMeshPro-Objekt der Klasse zuweisen
24                    textMeshPro = tmp;
25                }
26            }
27        }
28    }

```

Codeabschnitt 4.17: Kind vom Gameobjekt bekommen

Dieser Code definiert eine Funktion *EditTextOfInformationObject*, die ein TextMeshPro-Objekt aus einem bestimmten untergeordneten Objekt des übergeordneten *InfoObject* objekt. Die Methode *GetTextMeshProFromChild* sucht nach einem spezifischen untergeordneten objekt namens *InformationTextS* welches den Text besitzt welcher angezeigt werden soll. Dieses versucht dann, das TextMeshPro-Objekt aus diesem Kindspielobjekt zu erhalten. Dadurch ermöglicht es den Zugriff und die Bearbeitung des Textes.

```

1 public void EditText(string username, string message) {
2     if (textMeshPro != null && username != null && message != null) {
3         textMeshPro.text = "Absender: " + username + "\n\nNachricht: " + message;
4     }
5 }
```

Codeabschnitt 4.18: Kind vom Gameobjekt bekommen

Dieser Code definiert eine Methode namens *EditText*, die verwendet wird, um den Text eines *TextMeshPro*-Objekts zu ändern. Die Methode akzeptiert zwei Parameter *username* und *message*, die den Absender und den Nachrichtentext repräsentieren welcher von dem Nutzer davor gesendet wurde. Die Methode überprüft zunächst, ob das *textMeshPro*-Objekt nicht null ist und ob sowohl *username* als auch *message* nicht null sind. Wenn diese Bedingungen erfüllt sind, wird der Text des *textMeshPro*-Objekts aktualisiert.

#### 4.4.9 Debugging Optionen

Während der Entwicklung dieses Projekts wurden spezifische Funktionen implementiert, um Fehler und Bugs im Code zu identifizieren. Zu diesem Zweck wurden zwei Funktionen mit den Bezeichnungen *debugRaycast* und *DebugCameraResolution* entwickelt. Darüber hinaus wurde ein *Plane* Gameobjekt namens *PixelDebug* erstellt, das als Hilfsmittel dient, um Fehler im Code zu diagnostizieren und zu beheben. Diese zusätzlichen Implementierungen tragen wesentlich dazu bei, die Fehlerfreiheit des entwickelten Systems zu verbessern, indem sie eine strukturierte und systematische Fehlerbehandlung ermöglichen.

(→) Schöditsch

##### 4.4.9.1 Anzeigen der Kabelerkennung

Das *GameObject* mit dem Namen *PixelDebug* dient als Hilfsmittel zur Visualisierung der Erkennung roter Pixel. Diese Einrichtung ermöglicht die Darstellung aller identifizierten roten Pixel als 2D-Textur auf dem *GameObject*, wodurch die vom Skript erkannten roten Pixel sichtbar werden. Die Steuerung der Aktivierung bzw. Deaktivierung der Objektdarstellung kann über das Skript *Deactivate Object* erfolgen. Diese Prozedur bietet eine effektive Methode zur Überprüfung der Pixelerkennung in der Umgebung.

(→) Schöditsch

Foto einfügen

##### 4.4.9.2 Debug Funktion von Raycasts

Um während dem Programmieren des Prozesses von Raycasts sicherzustellen, dass in der Funktion *positionVirtualObject* die korrekte Position getroffen, wurde kann man die Funktion *debugRaycast* verwenden. Diese Funktion dient der Visualisierung von dem getroffenen Punkte des Raycasts. Zur Ausführung dieser Funktion muss das *hit-Objekt* des Raycasts übergeben werden. Dies gewährleistet eine präzise Überprüfung und gegebenenfalls Anpassung der Raycast-Positionen für eine genauere Abbildung der realen Umgebung. Die Verwendung der *debugRaycast*-Funktion ermöglicht somit eine verbesserte Qualität und Zuverlässigkeit in Anwendungen, die auf Raycasting basieren.

(→) Schöditsch

```

1 void debugRaycast(ARRaycastHit hit) {
2     Debug.Log("Ray hit position: " + hit.pose.position);
3     GameObject instantiatedObject = Instantiate(virtualObject, hit.pose.position,
        Quaternion.identity);
```

4 }

Codeabschnitt 4.19: Debug Funktion von Raycasts

#### 4.4.9.3 Debug Funktion von der Kamera Benutzung

(→) Schö-

Diese Funktion kann verwendet werden, um sicherzustellen, dass die Anwendung ordnungsgemäß auf die Kamera zugreift und eine korrekte Bildauflösung erhält.

ditsch

```

1 void DebugCameraResolution() {
2     /* Ueberpruefen, ob Kamerageraete verfuegbar sind */
3     if (WebCamTexture.devices.Length > 0) {
4         /* Das erste Kamerageraet auswählen
5            WebCamDevice frontCamera = WebCamTexture.devices[0];
6
7         /* Eine WebCamTexture mit dem ausgewählten Kamerageraet erstellen */
8         WebCamTexture webcamTexture = new WebCamTexture(frontCamera.name);
9
10        /* Die Textur abspielen, um sicherzustellen, dass die Kamera ordnungsgemäß
11           initialisiert wird */
12        webcamTexture.Play();
13
14        /* Ueberpruefen, ob die Kamera eine gueltige Auflösung zurueckgibt */
15        if (webcamTexture.width > 0 && webcamTexture.height > 0) {
16            /* Auflösung protokollieren */
17            Debug.Log("Selected Resolution: " + webcamTexture.width + "x" +
18                      webcamTexture.height);
19        }
20        else {
21            /* Fehlermeldung ausgeben, wenn die Kamera keine gueltige Auflösung
22               zurueckgibt */
23            Debug.LogError("Failed to get a valid camera resolution.");
24        }
25    }
26    else {
27        /* Fehlermeldung ausgeben, wenn keine Kamerageräte gefunden werden*/
28        Debug.LogError("No camera devices found.");
29    }
30 }
```

Codeabschnitt 4.20: Debug Funktion von der Kamera Benutzung

Durch den bereitgestellten Code wird überprüft, ob die Kamera eine gültige Auflösung liefert. Zunächst wird geprüft, ob überhaupt ein Kameragerät verfügbar ist. Wenn ja, wird das erste Kameragerät ausgewählt und eine Textur mit diesem Gerät erstellt. Anschließend wird die Textur abgespielt, um sicherzustellen, dass die Kamera ordnungsgemäß initialisiert wird. Wenn die Kamera eine gültige Auflösung zurückgibt, wird die Auflösung protokolliert. Andernfalls wird eine Fehlermeldung ausgegeben. Falls keine Kamerageräte gefunden werden, wird ebenfalls eine entsprechende Fehlermeldung ausgegeben.

#### 4.4.10 Andere Versuche / Probleme bei der Programmierung

(→) Schö-

In diesem Kapitel werden neben den Prototypen und den festgestellten Problemen während der inkrementellen Entwicklung dieses Teilbereichs auch weitere Versuche und Ansätze innerhalb des Projekts behandelt.

ditsch

#### 4.4.10.1 Kabel Erkennung Version 1

In der ersten Version zur Erkennung des roten Kabels, wurde das mit der Hololens aufgenommene Foto, mit den HauptThread durch jeden Pixel der Textur iteriert und überprüft, ob es hauptsächlich rot ist. Wenn die Rotkomponente des Pixels über einem bestimmten Schwellenwert liegt und die Grün- und Blaukomponenten unterhalb eines bestimmten Schwellenwerts liegen, wird das Pixel auf Rot gesetzt. Andernfalls wird das Pixel auf Schwarz gesetzt.

(→) Schöditsch

```

1 void editTextureV2(Texture2D textureToEdit) {
2     int textureWidth = textureToEdit.width;
3     List<int> yCoordinates = new List<int>();
4     List<int> xCoordinates = new List<int>();
5     for (int y = 0; y < textureToEdit.height; y++) {
6         for (int x = 0; x < textureWidth; x++) {
7             int index = y * textureWidth + x;
8             // ueberprueft ob der gepruefte Pixel rot ist
9             if (pixels[index].r > 0.55f && pixels[index].g < 0.5f && pixels[
index].b < 0.5f) {
10                 yCoordinates.Add(y);
11                 xCoordinates.Add(x);
12             }
13         }
14     }
15     int averageX = getCenterCoordinate(xCoordinates);
16     int averageY = getCenterCoordinate(yCoordinates);
17     /* Fuegt den kleinsten X und den durchschnittlichen Y Punkt als Start Punkt zu
redPixelCoordinates hinzu */
18     redPixelCoordinates.Add(getSideCoordinate(xCoordinates, true, averageY));
19     /* Fuegt den durchschnittlichen X und den durchschnittlichen Y Punkt als
Mittelpunkt Punkt zu redPixelCoordinates hinzu */
20     redPixelCoordinates.Add(new Vector2(averageX, averageY));
21     /* Fuegt den höchsten X und den durchschnittlichen Y Punkt als Mittelpunkt Punkt
zu redPixelCoordinates hinzu */
22     redPixelCoordinates.Add(getSideCoordinate(xCoordinates, false, averageY));
23 }
```

Codeabschnitt 4.21: Erste Version der Kabel Erkennung

Darüber hinaus erfolgt die Speicherung der X- und Y-Koordinaten der roten Pixel. Anschließend wird der Durchschnitt der X- und Y-Koordinaten dieser roten Pixel berechnet. Daraufhin werden drei Gesamtpositionen verwendet, über denen sich das Objekt bewegt. Die erste Position ist diejenige mit der geringsten X-Koordinate und der durchschnittlichen Y-Koordinate aller Y-Koordinaten, die einen roten Pixel aufweisen. Die zweite Position ist der Mittelpunkt zwischen dem Durchschnitt der X-Koordinaten und dem Durchschnitt der Y-Koordinaten. Die dritte Position ist der Endpunkt mit der höchsten X-Koordinate und dem durchschnittlichen Y-Wert. Diese Informationen werden in einer zweidimensionalen Vektorliste (redPixelCoordinates) gespeichert.

Nachteile an diesen Vorgänge sind das diese Funktion mit nur einem Thread durchgeführt wird, das ist nicht besonders effizient und bedeutet längere Ladezeiten. Dazu werden nur drei Positionen genommen welche nicht sehr genau sind, das bedeutet dann dass, das Paket somit nicht wirklich gut über das Kabel bewegt.

#### 4.4.11 Starten und Nutzen der Webseite

Dieser Abschnitt beschreibt den Prozess des Starts und der Nutzung der Webseite gemäß dem Anwendungsfall **??.** Die Webseite spielt eine zentrale Rolle im ersten Anwendungsszenario, dem Nachrichtenaustausch. Ziel ist es, dass zwei Geräte miteinander kommunizieren können. Für eine optimale Benutzererfahrung bedarf es daher einer intuitiven Benutzeroberfläche. Die Entscheidung, ein Web-Interface zu verwenden, basiert auf diesen zwei Faktoren:

→  
HAYLAZ

- **Breiter Geräte-Support:** Web-Interfaces sind auf einer Vielzahl von Geräten nutzbar, darunter Desktop-Computer und Laptops. Diese Vielseitigkeit gewährleistet eine breite Verfügbarkeit und Nutzbarkeit für Benutzer unabhängig von ihrem verwendeten Gerät, Betriebssystem oder der Rechenleistung.
- **Einfache Umsetzung:** Die Entwicklung eines Web-Interfaces gestaltet sich im Vergleich zu anderen Plattformen als verhältnismäßig unkompliziert. Dies liegt daran, dass Webtechnologien weit verbreitet, gut dokumentiert und leicht zugänglich sind.

Die gesamte Anwendung ist bewusst einfach und minimalistisch gehalten. Folglich befindet sich der gesamte Code der Webseite in einer einzigen Datei, *Level1-FrontEnd/index.html*. Im Folgenden wird erläutert, wie diese Datei ausgeführt wird, um die Webseite im lokalen Netzwerk bereitzustellen.

##### 4.4.11.1 Webseite starten

Um die Webseite zu starten, wird ein lokaler Web-Server benötigt. Ein beliebter und einfacher Weg, dies zu erreichen, ist die Verwendung des Node.js basierten HTTP-Daemons namens *http-server*. Dieser HTTP-Server ist leichtgewichtig, einfach einzurichten und erfordert keine komplizierte Konfiguration. Durch die Verwendung von *http-server* kann die Webseite schnell lokal gehostet werden, was besonders praktisch ist, wenn sie während der Entwicklung getestet werden muss.<sup>23</sup>

Eine der einfachsten Möglichkeiten, den *http-server* zu installieren, ist die Verwendung des Node Package Managers (npm). npm ist ein leistungsfähiges Werkzeug, das es Entwicklern ermöglicht, JavaScript-Pakete einfach zu installieren, zu verwalten und zu aktualisieren. Es ist integraler Bestandteil der Node.js-Umgebung und bietet Zugang zu einer Vielzahl von Paketen und Bibliotheken, die von der Community entwickelt wurden. Durch die Installation des *http-server*-Pakets über npm können Entwickler schnell einen lokalen Web-Server einrichten, ohne sich um die Einzelheiten der Serverkonfiguration kümmern zu müssen.

Um den HTTP-Server mithilfe von npm auf einem Windows-Gerät zu installieren, muss ein Befehl in der Eingabeaufforderung (cmd) ausgeführt werden. Dieser Befehl ermöglicht die Installation des Servers auf dem Gerät und lautet wie folgt: *npm install http-server -g*. Durch die Ausführung dieses Befehls wird der http-server global auf dem Windows-Gerät installiert, was bedeutet, dass er von überall im System aus aufgerufen werden kann.

Nach der Installation des Servers kann die Webseite im Ordner, in dem sich die *index.html*-Datei befindet (standardmäßig im *Level1-FrontEnd* Ordner), mit dem Befehl *http-server* und den Flags *-c -1 -cors* gestartet werden.

Die Flags haben folgende Bedeutungen:

Der Flag *-c* legt die Cache-Zeit der Webseite fest; Durch Kombination mit *-1* wird das Caching deaktiviert. Die Option *-cors* aktiviert CORS (Cross-Origin Resource Sharing) auf dem HTTP-Server. Weitere Informationen zu CORS finden Sie im Abschnitt 4.4.11.2.

---

<sup>23</sup>http-server **http-server**

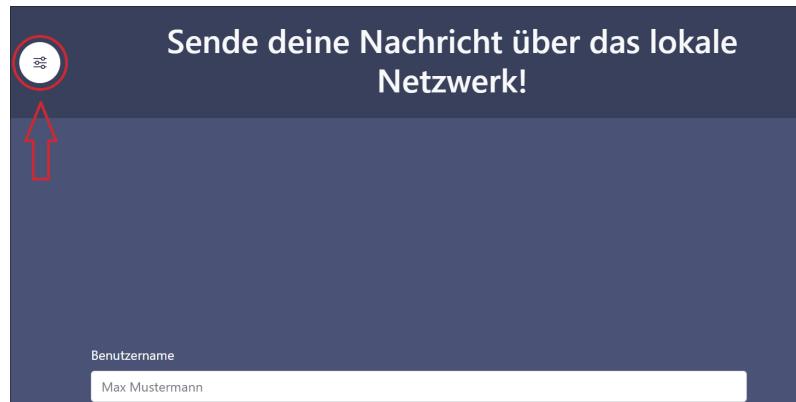
Der vollständige Befehl lautet dann: `http-server -c-1 -cors index.html`.

In dem gegebenen Befehl wird `index.html` als Argument übergeben. Dies bedeutet, dass der `http-server` darauf eingestellt ist, die Datei `index.html` zu bedienen, wenn die Wurzel-URL des Servers aufgerufen wird. Der Server wird den Inhalt dieser Datei dem Client zurückgeben, der die Anfrage gestellt hat. Wie bereits erwähnt ist der gesamte Inhalt der Webseite in dieser Datei enthalten.

Der Server wird standardmäßig auf dem Port 8080 gestartet. Falls ein anderer Port gewünscht ist, kann dieser mit dem Flag `-p` und der gewünschten Portnummer geändert werden.

Nach dem Starten des Webservers kann die Webseite in einem beliebigen Browser über die IP-Adresse des Geräts oder über `localhost` aufgerufen werden. Die IP-Adresse des Geräts kann mit dem Befehl `ipconfig` in der *Eingabeaufforderung (cmd)* abgefragt werden.

Nach dem erfolgreichen Starten der Webseite muss zunächst die IP-Adresse der HoloLens auf der Webseite eingegeben werden. Dies geschieht über den Button in der oberen linken Ecke der Webseite, wie auf der Abbildung 4.17a erkennbar. Die IP-Adresse wird in das Eingabefeld eingetragen und mit dem Button *Verbinden* bestätigt. Erst nachdem die IP-Adresse der AR-Brille eingetragen wurde, können Nachrichten gesendet werden.



(a) Position des Buttons



(b) Fenster der Webseite Einstellungen

Da die Webseite in einem Szenario mit zwei Geräten verwendet wird, muss eingestellt werden, auf welcher Seite des Kabels sich das aktuelle Gerät befindet. Dies ist erforderlich, um die Nachrichten richtig in der virtuellen Umgebung zu simulieren. Dazu wählen Sie in der Combobox, siehe Abb. 4.17b, entweder *Rechtes Gerät* oder *Linkes Gerät*, abhängig von der Position des Trägers der Brille.

Nach der erfolgreichen Konfiguration der Webseite können nun Nachrichten versendet werden.

#### 4.4.11.2 CORS

CORS, oder Cross-Origin Resource Sharing, ist ein Sicherheitsmechanismus, der in modernen Webbrowsern implementiert ist. Seine Hauptaufgabe besteht darin, die Interaktionen zwischen verschiedenen Domains zu regulieren. In der Standardeinstellung erlauben Webbrowser aus Sicherheitsgründen nur Anfragen an Ressourcen, die sich auf derselben Domain befinden, von der die Anfrage stammt. Dies wird als Same-Origin-Policy bezeichnet.

CORS erweitert diese Grundfunktionalität, indem es den Zugriff auf Ressourcen von anderen Domains ermöglicht, vorausgesetzt, der Server, auf dem sich die Ressourcen befinden, erlaubt dies explizit. Dies ermöglicht eine sicherere und kontrolliertere Interaktion zwischen verschiedenen Domains.

In unserem speziellen Fall, da unsere Kommunikation ausschließlich im lokalen Netzwerk stattfindet, ist es nicht unbedingt notwendig, strenge Sicherheitsmaßnahmen zu ergreifen. Dennoch ist es wichtig, dass wir bestimmte Header in allen Antworten einfügen, die von der Brille zurückgeschickt werden. Alle Anfragen werden im *RequestManager.cs* Skript abgearbeitet. Genauere Infos sind im Abschnitt ?? zu finden.

Diese Header welche einzubauen sind lauten:

```
1 context.Response.Headers.Add("Access-Control-Allow-Origin", "*");
2 context.Response.Headers.Add("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
3 context.Response.Headers.Add("Access-Control-Allow-Headers", "Content-Type");
```

- **“Access-Control-Allow-Origin“, “\*”:** Dieser Header gibt an, von welchen Ursprüngen aus die Ressource zugänglich ist. Durch das Einstellen von "\*" wird allen Ursprüngen erlaubt, auf die Ressource zuzugreifen, was als "Wildcard" für alle Ursprünge fungiert.
- **“Access-Control-Allow-Methods“, "GET, POST, OPTIONS":** Dieser Header gibt an, welche HTTP-Methoden für Cross-Origin-Anfragen erlaubt sind. Da wir nur GET, POST und OPTIONS-Anfragen an den Server senden, sind auch nur diese erlaubt, was bedeutet, dass Anfragen mit diesen Methoden von anderen Ursprüngen aus zugelassen werden.
- **“Access-Control-Allow-Headers“, "Content-Type":** Dieser Header gibt an, welche zusätzlichen Header in einer Anfrage aus einem anderen Ursprung erlaubt sind. In diesem Fall ist es nur der Header "Content-Type", was bedeutet, dass Anfragen von anderen Ursprüngen nur diesen spezifischen Header enthalten dürfen.

Durch das Hinzufügen dieser drei Zeilen zu unserer HTTP-Antwort fügen wir CORS-Header hinzu, die Cross-Origin-Anfragen für Ressourcen von unserem Server ermöglichen. Sie definieren auch genau, welche HTTP-Methoden und Header in diesen Anfragen erlaubt sind.

Falls beim Senden der Anfragen ein CORS-Fehler auftritt, liegt das vermutlich an der Konfiguration des Servers. Das Problem kann in Ausnahmefällen (zum Beispiel bei kleineren Testläufen) auf folgende Art und Weise umgangen werden:

- **Deaktivieren im Browser:** Die Deaktivierung von CORS im Browser würde das Problem beheben, birgt jedoch gewisse Risiken. CORS schützt den Benutzer vor schädlichen Webseiten, daher wird davon abgeraten. Die Deaktivierung erfolgt je nach Browser und ist nicht einfach zu finden. Stattdessen sollten die nachfolgende Lösung in Betracht gezogen werden.
- **Verwendung von Plugins:** Es gibt Plugins, die CORS-Header von der Clientseite hinzufügen. Wir haben das Plugin *moesif-origin-cors-change* getestet, das nur für den Chrome-Browser von Google verfügbar ist. Nach der Installation kann das Plugin bei

Bedarf aktiviert/deaktiviert werden.

Es ist wichtig zu beachten, dass beide Lösungsvorschläge gewisse Sicherheitsrisiken bergen und nur im lokalen Netzwerk getestet wurden. Sie sind höchstens für erste Prototypen akzeptabel, wenn überhaupt. Nach der Verwendung unseres Programms sollten beide Konfigurationen wieder rückgängig gemacht werden.

#### 4.4.11.3 Bedienung der Webseite

TODO: Bedienung und korrekte Simulation beschreiben.

#### 4.4.12 Frontend der Webseite

In diesem Abschnitt wird die Struktur und Entwicklung des Frontends der Webseite erläutert. Besonderes Augenmerk wird auf die zugrundeliegenden Entscheidungen bezüglich der Programmiersprachen sowie des Designprozesses gelegt.

→ LAM-  
PEL

Nach mehreren Entwicklungszyklen im ersten Anwendungsszenario wurde entschieden, eine Webseite in das Projekt zu integrieren. Die Funktionalität der Webseite sollte nahtlos in das Szenario integriert werden. Hierzu können Nachrichten und Absender direkt auf der Webseite eingetragen werden, um sie anschließend in die HoloLens-Applikation zu übertragen.

##### 4.4.12.1 Designprozess

Beim Entwurf einer Webseite ist es essentiell, vor der eigentlichen Umsetzung die Farbpalette und das generelle Aussehen der Seite zu konzipieren. Da der Fokus auf der Anwendung auf der HoloLens liegt, wurde bewusst darauf geachtet, der Webseite nicht zu viel Aufmerksamkeit zu schenken. Sie sollte lediglich die grundlegenden Funktionen erfüllen, ohne dabei komplex zu sein. Aus diesem Grund ist die Webseite simpel gestaltet und verfügt über wenige Eingabefelder und wenig Text.

##### Webseitenprototyp

Der Prototyp der Webseite, wie in Abbildung 4.18 dargestellt, gibt einen Überblick über das Frontend-Design. Das Zahnrad links oben öffnet ein Fenster, in dem die IP-Adresse der HoloLens eingegeben werden kann, um Nachrichten korrekt zu versenden. Auf der Hauptseite gibt es Eingabefelder für Benutzername und Nachricht. Der zeitlich sortierte Nachrichtenverlauf wird am rechten Rand angezeigt.

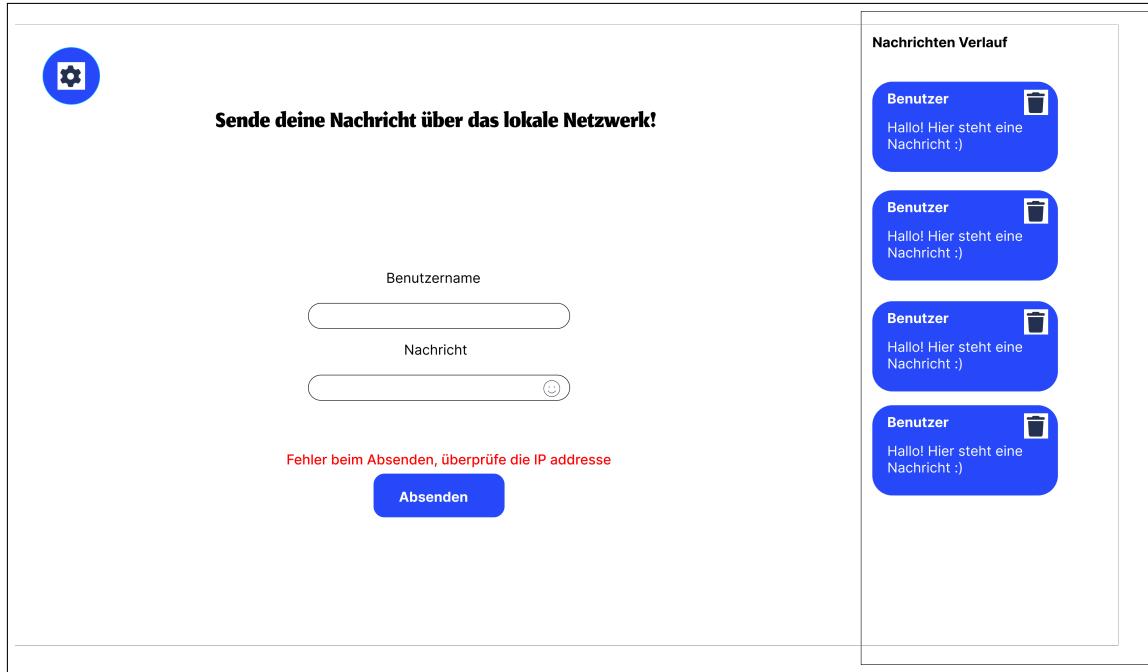


Abbildung 4.18: Prototypische Darstellung der Webseitenansicht aus Benutzersicht

### Ausprogrammierte Webseite

Die endgültige Version der Webseite, wie in Abbildung 4.19 gezeigt, enthält alle gewünschten Funktionen. Die Farbpalette orientiert sich stark am Hauptmenü und den Standardfarben der HoloLens. Es werden verschiedene Blautöne verwendet, um eine konsistente und benutzerfreundliche Benutzererfahrung zu gewährleisten.



Abbildung 4.19: Darstellung der fertigen Webseitenansicht aus Benutzersicht

Die Umsetzung des Designs hat zum Ziel, Verwirrungen zu vermeiden und eine nahtlose

Integration der Webseite in das Gesamtsystem zu gewährleisten.

#### 4.4.12.2 Verwendete Programmiersprachen

Für die Entwicklung der Webseite wurden hauptsächlich die Programmiersprachen HTML, CSS und JavaScript verwendet. Zur Verbesserung des Designs und der Benutzererfahrung wurde außerdem das Framework Bootstrap eingesetzt.

##### HTML (Hypertext Markup Language)

HTML ist die grundlegende Struktursprache des World Wide Web. Es ermöglicht die Erstellung von Webseiten durch die Definition von Strukturelementen wie Überschriften, Absätzen, Listen und Links. HTML verwendet Tags, um den Browsern mitzuteilen, wie die Inhalte einer Webseite strukturiert und präsentiert werden sollen.<sup>24</sup>

##### CSS (Cascading Style Sheets)

CSS ist eine Stylesheet-Sprache, die verwendet wird, um das Aussehen und die Formatierung von HTML-Dokumenten zu steuern. Es ermöglicht die Definition von Farben, Schriftarten, Layouts und anderen visuellen Eigenschaften einer Webseite.<sup>25</sup>

CSS funktioniert durch das Zuweisen von Regeln und Stilen zu HTML-Elementen. Diese Regeln können in einer externen CSS-Datei definiert oder direkt im HTML-Dokument eingebettet werden.

##### JavaScript

JavaScript ist eine dynamische Programmiersprache, die hauptsächlich für die clientseitige Entwicklung von Webanwendungen verwendet wird. Sie ermöglicht die Interaktion mit dem Benutzer, das Ändern von Inhalten in Echtzeit und die Steuerung des Verhaltens einer Webseite.<sup>26</sup>

JavaScript verbessert die Funktionalität einer Webseite, indem es auf Benutzerinteraktionen reagiert, Formulare validiert, Animationen erstellt und vieles mehr.

##### Integration von Bootstrap

Bootstrap ist ein Open-Source-Framework für die Frontend-Entwicklung von Webseiten und Webanwendungen. Es bietet vorgefertigte HTML- und CSS-Vorlagen für Typografie, Formulare, Buttons, Navigation und andere UI-Komponenten.<sup>27</sup>

Bootstrap erleichtert die Erstellung responsiver und ästhetisch ansprechender Webseiten, indem es eine Reihe von vorgefertigten Designelementen und Layoutoptionen bereitstellt.

Die Integration von Bootstrap in eine HTML-Datei erfolgt durch das Einbinden der Bootstrap-Bibliotheksdateien in den <head>-Bereich des HTML-Dokuments. Dies kann entweder über ein CDN (Content Delivery Network) oder durch das Herunterladen und Einbinden der lokalen Bootstrap-Dateien erfolgen.<sup>28</sup>

Anschließend können Bootstrap-Komponenten und -Klassen innerhalb der HTML-Datei verwendet werden, um das Design und die Funktionalität der Webseite zu verbessern.

<sup>24</sup>Mozilla Developer Network [HTML](#)

<sup>25</sup>Mozilla Developer Network [CSS](#)

<sup>26</sup>Mozilla Developer Network [Javascript](#)

<sup>27</sup>Bootstrap Dokumentation [Bootstrap](#)

<sup>28</sup>Bootstrap Dokumentation [CDN-Links](#)

#### 4.4.12.3 Funktionen der Webseite

Die Eingabefelder des Frontends brauchen Javascript Funktionen, um die gewünschten Tätigkeiten durchzuführen und so zu funktionieren wie eigentlich geplant.

