
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 17. Februar 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 17. Februar 2024

Verfasser / Verfasserinnen:

Moritz SKREPEK

Dustin LAMPEL

Seref HAYLAZ

Jonas SCHODITSCH

Inhaltsverzeichnis

Eidesstattliche Erklärung	i
Vorwort	v
Diplomarbeit Dokumentation	vi
Diploma Thesis Documentation	viii
Kurzfassung	x
Abstract	xi
1 Einleitung	1
1.1 Ausgangslage	1
1.2 Auslöser	1
1.3 Aufgabenstellung	1
1.4 Team	2
1.4.1 Aufteilung	2
2 Grundlagen	3
2.1 Vorgehensmodelle	3
2.2 Scrum	3
2.2.1 Die drei Rollen in Scrum	3
2.2.2 Begründung der Auswahl	5
2.3 Projektmanagement-Tools	5
2.3.1 GitHub	5
2.3.2 Jira	6
2.4 Konzeption von Fragebögen	6
2.4.1 Planung der Fragebogenkonstruktion	6
2.4.2 Formulierung der Fragen	8
2.4.3 Arten von Fragen	9
2.4.4 Struktur und Gliederung von Fragebögen	10
2.4.5 Mögliche Verfälschung des Resultats	10
2.4.6 Auswertung von Fragebögen	11
3 Produktspezifikationen	12
3.1 Anforderungen und Spezifikationen	12
3.1.1 Use Cases	12
3.2 Design	12
3.2.1 Abläufe	14

3.2.2	Mockups	15
3.3	Eingesetzte Technologien	15
3.3.1	Kriterien	15
3.3.2	Game Engine: Konzeption und Funktion	15
3.3.2.1	Game Engine Auswahl und Wechsel im Projektverlauf	15
3.3.3	Unity foundation packages	17
3.3.3.1	MRTK3	17
3.3.3.2	Microsoft OpenXR Plugin	17
3.3.4	Modellierungsprogramm	18
3.3.4.1	Wie funktioniert Blender im Allgemeinen?	18
4	Feinkonzept und Realisierung	19
4.1	Entwicklungsumgebungen	19
4.1.1	Visual Studio 2022	19
4.1.2	Unity	19
4.1.2.1	Multidisziplinäre Unterstützung und Integration	19
4.1.2.2	Szenengestaltung und Asset-Management	19
4.1.2.3	Programmierung und Skripterstellung	20
4.1.2.4	Unterstützung für Augmented Reality (AR) und Virtual Reality (VR)	20
4.1.2.5	Erweiterte Debugging- und Profiling-Werkzeuge	20
4.1.3	Aufbau einer Unity-Applikation	20
4.1.4	Lebenszyklusmethoden in Unity	20
4.1.5	Manager in Unity	21
4.2	Objektdesign mittels Blender	22
4.2.1	Rendering und Optimierung für AR	22
4.2.2	Export- und Integrationsprozess	23
4.2.3	Blender-Add-Ons und Plugins	24
4.2.3.1	Looptools: Optimierung von Topologie und Oberflächen	24
4.2.3.2	Images as Planes: Effiziente Integration von Texturen	24
4.2.4	Blender - Modes	24
4.2.5	Blender - Hierarchie	26
4.2.5.1	Blender - Modifier	27
4.3	Menü	29
4.3.1	Erstentwurf	29
4.3.1.1	Probleme beim Erstentwurf	30
4.3.2	Finalversion	31
4.3.3	Laden der Level	33
4.3.4	Setzen des Menüs	34
4.3.5	UI/UX	35
4.4	Ping Level	36
4.4.1	Object Tracking	36
4.4.2	Kurvenberechnung	36
4.5	Knapsack Problem Level	36
4.5.1	Knapsack-Problem Level Hirarchie	36
4.5.2	Unity Prefabs	38
4.5.3	Nutzung von QR-Codes	39
4.5.3.1	Struktur und Inhalt eines QR-Codes	39
4.5.3.2	QR-Code-Tracking	40

4.5.3.3	Interaktion mit QR-Codes	41
4.5.3.4	Bestimmen der Position QR-Codes	41
4.5.3.5	Visualisierung von QR-Codes	41
4.5.3.6	Zugriff auf QR-Codes bereitstellen	42
4.5.4	Inventory Placement Controller Game Objekt	42
4.5.4.1	InventoryPlacementController Klassenvariablen	43
4.5.4.2	Startverzögerung	44
4.5.4.3	Frame-Aktualisierung zur Identifikation des gewünschten AR-Planes	45
4.5.5	Inventory Controller Game Objekt	49
4.5.5.1	InventoryController Klassenvariablen	50
4.5.5.2	Start des InventoryControllers	51
4.5.5.3	Bounds des Inventars aktualisieren	51
4.5.5.4	Bounds des Inventars ermitteln	52
4.5.5.5	Inventory Bounds erweitern	52
4.5.5.6	ID Grid Initialisierung	53
4.5.5.7	Update-Funktion	54
4.5.5.8	Neue oder entfernte Items erkennen	54
4.5.6	EventManager	61
4.5.7	Knapsack Solver Game Objekt	62
4.5.7.1	KnapsackSolver Klassenvariablen	63
4.5.7.2	Start des KnapsackSolvers	64
4.5.7.3	Starten der Berechnung	64
4.5.7.4	Knapsack-Algorithmus	66
4.5.7.5	Berechnung des eigenen Inventars	72
4.5.7.6	InfoMesh aktualisieren	73
4.5.8	Best Solution Prefab Game Objekt	73
4.5.8.1	Script Aufruf	75
4.5.8.2	PerfectSolutionVisualizer Klassenvariablen	76
4.5.8.3	Start des PerfectSolutionVisualizer	76
4.5.8.4	Setzen der neuen Position der perfekten Lösung	77
4.5.8.5	Perfekte Lösung füllen	77
4.5.9	Unit-Tests	79
4.6	Performance und Qualitätssicherung	79
4.6.1	Unit-Tests	79
4.6.2	Unity Test Framework	79
4.6.3	Performance-Messung	79
5	Zusammenfassung und Abschluss	83
5.1	Ergebnis	83
5.2	Abnahme	83
5.3	Zukunft	83
A	Mockups	84
A.1	UI/UX	84
A.2	Hauptmenu Level Design	84
A.3	Ping-Paket Level Design	84
A.4	Knapsack-Level Design	84
B	Literatur	85

Vorwort

Die vorliegende Diplomarbeit wurde im Zuge der Reife- und Diplomsprüfung im Schuljahr 2023 / 24 an der Höheren Technischen Bundeslehr- und Versuchsanstalt Wiener Neustadt verfasst. Die Grundlegende zu dem arbeiten mit der Microsoft HoloLens2 lieferte uns unser Betreuer Mag. BEd. Markus Reis. Das Ergebniss dieser Diplomarbeit ist eine augmented reality Applikation für die Verwendung 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	Darstellung von zwei ausgewählten IT-Grundprinzipien mittels der Microsoft HoloLens2.
------------------	---

Realisierung	Implementiert wurde eine 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 eines Pings, Veranschaulichung Knapsack-Problem) für den Einsatz im Unterricht sowie beim Tag der offenen Tür visuell, interaktiv und spielerisch darzustellen.
------------	---

<p>Typische Grafik, Foto etc. (mit Erläuterung)</p>	<p>Das vorliegende Bild stellt das Logo der AR-Applikation dar.</p>  <p>Applied Augmented Reality IN EDUCATION</p>
---	--

<p>Teilnahme an Wettbewerben, Auszeichnungen</p>	
--	--

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

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

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 a functional augmented reality application based on the AR device HoloLens2 from Microsoft, which enables selected technical topics in the field of computer science (visualization of a ping, illustration of the Backpack problem) for use in lessons and on the day of open door visually, interactively and playfully.
---------	---

Illustrative graph, photo
(incl. explanation)

This image represents the logo of the AR application.



Applied Augmented Reality

IN EDUCATION

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 Abschlussarbeit 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 Levels. Darunter, dass Hauptmenu, das Ping-Level und das Knapsack-Problem-level, welche in Unity implementiert wurden.

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. Der Einsatz von Unity als Entwicklungsplattform ermöglichte eine umfassende und wissenschaftlich fundierte Umsetzung dieses Projekts.

Abstract

This diploma thesis focuses on the development of an educational application for HTL Wiener Neustadt using the Unity platform. The implementation took the form of an augmented reality (AR) application specifically designed for the Microsoft HoloLens 2.

The application comprises three distinct levels, namely the main menu, the Ping Level, and the Knapsack-Problem Level, all implemented using Unity.

During the open house event, the application enables students to explore two fundamental principles of computer science in a playful and engaging manner through augmented reality. This provides students with the opportunity to discover whether they have an interest in such topics. The utilization of Unity as the development platform facilitated a comprehensive and scientifically grounded realization of this project.

Kapitel 1

Einleitung

1.1 Ausgangslage

Um dem IT-Fachkräftemangel entgegenzuwirken, muss die Ausbildung im MINT-Bereich attraktiviert werden. Diese Diplomarbeit will hier, unterstützt durch das Förderprogramm "Wissenschaft trifft Schule" des Landes NÖ, einen wichtigen Beitrag leisten. Dazu 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" bekommen mit dieser Applikation die neusten Technologien vorgeführt und erkennen dadurch, dass die Schule sich auf einen sehr hohen Technologiestandard befindet. Dadurch kommt es zu einer deutlich erhöhten Nachfrage bei zukünftigen Bewerbungen für die Abteilung Informatitionstechnik. Weiters wird nach Außen hin der Ruf der Schule gestärkt und diese präsentiert sich damit als attraktiver Ausbildungsstandort für die zukünftigen Mitarbeiter vieler Unternehmen.

1.3 Aufgabenstellung

Erstellen des Levelinhals mit der Verwendung von 2 realen Laptops. Mit Hilfe der HoloLens wird ein 3D modelliertes Ping Paket auf dem Kabel, dass die zwei Laptops verbindet dargestellt. Wenn der Benutzer auf der Tastatur auf die "ENTER" Taste drückt, wird ein Ping Befehl ausgeführt und die modellierten Pakete werden durch die HoloLens auf dem Netzwerkabel dargestellt. Dies veranschaulicht dem Benutzer den eigentlich nicht sichtbaren Ping von einem auf den anderen Laptop

Erstellen des zweiten Levels in dem der Benutzer das bekannte Rucksack oder auch Knapsack Problem lösen soll. Durch die HoloLens wird auf einem Tisch ein Spielartiges 2D-Inventar mit einer fix definierten Größe visuell dargestellt. Verwendet werden dabei typisch reale Gegenstände eines HTL Schülers die im Täglichen Gebrauch sind. Z.B.: Laptop, Maus, Tastatur, Block, usw... Bei jedem Item können, wenn es in die Hand genommen wird über einen QR-Code der auf diesem Item befestigt ist, alle möglichen Information des Items angezeigt werden. Die Aufgabe des Benutzer ist es mit den gegebene Items das Inventar best möglich zu befüllen und dadurch den best möglichen Wert pro Volumensprozent zu erreichen. Auf dem Tisch liegen verteilt viele Items, die aber nicht alle in das Inventar passen. Jedes einzelne Item kann der Benutzer aufheben und beliebig in das Inventar le-

gen. Bei jedem neudazugelegtem Item, wird per Knopfdruck auf den SSolve-Button"der aktuelle Inventarwert berechnet und angezeigt. Am Ende kann sich der User auch noch über einen Menupunkt entscheiden, ob er die perfekte Lösung sehen will oder nicht. Wenn sich der User dazu entschieden die perfekte Lösung anzuzeigen, wird Vertikal über dem vormalen Inventar noch ein Inventar projiziert, dass das normale Inventar wiederspiegelt aber mit 3D-Modellierten Objekten.

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 Grundidee für die Darstellung zweier IT-Grundprinzipien mittels der Microsoft HoloLens2 hatte. Das Entwickelte System lässt sich in das Hauptmenu, das Ping-Paket-Level und das Knapsack- Problem-Level gliedern. Die Implementierung des Hauptmenus übernahm Dustin LAMPEL, dabei verwendete er für die UI/UX das UX-Tools-Plug-Ins für Mixed Reality. Die Umsetzung des Pink-Paket-Levels übernahm SCHODITSCH Jonas mittels Objekt- Tracking und Picture-Taking. Für die Implementierung des Knapsack-Problem-Levels waren SKREPEK Moritz und HAYLAZ Seref mittels Verwendung von Plane-detection, QR- Code Tagging und Tracking, Knapsack Algorithmus, 3D Unity Game Objekte und Unit- tests.

Kapitel 2

Grundlagen

In diesem Kapitel werden das Vorgehensmodell und alle Tools, die für die erfolgreiche Abwicklung des Projekts nötig sind, erläutert.

2.1 Vorgehensmodelle

Im Vorfeld der Durchführung des Projekts wurden Informationen über diverse Vorgehensmodelle gesammelt. Für das Projektteam war schnell klar, dass ein agiles Modell gewählt werden sollte, da somit das Projekt dynamischer geplant und durchgeführt werden kann. Die Auswahl für Scrum stand direkt bei Projektbeginn fest. In dem folgenden Abschnitt wird dieses Vorgehensmodell genauer erklärt und unsere Entscheidung anschließend begründet.

2.2 Scrum

Scrum¹ repräsentiert ein agiles Projektmanagement-Framework, das auf die effiziente Entwicklung von Produkten und Software abzielt. Es legt besonderen Wert auf Zusammenarbeit, Anpassungsfähigkeit und die kontinuierliche Bereitstellung funktionsfähiger Inkremente innerhalb kurzer Entwicklungszyklen, den sogenannten Sprints.

Die zuvor skizzierte Definition gewährt einen knappen Einblick in das agile Vorgehensmodell Scrum. Die herausragenden Merkmale dieses Modells sind:

- Drei zentrale Rollen, die im Folgenden näher erläutert werden.
- Der Product Backlog, der sämtliche Anforderungen enthält.
- Eine iterative und zeitlich definierte Entwicklung von Produkten.
- Die autonome Arbeitsweise des Teams.
- Gleichberechtigung aller Teammitglieder.

2.2.1 Die drei Rollen in Scrum

- **Product Owner²:** Der Product Owner trägt die Verantwortung für die Pflege des Product Backlogs und vertritt dabei die fachliche Auftraggeberseite sowie sämtliche Stakeholder. Ein zentrales Anliegen ist die Priorisierung der Elemente im Product

¹Quelle: Scrum Alliance **WHAT-IS-SCRUM**

²Scrum-Rolle **Product-Owner**

Backlog, um den geschäftlichen Wert des Produkts zu maximieren und die Möglichkeit für frühe Veröffentlichungen essenzieller Funktionalitäten zu schaffen. Der Product Owner nimmt nach Möglichkeit an den täglichen Scrum-Meetings teil, um auf passive Weise Einblicke zu gewinnen. Zudem steht er dem Team für Rückfragen zur Verfügung, um einen reibungslosen Informationsaustausch zu gewährleisten.

- **Scrum Master³:** Der Scrum Master übernimmt eine zentrale Rolle im Scrum-Prozess und ist für die korrekte Umsetzung desselben verantwortlich. Als Vermittler und Unterstützer fungiert er als Facilitator, der darauf abzielt, einen maximalen Nutzen 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 liegt in seiner Verantwortung, das Team vor unberechtigten Eingriffen während des Sprints zu schützen.
- **Team⁴:** Das Team, bestehend aus vier bis zehn Mitgliedern, idealerweise sieben, zeichnet sich durch eine interdisziplinäre Zusammensetzung aus, die Entwickler, Architekten, Tester und technische Redakteure einschließt. Durch Selbstorganisation agiert das Team eigenständig und übernimmt die Verantwortung als sein eigener Manager. Es besitzt die Befugnis, autonom über die Aufteilung von Anforderungen in Aufgaben zu entscheiden und diese auf die einzelnen Mitglieder zu verteilen, wodurch der Sprint Backlog aus dem aktuellen Teil des Product Backlog entsteht.

Alle Anforderungen an das Produkt werden in sogenannten User Stories, vorrangig erstellt durch den Product Owner, im Product Backlog gesammelt. In einem Intervall, bezeichnet als Sprint, werden die User Stories abgearbeitet. Die Projektentwicklung nach Scrum besteht aus fünf zentralen Elementen:

- **Sprint: Planning Meeting⁵:** Im Sprint Planning Meeting wird das Ziel des folgenden Sprints definiert. Hierbei werden die Anforderungen im Project Backlog, die in diesem Sprint umgesetzt werden sollen, in einzelne Aufgaben zerlegt und anschließend im Sprint Backlog gesammelt.
- **Sprint⁶:** Ein Sprint repräsentiert eine Entwicklungsphase, während der eine voll funktionsfähige und potenziell veröffentlichte Software entsteht. Die Dauer eines solchen Sprints beträgt typischerweise zwischen 1 und 4 Wochen.
- **Daily Scrum⁷:** Der Daily Scrum ist ein kurzes Teammeeting, in dem Teammitglieder darüber informieren, welche Aufgaben seit dem letzten Meeting abgeschlossen wurden, woran bis zum nächsten Meeting gearbeitet werden muss und wo momentane Probleme existieren. Auf diese Weise sind alle Teammitglieder stets auf dem aktuellen Stand, was die Lösung aufkommender Probleme erleichtert.
- **Sprint Review⁸:** In diesem Meeting präsentiert das Entwicklungsteam die im Sprint abgeschlossenen Arbeitsergebnisse, beispielsweise fertige Produktinkremente, den Stakeholdern, zu denen Produktbesitzer, Kunden, Führungskräfte und andere Interessengruppen gehören.

³Scrum-Rolle Scrum-Master

⁴Scrum-Rolle Team

⁵Scrum-Meetings Sprint-planing-meeting

⁶Scrum-Meetings Sprint

⁷Scrum-Meetings Daily-Scrum

⁸Scrum-Meetings Sprint-Review

- **Sprint Retrospective⁹**: Die Sprint Retrospective dient primär dazu, dass das Scrum-Team (bestehend aus dem Entwicklungsteam, dem Scrum Master und dem Product Owner) gemeinsam den abgeschlossenen Sprint reflektiert und Möglichkeiten zur kontinuierlichen Verbesserung identifiziert.

Durch diese Elemente kann ein optimaler Projektlauf gewährleistet werden. Das Projekt bleibt jederzeit offen für Änderungen, und durch eine enge Zusammenarbeit mit dem Kunden können Missverständnisse und Probleme frühzeitig behandelt und kommuniziert werden.

2.2.2 Begründung der Auswahl

Die Applied Augmented Reality in Education Applikation besteht aus 3 verschiedenen Level. Im Team welches aus vier Schülern bestand übernahm jede Person einen Teilbereich oder arbeiteten gemeinsam an einem dieser Level mit Unteraufgaben in diesem Level. Unterstützt wurde man von einem Lehrer, der stets für Fragen bereitstand und oftmals in beratender Form vorhanden war. Als Vorgehensmodell wählte das Team das agile Modell Scrum. Die von Scrum gegebenen Richtlinien konnten leicht eingehalten werden, da das Team täglich in der Schule aufeinander traf als auch privat Kontakt hatten. Jederart Änderung, Problem oder Änderungen und anderartige Dinge konnten daher leicht kommuniziert und besprochen werden. Am Ende jedes Sprints wurden die erreichten Ergebnisse mit dem Betreuer besprochen, sowie die Neuerungen vorgestellt. In den Sprintreviews konnte somit Feedback zu den Ergebnissen gesammelt werden und von dem Betreuer konnten neue Ansichten und Denkweisen angebracht und integriert werden. Durch die Sprint Retroperspektive konnten die Schüler einen größeren Mehrwert aus der Projektentwicklung schöpfen, da sie neben der Verwendung des Scrum-Prozesses auch ihre Fähigkeiten in den einzelnen Bereichen, durch das Besprechen der positiven und negativen Aspekte verbessern.

2.3 Projektmanagement-Tools

Um einen positiven Verlauf des Projekts zu ermöglichen, benötigt man die unterstützenden Tools beim Projektmanagement sowie die Verwaltung von Dateien.

2.3.1 GitHub

Als sogenanntes Repository für die Source Code Dateien wurde GitHub mit der dazugehörigen Webanwendung verwendet. Zu Beginn des Projekts stand die Entscheidung an, welche Technologie und welcher Anbieter für das Versionskontrollsysteem gewählt werden sollten. Neben GitHub gibt es andere namhafte Anbieter solcher Verwaltungssysteme, darunter GitLab und SourceForge.

Ausschlaggebend für die Wahl von GitHub waren mehrere Punkte. Zum einen ist GitHub eine kostenlose Lösung, die es ermöglicht, ein privates Projekt mit mehreren Mitgliedern ohne Kosten anzulegen. Im Gegensatz dazu bieten manche Plattformen nur eine begrenzte Anzahl von Mitgliedschaften in kostenfreien Projekten an. Die Registrierung erforderte lediglich einen Account.

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

⁹Scrum-Meetings Sprint-Retroperspektiv

2.3.2 Jira

Als sogenanntes Verwaltungstool für die Vorgänge im Projekt wurde Jira mit der dazugehörigen Webanwendung verwendet. Auch hier stand zu Projektbeginn die Frage im Raum, welche Technologie und welcher Anbieter für das Aufgabenmanagement gewählt werden sollten. Neben Jira gibt es weitere namhafte Anbieter solcher Tools, darunter VivifyScrum und KanBan.

Die Wahl von Jira basierte auf mehreren Überlegungen. Zum einen ist Jira eine kostenlose 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.

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.

2.4 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.

2.4.1 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 zu verwenden 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 Erhebungsbogens 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.

Um das Verständnis für die Gestaltung der Fragebögen und für die Formulierung der Fragen zu erleichtern, wird in diesem Abschnitt ein *Fallbeispiel* vorgestellt, auf das bei der weiteren Erläuterung zurückgegriffen wird.

Während des Tages der offenen Tür im Schuljahr 2023/2024 (25 Personen) und in einer Abschlussklasse (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 Konzepte verstanden wurden und ob potenzielle Verbesserungsmöglichkeiten vorliegen. Eine quantitative Analyse hat ergeben, dass etwa ein Drittel der Teilnehmer des Tages der offenen Tür die Konzeptionen verstanden hat und keine spezifischen Verbesserungsvorschläge geäußert hat.

Anschließend wurden vier 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. Die Ergebnisse zeigen, dass die vermittelten Konzepte verstanden wurden. Allerdings wurden am Tag der offenen Tür im Vergleich zu den befragten Personen viele Verbesserungsvorschläge geäußert. Die Befragten gaben an, dass diese Vorschläge aufgrund des frühen Entwicklungsstadiums der Anwendung entstanden sind und einige essenzielle Aspekte noch nicht oder nur teilweise implementiert waren, wodurch sie unklar blieben.

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.¹⁰

Abbildung 2.1: Zusammenhang zwischen der *Grundgesamtheit* und *Stichprobe*

Anhand des Fallbeispiels wird verdeutlicht, dass es keinen Sinn macht, die Umfrage an sehr jungen Leuten durchzuführen, da diese Personen zum größten Teil noch nicht wirklich tiefer in die Informatik geblickt haben und dadurch keinerlei Verbesserungsvorschläge geben können.

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.¹¹

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. Dies wird anhand eines Fallbeispiels veranschaulicht.

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.¹²

Die drei Gütekriterien stehen in einem wechselseitigen Zusammenhang. Nur, wenn die Objektivität gegeben ist, kann die Reliabilität gewährleistet werden. Ist die Reliabilität gering, kann die Validität nur mit einer gewissen Unsicherheit vorhergesagt werden.¹³

2.4.2 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.

¹⁰Vgl. Mayer, **Interview und schriftliche Befragung**, S. 57 ff.

¹¹Vgl. Mayer, **Interview und schriftliche Befragung**, S. 36

¹²Vgl. Mayer, **Interview und schriftliche Befragung**, S. 54 ff., S. 88.

¹³Vgl. Bühner, **Einführung in die Test- und Fragebogenkonstruktion**, S. 33 f.

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.
- Die Fragen prägnant formulieren.
- Belastende Begriffe wie *Ehrlichkeit* werden vermieden.
- Hypothetische Formulierungen sollten ausgeschlossen werden.
- Fokussierung auf ein bestimmtes Thema für jede einzelne Frage.
- Vermeidung von Überforderung durch die Bereitstellung einer angemessenen Menge an Informationen pro Frage.
- Doppelte Verneinungen sind zu vermeiden.¹⁴

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.

2.4.3 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 das Fehlen einer 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.¹⁵

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

¹⁴Vgl. Mayer, **Interview und schriftliche Befragung**, S. 89.

¹⁵Vgl. Scholl, **Die Befragung**, S. 157.

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.

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.¹⁶

2.4.4 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 der Strukturierung von Fragebögen. *Geschlossene Fragen* mit vorgegebenen Antwortmöglichkeiten ermöglichen eine effiziente *Quantifizierung*, während *offene Fragen* vertiefende qualitative Einblicke liefern können. Die geschickte Kombination beider Typen ermöglicht eine umfassende Datenerhebung.

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 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.

2.4.5 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

¹⁶Vgl. Scholl, **Die Befragung**, S. 164 ff.

in Frage gestellt wird.¹⁷

Eine Vielzahl von Faktoren kann Verfälschungen auslösen. Gesellschaftliche Normen üben oft Druck auf den Einzelnen aus, sich selbst in einem positiven Licht darzustellen, 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.¹⁸

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.

2.4.6 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.

