
DIPLOMARBEIT

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 16. Januar 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 16. Januar 2024

Verfasser / Verfasserinnen:

Moritz SKREPEK

Dustin LAMPEL

Seref HAYLAZ

Jonas SCHODITSCH

Inhaltsverzeichnis

Eidesstattliche Erklärung	i
Vorwort	iv
Diplomarbeit Dokumentation	v
Diploma Thesis Documentation	vii
Kurzfassung	ix
Abstract	x
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 Fragebogenkonzeption	6
2.4.2 Abfassung der Fragen	6
2.4.3 Arten von Fragen	7
2.4.4 Struktur und Gliederung von Fragebögen	7
2.4.5 Mögliche Verfälschung des Resultats	7
2.4.6 Auswertung von Fragebögen	7
3 Produktspezifikationen	8
3.1 Anforderungen und Spezifikationen	8
3.1.1 Use Cases	8
3.2 Design	8
3.2.1 Abläufe	8

3.2.2	Mockups	8
3.3	Eingesetzte Technologien	8
3.3.1	Kriterien	8
3.3.2	Game Engine: Konzeption und Funktion	8
3.3.3	Unity foundation packages	10
3.3.4	Modellierungsprogramm	11
4	Feinkonzept und Realisierung	14
4.1	Entwicklungsumgebungen	14
4.1.1	Visual Studio 2022	14
4.1.2	Unity	14
4.1.3	Aufbau einer Unity-Applikation	15
4.1.4	Lebenszyklusmethoden in Unity	15
4.1.5	Manager in Unity	16
4.2	Objektdesign mittels Blender	17
4.2.1	Rendering und Optimierung für AR	17
4.2.2	Export- und Integrationsprozess	18
4.2.3	Blender-Add-Ons und Plugins	19
4.3	Hauptmenu	19
4.3.1	Beschreiben der Buttons	19
4.3.2	Laden der Level	20
4.3.3	UI/UX	20
4.4	Ping Level	21
4.4.1	Object Tracking	21
4.4.2	Kurvenberechnung	21
4.5	Knapsack Problem Level	21
4.5.1	Knapsack-Problem Level Hirarchie	22
4.5.2	Verwendung von QR-Codes	24
4.5.3	Inventory Placement Game Objekt	25
4.5.4	Inventory Controller Game Objekt	31
4.5.5	Knapsack Algorithmus Definition	41
4.5.6	Knapsack Algorithmus Implementierung	42
4.5.7	Unit-Tests	47
4.6	Performance	47
5	Zusammenfassung und Abschluss	48
5.1	Ergebnis	48
5.2	Abnahme	48
5.3	Zukunft	48
A	Mockups	49
A.1	UI/UX	49
A.2	Hauptmenu Level Design	49
A.3	Ping-Paket Level Design	49
A.4	Knappsack-Level Design	49
B	Literatur	50

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 Betruer 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 Microsoft 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 Augmented Reality-Applikation auf Basis des AR-Devices HoloLens2 von Microsoft, die es ermöglicht ausgewählte technische Themenstellungen im Bereich Informatik (Visualisierung eines Pings, Veranschaulichung Knapack-Problem) für den Einsatz im Unterricht sowie beim Tag der offenen Tür visuell, interaktiv und spielerisch darzustellen.
------------	---

	HÖHERE TECHNISCHE BUNDES- LEHR- UND VERSUCHSANSTALT WIENER NEUSTADT
	Fachrichtung: Informatik Ausbildungsschwerpunkt: Softwareengineering

Typische Grafik, Foto etc. (mit Erläuterung)	Das vorliegende Bild stellt das Logo der AR-Applikation dar. <div data-bbox="742 922 1233 1142" data-label="Image"> </div>
---	---

Teilnahme an Wettbewerben, Auszeichnungen	
---	--

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

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

Diploma Thesis Documentation

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

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

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

Results	Planning, design, development and testing of 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.
---------	---

This image represents the logo of the AR application.