```

1 function isNumberKey(evt) {
2     var charCode = (evt.which) ? evt.which : evt.keyCode;
3     if (charCode != 46 && charCode > 31 && (charCode < 48 || charCode > 57)) {
4         return false;
5     }
6     return true;
7 }
```

Codeabschnitt 4.22: Javascript | Überprüfung, ob die Eingabe eine Zahl oder ".ist

Die Funktion überprüft, ob die gedrückte Taste eine Zahlentaste ist. Sie wird auf der Webseite zur Überprüfung der Eingabe von der IP-Adresse genutzt, um sicherzustellen, dass nur Zahlen eingegeben werden können.

Die Funktion nimmt ein Event-Objekt als Parameter entgegen, das Informationen über das Tastaturereignis enthält. Der charCode wird aus dem Event-Objekt extrahiert, der den Unicode-Wert der gedrückten Taste darstellt. Die Funktion überprüft, ob die gedrückte Taste eine Zahl ist (der Unicode-Wert liegt im Bereich von 48 bis 57) oder der Punkt ist (der Unicode-Wert ist 46). Falls die gedrückte Taste im Bereich von 48 bis 57 liegt oder der Punkt ist, gibt die Funktion true zurück, was bedeutet, dass die Eingabe akzeptiert wird. Andernfalls gibt die Funktion false zurück, was bedeutet, dass die Eingabe nicht akzeptiert wird.

```

1 async function validateAndSendMessage() {
2     var username = document.getElementById("username").value.trim();
3     var message = document.getElementById("message").value.trim();
4
5     if (username !== '' && message !== '') {
6         const messageSent = await sendMessage(username, message);
7
8         if (!messageSent) {
9             alert("Message could not be sent.");
10        }
11    } else {
12        alert("Please fill in both fields!");
13    }
14 }
```

Codeabschnitt 4.23: Javascript | Validierung und Senden der Message

Die Funktion validiert die Eingaben in den Feldern für Benutzername und Nachricht und sendet die Nachricht nur ab, wenn beide Felder ausgefüllt sind.

Zunächst ruft die Funktion die Werte der Formularfelder für Benutzername und Nachricht ab und entfernt führende und abschließende Leerzeichen mit der trim()-Methode. Anschließend überprüft sie, ob sowohl der Benutzername als auch die Nachricht nicht leer sind. Wenn beide Felder nicht leer sind, wird die Funktion sendMessage(username, message) aufgerufen, um die Nachricht zu senden. Wenn die Nachricht erfolgreich gesendet wurde, wird nichts weiter unternommen. Andernfalls wird eine Warnmeldung angezeigt, dass die Nachricht nicht gesendet werden konnte. Sollte eines der Felder leer sein, wird dem Benutzer eine Warnmeldung angezeigt, die ihn darauf hinweist, beide Felder auszufüllen.

#### 4.4.13 Backend der Webseite

→  
HAYLAZ

Im vorherigen Abschnitt (??) wurde der Designprozess der Webseite erläutert. Im Folgenden wird dieses Thema durch eine Beschreibung des Backend-Bereichs erweitert.

Die Webseite fungiert als Schnittstelle für die Kommunikation zwischen den beiden Laptops, die für das Szenario benötigt werden. Diese Kommunikation erfolgt über die Hololens, die gewissermaßen als "Mittelsmann" fungiert und die Datenübertragung ermöglicht.

#### 4.4.13.1 Protokoll zur Datenübertragung

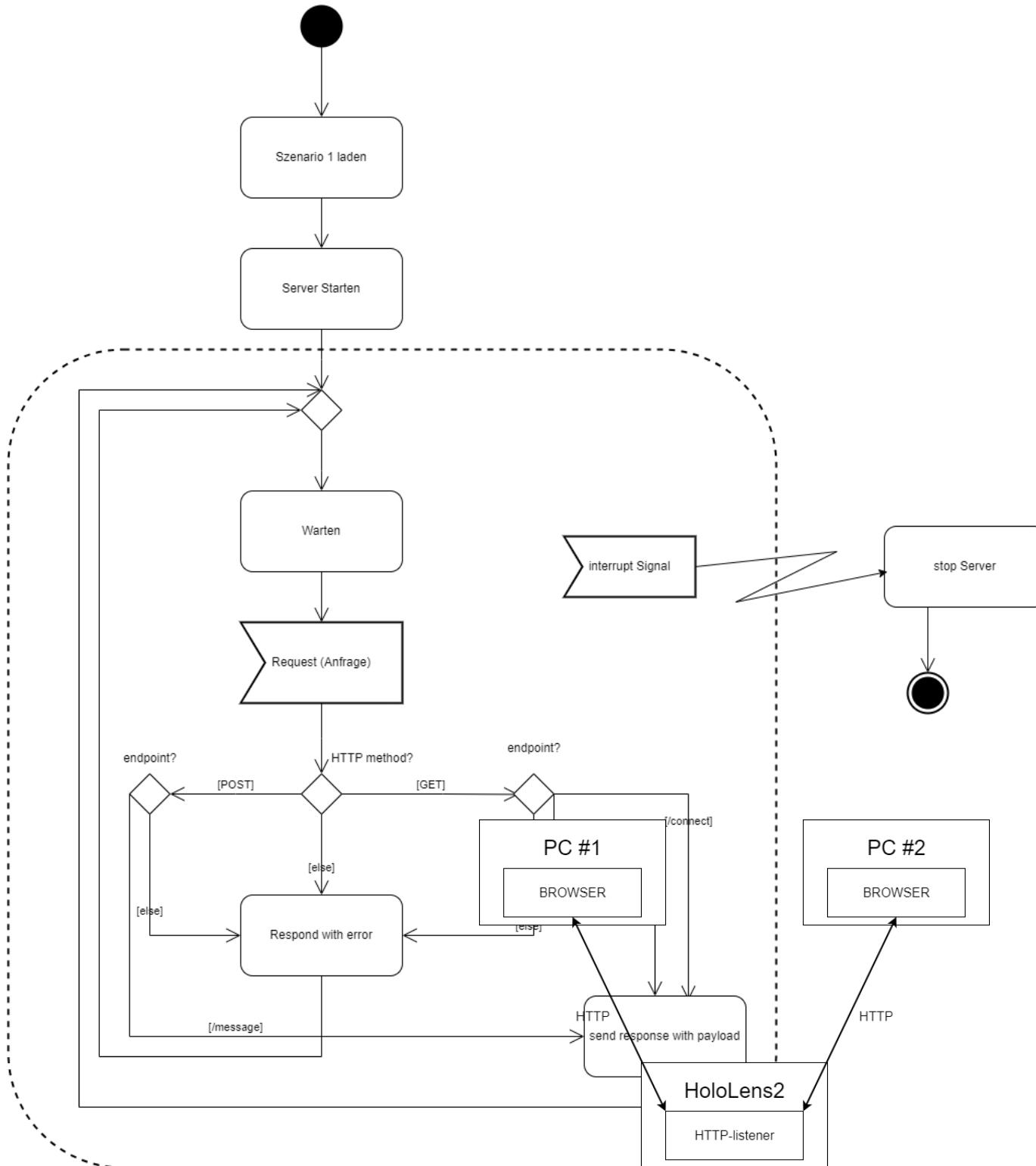


Abbildung 4.20: Aktivitätsdiagramm des Servers

Abbildung 4.21: Kommunikationsstruktur

In Abbildung 4.20 ist der Ablauf aller möglichen Szenarien dargestellt, wie eine solche Datenübertragung zustande kommen kann. Im Folgenden werden die einzelnen Schritte und ihre Funktionen beschrieben:

- **Start des Servers:** Auf der Hololens befindet sich ein Game-Objekt namens *httpListener* mit einem Skript namens *RequestManager.cs*. Beim Laden des ersten Szenarios wird ein HTTP-Listener in einem eigenen Thread gestartet. Erst wenn dieser läuft, kann auf Anfragen der Webseite reagiert werden. Eine ausführlichere Beschreibung der Funktionalität des Servers findet sich im Abschnitt 4.4.13.2.
- **Verbindungsaufbau:** Bevor ein Nachrichtenpaket gesendet werden kann, muss die IP-Adresse der Hololens registriert werden. Obwohl die Verbindungstestfunktion nicht zwingend erforderlich ist, um Nachrichten an diese IP zu senden, kann ein visuelles Feedback dem Benutzer anzeigen, ob die eingegebene IP-Adresse gültig ist und ob der Server erreichbar ist. Dies trägt zur Benutzerfreundlichkeit bei, indem es dem Benutzer ermöglicht, leichter zu erkennen, ob alles ordnungsgemäß funktioniert oder ob möglicherweise Probleme auftreten.
- **Empfangen von Nachrichten:** Auf der Hololens wird eine Liste aktueller Nachrichten geführt. Beide Laptops rufen alle fünf Sekunden mit einem *HTTP GET-Request* an den Endpunkt */get-messages* diese Liste ab. Dies ermöglicht eine kontinuierliche Aktualisierung des Nachrichtenverlaufs (??, ??) und zeigt dem Benutzer stets die neuesten Nachrichten an.
- **Versenden von Nachrichten:** Um eine Nachricht von einem Laptop auf den anderen zu senden, muss ein *http post request* an den Endpoint */message* mit dem Benutzernamen und der Nachricht als Payload gesendet werden.

Es mag so aussehen, als ob das Senden von Requests für den Benutzer möglicherweise als unmöglich erscheint, aber im Abschnitt ?? wird gezeigt, dass dies alles durch eine benutzerfreundliche Benutzeroberfläche (UI) erreicht wird.

Es ist festzuhalten, dass die beiden Laptops niemals direkt miteinander kommunizieren, sondern alle Nachrichten über den Server, die Hololens, laufen.

#### 4.4.13.2 Bearbeitung von Anfragen

Die Bearbeitung von Anfragen erfolgt im Skript *RequestManager.cs*. Der Listener wird in einem eigenen Thread gestartet, da das Warten auf Nachrichten den Thread, auf dem der Listener läuft, blockiert. Nach dem Erhalt einer Anfrage wird ein neues *HttpListenerContext*-Objekt erstellt, und die Verarbeitung der Anfrage erfolgt in dem Thread, der diesen Kontext bearbeitet. Wenn eine HTTP-Anfrage empfangen wird, ist der erste Schritt, zu überprüfen, um welche Art von Anfrage es sich handelt. Ein POST und ein GET werden unterschiedlich verarbeitet. Bei Verwendung einer anderen HTTP-Methode wird ein *Method Not Allowed* Fehler zurückgesendet. Bei der Wahl des Endpunkts passiert dasselbe; jede nicht unterstützte Anfrage auf einen Endpunkt wird mit einem *Not Found* Fehler beantwortet.

Es werden in unserem Fall nur drei Endpunkte unterstützt:

- **Testen der Verbindung auf /connect**
- **Empfangen von Nachrichten auf /get-messages**
- **Senden einer Nachricht auf /message**

Um erfolgreich eine Anfrage an den Server zu senden, muss sie an die folgende Adresse gehen: <http://IP-Adresse der Hololens:Port/Endpoint>. Die IP-Adresse der Hololens

wird manuell auf der Webseite eingegeben. Der Listener läuft standardmäßig auf dem Port 9090.

(Die erlaubten Endpunkte sind wie oben aufgeführt fallbasiert gelistet.)

#### 4.4.14 Object Tracking

Durch Verwendung von bereitgestellten Technologien der HoloLens2 werden die zwei PCs → SCHO-DITSCH und das Kabel getracked.

#### 4.4.15 Kurvenberechnung

Durch Berechnung der Kurve wird das Kabel als Kurve gespeichert und dadurch wird es ermöglicht, dass das 3D-Ping-Paket über diese Kurve von einem PC zum anderen läuft.

### 4.5 Knapsack Problem Szenario

Im zweiten Anwendungsszenario dieser Applikation liegt der Fokus auf dem bekannten Problem des Knapsack-Problems. Ziel dieses Szenarios ist es, dieses Informatikproblem mithilfe von Augmented Reality (AR) visuell und spielerisch darzustellen. Das Knapsack-Problem ist ein klassisches Problem der Informatik, das nicht nur an der Höheren Technischen Lehranstalt (HTL) vermittelt wird, sondern auch von Schülern eigenständig programmiert werden soll. Dieser Ansatz dient dazu, den Anwendern einen Einblick in die Informatik zu geben und möglicherweise Interessen für den Bereich zu wecken. → SKREPEK

Das Unity-Anwendungsszenario für die HoloLens 2 bietet insgesamt eine interaktive und visuelle Erfahrung, bei der der Benutzer nicht nur das Knapsack-Problem verstehen, sondern auch praktisch anwenden kann. Die Integration von AR ermöglicht es dem Benutzer, das Problem in einer realen Umgebung zu erleben und die Optimierungsmöglichkeiten direkt zu visualisieren und zu manipulieren. Diese immersive Herangehensweise kann dazu beitragen, das Verständnis des Knapsack-Problems zu vertiefen und das Interesse an der Informatik zu fördern.

In diesem Abschnitt werden alle *GameObjects*, *Komponenten*, *Scripts* und *Klassen* genauer erklärt, die in der Unity-Szene des Knapsack-Problems verwendet werden, um dieses zu realisieren.

#### 4.5.1 Knapsack-Problem Szenario Hirarchie

Für einen sicheren und funktionalen Ablauf ist der Aufbau beziehungsweise die Hirarchie der Unity Szene von großer Bedeutung. In diesem Abschnitt wird anhand der Abbildung 4.22 verdeutlich wie die Unity Szene für die Implementierung des zweiten Anwendungsszenarios für das Knapsack-Problem aufgebaut ist und es wird zusätzlich kurz darauf eingegangen für was welches Objekt steht und welche Aufgabe dieses trägt. → SKREPEK

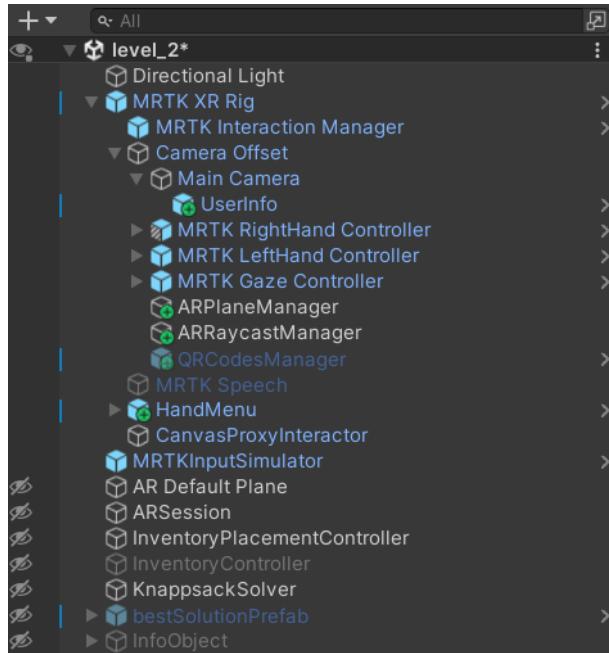


Abbildung 4.22: Knapsack-Problem Szenario Hirarchie im Unity Editor.

In dieser Abbildung ist die Hirarchie un der Inhalts des Knapsack-Problem Szenarios zu sehen. Anhand desen ist zu erkennen, dass diese Unity Szene aus vielen wichtigen Komponenten besteht, die alle zusammenspielen, um das gewünschte Ergebnis zu erzielen. Darunter sind folgende Objekte:

- **level-2:** Ist die Unity Szene selbst, in der alle Game-Objekte enthalten sind.
- **MRTK XR Rig:** Grundbaustein für das Entwickeln einer XR-Applikation. Enthält wichtig Komponenten wie Controller für das tracken der Hände, für den User Gaze, etc.
- **UserInfo:** Text Objekt, dass in der *main camera* liegt, damit dieses ständig im Sichtfeld des Benutzers liegt.
- **Managers:** Die drei Manager *ARPlaneManager*, *ARRaycastManager* und *QRCodeManager* sind wichtiger Bestandteil um ARPlanes zu scannen, Raycasting durchzuführen und QRCodes zu erkennen.
- **HandMenu:** Knopf um in das Hauptmenu-Szenario zurück zu gelangen.
- **AR Default Plane:** Ist das Grund-Objekt für das markieren von erkannten AR-Planes.
- **ARSession:** Ist die Hauptkomponente, die die AR-Funktionalitäten steuert und koordiniert.
- **InventoryPlacementController:** Game Objekt, welches das platzieren des Inventar-Objekts handhabt.
- **InventoryController:** Game Objekt, welches das erkennen von neuen Objekten innerhalb des Inventar-Objekts handhabt.
- **KnapsackSolver:** Game Objekt, welches den Knapsack Algorithmus implementiert und das eigene Inventar berechnet.
- **bestSolutionPrefab:** Prefab, welches die perfekte Lösung wiederspiegelt.
- **InfoObject:** Dient der visualisierung von berechneten Werten und Fehlermeldungen.

In der Abbildung ist ausßredem zu sehen, dass ein Paar Game Objekte ausgegraut und nicht ausgegraut sind und, dass neben ein paar Game Objekten ein durchgestrichenes Auge zu sehen ist. Wenn ein Game Objekt im Unity Editor ausgegraut ist bedeutet das, dass dieses GameObjekt und somit alle angehängten Scripts von diesem Game Objekt deaktiviert sind. Das bedeutet, dass dieses Game Objekt samt allen Scripts zu Szenenbeginn nicht aufgerufen und somit auch nicht ausgeführt werden. Nicht ausgegraute Game Objekte widerum sind daher genau das Gegenteil. Das beudetet, dass das Game Objekt selbst samt allen angehängten Scripts alle aktiviert sind und somit zu Szenenbeginn aufgerufen und ausgeführt werden.

Wenn neben einem Game Objekt das durchgestrichene Auge zu sehen ist bedeutet das nur, dass dieses Game Objekt im Unity Editor nicht zu sehen ist. Andererseits, wenn kein Zeichen neben dem Game Objekt zu sehen ist, ist dieses Objekt im Unity Editor sichtbar. Dies dient dazu, dass falls in der Unity Szene viele Game Objekte vorhanden sind, dass man diejenige ausblendet die nicht im Editor sichtbar sein müssen wie zum Beispiel Tesh Meshes oder Lables.

#### 4.5.2 Nutzung von QR-Codes

Im vorherigen Abschnitt wurde bereits erwähnt, dass QR-Codes in diesem Level verwendet werden, um verschiedene Elemente zu repräsentieren. Diese QR-Codes spielen eine entscheidende Rolle, indem sie dazu dienen, vielfältige Informationen zu den einzelnen Gegenständen, die für das Knapsack-Inventar benötigt werden, zu speichern und sie anschließend in einer virtuellen Umgebung abzubilden. Im folgenden Abschnitt möchten wir näher darauf eingehen, wie genau diese QR-Codes generiert werden und welchen Zweck sie innerhalb der Augmented Reality (AR)-Applikation erfüllen. Dabei wird insbesondere betrachtet, wie die Codes generiert werden und auf welche Weise sie innerhalb der Anwendung zur Interaktion mit den realen Objekten verwendet werden.

→  
HAYLAZ

##### 4.5.2.1 Struktur und Inhalt eines QR-Codes

Die Informationen, die in einem QR-Code gespeichert werden können, sind begrenzt. In unserem Anwendungsfall wird lediglich eine einzelne Zahl im Bereich von 1 bis 11 abgespeichert. Diese Zahlen repräsentieren die 11 verschiedenen Modelle, die wir unterscheiden möchten. Da nur eine Zahl gespeichert wird, genügt ein QR-Code der Größe 21x21 Module (Version 1). Die geringe Anzahl von Modulen ermöglicht eine schnellere Erkennung, auch über größere Distanzen.

TODO: Testen und Grafik erstellen um zu zeigen das es eine Rolle spielt welche version wir verwenden + wie groß die sind

Die zugehörigen Zahlen werden in der Software, genauer gesagt in der Klasse *QRItem.cs*, mit Informationen verknüpft. Im folgenden Codeausschnitt wird dies verdeutlicht:

```

1 public class QRItem
2 {
3     public struct QRData
4     {
5         public int id;
6         public string name;
7         public Vector3 position;
8         public int weight;
9         public int value;
10    }
11
12    public QRData qrData;

```