¹⁷Vgl. Bühner, **Einfuehrung in die TEst und Fragebogenkonstruktion**, S. 56.

¹⁸Vgl. Bühner, **Einfuehrung in die TEst und Fragebogenkonstruktion**, S. 59.

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 steht der allgemeine Text für die Anforderungen und Spezifikationen

3.1.1 Use Cases

Hier steht der allgemeine Text für die Use Cases

3.2 Design

Hier steht der allgemeine Text für das Design

3.2.1 Abläufe

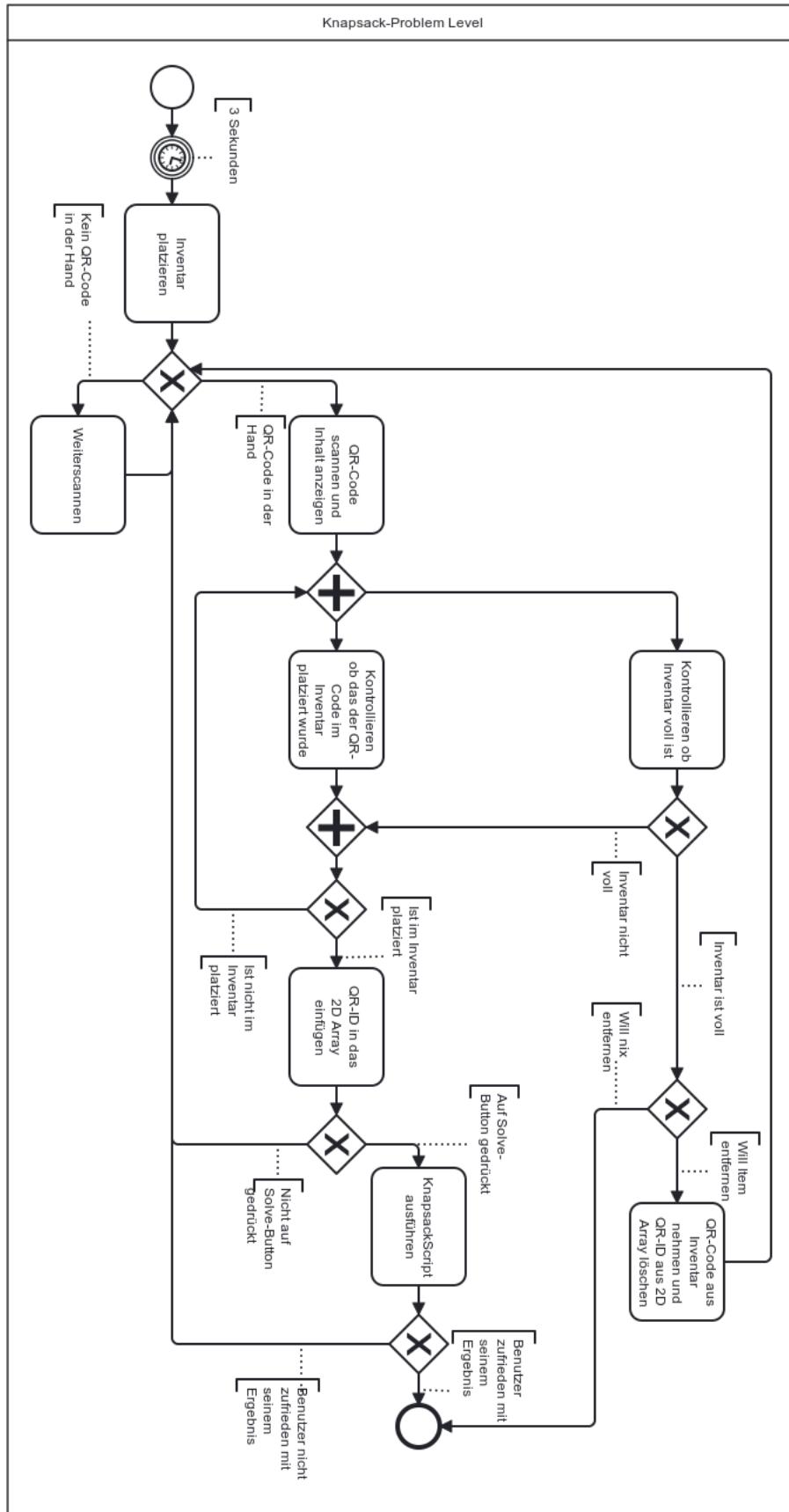


Abbildung 3.1: Ablaufdiagramm des Knapsack-Problem Levels

3.2.2 Mockups

Hier steht der allgemeine Text für die Mockups

3.3 Eingesetzte Technologien

3.3.1 Kriterien

Bei der Auswahl der eingesetzten Technologien war es besonders wichtig, dass diese möglichst zuverlässig und bereits etabliert sind. Die Technologien sollen ausfallsicher, leicht benutzbar und vor allem eine performant Verwendung der Applikation sicherstellen.

3.3.2 Game Engine: Konzeption und Funktion

Eine Game Engine stellt eine hochentwickelte und modulare Entwicklungsumgebung dar, die speziell für die Konzeption, Gestaltung und Implementierung von interaktiven digitalen Spielen entwickelt wurde. Als komplexe Softwarearchitektur bildet sie das Grundgerüst für die Realisierung von Spieleprojekten, wobei sie eine Vielzahl von Funktionen und Werkzeugen bereitstellt, um den Entwicklungsprozess zu erleichtern und zu optimieren.

Im Kern vereint eine Game Engine verschiedene Module, die für unterschiedliche Aspekte der Spieleentwicklung zuständig sind. Dazu gehören unter anderem die Grafik-Engine, die Physik-Engine, die Audio-Engine sowie Mechanismen für Kollisionserkennung, Animationen und Künstliche Intelligenz. Durch diese modulare Struktur ermöglicht die Game Engine eine effiziente und ressourcenschonende Entwicklung, indem sie Entwickler von der tiefen Implementierung grundlegender Funktionen entlastet.

Die Grafik-Engine ist dabei verantwortlich für die Darstellung visueller Elemente, von 2D-Grafiken bis hin zu komplexen 3D-Welten. Sie bietet Mechanismen zur Berechnung von Lichteffekten, Schatten, Texturen und Animationen. Die Physik-Engine simuliert realistische physikalische Interaktionen, um eine authentische Umgebungsgestaltung und realitätsnahe Bewegungen der Spielemente zu gewährleisten. Die Audio-Engine ermöglicht die Integration von Klängen und Musik, um die Spielerfahrung zu vertiefen.

Ein entscheidendes Merkmal einer Game Engine ist auch die Integration von Programmiersprachen wie C++ oder C, die es Entwicklern erlauben, spezifische Spiellogik und Interaktionen zu implementieren. Dieser Aspekt ermöglicht die Flexibilität und Anpassbarkeit der Engine an die individuellen Anforderungen eines Spieleprojekts.

In der Gesamtheit fungiert die Game Engine als zentrale Schaltstelle für die kreative Entfaltung von Entwicklerteams, indem sie eine umfassende Plattform für das Design und die Umsetzung von Spieleideen bietet. Ihre Funktionen reichen von der effizienten Ressourcenverwaltung bis hin zur Bereitstellung von Werkzeugen für das Testen, Debuggen und Optimieren von Spielen.

Die Auswahl einer geeigneten Game Engine ist eine strategische Entscheidung und hängt von den spezifischen Anforderungen eines Projekts ab. In diesem Kontext wurden die beiden führenden Game Engines, Unity und Unreal Engine, evaluiert, wobei Unity aufgrund seiner Programmiersprache C, dem einfachen Einstieg für Anfänger, der exzellenten Dokumentation und der umfangreichen Tutorial-Ressourcen als präferierte Wahl für das vorliegende Projektteam hervorging.

3.3.2.1 Game Engine Auswahl und Wechsel im Projektverlauf

Zu Projektbeginn standen zwei der führenden Game Engines zur Auswahl – die Unreal Engine und Unity. Eine Game Engine fungiert als komplexe Softwareumgebung, speziell

entwickelt für das Design und die Entwicklung von digitalen Spielen. Die Auswahl einer geeigneten Engine beeinflusst maßgeblich den Entwicklungsprozess und den Erfolg eines Projekts.

Nach einer gründlichen Recherche und Evaluierung der beiden Optionen entschied sich das Projektteam für Unity als präferierte Game Engine. Diese Entscheidung wurde durch mehrere Schlüsselfaktoren gestützt:

- **Programmiersprache: C** - Die Verwendung der Programmiersprache C erwies sich als entscheidend, da sie sich als äußerst effizient und benutzerfreundlich herausstellte.
- **Einfacher Einstieg für Anfänger** - Unity bietet einen leicht verständlichen Einstieg in die Spieleentwicklung, was besonders für Teammitglieder mit unterschiedlichem Erfahrungsniveau von Vorteil ist.
- **Sehr gute Dokumentation** - Die umfassende Dokumentation von Unity spielte eine zentrale Rolle für effiziente Entwicklung und Problemlösung im gesamten Projektverlauf.
- **Hohe Anzahl an Tutorials** - Unity überzeugte mit einer reichhaltigen Sammlung von Tutorials und Schulungsmaterialien, die eine kontinuierliche Weiterbildung und schnelle Lösung von Herausforderungen ermöglichten.

Ursprünglich war die Unreal Engine aufgrund ihres beliebten Blueprint-Scripting-Systems in Betracht gezogen worden. Jedoch traten im Verlauf der Entwicklungsarbeit spezifische Herausforderungen auf, die zu einer strategischen Entscheidung für den Wechsel zur Unity Game Engine führten. Die Herausforderungen umfassten:

1. **Mangelhafte Dokumentation für AR-Entwicklung in der Unreal Engine:** Die unzureichende Dokumentation für die Entwicklung von Augmented Reality (AR)-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 anspruchsvoller 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. **Mangelhafte Integration von AR-spezifischen Werkzeugen:** Schwächen in der Integration von AR-spezifischen Entwicklungswerkzeugen in der Unreal Engine erschweren eine nahtlose Interaktion mit AR-Plattformen und die optimale Nutzung ihrer Funktionen.
5. **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 der Unreal Engine zu Unity zu wechseln. Der Wechsel ermöglichte ei-

ne effizientere und zielführende Entwicklung der AR-Applikation, gestützt durch Unity's umfassende Unterstützung, detaillierte Dokumentation und breite Community-Ressourcen.

3.3.3 Unity foundation packages

In dem folgenden Abschnitt wird erklärt welche Packages in die Unity Applikation eingeführt werden müssen um die Entwicklung einer Augmented Reality Applikation ohne Problem ermöglichen zu können.

3.3.3.1 MRTK3

Das Mixed Reality Toolkit (MRTK)¹ ist eine Sammlung von Tools, Skripten und Ressourcen, die speziell für die Entwicklung von Mixed-Reality-Anwendungen, einschließlich Augmented Reality, in Unity entwickelt wurden. MRTK3 ist eine Weiterentwicklung der vorherigen Versionen und bietet viele Vorteile für AR-Anwendungen:

- Interaktions- und Benutzerführung:
MRTK3 stellt eine Reihe von Interaktionskomponenten und -systemen zur Verfügung, die es Entwicklern ermöglichen, intuitivere Benutzererfahrungen in AR-Anwendungen zu gestalten. Dies umfasst Dinge wie das Platzieren von Objekten in der realen Welt, die Verfolgung von Handgesten und die Unterstützung von Blickverfolgung.
- Standardisierte APIs:
Durch die Verwendung von MRTK3 kannst du auf standardisierte APIs und Komponenten zugreifen, die speziell für AR-Anwendungen entwickelt wurden. Dies erleichtert die Implementierung von Funktionen wie Handgesten, Sprachsteuerung und Objektplatzierung.
- Einfache Konfiguration und Anpassung:
MRTK3 bietet eine einfache Konfiguration und Anpassung über die Unity-Oberfläche. Dies erleichtert die Anpassung deiner AR-Anwendung an spezifische Anforderungen und Use Cases.

3.3.3.2 Microsoft OpenXR Plugin

Das Microsoft OpenXR Plugin² ist eine Sammlung von Tools ist ein wichtiges Plugin für Unity, das die Integration von OpenXR-Unterstützung in die AR-Anwendung ermöglicht. OpenXR ist ein offener Industriestandard, der die Entwicklung von XR (Extended Reality)-Anwendungen, einschließlich Augmented Reality, erleichtert. Anschließend ein paar Punkte wieso dieses Plugin so wichtig ist:

- Geräteunabhängigkeit:
Durch die Verwendung von OpenXR und dem Microsoft OpenXR Plugin kann die AR-Anwendung auf verschiedenen XR-Geräten ausgeführt werden, ohne die Kernfunktionalität für jedes einzelne Gerät neu entwickeln zu müssen. Dies gewährleistet eine reibungslose Interaktion mit der HoloLens 2 und anderen XR-Geräten.
- Leistungssteigerung und Stabilität:
Die Nutzung von OpenXR und des Microsoft OpenXR Plugins kann die Leistung und Stabilität der AR-Anwendung erheblich verbessern. Sie gewährleisten eine reibungslose Ausführung der Anwendung auf dem Zielsystem und bieten eine optimale Benutzererfahrung.

¹Microsoft **MRTK3**

²Khronos **OpenXR**

3.3.4 Modellierungsprogramm

Die Erstellung der 3D-Modelle für die beiden Level erfordert ein Rendering-Programm. Die Entscheidung für das Rendering-Programm Blender wurde bereits zu Beginn des Projekts getroffen.

Diese Wahl basiert auf folgenden Gründen:

- **Kostenfrei und Open Source**

Blender ist kostenfrei und quelloffen, was bedeutet, dass es ohne Lizenzkosten genutzt werden kann. Dies ist besonders attraktiv bei der Entwicklung von AR-Anwendungen mit begrenztem Budget.

- **Echtzeit-Rendering**

Blender verfügt über einen Echtzeit-Renderer namens Eevee, der schnelle Vorschauen und Renderings ermöglicht. Dies ist hilfreich, um AR-Inhalte in Echtzeit anzuzeigen und zu überprüfen.

- **Integration mit AR-Frameworks**

Obwohl Blender keine direkte Unterstützung für AR-Funktionen bietet, können die erstellten 3D-Modelle und Animationen in AR-Entwicklungsumgebungen wie Unity oder Unreal Engine importiert werden, um dort AR-spezifische Funktionalitäten hinzuzufügen.

3.3.4.1 Wie funktioniert Blender im Allgemeinen?

Die nachfolgende Beschreibung hebt die Schlüsselaspekte und die Funktionalität von Blender für unseren speziellen Anwendungsbereich hervor.

- **Benutzeroberfläche und Interaktion**

Die Benutzeroberfläche von Blender ist komplex gestaltet, aber hoch anpassbar. Sie enthält 3D-Modelle, Ansichten, Fenster und Panels. Benutzer interagieren mit Objekten und Werkzeugen über Maus- und Tastaturbefehle, wobei erfahrene Nutzer Hotkeys oder Shortcuts verwenden können, um effizienter zu arbeiten.

- **3D-Modellierung**

Blender ermöglicht die Erstellung von 3D-Modellen durch die Verwendung von Primitiven wie Würfeln, Kugeln, Flächen und Kurven. Diese können dann bearbeitet und modifiziert werden, um komplexe Formen zu erstellen. Modellierungswerzeuge umfassen Extrusion, Verschiebung, Skalierung und Rotation.

- **Materialien und Texturen**

Zur Erzeugung realistischer Oberflächen können Materialien erstellt und Texturen auf Objekte angewendet werden. Blender erlaubt die Feinanpassung von Materialeigenschaften wie Diffusreflexion, Glanz, Transparenz und Emission.

- **Gemeinschaft und Ressourcen**

Blender verfügt über eine engagierte Benutzergemeinschaft, die umfassende Dokumentation, Tutorials und Foren bereitstellt. Diese Ressourcen erleichtern die Einarbeitung und die Lösung von Problemen.

Blender kommt in unserer Diplomarbeit in beiden Leveln zum Einsatz. Die Hauptanwendung des Programms findet im Level 2 statt, wo Blender für die digitale Modellierung wichtiger täglicher Gegenstände von Schülern genutzt wird. Das Ziel ist es, am Ende eine umfangreiche Sammlung von Objekten zu haben, um den Benutzern eine vielfältige Auswahl zu bieten.

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.

4.1.3 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.

4.1.4 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.

4.1.5 Manager in Unity

Für eine präzise und immersive Umsetzung von Augmented-Reality-(AR-)Applikationen werden spezielle Manager eingesetzt. Diese Manager bieten essenzielle Funktionen, die für eine erfolgreiche Umsetzung der verschiedenen Szenarien unerlässlich sind.

- **ARPlaneManager¹:** Der ARPlaneManager in Unity ist eine Komponente, die im Kontext von Augmented Reality (AR) eingesetzt wird, um horizontale Flächen in der realen Welt zu erkennen und zu verfolgen. Diese Flächen können beispielsweise Böden, Tische oder andere flache Oberflächen sein. Der ARPlaneManager gehört zum Unity-eigenen Mixed Reality Toolkit 3. Bietet Funktionen zur erleichterten Integration von AR-Elementen in die reale Umgebung.

Die Hauptaufgaben des ARPlaneManagers umfassen:

- **Erkennung horizontaler Flächen:** Der Manager identifiziert automatisch horizontale Flächen in der Umgebung des Benutzers. Dies ermöglicht es, virtuelle Objekte präzise auf diesen Flächen zu platzieren.
- **Verfolgung der Flächenbewegung:** Sobald Flächen erkannt wurden, verfolgt der ARPlaneManager ihre Bewegungen in Echtzeit. Dies ist besonders wichtig, um virtuelle Inhalte stabil auf den realen Flächen zu halten.
- **Texturmarkierung der Flächen:** Die erkannten Flächen können mit Texturen markiert werden, um ihre Grenzen für den Benutzer sichtbar zu machen und die Integration von virtuellen Objekten zu verbessern.
- **Unterstützung beim Platzieren von Objekten:** Der ARPlaneManager erleichtert das Platzieren von virtuellen 3D-Objekten in der realen Welt, indem er eine Referenz für die Position und Ausrichtung der erkannten Flächen bereitstellt.

¹Unity Managers

- **ARRaycastManager²:** Der ARRaycastManager in Unity ist eine Komponente, die im Kontext von Augmented Reality (AR) genutzt wird, um Raycasts von einem Ursprungspunkt, wie beispielsweise der Kamera der HoloLens 2, durchzuführen. Diese Raycasts treffen auf zuvor markierte und verfolgte Ebenen. Der ARRaycastManager ist Teil des Unity-eigenen Mixed Reality Toolkit 3. Und erlaubt die präzise Positionierung von virtuellen 3D-Objekten in der realen Welt.

Die Hauptaufgaben des ARRaycastManagers umfassen:

- **Durchführung von Raycasts:** Der Manager führt Raycasts von einem Ursprungspunkt aus, um Kollisionen mit bereits markierten und verfolgten Ebenen zu identifizieren.
- **Genauigkeit bei der Platzierung von Objekten:** Durch die Nutzung von Raycasts ermöglicht der ARRaycastManager eine genaue Platzierung von virtuellen 3D-Objekten in der realen Welt, basierend auf Benutzerinteraktionen.

Die erfolgreiche Umsetzung der funktionalen Anforderungen in den spezifischen Augmented-Reality-(AR-) Anwendungsszenarien des *Knapsack Problem Levels* sowie des *Ping Levels* hängt maßgeblich von der Integration und Anwendung der Manager ab, insbesondere des ARPlaneManagers und ARRaycastManagers. Diese Manager sind von grundlegender Bedeutung für die Schaffung einer qualitativ hochwertigen, präzisen und immersiven Benutzererfahrung.

Im Kontext des *Knapsack-Problem-Levels* spielt der ARPlaneManager eine zentrale Rolle. Er identifiziert und markiert horizontale Flächen in der Benutzerumgebung, die entscheidend für die genaue Platzierung von virtuellem Inventar sind. Die automatische Erkennung und kontinuierliche Verfolgung dieser Flächen durch den ARPlaneManager gewährleisten eine stabile Integration von AR-Elementen in die reale Umgebung.

Der ARRaycastManager führt Raycasts von der HoloLens 2-Kamera aus und identifiziert Kollisionen mit markierten Ebenen. Diese Funktionalität ist entscheidend für die präzise Positionierung von virtuellen 3D-Objekten in der realen Welt, insbesondere im Anwendungsfall des *Knapsack Problem Levels*. Der ARRaycastManager ermöglicht eine exakte Platzierung des Inventars basierend auf Benutzerinteraktionen.

Im speziellen Anwendungsfall des *Ping Levels* spielt der PlaneManager eine kritische Rolle. Er identifiziert die Fläche, auf der der Raycast auftrifft, um eine präzise Interaktion und Platzierung von AR-Elementen entsprechend den Benutzeraktionen zu ermöglichen.

Insgesamt sind diese Manager wichtige Ressourcen, die 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 auf dem *Knapsack-Problem-Level* als auch auf dem *Ping-Level* gewährleistet.

4.2 Objektdesign mittels Blender

4.2.1 Rendering und Optimierung für AR

Bei der Erstellung von 3D-Modellen für Augmented Reality (AR) ist die Optimierung entscheidend, um eine reibungslose Erfahrung auf Geräten wie der Hololens 2 zu gewährleisten. In Blender können verschiedene Techniken angewendet werden, um die Modelle für AR zu optimieren.

²Unity RaycastManager

Eine dieser Techniken ist die **Polygonreduktion**, bei der die Anzahl der Polygone in den Modellen reduziert wird, um die Belastung für die Hardware zu verringern. Blender bietet Werkzeuge wie den Decimate Modifier, um die Anzahl der Polygone effizient zu reduzieren, ohne die visuelle Qualität stark zu beeinträchtigen. Es ist essentiell, von Anfang an eine Modellierungspraxis mit geringer Polygonanzahl zu berücksichtigen. Ein erfahrener Modellierer kann identische Figuren mit reduziertem Polygonaufwand im Vergleich zu einem Anfänger erstellen, aufgrund seines fundierten Wissens über die Modellierung von Formen.

Bei der **Texturenoptimierung** sollte auf die Größe und Qualität der Texturen geachtet werden, da übermäßig große Texturen die Leistung beeinträchtigen können. Blender ermöglicht die Anpassung von Texturauflösung und -komprimierung. Im Verlauf der Texturierung wurde die Hololens mehrmals in Verbindung mit den Texturen integriert, um sicherzustellen, dass keine signifikanten Leistungseinbußen auftreten. Wenn Beeinträchtigungen festgestellt wurden, wurden Anpassungen vorgenommen, indem die Auflösung oder die Reflexionsstufen modifiziert wurden.

Es ist empfehlenswert, verschiedene Detailstufen zu implementieren, insbesondere wenn sich der Betrachter von einem Modell entfernt. Dies kann erreicht werden, indem verschiedene Modellversionen mit unterschiedlichen Polygonanzahlen erstellt werden. (**hab ich nicht gemacht, vielleicht mach ichs aber noch deswegen lass ich das stehen**)

4.2.2 Export- und Integrationsprozess

Die nahtlose Integration von Blender-Modellen in AR-Entwicklungsumgebungen ist entscheidend. Es sollten folgende Aspekte berücksichtigt werden:

Dateiformat

Blender unterstützt einige Dateiformate für den Export, aber da wir Unity nutzen, haben wir uns für Filmbox (FBX) entschieden. Das FBX-Dateiformat (Filmbox) ist ein proprietäres Dateiformat, das von Autodesk entwickelt wurde. Es dient dem Austausch von 3D-Modellen, Animationen, Texturen und anderen Szenendaten zwischen verschiedenen 3D-Anwendungen. FBX speichert Informationen über geometrische Formen, Materialien, Animationen, Kameras und Lichtquellen in einer hierarchischen Struktur.

Das FBX-Format basiert auf einer offenen Architektur, die es ermöglicht, komplexe 3D-Szenen mit verschiedenen Softwareanwendungen zu teilen. Es unterstützt dabei nicht nur die Geometrie und Materialien, sondern auch Animationen und andere wichtige Parameter. FBX verwendet eine hierarchische Struktur aus sogenannten Nodes, die verschiedene Elemente der 3D-Szene repräsentieren.

FBX-Dateien können sowohl binäre als auch ASCII-Formate haben. Das binäre Format ist kompakter und speichert die Daten in einem für Maschinen optimierten Binärformat. Im Gegensatz dazu ist das ASCII-Format besser lesbar für Menschen und erleichtert die Handbearbeitung von Dateien. **Koordinatensysteme**

Vor und während der Modellierung wurde oft geprüft, ob die Modelle eine sinnvolle Größenrelation zueinander haben. Zudem wurden alle Objekte am Ursprungspunkt und in dieselbe Richtung modelliert, um eine einheitliche Sammlung an fertigen Modellen zu erhalten und Verwirrungen zu vermeiden.

4.2.3 Blender-Add-Ons und Plugins

Es wurden einige Blender-Add-Ons und Plug-Ins verwendet, insbesondere aber das Plug-In LoopTools³ und das Import-Export Add-On Images as Planes⁴.

4.2.3.1 Looptools: Optimierung von Topologie und Oberflächen

Das Add-On Looptools hat sich bei der Optimierung der Topologie und der Oberflächen meiner 3D-Modelle als sehr nützlich erwiesen. Durch die Verwendung von Werkzeugen wie Circle konnte ich komplexere geometrische Formen aus einer einfachen Oberfläche extrahieren und gleichzeitig sicherstellen, dass die Topologie meiner Modelle sowohl ästhetisch ansprechend als auch für die weitere Bearbeitung geeignet ist.