HTBLuVA Wiener Neustadt
Dr.-Eckener-Gasse 2
A 2700 Wiener Neustadt

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, das 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 Informationstechnik. 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 Levelinhalts 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 Netzkabel 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 Informationen des Items angezeigt werden. Die Aufgabe des Benutzers ist es mit den gegebenen Items das Inventar bestmöglich zu befüllen und dadurch den bestmöglichen 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 Menüpunkt entscheiden, ob er die perfekte Lösung sehen will oder nicht. Wenn sich der User dazu entscheidet die perfekte Lösung anzuzeigen, wird Vertikal über dem vormalen Inventar noch ein Inventar projiziert, dass das normale Inventar widerspiegelt aber mit 3D-Modelierten 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 Owner**²: 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 Master**³: 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.
- **Team**⁴: 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 Meeting**⁵: Im Sprint Planning Meeting wird das Ziel des folgenden Sprints definiert. Hierbei werden die Anforderungen im Product Backlog, die in diesem Sprint umgesetzt werden sollen, in einzelne Aufgaben zerlegt und anschließend im Sprint Backlog gesammelt.
- **Sprint**⁶: 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 Scrum**⁷: 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 Review**⁸: 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 Retrospective**⁹: 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 Projektablauf 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 stetz 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 Versionskontrollsystem 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 Fragebogenkonzeption

Die Konzeption und Gestaltung eines Fragebogens ist der wichtigste Schritt bei der Planung. Eine gut überlegte Planungsphase führt zu besseren Ergebnissen und dadurch auch eine leichtere Evaluierung. Folgende Entscheidung müssen daher schon im Vorfeld definiert und getroffen werden:

- Inhalt: evtl. bestehende Fragebögen verwenden oder anpassen.
- Umfang: Eher kurz halten (In Abhängigkeit von den Zielen).
- Ablauf und zeitlicher Rahmen: postalisch (längere Rücklaufzeit) oder elektronisch ¹⁰
- Zielgruppe: Vollbefragung oder Stichproben ¹¹

2.4.2 Abfassung der Fragen

Der Erfolg einer Umfrage benötigt eine genau Vorbereitung. Im Vorfeld muss klar sein, dass nur einzelne Ausschnitte eines Themengebietes behandelt werden können. Diese Ausschnitte müssen daher umso enger und genauer definiert werden. Hier ist daher vor allem die eindeutige Formulierung der Fragen wichtig.

im Vordergrund bei der Fragenformulierung stehen hier die Verständlichkeit bzw. die Unmissverständlichkeit. Folgende Regeln zur Formulierung sollen daher eingehalten werden:

- Einfache Wörter: Wörter, keine Fachausdrücke, anderssprachige Wörter oder Fremdwörter
- Formulierung: Möglichst kurz

¹⁰Vgl. Buehner S. -

¹¹Vgl. Mayer S. -

- Keine belastenden Wörter verwenden (z.B.: Ehrlichkeit, etc...)
- Keine hypothetischen Formulierungen
- Nur auf einen bestimmten Sachverhalt beziehen
- Keine Überforderung (Nicht zu viele Informationen auf einmal)
- Keine doppelten Verneinungen ¹²

Diese Kriterien gelten für eine schriftliche Befragung. Um das Resultat dieser Umfrage nicht zu verfälschen darf der Interviewer keine Extrafragen oder Umformulierungen an den gestellten Fragen tätigen.

2.4.3 Arten von Fragen

Je nach Anforderungsbedingungen wird zwischen einer offenen und geschlossenen Frage unterschieden ¹³

2.4.4 Struktur und Gliederung von Fragebögen

Hier wird verfasst wie die Allgemeine Struktur und Gliederung von Fragebögen aussehen soll. ¹⁴

2.4.5 Mögliche Verfälschung des Resultats

Welche Arten von Verfälschungen gibt es und diese Beschreiben Ursachen dafür beschreiben. ¹⁵

2.4.6 Auswertung von Fragebögen

Wie werten wir die Fragebögen aus? ¹⁶

¹²Vgl. Mayer S. -

¹³Vgl. Mayer S. -

¹⁴Vgl. Buehner S. -

¹⁵Vgl. Buehner S. -

¹⁶Vgl. Mayer S. -

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

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 vorallem 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 Spielelemente 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.