```

13
14     public Dictionary<int, QRData> items = new Dictionary<int, QRData>()
15     {
16         {1, new QRData { id = 1, name = "Laptop", weight = 70, value = 100 }},
17         {2, new QRData { id = 2, name = "Router", weight = 25, value = 50 }},
18         {3, new QRData { id = 3, name = "Maus", weight = 20, value = 30 }},
19         // ...
20         {11, new QRData { id = 11, name = "Handy", weight = 30, value = 100 }}
21     };
22
23     public QRItem(int id)
24     {
25         items.TryGetValue(id, out qrData);
26     }
27 }
```

In dieser Klasse wird ein Dictionary verwendet, das den Gegenständen die folgenden Informationen zuordnet:

- **Id:** Die numerische Kennung im QR-Code.
- **Name:** Die Bezeichnung des Gegenstandes, das dieser QR-Code repräsentiert.
- **Position:** Die Position des Gegenstandes in der virtuellen Umgebung.
- **Weight:** Das Gewicht des Gegenstandes.
- **Value:** Der Wert des Gegenstandes.

Diese Informationen spielen eine wesentliche Rolle in der weiteren Anwendung des Knapsack-Algorithmus.

#### 4.5.2.2 QR-Code-Tracking

Das Tracking der QR-Codes erfolgt mithilfe des *QRCodeManager.cs* Skripts. Dieses Klasse ist ein Singleton, das die Erkennung und Verfolgung der QR-Codes steuert.

Nach der Erkennung eines QR-Codes erfolgen eine Reihe von Schritten, um diese Informationen zu speichern, verarbeiten und zuletzt darzustellen. Hier eine kurze Übersicht:

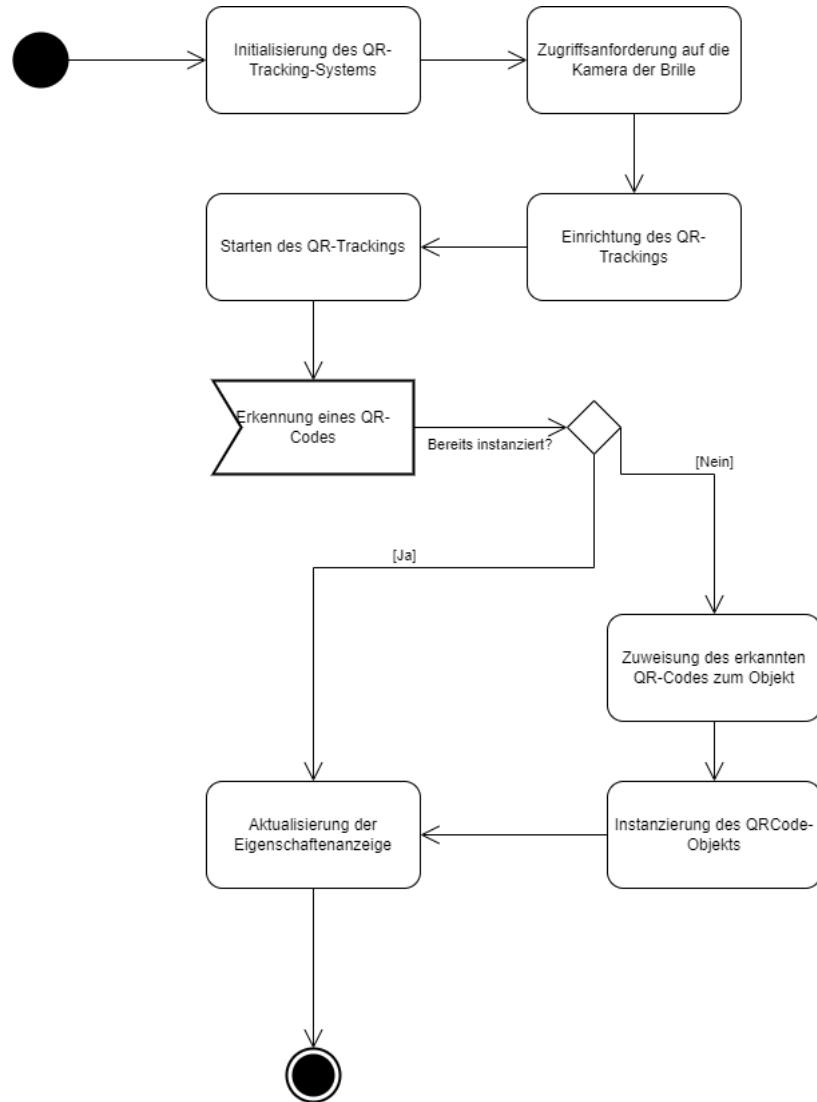


Abbildung 4.23: Ablaufdiagramm des QR-Code-Trackings.

- **Initialisierung des QR-Tracking-Systems:** Um das QR-Tracking-System zu aktivieren, wird die erforderliche Komponente gestartet, um die Erkennung von QR-Codes zu ermöglichen. Diese Komponente ist das QRCodeManager-Game-Objekt, das weitere Game-Objekte instanziert, wie beispielsweise den *QRCodeVisualizer*, der für die Visualisierung benötigt wird (siehe Abschnitt ??). Dabei werden Tracking-Algorithmen gestartet, die für die Lokalisierung und Identifizierung von QR-Codes in der Umgebung benötigt werden. Diese Initialisierung erfolgt zu Beginn der Anwendungsaktivität.
- **Zugriffsanforderung auf die Kamera der Brille:** TODO: Hier ein Bild von "Camera access needed" einfügen Für das QR-Tracking wird der Zugriff auf die Kamera der Augmented-Reality-Brille benötigt. Eine Zugriffsanfrage wird gestellt, um die notwendigen Berechtigungen zu erhalten. Dieser Schritt ist entscheidend, um visuelle Daten von der Kamera zu erhalten und QR-Codes in der physischen Umgebung zu erkennen.
- **Einrichtung des QR-Trackings:** Die Einrichtung des QR-Trackings umfasst die Konfiguration von Parametern und Einstellungen, die für die korrekte Funktion des

Tracking-Systems erforderlich sind. Dazu gehören Kalibrierungsschritte, die Anpassung an die Umgebung und die Festlegung von Erkennungsbereichen. Eine ordnungsgemäße Einrichtung gewährleistet eine zuverlässige und präzise Erkennung von QR-Codes.

- **Starten des QR-Trackings:** Das System sucht aktiv nach QR-Codes in der Umgebung, um sie zu erkennen. Die Kamera erfasst kontinuierlich visuelle Daten, welche von den Tracking-Algorithmen analysiert werden, um QR-Codes zu identifizieren. Das Starten des Trackings markiert den Beginn des fortlaufenden Erkennungsprozesses.
- **Erkennung eines QR-Codes (Event):** Sobald die Kamera einen QR-Code erfasst, wird dieser automatisch erkannt. Dieses Ereignis signalisiert die erfolgreiche Erkennung eines QR-Codes und liefert Informationen über den erkannten QR-Code, wie seine Daten und Position. Eine detaillierte Beschreibung des genauen Ablaufs der Erkennung und Positionsbestimmung ist im Abschnitt ?? zu finden.
- **Zuweisung des erkannten QR-Codes zum Objekt:** Nach der Erkennung wird überprüft, ob der erkannte QR-Code bereits in der Anwendung registriert ist. Falls dies der Fall ist, wird der erkannte QR-Code einem entsprechenden Objekt in der virtuellen Umgebung zugeordnet.
- **Instanzierung des QRCode-Objekts:** Nach dem Scannen eines QR-Codes wird ein QR-Code-Prefab erzeugt und in der Szene platziert.

Dieses Objekt dient als Repräsentation des gescannten QR-Codes und wird als QR-Objekt bezeichnet. Es enthält visuelle Darstellungen, Interaktionsmöglichkeiten und weitere relevante Informationen über den zugehörigen QR-Code. Die Instanziierung ermöglicht eine nahtlose Integration des gescannten QR-Codes in die virtuelle Umgebung. Weitere Informationen zur Visualisierung von QR-Codes finden Sie in Abschnitt ??.

- **Aktualisierung der Eigenschaftenanzeige:** Die Aktualisierung der Eigenschaftsanzeige dient dazu, visuelle und informative Darstellungen des erkannten QR-Codes zu aktualisieren. Hierbei werden die Position, Größe, visuelle Darstellung und zugehörige Informationen des QR-Codes aktualisiert. Durch die Aktualisierung wird sichergestellt, dass die Benutzer stets die neuesten Informationen über das durch den QR-Code repräsentierte Objekt erhalten.

#### 4.5.2.3 Interaktion mit QR-Codes



Abbildung 4.24: Bauklötzte mit QR-Codes

TODO: neues besseres Bild einfügen

Durch die Verwendung der HoloLens können wir dem Benutzer eine visuelle Darstellung einer virtuellen Welt bieten. Um eine Verbindung zwischen der realen und der virtuellen Welt herzustellen, nutzen wir QR-Codes. Diese dienen in der realen Welt als Repräsentationen jener Objekte, die wir in der virtuellen Welt darstellen möchten.

Wie in 4.24 zu sehen ist, sind die Bauklötzte mit QR-Codes versehen. Diese Bauklötzte repräsentieren die Gegenstände, die der Benutzer in sein Inventar aufnehmen kann. Wenn der Benutzer einen Bauklotz aufhebt und ihn der HoloLens nähert, wird der QR-Code gescannt. Dadurch werden Informationen wie das zugehörige 3D-Modell, der Wert, das Gewicht und der Name des Gegenstands angezeigt. Auf diese Weise können wir eine nahtlose Verbindung zwischen der realen und der virtuellen Welt herstellen.

Dem Benutzer wird die Möglichkeit geboten, die Bauklötzte physisch zu berühren, aufzuheben und zu fühlen. Diese sensorische Erfahrung trägt dazu bei, die Immersion des Benutzers zu verbessern und ihm ein besseres Verständnis der virtuellen Welt zu ermöglichen. Dazu muss laufend die Position der Bauklötzte und somit auch der QR-Codes getrackt werden, wie im Abschnitt 4.5.2.4 beschrieben.

#### 4.5.2.4 Positionsbestimmung von QR-Codes

Um die räumliche Position eines QR-Codes exakt zu bestimmen, ist ein spezieller Prozess notwendig. Dieser Schritt ist entscheidend, um physische QR-Codes in der virtuellen Realität darstellen und mit ihnen interagieren zu können. Die Positionierung der QR-Codes erfolgt mit Hilfe einer Klasse namens *QRWatcher*, die im SDK-Plugin *Microsoft.MixedReality.QR* enthalten ist.

Die *QRWatcher*-Klasse ermöglicht die Erkennung und Verfolgung von QR-Codes in der physischen Umgebung. Sie stellt eine Vielzahl von Funktionen und Ereignissen zur Verfügung, die für die Weiterverarbeitung von QR-Codes notwendig sind. Die Verarbeitung erfolgt in den folgenden Schritten:

- **Kamera-Feed abrufen:** Nach der Initialisierung der *QRWatcher*-Klasse und dem Start des Trackings wird zunächst der Kamerafeed der HoloLens abgerufen.
- **Bildverarbeitung:** Der Kamerafeed wird mit den Algorithmen und Methoden der *QRWatcher*-Klasse verarbeitet, um QR-Codes zu identifizieren.
- **Erkennung eines QR-Codes:** Sobald ein QR-Code erkannt wird, wird ein Ereignis ausgelöst. Dieses Ereignis enthält Informationen über den erkannten QR-Code, wie dessen Daten und Position, und kann von anderen Klassen subskribiert werden.
- **Übersetzung der Positionen:** Die Positionen der erkannten QR-Codes werden in die Weltkoordinaten der HoloLens übersetzt, um sie in der virtuellen Welt zu verankern. Dazu wird ein *Spatial Graph Node*(4.5.2.6) erstellt und die Position des QR-Codes an diesen gebunden. **Zuordnung des QR-Codes:** Nach der Erkennung wird überprüft, ob der erkannte QR-Code bereits in der Anwendung registriert ist. Falls ja, werden die Positionsinformationen des QR-Codes aktualisiert, um mögliche Bewegungen in der physischen Umgebung zu berücksichtigen. Anschließend wird ein neuer *Spatial Graph Node* an dieser aktualisierten Position platziert. Dadurch wird sichergestellt, dass das virtuelle Objekt weiterhin korrekt mit dem QR-Code verbunden ist und an der aktuellen Position in der virtuellen Umgebung angezeigt wird.

Nach Abschluss dieser Schritte sind die Positionen der vorhandenen QR-Codes bestimmt und können im nächsten Schritt in der virtuellen Welt dargestellt werden.

#### 4.5.2.5 Visualisierung von QR-Codes

TODO: Bild von einem QR Prefab einfügen Nachdem die genaue Platzierung der QR-Codes bestimmt wurde, muss ihre Visualisierung in der virtuellen Welt umgesetzt werden. Hierfür nutzen wir die Funktionalität von Unity Prefabs, die die Erstellung visueller Präsentationen der QR-Codes und ihre nahtlose Integration in die virtuelle Umgebung ermöglichen.

Die Klasse *QRWatcher* kann drei Events auslösen: das Erkennen eines neuen QR-Codes, das Erkennen eines bereits bekannten QR-Codes und das Löschen eines QR-Codes.

Die *Visualizer*-Klasse reagiert auf alle diese Events. Beim erstmaligen Erkennen eines QR-Codes wird ein QR-Prefab initialisiert. Bei erneutem Erkennen werden die Positionsdaten aktualisiert, und beim Entfernen wird entsprechend der QR-Code-Prefab aus der Szene entfernt.

Jedes Prefab enthält alle relevanten Informationen für die Darstellung des QR-Codes, einschließlich des 3D-Modells, des Namens, des Werts und anderer Daten, die für Interaktionen innerhalb der virtuellen Umgebung erforderlich sind. Um die Funktionalität des QR-Codes sicherzustellen, wird dem Prefab das Skript *QRCodes.cs* zugewiesen, das die

visuelle Darstellung und alle damit verbundenen Interaktionen steuert.

TODO: Bild Aufbau Prefabs

In diesem Abschnitt werden die Eigenschaften eines einzelnen QR-Codes erläutert. Wie zu erkennen ist, verfügt dieses Prefab über nur wenige Eigenschaften, nämlich:

- **Canvas:** Der Canvas enthält drei Labels: "Name", "Gewicht" und "Wert", welche die für den Benutzer relevanten Eigenschaften eines Gegenstandes anzeigen.
- **GameObjects von 1 bis 11:** Diese GameObjects repräsentieren die 3D-Modelle, die der QR-Code darstellen soll. Weitere Details zu den 3D-Modellen sind im Abschnitt ?? zu finden. Jedes Prefab enthält die Modelle aller möglichen Items, die angezeigt werden können. Die Nummerierung erleichtert die Iteration durch diese Modelle in der Szene, da die QR-Codes eine eindeutige ID haben und mit einem numerischen Namen leichter darauf zugegriffen werden kann.

```

1 void Start()
2 {
3     //... weiterer Code
4
5     // Die physikalische Größe des QR-Codes wird abgerufen; Wird für Berechnungen in
6     // der UpdatePropertiesDisplay() Funktion benötigt
7     PhysicalSize = qrCode.PhysicalSideLength;
8
9     try
10    {
11        // (1) Ein neues QRItem wird erstellt und mit den Daten des gescannten
12        // QR-Codes initialisiert
13        item = new QRItem(int.Parse(qrCode.Data));
14
15        // (2) Das entsprechende 3D-Modell wird anhand der ID des QR-Items gefunden
16        model = gameObject.transform.Find(item.qrData.id.ToString()).gameObject;
17
18        // (2) Falls ein 3d-Modell gefunden wurde wird sie in der AR-Umgebung sichtbar
19        // gesetzt
20        model.SetActive(model != null);
21    }
22    catch (System.Exception e)
23    {
24        // Fehlerbehandlung, falls das Parsing der QR-Code-Daten fehlschlägt
25        Debug.LogError("Error parsing QR Code data: " + e.Message);
26    }
27
28    // (3)Labels werden mit den entsprechenden Werten befüllt
29    setLabels();
30 }
```

Codeabschnitt 4.24: Instanzierung eines QR-Prefabs

Nach der Instanzierung eines QR-Prefabs wird der folgende Codeabschnitt (4.24) ausgeführt. Das Skript *QRCode.cs*, in dem sich dieser Abschnitt befindet, erhält bei der Instanzierung lediglich den Inhalt des gescannten QR-Codes übermittelt, nämlich die ID. (1) Anhand dieser ID werden aus einer Liste möglicher Objekte die weiteren erforderlichen Informationen entnommen und innerhalb der Klasse gespeichert. (2) Zudem wird das entsprechende 3D-Modell in die Szene eingefügt. (3) Zuletzt werden die Labels mit den richtigen Werten (Name, Gewicht, Wert) befüllt, 4.25 ist der genauere Codeabschnitt. Diese Aktionen werden bei jeder Instanzierung nur einmal ausgeführt. Im Gegensatz dazu wird die folgende Funktion (4.26) bei jeder Frame aktualisierung der Brille ausgeführt und befindet sich daher in der *Update()* Funktion. Weitere Informationen zu den Lebenszyklusmethoden befinden sich in diesem abschnitt: 4.1.2.7

```

1 private void setLabels()
2 {
3     //Das GameObject mit den Labels wird gespeichert
4     labels = gameObject.transform.Find("QRLabeleds").gameObject;
5
6     try
7     {
8         //Der Canvas indem sich die Labels befinden wird gespeichert
9         Transform canvas = labels.transform.Find("Canvas");
10
11        //Die Text-Komponenten indem sich der anzugebende Text befindet wird
12        //gespeichert
13        TextMeshProUGUI nameLabel = canvas.Find("nameLabel").gameObject.GetComponent<
14        TextMeshProUGUI>();
15        TextMeshProUGUI valueLabel = canvas.Find("valueLabel").gameObject.GetComponent<
16        TextMeshProUGUI>();
17        TextMeshProUGUI weightLabel = canvas.Find("weightLabel").gameObject.
18        GetComponent<TextMeshProUGUI>();
19
20        // Die Texte der Labels werden mit den entsprechenden Daten des QR-Items
21        aktualisiert
22        nameLabel.text = "Name: " + item.qrData.name;
23        valueLabel.text = "Wert: " + item.qrData.value.ToString();
24        weightLabel.text = "Gewicht: " + item.qrData.weight.ToString();
25
26    }
27    catch (System.Exception e)
28    {
29        // Fehlerbehandlung, falls beim abfragen der Modelle etwas schief läuft
30        Debug.LogError("Error setting labels: " + e.Message);
31    }
32 }

```

Codeabschnitt 4.25: Setzen der Labels

Dieser Code ist dafür verantwortlich, die Textlabels im Canvas entsprechend mit den Daten des gescannten QR-Codes zu aktualisieren. Zuerst wird das GameObject "QRLabeleds" gefunden, das die Labels enthält. Dann werden die Textkomponenten innerhalb des Canvas-GameObjects gesucht und mit den entsprechenden Daten aktualisiert. Dieser Prozess wird jedes Mal durchgeführt, wenn ein neuer QR-Code gescannt und instanziert wird.

```

1 void UpdatePropertiesDisplay()
2 {
3     if (qrCode != null && lastTimeStamp != qrCode.SystemRelativeLastDetectedTime.Ticks
4     )
5     {
6         PhysicalSize = qrCode.PhysicalSideLength;
7
8         item.qrData.position = new Vector3(PhysicalSize / 2.0f, PhysicalSize / 2.0f,
9         0.0f);
10
11         labels.transform.localPosition = new Vector3(item.qrData.position.x + 0.05f,
12         item.qrData.position.y, item.qrData.position.z);
13
14         labels.transform.localScale = new Vector3(PhysicalSize * 0.02f, PhysicalSize *
15         0.02f, 0.005f);
16     }
17 }

```

```

12     model.transform.localPosition = item.qrData.position;
13
14     lastTimeStamp = qrCode.SystemRelativeLastDetectedTime.Ticks;
15 }
16
17 }
```

Codeabschnitt 4.26: Aktualisieren der Position

Die Funktionsweise der Positionsbestimmung ist im Abschnitt (4.5.2.4) beschrieben. Im Codeabschnitt 4.26 werden die bereitgestellten Positionsdaten verwendet, um die Position der Modelle zu aktualisieren. Hier ist eine schrittweise Erläuterung dessen, was innerhalb der Funktion passiert:

- Zunächst wird überprüft, ob der QR-Code nicht null ist und ob sich die Zeitmarke des letzten erkannten QR-Codes seit dem letzten Mal geändert hat. Diese Überprüfung stellt sicher, dass die Aktualisierung der Anzeige nur erfolgt, wenn sich tatsächlich etwas geändert hat, was zur Effizienz und Leistung der Anwendung beiträgt.
- Die Position des QR-Codes (*item.qrData.position*) wird so festgelegt, dass er sich in seinem lokalen Koordinatensystem zentriert. Dazu wird die Position auf (*PhysicalSize / 2.0f, PhysicalSize / 2.0f, 0.0f*) gesetzt, wobei *PhysicalSize* die Größe des QR-Codes ist. Diese Festlegung ermöglicht es, dass der QR-Code unabhängig von seiner Größe korrekt positioniert wird.
- Die Position der labels wird relativ zur Position des QR-Codes eingestellt, mit einem Offset von *0.05f* nach rechts, aber auf derselben Höhe (y-Koordinate) und Tiefe (z-Koordinate). Die Skalierung der labels wird auf (*PhysicalSize \* 0.02f, PhysicalSize \* 0.02f, 0.005f*) gesetzt. Dies bedeutet, dass die Größe der Labels entsprechend der Größe des QR-Codes angepasst wird.
- Die Position des 3D-Modells, das mit dem QR-Code verbunden ist wird auf die Position des QR-Codes gesetzt. Schließlich wird *lastTimeStamp* auf die aktuelle Zeitmarke des QR-Codes gesetzt, um festzuhalten, wann der QR-Code zuletzt erkannt wurde.

#### 4.5.2.6 Spatial Graph Node

Ein weiteres wichtiges Konzept, das bei der Übersetzung der Koordinaten verwendet wird, sind sogenannte “Spatial Graph Nodes“. Diese werden durch das OpenXR-Plugin von Microsoft bereitgestellt und dienen in unserem Fall dazu, die Tracking-Informationen des QR-Codes in der virtuellen Umgebung abzubilden.

Dieses Skript ist mit dem QR-Code-Objekt verbunden und wandelt die Koordinaten des realen QR-Codes in das Unity-Koordinatensystem um. Darüber hinaus platziert das Skript den virtuellen QR-Code im Szenenaufbau genau an derselben Stelle wie den realen QR-Code. Diese Verwendung von Spatial Graph Nodes ermöglicht eine präzise Zuordnung von physischen und virtuellen Objekten in der Mixed-Reality-Umgebung.

#### 4.5.2.7 Zugriff auf QR-Codes bereitstellen

Wie bereits im Abschnitt ?? erwähnt, benötigen andere Klassen in der Anwendung Zugriff auf die aktuell erkannten QR-Codes in der Szene, um entsprechend reagieren zu können. Dies ist beispielsweise für die Abfrage erforderlich, ob und welche QR-Codes sich im Inventar befinden. Die Bereitstellung dieser Option war eine Herausforderung. Um keine Performance-Einbußen auf der Hololens zu verursachen, läuft der Prozess des QR-Code-Trackings auf mehreren Threads. Die aktuell erkannten QR-Codes werden in einem SortedDictionary gespeichert, welches von anderen Teilen der Anwendung abgefragt werden

kann. Da auf dieses Objekt von mehreren Threads zugegriffen wird, muss es mit einem *lock* geschützt werden, um Inkonsistenzen zu vermeiden. Hierbei handelt es sich um eine Sperre, die verhindert, dass mehrere Threads gleichzeitig auf das gleiche Objekt zugreifen. Auf diese Weise wird sichergestellt, dass die Daten nicht inkonsistent werden und dass die Anwendung stabil und zuverlässig bleibt. Da dadurch bestimmten Threads Zugriff verweigert wird (um die Daten zu schützen), kann es zu Verzögerungen kommen, bis die Daten verfügbar sind. Jedoch ist dieser Nachteil in unserem Anwendungsfall nicht von großer Bedeutung, da wir selten zur gleichen Zeit auf die Daten zugreifen und dadurch die Verzögerung nicht bemerkbar ist.

#### 4.5.3 Platzieren des Inventar Prefabs

→ SKREPEK

Eine präzise Interaktion zwischen der realen Welt und der augmentierten Realität erfordert ein tiefgreifendes Verständnis der Umgebung und eine akkurate Erfassung ihrer Eigenschaften. Um sicherzustellen, dass virtuelle Objekte nahtlos in die physische Umgebung integriert werden können, ist der Zugriff auf die Kamera unerlässlich. Die Kamera erfasst präzise die Details der Umgebung, um relevante Ebenen zu identifizieren. Diese sind wichtig für eine präzise Platzierung von virtuellen Objekten und eine realistische Interaktion zwischen der realen und der augmentierten Welt.

Dieser Abschnitt beschäftigt sich mit der Verwendung des Unity Game Objekts *Inventory Placement Controller* und dessen angehängter Skript-Komponente. Die *InventoryPlacementController* Klasse bildet die Grundlage für die korrekte Platzierung des Inventar-Prefabs und gewährleistet einen reibungslosen Ablauf des Anwendungsszenarios des Knapsack-Problems. Es wird erläutert, wie das Inventar-Prefab strukturiert ist, welche Rolle der User Gaze spielt und wie sie implementiert wird. Außerdem wird die Logik zur Platzierung des Inventar-Prefabs ausführlich beschrieben.

##### 4.5.3.1 Aufbau und Hirarchie des Inventar-Prefabs

Die Gestaltung und Struktur des Prefabs sind von grundlegender Bedeutung, da dieses Objekt die Basis für die Interaktion zwischen Benutzer und dem virtuellen Objekt bildet. Eine sorgfältige Ausarbeitung dieses Prefabs trägt wesentlich zur Gesamterfahrung des Benutzers bei. Es erleichtert dem Benutzer, den Überblick zu behalten, verbessert die Navigation und fördert eine reibungslose Interaktion. Das Prefab beinhaltet nicht nur ein einfaches Raster zur Darstellung des Inventars, sondern auch zusätzliche 3D-Modelle, die das Gesamtbild vervollständigen.

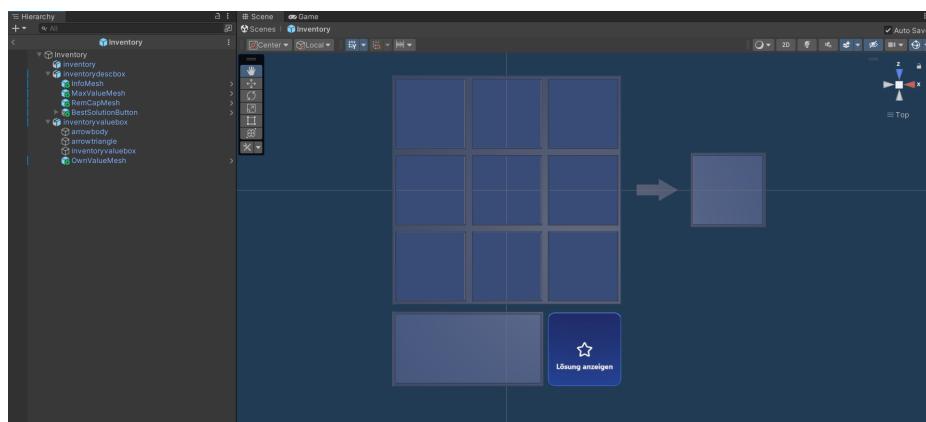


Abbildung 4.25: Aufbau des Inventar-Prefabs im Unity Editor

Die Abbildung 4.25 veranschaulicht das gesamte Prefab im Unity Editor und bietet gleichzeitig Einblicke in die Hierarchie. Es wird deutlich, dass das Prefab aus drei Hauptmodellen besteht, wobei jedes dieser Hauptmodelle weitere Unterobjekte enthält. Diese drei Hauptmodelle sind:

1. **inventory**: Dieses 3x3-Raster bildet die Hauptkomponente des Prefabs, mit dem der Benutzer interagiert, um Gegenstände zu platzieren und zu organisieren.
2. **inventorydescbox**: Das Feld unterhalb des Inventar-Rasters gibt dem Benutzer während des Spielverlaufs Feedback. Es enthält Warnungen, ob ein Gegenstand zu schwer ist, welcher maximale Wert erreicht werden kann und wie viel Kapazität im Inventar noch frei ist. Dem Objekt sind drei *TextMesh*-Elemente untergeordnet. Diese Elemente dienen dazu, dem Benutzer das genannte Feedback visuell anzeigen zu können.
3. **bestSolutionButton**: Der Knopf neben der *inventorydescbox* ermöglicht dem Benutzer, auf Knopfdruck eine perfekte Lösung für das Inventar zu visualisieren. Dadurch kann der Benutzer eine konkrete Vorstellung davon erhalten, wie eine solche Lösung aussehen könnte und wie diese im Vergleich zur eigenen Lösung ist. Dies erleichtert die Erkundung einer perfekten Anordnung von Gegenständen im Inventar, um gegebenenfalls Anpassungen am eigenen Inventar vorzunehmen.
4. **inventoryvaluebox**: Der Pfeil und die Box auf der rechten Seite des Inventars zeigen dem Benutzer an, welchen Wert er im Vergleich zum maximal erreichbaren Wert erreicht hat. Diese Information wird durch das Element *TextMesh* unterhalb visualisiert. Diese Anzeige ist von großer Bedeutung, da sie dem Benutzer signalisiert, wie effektiv die zusammengestellte Lösung ist. Durch die klare Darstellung des aktuellen Werteverhältnisses erhält der Benutzer unmittelbares Feedback über die Qualität seiner aktuellen Lösung.

Die Struktur und das Design des Prefabs sind nicht nur ästhetisch ansprechend, sondern auch funktional und im Einklang mit der übrigen UI/UX der Anwendung. Sie ermöglicht eine effiziente und benutzerfreundliche Interaktion. Die Anordnung der einzelnen Elemente ist gut durchdacht und bietet dem Benutzer eine klare Orientierung, was die Benutzererfahrung insgesamt verbessert.

#### 4.5.3.2 User Gaze in AR-Anwendungen

In Augmented-Reality-Anwendungen spielt der *User-Gaze* eine zentrale Rolle, da er die Interaktion zwischen Benutzer und virtueller Umgebung ermöglicht. Er erfasst und interpretiert dabei die Blickrichtung und -position des Benutzers, was dessen Aufmerksamkeit in der realen Welt widerspiegelt. Der Gaze ist somit entscheidend, um zu bestimmen, worauf der Benutzer fokussiert und wie er mit virtuellen Objekten interagiert.

Durch die Verfolgung des User Gazes können in AR-Anwendungen verschiedene Interaktionen ausgelöst werden. Beispielsweise kann der Benutzer durch einfaches Anschauen eines virtuellen Objekts weitere Informationen anzeigen lassen oder eine Aktion auslösen. Diese intuitive Form der Interaktion ohne physische Eingabegeräte trägt wesentlich zur Benutzerfreundlichkeit bei.

Eine präzise Erfassung und Interpretation des Benutzerblicks ist entscheidend für die Realisierung einer nahtlosen und immersiven AR-Erfahrung. Durch die präzise Verfolgung der Blickrichtung des Benutzers können AR-Anwendungen schnell und intuitiv auf Benutzeraktionen reagieren. Dadurch wird eine beeindruckende und immersive Interaktion zwischen dem Benutzer und der virtuellen Umgebung ermöglicht.

#### 4.5.3.3 Der Inventory Placement Controller

Das Unity Game Objekt *Inventory Placement Controller* fungiert als grundlegender Baustein und Ausgangspunkt für das Anwendungsszenario des Knapsack-Problems. Dieses Objekt übernimmt die Aufgabe dem Benutzer bei Szenenstart Anweisungen zu vermitteln und das Inventar-Prefab präzise zu platzieren und zu visualisieren. Mit der *InventoryPlacementController.cs* Skript-Komponente ausgestattet, implementiert dieses Objekt die *InventoryPlacementController* Klasse, welche die Logik für die Bestimmung des Platzierungsstandorts anhand des Benutzer-Gazes und den weiteren Verlauf dieses Anwendungsszenarios steuert.

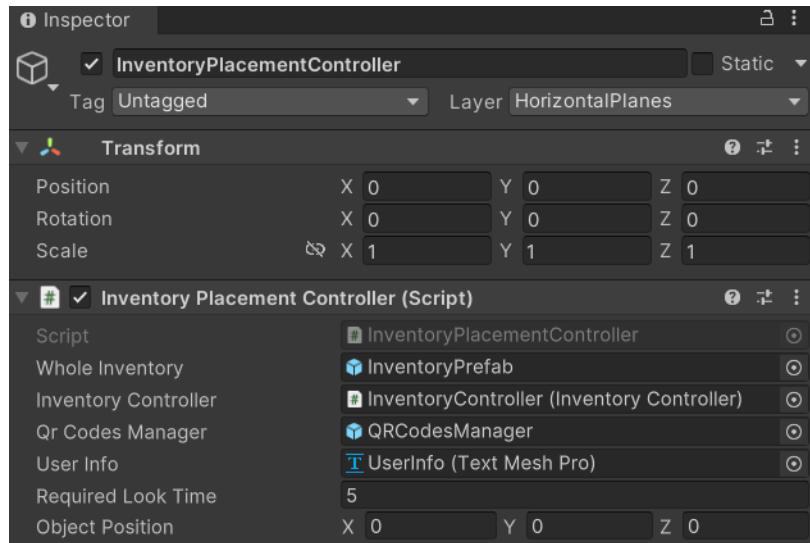


Abbildung 4.26: Das inventoryPlacementController-Objekt im Unity Editor

Abbildung 4.26 zeigt das Game Objekt im Unity Inspektor. Ebenfalls zu sehen ist die angehängte Skript-Komponente, die dem Inventory Placement Controller Game Objekt zugeordnet ist. Innerhalb dieser Komponente sind die öffentlichen Klassenvariablen des Skripts aufgeführt, die direkt im Inspektor übergeben und manipuliert werden können. Diese Variablen spielen eine wichtige Rolle bei der Konfiguration und Steuerung des Inventory Placement Controllers. Zu den relevanten Variablen und ihrer Bedeutung gehören:

- **Inventory Object:** Referenz auf das Inventar Modell aus dem Prefab Ordner, welches platziert werden soll.
- **Inventory Controller:** Verweis auf das Inventory Controller Skript.
- **QR Codes Manager:** Ein Verweis auf das QRCodeManager Game Objekt aus der Szene.
- **User Info:** Referenz auf das in der `main camera` liegende *TextMesh*.
- **Required Look Time:** Diese Variable definiert die vorgeschriebene Zeit, die der Benutzer auf eine Fläche blicken muss, um das Inventar darauf zu platzieren.
- **Object Position:** Variable, in der die Platzier-Position des Inventars gespeichert wird, um später darauf zugreifen zu können.

Die korrekte Übergabe und Konfiguration dieser Objekte und Werte an das Unity Game Objekt sind entscheidend für die gewünschte und korrekte Funktionsweise des Platzierungs-vorgangs des Inventar-Prefabs. Durch die Anpassung der Variablen und ihrer Werte kann die Platzierungs-dynamik präzise gesteuert werden, um eine reibungslose Benutzererfahrung

zu gewährleisten.

#### 4.5.3.4 Implementierung des User Gazes

Im Falle des Knapsack-Anwendungsszenarios wird der Gaze des Benutzers genutzt, um festzustellen, ob dieser auf eine durch den *AR Plane Manager* erfasste Ebene (*AR-Plane*) gerichtet ist. Diese Feststellung ist von großer Bedeutung, um zu bestimmen, ob der Benutzer auf eine geeignete Fläche blickt, auf der das Inventar-Prefab platziert werden kann. Die Klasse *InventoryPlacementController* verwendet zwei Funktionen, um den Blick des Benutzers auf ein *ARPlane* zu erkennen und, um dieses *ARPlane* zu ermitteln und zurückzugeben. In diesem Abschnitt wird die Funktion, welche den Blick-Status ermittelt, anhand des Source-Codes erklärt.