4.2.3.2 Images as Planes: Effiziente Integration von Texturen

Das Add-On Images as Planes ermöglichte die nahtlose Integration von Texturen in meine 3D-Modelle. Durch die direkte Umwandlung von Bildern in ebene Flächen konnte ich realistische Texturen auf meine Modelle anwenden. Die Effizienz dieses Plug-ins trug dazu bei, den Arbeitsprozess zu beschleunigen und die Gesamtqualität meiner erstellten Objekte zu verbessern. Mit dem Plugin war es außerdem möglich, Vorschaubilder für die Modellierung hinter meinem Objekt zu platzieren, um das reale Objekt näher und realistischer zu modellieren.

4.2.4 Blender - Modes

In Blender gibt es zahlreiche verschiedene Modi⁵ die es dir erlauben verscheidene Aspekte eines Objekts zu bearbeiten.

- **Object Mode:** Der Default Modus, welcher für alle Objekt-typen zur Verfügung steht. Erlaubt die bearbeitung von Position, Rotation, Skalierung, Duplizierung und so weiter.

³Blender LoopTools

⁴Blender Images as Planes

⁵Blender Modi

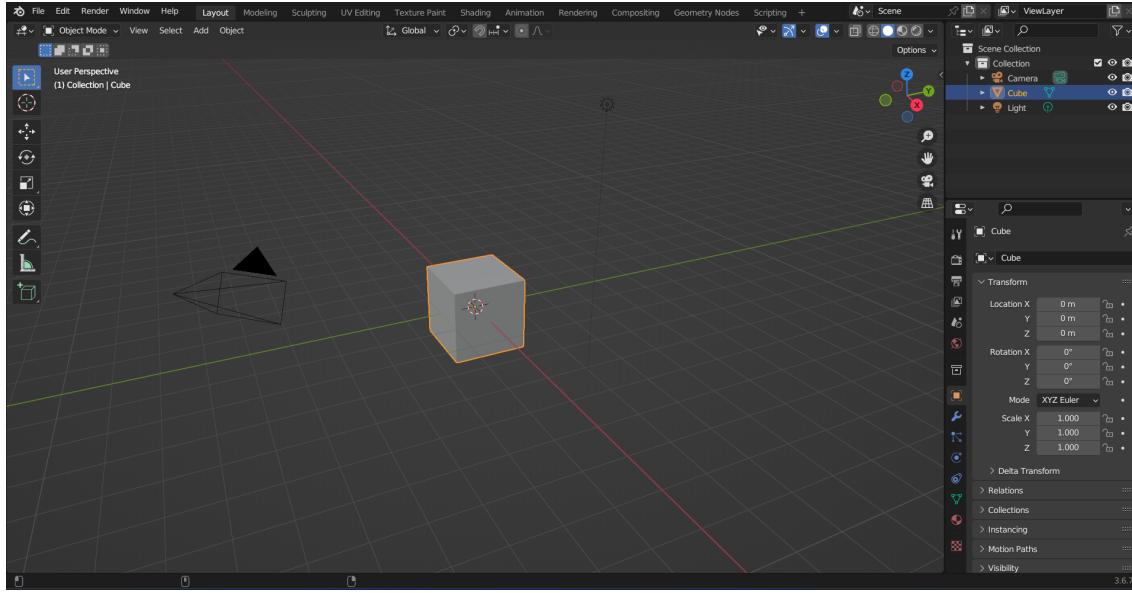


Abbildung 4.1: Standard Ansicht im Object Mode

- **Edit Mode:** Ist ein Modus für das Editieren einer Objektform mit verschiedenen Werkzeugen. Man kann die einzelnen Vertices⁶, Kanten und Flächen auf Basis von verschiedenen Kontrollpunkten bearbeiten.

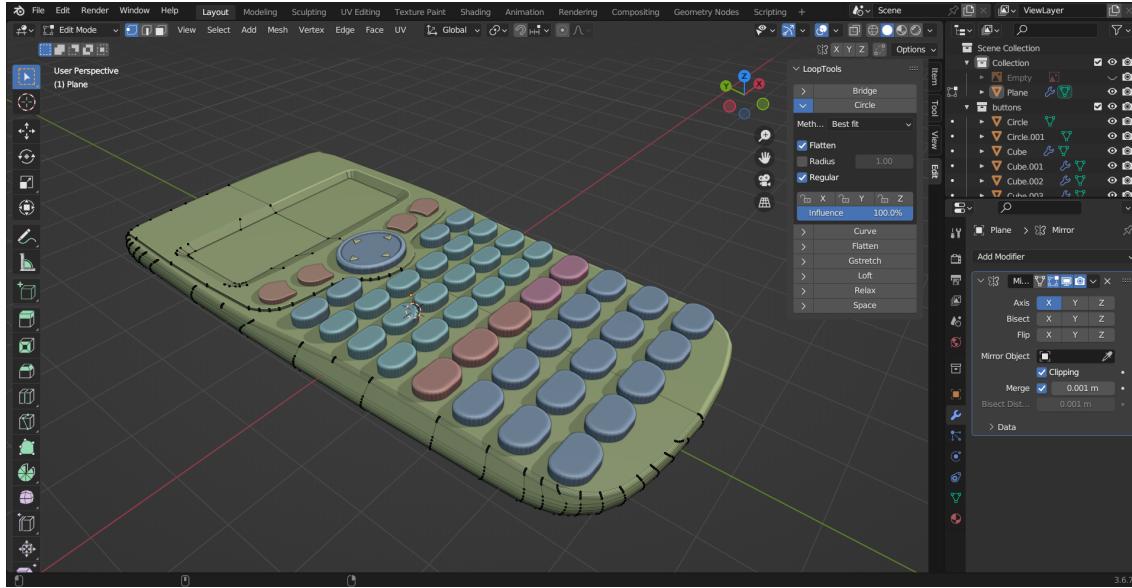


Abbildung 4.2: Edit Mode Ansicht auf das Calculator Objekt

- **Texture Paint Mode:** Ein Mesh-Only⁷ Modus der es dir ermöglicht Texturen direkt auf das Model im 3D-Viewport zu zeichnen.

⁶Blender Vertices

⁷Blender Mesh

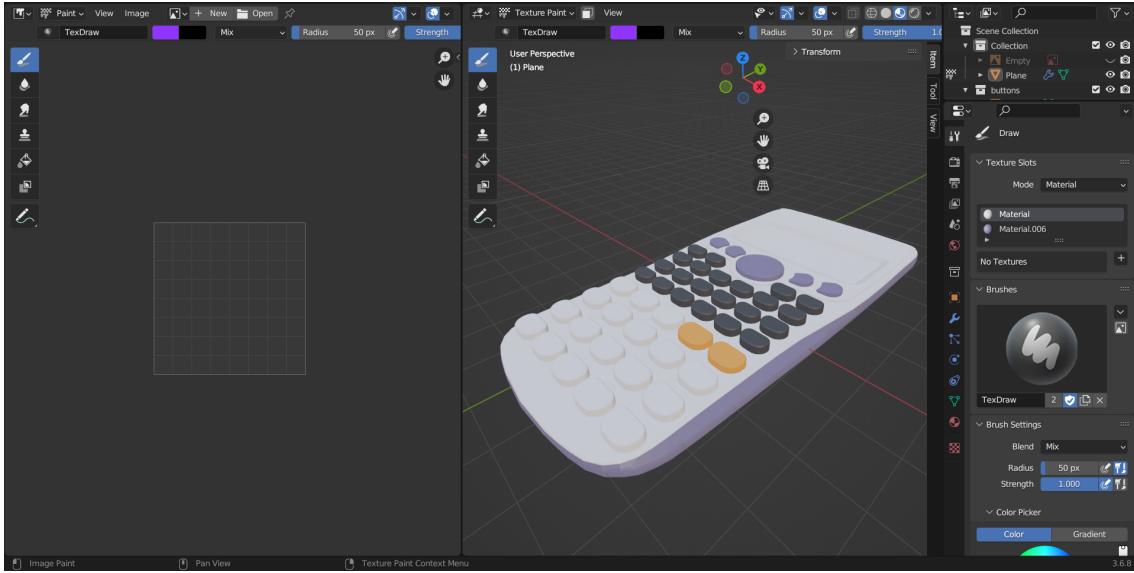


Abbildung 4.3: Standard Ansicht im Texture Paint Mode

4.2.5 Blender - Hierarchie

Im folgenden Abschnitt wird erklärt, wie Blender aufgebaut ist und wie die einzelnen verwendeten Werkzeuge und Modifier⁸ funktionieren. Es wurden einige Modelle erstellt, aber für ein einfacheres Verständnis wurde für alle Beispielbilder das Taschenrechner-Modell verwendet.

⁸Blender Modifier

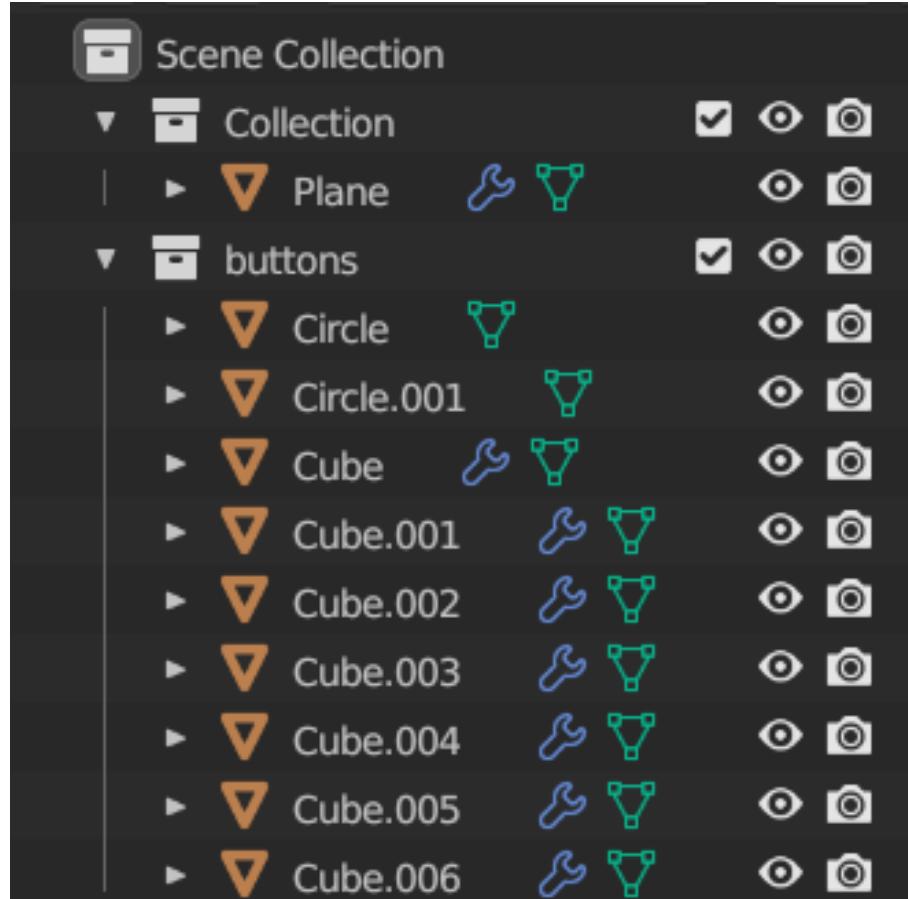


Abbildung 4.4: Ansicht auf die Hierarchie des Taschenrechner Modells

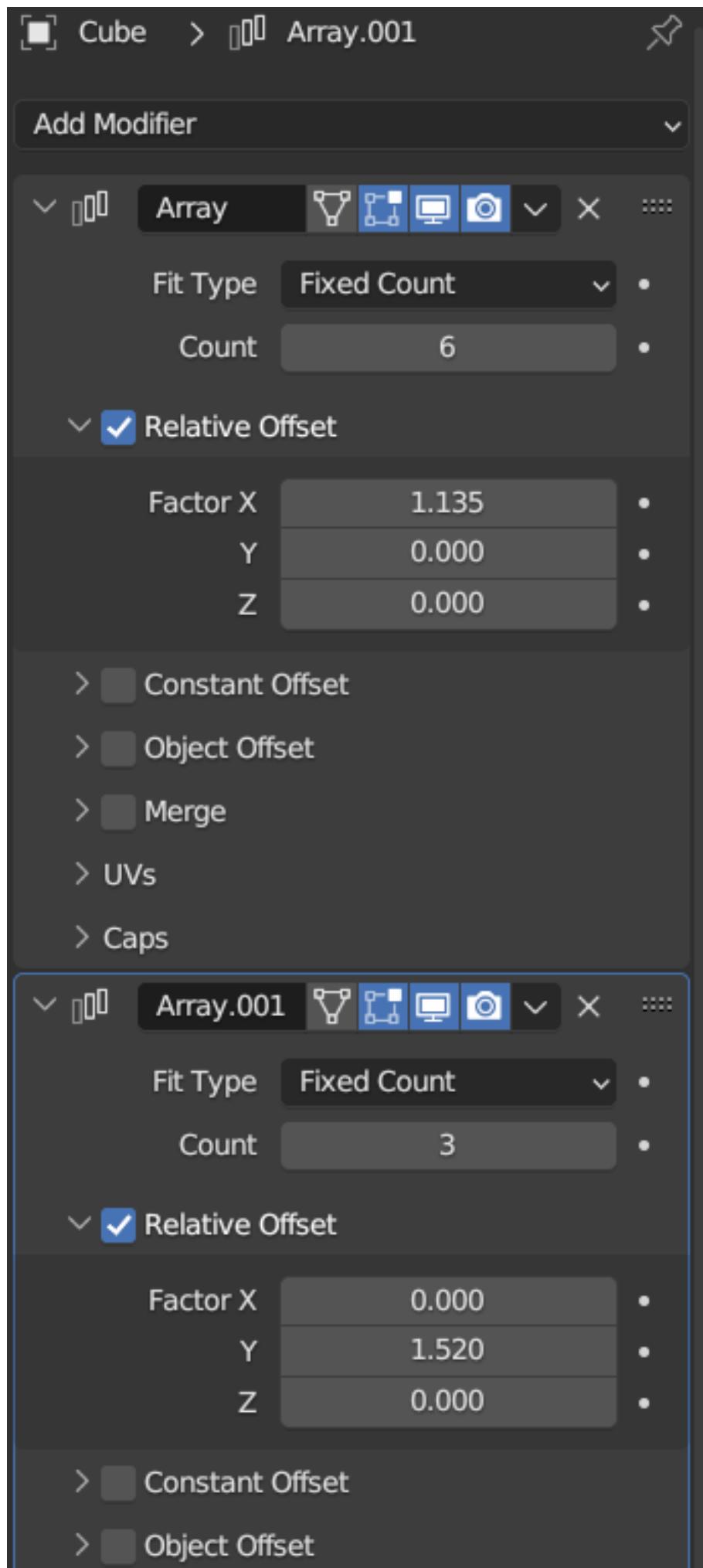
In dieser Abbildung ist die Hierarchie und der Inhalt des Hauptmenüs zu sehen. Wie zu sehen ist, besteht die Szene aus vielen wichtigen Komponenten die zusammenspielen, um die gewünschte Funktion zu erzielen. Darunter sind folgende Objekte:

- **Scene Collection:** Die umfassende Collection enthält das gesamte Modell mit allen Teilen. Collections werden mehrmals in der Abbildung verwendet, um die Hierarchie besser zu strukturieren. Sie haben jedoch keinen Einfluss auf das Objekt selbst.
- **Plane:** Das Mesh dient zur groben Formgebung des Taschenrechners.
- **Circles/Cubes:** Die Circles und Cubes sind die einzelnen Buttons.

In der Abbildung ist zu erkennen, dass einige Formen mit einem blauen/grünen Zeichen markiert sind. Das blaue Zeichen kennzeichnet einen Modifier, das heißt, das Objekt hat einen oder mehrere Modifier. Das grüne Zeichen steht für Data Properties und zeigt an, dass sich diese verändert haben.

4.2.5.1 Blender - Modifier

Im vorherigen Abschnitt wurden erwähnt das Modifier verwendet werden. Modifier ermöglichen verschiedene zusätzliche Funktionen an einem Objekt.



In der Abbildung ist der Array Modifier zweimal zu sehen. Dieser wurde verwendet, um nicht jeden Knopf des Taschenrechners einzeln modellieren zu müssen, sondern nur einen bestimmten Knopf horizontal oder vertikal mit beliebiger Versetzung zu kopieren. Dabei haben die einzelnen Zeilen verschiedene Funktionen.

- **Fit Type:** Es besteht die Möglichkeit, entweder eine feste Anzahl von Objektkopien auszuwählen oder eine Länge anzugeben, die dem Array entspricht.
- **Count:** In dem Feld rechts daneben steht die Anzahl, wie oft das Objekt kopiert werden soll.
- **Relative Offset:** Die relative Verschiebung basiert immer auf dem zuvor kopierten Modell und hat drei Faktoren, die jeweils eine der drei Koordinaten darstellen. In diesem Fall soll es um 1.520 auf der x-Achse nach rechts verschoben werden.

4.3 Menü

Im folgenden Abschnitt wird die hierarchische Struktur des Menüs in Unity erläutert und die Schritte beschrieben, die zur Finalisierung der Implementierung durchlaufen wurden.

Die iterative Entwicklung des Menüs durchlief mehrere Phasen, beginnend mit der Konzeption und Planung der Benutzeroberfläche bis hin zur finalen Umsetzung. Dieser Prozess umfasste die Entscheidungen zum Design des Layouts, des Farbschemas und der Interaktionselemente. Anschließend wurden die Funktionalitäten programmiert.

Während der Entwicklung wurden regelmäßig Überprüfungen und Anpassungen vorgenommen, 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 finalen 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 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 es verwenden kann.

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.

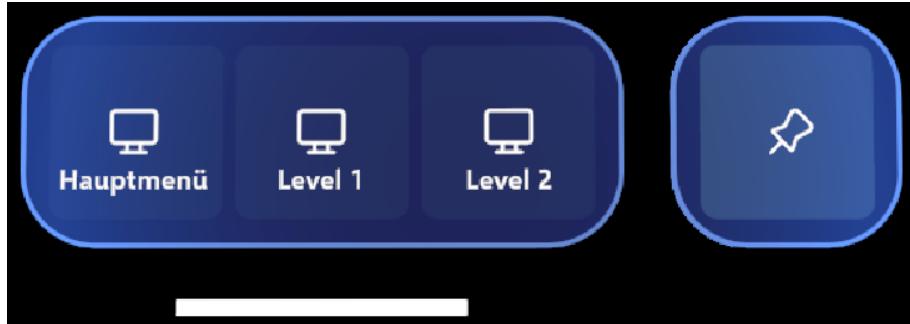


Abbildung 4.6: Darstellung der Menübar im Unity Editor

Die Abbildung 4.6 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 Initiation des Ladevorgangs für die verschiedenen Spielstufen. Hervorzuheben ist der Pin-Button in Kombination mit dem begleitenden weißen Balken. Diese Funktion ermöglicht es dem Benutzer, das Menü an gewünschten Positionen zu verankern und bietet somit Flexibilität bei der Positionierung entsprechend individueller Präferenzen. Ein solcher Einsatz ist opportunistisch, 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 würde.

In der Entwurfsphase 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. Dies führte zu einer Abhängigkeit von den vorgefertigten Designelementen und Funktionalitäten, was möglicherweise die Einzigartigkeit und das spezifische Designkonzept des Projekts beeinträchtigte.

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.

4.3.1.1 Probleme beim Erstentwurf

Problem 1: Das Hauptziel bestand darin, ein benutzerfreundliches Menü zu gestalten, das alle erforderlichen Funktionen enthält und dennoch übersichtlich ist. Nach mehreren Iterationen und Tests mit Probanden, die nicht mit dem Projekt vertraut waren, stellte sich heraus, dass der ursprüngliche Entwurf erhebliche Mängel bei der Dokumentation und Erklärung der verschiedenen Spielstufen 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 erreichenden Ziele vermittelt.

Die Identifizierung dieses Problems betont die Wichtigkeit einer umfassenden Benutzerforschung 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 Spiellevel und ihrer Aufgaben zu adressieren und somit die Benutzererfahrung zu verbessern.

Problem 2: Während des Testprozesses stellte sich heraus, dass das Navigationsmenü oft eine Behinderung darstellte, insbesondere für Entwickler, die Funktionen wiederholt testeten und Levels 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.

Um dieses Problem zu lösen, war die Entwicklung eines neuen Konzepts notwendig, das die Schwächen des ursprünglichen Entwurfs anspricht und möglicherweise sogar vollständig beseitigt. Dazu mussten die Anforderungen und Ziele des Navigationsmenüs sorgfältig neu bewertet und kreativ neu gestaltet werden, um eine nahtlose Integration in den Entwicklungsprozess zu gewährleisten und die Produktivität des Teams zu steigern.

4.3.2 Finalversion

Für das neue Konzept wurde eine Analyse populärer und trendiger AR/VR-Spiele durchgeführt, um die Gründe für die verbesserte Benutzererfahrung in diesen Spielen zu untersuchen. Dabei wurden spezifische Merkmale und Designentscheidungen identifiziert, die zu einer höheren Nutzerzufriedenheit führen. Auf dieser Grundlage wurde das Design des Menüsystems vollständig überarbeitet, wobei das bisherige Nahmenü entfernt wurde.

Anstelle des Nahmenüs wurden vorgefertigte Objekte in das Hauptmenü integriert, welche zuvor mit Blender modelliert wurden. Diese Objekte ermöglichen es dem Nutzer, direkt mit der Spielumgebung zu interagieren, bevor das eigentliche Spiel gestartet wird. Durch diese Interaktionsmöglichkeiten wird dem Benutzer eine immersive Erfahrung geboten, die es ihm ermöglicht, sich bereits vor Spielbeginn mit der Welt vertraut zu machen und eine Verbindung zu ihr aufzubauen. 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, indem sie dem Benutzer klare Anhaltspunkte und Handlungsmöglichkeiten bieten.

Diese Neuausrichtung des Menüdesigns basiert auf einer gründlichen Analyse aktueller Trends und bewährter Praktiken im Bereich der AR/VR-Spiele. Das Ergebnis ist ein

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

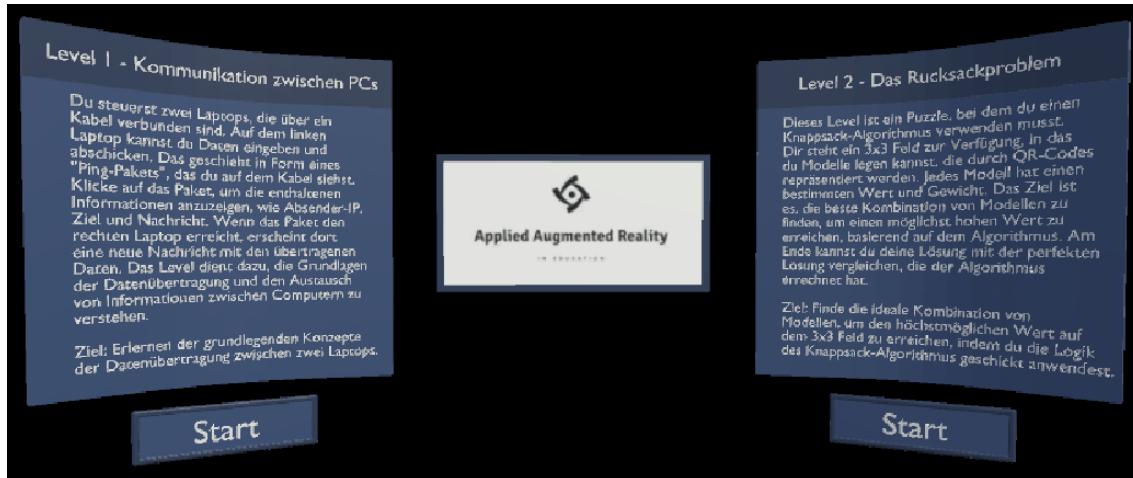


Abbildung 4.7: Darstellung des Finalen Menüs im Unity Editor

Die in der Abbildung 4.7 dargestellten Objekte bestehen aus zwei separaten Tafeln, die durch unser Diplomarbeitslogo voneinander getrennt sind. Jede Tafel enthält den Titel des Levels 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.

Im Gegensatz zum Erstentwurf, der ein einheitliches Menü für alle Spiellevel vorsah, weist diese Version eine Differenzierung auf. Die Entscheidung, das Menü innerhalb der einzelnen Level 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 Levels entfernt und durch ein simples Handmenü ersetzt.

Die Neugestaltung des Menüs innerhalb der Spiellevel fördert die Immersion und Konzentration des Benutzers auf das Spielerlebnis, indem visuelle Ablenkungen minimiert werden. Die Verwendung eines Handmenüs ermöglicht eine unkomplizierte Interaktion, die nahtlos in das Spiel integriert ist und so die Benutzererfahrung verbessert.



Abbildung 4.8: Darstellung des Finalen Menüs im Unity Editor

Wie in der Abbildung 4.8 gezeigt, besteht dieses Menü ausschließlich aus einem Zurück-Knopf, 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 Zurückknopf 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 visuelle Ablenkung minimiert und die Benutzererfahrung optimiert, indem nur relevante Informationen und Interaktionselemente präsentiert werden, wenn sie benötigt werden.

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 Spielleveln 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.

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.