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 erschwerten 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 eine 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.

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

¹Microsoft **MRTK3**

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.

Microsoft OpenXR Plugin

Das Microsoft OpenXR Plugin ² 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.

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

²Khronos **OpenXR**

erstellten 3D-Modelle und Animationen in AR-Entwicklungsumgebungen wie Unity oder Unreal Engine importiert werden, um dort AR-spezifische Funktionalitäten hinzuzufügen.

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. Modellierungswerkzeuge 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.

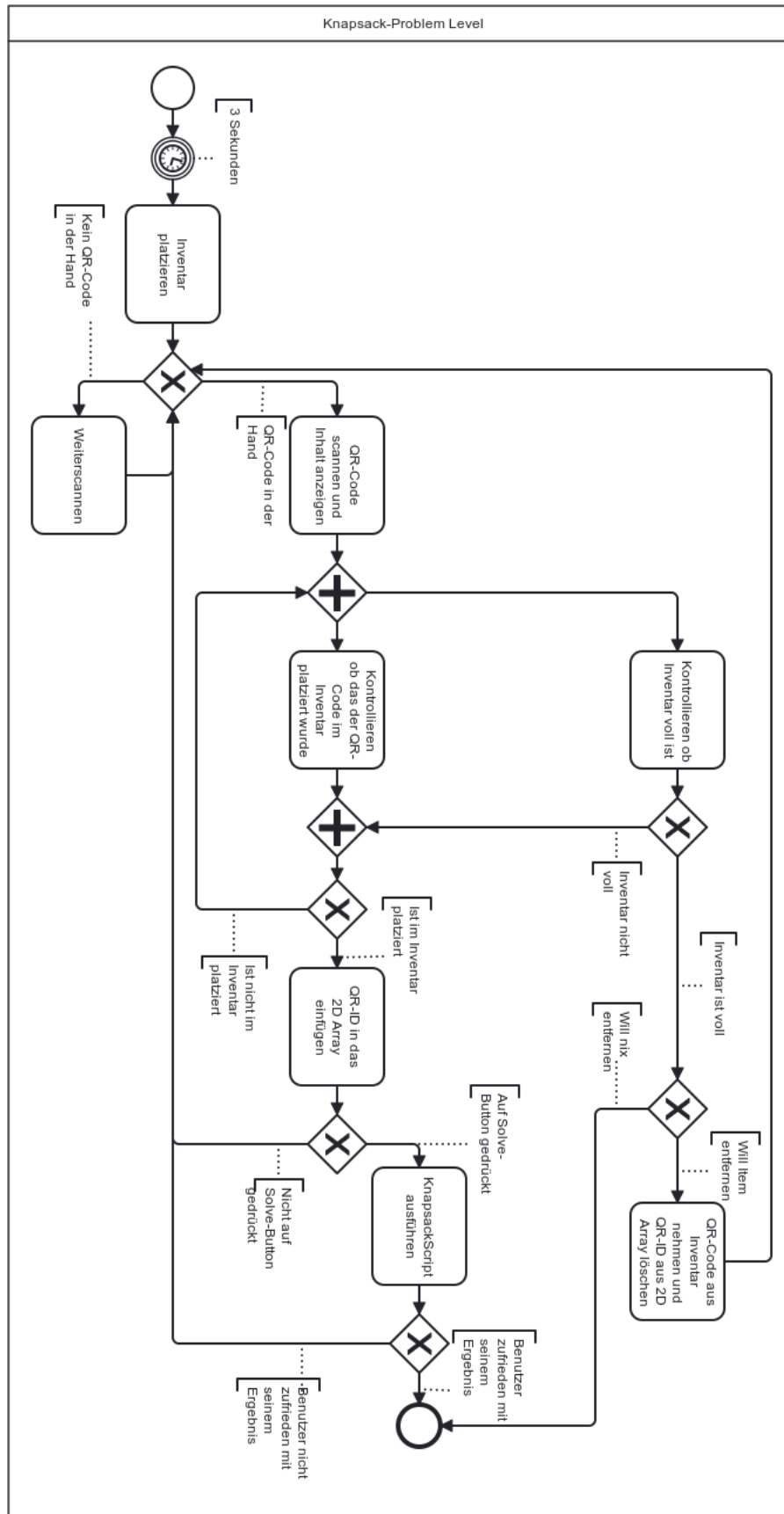


Abbildung 3.1: Ablaufdiagramm des Knapsack-Problem Levels

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 Texturoptimierung 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 LoopTools ³ und das Import-Export Add-On Images as Planes ⁴ verwendet.

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.

4.3 Hauptmenu

Das Hauptmenü dient als Implementierung des Basic UI/UX Systems. Nach umfassender Recherche und internen Abstimmungen im Team haben wir uns für die Umsetzung einer freibeweglichen Menüleiste entschieden, wie in Abbildung 4.1 dargestellt.

Diese Menüstruktur ermöglicht es dem Benutzer, sich im Raum frei zu bewegen, während das Menü stets auf Hüfthöhe mitfliegt.



Abbildung 4.1: Die Menüleiste besteht aus drei Buttons, einem Pin-Button und einer Grab-Bar.

4.3.1 Beschreiben der Buttons

Die Funktionalitäten der Buttons sind wie folgt:

³Blender **LoopTools**

⁴Blender **Images as Planes**

- **Debug-Button**