```

1 private bool IsPointerOverPlane() {
2     Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
3     RaycastHit hit;
4     if (Physics.Raycast(ray, out hit)) {
5         ARPlane plane = hit.collider.GetComponent<ARPlane>();
6         return (plane != null);
7     }
8     return false;
9 }
```

Codeabschnitt 4.27: Funktion zur Überprüfung des User Gazes

Um den Status des User Gazes zu überprüfen und damit festzustellen, ob der Benutzer auf ein AR-Plane blickt, wird ein Strahl aus der Hauptkamera (*Camera.main*) in Richtung der aktuellen *Blickrichtung* und *Position* des Benutzers geschossen. Wenn dieser Strahl mit einem virtuellen Objekt in der Szene kollidiert, wird überprüft, ob es sich bei dieser Kollision um ein AR-Plane handelt.

Diese Überprüfung erfolgt über den Zugriff auf die Kollisionskomponente (*collider*) des getroffenen Objekts (*hit*), um auf die spezifische Komponente AR-Plane zuzugreifen. Wenn das getroffene Objekt als AR-Plane identifiziert werden kann, gibt die Funktion *true* zurück, woraus geschlossen werden kann, dass der Benutzer seinen Blick auf ein AR-Plane gerichtet hat. Ansonsten gibt diese Funktion *false* zurück, was das Gegenteil bedeutet.

#### 4.5.3.5 Platzierungskontrolle und ARPlane-Überwachung

Da der Benutzer während des Platzierungsvorgangs möglicherweise seine Blickrichtung ändert, ist es wichtig, die kontinuierlich zu überprüfen, oder ob ein neues AR-Plane im Fokus des Benutzers liegt. Diese fortlaufende Überprüfung ist entscheidend, um sicherzustellen, dass bei einer Änderung des Blicks das neue im Fokus liegende AR-Plane ausgewählt wird und der Vorgang reibungslos fortgesetzt werden kann. Um diese Funktionalität zu realisieren, wird die Funktion *Update()* verwendet.

Diese Funktion ist von entscheidender Bedeutung, da sie sicherstellt, dass der aktuelle Stand des User Gazes kontinuierlich überprüft wird. Dadurch wird gewährleistet, dass das Inventar-Prefab präzise platziert werden kann, selbst wenn sich die Blickrichtung des Benutzers ändert. Um die Logik, die diesem Prozess zugrunde liegt, zu veranschaulichen, wird im Folgenden der Source-Code erklärt.

```

1 void Update() {
2     if (!objectPlaced && canStartScript) {
3         if (IsPointerOverPlane()) {
4             ARPlane currentPlane = GetCurrentPlaneUnderGaze();
5             if (currentPlane != null) {
```

```

6         if (selectedDeskPlane == null || selectedDeskPlane != currentPlane) {
7             selectedDeskPlane = currentPlane;
8             lookStartTime = Time.time;
9         }
10        float timeLookedAtPlane = Time.time - lookStartTime;
11        userInfo.text = ((int)requiredLookTime - (int)timeLookedAtPlane).
12        ToString();
13        if (timeLookedAtPlane >= requiredLookTime) {
14            PlaceObjectOnDesk(selectedDeskPlane);
15            objectPlaced = true;
16            userInfo.text = "";
17        } else {
18            selectedDeskPlane = null;
19            userInfo.text = "Schauen Sie auf einen Tisch";
20        }
21    } else {
22        selectedDeskPlane = null;
23        userInfo.text = "Schauen Sie auf einen Tisch";
24    }
25}
26

```

Codeabschnitt 4.28: Funktion um Usergaze zu verwenden

Generell lässt sich die Funktionsweise dieses Codes in neun Phasen unterteilen, um das Inventar-Prefab entgültig zu platzieren. Diese Phasen sind:

- Platzierungsstatus und Skript-Startbedingung:** Zunächst wird überprüft, ob das Prefab für das Inventar noch nicht platziert wurde (`objectPlaced`) und, ob das Skript gestartet werden kann (`canStartScript`). Dies gewährleistet, dass die Platzierung nur erfolgt, wenn beide Bedingungen erfüllt sind.
- Bestimmung des User Gazes über einem AR-Plane:** Anschließend wird geprüft, ob sich der User Gaze über einem AR-Plane befindet. Hierfür wird die Funktion `IsPointerOverPlane()` aufgerufen, welche feststellt, ob sich der Benutzergaze über einem AR-Plane befindet.
- Identifikation des aktuellen ARPlane:** Wenn bestätigt wird, dass sich der User Gaze über einem AR-Plane befindet, wird das aktuelle AR-Plane identifiziert. Hierfür wird die Funktion `GetCurrentPlaneUnderGaze()` aufgerufen, welche das AR-Plane bestimmt, auf den der Benutzergaze gerichtet ist.
- AR-Plane aktualisieren und Timer starten:** Nach Identifikation des AR-Planes wird überprüft, ob es sich um ein anderes AR-Plane als dieses handelt, auf welches der Benutzer zuletzt geschaut hat. Falls dies der Fall ist, wird die Variable `selectedDeskPlane` aktualisiert und die Startzeit des Blicks auf dem neuen AR-Plane wird gespeichert.
- Messung der Blickdauer und Anzeige der Zeit:** Es wird die Dauer gemessen, die der Benutzer bereits auf das aktuelle AR-Plane blickt. Hierfür wird die Zeitdifferenz seit dem Start des Blicks auf dieses AR-Plane berechnet. Zusätzlich wird dem Benutzer mittels des TextMeshes `userInfo` in Form eines *Countdowns* angezeigt, wie lange er noch auf dieses ARPlane blicken muss.
- Erforderte Blickdauer erreicht:** Es wird überprüft, ob die gemessene Zeit die erforderliche Zeit überschreitet, die der Benutzer benötigt, um das Inventar-Prefab auf dem AR-Plane zu platzieren.
- Platzierung des Inventar Prefabs:** Nachdem die erforderliche Blickdauer erreicht

- wurde, wird die Funktion `PlaceObjectOnDesk()` (siehe Codeabschnitt 4.29) aufgerufen, um das Inventar-Prefab auf dem ausgewählten AR-Plane zu platzieren. (siehe Abschnitt 4.5.3.7)
8. **Platzierungsstatus:** Nachdem das Objekt erfolgreich platziert wurde, wird die Variable `objectPlaced` auf `true` gesetzt, um anzudeuten, dass das Objekt platziert wurde.
  9. **Benachrichtigung bei fehlendem AR-Plane:** Wenn sich der User Gaze nicht über einem AR-Plane befindet oder kein AR-Plane erkannt wird, wird die Benutzeroberfläche entsprechend aktualisiert. Der Benutzer wird darüber informiert, dass er auf ein AR-Plane blicken muss, um das Inventar-Prefab erfolgreich zu platzieren.

#### 4.5.3.6 Platzierung des Prefabs

Die Platzierung des Inventar-Prefabs markiert den Abschluss des Skripts *Inventory Placement Controllers*. Die Funktion `PlaceObjectOnDesk()` platziert das Prefab auf dem AR-Plane, auf welches der Benutzer blickt und aktiviert bzw. deaktiviert weitere Game Objekte sowie Skripte, um einen nahtlosen Fortlauf der Anwendung zu gewährleisten.

```

1  private void PlaceObjectOnDesk(ARPlane deskPlane) {
2      qrCodesManager.SetActive(true);
3      objectPosition = deskPlane.center + Vector3.up * heightOffset;
4      wholeInventory.transform.position = objectPosition;
5      wholeInventory.transform.rotation = Quaternion.Euler(0, 180f, 0);
6      wholeInventory.SetActive(true);
7      GameObject inventory = wholeInventory.transform.Find("inventory").gameObject;
8      inventoryController.SetInventoryObject(inventory);
9      inventoryController.gameObject.SetActive(true);
10     gameObject.SetActive(false);
11 }
```

Codeabschnitt 4.29: Funktion zum Platzieren des Inventarobjekts

Die Platzierung des Inventar-Prefabs erfolgt in vier Hauptschritten, die den Ablauf des Platzierungsprozesses steuern und die Kontinuität des AR-Anwendungsszenarios sicherstellen.

1. **Aktivierung des QRCodeManagers:** Der QRCodeManager wird aktiviert, um sicherzustellen, dass QR-Codes nach Abschluss dieses Skripts erfolgreich erfasst werden können.
2. **Berechnung der Position und Rotation:** Die Position des Inventar-Prefabs auf dem AR-Plane wird berechnet, um sicherzustellen, dass es an der richtigen Stelle platziert wird. Die Rotation des Objekts wird angepasst, um eine korrekte Ausrichtung zu gewährleisten. Schließlich wird das Objekt aktiviert, um es für den Benutzer sichtbar zu machen.
3. **Extraktion und Übergabe des Inventar-Objekts:** Das Inventar-Objekt wird aus dem Prefab extrahiert und an den *Inventory Controller* übergeben. Dadurch wird eine nahtlose Fortsetzung der Interaktion mit dem Inventar ermöglicht.
4. **Deaktivierung des Objekts:** Abschließend wird das eigene Game-Objekt deaktiviert, um das Ende des Platzierungsvorgangs des Inventar-Prefabs zu markieren. Dadurch wird dem Benutzer signalisiert, dass der Platzierungsvorgang abgeschlossen ist und eine klare Abgrenzung erleichtert.

Diese Schritte sind essenziell für einen reibungslosen Ablauf des Platzierungsprozesses des Inventar-Prefabs und tragen zu einer effizienten und benutzerfreundlichen Interaktion

innerhalb des AR-Anwendungsszenarios bei.

#### 4.5.3.7 Visualisierung des Platzierungsablaufs

Um den Prozess des Platzierens des Inventar-Prefabs aus der Perspektive des Benutzers besser zu veranschaulichen, werden in den folgenden drei Abbildungen ??, ?? und ?? die Schritte dieses Vorgangs dargestellt. Dabei illustriert die letzte Abbildung ?? den Endzustand mit dem platzierten Inventar sowie einer darauf folgenden Anweisung für den Benutzer.

Diese visuelle Darstellung bietet eine ergänzende Perspektive auf den Ablauf des Platzierens und ermöglicht es, die einzelnen Schritte des Prozesses besser zu verstehen und den Abschluss des Platzierungsvorgangs nachzuvollziehen.



Die erste Abbildung zeigt, dass der Benutzer zu diesem Zeitpunkt nicht auf eine horizontale Fläche blickt. Daraufhin wird eine Anweisung angezeigt, dass auf eine horizontale Fläche geblickt werden muss, um das Inventar-Prefab platzieren zu können. Diese Anweisung dient als Orientierungshilfe, um den ersten Schritt des Platzierungsprozesses erfolgreich abzuschließen.

Die zweite Abbildung zeigt, dass der Benutzer seinen Blick nun auf einen Tisch, d.h. eine horizontale Fläche, gerichtet hat. Diese Aktion startet einen Countdown, der dem Benutzer visuell anzeigt, wie lange er noch auf diese Fläche blicken muss, um das Inventar-Prefab erfolgreich zu platzieren.

Die dritte Abbildung zeigt den Abschluss des Platzierungsprozesses, bei dem das Inventar-Prefab erfolgreich platziert wurde. Nach Abschluss dieser Phase erhält der Benutzer weitere Anweisungen für die nächsten Schritte, um das Inventar effektiv im Spiel zu verwenden.

#### 4.5.4 Inventarverwaltung

→ SKREPEK

Eine effiziente Handhabung und Überwachung des platzierten Inventars ist ein zentraler Aspekt im Kontext des Rucksack-Problems. Es ist wichtig, das Inventar fortlaufend und präzise in Echtzeit zu erfassen und zu aktualisieren, um sämtliche Veränderungen - sei es das Hinzufügen oder Entfernen eines Gegenstands seitens des Benutzers - unmittelbar zu registrieren. Dank dieses Echtzeit-Feedbacksystems ist es möglich, direkt auf Veränderungen zu reagieren und dem Benutzer entsprechendes Feedback zu geben.

Dieser Abschnitt beschreibt das *Inventory Controller* Unity Game Objekt und seine angehängte Skript-Komponente im Detail. Die Klasse *InventoryController* ist das Herzstück für die Überwachung und Verwaltung des Inventars und gewährleistet einen reibungslosen und kontinuierlich aktualisierten Ablauf des Anwendungsszenarios des Rucksack-Problems.

in Echtzeit. Es wird erläutert, wie diese Klasse funktioniert und welche Bedeutung die Begrenzungen eines Objekts in diesem Kontext haben.

#### 4.5.4.1 Der Inventory Controller

Das Unity Game Objekt **Inventory Controller** ist von entscheidender Bedeutung für die nahtlose Interaktion und Kontrolle des Inventars innerhalb des Anwendungsszenarios des Rucksack-Problems. Diese zentrale Einheit fungiert als Dreh- und Angelpunkt für die Erfassung und Verarbeitung sämtlicher Änderungen am Inventar, die durch den Benutzer initiiert werden. Ausgestattet mit der Skript-Komponente *InventoryController.cs*, verkörpert das *Inventory Controller* Objekt die *InventoryController* Klasse, welche nicht nur die Verwaltung des Inventars, sondern auch dessen Echtzeitüberwachung steuert. Diese Klasse bildet das Rückgrat für die reibungslose und kontinuierliche Aktualisierung des Inventars gemäß den Handlungen des Benutzers.

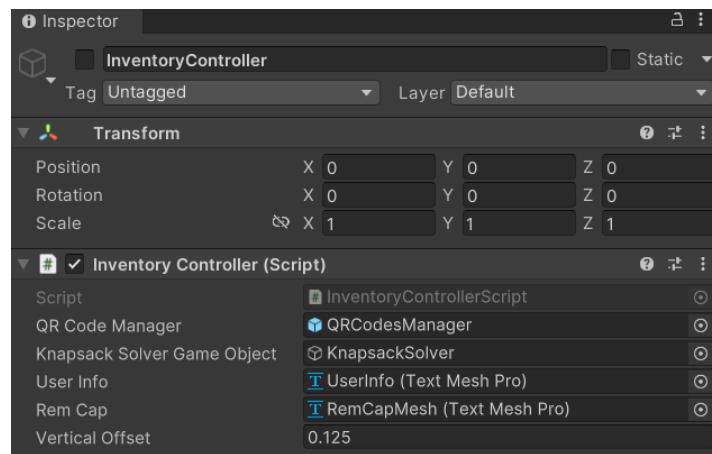


Abbildung 4.27: InventoryController Objekt im Editor

Abbildung 4.27 zeigt das Game Objekt im Unity Inspektor, welches den Inventory Controller repräsentiert. Es ist erkennbar, dass diesem Objekt eine Skript-Komponente angehängt ist, die dem Inventory Controller zugeordnet ist. Innerhalb dieser Komponente werden die öffentlichen Klassenvariablen des Skripts aufgeführt, welche direkt im Inspektor eingesehen und manipuliert werden können. Diese Variablen sind wichtig für die Konfiguration und Steuerung des Inventory Controllers. Folgende Variablen sind relevant und haben eine bestimmte Bedeutung:

- **QR Code Manager:** Referenz auf das *QRCodeManager* Game Objekt aus der Szene.
- **Knapsack Solver Game Objekt:** Ein Verweis auf das *KnapsackSolver* Game Objekt aus der Szene.
- **User Info:** Referenz auf das in der *main camera* liegende *TextMesh*.
- **Vertical Offset:** Float-Wert, um den die Begrenzungen (*Bounds*) des Inventory-Objekts auf der X-Achse erweitert werden.

Die korrekte Übergabe und Konfiguration dieser Objekte und Werte an das Unity Game Objekt ist von entscheidender Bedeutung für die gewünschte und fehlerfreie Funktionsweise der Inventarverwaltung. Durch die Anpassung dieser Werte und Variablen lässt sich das Verhalten der Verwaltungsdynamik präzise steuern, um eine nahtlose Benutzererfahrung sicherzustellen.

#### 4.5.4.2 Begrenzungen eines Objekts

Die Begrenzungen eines Objekts, auch als *Bounds* bezeichnet, sind ein essenzielles Konzept in der Computerspielentwicklung und anderen Bereichen der Computergrafik. Sie definieren den umschließenden *Raum* oder *Bereich*, den ein Objekt einnimmt. Diese Begrenzungen sind von grundlegender Bedeutung für die räumliche Positionierung von Objekten sowie für die Kollisions- und Interaktionsprüfung zwischen verschiedenen Elementen innerhalb einer Szene.<sup>29</sup>

In Unity werden die Begrenzungen eines Objekts über die Eigenschaften der Klasse *Bounds* verwaltet. Jedes Game Object in Unity ist mit einer *Collider*-Komponente ausgestattet, der unter anderem die Begrenzungen des Objekts definiert. Die ermittelten Begrenzungen eines Objekts haben eine Vielzahl von Anwendungsbereichen:

1. **Kollisionserkennung:** Durch den Vergleich der Begrenzungen zweier Objekte können Kollisionen zwischen diesen erkannt werden. Dieser Prozess ist von grundlegender Bedeutung für die physikalische Interaktion von Objekten innerhalb einer digitalen Umgebung und ermöglicht die Umsetzung realistischer Verhaltensweisen.
2. **Platzierung von Objekten:** Die Begrenzungen dienen als Leitfaden für die Platzierung neuer Objekte innerhalb einer Szene. Durch die Gewährleistung, dass die Positionierung innerhalb eines definierten Bereichs oder Umfelds erfolgt, wird die konsistente Gestaltung und Strukturierung der digitalen Welt ermöglicht.
3. **Bewegung und Rotation von Objekten:** Bei der Bewegung oder Rotation von Objekten können die Begrenzungen genutzt werden, um sicherzustellen, dass diese innerhalb eines vordefinierten Gültigkeitsbereichs bleiben. Dies gewährleistet nicht nur die Kohärenz der Szene, sondern auch die Vermeidung von unerwünschten Interferenzen oder Überschneidungen zwischen verschiedenen Elementen.

Die effektive Nutzung der Begrenzungen eines Objekts trägt somit wesentlich zur Präzision und Stabilität digitaler Szenen bei, indem sie eine solide Grundlage für die Positionierung, Bewegung und Interaktion von Objekten innerhalb einer virtuellen Umgebung bildet.

#### 4.5.4.3 Problem bei Standardbegrenzungen

Die Standardbegrenzungen des Inventar-Objekts sind so festgelegt, dass sie genau die räumliche Fläche dieses Objekts abdecken. Dies kann jedoch zu Problemen führen, insbesondere wenn sich ein Gegenstand innerhalb dieser Grenzen befindet, aber möglicherweise nicht erkannt wird. Dies tritt auf, weil die QR-Codes auf realen Objekten (Holzplättchen mit einer Stärke / Höhe von 20 mm) angebracht sind und daher die Position entlang der y-Achse (Höhe) verschoben wird.

Um dieses Problem zu lösen, müssen die Begrenzungen erweitert werden. Dies ermöglicht es dem *Inventory Controller*, auch Gegenstände zu erfassen, die sich möglicherweise innerhalb des ursprünglichen Begrenzungsbereichs befinden, aber aufgrund der vertikalen Verschiebung nicht erkannt wurden. Durch die Erweiterung der Grenzen wird sichergestellt, dass alle relevanten Objekte ordnungsgemäß erfasst und im Inventar berücksichtigt werden.

Diese Erweiterung der Begrenzungen ist insbesondere wichtig, um die Zuverlässigkeit und Effizienz des *Inventory Controller* sicherzustellen, insbesondere in Umgebungen, in denen die vertikale Positionierung der Objekte variiert und demnach eine präzise Erfassung erforderlich ist.

---

<sup>29</sup>Unity Dokumentation **Bounds**

#### 4.5.4.4 Erweitern der Standardbegrenzungen

Um das Problem der Nichterkennung von neu hinzugefügten Gegenständen im Inventar aufgrund der Begrenzungen des Inventar-Objekts zu lösen, müssen die Standardbegrenzungen des Objekts gründlich überprüft und erweitert werden. Dieser Schritt wird durch die sorgfältige Implementierung der Funktion `UpdateInventoryBounds()` realisiert. Diese Funktion ermöglicht es, die Grenzen des Inventar-Objekts anzupassen und entsprechend der aktuellen Anforderungen zu erweitern.

Die Herausforderung besteht darin, sicherzustellen, dass das Inventar sämtliche neu hinzugefügten Gegenstände korrekt erkennt und effizient verwaltet, selbst wenn diese die ursprünglichen Begrenzungen des Objekts überschreiten. Durch die Aktualisierung der Standardbegrenzungen können solche Änderungen sofort erfasst und berücksichtigt werden, wodurch ein unterbrechungsfreier Ablauf des *Inventory Controllers* gewährleistet wird.

Die Funktion besteht aus drei Phasen, die durchlaufen werden, um die Begrenzungen des Objekts zu erweitern. Diese sind die folgenden:

1. `UpdateInventoryBounds()`: Diese Funktion ist die zentrale Methode, die aufgerufen wird, um die Begrenzungen des Inventars zu aktualisieren. Sie überprüft zunächst, ob ein Inventarobjekt vorhanden ist und ruft anschließend Hilfsfunktionen auf, um den Rest abzuarbeiten. Die Begrenzungen werden entsprechend des vertikalen Versatzes erweitert.

```

1 private void UpdateInventoryBounds() {
2     if (inventoryObject != null) {
3         Bounds localBounds = GetBounds(inventoryObject);
4         ExtendBounds(ref localBounds, verticalOffset);
5         inventoryBounds = localBounds;
6     }
7 }
```

2. `GetBounds()`: Diese Methode dient dazu, die Grenzen des übergebenen Spielobjekts zu bestimmen. Sie verwendet die Renderer-Komponenten des Objekts, um die Grenzen zu erhalten, und fällt auf die Position des Objekts zurück, falls kein Renderer vorhanden ist.

```

1 private Bounds GetBounds(GameObject obj) {
2     Renderer renderer = obj.GetComponent<Renderer>();
3     return renderer != null ? renderer.bounds : new Bounds(obj.transform.position,
4         Vector3.one);
```

3. `ExtendBounds()`: Hier handelt es sich um die Funktion, die die eigentliche Erweiterung der Begrenzungen durchführt. Durch die Verwendung einer Referenz ist es möglich, direkt die Begrenzungen des originalen Objekts zu bearbeiten. Sie passt sowohl das Zentrum als auch die Ausdehnung der Begrenzung auf der y-Achse entsprechend des `offset` an.

```

1 private void ExtendBounds(ref Bounds bounds, float offset) {
2     bounds.center = new Vector3(bounds.center.x, bounds.center.y + offset / 2,
3         bounds.center.z);
4     bounds.extents = new Vector3(bounds.extents.x, bounds.extents.y + offset / 2,
5         bounds.extents.z);
6 }
```

Diese Funktionen stellen sicher, dass die Begrenzungen des Inventars aktualisiert werden, um sicherzustellen, dass alle relevanten Objekte ordnungsgemäß erfasst und berück-

sichtigt werden. Dies trägt dazu bei, die Zuverlässigkeit und Effizienz des *Inventory Controllers* zu verbessern, insbesondere in Umgebungen, in denen die vertikale Position der Objekte variiert.

### Vorher-Nachher-Vergleich der Begrenzungen

Um den Effekt der Funktion `UpdateInventoryBounds()` auf die Begrenzungen des Objekts zu verdeutlichen, werden in den folgenden Abbildungen der Zustand vor und nach dem Aufruf der Funktion gegenübergestellt.

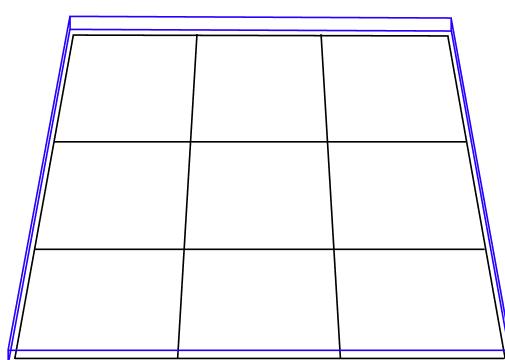


Abbildung 4.28: Begrenzungen vor dem Funktionsaufruf

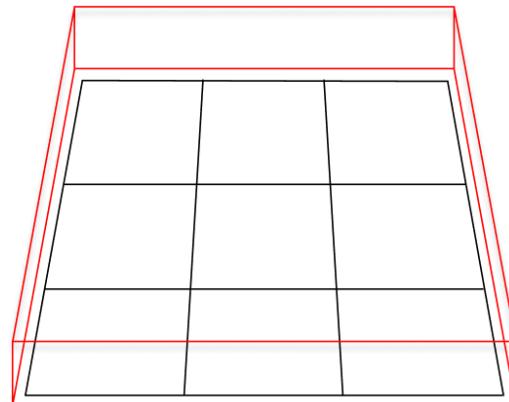


Abbildung 4.29: Begrenzungen nach dem Funktionsaufruf

Abbildung 4.28 zeigt die Standardbegrenzungen des Inventar-Objekts vor dem Funktionsaufruf. Diese Begrenzungen schließen direkt am Rand des Objekts ab, was in dieser Anwendung potenziell zu Fehlern führen kann. Abbildung 4.29 zeigt die Begrenzungen nach dem Funktionsaufruf, bei dem die Standardbegrenzungen erweitert wurden. Durch diese Erweiterung können keine Probleme mehr auftreten.

### Lokal- und Weltposition

In Unity bezieht sich die *lokale Position* eines Objekts auf seine Position relativ zu seinem Elternobjekt oder zum Koordinatenursprung, wenn es kein Elternobjekt hat. Diese Position wird relativ zu den Achsen des Objekts selbst angegeben, unabhängig von der umgebenden Szene.

Im Gegensatz dazu bezeichnet die *Weltposition* eines Objekts seine Position im globalen Koordinatensystem der Szene. Sie berücksichtigt die Position des Objekts relativ zum Koordinatenursprung der Szene sowie alle Transformationen, die auf das Objekt angewendet wurden, einschließlich Verschiebung, Drehung und Skalierung.

Die Berechnung der Weltposition eines Objekts erfolgt durch Umrechnung seiner lokalen Position relativ zu seinem Elternobjekt oder zum Koordinatenursprung in die Szene. Diese Berechnung wird durch eine einfache Vektoraddition realisiert, wobei die lokale Koordinate des Objekts zur Weltkoordinate addiert wird, um die lokale Position in eine globale Koordinate umzuwandeln. Dadurch wird die tatsächliche Position des Objekts in der Welt bestimmt, was Interaktionen mit anderen Objekten oder Koordinaten in der Szene ermöglicht.<sup>30</sup>

<sup>30</sup>Unity Dokumentation **Lokal- und Weltposition**

#### 4.5.4.5 Erkennen hinzugefügter oder entfernter Gegenstände

Das Herzstück des *Inventory Controllers* ist die `Update()` Funktion. Sie erkennt neu hinzugefügte und entfernte Gegenstände, nachdem der Benutzer sie dem Inventar hinzugefügt oder entnommen hat. Zusätzlich gewährleistet diese Funktion, dass bei jeder Änderung im Inventar eine Neuermittlung der Gesamtwerte durchgeführt wird. Diese Aktualisierung wird genutzt, um die drei *TextMesh* Elemente des Inventar-Prefabs zu aktualisieren und dem Benutzer die aktuellen Werte und Informationen des Inventars zu visualisieren. Im Folgenden wird der Source-Code dieser Funktion zusammen mit einer detaillierten Erläuterung präsentiert:

```

1 void Update() {
2     lock (activeQRObjects) {
3         foreach (var item in activeQRObjects.Values) {
4             QRCode qRCode = item.GetComponent<QRCode>();
5             Vector3 worldPosition = item.transform.TransformPoint(qRCode.item.qrData.
position);
6             if (item != null && inventoryBounds.Contains(worldPosition)) {
7                 int itemId = qRCode.item.qrData.id;
8                 if (!processedItems.Contains(itemId)) {
9                     if (currWeight + qRCode.item.qrData.weight <= cap) {
10                         userInfo.text = "";
11                         processedItems.Add(itemId);
12                         Vector2 startGridPosition = CalculateGridPosition(
worldPosition);
13                         idGrid[(int)startGridPosition.x, (int)startGridPosition.y] =
itemId;
14                         knapsackSolver?.UpdateInfoMesh("", Color.white);
15                         currWeight += qRCode.item.qrData.weight;
16                         remCap.text = "Verbleibendes Gewicht: " + (cap - currWeight) +
"/" + cap;
17                         EventManager.GridUpdate(idGrid);
18                 } else {
19                     knapsackSolver?.UpdateInfoMesh("Item hat zu viel Gewicht!",
Color.red);
20                 }
21             }
22         } else if (!inventoryBounds.Contains(worldPosition) && processedItems.
Contains(qRCode.item.qrData.id) && ContainsId(qRCode.item.qrData.id)) {
23             int itemId = qRCode.item.qrData.id;
24             processedItems.Remove(itemId);
25             RemoveItem(itemId);
26             currWeight -= qRCode.item.qrData.weight;
27             remCap.text = "Verbleibendes Gewicht: " + (cap - currWeight) + "/" +
cap;
28             EventManager.GridUpdate(idGrid);
29         }
30     }
31 }
32 }
```

Codeabschnitt 4.30: Neue / Entfernte Items erkennen

Im Wesentlichen kann die vorliegende Funktion in mehrere einzelne Schritte unterteilt werden, die den Verwaltungsvorgang von neuen und entfernten Gegenständen steuert. Um diese Funktion dementsprechend Schritt für Schritt zu erklären, sind hier die einzelnen Schritte samt Erklärung, was diese beinhalten beschrieben:

- 1. Sperren der aktiven QRCode Objekte:** Bevor die Verarbeitung der QRCode-Objekte beginnt, wird das Dictionary `activeQRObjects` gesperrt, um sicherzustellen,

dass keine anderen Teile des Systems gleichzeitig darauf zugreifen und es möglicherweise zu einem inkonsistenten Datenstand kommt.

2. **Iteration durch die QRCode-Objekte:** Innerhalb der gesperrten Region wird eine `foreach`-Schleife verwendet, um durch alle Einträge des `activeQRObjects` Dictionaries zu iterieren. Jeder QRCode-Gegenstand wird einzeln überprüft und verarbeitet, um sicherzustellen, dass alle relevanten Aktionen auf jedes Element angewendet werden.
3. **QRCode-Extraktion und Berechnung der Weltposition:** In dieser Phase wird die QRCode-Komponente aus dem Gegenstandsobjekt extrahiert und anschließend die Weltposition des Gegenstands berechnet. Dazu wird die lokale Position des Gegenstands relativ zu seinem Elternelement transformiert, um seine exakte Position im globalen Koordinatensystem zu ermitteln. Hierfür müssen die Funktionen `GetComponent<QRCode>()` und `TransformPoint()` aufgerufen werden. Dieser Schritt ist wichtig, da die Position des Gegenstands vor der Berechnung nur in lokalen Koordinaten gespeichert ist. Ohne diese Umwandlung wäre eine Interaktion zwischen dem QR-Objekt und dem Inventar nicht möglich.
4. **Überprüfung der Gegenstandsposition innerhalb der Inventargrenzen:** Ziel dieser Überprüfung ist es sicherzustellen, dass die berechnete Weltposition des Gegenstands innerhalb der festgelegten Grenzen des Inventars liegt. Nur Gegenstände, die sich innerhalb dieser Grenzen befinden, werden für das Inventar berücksichtigt. Es wird die Funktion `Contains()` aufgerufen, um zu prüfen, ob die Position innerhalb der Grenzen liegt.
5. **Prüfung auf bereits erfolgte Verarbeitung des Gegenstands:** In diesem Schritt wird überwacht, ob der betreffende Gegenstand bereits im Inventar verarbeitet wurde, um doppelte Einträge zu verhindern und die Datenkonsistenz zu wahren. Die Funktion `Contains()` wird aufgerufen, um zu überprüfen, ob der Gegenstand bereits in der Liste der verarbeiteten Gegenstände (`processedItems`) enthalten ist.
6. **Überprüfung des Gewichtslimits:** Diese Überprüfung dient dazu festzustellen, ob das Hinzufügen des Gegenstands das vorgegebene Gewichtslimit des Inventars überschreiten würde. Dadurch wird sichergestellt, dass das Inventar nicht überladen wird und das Gewichtslimit eingehalten wird. Hierbei wird die Funktion `UpdateInfoMesh()` aufgerufen, um das `InfoMesh` zu aktualisieren.
7. **Hinzufügen des Gegenstands zum Inventar:** Nach erfolgreicher Prüfung wird der Gegenstand dem Inventar hinzugefügt. Dabei werden interne Datenstrukturen aktualisiert, einschließlich der Berechnung der Gitterposition des Gegenstands mithilfe der Funktion `CalculateGridPosition()`. Das `Info Mesh` wird ebenfalls aktualisiert, um dem Benutzer aktuelle Informationen bereitzustellen. Außerdem wird die verbleibende Kapazität durch Aktualisierung des `remCap` TextMesh-Elements dargestellt. Die Funktion `GridUpdate()` informiert schließlich andere Teile des Systems über die Aktualisierung des Inventars.
8. **Überprüfung auf Entfernung:** Im Fall, dass der aktuelle Gegenstand außerhalb der Begrenzungen des Inventars liegt, jedoch noch in `processedItems` und `idGrid` vorhanden ist, bedeutet das, dass der Benutzer diesen entfernt hat. Wenn das der Fall ist, wird dieser mithilfe der Funktion `RemoveItem` und `GridUpdate()` aus dem Inventar entfernt und aktualisiert.

#### 4.5.4.6 Zustände des Inventars aus der Sicht Benutzers

In diesem Abschnitt werden die verschiedenen Zustände des Inventars aus Sicht des Benutzers anhand von Fallbeispielen detailliert betrachtet. Das Inventar durchläuft unterschiedliche Zustände, darunter das Hinzufügen und Entfernen von Gegenständen sowie das Überschreiten des Gewichtslimits. Es ist von großer Bedeutung, diese Zustände zu veranschaulichen, um ein tieferes Verständnis für die Funktionsweise des *Inventory Controllers* zu erlangen und um einen klaren Überblick zu verschaffen. Jeder dieser Zustände wird durch visuelle Darstellungen illustriert, um zu veranschaulichen, wie sich das Inventar sowie die zugrundeliegende Datenstruktur in verschiedenen Situationen verhalten. Dabei wird auch darauf eingegangen, wie der Benutzer durch die grafische Benutzeroberfläche (GUI) über diese Zustände informiert wird.

##### Gegenstand zum Inventar hinzugefügt

Dieser Zustand tritt ein, sobald der Benutzer einen Gegenstand erfolgreich in dem Inventar platziert hat. Das Inventar reagiert umgehend und integriert den neuen Gegenstand nahtlos in seine Struktur. Die nachfolgende Abbildung 4.30 visualisiert diesen Zustand.

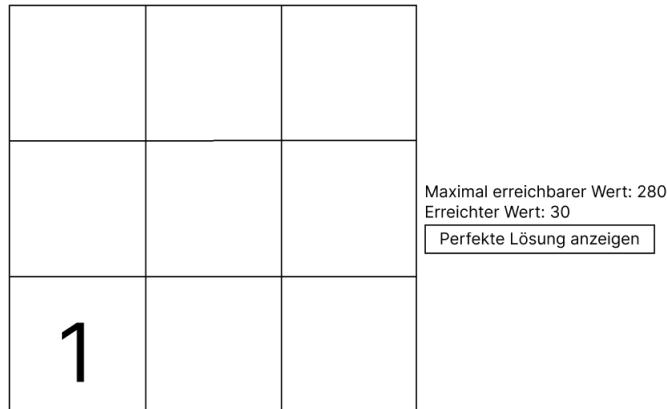


Abbildung 4.30: Inventar mit erfolgreich hinzugefügtem Gegenstand

Die Abbildung verdeutlicht die gelungene Integration des neuen Gegenstands in das Inventar. Durch eine klare visuelle Darstellung signalisiert das *Info Mesh* dem Benutzer unmissverständlich den aktualisierten Zustand des Inventars und bestätigt die erfolgreiche Platzierung des Gegenstands.

Im Hintergrund wird das `usedItems`-Array aktualisiert, um den neuen Gegenstand zu integrieren:

$$\text{usedItems} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Gleichzeitig wird der Gegenstand in das `processedItems`-Array aufgenommen, um sicherzustellen, dass er nicht doppelt berücksichtigt wird:

$$\text{processedItems} = \{1\}$$

##### Gegenstand aus dem Inventar entfernt

Dieser Zustand tritt ein, sobald der Benutzer einen Gegenstand aus dem Inventar entfernt hat, wie in Abbildung 4.31 dargestellt.

5		
	3	

Maximal erreichbarer Wert: 280  
 Erreichter Wert: 95

Abbildung 4.31: Inventar nach Entfernen eines Gegenstands

Die Abbildung zeigt das Inventar nach Entfernen des Gegenstands aus der Position 4. Das *Info Mesh* signalisiert dem Benutzer deutlich den aktualisierten Zustand des Inventars und bestätigt die erfolgreiche Entfernung des Gegenstands.

Im Hintergrund wird das `usedItems`-Array aktualisiert, um den entfernten Gegenstand zu berücksichtigen:

$$\text{usedItems(vorher)} = \begin{bmatrix} 0 & 0 & 0 \\ 5 & 1 & 0 \\ 0 & 3 & 0 \end{bmatrix}$$

$$\text{usedItems(nachher)} = \begin{bmatrix} 0 & 0 & 0 \\ 5 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix}$$

Gleichzeitig wird der Gegenstand aus dem `processedItems`-Array entfernt, um sicherzustellen, dass er nicht erneut verarbeitet wird:

$$\text{processedItems (vorher)} = \{1\}$$

$$\text{processedItems (nachher)} = \{\}$$

### Gegenstand zu schwer für das Inventar

Dieser Zustand tritt ein, wenn der Benutzer versucht, einen Gegenstand hinzuzufügen, der das Gewichtslimit des Inventars überschreitet, wie in Abbildung 4.32 illustriert.

	9	
5		
1	3	

Item hat zu viel Gewicht!  
Maximal erreichbarer Wert: 280  
Erreichter Wert: 250  
[Perfekte Lösung anzeigen](#)

Abbildung 4.32: Inventar mit einem Gegenstand, der das Gewichtslimit überschreitet

Die Abbildung zeigt das Inventar mit mehreren hinzugefügten Gegenständen. Die Reihenfolge in der diese Gegenstände hinzugefügt wurden, ist die folgende: 1, 5, 3, 9. Dies ist wichtig, weil der zuletzt hinzugefügt Gegenstand den zu schweren Gegenstand widerspiegelt. Auch zu sehen ist, dass das *Info Mesh* den Benutzer deutlich über das Überschreiten des Gewichtslimits informiert und das Hinzufügen des Gegenstands verhindert.

Im Hintergrund bleibt das `usedItems`-Array unverändert, da der Gegenstand nicht hinzugefügt wurde:

$$\text{usedItems} = \begin{bmatrix} 0 & 0 & 0 \\ 5 & 0 & 0 \\ 1 & 3 & 0 \end{bmatrix}$$

Das `processedItems`-Array bleibt ebenfalls unverändert:

$$\text{processedItems} = \{1, 5, 3\}$$

#### 4.5.5 EventManager

Der *EventManager* ist ein entscheidendes Game-Objekt, das als zentrales Kommunikationselement fungiert und die Interaktion zwischen verschiedenen Komponenten und Klassen innerhalb der Anwendung ermöglicht. Implementiert als Klasse im Skript *EventManager.cs*, das dem Game-Objekt angehängt ist, spielt er eine unverzichtbare Rolle trotz seiner Kompatibilität, da er die Schnittstelle für die Kommunikation zwischen verschiedenen Klassen und Game-Objekten bereitstellt.

→  
HAYLAZ

Das Konzept der Ereignisse ermöglicht die Kommunikation zwischen verschiedenen Teilen eines Programms, ohne dass direkte Abhängigkeiten zwischen diesen Teilen bestehen müssen. Andere Teile des Codes können sich auf diese Ereignisse registrieren, um benachrichtigt zu werden, wenn sie auftreten. In diesem Zusammenhang werden spezifische Ereignisse definiert:

- **Nachrichteneingang:** Dieses Event wird ausgelöst, wenn im ersten Level eine Nachricht von einem Laptop empfangen wird. Nach dem Auslösen wird ein Ping-Paket über ein Kabel auf der Brille simuliert.
- **Nachricht versendet:** Dieses Event wird ausgelöst, wenn im ersten Level eine Nachricht an einen Laptop gesendet wird. Es dient dazu die Empfangene Nachricht an den zweiten Laptop im richtigen Moment (nach Abschluss der Simulation) weiter zu leiten.

- **Inventar Aktualisierung:** Dieses Event wird ausgelöst, wenn im zweiten Level ein Item ins Inventar gelegt und das Inventar aktualisiert wird. Nach der Aktualisierung wird der aktuelle Wert des Inventars berechnet. Dadurch können wir auf eine ständige Neuberechnung des Inventars verzichten und limitierte Ressourcen sparen.

Die Klasse *EventManager* definiert diese Ereignisse als statische Ereignisse und stellt Methoden bereit, um die Ereignisse auszulösen. Dies erleichtert die globale Verwendung des Ereignissystems in verschiedenen Teilen des Codes. Die Aktionen (Actions) sind statisch, was bedeutet, dass sie auf Klassenebene definiert sind und keine Instanz der Klasse benötigen, um aufgerufen zu werden. Die Methode *Invoke* wird verwendet, um die Ereignisse auszulösen.

```

1 public static class EventManager
2 {
3     // Szenario 1
4     // Ereignis fÃ¼r den Empfang einer Nachricht
5     public static event System.Action<int, string, string> OnMessageReceived;
6     // Methode zum AuslÃ¶sen des Ereignisses fÃ¼r den Empfang einer Nachricht
7     public static void ReceiveMsg(int idx, string username, string message) =>
        OnMessageReceived?.Invoke(idx, username, message);
8
9     // Ereignis fÃ¼r das Senden einer Nachricht
10    public static event System.Action<int, string, string> OnMessageSend;
11    // Methode zum AuslÃ¶sen des Ereignisses fÃ¼r das Senden einer Nachricht
12    public static void SendMsg(int idx, string username, string message) => OnMessageSend
        ?.Invoke(idx, username, message);
13
14    // Szenario 2
15    // Ereignis fÃ¼r die Aktualisierung des Inventars
16    public static event System.Action<int[,]> OnGridUpdate;
17    // Methode zum AuslÃ¶sen des Ereignisses fÃ¼r die Aktualisierung des Inventars
18    public static void GridUpdate(int[,] grid) => OnGridUpdate?.Invoke(grid);
19 }
```

Der gezeigte Codeabschnitt definiert ein Ereignis und eine Methode, die verwendet werden, um auf die drei genannten Fälle zu reagieren. Durch die Definition des Ereignisses wird eine Möglichkeit geschaffen, dass andere Teile des Codes sich darauf registrieren können, um bei Auftreten des Ereignisses benachrichtigt zu werden.

Zum Beispiel wird in diesem Kontext das *OnGridUpdate*-Ereignis definiert, um die Aktualisierung des Inventars zu signalisieren. Die Methode *GridUpdate* dient dazu, dieses Ereignis auszulösen, indem sie es aufruft und dabei Informationen über das aktualisierte Inventar übergibt.

Um auf das aktualisierte Inventar zu reagieren, könnten andere Teile des Codes sich auf dieses Ereignis registrieren, indem sie eine Methode oder einen Handler bereitstellen, der ausgeführt wird, wenn das Ereignis eintritt, wie folgt:

```

1 //Dieser Code Abschnitt befindet sich in der Klasse: KnapsackSolver.cs
2
3 void Start()
4 {
5     items = new QRItem(0).items;
6     // Die SetInventory Methode wird als Ereignishandler fÃ¼r das OnGridUpdate Ereignis
     // registriert
7     EventManager.OnGridUpdate += SetInventory;
8 }
9
10 public void SetInventory(int[,] newInventory)
11 {
```

```

12 inventory = newInventory;
13 CalculateKnapsack();
14 }
```

Nachdem die Szene geladen wurde, registriert sich die Klasse *KnapsackSolver* für das Ereignis *OnGridUpdate*, indem sie die Methode *SetInventory* als Ereignishandler verwendet. Das bedeutet, dass jedes Mal, wenn das Ereignis *OnGridUpdate* ausgelöst wird, die Methode *SetInventory* aufgerufen wird, um den aktuellen Wert des neuen Inventars zu berechnen. Weiteres zur Berechnung des Inventarwertes ist in diesem Abschnitt zu finden, ??.

```
1 EventManager.GridUpdate(idGrid);
```

Das Auslösen des *OnGridUpdate* Events wird durch den obigen Codeabschnitt erreicht. Dieser Codeabschnitt befindet sich in der *InventoryController* Klasse und wird aufgerufen, wenn ein neues Item hinzugefügt oder entfernt wird.

Insgesamt ermöglicht dieser simple Code eine lose Kopplung zwischen verschiedenen Teilen des Programms, indem Ereignisse verwendet werden, um auf bestimmte Aktionen zu reagieren, ohne dass die beteiligten Teile voneinander wissen müssen. Dadurch wird die Modularität, Erweiterbarkeit und Wartbarkeit der Anwendung gefördert.

#### 4.5.6 Das Knapsack Problem

→ SKREPEK

Dieser Abschnitt befasst sich mit dem Knapsack-Problem, ein Kombinatorisches Optimierungsproblem. Dazu wird auch die Problemstellung, welche Varianten und Typen des Problems es gibt, die zwei bekanntesten Algorithmen zum Lösen des Problems und anschließend die Implementierung um zu das Problem zu lösen, erläutert.

##### 4.5.6.1 Problemstellung

Das Knapsack-Problem befasst sich mit der Herausforderung, einen Rucksack optimal zu befüllen, unter der Ausnahme, dass dieser eine begrenzte Kapazität aufweist. Um ihn zu füllen, steht eine Reihe von Gegenständen zu Auswahl, von denen jeder einen individuellen Wert, der einen bestimmten Nutzen oder eine Wichtigkeit repräsentiert, hat. Jeder Gegenstand weist weiteres ein bestimmtes individuelles Gewicht, welches genau beschreibt, wie viel an Platz dieser Gegenstand im Rucksack einnimmt, auf.

Das grundlegende Ziel beim Knapsack-Problem besteht darin, eine Auswahl von Gegenständen zu treffen, die in den Rucksack passt und gleichzeitig den Gesamtwert der ausgewählten Gegenstände maximiert. Dabei darf die Summe der Gewichte der ausgewählten Gegenstände die Kapazität des Rucksacks nicht überschreiten. Dies führt zu einer Herausforderung, bei der eine perfekte Lösung gefunden werden muss, um den bestmöglichen Nutzen aus den verfügbaren Gegenständen zu ziehen.

##### 4.5.6.2 Typen des Knapsack-Problems

Das Knapsack-Problem kann in verschiedene Typen unterteilt werden, die jeweils *unterschiedliche Bedingungen* und *Anforderungen* haben. Nachfolgend werden die wichtigsten Typen des Knapsack-Problems erläutert.

##### 0/1 Knapsack-Problem

Das 0/1-Knapsack-Problem bezieht sich auf die effiziente Auswahl von Gegenständen, um den maximalen Gesamtwert in einem begrenzten Rucksack zu erreichen. Gegeben sind

$n$  Gegenstand, wobei jedem Gegenstand ein bestimmtes Gewicht ( $w$ ) und ein Wert ( $v$ ) zugeordnet ist. Außerdem steht ein Rucksack mit einer Kapazität ( $c$ ) zur Verfügung. Das Ziel besteht darin, jene Gegenstände in den Rucksack zu legen, sodass die Summe der Werte der ausgewählten Gegenstände maximal ist.

Es ist wichtig zu beachten, dass beim 0/1-Knapsack-Problem entweder ein Gegenstand *vollständig* in den Rucksack gepackt wird oder *überhaupt nicht*. Es gibt keine Möglichkeit, einen Gegenstand *teilweise* in den Rucksack zu legen. Diese Beschränkung erfordert eine sorgfältige Auswahl der Gegenstände, um den verfügbaren Platz im Rucksack optimal zu nutzen und gleichzeitig den Gesamtwert zu maximieren.<sup>31</sup>

### Fraktionales Knapsack-Problem

Das fraktionale Knapsack-Problem befasst sich mit der effizienten Verteilung von Gegenständen in einem Rucksack, um den Gesamtwert im Rucksack zu maximieren. Dabei werden die Gewichte und Werte von  $n$  Gegenständen gegeben, und das Ziel besteht darin, diese Gegenstände in einen Rucksack mit der *Kapazität*  $c$  zu legen, um den maximalen Gesamtwert zu erreichen. Im Gegensatz zum klassischen 0/1-Knapsack-Problem, bei dem entweder ein Gegenstand vollständig in den Rucksack gepackt wird oder nicht, erlaubt das fraktionale Knapsack-Problem das Aufteilen von Gegenständen, z.B.: Das halbieren eines Gegenstands, um den Gesamtwert im Rucksack zu maximieren. Diese Flexibilität ermöglicht es, eine perfekte Lösung zu finden, indem die Gegenstände entsprechend ihrer Wertigkeit und Gewichtung effizient verteilt werden.<sup>32</sup>

### Begrenztes Knapsack-Problem

Das begrenzte Knapsack-Problem bezieht sich auf die effiziente Auswahl von Gegenständen, um den maximalen Gesamtwert unter Berücksichtigung eines begrenzten Gewichts zu erreichen. Angenommen, es gibt  $n$  Gegenstände, wobei jedem Gegenstand ein bestimmtes Gewicht ( $w$ ) und ein Wert ( $v$ ) zugeordnet ist. Die Aufgabe besteht darin, den Gesamtwert zu maximieren, indem von jedem gegebenen Gegenstand nur eine begrenzte Anzahl von Instanzen, deren Gesamtgewicht nicht größer als das maximale Gewicht  $w$  ist, verwendet werden darf.<sup>33</sup>

### Unbegrenztes Knapsack-Problem

Das unbeschränkte Knapsack-Problem befasst sich mit der effizienten Auswahl von Gegenständen, um den maximalen Gesamtwert zu erreichen, wobei das Gesamtgewicht nicht größer als eine vorgegebene Kapazität ist. Angenommen, es gibt ein Rucksackgewicht  $c$  und eine Menge von  $n$  Gegenständen, von denen jeder einen bestimmten Wert ( $v$ ) und ein Gewicht ( $w$ ) hat. Das Ziel besteht darin, die maximale Menge zu berechnen, die genau dieses Gewicht erreichen kann.

Im Gegensatz zum Begrenzten Knapsack-Problem, bei dem die Anzahl der Instanzen eines Gegenstands begrenzt ist, können beim unbeschränkten Knapsack-Problem *beliebig viele* Instanzen *dieselben* Gegenstands verwendet werden. Dies ermöglicht eine flexiblere Auswahl und Nutzung der verfügbaren Gegenstände, um den maximalen Gesamtwert zu erzielen, der das vorgegebene Gewicht nicht überschreitet.<sup>34</sup>

---

<sup>31</sup>GeeksForGeeks, **0/1 Knapsack-Problem**

<sup>32</sup>GeeksForGeeks, **Fractional Knapsack Problem**

<sup>33</sup>GeeksForGeeks, **Bounded Knapsack Problem**

<sup>34</sup>GeeksForGeeks, **Unbounded Knapsack**

#### 4.5.6.3 Variationen des Knapsack-Problems

Im Laufe der Zeit hat das Knapsack-Problem verschiedene Variationen hervorgebracht, die jeweils unterschiedliche Aspekte und Einschränken des Problems berücksichtigen. Variationen beziehen sich hier auf unterschiedliche Varianten des Hauptproblems, also z.B. hat das 0/1 Knapsack-Problem verschiedene Varianten mit unterschiedlichen zusätzlichen Einschränkungen und Aspekten. Nachfolgend werden die bekanntesten Variationen erläutert

##### Mehrzieliges Knapsack-Problem

Das mehrzielige Knapsack-Problem erweitert das klassische Knapsack-Problem, indem es mehrere *Zielkriterien* berücksichtigt. Anstatt nur den Gesamtwert der ausgewählten Gegenstände zu maximieren, sollen nun mehrere Ziele *gleichzeitig* optimiert werden. Zum Beispiel könnte neben der *Maximierung* des Gesamtwerts auch die *Minimierung* des Gesamtgewichts oder anderer Kosten angestrebt werden. Diese Variation des Problems führt zu komplexeren Optimierungsaufgaben, da Kompromisse zwischen den verschiedenen Zielen gefunden werden müssen.<sup>35</sup>

##### Multidimensionales Knapsack-Problem

Beim multidimensionalen Knapsack-Problem hat jeder Gegenstand nicht nur ein Gewicht, sondern wird durch einen *m-dimensionalen* Vektor repräsentiert, der verschiedene *Merkmale* oder *Eigenschaften* des Gegenstands darstellt. Entsprechend ist auch die Kapazität des Rucksacks ein *m-dimensionaler* Vektor. Diese Variation des Problems entsteht häufig in realen Anwendungen, in denen die Gegenstände durch mehrere Merkmale charakterisiert werden, wie zum Beispiel *Größe, Form, Farbe* oder *Material*.<sup>36</sup>

##### Mehrere Knapsack-Probleme

Das multiple Knapsack-Problem *erweitert* das klassische Knapsack-Problem, indem es *mehrere* Rucksäcke oder Behälter einführt, in die die Gegenstände verteilt werden können. Im Gegensatz zum klassischen Problem, bei dem alle Gegenstände in einen einzelnen Rucksack gepackt werden müssen, können hier mehrere Rucksäcke genutzt werden, um die Gegenstände aufzunehmen. Diese Variation ist relevant in Situationen, in denen die Gegenstände auf verschiedene Weise organisiert oder verwendet werden sollen.<sup>37</sup>

##### Quadratisches Knapsack-Problem

Das quadratische Knapsack-Problem bezieht sich auf eine Variation, bei der das Ziel darin besteht, eine quadratische Zielfunktion zu maximieren, die von den ausgewählten Gegenständen abhängt. Diese Zielfunktion kann beispielsweise den Gesamtnutzen oder den Gesamtwert der ausgewählten Gegenstände darstellen und ist oft von quadratischer Form in Bezug auf die Variablen, die die Auswahl der Gegenstände repräsentieren. Die Lösung dieses Problems erfordert spezielle Techniken zur Bewältigung der quadratischen Struktur der Zielfunktion.<sup>38</sup>

---

<sup>35</sup>Wikipedia, **Multi-objective Knapsack-Problem**

<sup>36</sup>Wikipedia, **Multi-dimensional Knapsack-Problem**

<sup>37</sup>Wikipedia, **Multi Knapsack-Problem**

<sup>38</sup>Wikipedia, **Quadratic Knapsack-Problem**

## Geometrisches Knapsack-Problem

Beim geometrischen Knapsack-Problem stehen eine Reihe von geometrischen Objekten mit unterschiedlichen *Formen* und *Größen* zur Auswahl, die in einen *rechteckigen Rucksack* gepackt werden sollen. Das Ziel besteht darin, die Gegenstände so anzurichten, dass der verfügbare Platz im Rucksack optimal genutzt wird und der Gesamtwert der ausgewählten Gegenstände maximiert wird. Diese Variation des Knapsack-Problems erfordert eine *Berücksichtigung* der *geometrischen Eigenschaften* der Objekte und kann in verschiedenen Anwendungen wie Layoutdesign oder Packungsproblemen auftreten.<sup>39</sup>

### 4.5.6.4 Ansätze zur Lösung des Knapsack-Problems

Das Knapsack-Problem bietet Raum für eine Vielzahl von Lösungsansätzen, die sich in ihrer *Komplexität*, *Effizienz* und *Genauigkeit* unterscheiden können. Diese Ansätze reichen von einfachen *heuristischen* Methoden bis hin zu *komplexen optimierten* Algorithmen. Abgesehen von den bekanntesten Ansätzen wie den *dynamischen* und dem *greedy* Ansatz, die später genauer betrachtet werden, gibt es weitere interessante Möglichkeiten das Knapsack-Problem anzugehen.

Ein Ansatz besteht darin, das Problem in kleinere Teilprobleme zu zerlegen und diese dann unabhängig voneinander zu lösen. Durch die Kombination der Lösungen dieser Teilprobleme kann eine Gesamtlösung gefunden werden. Diese Methode ist besonders nützlich, wenn die Problemgröße groß ist und eine vollständige Suche nach einer optimalen Lösung zu aufwändig ist.

Eine weite Herangehensweise ist die Verwendung von *Metaheuristiken*, wie etwa *genetische Algorithmen* oder *Partikelschwarmoptimierung*. Die Algorithmen basieren auf biologischen oder sozialen Konzepten und verwenden probabilistische Techniken, um Lösungen zu finden, die möglicherweise nicht optimal, aber dennoch akzeptable sind. Sie eignen sich gut für komplexe Probleme, bei denen eine exakte Lösung schwer zu erreichen ist.<sup>40</sup>

Eine neuere Entwicklung in der Lösung des Knapsack-Problems ist der Einsatz von *künstlicher Intelligenz*. Durch den Einsatz von *Datenanalyse* und *statistischen Methoden* können Modelle trainiert werden, um Muster in den Eigenschaften der Gegenstände und den Anforderungen des Rucksacks zu erkennen und optimale Packstrategien vorherzusagen.

### 4.5.6.5 Dynamischer Programmieransatz

Der dynamische Programmieransatz ist eine leistungsfähige Methode zur Lösung des Knapsack-Problems, die auf der Idee beruht, das Problem in kleinere Teilprobleme zu zerlegen und die Lösungen dieser Teilprobleme systematisch zu kombinieren, um die optimale Gesamtlösung zu finden. Diese Methode eignet sich besonders gut für Probleme, bei denen Teilprobleme sich überlappen und dieselben Teillösungen verwendet werden können, um mehrere Teilprobleme zu lösen.

Der folgende Pseudocode veranschaulicht den dynamischen Algorithmus zur Lösung des Knapsack-Problems:

```

1 FUNCTION knapsackDynamic(weights[], values[], capacity)
2     n = length(weights)
3     DECLARE Tabelle[n + 1][capacity + 1]
4     FOR i FROM 0 TO n DO
5         FOR w FROM 0 TO capacity DO

```

<sup>39</sup>Wikipedia, **Geometric Knapsack-Problem**

<sup>40</sup>Hindawi, **Solving the 0/1 Knapsack Problem Using Metaheuristic and Neural Networks**, S. 7 ff.