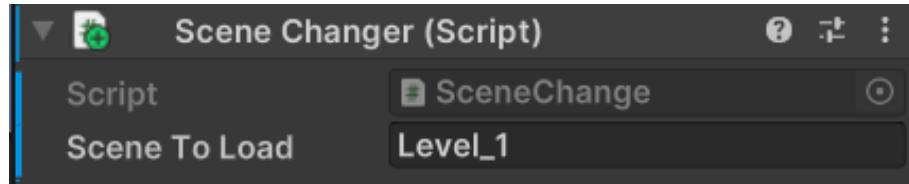


Abbildung 4.9: Darstellung der SceneToLoad Variable, anhand des Level1 Buttons.

```

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 }
```

Listing 4.1: Auf Knopfdruck Szene wechseln.

Die Hauptkomponenten des Codes sind:

- `sceneToLoad`: Dies ist eine öffentliche Variable vom Typ String, die den Namen der zu ladenden Szene enthält. Sie kann im Unity-Inspector definiert werden, um die Zielszene anzugeben (siehe Zeile 6).
- `ChangeScene()`: Dies ist eine öffentliche Methode, die aufgerufen wird, um die Szene zu wechseln. Sie prüft dass die aktive Szene nicht mit der in `sceneToLoad` angegebenen Szene identisch ist.
- `SceneManager.GetActiveScene().name`: Dies ruft den Namen der aktuell geladenen Szene ab.
- `PlayerPrefs.DeleteAll()`: Diese Methode löscht alle gespeicherten PlayerPrefs, was hilfreich sein kann, um sicherzustellen, dass keine alten Daten zwischen den Szenenwechseln erhalten bleiben (siehe Zeile 12). Ist vor allem wichtig, da wir ohne das Löschen der PlayerPrefs Probleme mit dem HoloLens Cache beim Wechseln innerhalb der Szenen hatten.
- `SceneManager.LoadScene(sceneToLoad, LoadSceneMode.Single)`: Diese Methode lädt die in `sceneToLoad` gespeicherte Szene im Single-Modus, d.h. die aktuelle Szene wird entladen und die neue Szene geladen (siehe Zeile 13).

4.3.4 Setzen des Menüs

Da sich das alte Menü in Version 1 automatisch mit dem Benutzer bewegte, war es kein Problem, es in das Hauptmenü zu laden, aber bei den neuen Modellen in Version 2 muss man das Menü selbst platzieren. Dafür gibt es das Skript `MenuSpawn.cs`.

```

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 }
```

Listing 4.2: Auf Knopfdruck Szene wechseln. label

Beim Start der Anwendung wird die Methode *SpawnPrefabMenu()* ausgeführt. Diese Methode ist für die Positionierung des Menüs verantwortlich. Die im Codeabschnitt ?? 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 *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.

4.3.5 UI/UX

Im folgenden Abschnitt werden die Ziele genannt, auf die sich bei der Menüerstellung konzentriert wurde.

- **Blickzentrierung und Interaktionsmodelle:** Die Menüleiste bewegt sich auf Hüft-höhe mit dem Benutzer mit und ermöglicht so eine natürliche und intuitive Interaktion. Der Benutzer kann die Bedienelemente einfach durch Blickkontakt auswählen, was die Benutzerfreundlichkeit erhöht und die Bedienung der Anwendung erleichtert.
- **Konsistenz im Design:** Die klare Gestaltung der Buttons mit aussagekräftigen Symbolen trägt zur Konsistenz und Benutzerfreundlichkeit bei. Eine einheitliche visuelle Sprache erleichtert es dem Benutzer, die Funktionen der Buttons zu verstehen, selbst wenn er sie zum ersten Mal verwendet.
- **Kontextsensitive Funktionen:** Der Debug-Button bietet erweiterte Funktionen, die speziell für Entwickler relevant sind. Diese kontextsensitiven Optionen sind wichtig für die Fehlersuche und tragen dazu bei, die Entwicklungszeit zu optimieren. Gleichzeitig bleibt die Benutzeroberfläche für den Endbenutzer sauber und übersichtlich.
- **Anpassungsmöglichkeiten für den Benutzer:** Die Option, das Menü mit dem Pin-Button zu fixieren und mit der Grab-Bar frei zu bewegen, gibt dem Benutzer die Kontrolle über die Positionierung des Menüs. Diese Anpassungsmöglichkeiten tragen dazu bei, die Anwendung an verschiedene Nutzerszenarien anzupassen und die individuellen Bedürfnisse der Benutzer zu berücksichtigen. Feedback und Animationen

4.4 Ping Level

In diesem Level wird das IT-Grundprinzip eines Pings zwischen zweier PCs dargestellt. Das Kabel zwischen den zwei PCs wird von der HoloLens getracked und mittels Kurvenberechnung wird dann eine unsichtbare Kurve über dieses Kabel gezeichnet. Wenn dann der Benutzer auf die Enter Taste auf einem PC drückt wird ein Ping-Paket simuliert und auf dieser Kurve von einem PC zu dem anderen geschickt.

4.4.1 Object Tracking

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

4.4.2 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 Level

Im zweiten Level dieses Projekts steht das Knapsack-Problem im Fokus. Ziel ist es, diesen Programmieralgorithmus mithilfe von Augmented Reality (AR) visuell darzustellen. Dieser Algorithmus wird nicht nur in der Höheren Technischen Lehranstalt (HTL) vermittelt, sondern die Benutzer sollen ihn auch selbst programmieren können.

Der Level beginnt damit, dass der Benutzer aufgefordert wird, auf eine horizontale, flache Oberfläche zu schauen. Diese Oberfläche kann ein Tisch, der Boden oder ähnliches sein. Der Benutzer wird dann gebeten, für eine bestimmte vorgegebene Zeit auf diese Oberfläche zu schauen. Nach Ablauf der vorgeschriebenen Zeit wird dann das Inventar als auch das *infoObjekt* platziert.

Das Inventar wird durch ein 3x3 zweidimensionales Gitter repräsentiert, ähnlich wie das Inventar in einem Spiel. Zusätzlich befinden sich auf der Oberfläche 11 Bauklötze, die mit QR-Codes versehen sind. Diese Bauklötze repräsentieren die Items, die der Benutzer in das Inventar legen kann. Durch Aufheben und Nahheranhalten an die HoloLens wird der QR-Code gescannt. Dadurch werden das dazugehörige 3D-Modell, der Wert, Gewicht und der Name des Items angezeigt. Diese Informationen sind für den Benutzer wichtig, um das Gewicht des Items und seinen Einfluss auf das Inventar zu verstehen.

Wenn der Bauklotz erfolgreich platziert wurde, wird automatisch der Knapsack-Algorithmus für die perfekte Lösung, und die Berechnung des eigenen Inventars gestartet um stets den aktuellen Wert zu sehen.

Insgesamt bietet dieses Unity Level für die HoloLens 2 eine interaktive und visuelle Erfahrung, bei der die Benutzer das Knapsack-Problem nicht nur verstehen, sondern auch praktisch anwenden können.

4.5.1 Knapsack-Problem Level Hirarchie

In dem folgendem Abschnitt wird darauf eingegangen wie eine Szene in dem Unity Editor aufgebaut ist und wie diese Grundsätzlich funktionieren.

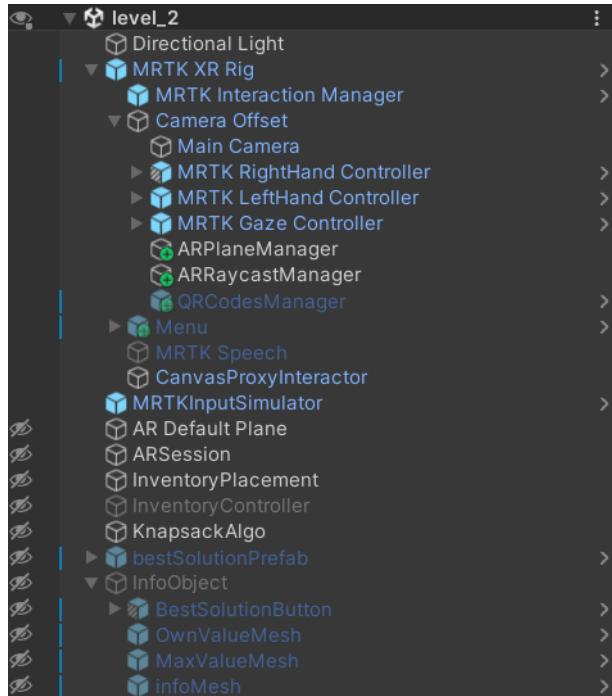


Abbildung 4.10: Knapsack-Problem Levels Hirarchie Unity Editor.

In dieser Abbildung ist die Hirarchie und der Inhalt des Knapsack-Problem Levels / Szene-2⁹ zu sehen. Wie zu sehen ist, besteht die Szene aus vielen wichtigen Komponenten die zusammenspielen, um das gewünschte Ergebnis zu erzielen. Darunter sind folgende Objekte:

- **level-2**

Die Scene in der Alle Unity-Game-Objekte enthalten sind.

- **AR Default Plane**

In der Augmented Reality (AR)-Entwicklung bezieht sich ein Plane¹⁰ normalerweise auf eine erkannte horizontale Fläche in der realen Welt, auf der virtuelle Objekte platziert werden können. Diese Flächen können zum Beispiel Tische, Böden oder andere ebene Oberflächen sein. Das Erkennen und Tracking von Planes ist entscheidend, um AR-Objekte realistisch in die Umgebung zu integrieren.

- **ARSession**

In Unity und AR Foundation bezieht sich die ARSession im Allgemeinen auf die Hauptkomponente, die die AR-Funktionalitäten steuert und koordiniert.

- **InventoryPlacementController**

Ist ein Game Objekt, dem das *InventoryPlacementController.cs* Script zugewiesen ist und bei Szenen-Start aktiv ist. Dieses Game Objekt kümmert sich darum, dass das Inventar richtig in der Augmented Reality Welt platziert wird.

- **InventoryController**

Das Game Objekt, dem das *InventoryController.cs* Script zugewiesen ist und nach Abschluss des InventoryPlacement Scripts aktiviert wird. Dieses Objekt ist die Hauptschnittstelle zwischen den QR-Codes und dem 3D Inventar Modell und kümmert sich darum neue QR-Codes innerhalb des Modells zu erkennen und zu verspeichern.

⁹Unity Scene

¹⁰Unity Plane

- **KnapsackSolver**

GameObjekt, dem das *KnapsackSolver.cs* Script zugewiesen ist und bei platzieren eines QR-Codes innerhalb des Inventars auslöst, den maximalen erreichbaren Wert, die perfekte Lösung und den Wert des selbst erstellten Inventars berechnet.

- **bestSolutionPrefab**

Dieses Objekt ist das Prefab¹¹ für die Perfekte Lösung. Diesem Objekt ist das 3D Modell des Inventars untergeordnet und diesem Inventar sind Prefabs für QR-Codes untergeordnet. In jeder Zelle des Inventars ist ein eigenes QRCode Prefab um anschließend die berechnete perfekte Lösung darstellen zu können. Diesem Objekt ist das *PerfectSolutionVisualizer.cs* Script zugewiesen, dass sich darum kümmert die perfekte Lösung anzuzeigen.

- **InfoObject**

Dieses Game Objekt ist eine Sammlung aus mehreren Unity Game Objekten. Dieses Objekt wird bei platzieren des Inventars neben dem Inventar mitplatziert. Bei den Objekten handelt es sich hierbei um den *BestSolutionButton*, der bei Knopfdruck einer der mehreren besten Lösungen visuell veranschaulicht, und drei *TextMeshes*, um die berechneten Werte, und auch Informationen anzuzeigen. Zusätzlich ist hier zusehen, dass diese 4 Objekte dem *infoObject* untergeordnet sind, was bedeutet, dass diese Objekte *children* von dem *infoObject* sind. Das übergeordnete Objekt wird hier dann als *parent* bezeichnet.

In der Abbildung ist 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 wiederum sind daher genau das Gegenteil. Das bedeutet, 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 Test Meshes oder Lables.

4.5.2 Unity Prefabs

Unity bietet eine äußerst praktische Funktion zur Erstellung und Wiederverwendung von Game-Objekten, Prefabs. Prefabs sind vorgefertigte Bausteine, die als Vorlagen dienen. Sie ermöglichen es, einmal erstellte Objekte als standardisierte Vorlagen zu speichern und dann beliebig oft in verschiedenen Szenen oder Projekten zu verwenden.

Diese Vorlagen bieten eine Reihe von Vorteilen. Einerseits ermöglichen Prefabs eine effiziente und konsistente Gestaltung von Spielen, indem sie die Wiederverwendung von Designelementen erleichtern. Stellen Sie sich vor, Sie haben eine komplexe Szene mit verschiedenen Objekten erstellt, darunter Charaktere, Umgebungen und Effekte. Anstatt jedes Mal von Grund auf neu zu beginnen, können Sie diese Elemente als Prefabs speichern und sie dann einfach in neuen Szenen wiederverwenden. Durch die Verwendung von Prefabs

¹¹Unity Prefab

können Sie nicht nur Zeit sparen, sondern auch sicherstellen, dass Ihr Spiel eine konsistente Designästhetik aufweist.

Außerdem ermöglichen Prefabs eine einfache Aktualisierung und Iteration von Game-Objekten. Wenn Sie beispielsweise Änderungen an einem bestimmten Objekt vornehmen müssen, können Sie einfach das entsprechende Prefab bearbeiten. Diese Änderungen werden automatisch auf alle Instanzen dieses Prefabs angewendet, die in Ihrer Szene verwendet werden. Prefabs sind ein unverzichtbares Werkzeug für die effektive Spieleentwicklung in Unity, da

sie den Prozess der Aktualisierung und Feinabstimmung von Spielen wesentlich effizienter machen. Durch ihre Verwendung können Entwickler Zeit sparen, die Konsistenz ihres Spiels gewährleisten und den Prozess der Aktualisierung und Iteration von Game-Objekten optimieren.

4.5.3 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 Objekten 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 Generierung der Codes erfolgt und auf welche Weise sie innerhalb der Anwendung zur Interaktion mit den realen Objekten verwendet werden.

4.5.3.1 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.

Die zugehörigen Zahlen erhalten in der Software, genauer gesagt in der Klasse *QRItem.cs*, einen Kontext. Der folgende Codeausschnitt zeigt dies:

```

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 Zahlen die folgenden Informationen zuordnet:

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

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

4.5.3.2 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:

- **Initialisierung des QR-Tracking-Systems:** Zur Aktivierung des QR-Tracking-Systems werden die erforderlichen Ressourcen und Komponenten gestartet, um die Erkennung von QR-Codes zu ermöglichen. 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:** 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 erkannt. Das Ereignis signalisiert, dass ein QR-Code erkannt wurde und gibt Informationen über den erkannten QR-Code, wie seine Daten und Position, weiter.

- **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 Eigenschaftsanzeige:** 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.

4.5.3.3 Interaktion mit QR-Codes

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 als Repräsentationen der realen Objekte, die wir in der virtuellen Welt darstellen möchten.

Wie in Abbildung X zu sehen ist, sind die Bauklötze mit QR-Codes versehen. Diese Bauklötze 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 und dem Benutzer eine interaktive Erfahrung bieten.

Dem Benutzer wird die Möglichkeit geboten, die Bauklötze 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.

4.5.3.4 Bestimmen der Position QR-Codes

4.5.3.5 Visualisierung von QR-Codes

Nachdem die genaue Platzierung der QR-Codes bestimmt wurde, steht die Aufgabe an, ihre Visualisierung in der virtuellen Welt umzusetzen.

Hierfür wird die Funktionalität von Unity Prefabs genutzt. Diese ermöglichen die Erstellung visueller Repräsentationen der QR-Codes und ihre nahtlose Integration in die virtuelle Umgebung. Weitere Informationen zu Prefabs und ihrer Funktionsweise finden Sie im Abschnitt ??.

Innerhalb jedes Prefabs sind alle relevanten Informationen enthalten, die für die korrekte Darstellung des QR-Codes erforderlich sind. Dazu gehören nicht nur das 3D-Modell des QR-Codes, sondern auch zugehörige Daten wie Name, Wert und Gewicht. Diese Daten

sind entscheidend für die Interaktionen innerhalb der virtuellen Umgebung.

Um die Funktionalität des QR-Codes in der virtuellen Welt zu gewährleisten, wird dem Prefab das Skript *QRCode.cs* zugewiesen. Dieses Skript steuert die visuelle Darstellung des QR-Codes sowie sämtliche Interaktionen, die damit verbunden sind. Durch die Zuweisung dieses Skripts wird sichergestellt, dass die QR-Codes nicht nur korrekt angezeigt, sondern auch innerhalb der virtuellen Umgebung interaktiv sind.

4.5.3.6 Zugriff auf QR-Codes bereitstellen

Wie bereits im Abschnitt ?? erwähnt, benötigen andere Teile der Anwendung Zugriff auf die aktuell erkannten QR-Codes in der Szene, um entsprechend darauf reagieren zu können. 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. Jedoch da wir Threads zugriff verweigern müssen, um die Daten zu schützen, kann es zu einer Verzögerung 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.

4.5.4 Inventory Placement Controller Game Objekt

Um eine präzise Interaktion zwischen der realen und augmentierten Realität zu gewährleisten, ist der Zugriff auf die Kamera erforderlich, um die physische Umgebung präzise zu erfassen. Durch die Analyse der erfassten Umweltdaten können relevante Ebenen identifiziert werden, die entscheidend sind, um eine akkurate Platzierung des Inventar-Objekts zu gewährleisten.

In diesem Abschnitt wird das *Inventory Placement Controller* Game Objekt mit dem angehängten *InventoryPlacementController.cs* Script behandelt. Letzteres beinhaltet die **InventoryPlacementController** Klasse, welche sämtliche Funktionen zur Berechnung und Platzierung des Inventars umfasst. Diese Funktionalitäten werden im Folgenden detailliert erläutert.

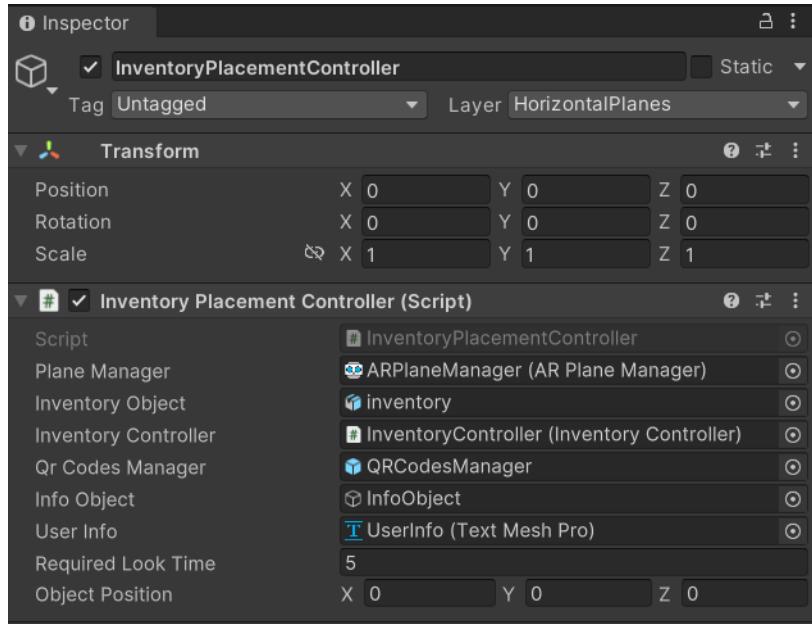


Abbildung 4.11: inventoryPlacement Objekt im Editor

Die Darstellung des *inventoryPlacementController*-Objekts im Unity Editor, wie sie in Abbildung 4.11 gezeigt wird, bietet eine Schnittstelle zur Konfiguration verschiedener Parameter. Hierzu zählen insbesondere die Zuweisung der Ebenenposition, die Feinjustierung der Koordinaten innerhalb des Unity Editors sowie die Verbindung mit der Komponente *InventoryPlacementController.cs*. Zusätzlich werden voreingestellte Werte für Schlüsselvariablen angezeigt, die für die Ausführung des Codes von entscheidender Bedeutung sind. Diese Auflistung und Erläuterung dieser Variablen umfasst:

- **Plane Manager:** Hierbei handelt es sich um eine Zuweisung zum Game Objekt des *ARPlanesManagers* aus der Szene Level 2.
- **Inventory Object:** Diese Referenz ist mit dem 3D-Modell des Inventars aus dem Prefab-Ordner verknüpft.
- **Inventory Controller:** Diese Variable ist mit dem Skript des *InventoryControllers* verbunden.
- **QR Codes Manager:** Hierbei handelt es sich um einen Verweis auf das Game Objekt des *QRCodeManagers* aus der Szene Level 2.
- **Info Object:** Diese Referenz ist mit dem Game Objekt *infoObject* aus der Szene Level 2 verbunden.
- **User Info:** Referenz auf das TextMesh in der *Main Camera*, dass zur Anzeige von Anweisungen und Tipps dient.
- **Required Look Time:** Diese Variable gibt die vorgeschriebene Zeitdauer an, für die der Benutzer auf ein Plane blicken muss, damit das Inventar platziert wird.

Durch den Unity Editor besteht die Möglichkeit, die Werte dieser Variablen direkt zu manipulieren, was direkte Auswirkungen auf die Ausführung des Codes hat. Diese direkte Manipulation bietet eine intuitive und effiziente Möglichkeit, die Einstellungen anzupassen und die Funktionalität des Programms entsprechend anzupassen.

4.5.4.1 InventoryPlacementController Klassenvariablen

```

1 public ARPlaneManager planeManager;
2 public GameObject inventoryObject;
3 public InventoryController inventoryController;
4 public GameObject qrCodesManager;
5 public GameObject infoObject;
6 public TextMeshPro userInfo;
7 public float requiredLookTime = 5.0f;
8 public Vector3 objectPosition;
9
10 private ARPlane selectedDeskPlane;
11 private float lookStartTime = -1f;
12 private bool objectPlaced = false;
13 private float heightOffset = 0.001f;
14
15 private bool canStartScript = false;

```

Listing 4.3: Klassenvariablen der InventoryPlacementController Klasse

Die dargestellten Klassenvariablen in Codeabschnitt 4.3 sind integraler Bestandteil der *InventoryPlacementController* Klasse. Öffentliche (**public**) Variablen repräsentieren Objekte und Werte, die im Unity Editor festgelegt und übergeben werden können oder von anderen Klassen für die Funktionalität benötigt werden. Dies ermöglicht einen direkten Zugriff auf diese Objekte sowohl innerhalb der eigenen Klasse als auch von anderen Klassen aus. Die Vektor-Variablen *objectPosition* spielt dabei eine entscheidende Rolle für die präzise Platzierung des Inventar-Objekts und ebenfalls spielt das TextMesh *userInfo* eine wichtige Rolle um dem Benutzer im Laufe der Applikation Anweisung und Tips zu geben.

Hingegen dienen die privaten (**private**) Variablen der lokalen Speicherung von Werten, die ausschließlich innerhalb der Klasse benötigt werden und von keiner anderen Klasse verwendet werden sollen.

Dieser Ansatz der Kapselung von Daten erlaubt eine klare Trennung zwischen öffentlich zugänglichen Daten und internen Variablen, was eine effiziente und strukturierte Entwicklung und Wartung des Codes ermöglicht.

4.5.4.2 Startverzögerung

```

1 void Start()
2 {
3     userInfo.text = "Schauen Sie auf einen Tisch";
4     StartCoroutine(DelayedStart());
5 }

```

Listing 4.4: Beginn des Inventory Placement Controllers

Die **Start()** Funktion wird zu Beginn des Levels für das "Knapsack-Problem" aufgerufen, wie im Codeausschnitt 4.4 gezeigt. Diese Methode setzt den Text des *userInfo* TextMeshes um den Benutzer den Ablauf anhand einer Anweisung zu erleichtern. Zusätzlich wird die Coroutine **DelayedStart()** initiiert, die eine Verzögerung einleitet, bevor das eigentliche Skript fortgesetzt wird.

Die Verzögerung wird gezielt eingesetzt, um dem *ARPlaneManager* ausreichend Zeit zu geben, die Ebenen in der Umgebung des Benutzers zu scannen und zu markieren. Dieser Schritt ist entscheidend, um eine stabile Grundlage für die spätere präzise Platzierung des Inventars in der virtuellen Umgebung zu schaffen.

Die Verwendung der Coroutine ermöglicht eine zeitgesteuerte Ausführung des Codes, wodurch die Umgebungserfassung des *ARPlaneManager* abgeschlossen werden kann, bevor

das Platzierungsskript aktiviert wird. Diese sorgfältig abgestimmte Startsequenz gewährleistet eine zuverlässige Erfassung der ARPlanes und legt somit den Grundstein für eine erfolgreiche Platzierung des Inventar-Objekts.