Der Debug-Button ist ausschließlich für Entwickler vorgesehen. Beim Aktivieren öffnet sich ein erweitertes Menü oberhalb des Hauptmenüs, das zusätzliche Funktionen für das Bugfixing während des Betriebs bereitstellt. Vorhandene Optionen umfassen einen Reset-Button für Level 2, der bei Problemen mit der Platzierung von Inventarobjekten verwendet werden kann.

- **Pin-Button**

Der Pin-Button ermöglicht das Fixieren des Menüs an einer geeigneten Stelle. Dies ist besonders nützlich, wenn der Benutzer sich an einen Tisch setzt und das Menü entsprechend positionieren möchte. Die Positionierung erfolgt mithilfe der Grab-Bar.

- **Grab-Bar**

Die weiße Leiste (Grab-Bar) kann festgehalten werden, um das Menü frei zu bewegen. In Kombination mit dem Pin-Button ist eine präzisere Platzierung möglich.

4.3.2 Laden der Level

Das Betätigen eines dieser Buttons (Level 1, Level 2) löst das Skript SceneChanger aus. Dieser Code wechselt die geladene Szene entsprechend. Die zu ladende Szene wird durch die Variable sceneToLoad definiert, die in Unity festgelegt wird und das gewünschte Level angibt.

```
1 using UnityEngine;
2 using UnityEngine.SceneManagement;
3 public class SceneChanger : MonoBehaviour
4 {
5     public string sceneToLoad;
6
7     // Assign this method to the button's onClick event in the Unity Editor.
8     public void ChangeScene()
9     {
10         SceneManager.LoadScene(sceneToLoad);
11         Debug.Log("Button clicked. Loading scene: " + sceneToLoad);
12     }
13 }
```

Listing 4.1: Auf Knopfdruck Szene wechseln.

4.3.3 UI/UX

Blickzentrierung und Interaktionsmodelle

Die Menüleiste bewegt sich auf Hüfthöhe 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 getrackt 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 getrackt.

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.