```

6      IF i == 0 OR w == 0 THEN
7          Tabelle[i][w] = 0
8      ELSE IF weights[i-1] <= w THEN
9          Tabelle[i][w] = MAX(values[i-1] + Tabelle[i-1][w - weights[i-1]], 
10         Tabelle[i-1][w])
11     ELSE
12         Tabelle[i][w] = Tabelle[i-1][w]
13     END IF
14   END FOR
15 RETURN Tabelle[n][capacity]
16 END FUNCTION

```

Codeabschnitt 4.31: Dynamischer Algorithmus

Die Funktion `knapsackDynamic()` implementiert den dynamischen Programmieransatz zur Lösung des Knapsack-Problems. Sie akzeptiert drei Parameter: ein Array von Gewichten (*weights*), ein Array von Werten (*values*) und die Kapazität des Rucksacks (*capacity*).

Zuerst wird die Länge des Gewichtsarrays *n* berechnet, um die Anzahl der verfügbaren Gegenstände zu bestimmen. Dann wird eine Tabelle *Tabelle* mit *n+1* Zeilen und *capacity+1* Spalten initialisiert. Diese Tabelle dient dazu, die optimalen Werte für verschiedene Teilprobleme zu speichern.

Anschließend werden zwei verschachtelte Schleifen verwendet, um alle möglichen Kombinationen von Gegenständen und Gewichten zu durchlaufen. Dabei wird für jedes Teilproblem in der Tabelle der optimale Wert berechnet. Die innere Schleife iteriert über die möglichen Kapazitäten des Rucksacks, während die äußere Schleife die verfügbaren Gegenstände durchläuft.

Für jedes Teilproblem wird überprüft, ob der aktuelle Gegenstand in den Rucksack passt. Wenn ja, wird der Wert dieses Gegenstands zu dem Wert addiert, der erreicht werden kann, wenn der Rucksack ohne diesen Gegenstand gefüllt wird. Andernfalls wird der Wert aus der vorherigen Zeile der Tabelle übernommen, da der aktuelle Gegenstand nicht in den Rucksack passt.

Schließlich wird der Wert in der untersten rechten Zelle der Tabelle zurückgegeben, der den maximal erreichbaren Gesamtwert des Rucksacks darstellt.

Dieser Algorithmus nutzt die Eigenschaften der optimalen Teilstruktur und des Überlappungsprinzips aus, um eine effiziente Lösung des Knapsack-Problems zu finden. Durch die systematische Berechnung und Speicherung der optimalen Werte für Teilprobleme ermöglicht der dynamische Programmieransatz eine Zeitkomplexität von  $O(n \cdot capacity)$ , was für viele praktische Anwendungen akzeptabel ist.

### Aufbau und Interpretation der Tabelle der Teilprobleme

Die Tabelle der Teilprobleme spielt eine entscheidende Rolle bei der systematischen Lösung des Knapsack-Problems mithilfe des dynamischen Programmieransatzes. Sie ist eine zweidimensionale Matrix, die während des Algorithmusverlaufs generiert wird und die optimalen Lösungen für verschiedene Teilprobleme des Knapsack-Problems enthält. Die Matrix ist wie folgt strukturiert:

$$dp = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

Jede *Zeile* in der Matrix entspricht den verschiedenen verfügbaren Gewichtskapazitäten des Rucksacks, beginnend bei *null* und schrittweise bis zur *maximalen Kapazität*.

Jede *Spalte* in der Matrix repräsentiert die Anzahl der bereits *berücksichtigten Gegenstände*, wobei jede Spalte die *Teilprobleme* für eine zunehmende Anzahl von Gegenständen darstellt.

Der letzte Eintrag in dieser Tabelle ( $a_{mn}$ ) repräsentiert den errechneten maximal erreichbaren Wert.

Um anschließend eine Zelle  $a_{mn}$  dieser Matrix zu interpretieren und den maximalen erreichbaren Wert ablesen zu können, ist Folgendes wichtig:

1. **m** gibt an, wie viel *Gewicht* bereits im Rucksack verbraucht wurde. Je weiter fortgeschritten dieser Wert ist, daher desto weiter unten in der Tabelle, desto mehr Gewicht wurde bereits verwendet.
2. **n** gibt an, wie viele *Gegenstände* bereits betrachtet wurden. Je weiter fortgeschritten, daher, desto weiter rechts in der Tabelle, desto mehr Gegenstände wurden bereits berücksichtigt.

Die Einträge in der Matrix werden durch den *dynamischen Programmieransatz* berechnet, indem die optimalen Werte für Teilprobleme *schrittweise kombiniert* werden, um den Wert für größere Teilprobleme zu bestimmen.

### Beispiel

Angenommen, der Rucksack hat eine Kapazität von 5 und es sind die folgenden 5 Gegenstände (indiziert von 1 bis 5) mit den folgenden Werten und Gewichten gegeben:

Gegenstand	Wert	Gewicht
1	10	5
2	6	4
3	8	3
4	3	2
5	7	1

Auf der Grundlage dieser Informationen wird während des Durchlaufs des Knapsack-Algorithmus die Tabelle der Teilprobleme erzeugt, die wie folgt aussieht:

$$dp = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 & 6 & 10 \\ 0 & 0 & 0 & 8 & 8 & 10 \\ 0 & 0 & 3 & 8 & 8 & 11 \\ 0 & 7 & 7 & 10 & 15 & 15 \end{bmatrix}$$

Auf der Grundlage dieser Tabelle kann der Schluss gezogen werden, dass der berechnete maximale Wert, der mit den gegebenen Objekten erreicht werden kann, 15 beträgt.

#### 4.5.6.6 Greedy-Ansatz

Im Gegensatz zur dynamischen Lösung wählt der Greedy-Ansatz Gegenstände basierend auf bestimmten Kriterien aus, um eine lokale Optimierung zu erreichen. Hierbei wird in jedem Schritt diejenige Entscheidung getroffen, die im Moment am vorteilhaftesten erscheint, ohne jedoch die Gesamtoptimierung im Auge zu behalten.

Der folgende Pseudocode veranschaulicht den Greedy-Algorithmus zur Lösung des Knapsack-Problems:

```

1 FUNCTION knapsackGreedy(weights[], values[], capacity)
2     n = length(weights)
3     DECLARE items[n]
4     FOR i FROM 0 TO n DO
5         items[i] = (values[i] / weights[i], weights[i], values[i])
6     END FOR
7     SORT items by ratio in descending order
8     totalValue = 0
9     currentWeight = 0
10    FOR i FROM 0 TO n DO
11        IF currentWeight + items[i].weight <= capacity THEN
12            currentWeight += items[i].weight
13            totalValue += items[i].value
14        ELSE
15            ratio = (capacity - currentWeight) / items[i].weight
16            totalValue += ratio * items[i].value
17            BREAK
18        END IF
19    END FOR
20    RETURN totalValue
21 END FUNCTION

```

Codeabschnitt 4.32: Greedy Algorithmus

Die Funktion `knapsackGreedy()` nimmt wie der dynamische Ansatz drei Parameter an: ein Array von Gewichten (*weights*), ein Array von Werten (*values*) und die Kapazität des Rucksacks (*capacity*). Die Funktionsweise dieses Ansatzes ist wie folgt:

1. Zunächst wird für jeden Gegenstand das Verhältnis von Wert zu Gewicht berechnet und in einer Liste von Tupeln gespeichert. Jedes Tupel enthält das Wert-Gewichts-Verhältnis sowie das Gewicht und den Wert des entsprechenden Gegenstands.
2. Die Liste der Gegenstände wird basierend auf dem Verhältnis von Wert zu Gewicht in absteigender Reihenfolge sortiert, um die Gegenstände mit dem höchsten Verhältnis zuerst zu betrachten.
3. Der Algorithmus durchläuft die sortierte Liste der Gegenstände und versucht, jeden Gegenstand dem Rucksack hinzuzufügen. Dabei wird überprüft, ob das Hinzufügen des Gegenstands das Gewichtslimit des Rucksacks überschreitet. Falls dies der Fall ist, wird ein Teil des Gegenstands entsprechend dem verbleibenden verfügbaren Gewicht im Rucksack hinzugefügt.
4. Nachdem alle Gegenstände überprüft wurden, wird der Gesamtwert der im Rucksack enthaltenen Gegenstände zurückgegeben. Dies stellt die Lösung des Problems dar.

Der Greedy-Algorithmus bietet eine einfache und effiziente Lösung für das Knapsack-Problem, die jedoch nicht immer die perfekte Lösung garantiert. Durch die Auswahl der

Gegenstände basierend auf lokalen Kriterien kann der Algorithmus zu suboptimalen Ergebnissen führen, insbesondere wenn die Gegenstände stark voneinander abhängen oder das Gewichtslimit des Rucksacks sehr restriktiv ist.

#### 4.5.6.7 Anwendungen des Knapsack-Problems

Die grundlegende Problemstellung des Knapsack-Problems hat breite Anwendung in verschiedenen *wissenschaftlichen* und *industriellen* Bereichen gefunden, die sich mit der Allokation von Ressourcen beschäftigen, und bildet die Grundlage für eine Vielzahl von Algorithmen und Anwendungen.

Dieses Kapitel untersucht die Anwendungen des Knapsack-Problems und seine Bedeutung in verschiedenen Domänen. Von der *Logistik* über die *Finanzplanung* bis hin zum *Ressourcenmanagement* hat das Knapsack-Problem einen relevanten Einfluss auf die moderne Technologie und Wirtschaft.

Die Anwendungen des Knapsack-Problems lassen sich in folgende Hauptbereiche unterteilen:

1. **Logistik:** Der Knapsack-Algorithmus wird in der Logistik angewendet, um den Transport von Gütern mit begrenzten Kapazitäten zu optimieren. Indem er die bestmögliche Auswahl von Gütern trifft, ermöglicht er Logistikunternehmen, ihre Transportkosten zu minimieren und die Effizienz ihrer Lieferketten zu steigern. Dies kann die Beladung von Containern oder die Organisation von Waren in Lagern umfassen.
2. **Finanzplanung:** In der Finanzplanung wird der Knapsack-Algorithmus genutzt, um Portfolios von Investitionen zu optimieren. Investoren können mithilfe dieses Algorithmus eine Auswahl von Wertpapieren treffen, die das Risiko minimieren und den erwarteten Ertrag maximieren. Dies kann die Diversifizierung von Anlagen, die Auswahl von Aktien oder die Verwaltung von Fonds umfassen.
3. **Ressourcenmanagement:** Der Knapsack-Algorithmus wird im Ressourcenmanagement verwendet, um die effiziente Nutzung begrenzter Ressourcen sicherzustellen. Dies kann in der Produktion erfolgen, wo Arbeitskräfte, Maschinen und Materialien effizient zugewiesen werden müssen, oder im Projektmanagement, um Zeit und Budgets zu optimieren. Durch die Anwendung des Knapsack-Problems können Unternehmen ihre Produktivität steigern und Kosten senken.<sup>41</sup>

Die Untersuchung dieser Anwendungen veranschaulicht die Vielseitigkeit und Effektivität des Knapsack-Problems bei der Lösung komplexer Optimierungsprobleme. Darüber hinaus trägt die Anwendung des Knapsack-Algorithmus dazu bei, industrielle Abläufe zu verbessern, Kosten zu senken und die Effizienz in verschiedenen Bereichen zu steigern.

#### 4.5.6.8 Auswahl des Implementierungsansatzes

Die Wahl des Implementierungsansatzes für den Knapsack-Algorithmus für die in dieser Arbeit vorgestellten Anwendungen, ist von entscheidender Bedeutung für den Erfolg des Projekts. Moderne Softwaresysteme stehen vor einer Vielzahl von Herausforderungen und Anforderungen, die eine sorgfältige Auswahl eines geeigneten Algorithmus erforderlich machen.

Die Entscheidung für den Implementierungsansatz erfolgte nach einer eingehenden Analyse der Anforderungen der Anwendung und einer gründlichen Untersuchung der verfügbaren Algorithmen sowie ihrer Eigenschaften. Dabei wurden verschiedene Faktoren berücksichtigt, darunter die Notwendigkeit einer optimalen Lösung, die Effizienz der Algorith-

---

<sup>41</sup>Wikipedia, **Knapsack-Problem Anwendungen**

musausführung, die Flexibilität in Bezug auf unterschiedliche Problemvarianten und die Genauigkeit der berechneten Ergebnisse.

Diese Überlegungen werden anschließend ausführlich diskutiert und die Gründe für die Wahl des dynamischen Ansatzes als Implementierungsstrategie für den Knapsack-Algorithmus erläutert:

1. **Perfekte Lösungsgarantie:** Der dynamische Ansatz bietet die Möglichkeit, eine perfekte Lösung für das Knapsack-Problem zu garantieren. Dies ist besonders wichtig in Anwendungen, in denen eine genaue und zuverlässige Lösung erforderlich ist, um optimale Entscheidungen zu treffen.
2. **Effizienz:** Obwohl der dynamische Ansatz im Vergleich zum Greedy-Ansatz einen höheren Rechenaufwand erfordert, bietet er dennoch eine Laufzeit-effiziente Lösung für das Knapsack-Problem. Durch die Verwendung von dynamischer Programmierung können Teilprobleme effizient gelöst und die Gesamtlösung optimiert werden.
3. **Flexibilität:** Der dynamische Ansatz ist flexibel und kann auf verschiedene Varianten des Knapsack-Problems angewendet werden, einschließlich 0/1-Knapsack, unbeschränktem Knapsack und anderen Typen (siehe Abschnitt 4.5.6.2). Dadurch ist er vielseitig einsetzbar und kann an die spezifischen Anforderungen einer Anwendung angepasst werden.
4. **Genauigkeit:** Durch die Verwendung des dynamischen Ansatzes können exakte Werte für den maximal erreichbaren Wert des Rucksacks und die optimale Auswahl von Gegenständen berechnet werden. Dies ermöglicht eine präzise Bewertung und Planung basierend auf den berechneten Ergebnissen.

In Anbetracht dieser Überlegungen wurde der dynamische Ansatz als die geeignete Methode zur Implementierung des Knapsack-Algorithmus in dieser Anwendung gewählt. Obwohl nicht alle verfügbaren Lösungsansätze eine perfekte Lösung garantieren können, bieten sie dennoch eine gute Balance zwischen Effizienz und Flexibilität. Aus diesem Grund erscheint der dynamische Ansatz als ideale Wahl für die Behandlung des Knapsack-Problems in diesem Kontext, da er die Gewissheit einer optimalen Lösung bietet.

#### 4.5.6.9 Auswahl der Variante und Typ des Problems

Die Wahl der Variante und des Typs des zu implementierenden Knapsack-Problems ist von entscheidender Bedeutung. Je nach Variation und Typ ergeben sich unterschiedliche Aufgabenstellungen mit verschiedenen Bedingungen.

Nach einer gründlichen Analyse der Aufgabenstellung des Projekts wurde die endgültige Entscheidung für den Typ und die zugehörige Variante getroffen. Dabei wurde festgestellt, dass das klassische Knapsack-Problem mit einer Kombination der beiden Problem-Typen 0/1-Knapsack-Problem und Begrenztes-Knapsack-Problem die geeignete Wahl ist. Die Entscheidung für den Typ 0/1 des Knapsack-Problems wurde getroffen, da die Aufgabenstellung des Anwendungsszenarios vorsieht, dass der Benutzer Gegenstände in einem Inventar nur vollständig hinzufügen oder gar nicht hinzufügen kann. Und die Kombination mit dem begrenzten Knapsack-Problem deswegen, weil in dieser Anwendung eine Begrenzung an Instanzen eines Gegenständen, die dem Inventar hinzugefügt werden können, gesetzt ist.

#### 4.5.6.10 Der Knapsack Solver

Das Unity Game Objekt *Knapsack Solver*, wie in Abbildung 4.33 dargestellt, implementiert die Lösung des Knapsack-Problems und berechnet gleichzeitig das zusammengestellte Inventar des Benutzers. Es stellt Feedback über diese Berechnungen bereit. Dieses Game

Objekt mit der angehängten Skript-Komponente *Knapsack Solver* implementiert die Klasse *KnapsackSolver*, welche die Logik zur Berechnung des maximal erreichbaren Werts einer optimalen Lösung sowie den Wert des selbst zusammengestellten Inventars steuert.

Die Klasse arbeitet eng mit der Klasse *InventoryController* zusammen, um sicherzustellen, dass die Berechnungen stets auf Grundlage des aktuellen individuell zusammengestellten Inventars des Benutzers erfolgen. Diese Interaktion zwischen den beiden Klassen gewährleistet eine präzise und zeitnahe Verarbeitung der inventarbezogenen Informationen innerhalb der Anwendung.

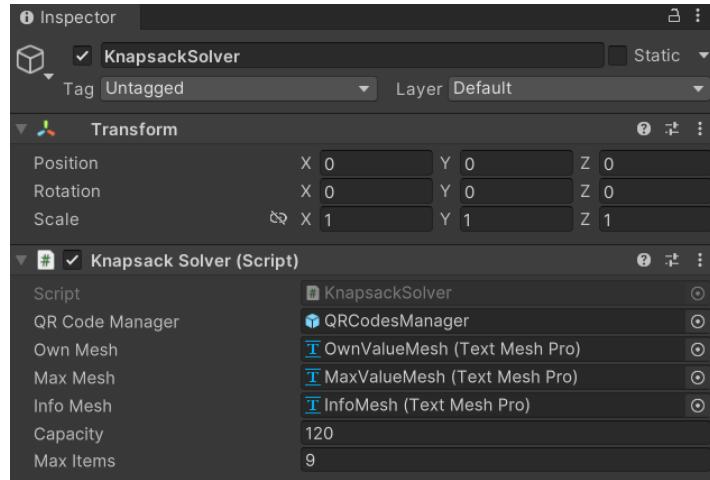


Abbildung 4.33: KnapsackSolver Objekt im Editor

Abbildung 4.33 zeigt das Unity-Game-Objekt im Unity Inspektor. Ebenfalls dargestellt ist die angehängte Skript-Komponente, die dem Game-Objekt zugeordnet ist. Innerhalb dieser Skript-Komponente sind die öffentlichen Klassenvariablen aufgeführt, die direkt im Editor übergeben und manipuliert werden können. Diese Variablen spielen eine entscheidende Rolle bei der Konfiguration und Steuerung des Knapsack Solvers. Zu den relevanten Variablen und ihrer Bedeutung gehören:

- **QRCodeManager:** Ein Verweis auf das *QRCodeManager* Game Objekt aus der Szene.
- **Own Mesh:** Referenz auf das *OwnValueMesh* TextMesh-Element aus dem Inventar Prefab
- **Max Mesh:** Referenz auf das *MaxValueMesh* TextMesh-Element aus dem Inventar Prefab
- **Info Mesh:** Referenz auf das *infoMesh* TextMesh-Element aus dem Inventar Prefab
- **Capacity:** Integerwert, der die Kapazität des Rucksacks festlegt.
- **Max Items:** Integerwert, der die maximale Anzahl an möglich platzierbaren Gegenständen im Inventar festlegt.

#### 4.5.6.11 Knapsack-Algorithmus Implementierung

Dieser Abschnitt behandelt die Implementierung der allgemeinen Variation des 0/1 Knapsack-Problems mittels des dynamischen Programmieransatzes. Die Funktion *KnapsackMaxValue()* innerhalb der Klasse *KnapsackSolver* implementiert den Algorithmus und ermittelt eine perfekte Lösung, indem sie die verwendeten Gegenstände verfolgt, um den maximalen Wert

zu erreichen. Die Funktion besteht grundsätzlich aus zwei Hauptabschnitten, um dieses Ergebnis zu erzielen:

1. **Algorithmus Implementation:** Zeilen 2 bis 23 aus Codeabschnitt 4.33 umfassen die Implementierung des Knapsack-Algorithmus.
2. **Backtracking:** Zeilen 24 bis 42 aus Codeabschnitt 4.33 dienen dazu, das Array `usedItems` zu erstellen, welches eine perfekte Lösung repräsentiert.

```

1 public int KnapsackMaxValue(out int[,] usedItems) {
2     int n = items.Count;
3     int[,] dp = new int[n + 1, capacity + 1];
4     bool[,] selected = new bool[n + 1, capacity + 1];
5     for (int i = 0; i <= n; i++) {
6         for (int w = 0; w <= capacity; w++) {
7             if (i == 0 || w == 0)
8                 dp[i, w] = 0;
9             else if (i <= maxItems && items[i].weight <= w) {
10                 int newValue = items[i].value + dp[i - 1, w - items[i].weight];
11                 if (newValue > dp[i - 1, w]) {
12                     dp[i, w] = newValue;
13                     selected[i, w] = true;
14                 } else {
15                     dp[i, w] = dp[i - 1, w];
16                     selected[i, w] = false;
17                 }
18             } else {
19                 dp[i, w] = dp[i - 1, w];
20                 selected[i, w] = false;
21             }
22         }
23     }
24     int[,] tempUsedItems = new int[3, 3];
25     int row = n;
26     int col = capacity;
27     int rowIndex = 0;
28     int colIndex = 0;
29     while (row > 0 && col > 0 && rowIndex < 3 && colIndex < 3) {
30         if (selected[row, col] && colIndex < maxItems) {
31             tempUsedItems[rowIndex, colIndex] = items[row].id;
32             col -= items[row].weight;
33             row--;
34             colIndex++;
35             if (colIndex >= 3) {
36                 colIndex = 0;
37                 rowIndex++;
38             }
39         } else {
40             row--;
41         }
42     }
43     usedItems = tempUsedItems;
44     return dp[n, capacity];
45 }
```

Codeabschnitt 4.33: Knapsack Algorithmus / Item Backtracking

Die vorliegende Implementierung des Lösungsansatzes für das Knapsack-Problem entspricht im Wesentlichen dem bereits erklärten Pseudocode. Ein entscheidender Unterschied besteht jedoch darin, dass diese spezielle Implementierung eine zusätzliche Bedingung berücksichtigt. Diese Bedingung besagt, dass bei der Berechnung des *maximal* erreichbaren

Werts und der perfekten Lösung höchstens 9 Gegenstände aus den verfügbaren Gegenständen einbezogen werden dürfen. Diese Einschränkung resultiert aus der Tatsache, dass der vorliegende Rucksack nur neun verfügbare Zellen hat, in denen ein Gegenstand platziert werden kann.

Um diese Anforderung zu erfüllen, wird die zusätzliche Bedingung in die Implementierung integriert. Dies geschieht in Zeile 9 des Codeabschnitts 4.33 durch folgende Bedingung:

$$i \leq \text{maxItems} \wedge \text{items}[i].\text{weight} \leq w \quad (4.1)$$

Während des Algorithmus wird zusätzlich ein Array aus booleschen Werten (`selected[] []`) erstellt, um die ausgewählten Gegenstände für den maximalen Wert zu markieren. Dieses Array spielt eine wichtige Rolle im zweiten Abschnitt der Funktion, um diese Gegenstände zurückzuverfolgen und die perfekte Lösung zusammenzustellen.

### Aufbau und Interpretation der *selected* Tabelle

Angenommen, der Rucksack hat eine Kapazität von 5 und es sind die folgenden 5 Gegenstände (indiziert von 1 bis 5) mit den folgenden Werten und Gewichten gegeben:

Gegenstand	Wert	Gewicht
1	10	5
2	6	4
3	8	3
4	3	2
5	7	1

Aufgrund dieser Angaben wird die `selected` Matrix nach Ausführung des Knapsack-Algorithmus basierend auf den ausgewählten Gegenständen gefüllt. Diese Matrix sieht dann folgendermaßen aus:

	0	1	2	3	4
0	false	false	false	false	false
1	false	false	false	false	true
2	false	false	false	true	true
3	false	false	true	true	true
4	false	true	true	true	true
5	false	true	true	true	true

Für die Interpretation der `selected`-Matrix ist es wichtig zu verstehen, wie sie funktioniert. Jede Zelle in dieser Matrix gibt an, ob der entsprechende Gegenstand in der perfekten Lösung des Knapsack-Problems enthalten ist (*true*) oder nicht (*false*). Als Beispiel zeigt die Zelle `selected[4][3]`, dass der Gegenstand 4 in der perfekten Lösung miteinbezogen wurde, als die Kapazität des Rucksacks 3 betrug. Diese Informationen ermöglichen folgende Schlussfolgerungen:

- Der Index **i** in `selected[i][j]` repräsentiert den Gegenstand, der in Betracht gezogen wird.
- Der Index **j** in `selected[i][j]` gibt die Kapazität des Rucksacks an, die für diese Teillösung verwendet wurde.

#### 4.5.6.12 Berechnung des zusammengestellten Rucksacks

Um dem Benutzer ein dementsprechendes Feedback zu geben und dadurch zu veranschaulichen, welchen Wert der selbst zusammengestellte Rucksack im Vergleich mit dem errechneten maximal erreichbaren Wert hat, wird hier eine Berechnung benötigt, um diesen Wert zu berechnen. Diese Berechnung wird in der Funktion `KnapsackInventoryValue()` gehandhabt.