```

1 private IEnumerator DelayedStart()
2 {
3     yield return new WaitForSeconds(3.0f);
4     canStartScript = true;
5 }
```

Listing 4.5: Verzögelter Start

Die Funktion **DelayedStart()** (siehe Codeabschnitt 4.5) wird im Kontext der **Start()** Funktion aufgerufen, um eine initialisierte Verzögerung von 3 Sekunden zu gewährleisten, bevor das Skript seine Ausführung fortsetzt. Diese gezielte Verzögerung ist von entscheidender Bedeutung, um dem *ARPlaneManager* genügend Zeit zu geben, um die Umgebung ungestört zu scannen. Durch die Aktivierung der Variable *canStartScript* wird der Zeitpunkt markiert, ab dem die Methode **Update()** ihre Ausführung fortsetzen kann.

4.5.4.3 Frame-Aktualisierung zur Identifikation des gewünschten AR-Planes

Angesichts der potenziell vielfältigen Umgebung von gescannten und markierten ARPlanes ist eine kontinuierliche Aktualisierung erforderlich, um sicherzustellen, dass das gewünschte ARPlane stets identifiziert wird. Dieser Prozess wird durch die Anwendung der **Update()** Funktion realisiert. Diese Methode überwacht und steuert den Auswahlvorgang des ARPlanes, um sicherzustellen, dass das korrekte ARPlane ausgewählt wird, selbst wenn der Benutzer seinen Blick auf ein andere ARPlane lenkt. Die dazugehörige Funktion sieht dementsprechend folgendermaßen aus:

```

1 void Update()
2 {
3     if (!objectPlaced && canStartScript)
4     {
5         if (IsPointerOverPlane())
6         {
7             ARPlane currentPlane = GetCurrentPlaneUnderGaze();
8             if (currentPlane != null)
9             {
10                 if (selectedDeskPlane == null || selectedDeskPlane != currentPlane)
11                 {
12                     selectedDeskPlane = currentPlane;
13                     lookStartTime = Time.time;
14                 }
15                 float timeLookedAtPlane = Time.time - lookStartTime;
16                 userInfo.text = ((int)requiredLookTime - (int)timeLookedAtPlane).
17                 ToString();
18                 if (timeLookedAtPlane >= requiredLookTime)
19                 {
20                     PlaceObjectOnDesk(selectedDeskPlane);
21                     objectPlaced = true;
22                     userInfo.text = "";
23                 }
24             }
25         }
26         else
27         {
28             selectedDeskPlane = null;
29             userInfo.text = "Schauen Sie auf einen Tisch";
30         }
31     }
32 }
```

```

28         }
29     }
30     else
31     {
32         selectedDeskPlane = null;
33         userInfo.text = "Schauen Sie auf einen Tisch";
34     }
35 }
36 }
```

Listing 4.6: Update Funktion

Diese **Update()** Funktion wird jeden Frame der Applikation aufgerufen, um den Status des Objekts stets aktuell zu halten. Wenn die zwei Bedingungen *!objectPlaced* und *canStartScript* in der *Haupt if-Bedingung* den boolschen Wert *true* liefern kann das die Funktion weiter ablaufen, um das Inventar-Objekt zu platzieren.

Zunächst wird in der nächsten *if*-Bedingung überprüft ob der Benutzer aktuell auf ein *ARPlane* blickt. Um dies zu Überprüfen wird hierzu die **IsPointerOverPlane()** aufgerufen. Der dazugehörige Code dieser Funktion:

```

1 private bool IsPointerOverPlane()
2 {
3     Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
4     RaycastHit hit;
5     if (Physics.Raycast(ray, out hit))
6     {
7         ARPlane plane = hit.collider.GetComponent<ARPlane>();
8         return (plane != null);
9     }
10    return false;
11 }
```

Listing 4.7: Überprüfen ob Benutzer auf ARPlane blickt

Die Funktion *IsPointerOverPlane()* wird verwendet, um die Überprüfung der Überlagerung des Blicks des Benutzers über einem *ARPlane* zu ermöglichen.

Zunächst wird ein *Ray* generiert, der ausgehend vom *Hauptkameraobjekt* zur aktuellen Position des Blicks auf dem Bildschirm erstellt wird. Dieser Vorgang wird mithilfe der Methode *Camera.main.ScreenPointToRay(Input.mousePosition)* realisiert.

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 Vektoren definiert wird. Diese Richtung kann durch verschiedene Methoden festgelegt werden, abhängig von der Anwendung, in der der Ray verwendet wird. Im 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 Beleuchtungsberechnung 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*.

Im nächsten Schritt wird ein *RaycastHit*-Objekt initialisiert, das dazu dient, relevante Informationen über potenzielle Kollisionen zwischen dem generierten Ray und den Objekten in der Szene zu erfassen.

Anschließend wird der Ray in die Szene "geschossen", wobei währenddessen geprüft wird, ob eine Kollision mit einem Objekt stattfindet. Sollte eine Kollision detektiert werden, werden sämtliche Details über diese Kollision in das *RaycastHit*-Objekt übertragen.

Im weiteren Verlauf wird überprüft, ob das kollidierte Objekt eine spezifische Art von Objekt repräsentiert, nämlich ein *ARPlane*. Diese Überprüfung erfolgt durch den Zugriff auf die *ARPlane*-Komponente des kollidierten Objekts mittels der Methode *hit.collider.GetComponent<ARPlane>()*. Wenn diese Komponente vorhanden ist, signalisiert die Rückgabe des Wertes *true*, dass der Benutzer zum gegenwärtigen Zeitpunkt auf ein *ARPlane* blickt.

Falls jedoch keine Kollision mit einem *ARPlane* festgestellt wird oder das kollidierte Objekt keine *ARPlane*-Komponente aufweist, wird *false* zurückgegeben. Dies deutet darauf hin, dass der Benutzer zu diesem Zeitpunkt keinen Blick auf ein *ARPlane* richtet.

Wenn diese Funktion den boolschen Wert *true* zurückliefert wird anschließend ein neues *ARPlane* definiert. Dieses *ARPlane* (*currentPlane*) wird durch den Aufruf der Funktion **GetCurrentPlaneUnderGaze()** initialisiert. Diese Funktion sieht folgendermaßen aus:

```

1 private ARPlane GetCurrentPlaneUnderGaze()
2 {
3     Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
4     RaycastHit hit;
5     if (Physics.Raycast(ray, out hit))
6     {
7         ARPlane plane = hit.collider.GetComponent<ARPlane>();
8         return plane;
9     }
10    return null;
11 }
```

Listing 4.8: Gewolltes *ARPlane* ermitteln

Die Funktion **GetCurrentPlaneUnderGaze()** operiert auf ähnliche Weise wie die **IsPointerOverPlane()** Funktion, jedoch mit dem entscheidenden Unterschied, dass sie ein *ARPlane*-Objekt als Rückgabewert definiert. Dieser Unterschied bezweckt, dass die Funktion das tatsächliche *ARPlane*-Objekt zurückgibt, auf das der Benutzer seinen Blick gerichtet hat.

Nach Abschluss der beiden Funktionen **GetCurrentPlaneUnderGaze()** und **IsPointerOverPlane()** wird überprüft, ob eine aktuelle Ebene gefunden wurde (*currentPlane != null*). Wenn dies der Fall ist, wird weiterhin geprüft, ob es sich um ein neues *ARPlane* handelt oder ob es sich um dieselbe Ebene handelt, die bereits ausgewählt wurde (*selectedDeskPlane == null || selectedDeskPlane != currentPlane*). Wenn diese *if*-Bedingung den boolschen Wert *true* ergibt, wird das momentan selektierte *ARPlane* der Variable *selectedDeskPlane* zugewiesen und der Timer *timer* wird gestartet, um aufzuzeichnen, wie lange der Benutzer auf dieses *ARPlane* blickt. Um dann abschließend noch zu visualisieren, wie lange er noch auf dieses *ARPlane* blicken muss, wird der Text des *userInfo* TextMeshes auf die aktuelle Blickzeit geändert.

Abschließend wird überprüft, ob die Zeit, die der Benutzer zu dem aktuellen Zeitpunkt auf dieses spezifische *ARPlane* geblickt hat, größer oder gleich der erforderlichen Blick-

zeit ist ($timeLookedAtPlane \geq requiredLookTime$). Falls diese Bedingung erfüllt ist, wird die Funktion **PlaceObjectOnDesk()** mit dem ausgewählten *ARPlane* aufgerufen, um das Inventar-Objekt zu platzieren. Zusätzlich wird der Status der booleschen Variable *objectPlaced* auf *true* gesetzt, um zu signalisieren, dass das Inventar-Objekt platziert wurde und der Inhalt des *userInfo* TextMeshes wird geleert. Der Code für das Platzieren dieses Objekts ist wie folgt:

```

1 private void PlaceObjectOnDesk(ARPlane deskPlane)
2 {
3     qrCodesManager.SetActive(true);
4     objectPosition = deskPlane.center + Vector3.up * heightOffset;
5     Quaternion objectRotation = Quaternion.Euler(-90f, 0f, 0f);
6     GameObject instantiatedObject = Instantiate(inventoryObject, objectPosition,
7         objectRotation);
8     instantiatedObject.transform.localScale = new Vector3(20f, 20f, 20f);
9     Vector3 infoObjectPosition = objectPosition - Vector3.forward * 4.415f + Vector3.
10    right * 0.4f;
11    infoObject.transform.position = infoObjectPosition;
12    infoObject.SetActive(true);
13    inventoryController.SetInventoryObject(instantiatedObject);
14    inventoryController.gameObject.SetActive(true);
15    planeManager.planePrefab.SetActive(false);
16    gameObject.SetActive(false);
17 }
```

Listing 4.9: Inventar-Objekt platzieren

Diese Funktion trägt eine tragende Rolle für den weiteren Verlauf der Applikation. Zu Beginn wird der *QRCodeManager* aktiviert um das erkennen und scannen von *QRcodes* zu ermöglichen.

Als nächstes wird die Mitte des *ARPlanes* berechnet um das Inventar-Objekt dementsprechend passend auf diesem zu platzieren. Damit das Inventar-Objekt richtig rotiert ist wird dieser anschließend um *90 Grad* auf der X-Achse rotiert, dass es flach liegt. Der nächste Schritt (*GameObject instantiatedObject = Instantiate(inventoryObject, objectPosition, objectRotation)*) ist dafür verantwortlich, dass das Objekt für den Benutzer sichtbar wird. Diese Funktion instanziert das Inventar-Objekt an der definierten Position (*objectPosition*) und mit der definierten Rotation (*objectRotation*). Zusätzlich wird es dann noch runterskaliert. Dieselben Aktionen werden ebenfalls mit dem *infoObject* vorgenommen mit dem einzigen Unterschied, dass die Position (*infoObjectPosition*) anders berechnet wird, dass dieses neben dem Inventar-Objekt liegt. Diese Object wird dann mittels der Funktion **.SetActive(true)** sichtbar.

Anschließend wird dem *inventoryController* dieses Inventar-Objekt übergeben, damit dieser mit derselben Instanz arbeitet wie der *inventoryPlacementController*. Dieser wird dann ebenfalls mit **.SetActive(true)** aktiviert. Das Ende des *inventoryControllers* stellt die Zeile *gameObject.SetActive(false)* dar. Diese Zeile bewirkt, dass das *inventoryPlacementController*-GameObject selbst deaktiviert wird.

Wenn kein aktuelles *ARPlane* gefunden wird oder, wenn der Blick des Benutzers nicht auf einem *ARPlane* liegt, wird in den beiden *else*-Zweigen das aktuelle ausgewählte *ARPlane* auf null gesetzt, um zuvor gespeicherte Auswahlen und davon möglich auftretende Konflikte zu vermeiden. Zusätzlich wird der Inhalt des *userInfo* TextMeshes wieder geändert, um dem Benutzer zu signalisieren, dass er wieder auf ein *ARPlane* blicken muss.

Nachdem das *InventoryPlacementController* Skript ordnungsgemäß deaktiviert und beendet wurde, wird dem Benutzer anschließend (Wie in folgender Abbildung 4.12 dargestellt) Ansicht präsentiert.

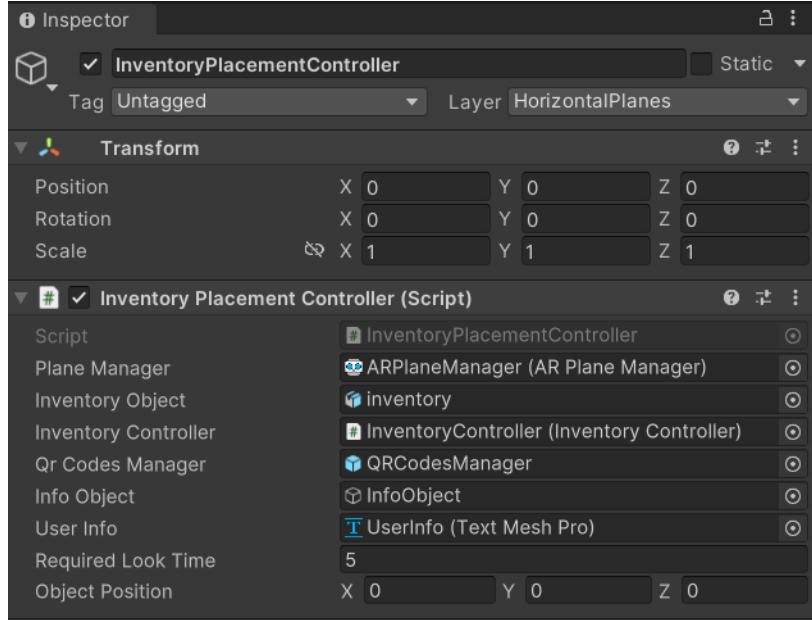


Abbildung 4.12: Sicht des Benutzers nach Platzierung des Inventars

4.5.5 Inventory Controller Game Objekt

Das *Inventory Controller* Game Objekt nimmt eine essenzielle Rolle bei der Überwachung und Verwaltung des Inventars in der Augmented Reality (AR)-Anwendung ein. Durch das angehängte Skript *InventoryController.cs* wird die **InventoryController**-Klasse implementiert. Diese Klasse ist dafür zuständig, neue Gegenstände im Inventar zu erkennen, den verfügbaren Platz zu überprüfen, die Zellenposition anhand der QR-Code-Koordinaten zu berechnen und die Positionen dieser Gegenstände in einem *2D Array* zu speichern.

Der Zugriff auf die Begrenzungen (*Bounds*) des Inventar-Modells ermöglicht eine kontinuierliche Überwachung, ob der Benutzer ein neues Element in das Inventar gelegt hat. Dieser Ansatz gewährleistet eine präzise Kontrolle über die Platzierung von Gegenständen im Inventar.

Das Skript *InventoryController.cs* interagiert aktiv mit den AR-Elementen, insbesondere den QR-Codes, um den Platzbedarf der Gegenstände zu überwachen und ihre Positionen im Inventar festzulegen. Zusätzlich kommuniziert der *InventoryController* auch mit dem Game Objekt *KnapsackSolver*, indem er das gespeicherte *2D Array* weitergibt, um entsprechende Berechnungen durchzuführen zu können.

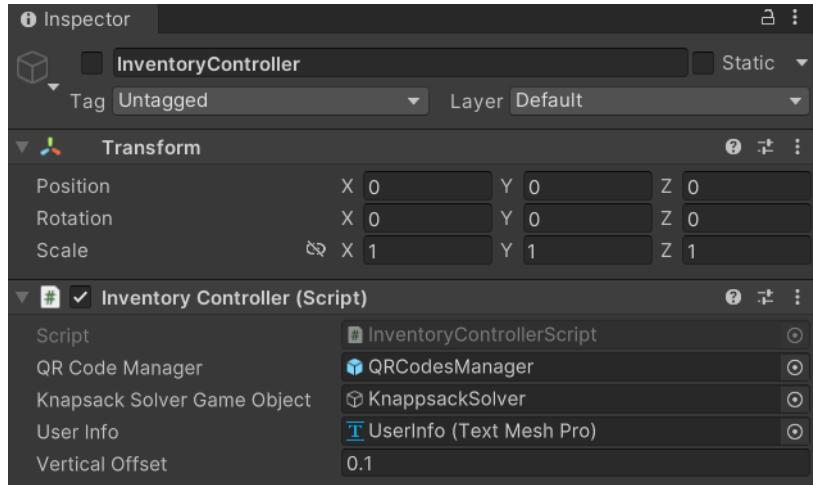


Abbildung 4.13: InventoryController Objekt im Editor

Abbildung 4.13 zeigt das *InventoryController*-Objekt im Unity Editor. Hier können verschiedene Einstellungen vorgenommen werden, darunter die Ebene, in der dieses Objekt liegt, die Koordinaten im Unity Editor und die angehängte Komponente *InventoryController.cs*. Zudem sind vordefinierte Werte für bestimmte Variablen sichtbar. Diese Variablen umfassen:

- **QRCodesManager:** Referenz auf das *QRCodesManager*-Game-Objekt aus der Level 2 Szene.
- **Knapsack Solver Game Objekt:** Referenz auf das *KnapsackAlgo*-Game-Objekt aus der Level 2 Szene.
- **User Info:** Referenz auf das TextMesh in der *Main Camera*, dass zur Anzeige von Anweisungen und Tipps dient.
- **Vertical Offset:** Float-Wert, um den die Begrenzungen (*Bounds*) des Inventory-Objekts auf der X-Achse erweitert werden.

4.5.5.1 InventoryController Klassenvariablen

```

1 public GameObject QRCodeManager;
2 public GameObject knapsackSolverGameObject;
3 public TextMeshPro userInfo;
4
5 private SortedDictionary<System.Guid, GameObject> activeQRObjects;
6
7 private GameObject inventoryObject;
8 public float verticalOffset = 0.1f;
9 private Bounds inventoryBounds;
10 private int numRows = 3;
11 private int numColumns = 3;
12 private int[,] idGrid;
13 private KnapsackSolver knapsackSolver;
14 private int cap;
15 private int currWeight = 0;
16 private string message;
17 private HashSet<int> processedItems;

```

Listing 4.10: Klassenvariablen der InventoryController Klasse

Die gezeigten Klassenvariablen im Codeabschnitt 4.10 sind Teil der *InventoryController* Klasse. Öffentliche (*public*) Variablen repräsentieren Objekte und Werte, die im Unity Editor festgelegt und übergeben werden oder von anderen Klassen für die Funktionalität benötigt werden. Dies ermöglicht einen direkten Zugriff auf diese Objekte in der eigenen oder einer anderen Klasse. Die Float-Variable *verticalOffset* spielt hier eine wichtige Rolle für die Erweiterung der Begrenzungen (*Bounds*) des Inventar-Modells und das TextMesh-Pro *userInfo* für das anzeigen von Anweisung und Tips für den Benutzer.

Die privaten (*private*) Variablen dienen der lokalen Speicherung von Werten, die von keiner anderen Klasse benötigt werden.

4.5.5.2 Start des InventoryControllers

Die **Start()** Funktion spielt eine Schlüsselrolle beim Initialisieren der erforderlichen Objekte und Skripte für den erfolgreichen Start des *InventoryControllers*. Diese Funktion wird zu Beginn des *InventoryControllers* aufgerufen und ist verantwortlich für die Deklaration und Einrichtung der notwendigen Komponenten. Entsprechender Code dieser Funktion:

```

1 void Start()
2 {
3     userInfo.text = "Platzieren Sie nun Gegenstände im Inventar";
4     activeQRObjects = QRCodeManager.GetComponent<QRCodeManager>().
5         qrCodesObjectsList;
6     knapsackSolver = knapsackSolver.GetComponent<KnapsackSolver>();
7     cap = knapsackSolver.capacity;
8     processedItems = new HashSet<int>();
9     UpdateInventoryBounds();
10    InitializeIDGrid();
}

```

Listing 4.11: Start Funktion des InventoryControllers

Die Objekte, die in dieser Funktion initialisiert werden, umfassen *activeQRObjects*, die für das Erkennen neuer Objekte im Inventar entscheidend sind. Zur Weitergabe des aktualisierten Inventars im späteren Verlauf des *InventoryController* wird das *KnapsackSolver*-Objekt benötigt. Zu Beginn dieser Funktion wird der Inhalt von dem *userInfo* TextMesh auf einen neuen Inhalt geändert um dem Benutzer einen Tip / eine Anweisung zu geben was, zu tun ist. Des Weiteren werden die Kapazität (*cap*) des *KnapsackSolver*-Objekts gespeichert und ein *processedItems-HashSet* erstellt, um zu verfolgen, welche Objekte im Inventar bereits verarbeitet wurden. Am Ende der **Start()** Funktion werden die beiden Funktionen **UpdateInventoryBounds()** und **InitializeGrid()** aufgerufen.

Es ist wichtig zu betonen, dass diese Funktion unmittelbar auf den vorherigen Codeabschnitt 4.10 verweist, wo die relevanten Variablen der *InventoryController*-Klasse definiert sind. Dies stellt sicher, dass die Funktionalitäten, die in der **Start()** Funktion verwendet werden, korrekt initialisiert und verwendet werden können.

4.5.5.3 Bounds des Inventars aktualisieren

Um sicherzustellen, dass ein Item innerhalb der Grenzen des Inventars platziert werden kann, werden in der **UpdateInventoryBounds()** Funktion einige Änderungen an dem Inventar-Objekt vorgenommen um dies zu realisieren.

```

1 private void UpdateInventoryBounds()
2 {
3     if (inventoryObject != null)
4     {
}

```

```

5     Bounds localBounds = GetBounds(inventoryObject);
6     ExtendBounds(ref localBounds, verticalOffset);
7     inventoryBounds = localBounds;
8 }
9 }
```

Listing 4.12: Funktion um Inventar Bounds zu erweitern

Zunächst werden die aktuellen Begrenzungen des Inventar-Objekts gespeichert, und dann werden sie mittels der **ExtendBounds()** Funktion erweitert. Schließlich werden diese aktualisierten Begrenzungen als neue Begrenzungen gespeichert.

Diese Funktion wird in einem größeren Kontext verwendet, insbesondere in der **Start()** Funktion des *InventoryControllers*, wie in Codeabschnitt 4.11 zu sehen ist. Dies stellt sicher, dass die Begrenzungen des Inventars korrekt aktualisiert werden, bevor das Inventar für den weiteren Gebrauch vorbereitet wird.

4.5.5.4 Bounds des Inventars ermitteln

Um erfolgreich auf die Begrenzungen des Inventar-Objekts zugreifen zu können, wird die Funktion **GetBounds()** aufgerufen.

Diese Funktion ist entscheidend, da sie den *Renderer* des Inventar-Modells verwendet, um die Begrenzungen zu ermitteln. Der Renderer ist verantwortlich für die Darstellung des Modells im Spiel. Wenn der Renderer vorhanden ist (*renderer != null*), gibt die Funktion die Begrenzungen des Modells zurück. Andernfalls wird eine neue Bounds-Instanz erstellt, die die Position des Objekts und eine Einheitsgröße hat.

```

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

Listing 4.13: Funktion um Bounds zu ermitteln

Die Funktion **GetBounds()**, wie in Codeabschnitt 4.13 dargestellt, wird von Codeabschnitt 4.12 aufgerufen, um die aktuellen Begrenzungen des Inventars zu erhalten. Diese Begrenzungen werden dann verwendet, um die Bounds des Inventars zu erweitern.

4.5.5.5 Inventory Bounds erweitern

Im weiteren Verlauf der **UpdateInventoryBounds()** Funktion, also nach dem Abschluss des Aufrufs der **GetBounds()** Funktion, wird anschließend die Funktion **ExtendBounds()** aufgerufen (siehe Codeabschnitt 4.14), die sich darum kümmert, dass die *Bounds* tatsächlich erweitert werden. Der zugehörige Code dieser Funktion:

```

1 private void ExtendBounds(ref Bounds bounds, float offset)
2 {
3     bounds.center = new Vector3(bounds.center.x, bounds.center.y + offset / 2, bounds.
4         center.z);
5     bounds.extents = new Vector3(bounds.extents.x, bounds.extents.y + offset / 2,
6         bounds.extents.z);
7 }
```

Listing 4.14: Funktion um Bounds zu erweitern

Diese Funktion nimmt zwei Parameter an. Der erste Parameter ist eine *Referenz* auf die *Bounds* des Inventar-Objekts. Die Verwendung der Referenz ermöglicht es, die Änderungen direkt an den ursprünglichen Bounds des Inventar-Objekts vorzunehmen, was effizienter ist als die Rückgabe einer neuen Bounds-Instanz. Der zweite Parameter ist das *Offset*, um das die *Bounds* des Objekts erweitert werden sollen.

Der Code der Funktion erweitert die Grenzen (*Bounds*) des Inventar-Objekts um das angegebene Offset. Dies geschieht, indem der Mittelpunkt der Bounds und ihre Ausdehnung entsprechend modifiziert werden. Die Höhe der Bounds wird um die Hälfte des Offset-Werts erhöht, um die Bounds sowohl nach oben als auch nach unten zu erweitern, während die anderen Dimensionen unverändert bleiben.

Nach Abschluss der **UpdateInventoryBounds()** (siehe Codeabschnitt 4.12) sehen die aktualisierten Bounds im Hintergrund dann wie auf Abbildung 4.14 zu sehen aus.

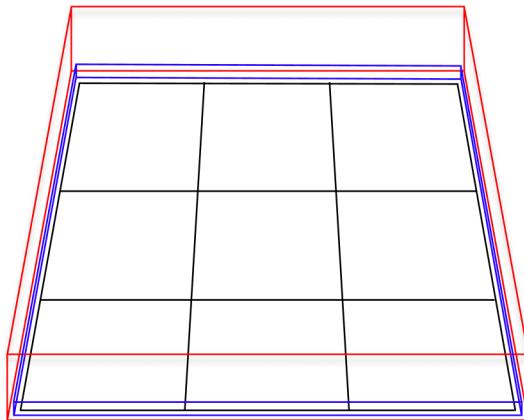


Abbildung 4.14: Erweiterte Bounds des Inventar Modells

Diese Abbildung zeigt eine schematische Darstellung des zweidimensionalen Inventars. Die blaue Umrandung repräsentiert die *Bounds* des Inventar-Objekts vor der Ausführung der Funktion **UpdateInventoryBounds()**. Nach ausführung der Funktion und erweiterung der *Bounds* ist nun anhand der blauen Umrandung deutlich, dass insbesondere die *Bounds* entlang der X-Achse erweitert wurden.

Die Erweiterung der Bounds ist von entscheidender Bedeutung, da die standard *Bounds* des Inventar-Objekts die Höhe des Objekts, auf dem der QRCode befestigt ist, nicht berücksichtigen. Dies könnte dazu führen, dass ein Gegenstand, der sich tatsächlich innerhalb der Bounds befindet, als außerhalb betrachtet wird.

Diese Anpassungen sind wichtig, um sicherzustellen, dass die Begrenzungen des Inventars korrekt erfasst werden und somit eine präzise Verortung von QR-Codes innerhalb der Augmented-Reality-Anwendung gewährleistet ist.

4.5.5.6 ID Grid Initialisierung

Als Abschluss der **Start()** Funktion wird die **InitializeIDGrid()** Funktion die, im folgenden Codeabschnitt 4.15 dargestellt ist, aufgerufen.

```

1 private void InitializeIDGrid()
2 {
3     idGrid = new int[numRows, numColumns];
4 }
```

Listing 4.15: Initialisierung des idGrids

Diese Funktion initialisiert das *idGrid*, das ein zweidimensionales Array darstellt und dazu dient, die Positionen der Gegenstände im Inventar zu speichern. Die Größe des Arrays wird durch die Variablen *numRows* und *numColumns* festgelegt, die zuvor in der Klasse definiert wurden.

4.5.5.7 Update-Funktion

Um stets den aktuellen Zustand des Inventars und der darin enthaltenen *Items* zu erhalten, wird hier die **Update()** Funktion verwendet. Diese Funktion wird jeden Frame neu aufgerufen. Der zugehörige Code dazu:

```

1 void Update()
2 {
3     UpdateGrid();
4 }
```

Listing 4.16: Initialisierung des idGrids

4.5.5.8 Neue oder entfernte Items erkennen

Das Kernstück des *InventoryController* stellt die **UpdateGrid()** dar. Diese Funktion ist hauptverantwortlich dafür, dass neue *Items* innerhalb des Inventars erkannt werden, *Items* die aus dem Inventar entfernt wurden dementsprechend entfernt werden und auch, dass bei jedem neuem oder entferntem *Item* das Inventar neu berechnet wird und darauffolgend die drei *TextMeshes* innerhalb des *infoObjects* aktualisiert werden, um dem Benutzer immer die aktuellen Inventarwerte und potentielle Fehlermeldungen anzuzeigen. Der Code zu dieser Funktion:

```

1 void UpdateGrid()
2 {
3     lock (activeQR0bjects)
4     {
5         foreach (var item in activeQR0bjects.Values)
6         {
7             QRCode qRCode = item.GetComponent<QRCode>();
8             Vector3 worldPosition = item.transform.TransformPoint(qRCode.item.qrData.
position);
9             if (item != null && inventoryBounds.Contains(worldPosition))
10            {
11                int itemId = qRCode.item.qrData.id;
12                if (!processedItems.Contains(itemId))
13                {
14                    if (currWeight + qRCode.item.qrData.weight <= cap)
15                    {
16                        userInfo.text = "";
17                        processedItems.Add(itemId);
18                        message = "";
19                        Vector2 startGridPosition = CalculateGridPosition(
worldPosition);
20                        idGrid[(int)startGridPosition.x, (int)startGridPosition.y] =
itemId;
21                        knapsackSolver?.UpdateInfoMesh(message);
22                        currWeight += qRCode.item.qrData.weight;
23                        EventManager.GridUpdate(idGrid);
24                    }
25                else
26                {
27                    message = "Item hat zu viel Gewicht!";

```

```

28             knapsackSolver?.UpdateInfoMesh(message);
29         }
30     }
31     else if (!inventoryBounds.Contains(worldPosition) && processedItems.
32 Contains(qRCode.item.qrData.id) && ContainsId(qRCode.item.qrData.id))
33     {
34         int itemId = qRCode.item.qrData.id;
35         processedItems.Remove(itemId);
36         RemoveItem(itemId);
37         currWeight -= qRCode.item.qrData.weight;
38         EventManager.GridUpdate(idGrid);
39     }
40 }
41 }
42 }
```

Listing 4.17: Neue / Entfernte Items erkennen

Zu Beginn der Funktion wird eine Sperre mittels einer *lock*-Anweisung auf das *Dictionary* der aktiven QR-Objekte (*activeQRObjects*) angewendet. Diese Maßnahme zielt darauf ab, die Datenkonsistenz zu gewährleisten, indem verhindert wird, dass mehrere *Threads* gleichzeitig auf dieses *Dictionary* zugreifen und dabei gleichzeitig Änderungen vornehmen. Ohne diese Sperre könnte es zu einem sogenannten Wettlaufzustand (*race condition*) kommen, bei dem mehrere Threads versuchen, gleichzeitig auf dasselbe Datenobjekt zuzugreifen und es möglicherweise in einen inkonsistenten Zustand bringen könnten.

Definition einer race condition

Ein Wettlaufzustand kann verschiedene Probleme verursachen, darunter inkonsistente oder fehlerhafte Datenoperationen. Als Beispiel könnten Threads gleichzeitig versuchen, ein Element aus dem *Dictionary* zu entfernen oder hinzuzufügen, was zu inkonsistenten Datenzuständen führen könnte. Durch die Anwendung der *lock*-Anweisung wird sichergestellt, dass jeweils nur ein Thread gleichzeitig auf das *Dictionary* zugreifen kann, wodurch potenzielle Konflikte vermieden werden und die Datenintegrität gewährleistet ist.

Nachdem die Sperre angewendet wurde, durchläuft die Funktion eine *foreach*-Schleife, um jedes Objekt im Dictionary *activeQRObjects* zu durchlaufen. Für jedes dieser Objekte wird die zugehörige *QRCode*-Komponente mittels der *GetComponent<QRCode>()* Funktion abgerufen, um auf die spezifischen Informationen des QR-Codes zuzugreifen. Die *Weltposition* des Objekts wird anschließend berechnet, indem die *lokale Position* des Objekts in seine Weltposition umgerechnet wird.

Definition Lokale und Weltposition

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

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

Die Berechnung der Weltposition eines Objekts erfolgt, indem seine lokale Position relativ zu seinem Elternobjekt oder zum Koordinatenursprung in die Szene umgerechnet

wird. Dies ermöglicht es, die tatsächliche Position des Objekts in der Welt zu bestimmen und mit anderen Objekten oder Koordinaten in der Szene zu interagieren.

Angenommen, gegeben ist ein Game Objekt namens *childObject*, dessen lokale Position relativ zu seinem Elternobjekt oder zum Koordinatenursprung (0, 0, 0) wie folgt definiert ist:

$$\text{localPosition} = (x_{\text{local}}, y_{\text{local}}, z_{\text{local}})$$

Die Weltposition des Elternobjekts (oder des Ursprungs) ist gegeben durch:

$$\text{parentWorldPosition} = (x_{\text{parent}}, y_{\text{parent}}, z_{\text{parent}})$$

Um die Weltposition des *childObjects* zu berechnen, wird stets die *lokale Position* des Objekts zu der *Welt Position* seines zugehörigen Elternobjekts addiert.

$$\text{worldPosition} = \text{parentWorldPosition} + \text{localPosition}$$

Dies ergibt die Weltposition des *childObject* im globalen Koordinatensystem.

Beispiel: Angenommen, das Elternobjekt liegt im globalen Koordinatensystem bei (2, 0, 0) und die lokale Position des *childObject* ist als (0, 1, 0) definiert. Dann lautet die Berechnung wie folgt:

$$\text{worldPosition} = (2, 0, 0) + (0, 1, 0) = (2, 1, 0)$$

Dies bedeutet, dass die Weltposition des *childObject* im globalen Koordinatensystem (2, 1, 0) ist.

Der folgende Schritt in der Funktion **UpdateGrid()** beinhaltet die Überprüfung der Gültigkeit des QR-Objekts (*item != null*) sowie die Feststellung, ob die berechnete Weltposition innerhalb der definierten Begrenzungen des Inventars liegen (*inventoryBounds.Contains(worldPosition)*). Im Falle der Erfüllung dieser Bedingungen (*true*), wird das QR-Objekt weiteren Analysen unterzogen.

Zunächst wird von dem aktuellen Item die *ID* mittels *qRCode.item.qrData.id* gespeichert, um im weiteren Verlauf mit dieser *ID* weiterarbeiten zu können. Nach diesem Schritt wird überprüft, ob die *ID* des Items bereits in der Liste der verarbeiteten Items *processedItems* enthalten ist. Wenn diese *ID* neu ist und noch nicht verarbeitet wurde, wird weiter überprüft, ob das Hinzufügen des Items die Maximale Kapazität (*cap*) des Inventars überschritten wird.

Im Falle, dass das Gewicht des Items zusammen mit dem aktuellen Gesamtgewicht (*currWeight*) die maximale Kapazität (*cap*) nicht überschreitet, wird der Text des *infoMesh* geleert, die ID des Items zur Liste der verarbeiteten Items (*processedItems*) hinzugefügt, um es als verarbeitet zu markieren. Des Weiteren wird die Position dieses Items innerhalb des Inventars mithilfe der Funktion **CalculateGridPosition()** berechnet und anschließend dem *idGrid* hinzugefügt. Zuletzt werden das *InfoMesh* aktualisiert sowie das aktuelle Gewicht (*currWeight*) und das Event **GridUpdate** des *EventManagers* mit dem aktualisierten *idGrid* ausgelöst. Der Code, um die Position des Items innerhalb des Inventars zu berechnen:

```

1 private Vector2 CalculateGridPosition(Vector3 objectPosition)
2 {
3     float cellWidth = inventoryBounds.size.x / numColumns;
4     float cellHeight = inventoryBounds.size.z / numRows;
5     int col = Mathf.FloorToInt((objectPosition.x - inventoryBounds.min.x) / cellWidth)
6     ;
7     int row = Mathf.FloorToInt((inventoryBounds.max.z - objectPosition.z) / cellHeight
8 );
9     return new Vector2(row, col);
10 }
```

Listing 4.18: Position des Items berechnen

Die Funktion beginnt damit, die Länge einer einzelnen Zelle anhand der *Bounds* in der X- und Y-Achse zu berechnen. Die Variable *cellWidth* repräsentiert die Breite jeder Zelle, während *cellHeight* die Höhe jeder Zelle im Raster des Inventars darstellt.

Anschließend werden die 2D-Koordinaten der Position des übergebenen Objekts im Raster des Inventars berechnet. Hierzu wird die X-Koordinate des Objekts relativ zur minimalen X-Grenze der Inventar-Bounds genommen und durch die Breite einer Zelle (*cellWidth*) geteilt. Der resultierende Wert wird auf die nächstgelegene ganze Zahl abgerundet (*Mathf.FloorToInt*) und repräsentiert somit die Spalte (*col*) im Raster.

Ebenso wird die Z-Koordinate des Objekts relativ zur maximalen Z-Grenze der Inventar-Bounds herbeigezogen und durch die Höhe einer Zelle (*cellHeight*) geteilt. Auch dieser Wert wird auf die nächstgelegene ganze Zahl abgerundet und repräsentiert die Zeile (*row*) im Raster.

Abschließend werden die berechneten Zeilen- und Spaltenwerte als 2D-Vektor (*Vector2*) zurückgegeben. Dieser Vektor repräsentiert die Position des Objekts im Inventar-Raster.

Um diesen Ablauf des hinzufügen eines Gegenstandes aus der Sicht des Benutzers als auch wie dieser Prozess dann im Hintergrund abläuft, wird hierfür ein Beispiel herangezogen.

Die visuelle Darstellung für den Benutzer, nachdem dieser einen Gegenstand in das Inventar hinzugefügt hat ist in folgender Abbildung 4.15 zu sehen.

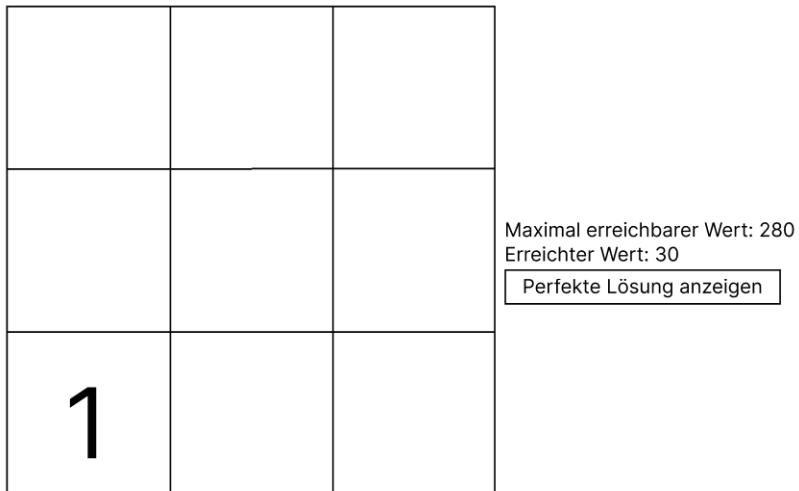


Abbildung 4.15: Item zu Inventar hinzugefügt

Auf dieser Abbildung ist zu sehen, dass in *Zelle: 7* das Item mit der *ID: 1*

hinzugefügt wurde.

Nachdem das Item im Inventar platziert wurde, sieht das *idGrid* folgendermaßen aus:

```
1 idGrid = [[0, 0, 0],  
2           [0, 0, 0],  
3           [1, 0, 0]];
```

Das HashSet *processedItems* sieht nach dem platzieren des Items dann folgendermaßen aus:

```
1 processedItems = [1];
```

In dem anderen Fall, dass das Gewicht des Items zusammen mit dem aktuellen Gesamtgewicht (*currWeight*) die maximale Kapazität (*cap*) überschreitet wird das TexMesh (*Info-Mesh*) mit der Nachricht, dass das Item zu schwer ist aktualisiert.

Um diesen Ablauf, in dem eine Vielzahl an Gegenständen in das Inventar hinzugefügt werden, wobei der zuletzt hinzugefügte Gegenstand zu schwer ist, wird hierfür erneut ein Beispiel herangezogen.

Der Zustand, indem der Benutzer mehrere Gegenstände zu dem Inventar hinzugefügt hat, wobei der zuletzt hinzugefügte Gegenstand zu schwer ist, ist in folgender Abbildung 4.16 zu sehen. Darauf folgend ist auch angegeben, wie dieser Ablauf im Hintergrund aussieht.

	9	
5		
1	3	

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

Abbildung 4.16: Item zu schwer für das Inventar

In Abbildung 4.16 sind vier verschiedene Gegenstände im Inventar enthalten. Diese haben folgende *IDs*: 1, 3, 5, 9. Die Reihenfolge, in der diese zum Inventar hinzugefügt wurden, lautet 1, 5, 3, 9. Das Item mit der *ID*: 9 wurde jedoch aufgrund seines zu hohen Gewichts nicht zu dem *idGrid* und *processedItems* hinzugefügt. Gleichzeitig wird eine Fehlermeldung angezeigt, um dem Benutzer zu signalisieren, dass der zuletzt hinzugefügte Gegenstand zu schwer für das aktuelle Inventar ist. Als Konsequenz bleiben *idGrid* und *processedItems*

unverändert und enthalten daher nicht die *ID*: 9. Im Hintergrund sieht daher das *idGrid* dementsprechend folgendermaßen aus:

```
1 idGrid = [[0, 0, 0],
2           [5, 0, 0],
3           [1, 3, 0]];
```

Das HashSet *processedItems* sieht nach dem Platzieren des Items dann so aus:

```
1 processedItems = [1, 5, 3];
```

Sobald der Benutzer dieses Item wieder aus dem Inventar entfernt, wird die Fehlermeldung nicht mehr angezeigt, und es kann ein passendes Item hinzugefügt werden.

Wenn sich das Item außerhalb der definierten Bounds befindet, jedoch zuvor im Inventar platziert wurde und sowohl in der Liste der verarbeiteten Items (*processedItems*) als auch im *idGrid* enthalten ist, deutet dies darauf hin, dass der Benutzer das Item aus dem Inventar entfernt hat. In diesem Fall wird die ID des Items aus der Liste der verarbeiteten Items (*processedItems*) sowie aus dem *idGrid* entfernt. Zusätzlich wird das aktuelle Gesamtgewicht (*currWeight*) aktualisiert und das Event **GridUpdate** des EventManagers mit dem aktualisierten idGrid ausgelöst. Wichtig anzumerken ist hier noch, dass in diesem *else-if*-Zweig die zwei Funktionen **RemoveItem()** und **ContainsId()** mit der aktuellen Item *ID* aufgerufen werden. Der zugehörige Code der beiden Funktionen:

```
1 private void RemoveItem(int id)
2 {
3     for (int i = 0; i < numRows; i++)
4     {
5         for (int j = 0; j < numColumns; j++)
6         {
7             if (idGrid[i, j] == id)
8             {
9                 idGrid[i, j] = 0;
10                return;
11            }
12        }
13    }
14 }
```

Listing 4.19: Item aus Liste entfernen

```
1 private bool ContainsId(int id)
2 {
3     for (int i = 0; i < numRows; i++)
4     {
5         for (int j = 0; j < numColumns; j++)
6         {
7             if (idGrid[i, j] == id)
8             {
9                 return true;
10            }
11        }
12    }
13    return false;
14 }
```

Listing 4.20: Überprüfen ob ID in Liste enthalten ist

Beide Funktionen operieren im Wesentlichen auf ähnliche Weise. Sie durchlaufen die Liste der verarbeiteten Items mithilfe einer verschachtelten *for-Schleife* und überprüfen, ob die als Parameter übergebene ID enthalten ist. Wenn dies der Fall ist, wird die **RemoveItem()** Funktion die ID an dieser Stelle durch den Wert 0 ersetzen, der keinen Inhalt repräsentiert. Die **ContainsId** Funktion hingegen überprüft lediglich, ob die ID in der Liste enthalten ist, und gibt den boolschen Wert *true* zurück, falls dies zutrifft, andernfalls *false*.

Der Fall, indem der Benutzer einen Gegenstand aus dem Inventar entfernt, wird in folgendem Beispiel veranschaulicht und erklärt, wie dieser Ablauf im Hintergrund abläuft.

In folgender Abbildung 4.17 ist der aktuelle Zustand eines selbst zusammengestellten Inventars zu sehen. In diesem Inventar sind die beiden Gegenstände mit den *IDs* 5 und 3 enthalten

5		
	3	

Maximal erreichbarer Wert: 280
 Erreichter Wert: 95

Abbildung 4.17: Item aus Inventar entfernt

Im Hintergrund sieht das *idGrid* jedoch wie folgt aus:

```
1 idGrid = [[0, 0, 0],  
2                 [5, 0, 0],  
3                 [1, 3, 0]];
```

Das HashSet *processedItems* sieht wie folgt aus:

```
1 processedItems = [1, 5, 3];
```

Anhand der Abbildung 4.17 und den gespeicherten *IDs* in *processedItems* und *idGrid* ist jedoch zu erkennen, dass hier noch die ID 1 gespeichert ist. Dies bedeutet, dass das Item mit der ID 1 in einem früheren Zustand im Inventar platziert wurde, aber jetzt außerhalb der Grenzen des Inventars liegt. Folglich wird diese ID aus dem *idGrid* und *processedItems* entfernt, und das *currWeight* wird entsprechend angepasst. Nach dem entfernen dieser ID sehen das *idGrid* und das *processedItems* Array wie folgt aus:

```
1 idGrid = [[0, 0, 0],  
2                 [5, 0, 0],  
3                 [0, 3, 0]];
```

```
1 processedItems = [5, 3];
```

4.5.6 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 Kompaktheit, da er die Schnittstelle für die Kommunikation zwischen verschiedenen Klassen und Game-Objekten bereitstellt.

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 das zweite 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     // Level 1
4     public static event System.Action<int, string, string> OnMessageReceived;
5     public static void ReceiveMsg(int idx, string username, string message) =>
6         OnMessageReceived?.Invoke(idx, username, message);
7
8     public static event System.Action<int, string, string> OnMessageSend;
9     public static void SendMsg(int idx, string username, string message) =>
10        OnMessageSend?.Invoke(idx, username, message);
11
12     // Level 2
13     public static event System.Action<int[,]> OnGridUpdate;
14     public static void GridUpdate(int[,] grid) => OnGridUpdate?.Invoke(grid);
15 }
```

Das ist die definition der Ereignisse und Methoden, die verwendet werden, um die Ereignisse auszulösen. Um auf diese Ereignisse zu reagieren, können andere Teile des Codes sich auf diese Ereignisse registrieren, welches wie folgend aussieht:

```

1 //Dieser Code Abschnitt befindet sich in der Klasse: KnapsackSolver.cs
2
3 void Start()
4 {
5     items = new QRItem(0).items;
6     EventManager.OnGridUpdate += SetInventory;
7 }
8
9 public void SetInventory(int[,] newInventory)
10 {
11     inventory = newInventory;
12     CalculateKnapsack();
13 }
```

Nach laden der Scene registriert sich die *KnapsackSolver* Klasse auf das *OnGridUpdate* Event mit der *setInventory* Methode. Dies bedeutet, dass jedes mal wenn das *OnGridUpdate* Event ausgelöst wird, die *setInventory* Methode aufgerufen wird und der aktuelle Wert des neuen Inventars berechnet wird.

```
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.

4.5.7 Knapsack Solver Game Objekt

Das *Knapsack Solver* Game Objekt spielt die Hauptrolle um Berechnungen durchzuführen. Durch das angehängte *KnapsackSolver.cs* Script wird die **KnapsackSolver** Klasse realisiert. Diese Klasse ist verantwortlich für die Berechnung der perfekten Lösung und ebenfalls für die Berechnung des selbst zusammengestellten Inventars des Benutzers.

Durch die Interaktion zwischen dem *InventoryController*, *KnapsackSolver* und der *EventManager* Klasse wird bei jedem neuen Item stets gewährleistet, dass das Inventar, mit dem gerechnet wird, stets das aktuelle ist.

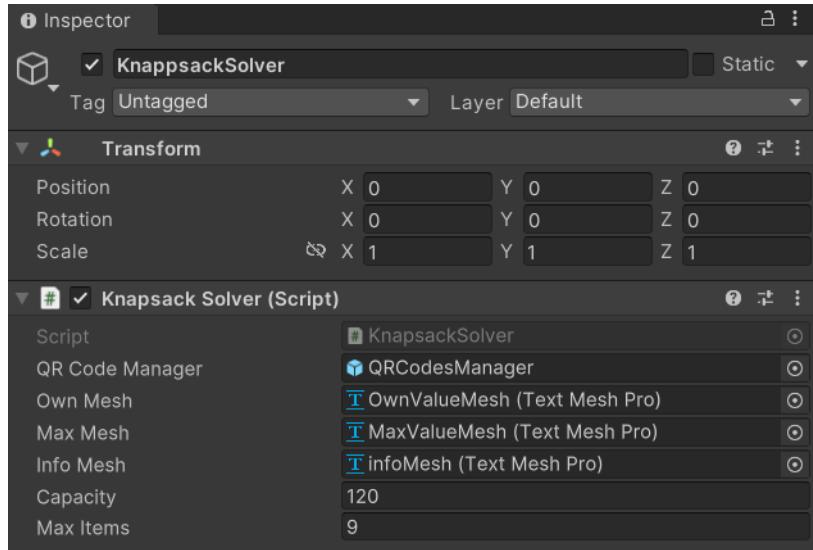


Abbildung 4.18: Knapsack Algorithmus Objekt im Editor

Die Abbildung 4.18 zeigt das *KnapsackSolver*-Objekt im Unity Editor. Hier können verschiedene Einstellungen vorgenommen werden, darunter die Ebene, in der dieses Objekt liegt, die Koordinaten des Objekts im Unity Editor selbst und die angehängte Komponente *KnapsackScript.cs*. Zudem sind vordefinierte Werte für bestimmte Variablen sichtbar. Diese Variablen umfassen:

- **QRCodesManager:** Referenz auf das *QRCodesManager* Game Objekt aus der Level 2 Szene.
- **Own Mesh:** Referenz auf das *OwnValueMesh* aus dem *infoObjekt* aus der Level 2 Szene.
- **Max Mesh:** Referenz auf das *.MaxValueMesh* aus dem *infoObjekt* aus der Level 2 Szene.
- **Info Mesh:** Referenz auf das *infoMesh* aus dem *infoObjekt* aus der Level 2 Szene.
- **Capacity:** Integer Wert der die Kapazität für den *Knapsack Algorithmus* festlegt.
- **Max Items:** Repräsentiert die maximale Anzahl an Items die in das Inventar gelegt werden können. Dies dient als extra Bedingung für den *Knapsack Algorithmus*.

4.5.7.1 KnapsackSolver Klassenvariablen

```

1 public GameObject QRCodeManager;
2 public TextMeshPro ownMesh;
3 public TextMeshPro maxMesh;
4 public TextMeshPro infoMesh;
5
6 public int[,] usedItems;
7 public int capacity = 120;
8 public Dictionary<int, QRData> items;
9 public int maxItems = 9;
10
11 private int[,] inventory;

```

Listing 4.21: Klassenvariablen des KnapsackSolvers

Die im Codeabschnitt 4.21 gezeigten Klassenvariablen gehören zur *KnapsackSolver*-Klasse. Diese Variablen werden verwendet, um Objekte und Werte innerhalb des Unity Editors zu repräsentieren, die entweder direkt festgelegt und übergeben werden oder von anderen Klassen aus Funktionalitätsgründen benötigt werden. Die öffentlichen (*public*) Variablen ermöglichen einen direkten Zugriff auf diese Objekte in der eigenen oder einer anderen Klasse.