```

1 public int KnapsackInventoryValue(int[,] inventory) {
2     if (inventory == null) {
3         throw new System.Exception("Inventory is null");
4     }
5     int totalValue = 0;
6     foreach (var item in items.Values) {
7         int itemId = item.id;
8         int itemValue = item.value;
9         for (int j = 0; j < inventory.GetLength(0); j++) {
10            for (int k = 0; k < inventory.GetLength(1); k++) {
11                if (inventory[j, k] == itemId) {
12                    totalValue += itemValue;
13                }
14            }
15        }
16    }
17    return totalValue;
18 }
```

Codeabschnitt 4.34: Funktion um eigenen Rucksack zu berechnen

Die Methode durchläuft mithilfe einer `foreach`-Schleife jeden Eintrag im Dictionary, welches alle vorhandenen Gegenstände enthält. Dadurch kann auf die ID und den Wert (`Value`) jedes Gegenstands zugegriffen werden. Anschließend iteriert die Methode durch das Array `usedItems`, welches eine perfekte Lösung repräsentiert, mittels zweier `for`-Schleifen, um jeden Eintrag in diesem Array zu überprüfen. Dabei wird festgestellt, ob an der jeweiligen Stelle `usedItems[i][j]` ein Wert gespeichert ist oder nicht. Falls das Array an der Indexposition `[i][j]` der ID des aktuellen Gegenstands entspricht, wird der Wert dieses Gegenstands zum Gesamtwert (`totalValue`) addiert.

Nach Abschluss der Berechnung des zusammengestellten Rucksacks wird dieser Wert zurückgegeben, um ihn im weiteren Verlauf der Applikation zu verwenden.

#### 4.5.7 Anzeigen der perfekten Lösung

→ SKREPEK

Im Anwendungsszenario "Knapsack-Problem" ist das Anzeigen einer perfekten Lösung von entscheidender Bedeutung, da dies dem Benutzer ermöglicht, eine perfekte Lösung zu visualisieren und zu verstehen, wie sie aussieht und welche Objekte sie enthält.

In diesem Abschnitt werden das Unity-Prefab *Best Solution Prefab* und die zugehörige Skript-Komponente im Detail beschrieben. Die Klasse *BestSolutionVisualizer* ist hier hauptsächlich für das Platzieren, Aktivieren und Deaktivieren der perfekten Lösung verantwortlich, um einen reibungslosen Ablauf zu gewährleisten. Es wird erklärt, wie das Prefab selbst aufgebaut ist, wie dieses Skript aufgerufen wird und wie genau diese Lösung platziert und gefüllt wird.

##### 4.5.7.1 Aufbau und Hirarchie des best Solution Prefabs

Das Design und die Struktur des Prefabs sind von entscheidender Bedeutung, da dieses Objekt als Grundlage für die Visualisierung einer perfekten Lösung dient und dem Benutzer

Feedback gibt. Die sorgfältige Ausarbeitung des Prefabs trägt wesentlich zur Gesamterfahrung des Benutzers bei, erleichtert die Übersicht und fördert eine reibungslose Interaktion. Das Prefab enthält nicht nur das einfache Inventar-Raster, sondern auch neun zusätzliche Elemente, die dem Inventar-Raster untergeordnet sind und jeweils innerhalb einer Zelle platziert sind.

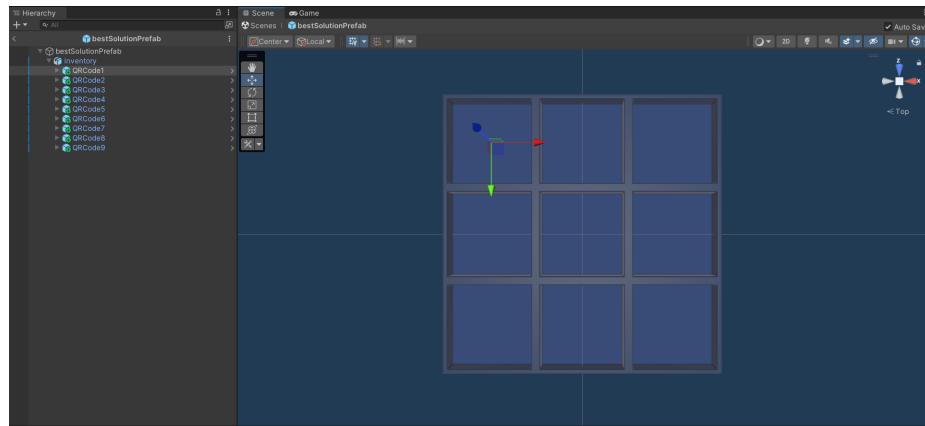


Abbildung 4.34: Prefab Hirarchie und Aufbau im Unity Editor

Die Abbildung 4.34 stellt das gesamte Prefab im Unity Editor visuell dar und gibt gleichzeitig einen Einblick in seine Hierarchie. Es wird deutlich, dass das Prefab aus einem Hauptmodell und neun Untermodellen besteht. Diese Modelle sind:

1. **inventory**: Das Raster, welches den eigentlichen Rucksack repräsentiert.
2. **QRItem**: Prefab, über das mithilfe der ID auf die 3D-Modelle zugegriffen wird, um diese darzustellen.

#### 4.5.7.2 Das Best Solution Prefab

Das Unity-Prefab **best Solution Prefab**, wie in Abbildung 4.35 dargestellt, implementiert die Logik zur Anzeige einer perfekten Lösung und gibt Feedback durch Visualisierung dieser Lösung. Dieses Prefab enthält die angehängte Skript-Komponente *Perfect Solution Visualizer*, die die Klasse **PerfectSolutionVisualizer** implementiert. Diese Klasse ist für das Platzieren und Ausfüllen der perfekten Lösung verantwortlich.

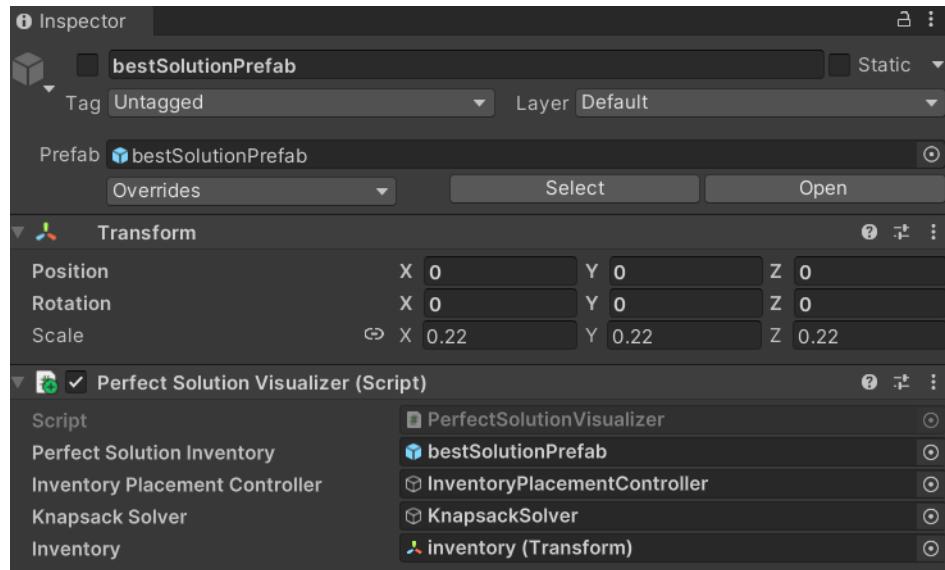


Abbildung 4.35: Game Objekt im Unity Editor

Auf dieser Abbildung ist das **bestSolutionPrefab** Game Objekt mit den zugehörigen Komponenten in Unity selbst zu sehen. An der Komponente, welches das Skript repräsentiert, ist zu sehen, dass dieses vier zu übergebende Objekte benötigt. Darunter sind die folgenden:

- **Perfect Solution Inventory:** Eine Referenz auf das *Prefab* für die perfekte Lösung.
- **Inventory Placement Controller:** Eine Referenz auf das *inventoryPlacementController* Game Objekt.
- **Knapsack Solver:** Eine Referenz auf das *Knapsack Solver* Game Objekt.
- **Inventory:** Eine Referenz auf das Raster Modell, dass im Best Solution Prefab enthalten ist.

Die korrekte Übergabe und Konfiguration dieser Objekte und Werte ist entscheidend für die Funktionalität dieser Klasse und die tatsächliche Darstellung der perfekten Lösung.

#### 4.5.7.3 Auslöser des Skripts

Die Anzeige der perfekten Lösung wird durch einen einfachen Knopfdruck ausgelöst, der sich innerhalb des *Inventar Prefabs* befindet. Wenn der Benutzer auf diesen Knopf drückt, wird das Skript zur Anzeige der perfekten Lösung gestartet. Dieses Skript führt die notwendigen Funktionen und Operationen aus, um die perfekte Lösung zu visualisieren.

Die Abbildung 4.36 zeigt die Verknüpfung des Scripts mit dem Knopf und verdeutlicht den Zusammenhang zwischen dem Knopf im *Inventar Prefab* und dem Start des Skripts.

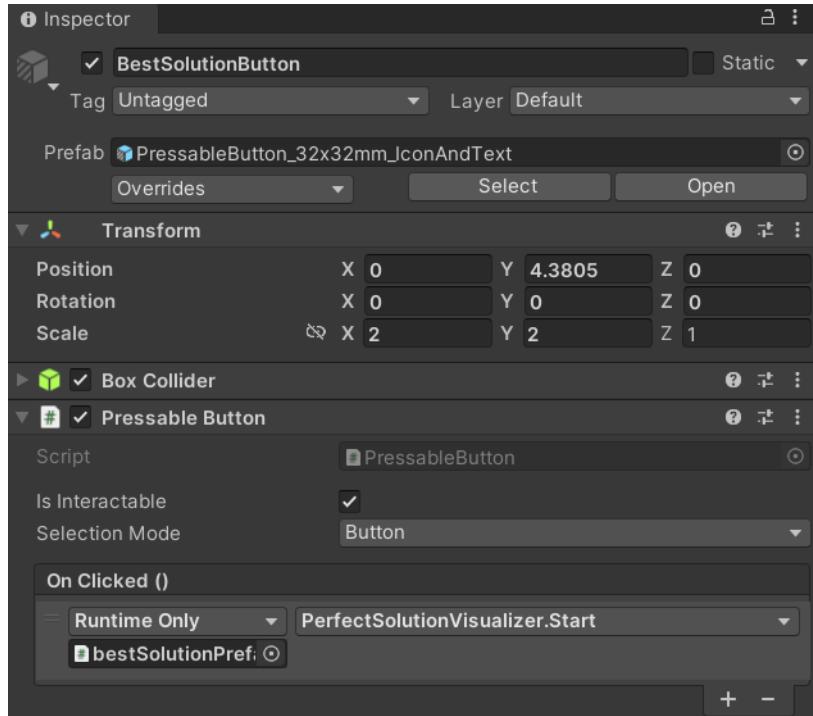


Abbildung 4.36: Script Aufruf bei Knopfdruck

In dieser Abbildung ist zu beachten, dass das interaktive Betätigen einer Schaltfläche durch die mitgelieferte Skript-Komponente *Pressable Button* ermöglicht wird. Diese Komponente stellt die Funktion `OnClicked()` bereit, die das einfache Betätigen der Schaltfläche umsetzt. Diese Funktion löst dann die `Start()` Funktion der `bestSolutionVisualizer` Klasse aus.

#### 4.5.7.4 Start des Perfect Solution Visualizer

Nachdem der Benutzer den **Best Solution Button** betätigt hat, wird der entscheidende Prozess zur Anzeige der perfekten Lösung gestartet. Dieser Prozess wird durch die Ausführung der Funktion `Start()` ausgelöst, wie im folgenden Codeabschnitt 4.35 dargestellt.

```

1 public void Start() {
2     isClicked = !isClicked;
3     if (isClicked == true) {
4         anchorScript = inventoryPlacementObject.GetComponent<
5             InventoryPlacementController>();
6         originalInventoryPosition = anchorScript.objectPosition;
7         knapsackSolver = KnapsackAlgoObject.GetComponent<KnapsackSolver>();
8         perfectSolution = knapsackSolver.usedItems;
9         setNewPosition();
10        inventoryObject.SetActive(true);
11        fillInventory();
12    } else {
13        inventoryObject.SetActive(false);
14    }
}

```

Codeabschnitt 4.35: PerfectSolutionVisualizer Start

Diese Funktion dient nicht nur als Auslöser für das Einblenden der Lösung, sondern auch für das Ausblenden der Lösung. Eine zentrale Aufgabe besteht darin, sicherzustellen, dass

der Benutzer die Lösung je nach Bedarf ein- oder ausblenden kann. Um dies zu erzielen wird bei jedem Scriptaufruf die boolsche Variable `isClicked`, die den aktuellen Stand der Sichtbarkeit des Modells repräsentiert, invertiert.

Abhängig von diesem boolschen Wert wird im weiteren Verlauf der Funktion entschieden, ob die perfekte Lösung sichtbar werden soll (`true`) oder nicht (`false`). Um das Prefab dann anschließend an der korrekten Position platzieren zu können wird auf die Positionsinformation des originalen platzierten Inventars-Prefabs (`objectPlaced`) zugegriffen und ebenfalls wird auf das Array, welches die errechnete perfekte Lösung repräsentiert (`usedItems`) zugegriffen um das Prefab im weiteren Verlauf anhand dieser füllen zu können. Abschließend werden die beiden Funktion `setNewPosition()` und `fillInventory()` aufgerufen die dies abschließen.

#### 4.5.7.5 Prefab der perfekten Lösung platzieren

Die Platzierung des Prefabs der perfekten Lösung ist wichtig, um zu garantieren, dass es für den Benutzer leicht und angenehm ist, diese zu sehen. Um dies zu garantieren, wird auf die Original-Position des Inventar-Objekts (`originalInventoryPosition`) zugegriffen und aufgrund dessen eine neue Position errechnet. Um es für den Benutzer jedoch angenehmer, d.h. besser für die allgemeine Benutzererfahrung, zu machen, die perfekte Lösung besser sehen zu können, wird abschließend die Rotation dieses Objekts geändert, um es um -45 Grad entlang der x-Achse zu kippen.

```

1 private void setNewPosition() {
2     Vector3 newPosition = originalInventoryPosition + Vector3.forward * 0.395f +
3         Vector3.up * 0.16f;
4     inventoryObject.transform.position = newPosition;
5     Quaternion objectRotation = Quaternion.Euler(-45f, 0f, 0f);
6     inventoryObject.transform.rotation = objectRotation;
7 }
```

Codeabschnitt 4.36: Neue Position setzen

#### 4.5.7.6 Prefab füllen

Aufgrund der Beschaffenheit des zuvor platzierten Objekts, das lediglich ein leeres Inventar-Objekt darstellt, ist es notwendig, dieses im nächsten Schritt mit den passenden Elementen zu füllen oder die geeigneten Modelle zu aktivieren, welche die perfekte Lösung repräsentieren. Diese Aufgabe wird durch den Aufruf der `fillInventory()` Funktion im weiteren Verlauf des Skripts realisiert.

```

1 private void fillInventory() {
2     for (int i = 0; i < numRows; i++) {
3         for (int j = 0; j < numColumns; j++) {
4             int id = perfectSolution[i, j];
5             if (perfectSolution[i, j] == 0)
6                 continue;
7             else {
8                 string qrCodeName = "QRCode" + (i * numColumns + j + 1);
9                 Transform qrCodeTransform = inventory.Find(qrCodeName);
10                if (qrCodeTransform != null) {
11                    Transform childTransform = qrCodeTransform.Find(id.ToString());
12                    if (childTransform != null) {
13                        childTransform.gameObject.SetActive(true);
14                    }
15                }
16            }
17        }
18    }
19 }
```

```

16     }
17 }
18 }
19 }
```

Codeabschnitt 4.37: Inventar fuellen

Diese Funktion `fillInventory()` durchläuft das zweidimensionale Array `perfectSolution`, das die perfekte Lösung darstellt. Für jeden Eintrag wird geprüft, ob die ID des Elements ungleich `Null` ist. Falls ja, wird ein eindeutiger Bezeichner für das QRItem generiert und das entsprechende Unterelement des QRItems anhand der ID im Inventar aktiviert. Ansonsten wird die Schleife übersprungen. Diese Funktion ermöglicht somit die Befüllung des Inventars mit den entsprechenden Modellen gemäß der perfekten Lösung.

#### 4.5.7.7 Zustand nach Knopfdruck

Nachdem der Benutzer den `Best Solution Button` betätigt hat und die entsprechende Skript-Komponente erfolgreich ausgeführt wurde, wird die perfekte Lösung platziert und vervollständigt. Anschließend erhält der Benutzer eine Ansicht, wie in Abbildung 4.37 dargestellt.

Abbildung 4.37: Perfekte Lösung

## 4.6 Performance und Qualitätssicherung

Die Sicherstellung einer hohen Performance und Qualität in Softwareprojekten ist entscheidend für den Erfolg und die Zufriedenheit der Nutzer. Ein zentraler Bestandteil dieser Bemühungen sind *Unit-Tests*, die sicherstellen, dass einzelne Komponenten korrekt funktionieren und den Erwartungen entsprechen. Im Rahmen dieses Projekts wurden Unit-Tests zur Überprüfung des Knapsack-Algorithmus (??), der für das Projekt von entscheidender Bedeutung ist, genutzt. Die Verwendung des Unity Test Frameworks ermöglichte es, diese Tests effizient zu erstellen und auszuführen.

→ HAYLAZ

### 4.6.1 Unity Test Framework

Das *Unity Test Framework* stellt eine leistungsstarke Suite von Werkzeugen bereit, mit der Entwickler umfassende Tests für ihre Unity-Skripte schreiben und durchführen können. Von der Überprüfung von Variablen bis hin zur Validierung von erwarteten Ergebnissen bietet das Framework eine breite Palette von Funktionen, um die Qualität von Unity-Anwendungen zu gewährleisten.

#### 4.6.1.1 Test-Runner

Der *Test-Runner* ist ein unverzichtbares Werkzeug innerhalb des *Unity Test Frameworks*, das Entwicklern ermöglicht, Tests einfach zu organisieren, auszuführen und zu überwachen. Durch die intuitive Benutzeroberfläche können Testfälle gruppiert, in spezifischen Reihenfolgen ausgeführt und detaillierte Berichte über den Testdurchlauf generiert werden. Die visuelle Darstellung in Abbildung ?? bietet einen schnellen Überblick über die Testergebnisse und ermöglicht eine präzise Analyse der Codequalität.

In der Abbildung ?? ist der *Test-Runner* dargestellt. Dort sind auch die implementierten Tests dieses Projekts ersichtlich, insgesamt vier Tests mit mehreren TestCases. Weitere Details dazu finden sich im Abschnitt ??.

### 4.6.2 Testfälle

Dieser Abschnitt bietet eine detaillierte Beschreibung der durchgeführten Tests. Alle einzelnen Testfälle sind innerhalb der Klasse *KnapsackSolverTests* (4.38) organisiert. Die Test-Klasse fungiert als eine Sammlung von Methoden, die verschiedene Aspekte einer spezifischen Funktionalität überprüfen, nämlich den *Knapsack Algorithmus*.

```

1 [TestFixture]
2 public class KnapsackSolverTests
3 {
4 // Eine Variable zur Speicherung der KnapsackSolver-Instanz
5 private KnapsackSolver knapsackSolver;
6
7 // Die SetUp-Methode wird vor jedem Testfall ausgeführt
8 [SetUp]
9 public void SetUp()
10 {
11 // Erstellen eines GameObjects für den KnapsackSolver
12 GameObject solverObject = new GameObject("KnapsackSolverObject");
13
14 // Dem GameObject wird eine Instanz von KnapsackSolver hinzugefügt
15 knapsackSolver = solverObject.AddComponent<KnapsackSolver>();
16 }
17
18 // Hier folgen die einzelnen Testfälle...
19 }
```

Codeabschnitt 4.38: Auszug aus der Test-Klasse

Wie ersichtlich ist, werden verschiedene Attribute verwendet, die vom Unity Test Framework (4.6.1) bereitgestellt werden. Diese Attribute haben bestimmte Zwecke und Funktionalitäten innerhalb des Testframeworks. Das *TestFixture-Attribut* kennzeichnet die Klasse *KnapsackSolverTests* als Testfixture, was dem Framework signalisiert, dass die Klasse Tests enthält, die ausgeführt werden sollen. Das *setUp-Attribut* bewirkt, dass vor jedem Test die Methode *setUp()* ausgeführt wird. Der Inhalt der SetUp-Methode stellt sicher, dass stets eine neue Instanz des *KnapsackSolvers* (??) erstellt wird, wodurch die Funktionalität der Tests immer in einer sauberen Ausgangssituation getestet werden kann.

Diese Struktur gewährleistet eine konsistente Umgebung vor jedem Testfall und ermöglicht es, dass die Tests isoliert und unabhängig voneinander ausgeführt werden können.

#### 4.6.2.1 Aufbau eines einzelnen Testfalls

```

1 [Test]
2 public void Knapsack.MaxValue_NoItems_ReturnsZero()
3 {
4 // Arrange
5 knapsackSolver.items = new Dictionary<int, QRData>();
6 knapsackSolver.capacity = 100;
7
8 // Act
9 int maxValue = knapsackSolver.Knapsack.MaxValue(out int[,] usedItems);
10
11 // Assert
```

```

12 Assert.AreEqual(0, maxValue, "The maximum value should be zero when there are no items
   .");
13 }

```

Codeabschnitt 4.39: einzelner Testfall

Ein Testfall im *Unity Test Framework* folgt einem bestimmten Aufbau, um sicherzustellen, dass die Funktionalität, die getestet werden soll, korrekt überprüft wird. Im obigen Beispiel (Code 4.39) ist ein Testfall dargestellt, der sicherstellt, dass der maximale Wert des Rucksacks korrekt berechnet wird, wenn keine Gegenstände im Rucksack vorhanden sind. Der Testfall besteht aus drei Hauptteilen: *Arrange*, *Act* und *Assert*.

Im Vorbereitungsabschnitt (*Arrange*) werden die erforderlichen Vorbedingungen für den Testfall eingerichtet. In diesem Fall werden dem KnapsackSolver keine Gegenstände hinzugefügt, und die Kapazität des Rucksacks wird auf 100 gesetzt.

Im Ausführungsabschnitt (*Act*) wird die tatsächliche Methode oder Funktionalität aufgerufen, die getestet werden soll. Hier wird die Methode `Knapsack.MaxValue` des KnapsackSolver-Objekts aufgerufen, um den maximalen Wert des Rucksacks zu berechnen.

Im Überprüfungsabschnitt (*Assert*) wird das erwartete Ergebnis mit dem tatsächlich berechneten Ergebnis verglichen. In diesem Fall wird überprüft, ob der maximal berechnete Wert tatsächlich null ist, da keine Gegenstände im Rucksack vorhanden sind. Wenn der Test fehlschlägt, wird eine entsprechende Fehlermeldung ausgegeben.

Dieser Aufbau ermöglicht eine klare und strukturierte Organisation von Testfällen, was die Lesbarkeit und Wartbarkeit des Testcodes verbessert.

#### 4.6.2.2 Testen von Einzelgegenständen im Rucksack

```

1 [TestCase(20, 50, 30, ExpectedResult = 50,
2   TestName = "Single item within capacity returns expected value.")]
3 [TestCase(30, 50, 20, ExpectedResult = 0,
4   TestName = "Single item exceeding capacity returns zero value.")]
5 [TestCase(30, 50, 30, ExpectedResult = 50,
6   TestName = "Single item exact capacity returns expected value.")]
7 public int Knapsack.MaxValue_SingleItem>ReturnsExpectedValue(
8   int weight, int value, int capacity)
9 {
10  // Arrange
11  var item = new QRData { id = 1, weight = weight, value = value };
12  knapsackSolver.items = new Dictionary<int, QRData> { { 1, item } };
13  knapsackSolver.capacity = capacity;
14
15  // Act
16  return knapsackSolver.Knapsack.MaxValue(out _);
17 }

```

Codeabschnitt 4.40: Testfall: einzelne Gegenstände

Die oben aufgeführten Testfälle zeigen verschiedene Szenarien zur Berechnung des maximalen Werts eines Rucksacks mit nur einem Gegenstand. Jeder Testfall wird mittels des `[TestCase]`-Attributs definiert und enthält das Gewicht, den Wert und die Kapazität des Rucksacks als Parameter. Das erwartete Ergebnis wird ebenfalls angegeben, um sicherzustellen, dass die berechneten Werte den Erwartungen entsprechen. Im Vorbereitungsabschnitt werden die notwendigen Vorbedingungen festgelegt, indem ein einzelner Gegenstand mit den angegebenen Gewicht und Wert dem KnapsackSolver hinzugefügt wird. Die Kapazität des Rucksacks wird entsprechend gesetzt. Im Ausführungsabschnitt wird die Methode `Knapsack.MaxValue` aufgerufen, um den maximalen Wert des Rucksacks

zu berechnen. Das zurückgegebene Ergebnis wird überprüft und mit dem erwarteten Ergebnis verglichen. Durch diese Testfälle wird sichergestellt, dass die Methode zur Berechnung des maximalen Werts unter verschiedenen Bedingungen korrekt arbeitet.

Im zweiten Testfall wird die Methode `KnapsackMaxValue_SingleItem_ReturnsExpectedValue` verwendet, die mit dem `[TestCase]`-Attribut versehen ist, um verschiedene Szenarien für die Berechnung des maximalen Werts eines Rucksacks mit nur einem Gegenstand zu testen. Im Gegensatz zum ersten Testfall (4.39), der die `[Test]`-Annotation verwendet, um einen einzelnen Testfall zu definieren, wird hier die `[TestCase]`-Annotation verwendet, um mehrere Testfälle innerhalb derselben Methode zu definieren.

Die Verwendung von `[TestCase]` ermöglicht eine kompaktere Darstellung mehrerer ähnlicher Testfälle, wodurch der Code übersichtlicher wird. Jeder Testfall wird durch die Parameter definiert, die an die Methode übergeben werden, und das erwartete Ergebnis wird durch das `ExpectedResult`-Attribut angegeben. Das `TestName`-Attribut trägt auch der Übersichtlichkeit und der Lesbarkeit bei. Dieser Name wird dann auch im *Test-Runner* (4.6.1.1) angezeigt, mit dem entsprechenden Bericht zum TestCase.

#### 4.6.2.3 Testen von mehreren Gegenständen im Rucksack

```

1 [TestCase(10, ExpectedResult = 0,
2   TestName = "All items exceeding capacity return zero value.")]
3 [TestCase(90, ExpectedResult = 220,
4   TestName = "All items within capacity return expected value.")]
5 [TestCase(70, ExpectedResult = 170,
6   TestName = "Multiple items within capacity, return optimal selection.")]
7 public int KnapsackMaxValue_MultipleItems_ReturnsExpectedValue(
8   int capacity)
9 {
10  // Arrange
11  var items = new Dictionary<int, QRData>
12  {
13    { 1, new QRData { id = 1, weight = 20, value = 50 } },
14    { 2, new QRData { id = 2, weight = 30, value = 70 } },
15    { 3, new QRData { id = 3, weight = 40, value = 100 } }
16  };
17  knapsackSolver.items = items;
18  knapsackSolver.capacity = capacity;
19
20  // Act
21  return knapsackSolver.KnapsackMaxValue(out _);
22 }
```

Codeabschnitt 4.41: Testfall: mehrere Gegenstände

Dieser Codeabschnitt demonstriert Testfälle für die Berechnung des maximalen Werts eines Rucksacks, wenn mehrere Gegenstände hinzugefügt werden. Ähnlich wie im vorherigen Abschnitt werden auch hier die Testfälle mit dem `[TestCase]`-Attribut definiert, wobei jedes Testcase nur die Kapazität des Rucksacks als Parameter hat und das erwartete Ergebnis angibt.

Im Vorbereitungsabschnitt (*Arrange*) werden die notwendigen Vorbedingungen festgelegt, indem eine Sammlung von Gegenständen erstellt wird. Jeder Gegenstand wird durch ein `QRData`-Objekt repräsentiert, das Gewicht und Wert enthält. Diese Gegenstände werden dann dem `knapsackSolver` hinzugefügt, und die Kapazität des Rucksacks wird entsprechend gesetzt.

Im Ausführungsabschnitt (*Act*) wird die Methode `KnapsackMaxValue` aufgerufen, um den maximalen Wert des Rucksacks zu berechnen. Das zurückgegebene Ergebnis wird dann

überprüft und mit dem erwarteten Ergebnis verglichen.

#### 4.6.2.4 Performance-Messung

```

1 [Test]
2 public void KnapsackPerformanceReportItemRange(){}
3 {
4 // Überschrift für den Leistungsbericht ausgeben
5 UnityEngine.Debug.Log("Item Count | Elapsed Time (ms)");
6 UnityEngine.Debug.Log("-----");
7
8 // Schleife durch verschiedene Gegenstandanzahlen
9 for (int itemCount = 50; itemCount <= 1000; itemCount += 50)
10 {
11 // Oberen Grenzwert für das Gegenstandsgewicht und die Kapazität basierend auf der
12 // Gegenstandanzahl berechnen
12 int weightUpperBound = itemCount * 2;
13 int capacity = itemCount * 5;
14
15 // Zufällige Gegenstände generieren
16 var items = GenerateRandomItems(itemCount, weightUpperBound);
17
18 // Gegenstände und Kapazität für den Rucksackkäufer festlegen
19 knapsackSolver.items = items;
20 knapsackSolver.capacity = capacity;
21
22 // Stoppuhr starten, um die verstrichene Zeit zu messen
23 var stopwatch = Stopwatch.StartNew();
24
25 // Die zu testende Methode aufrufen
26 knapsackSolver.KnapsackMaxValue(out _);
27
28 // Stoppuhr anhalten nach dem Methodenaufruf
29 stopwatch.Stop();
30
31 // Anzahl der Gegenstände und verstrichene Zeit für diese Iteration ausgeben
32 UnityEngine.Debug.Log($"{itemCount,-10} | {stopwatch.ElapsedMilliseconds,-15}");
33 }
34 }
```

Codeabschnitt 4.42: Codeabschnitt: Performance Messung bei hoher Anzahl an Gegenständen

Der vorliegende Codeabschnitt implementiert eine Testmethode namens *KnapsackPerformanceReport*, die darauf abzielt, die Leistung des Knapsack Algorithmus unter verschiedenen Lastszenarien zu bewerten. Der Code ist auf die Performance-Messung des Algorithmus fokussiert und zielt darauf ab, die Laufzeit des Algorithmus in Abhängigkeit von der Anzahl der Gegenstände im Rucksack zu erfassen.

Die Methode beginnt mit der Ausgabe einer Überschrift, die das Format des folgenden Leistungsberichts festlegt. Im Hauptteil der Methode wird eine Schleife durchlaufen, die verschiedene Anzahlen von Gegenständen im Rucksack simuliert. Die Anzahl der Gegenstände steigt schrittweise von 50 bis 1000, wobei in jedem Schritt 50 Gegenstände hinzugefügt werden. Innerhalb der Schleife werden zunächst der obere Grenzwert für das Gewicht der Gegenstände und die Kapazität des Rucksacks basierend auf der aktuellen Anzahl von Gegenständen berechnet. Anschließend werden zufällige Gegenstände generiert, die in den Rucksack aufgenommen werden sollen. Die generierten Gegenstände sowie die Kapazität des Rucksacks werden dann dem Algorithmus zugewiesen. Um die Leistung des

Algorithmus zu messen, wird für jede Iteration der Schleife eine Stoppuhr gestartet, bevor die Methode "Knapsack.MaxValue" aufgerufen wird, um den maximalen Wert des Rucksacks zu berechnen. Nach Beendigung des Methodenaufrufs wird die Stoppuhr gestoppt, und die verstrichene Zeit wird zusammen mit der Anzahl der Gegenstände im Rucksack protokolliert.