Das *2D int Array inventory* spielt eine entscheidende Rolle im Verlauf des *KnapsackSolvers*, da es verwendet wird, um das individuell vom Benutzer zusammengestellte Inventar zu verarbeiten und zu berechnen.

Die privaten (*private*) Klassenvariablen dienen hauptsächlich dem lokalen Speichern von Werten, die nur innerhalb der *KnapsackSolver*-Klasse benötigt werden und keinen Zugriff von außen erfordern.

4.5.7.2 Start des KnapsackSolvers

Die **Start()** Funktion, die in folgendem Codeabschnitt 4.23 abgebildet ist, ist der Startpunkt des *KnapsackSolvers*.

```
1 void Start()
2 {
3     items = new QRItem(0).items;
4     EventManager.OnGridUpdate += SetInventory;
5 }
```

Listing 4.22: Klassenvariablen der InventoryController Klasse

Zu Beginn der Funktion wird dem Dictionary *items* ein neues Objekt des Typs *QRItem* zugewiesen, wobei die *ID* 0 übergeben wird. Dies ermöglicht den Zugriff auf das Dictionary in der *QRItem* Klasse welches alle Daten der einzelnen Items enthält. Anschließend wird das Event *OnGridUpdate* des *EventManager*s ausgelöst und die Funktion **SetInventory()** aufgerufen. Letztere ist eine Rückruffunktion, die als Reaktion auf das Ereignis aufgerufen wird und die Aufgabe hat, das Inventar zu aktualisieren. Diese Funktion sieht wie folgt aus:

```
1 public void SetInventory(int[,] newInventory)
2 {
3     inventory = newInventory;
4     CalculateKnapsack();
5 }
```

Listing 4.23: Inventar setzen

4.5.7.3 Starten der Berechnung

Um stets mit dem aktuellen Inventar zu rechnen, wird bei jedem mal, bei dem die **SetInventory()** Funktion aufgerufen wird, das aktuelle Inventar (*inventory*) mit dem neuen Inventar in dem ein neues Item enthalten ist (*newInventory*) aktualisiert. Um anschließend die tatsächlichen Werte zu berechnen, wird die **CalculateKnapsack()** Funktion aufgerufen. Der Code zu dieser Funktion:

```
1 void CalculateKnapsack()
2 {
3     int maxValue = Knapsack.MaxValue(out usedItems);
4     int inventoryValue = -1;
5     maxMesh.text = "Maximal erreichbarer Wert: " + maxValue.ToString();
6     try
```

```

7   {
8     inventoryValue = KnapsackInventoryValue(inventory);
9     if (maxValue == inventoryValue)
10    {
11      infoMesh.color = Color.green;
12      infoMesh.text = "Maximale Punktzahl erreicht";
13    }
14    else
15    {
16      infoMesh.text = "";
17    }
18    ownMesh.text = "Erreichter Wert: " + inventoryValue.ToString();
19  }
20  catch (Exception e)
21  {
22    Debug.LogError("Error calculating inventory value: " + e.Message);
23  }
24 }
```

Listing 4.24: Berechnungsfunktion

Die Funktion **CalculateKnapsack()** startet den Algorithmus zur Berechnung des Knapsackproblems. Zunächst wird eine Variable *maxValue* initialisiert, welche den maximal erreichbaren Wert des Rucksacks repräsentiert. Diese wird durch den Aufruf der Funktion **KnapsackMaxValue()** initialisiert, wobei auch eine Liste von *usedItems* als Ausgabe zurückgegeben wird. Diese Liste repräsentiert die in der *perfekten Lösung* enthaltenen Items.

Anschließend wird die Variable *inventoryValue* initialisiert und auf -1 gesetzt, um in dem Fall eines Fehlers keinen Programm-Crash zu verursachen. Die Funktion **KnapsackMaxValue()** wird anhand eines *try-catch* Blocks versucht auszuführen, wobei der Wert des aktuellen Inventars als Argument übergeben wird. Falls dieser Wert erfolgreich berechnet wird, wird er der Variable **inventoryValue** zugewiesen.

Nach der Berechnung des maximal erreichbaren Werts (*maxValue*) des Rucksacks wird geprüft, ob dieser Wert mit dem Wert des aktuellen Inventars (*inventoryValue*) übereinstimmt. Eine solche Übereinstimmung deutet darauf hin, dass der Benutzer eine perfekte Lösung gefunden hat. Diese Optimalität wird visuell durch die Farbänderung des *infoMesh* auf Grün und die Anzeige einer entsprechenden Erfolgsmeldung verdeutlicht. Im Gegensatz dazu wird bei Abweichungen zwischen dem maximal erreichbaren Wert und dem Wert des aktuellen Inventars der Text des *infoMesh* gelöscht, um vorherige Meldungen zu entfernen und das Feedbackfeld zu bereinigen.

Nachdem der Wert des zusammengestellten Inventars berechnet wurde, wird dieser im *ownMesh* angezeigt. Sollte während dieser Berechnung ein Fehler auftreten, wird eine entsprechende Fehlermeldung in die *Log-Datei* geschrieben, um die Fehlerbehandlung und -verfolgung zu unterstützen.

Um den beschriebenen Zustand der Erreichung einer perfekten Lösung zu verdeutlichen, wird im Folgenden ein Beispiel herangezogen.

Wie der beschriebene Ablauf, des findens einer perfekten Lösung aussieht, ist in folgender Abbildung 4.19 zu sehen.

Abbildung 4.19: Sicht des Benutzers nach finden einer perfekten Lösung

Auf dieser Abbildung ist zu sehen, dass das Inventar mit den Gegenständen mit folgenden *IDs*: , , , , , gefüllt ist und der errechnete Wert dieses Inventars

dem der perfekten Lösung gleicht. Diese Erkenntnis ist anhand des *infoMesh* zu erkennen in dem eine Erfolgsnachricht in grüner Farbe dargestellt ist.

4.5.7.4 Knapsack-Algorithmus

Der Knapsack-Algorithmus ist ein grundlegendes Werkzeug in der Informatik, das sich mit der optimalen Ressourcenallokation befasst, insbesondere wenn es darum geht, eine begrenzte Menge an Ressourcen effizient zu nutzen, um einen bestimmten Nutzen oder Gewinn zu maximieren. Der Begriff **Knapsack** bezieht sich dabei auf die metaphorische Idee, einen Rucksack mit einer beschränkten Kapazität mit Gegenständen zu füllen, die jeweils unterschiedliche Werte und Gewichte aufweisen.

Problemstellung

In der Problemstellung des Knapsack-Algorithmus wird ein Rucksack mit begrenzter Kapazität vorgegeben, sowie eine Menge von Gegenständen, von denen jeder einen spezifischen Wert und ein spezifisches Gewicht besitzt. Das Ziel besteht darin, eine Auswahl dieser Gegenstände zu treffen, die in den Rucksack passt und gleichzeitig den Gesamtwert maximiert.

Unterschiedliche Ansätze zur Lösung des Knapsack-Problems

Der Knapsack-Algorithmus kann auf verschiedene Weisen implementiert werden, wobei zwei der gängigsten Ansätze der dynamische Programmieransatz und der Greedy-Ansatz sind.

Dynamischer Programmieransatz

Bei dem *dynamischen Programmieransatz* wird eine Tabelle erstellt, um Teilprobleme zu lösen und die optimale Lösung zu berechnen. Dies geschieht durch die systematische Aufteilung des Problems in kleinere, leichter zu lösende Teilprobleme. Dieser Ablauf ist in dem folgenden Pseudocode zu sehen:

```

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
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

```

Listing 4.25: Dynamischer Algorithmus

Die Funktion **knapsackDynamic** nimmt drei Parameter an: ein Array von Gewichten (*weights*), ein Array von Werten (*values*) und die Kapazität des Rucksacks (*capacity*).

Zunächst wird die Länge des Gewichtsarrays als n gespeichert, und eine Tabelle *Tabelle* mit $n+1$ Zeilen und $capacity+1$ Spalten wird erstellt.

Anschließend werden zwei verschachtelte Schleifen verwendet, um alle möglichen Kombinationen von Gegenständen und Gewichten zu durchlaufen. In jedem Schritt wird überprüft, ob der aktuelle Gegenstand in den Rucksack passt (d.h. ob sein Gewicht kleiner oder gleich der aktuellen Kapazität ist). Falls ja, wird der Wert dieses Gegenstands zu dem Wert addiert, der erreicht werden kann, wenn der Rucksack ohne diesen Gegenstand gefüllt wird (*Tabelle*[i-1][w - weights[i-1]]). Andernfalls wird der Wert aus der vorherigen Zeile der Tabelle übernommen.

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

Aufbau und Interpretation der Tabelle der Teillösungen

Die folgende Matrix stellt die Tabelle der Teillösungen dar, die im Verlauf der Berechnung des Knapsack-Problems generiert wird.

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

Jede *Zeile* der obrigen 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.

Um anschließend eine Zelle a_{mn} dieser Matrix zu interpretieren, ist Folgendes wichtig zu wissen:

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 und 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 und 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.

Greedy-Ansatz

Im Gegensatz zu der 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. Dieser Ablauf ist in dem folgenden Pseudocode zu sehen:

```

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
11  FOR i FROM 0 TO n DO
12    IF currentWeight + items[i].weight <= capacity THEN
13      currentWeight += items[i].weight
14      totalValue += items[i].value
15    ELSE
16      ratio = (capacity - currentWeight) / items[i].weight
17      totalValue += ratio * items[i].value
18      BREAK
19    END IF
20  END FOR
21  RETURN totalValue
22 END FUNCTION

```

Listing 4.26: Greedy Algorithmus

Die Funktion **knapsackGreedy()** nimmt gleich 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.

Anwendungen

Der Knapsack-Algorithmus, ein fundamentales Werkzeug der Informatik, findet in einer Vielzahl von Anwendungsbereichen Anwendung, darunter:

1. **Logistik:** In der Logistik wird der Knapsack-Algorithmus verwendet, um den effizientesten Transport von Gütern mit begrenzten Kapazitäten zu planen. Durch die Auswahl der optimalen Kombination von Gütern können Logistikunternehmen ihre Transportkosten minimieren und die Effizienz ihrer Lieferketten verbessern.
2. **Finanzplanung:** In der Finanzplanung wird der Knapsack-Algorithmus genutzt, um das Portfolio von Investitionen zu optimieren. Investoren können mithilfe des

Algorithmus eine Auswahl von Wertpapieren treffen, die das Risiko minimieren und gleichzeitig den erwarteten Ertrag maximieren.

3. **Ressourcenmanagement:** Der Knapsack-Algorithmus wird auch im Ressourcenmanagement eingesetzt, um die effiziente Nutzung von begrenzten Ressourcen zu maximieren. Dies kann beispielsweise in der Produktion oder im Projektmanagement erfolgen, um die Nutzung von Arbeitskräften, Maschinen oder Zeit zu optimieren.
4. **Netzwerkoptimierung:** In der Netzwerkoptimierung findet der Knapsack-Algorithmus Anwendung bei der Planung und Optimierung von Netzwerken, wie beispielsweise bei der Routenplanung in Transportnetzwerken oder bei der Zuweisung von Ressourcen in Computernetzwerken.

Durch seine Vielseitigkeit und Effektivität ist der Knapsack-Algorithmus zu einem unverzichtbaren Werkzeug in verschiedenen Bereichen der Informatik und angewandten Mathematik geworden. Seine Fähigkeit, komplexe Probleme der Ressourcenzuweisung und Optimierung zu lösen, macht ihn zu einem wichtigen Bestandteil vieler industrieller Anwendungen und wissenschaftlicher Forschungsprojekte.

Auswahl des Ansatzes für die Knapsack-Algorithmus Implementierung

Bei der Implementierung des Knapsack-Algorithmus in dieser Applikation wurde bewusst der dynamische Ansatz gewählt. Diese Entscheidung basiert auf einer Reihe von Gründen, die im Folgenden erläutert werden:

1. **Optimale Lösungsgarantie:** Der dynamische Ansatz bietet die Möglichkeit, eine optimale 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 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 Variationen. 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 Applikation gewählt. Seine Fähigkeit, eine optimale Lösung zu garantieren, kombiniert mit seiner Effizienz und Flexibilität, macht ihn zu einer idealen Wahl für die Behandlung des Knapsack-Problems in diesem Kontext.

Knapsack-Algorithmus Implementierung

Der folgende Abschnitt beschreibt die Funktion **KnapsackMaxValue()**. Diese implementiert den dynamischen Ansatz des Knapsack-Problems und gibt zusätzlich ein 2D-Array zurück, in dem eine perfekte Lösung gespeichert ist. Die Funktion besteht im Wesentlichen aus zwei Teilen. Der erste Teil beinhaltet die Implementierung des Knapsack-Algorithmus.

Der zweite Teil dient dazu, die verwendeten Items zur Erreichung des maximalen Wertes zu verfolgen und daraus die perfekte Lösung in Form des *usedItems* Array zusammenzustellen. Der Code dieser Funktion lautet:

```

1 public int KnapsackMaxValue(out int[,] usedItems)
2 {
3     int n = items.Count;
4     int[,] dp = new int[n + 1, capacity + 1];
5     bool[,] selected = new bool[n + 1, capacity + 1];
6     for (int i = 0; i <= n; i++)
7     {
8         for (int w = 0; w <= capacity; w++)
9         {
10            if (i == 0 || w == 0)
11                dp[i, w] = 0;
12            else if (i <= maxItems && items[i].weight <= w)
13            {
14                int newValue = items[i].value + dp[i - 1, w - items[i].weight];
15                if (newValue > dp[i - 1, w])
16                {
17                    dp[i, w] = newValue;
18                    selected[i, w] = true;
19                }
20                else
21                {
22                    dp[i, w] = dp[i - 1, w];
23                    selected[i, w] = false;
24                }
25            }
26            else
27            {
28                dp[i, w] = dp[i - 1, w];
29                selected[i, w] = false;
30            }
31        }
32    }
33
34 // Backtrack to find selected items
35 int[,] tempUsedItems = new int[3, 3];
36 int row = n;
37 int col = capacity;
38 int rowIndex = 0;
39 int colIndex = 0;
40 while (row > 0 && col > 0 && rowIndex < 3 && colIndex < 3)
41 {
42     if (selected[row, col] && colIndex < maxItems)
43     {
44         tempUsedItems[rowIndex, colIndex] = items[row].id;
45         col -= items[row].weight;
46         row--;
47         colIndex++;
48         if (colIndex >= 3)
49         {
50             colIndex = 0;
51             rowIndex++;
52         }
53     }
54     else
55     {
56         row--;
57     }
}

```

```

58     }
59     usedItems = tempUsedItems;
60     return dp[n, capacity];
61 }
```

Listing 4.27: Knapsack Algorithmus / Item Backtracking

Die Implementierung des Knapsack-Algorithmus in den Zeilen drei bis 32 der Funktion **KnapsackMaxValue()** dieses Abschnitts entspricht im Wesentlichen dem bereits erklärten Pseudocode. Ein wesentlicher Unterschied besteht jedoch darin, dass diese spezielle Implementierung eine zusätzliche Bedingung berücksichtigt. Diese Bedingung besagt, dass zur Berechnung des maximalen Werts und damit der optimalen Lösung maximal 9 Gegenstände in die Berechnung miteinbezogen werden können. Dies ist darauf zurückzuführen, dass im vorliegenden Inventar-Objekt nur 9 verfügbare Zellen vorhanden sind, in die ein Gegenstand eingefügt werden kann. Die Erfüllung dieser Bedingung wird in dem *else-if*-Zweig $i \leq \maxItems \wedge \text{items}[i].weight \leq w$ gewährleistet.

Zusätzlich wird ein Array von booleschen Werten namens *selected* erstellt, um im Verlauf der Berechnung diejenigen Gegenstände zu markieren, die ausgewählt wurden (*selected* = *true*). Dieses Array wird verwendet, um im zweiten Teil der Funktion die perfekte Lösung zusammenzustellen.

Der zweite Teil dieses Codes, der sich um das *Backtracking* der in der perfekten Lösung verwendeten Items kümmert, beginnt damit, dass Variablen initialisiert werden, um den aktuellen Zeilen- und Spaltenindex in dem *selected* Array zu verfolgen, sowie *Indizes* für das temporäre Array, die die ausgewählten Gegenstände speichert. Zunächst wird eine Schleife gestartet, um durch das *selected* Array zu iterieren und die ausgewählten Gegenstände zu identifizieren.

Während der Iteration werden Bedingungen überprüft, um zu entscheiden, ob ein Gegenstand ausgewählt wurde. Wenn ein Gegenstand ausgewählt wird, wird seine *ID* in das temporäre Array (*tempUsedItems*) eingefügt, und die Position in der Tabelle wird aktualisiert, indem das Gewicht des ausgewählten Gegenstands von der aktuellen Kapazität subtrahiert wird. Die Schleife durchläuft die Tabelle und fährt fort, bis entweder die erste Zeile oder Spalte erreicht wird oder die maximal zulässige Anzahl von ausgewählten Gegenständen erreicht ist.

Wenn ein Gegenstand ausgewählt wird, wird seine ID in die temporäre Matrix eingefügt, um die ausgewählten Gegenstände zu speichern. Die Indizes der temporären Matrix werden aktualisiert, um den nächsten verfügbaren Speicherplatz zu zeigen. Andernfalls wird der Zeilenindex dekrementiert, um zur vorherigen Zeile in des *selected* Arrays zu gehen.

Am Ende der Funktion wird der Wert von *usedItems* mit dem ermittelten Array *tempUsedItems* überschrieben und der maximale Wert des Rucksacks wird zurückgegeben.

Aufbau und Interpretation der selected Tabelle

Angenommen, 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

Und die *Kapazität* des Rucksacks liegt bei 5. 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.

4.5.7.5 Berechnung des eigenen Inventars

Die letzte Berechnung die in dem *KnapsackSolver* durchgeführt wird, ist die Berechnung des eigenen Inventars. Dies wird mittels der **KnapsackInventoryValue()** Funktion erreicht. Der Code dieser Funktion:

```

1 public int KnapsackInventoryValue(int[,] inventory)
2 {
3     if (inventory == null)
4     {
5         throw new System.Exception("Inventory is null");
6     }
7     int totalValue = 0;
8     foreach (var item in items.Values)
9     {
10         int itemId = item.id;
11         int itemValue = item.value;
12         for (int j = 0; j < inventory.GetLength(0); j++)
13         {
14             for (int k = 0; k < inventory.GetLength(1); k++)
15             {
16                 if (inventory[j, k] == itemId)
17                 {
18                     totalValue += itemValue;
19                 }
20             }
21         }
22     }
23     return totalValue;
24 }
```

Listing 4.28: Funktion um eigenes Inventar zu berechnen

In dieser Funktion wird der Gesamtwert des aktuellen Inventars berechnet, welches durch das zweidimensionale Array (*inventory*) repräsentiert wird. Zunächst wird überprüft, ob das übergebene Inventar korrekt gesetzt wurde und ob es *null* ist oder nicht. Falls es *null* ist, wird eine *Exception* geworfen. Falls das Inventar nicht leer ist, wird die Variable *totalValue* initialisiert, um den Gesamtwert des Inventars zu speichern. Anschließend wird eine *foreach*-Schleife über jedes Element des Dictionaries *items* durchgeführt, wobei die *ID* und der Wert *value* jedes Elements ermittelt und in *itemId* und *itemValue* gespeichert werden.

Anschließend werden zwei verschachtelte *for*-Schleifen verwendet, um jedes Element im *inventory*-Array zu durchlaufen. Wenn die *ID* des Gegenstands im *inventory*-Array gefunden wird, wird der Wert dieses Gegenstands zum Gesamtwert *totalValue* addiert. Nach Abschluss der beiden *for*-Schleifen wird dieser ert schließlich zurückgegeben.

4.5.7.6 InfoMesh aktualisieren

Der *KnapsackSolver* enthält zwei Funktionen, die sowohl in der Klasse selbst als auch in der *InventoryController* Klasse gebraucht werden. Die Funktion **SetInventory()** wurde bereits erklärt und die zweite Funktion ist die folgende:

```

1 public void UpdateInfoMesh(string input)
2 {
3     infoMesh.color = Color.red;
4     infoMesh.text = input;
5 }
```

Listing 4.29: Funktion um InfoMesh zu verändern

Diese Funktion wird von der Klasse *InventoryController* verwendet und sie dient dazu, das *infoMehs* mit einem neuen Text zu aktualisieren und rot zu färben, um dem Benutzer Fehler anzuzeigen.

4.5.8 Best Solution Prefab Game Objekt

Das *best solution prefab* nimmt eine zentrale Rolle im weiteren Verlauf der Applikation ein. Mittels des angefügten *PerfectSolutionVisualizer.cs* Skripts wird die **PerfectSolutionVisualizer** Klasse implementiert, die die Visualisierung der zuvor berechneten optimalen Lösung ermöglicht. Diese Visualisierung ist von entscheidender Bedeutung, um dem Benutzer die bestmögliche Lösung aufzuzeigen, ihr Erscheinungsbild zu veranschaulichen und die enthaltenen Elemente dieser Lösung deutlich zu machen.

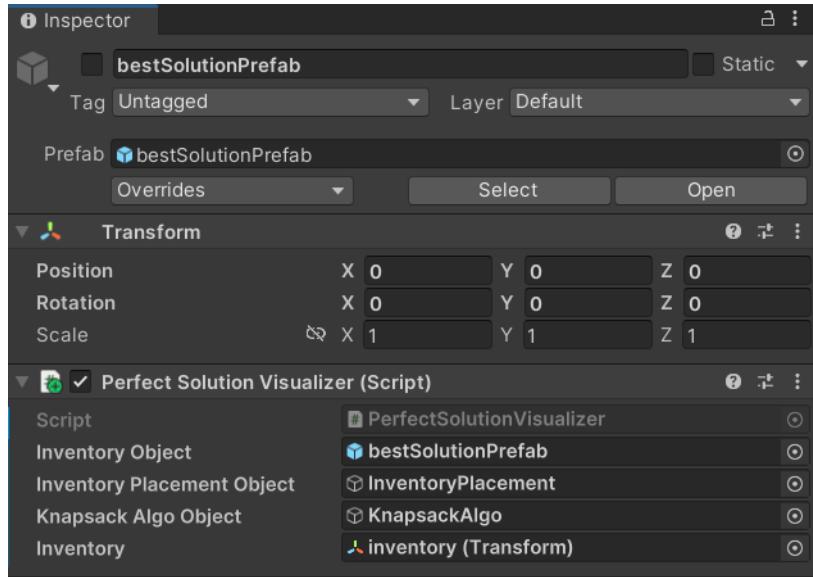


Abbildung 4.20: bestSolutionPrefab im Unity Editor

In Abbildung 4.20 ist das *bestSolutionPrefab* im Unity Editor dargestellt. Hier können verschiedene Einstellungen vorgenommen werden, darunter die Ebene, in der dieses Objekt platziert ist, die Koordinaten des Objekts im Unity Editor selbst sowie die angehängte Komponente *PerfectSolutionVisualizer.cs*. Des Weiteren sind vordefinierte Werte für bestimmte Variablen sichtbar. Diese Variablen umfassen:

- **Inventory Object:** Eine Referenz auf das *Prefab* für die perfekte Lösung.
- **Inventory Placement Object:** Eine Referenz auf das *inventoryPlacement* Game Objekt.
- **Knapsack Algo Object:** Eine Referenz auf das *KnapsackSolver* Game Objekt.
- **Inventory:** Eine Referenz auf das *Inventory* Modell, das im Best Solution Prefab enthalten ist.

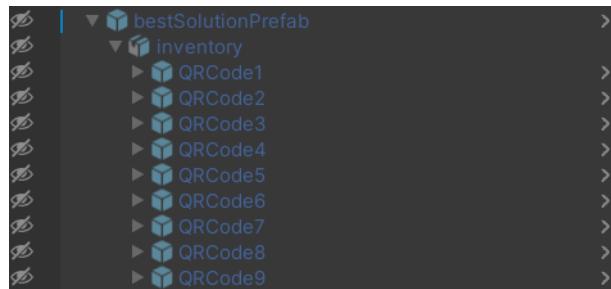


Abbildung 4.21: Inventar Prefab Hirarchie im Unity Editor

Abbildung 4.21 zeigt die Struktur des Prefabs für die perfekte Lösung. Hierbei ist zu erkennen, dass das Haupt-Game-Objekt das Game-Objekt *Inventory* enthält. Dieses *Inventory*-Game-Objekt repräsentiert das 3D-Modell des Inventars. Des Weiteren hat dieses *Inventory*-Game-Objekt mehrere untergeordnete Prefabs. Diese *QRItem*-Prefabs sind entscheidend, um basierend auf der ID das entsprechende Modell anzuzeigen. Die Gesamtstruktur des Prefabs sieht dann im Unity-Editor wie folgt aus:

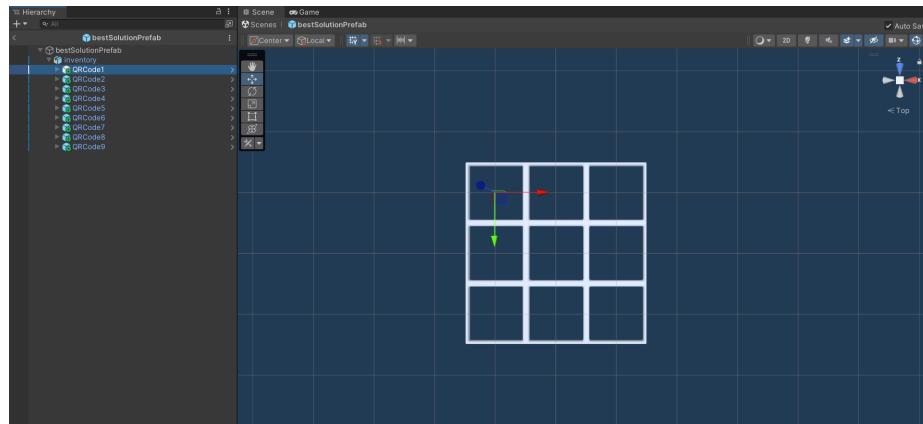


Abbildung 4.22: Inventar Prefab im Unity Editor

In dieser Abbildung ist zu erkennen, dass das *QRItem1*-Prefab in der linken oberen Ecke (*Index 0 des inventars*) positioniert ist. Des Weiteren sind alle *QRItems* in den Zellen des Inventars angeordnet, um später anhand eines Index das entsprechende Array zu durchlaufen, in dem die perfekte Lösung gespeichert ist. Auf diese Weise können die zugehörigen Modelle in den entsprechenden *QRItems* aktiviert und angezeigt werden.

4.5.8.1 Script Aufruf

Um die perfekte Lösung zu visualisieren, wird das *PerfectSolutionVisualizer.cs*-Skript durch einen Knopfdruck gestartet. Der zugehörige Knopf, der für das Auslösen dieses Skripts zuständig ist, befindet sich in dem Objekt *infoObject*. Der Aufruf dieses Skripts ist in der folgenden Abbildung 4.23 dargestellt:

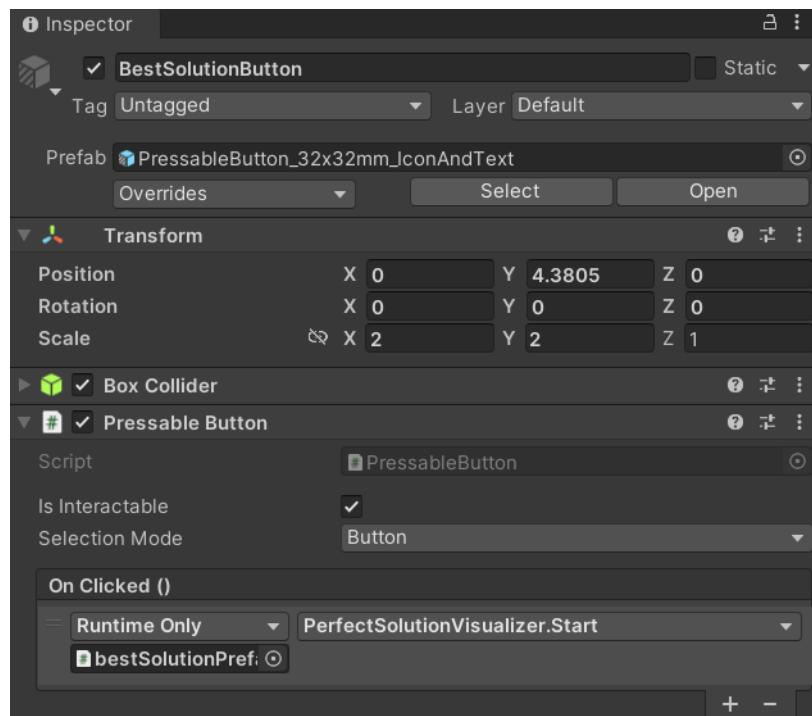


Abbildung 4.23: Script Aufruf bei Knopfdruck

In dieser Abbildung sind die Komponenten des *BestSolutionButton* zu sehen. Der zentrale Bestandteil dieser Abbildung ist das Skript *PressableButton*. Dieses Skript stellt die Funktion **OnClicked()** bereit, die einen einfachen Knopfdruck implementiert. In dieser Funktion wird in diesem Fall die *Start()*-Funktion des Skripts *PerfectSolutionVisualizer.cs* aufgerufen, um das Skript zu starten.

4.5.8.2 PerfectSolutionVisualizer Klassenvariablen

```

1 public GameObject inventoryObject;
2 public GameObject inventoryPlacementObject;
3 public GameObject KnapsackAlgoObject;
4 public Transform inventory;
5
6 private PlaceObjectOnLookedAtDesk anchorScript;
7 private KnapsackScript knapsackScript;
8 private Vector3 originalInventoryPosition;
9 private int[,] perfectSolution;
10 private bool isClicked = false;
11 private int numRows = 3;
12 private int numColumns = 3;
```

Listing 4.30: Klassenvariablen des PerfectSolutionVisualizer

Die in Codeabschnitt 4.30 gezeigten Klassenvariablen gehören zu der *PerfectSolutionVisualizer* Klasse. Diese Variablen werden verwendet, um Objekte und Werte innerhalb des Unity Editors zu repräsentieren, die entweder direkt festgelegt und übergeben werden oder von anderen Klassen aus Funktionalitätsgründen benötigt werden. Die öffentlichen (*public*) Variablen ermöglichen einen direkten Zugriff auf diese Objekte in der eigenen oder einer anderen Klasse.

Besonders wichtig ist der Zugriff auf die Variablen *inventoryPlacementObject* und *KnapsackAlgoObject*, um im weiteren Verlauf dieses Skripts auf die beiden angehängten Skripte dieser Game-Objekte zuzugreifen. Diese Skripte speichern die Position des Inventars und das *usedItems*-Array, die für das Anzeigen der perfekten Lösung benötigt werden. Die Position des originalen Inventars ist notwendig, um die perfekte Lösung korrekt zu platzieren, während das *usedItems*-Array später benötigt wird, um das *bestSolutionPrefab* anhand der gespeicherten Werte zu befüllen.

4.5.8.3 Start des PerfectSolutionVisualizer

Nachdem der *BestSolutionButton* vom Benutzer gedrückt wurde, wird anschließend die *Start()* Funktion aufgerufen, die den Prozess für das Anzeigen der perfekten Lösung startet. Der zugehörige Code lautet:

```

1 public void Start()
2 {
3     isClicked = !isClicked;
4     if (isClicked == true)
5     {
6         anchorScript = inventoryPlacementObject.GetComponent<
7             InventoryPlacementController>();
8         originalInventoryPosition = anchorScript.objectPosition;
9         knapsackSolver = KnapsackAlgoObject.GetComponent<KnapsackSolver>();
10        perfectSolution = knapsackSolver.usedItems;
11        printItems();
12        setNewPosition();
13        inventoryObject.SetActive(true);
```

```

13     fillInventory();
14 }
15 else
16 {
17     inventoryObject.SetActive(false);
18 }
19 }
```

Listing 4.31: PerfectSolutionVisualizer Start

Um sicherzustellen, dass der Benutzer die perfekte Lösung sowohl anzeigen (aktivieren) als auch wieder verstecken (deaktivieren) kann, wird bei jedem Aufruf der **Start()** Funktion die boolsche Variable *isClicked* negiert.

Wenn *isClicked* den Wert *true* hat, werden die *objectPosition* der Klasse *InventoryPlacementController* und das *usedItems*-Array der Klasse *KnapsackSolver* gespeichert. Anschließend werden die Funktionen **setNewPosition()** und **fillInventory()** aufgerufen, um die genaue Positionierung der perfekten Lösung und das Befüllen des Inventars mit den entsprechenden Objekten zu gewährleisten.

Wenn *isClicked* jedoch den Wert *false* annimmt, wird das gesamte *inventoryObject* durch die Verwendung der **SetActive()** Funktion deaktiviert. Dadurch wird es unsichtbar für den Benutzer. Dies ermöglicht eine einfache Handhabung der Anzeige und Ausblendung der perfekten Lösung.

4.5.8.4 Setzen der neuen Position der perfekten Lösung

Da die perfekte Lösung so positioniert werden muss, dass sie für den Benutzer leicht sichtbar ist, wird die folgende Funktion aufgerufen:

```

1 private void setNewPosition()
2 {
3     Vector3 newPosition = originalInventoryPosition + Vector3.forward * 0.5f + Vector3
4         .up * 0.205f;
5     inventoryObject.transform.position = newPosition;
6     Quaternion objectRotation = Quaternion.Euler(-45f, 0f, 0f);
7     inventoryObject.transform.rotation = objectRotation;
}
```

Listing 4.32: Neue Position setzen

Zunächst wird eine neue Position (*newPosition*) für das Inventarobjekt berechnet. Diese Position basiert auf der ursprünglichen Position des Inventarobjekts (*originalInventoryPosition*). Die neue Position wird um 0,5 Einheiten entlang der Vorwärtsachse (z-Achse) und um 0,205 Einheiten entlang der Aufwärtsachse (y-Achse) verschoben. Die Position des Inventarobjekts wird auf die berechnete *newPosition* gesetzt, wodurch das Objekt an die neue Position verschoben wird. Anschließend wird eine Quaternion-Rotation (*objectRotation*) erstellt, um das Inventarobjekt um -45 Grad entlang der x-Achse zu kippen. Schließlich wird die Rotation des Inventarobjekts auf die berechnete *objectRotation* gesetzt, um das Objekt entsprechend zu kippen.

4.5.8.5 Perfekte Lösung füllen

Aufgrund dessen, dass das zuvor platzierte Objekt lediglich ein leeres Raster darstellt, ist es erforderlich, dieses Raster anschließend mit den entsprechenden Elementen zu füllen, um die perfekte Lösung zu repräsentieren. Zu diesem Zweck wird im weiteren Verlauf die Funktion **fillInventory()** aufgerufen. Der Code dieser Funktion:

```

1  private void fillInventory()
2  {
3      for (int i = 0; i < numRows; i++)
4      {
5          for (int j = 0; j < numColumns; j++)
6          {
7              int id = perfectSolution[i, j];
8              if (perfectSolution[i, j] == 0)
9                  continue;
10             else
11             {
12                 string qrCodeName = "QRCode" + (i * numColumns + j + 1);
13                 Transform qrCodeTransform = inventory.Find(qrCodeName);
14                 if (qrCodeTransform != null)
15                 {
16                     Transform childTransform = qrCodeTransform.Find(id.ToString());
17                     if (childTransform != null)
18                     {
19                         childTransform.gameObject.SetActive(true);
20                     }
21                 }
22             else
23             {
24                 Debug.LogError($"QRCode {qrCodeName} not found in the inventory");
25             }
26         }
27     }
28 }
29 }
```

Listing 4.33: Inventar füllen

Die vorliegende Funktion durchläuft das zweidimensionale Array *perfectSolution*, das die perfekte Lösung repräsentiert. Zu Beginn wird die ID an der Stelle $[i, j]$ des Arrays gespeichert. An jeder Position wird zunächst überprüft, ob die gespeicherte ID gleich 0 ist. In diesem Fall wird der Schleifendurchlauf mit *continue* übersprungen, da der Wert 0 darauf hinweist, dass an dieser Stelle kein Item liegt.

Wenn der Wert an der Stelle $[i, j]$ größer als 0 ist, deutet dies darauf hin, dass an dieser Position im Inventar ein konkretes Element vorliegt. Zur Identifizierung dieses Elements und zum Auffinden des entsprechenden QRItems wird ein stringbasierter Bezeichner *qrCodeName* generiert. Dieser Bezeichner wird durch die Konkatenation des Präfixes *QRCode* mit dem Ergebnis der Berechnung $i \times \text{numColumns} + j + 1$ erzeugt. Diese Berechnung berücksichtigt die aktuelle Position im zweidimensionalen Array und ermöglicht die Erstellung eines eindeutigen Bezeichners für das QRItem. Auf diese Weise wird das QRItem erfolgreich identifiziert und kann anschließend im Inventar lokalisiert werden.

Um diesen Vorgang der Identifikation des korrekten *QRItem* Prefabs besser zu veranschaulichen wird hierfür ein Beispiel herangezogen, dass die Berechnung anhand von Testwerten durchführt.

Angenommen in der Schleife hat **i** den Wert 1 und **j** den Wert 1. Dies bedeutet, dass das Array momentan an der Stelle [1, 1] steht. Das **bestSolutionPrefab** ist so strukturiert, dass der Index nicht mit 0, sondern mit 1 beginnt. Daher wird in der Berechnung am Ende +1 hinzugefügt, um dies zu berücksichtigen. Somit wird an der Stelle $[i, j]$ das *QRItem5* dem Array zugeordnet.

Nachdem das richtige *QRItem* identifiziert wurde, wird anschließend das dem Inventar-Objekt untergeordnete *QRItem* Prefab mit diesem Namen gespeichert. Wenn dieses Objekt ungleich null ist, wird von diesem Prefab das untergeordnete Modell anhand der zuvor gespeicherten *ID* aktiviert, um es in dem Inventar-Raster anzuzeigen. Andernfalls wird eine Fehlermeldung in die *Logdatei* geschrieben. Der Zustand nach dem Abschluss dieser Funktion ist in der folgenden Abbildung 4.24 zu sehen.

Abbildung 4.24: Perfekte Lösung

4.5.9 Unit-Tests

Durch Hilfe von Unit-Tests wird versichert, dass der implementierte Knapsack-Algorithmus richtig und performant funktioniert.

4.6 Performance und Qualitätssicherung

4.6.1 Unit-Tests

Unit-Tests sind ein wichtiger Bestandteil der Qualitätssicherung. Sie ermöglichen die Überprüfung der korrekten Funktionalität einzelner Komponenten und stellen sicher, dass sie wie erwartet arbeiten. Im Rahmen dieses Projekts wurden Unit-Tests verwendet, um den Knapsack-Algorithmus zu überprüfen. Die Tests wurden mithilfe des Unity Test Frameworks erstellt und ausgeführt. Dieses Tool wurde speziell für die Erstellung und Ausführung von Unit-Tests in Unity entwickelt. Durch die sorgfältige Gestaltung der Tests wurde sichergestellt, dass der Algorithmus nicht nur die erwarteten Ergebnisse zurückgibt, sondern auch die vorgegebene Laufzeit einhält. Die Tests wurden innerhalb der Entwicklungsumgebung von Unity durchgeführt. Die Ergebnisse wurden genau überwacht, um sicherzustellen, dass der Algorithmus zuverlässig funktioniert.

4.6.2 Unity Test Framework

Das Unity Test Framework bietet eine breite Palette von Funktionen, mit denen Entwickler umfassende Tests für ihre Unity-Skripte schreiben und ausführen können. Dabei können verschiedene Aspekte der Skripte auf ihre korrekte Funktionalität überprüft werden. Dazu gehören das Testen von Variablen, das Aufrufen von Funktionen und das Überprüfen von erwarteten Ergebnissen. Entwickler können dank dieser umfangreichen Funktionen sicherstellen, dass ihre Unity-Anwendungen robust und zuverlässig sind. So schaffen sie eine solide Grundlage für die Qualitätssicherung.

4.6.3 Performance-Messung

Die Performance des Knapsack-Algorithmus wurde mithilfe von Unit-Tests überprüft. Dabei wurde die Laufzeit des Algorithmus für verschiedene Eingabegrößen gemessen. Die Ergebnisse wurden sorgfältig analysiert, um sicherzustellen, dass der Algorithmus die erwartete Laufzeit einhält. Es wurden verschiedene Algorithmen und Implementierungen getestet, um die optimale Leistung zu erzielen. Die Ergebnisse der Performance-Messung wurden genau überwacht, um sicherzustellen, dass der Algorithmus zuverlässig und effizient arbeitet.

```
1 public class AlgoTest
2 {
```

```

3     GameObject testObject;
4     KnapsackScript knapsackScript;
5     int[,] usedItems;
6     int capacity = 120;
7
8     [SetUp]
9     public void Setup()
10    {
11        // Vor jedem Test eine Instanz von KnapsackScript erstellen
12        testObject = new GameObject();
13        knapsackScript = testObject.AddComponent<KnapsackScript>();
14    }
15
16     [TearDown]
17     public void Teardown()
18    {
19        // Das Test-GameObject nach jedem Test zerstören
20        GameObject.DestroyImmediate(testObject);
21    }
22
23     // Ein Test verhält sich wie eine normale Methode
24     [Test]
25     public void KnapsackMaxValue_ReturnsCorrectValue()
26    {
27        // Gibt eine Menge von Gegenständen für den Rucksack an
28        Dictionary<int, QRData> items = new Dictionary<int, QRData>()
29        {
30            {1, new QRData { id = 1, weight = 50, value = 100 }},
31            {2, new QRData { id = 2, weight = 25, value = 50 }},
32            {3, new QRData { id = 3, weight = 20, value = 30 }},
33            {4, new QRData { id = 4, weight = 10, value = 15 }},
34            {5, new QRData { id = 5, weight = 10, value = 5 }},
35            {6, new QRData { id = 6, weight = 25, value = 50 }},
36            {7, new QRData { id = 7, weight = 10, value = 10 }},
37            {8, new QRData { id = 8, weight = 5, value = 50 }},
38            {9, new QRData { id = 9, weight = 5, value = 15 }},
39            {10, new QRData { id = 10, weight = 20, value = 15 }},
40        };
41
42        System.Random random = new System.Random();
43        for (int i = 11; i <= 100; i++)
44        {
45            int randomWeight = random.Next(1, 51);
46            int randomValue = random.Next(1, 101);
47
48            items.Add(i, new QRData { id = i, weight = randomWeight, value = randomValue });
49        }
50
51        knapsackScript.capacity = capacity;
52        knapsackScript.maxItems = items.Count;
53        knapsackScript.items = items;
54
55        // Messung der Ausführungszeit für KnapsackMaxValue
56        System.Diagnostics.Stopwatch knapsackMaxValueStopwatch = System.Diagnostics.
57        Stopwatch.StartNew();
58        int maxValue = knapsackScript.KnapsackMaxValue(out usedItems);
59        knapsackMaxValueStopwatch.Stop();
60        Debug.Log($"Die Ausführungszeit von KnapsackMaxValue beträgt: {knapsackMaxValueStopwatch.ElapsedMilliseconds} ms");
61
62        // Messung der Ausführungszeit für KnapsackMaxValueRecursive

```

```

61     System.Diagnostics.Stopwatch recursiveStopwatch = System.Diagnostics.Stopwatch
62     .StartNew();
63     int maxValueComp = KnapsackMaxValueRecursive(items.Count, capacity, items, new
64     Dictionary<(int, int), int>());
65     recursiveStopwatch.Stop();
66     Debug.Log($"Die Ausf黨rungszeit von KnapsackMaxValueRecursive betr鋗gt: {recursiveStopwatch.ElapsedMilliseconds} ms");
67
68 }
69
70 // Rekursive Implementierung des Knapsack-Algorithmus zur Berechnung des maximalen
71 // Werts
72 public int KnapsackMaxValueRecursive(int n, int remainingCapacity, Dictionary<int,
73     QRData> items, Dictionary<(int, int), int> memo)
74 {
75     if (n == 0 || remainingCapacity == 0)
76         return 0;
77
78     if (memo.TryGetValue((n, remainingCapacity), out int memoizedValue))
79         return memoizedValue;
80
81     if (items[n].weight > remainingCapacity)
82     {
83         memo[(n, remainingCapacity)] = KnapsackMaxValueRecursive(n - 1,
84         remainingCapacity, items, memo);
85         return memo[(n, remainingCapacity)];
86     }
87
88     int includedValue = items[n].value + KnapsackMaxValueRecursive(n - 1,
89         remainingCapacity - items[n].weight, items, memo);
90     int excludedValue = KnapsackMaxValueRecursive(n - 1, remainingCapacity, items,
91         memo);
92
93     int result = Math.Max(includedValue, excludedValue);
94     memo[(n, remainingCapacity)] = result;
95
96     return result;
97 }
98 }
```

Listing 4.34: Inventar fĂ4llen

Dies ist eine Testklasse in C, welche den Knapsack-Algorithmus implementiert und testet.

Die Klasse AlgoTest enthält Testfunktionen für zwei verschiedene Knapsack-Algorithmen.

Es werden mehrere Variablen definiert, darunter ein testObject, das zu testende knapsack-Script und capacity, die maximale Größe des Rucksacks, die für die Tests benötigt wird. Die Methoden SetUp und TearDown sind mit den Attributen [SetUp] und [TearDown] versehen. Diese Attribute werden vom Framework erkannt und dienen dazu, nach jedem Test die Umgebung zu reinigen und neu aufzusetzen. Es wird eine Instanz des KnapsackScript erstellt und wieder zerstört, um sicherzustellen, dass die Tests in einer sauberen Umgebung ausgeführt werden.

Der Unit-Test KnapsackMaxValueReturnsCorrectValue testet die Methode KnapsackMaxValue des Knapsack-Script und vergleicht die Ergebnisse mit der Implementierung des iterativen Knapsack-Algorithmus. Anschließend wird die Ausführungszeit sowohl der iterative als auch der rekursiveen Implementierung verglichen.

Die Methode KnapsackMaxValueRecursive ist eine rekursive Implementierung des Knapsack-Algorithmus. Der maximale Wert, den der Rucksack aufnehmen kann, wird berechnet, in-

dem die Elemente rekursiv entweder eingeschlossen oder ausgeschlossen werden. Es wurde darauf geachtet, verschiedene Implementierungen zu testen, um sicherzustellen, dass die optimale Implementierung verwendet wird.

Die Testmethode Knapsack.MaxValue Returns *CorrectValue wird für jede Konfiguration von Gegenständen der Algoritmus korrekt funktioniert.*

Dabei wird die rekursive Implementierung Knapsack.MaxValueRecursive als Vergleich verwendet, um sicherzustellen, dass auch die iterative Implementierung Knapsack.MaxValue korrekt ist.

Kapitel 5

Zusammenfassung und Abschluss

5.1 Ergebnis

Hier steht der allgemeine Text für das Ergebnis

5.2 Abnahme

Hier steht der allgemeine Text für das Abnahme

5.3 Zukunft

Hier steht der allgemeine Text für die Zukunft

Anhang A

Mockups

A.1 UI/UX

A.2 Hauptmenu Level Design

A.3 Ping-Paket Level Design

A.4 Knapsack-Level Design

Anhang B

Literatur

- Blender. URL: <https://www.blender.org/about/> (besucht am 06.10.2023).
- Scrum Alliance Inc. WHAT IS SCRUM? URL: <https://www.scrumalliance.org/about-scrum#> (besucht am 06.10.2023).
- Scrum-Master.de Scrum-Rollen - Product Owner. URL: https://scrum-master.de/Scrum-Rollen/Scrum-Rollen_Product_Owner (besucht am 10.11.2023).
- Scrum-Master.de Scrum-Rollen - Scrum Master. URL: https://scrum-master.de/Scrum-Rollen/Scrum-Rollen_ScrumMaster (besucht am 10.11.2023).
- Scrum-Master.de Scrum-Rollen - Team. URL: https://scrum-master.de/Scrum-Rollen/Scrum-Rollen_Team (besucht am 10.11.2023).
- Scrum-Master.de Scrum-Meetings - Sprint - Team. URL: <https://scrum-master.de/Scrum-Meetings/Sprint> (besucht am 10.11.2023).
- Scrum-Master.de Scrum-Meetings - Sprint Planing Meeting - Team. URL: https://scrum-master.de/Scrum-Meetings/Sprint_Planning_Meeting (besucht am 10.11.2023).
- Scrum-Master.de Scrum-Meetings - Daily Scrum Meeting - Team. URL: <https://scrum-master.de/Scrum-Meetings/Sprint> (besucht am 10.11.2023).
- Scrum-Master.de Scrum-Meetings - Sprint Review Meeting - Team. URL: https://scrum-master.de/Scrum-Meetings/Sprint_Review_Meeting (besucht am 10.11.2023).
- Scrum-Master.de Scrum-Meetings - Sprint Retroperspektiv Meeting - Team. URL: https://scrum-master.de/Scrum-Meetings/Sprint_Review_Meeting (besucht am 10.11.2023).
- Microsoft. 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. 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. 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. Plane. URL: <https://docs.unity3d.com/ScriptReference/Plane.html> (besucht am 13.12.2023).
- Unity. Scenes. URL: <https://docs.unity3d.com/Manual/CreatingScenes.html> (besucht am 13.12.2023).
- Unity. Prefabs. URL: <https://docs.unity3d.com/Manual/Prefabs.html> (besucht am 15.01.2024).

- Unity. Bounds. URL: <https://docs.unity3d.com/ScriptReference/Bounds.html> (besucht am 16.01.2024).
- Unity. 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.ARRaycastHit.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/ (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 (besucht am 20.11.2023).
- Blender. Vertices. URL: 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).
- Autodesk FBX. Getting started. URL: 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 (besucht am 05.12.2024).

- 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* 3. Aufl. Stuttgart: utb GmbH, 2014.
- Mayer, Horst. *Interview und schriftliche Befragung. Entwicklung, Durchführung und Auswertung* 3. Aufl. München: R. Oldenbourg, 2008.
- Bühner, Markus. *Einführung in die Test- und Fragebogenkonstruktion*. 3. Aufl. München: Pearson Studium, 2021.