```

1 [Test]
2 public void KnapsackPerformanceReportRangeIteration()
3 {
4 // Äberschrift fÃ¼r den Leistungsbericht ausgeben
5 UnityEngine.Debug.Log("Iteration | Elapsed Time (ms)");
6 UnityEngine.Debug.Log("-----");
7
8 // Konstante fÃ¼r die Anzahl der GegenstÃ¤nde
9 int itemCount = 20;
10
11 // Oberen Grenzwert fÃ¼r das Gegenstandsgewicht basierend auf der Gegenstandanzahl
12 // berechnen
12 int weightUpperBound = itemCount * 2;
13
14 // Schleife fÃ¼r 1000 Mal wiederholen
15 for (int i = 1; i <= 500; i++)
16 {
17 // Oberen Grenzwert fÃ¼r die KapazitÃ¤t basierend auf der Gegenstandanzahl berechnen
18 int capacity = itemCount * 5;
19
20 // ZufÃ¤llige GegenstÃ¤nde generieren (festgelegte Anzahl von GegenstÃ¤nden)
21 var items = GenerateRandomItems(itemCount, weightUpperBound);
22
23 // GegenstÃ¤nde und KapazitÃ¤t fÃ¼r den RucksacklÃ¶ser festlegen
24 knapsackSolver.items = items;
25 knapsackSolver.capacity = capacity;
26
27 // Stoppuhr starten, um die verstrichene Zeit zu messen
28 var stopwatch = Stopwatch.StartNew();
29
30 // Die zu testende Methode aufrufen
31 knapsackSolver.Knapsack.MaxValue(out _);
32
33 // Stoppuhr anhalten nach dem Methodenaufruf
34 stopwatch.Stop();
35
36 // Durchlaufnummer und verstrichene Zeit fÃ¼r diese Iteration ausgeben
37 UnityEngine.Debug.Log($"{i,-9} | {stopwatch.ElapsedTicks / (Stopwatch.Frequency /
(1000.0 * 1000.0)):F2}")

```

Codeabschnitt 4.43: Codeabschnitt: Performance Messung bei 500 Aufrufen von 20 GegenstÃ¤nden

Dieser Codeabschnitt *KnapsackPerformanceReportRangeIteration* (4.43) unterscheidet sich vom anderen Codeabschnitt *KnapsackPerformanceReportItemRange* (4.42) hauptsächlich in Bezug auf die Parameter, die für die Leistungsmessung verwendet werden.

Im Abschnitt *KnapsackPerformanceReportRangeIteration* wird die Leistung des Knapsack-Algorithmus bei einer festen Anzahl von 20 Gegenständen und 500 Durchläufen gemessen. Dies bedeutet, dass die Anzahl der Gegenstände konstant bleibt, während die Iterationen variiert werden, um die Konsistenz der Messung sicherzustellen und die Auswirkungen wiederholter Durchläufe zu beobachten. Im Gegensatz dazu bewertet der Abschnitt *KnapsackPerformanceReportItemRange* die Leistung des Algorithmus bei einer variablen Anzahl von Gegenständen, wobei die Anzahl der Gegenstände von 50 bis 1000 in Schritten

von jeweils 50 erhöht wird. Dies ermöglicht die Untersuchung der Leistung des Algorithmus bei verschiedenen Lastszenarien und die Identifizierung möglicher Zusammenhänge zwischen der Anzahl der Gegenstände und der Ausführungszeit des Algorithmus.

Darüber hinaus verwendet der Abschnitt KnapsackPerformanceReportItemRange eine andere Methode zur Messung der verstrichenen Zeit, indem die Methode ElapsedMilliseconds der Stopwatch-Klasse verwendet wird, um die Zeit in Millisekunden zu erfassen. Im Gegensatz dazu verwendet der Abschnitt KnapsackPerformanceReportRangeIteration die Berechnung der verstrichenen Ticks und deren Umrechnung in Mikrosekunden für eine genauere Messung.

Basierend auf den Performance-Messungen können folgende Schlussfolgerungen gezogen werden:

- **Laufzeitverhalten:** Die Laufzeit des Algorithmus steigt mit zunehmender Anzahl der Gegenstände im Rucksack. Dies ist zu erwarten, da der Algorithmus mehr Berechnungen durchführen muss, um den maximalen Wert des Rucksacks zu bestimmen, wenn mehr Gegenstände vorhanden sind.
- **Nichtlineares Wachstum:** Die Laufzeit des Algorithmus scheint nicht linear mit der Anzahl der Gegenstände zu wachsen. Dies deutet darauf hin, dass der Algorithmus möglicherweise eine Laufzeitkomplexität hat, die größer als linear ist, wie beispielsweise  $O(n \log n)$  oder sogar  $O(n^2)$ .
- **Ausreißer:** Es gibt einige Ausreißer in den gemessenen Zeiten. Die zufällig generierten Werte könnten bestimmte Bedingungen schaffen, die den Algorithmus vor größere Herausforderungen stellen. Zum Beispiel könnten in bestimmten Fällen die generierten Gegenstände ein ungünstiges Muster aufweisen, das dazu führt, dass der Algorithmus mehr Zeit benötigt, um den optimalen Wert zu berechnen.
- **Stabilität:** Trotz einiger Ausreißer scheint der Algorithmus insgesamt stabil zu sein, da die gemessenen Zeiten für ähnliche Anzahlen von Gegenständen relativ konsistent sind.

Insgesamt kann festgestellt werden, dass die aktuelle Leistung des Knapsack-Algorithmus für unseren Anwendungsbereich mehr als ausreichend ist, da wir nicht mit einer derart großen Anzahl an Gegenständen arbeiten. Die HoloLens2 ist in der Lage, die optimale Lösung in wenigen Millisekunden und ohne Leistungseinbußen zu liefern. Selbst bei wiederholten Aufrufen spielen ungünstige Muster keine signifikante Rolle.

# Kapitel 5

## Zusammenfassung und Abschluss

### 5.1 Ergebnis

Die Umsetzung des Projekts verlief erfolgreich und alle geplanten Ziele wurden erreicht. Als Ergebnis stehen eine Augmented Reality-Anwendung und eine Website zur Verfügung, die im Unterricht als Lehrmittel eingesetzt werden können und auch am Tag der Offenen Tür genutzt werden können, um diesen interessanter zu gestalten.

### 5.2 Abnahme

Die Abnahme des Projekts erfolgte am 03.04.2024 mit der Abgabe dieser Diplomarbeit. Die HTBLuVA Wiener Neustadt plant, diese Applikation in naher Zukunft im Unterricht sowie am Tag der offenen Tür einzusetzen.

### 5.3 Zukunft

Die Zukunftsperspektive bietet verschiedene Möglichkeiten zur Weiterentwicklung und Erweiterung der vorliegenden Projektarbeit. Ein möglicher Ansatz besteht darin, das vorhandene Wissen und die Technologie an ein nachfolgendes Team weiterzugeben, um zusätzliche Anwendungsszenarien zu entwickeln. Durch die Initialisierung des Projekts auf der HoloLens 2 durch das aktuelle Projektteam steht den Nachfolgern eine solide Grundlage zur Verfügung. Diese bietet bereits eine Vielzahl von behobenen Problemen und dokumentierten Lösungsansätzen. Dadurch können Zeit und Ressourcen gespart werden, und der Fokus kann verstärkt auf die Erweiterung und Verbesserung des bestehenden Systems gelegt werden.

Während der Projektarbeit wurden im Team verschiedene Ideen für potenzielle Anwendungsszenarien diskutiert und konzipiert. Eine vielversprechende Idee wurde identifiziert, die das Spektrum der Anwendungsmöglichkeiten der HoloLens 2 erweitern könnte:

- **PC-Zusammenbau-Simulator:** Dieses interaktive Szenario ermöglicht es dem Benutzer, den detaillierten Zusammenbau eines PCs virtuell durchzuführen. Der Benutzer kann aus einem umfangreichen Katalog verschiedener Bauteile wählen und diese gemäß den Anweisungen in den PC einbauen. Das Szenario führt den Benutzer durch die einzelnen Schritte des Zusammenbaus und erklärt die Funktionen der verschiedenen Komponenten.

## Anhang A

### Mockups

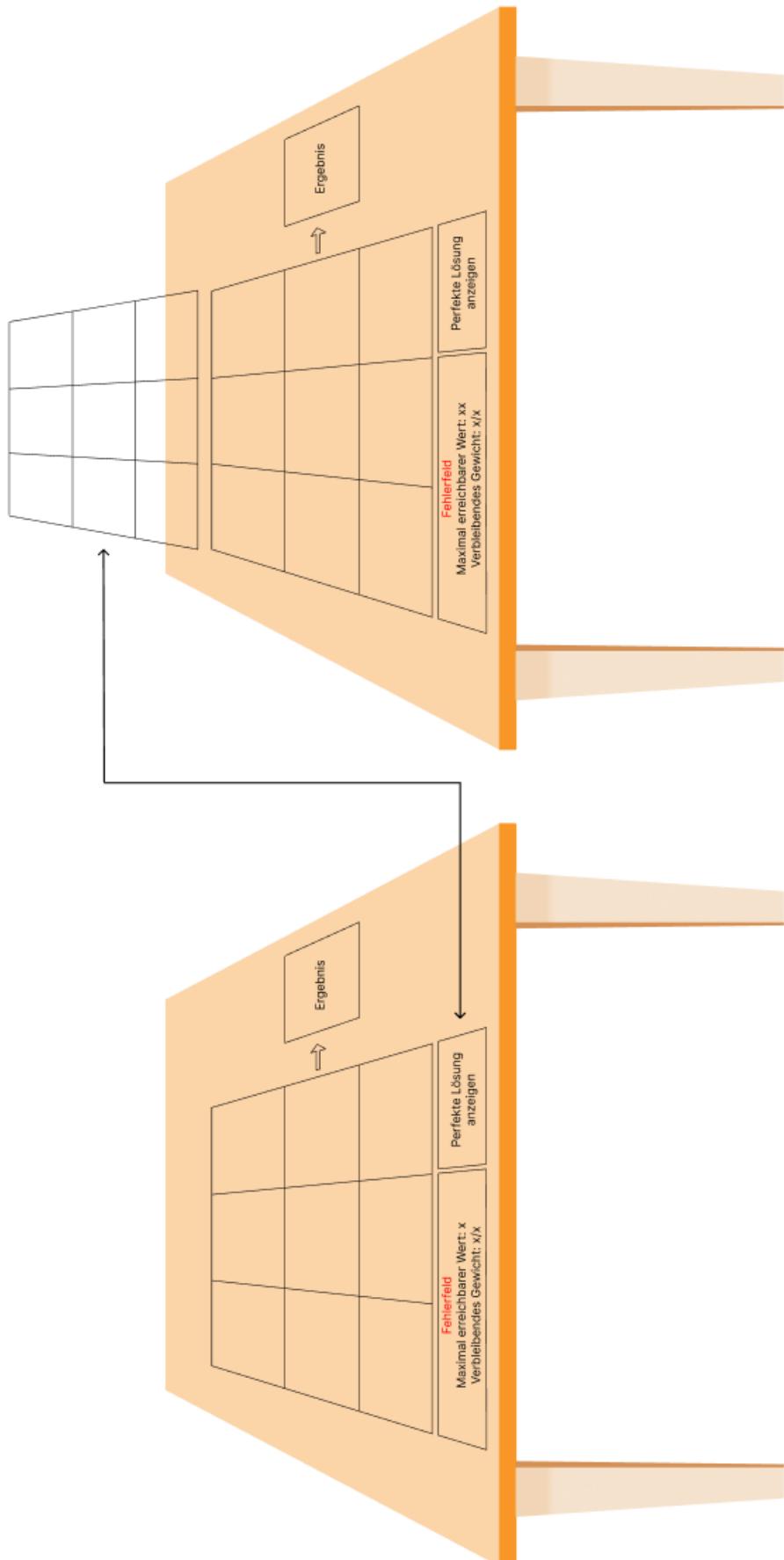
#### A.1 UI/UX

## A.2 Hauptmenu

### A.3 Nachrichtenaustausch Anwendungsszenario



#### A.4 Knapsack-Problem Anwendungsszenario



## Anhang B

# Literatur

- Blender. URL: <https://www.blender.org/about/> (besucht am 06.10.2023).
- Unity Website. URL: <https://unity.com/> (besucht am 13.03.2024).
- Unreal Engine Website. URL: <https://www.unrealengine.com/> (besucht am 13.03.2024).
- Unreal Engine Dokumentation. Blueprint Visual Scripting. URL: <https://docs.unrealengine.com/5.3/en-US/blueprints-visual-scripting-in-unreal-engine/> (besucht am 13.03.2024).
- Scrum.org. What is Scrum? URL: <https://www.scrum.org/learning-series/what-is-scrum/what-is-scrum> (besucht am 06.10.2023).
- Scrum.org. What is a Product Owner? URL: <https://www.scrum.org/learning-series/what-is-scrum/the-scrum-team/what-is-a-product-owner> (besucht am 10.11.2023).
- Scrum.org. What is a Scrum Master? URL: <https://www.scrum.org/learning-series/what-is-scrum/the-scrum-team/what-is-a-scrum-master> (besucht am 10.11.2023).
- Scrum.org. What is a Developer? URL: <https://www.scrum.org/learning-series/what-is-scrum/the-scrum-team/what-is-a-developer> (besucht am 10.11.2023).
- Scrum.org. What is a Sprint? URL: <https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-a-sprint> (besucht am 10.11.2023).
- Scrum.org. What is a Sprint Planing? URL: <https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-sprint-planning> (besucht am 10.11.2023).
- Scrum.org. What is a Daily Scrum? URL: <https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-a-daily-scrum> (besucht am 10.11.2023).
- Scrum.org. What is a Sprint Review? URL: <https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-a-sprint-review> (besucht am 10.11.2023).
- Scrum.org. What is a Sprint Retroperspective? URL: <https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-a-sprint-retrospective> (besucht am 10.11.2023).
- Unity Dokumentation GameObjects. URL: <https://docs.unity3d.com/Manual/GameObjects.html> (besucht am 18.02.2024).
- Microsoft Dokumentation. MIXED REALITY TOOLKIT 3. URL: <https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-overview/> (besucht am 05.11.2023).
- Khronos Group. OPENXR. URL: <https://www.khronos.org/openxr/> (besucht am 05.11.2023).
- Medium. CREATING MANAGER CLASSES IN UNITY. URL: <https://sneakydaggergames.medium.com/creating-manager-classes-in-unity-a77cf7edcba5> (besucht am 05.11.2023).
- Unity Dokumentation. AR Plane Manager. URL: <https://docs.unity.cn/Packages/com.unity.xr.arfoundation@4.1/manual/plane-manager.html> (besucht am 05.11.2023).
- Unity Dokumentation. AR Raycast Manager. URL: <https://docs.unity.cn/Packages/com.unity.xr.arfoundation@5.0/api/UnityEngine.XR.ARFoundation.ARRaycastManager.html>

- (besucht am 05.11.2023).
- Unity Dokumentation. Plane. URL: <https://docs.unity3d.com/ScriptReference/Plane.html> (besucht am 13.12.2023).
  - Unity Dokumentation. Scenes. URL: <https://docs.unity3d.com/Manual/CreatingScenes.html> (besucht am 13.12.2023).
  - Unity Dokumentation. Prefabs. URL: <https://docs.unity3d.com/Manual/Prefabs.html> (besucht am 15.01.2024).
  - Unity Dokumentation. Bounds. URL: <https://docs.unity3d.com/ScriptReference/Bounds.html> (besucht am 16.01.2024).
  - Unity Dokumentation. Renderer. URL: <https://docs.unity3d.com/ScriptReference/Renderer.html> (besucht am 16.01.2024).
  - Color Doku, URL: <https://docs.unity3d.com/ScriptReference/Color.html%7D> besucht am 4.11.2023).
  - Trackable Doku, URL: <https://docs.unity3d.com/2019.2/Documentation/ScriptReference/Experimental.XR.TrackableType.html%7D> (besucht am 18.11.2023).
  - ARRaycastHit Doku, URL: <https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.0/api/UnityEngine.XR.ARFoundation.ARaycastHit.html%7D> (besucht am 18.11.2023).
  - PhotoCaputre, URL: <https://docs.unity3d.com/ScriptReference/Windows.WebCam.PhotoCapture.html%7D> (besucht am 2.11.2023).
  - Unity. TextMeshPro. URL: <https://docs.unity3d.com/Manual/com.unity.textmeshpro.html> (besucht am 15.12.2023).
  - Foto-/Videokamera in Unity, URL: <https://learn.microsoft.com/de-de/windows/mixed-reality/develop/unity/locatable-camera-in-unity%7D> (besucht am 2.11.2023)
  - Microsoft. Buttons — MRTK2. URL: <https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/ux-building-blocks/button?view=mrtkunity-2022-05> (besucht am 7.11.2023).
  - Microsoft. Menü Nahe — MRTK2. URL: <https://learn.microsoft.com/de-de/windows/mixed-reality/mrtk-unity/mrtk2/features/ux-building-blocks/near-menu?view=mrtkunity-2022-05> (besucht am 8.11.2023).
  - Unity. Load scene on button press. URL: [https://blog.insane.engineer/post/unity\\_button\\_load\\_scene/](https://blog.insane.engineer/post/unity_button_load_scene/) (besucht am 8.11.2023).
  - Blender. Can't see added cube on my scene collection [duplicate]. URL: <https://blender.stackexchange.com/questions/162424/cant-see-added-cube-on-my-scene-collection> (besucht am 15.11.2023).
  - Blender. How do I Inset a face equally? URL: <https://blender.stackexchange.com/questions/50876/how-do-i-inset-a-face-equally> (besucht am 17.11.2023).
  - Blender. Modifier. URL: <https://docs.blender.org/manual/en/latest/modeling/modifiers/index.html> (besucht am 10.12.2023).
  - Blender. Object Modes. URL: <https://docs.blender.org/manual/en/latest/editors/3dview/modes.html> (besucht am 21.12.2023).
  - Blender. Loop Tools. URL: [https://docs.blender.org/manual/en/latest/addons/mesh\\_looptools.html](https://docs.blender.org/manual/en/latest/addons/mesh_looptools.html) (besucht am 20.11.2023).
  - Blender. Vertices. URL: [https://docs.blender.org/manual/en/latest/scene\\_layout/object\\_properties/instancing/verts.html](https://docs.blender.org/manual/en/latest/scene_layout/object_properties/instancing/verts.html) (besucht am 05.01.2024).
  - Blender. Meshes. URL: <https://docs.blender.org/manual/en/latest/modeling/meshes/index.html> (besucht am 10.11.23).

- Blender. Array-Modifier. URL: <https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/array.html> (besucht am 11.01.2024)
- Blender. Extrude. URL: [https://docs.blender.org/manual/de/dev/grease\\_pencil/modes/edit/point\\_menu.html#extrude](https://docs.blender.org/manual/de/dev/grease_pencil/modes/edit/point_menu.html#extrude) (besucht am 20.2.2024)
- Autodesk FBX. Getting started. URL: [https://help.autodesk.com/view/FBX/2020/ENU/?guid=FBX\\_Developer\\_Help\\_welcome\\_to\\_the\\_fbx\\_sdk\\_html](https://help.autodesk.com/view/FBX/2020/ENU/?guid=FBX_Developer_Help_welcome_to_the_fbx_sdk_html) (besucht am 07.01.2024).
- Autodesk FBX. URL: <https://www.autodesk.com/products/fbx/overview> (besucht am 07.01.2024).
- Blender. Images as Planes. URL: [https://docs.blender.org/manual/en/latest/addons/import\\_export/images\\_as\\_planes.html](https://docs.blender.org/manual/en/latest/addons/import_export/images_as_planes.html) (besucht am 05.12.2024).
- Mozilla Developer Network. HTML. URL: <https://developer.mozilla.org/en-US/docs/Web/HTML> (besucht am 19.02.2024).
- Mozilla Developer Network. CSS. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS> (besucht am 19.02.2024).
- Mozilla Developer Network. JavaScript. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (besucht am 19.02.2024).
- Bootstrap Dokumentation. URL: <https://getbootstrap.com/docs/5.1/getting-started/introduction/> (besucht am 20.02.2024).
- Bootstrap Dokumentation. CDN-Links. URL: <https://getbootstrap.com/docs/5.2/getting-started/introduction/> (besucht am 20.02.2024).
- Unity. QR-Code Tracking. URL: <https://learn.microsoft.com/en-us/samples/microsoft/mixedreality-qrcode-sample/qr-code-tracking-in-unity/> (besucht am 2.11.2023).
- Unity. QR-Code Tracking Overview. URL: <https://learn.microsoft.com/de-de/windows/mixed-reality/develop/advanced-concepts/qr-code-tracking-overview> (besucht am 30.10.2023).
- Unity. SpacialGraphNode Class. URL: <https://learn.microsoft.com/de-de/dotnet/api/microsoft.mixedreality.openxr.spatialgraphnode?view=mixedreality-openxr-plugin-1.9> (besucht am 2.11.2023).
- Scholl, Armin. *Die Befragung. Sozialwissenschaftliche Methode und kommunikationswissenschaftliche Anwendungen.* 1. Aufl. Konstanz: UVK, 2003.
- Mayer, Horst. *Interview und schriftliche Befragung. Entwicklung, Durchführung und Auswertung.* 3. Aufl. München: R. Oldenbourg, 2006.
- Bühner, Markus. *Einführung in die Test- und Fragebogenkonstruktion.* 1. Aufl. München: Pearson Studium, 2004.
- Claus Brell. *Statistik von Null auf Hundert.* 2. Aufl. Berlin: Springer.
- Unity. GameObjects. URL: <https://docs.unity3d.com/Manual/GameObject.html> (besucht am 15.12.2023).
- Unity. Texture2D. URL: <https://docs.unity3d.com/ScriptReference/Texture2D.html> (besucht am 15.12.2023).
- Unity. Canvas. URL: <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html> (besucht am 21.02.2024)
- Unity. Job system overview. URL: <https://docs.unity3d.com/Manual/JobSystemOverview.html> (besucht am 21.02.2024)
- Unity. PhotoCapture. URL: <https://docs.unity3d.com/Manual/JobSystemOverview.html> (besucht am 22.02.2024)
- Unity. Asset Store. URL: [https://support.unity.com/hc/en-us/articles/210142503-What-is-the-Unity-Asset-Store-and-how-do-I-purchase-Assets#:~:text=The%20Unity%20Asset%](https://support.unity.com/hc/en-us/articles/210142503-What-is-the-Unity-Asset-Store-and-how-do-I-purchase-Assets#:~:text=The%20Unity%20Asset%20Store%20is%20a%20central%20location%20where%20you%20can%20purchase%20and%20download%20assets%20for%20your%20Unity%20project.)

20Store%20is%20home%20to%20a%20growing%20library,examples%2C%20tutorials%20and%20Extension%20Assets (besucht am 28.02.2024).

- Blender. Allgemein. URL: <https://www.blender.org/> (besucht am 27.02.2024).
- Autodesk. 3DS Max. URL: <https://www.autodesk.de/products/3ds-max/overview?term=1-YEAR&tab=subscription> (besucht am 27.02.2024).
- Maxon. Cinema 4d. URL: <https://www.maxon.net/de/cinema-4d> (besucht am 28.02.2024).
- GeeksForGeeks.org Introduction to Knapsack Problem, its Types and How to solve them. URL: <https://www.geeksforgeeks.org/introduction-to-knapsack-problem-its-types-and-how-to-solve-them/> (besucht am 18.02.2024).
- GeeksForGeeks. Fraktionales Knapsack-Problem. URL: <https://www.geeksforgeeks.org/fractional-knapsack-problem/> (besucht am 11.03.2024).
- GeeksForGeeks. 0/1 Knapsack-Problem. URL: <https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/> (besucht am 11.03.2024).
- GeeksForGeeks. Begrenztes Knapsack-Problem. URL: <https://www.geeksforgeeks.org/extended-knapsack-problem/> (besucht am 11.03.2024).
- GeeksForGeeks. Unbegrenztes Knapsack-Problem. URL: <https://www.geeksforgeeks.org/unbounded-knapsack-repetition-items-allowed/> (besucht am 11.03.2024).
- Wikipedia. Mehrzieliges Knapsack-Problem. URL: [https://en.wikipedia.org/wiki/Knapsack\\_problem#Multi-objective\\_knapsack\\_problem](https://en.wikipedia.org/wiki/Knapsack_problem#Multi-objective_knapsack_problem) (besucht am 11.03.2024).
- Wikipedia. Multidimensionales Knapsack-Problem. URL: [https://en.wikipedia.org/wiki/Knapsack\\_problem#Multi-dimensional\\_knapsack\\_problem](https://en.wikipedia.org/wiki/Knapsack_problem#Multi-dimensional_knapsack_problem) (besucht am 11.03.2024).
- Wikipedia. Mehrere Knapsack-Probleme URL: [https://en.wikipedia.org/wiki/Knapsack\\_problem#Multiple\\_knapsack\\_problem](https://en.wikipedia.org/wiki/Knapsack_problem#Multiple_knapsack_problem) (besucht am 11.03.2024).
- Wikipedia. Quadratisches Knapsack-Problem. URL: [https://en.wikipedia.org/wiki/Knapsack\\_problem#Quadratic\\_knapsack\\_problem](https://en.wikipedia.org/wiki/Knapsack_problem#Quadratic_knapsack_problem) (besucht am 11.03.2024).
- Wikipedia. Geometrisches Knapsack-Problem. URL: [https://en.wikipedia.org/wiki/Knapsack\\_problem#Geometric\\_knapsack\\_problem](https://en.wikipedia.org/wiki/Knapsack_problem#Geometric_knapsack_problem) (besucht am 11.03.2024).
- Hindawi - Journals. Solving the 0/1 Knapsack Problem Using Metaheuristic and Neural Networks for the Virtual Machine Placement Process in Cloud Computing Environment. URL: <https://www.hindawi.com/journals/mpe/2023/1742922/> (besucht am 11.03.2024).
- Wikipedia - Knapsack-Problem Anwendungen. URL: [https://en.wikipedia.org/wiki/Knapsack\\_problem#Applications](https://en.wikipedia.org/wiki/Knapsack_problem#Applications) (besucht am 11.03.2024).
- VivifyScrum - Website. URL: <https://app.vivifyscrum.com> (besucht am 12.03.2024).
- Trello - Website. URL: <https://Trello.com> (besucht am 12.03.2024).
- Unity. Raycast. URL: <https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.0/manual/raycast-manager.html> (besucht am 10.03.2024)
-