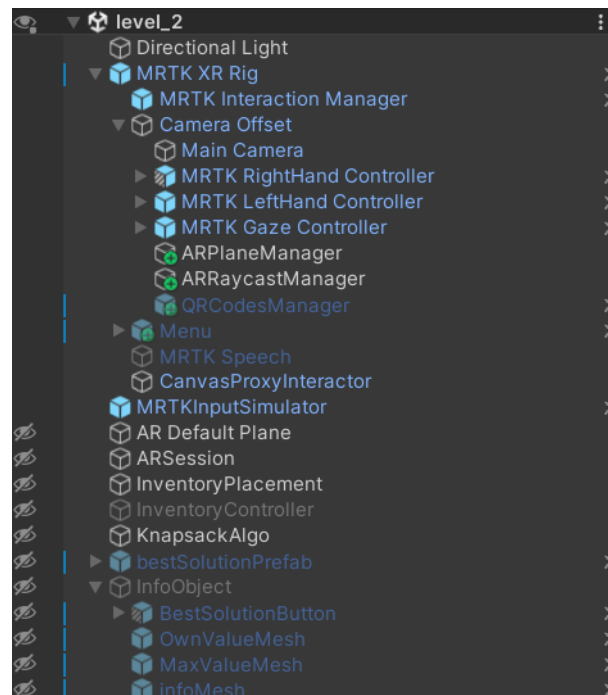


Abbildung 4.2: Knapsack-Problem Levels Hirarchie Unity Editor.

In dieser Abbildung ist die Hirarchie und der Inhalt des Knapsack-Problem Levels / Szene⁵ 2 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 Plane⁶ 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.
- **InventoryPlacement:** Ist ein Game Objekt, dem das *Knapsackscript.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.

⁵Unity **Scene**

⁶Unity **Plane**

- **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.
- **KnapsackAlgo:** GameObjekt, dem das *KnapsackAlgo.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 Prefab⁷ 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 QR Code 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.

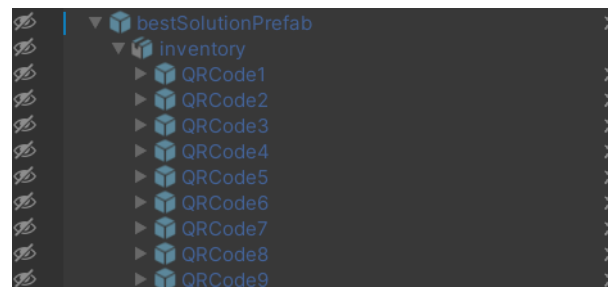


Abbildung 4.3: bestSolutionPrefab Objekt Hierarchie.

- **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

⁷Unity **Prefab**

man diejenige ausblendet die nicht im Editor sichtbar sein müssen wie zum Beispiel Tesh Meshes oder Lables.

4.5.2 Verwendung von QR-Codes

Im vorangegangenen Abschnitt wurde bereits darauf hingewiesen, dass QR-Codes in diesem Level verwendet werden, um die verschiedenen Items 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. Hierbei 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.

4.5.2.1 Inhalt der QR-Codes

Die Informationen, die in einem QR-Code gespeichert werden, 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.

4.5.2.2 QR-Code Tracking

Das Tracking der QR-Codes erfolgt mithilfe des *QRCodeManager.cs* Skripts. Dieses Klasse ist ein Singleton, das die Erkennung und Verfolgung der QR-Codes steuert. Das folgende UML-Diagramm zeigt die wichtigsten Methoden und zusammenhängenden Klassen: Nach der Erkennung eines QR-Codes erfolgen eine Reihe von Schritten, um diese Informationen zu speichern, verarbeiten und zuletzt darzustellen. Hier eine kurze Übersicht:

4.5.3 Inventory Placement 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 zu erfassen. Durch die Analyse der erfassten Daten können Ebenen identifiziert werden, die entscheidend sind, um eine genaue Platzierung von Inventargegenständen zu erreichen.

In diesem Abschnitt wird das *Inventory Placement* Game Objekt mit dem angehängten *AnchorScript.cs* behandelt. Dieses Script enthält die **PlacedObjectOnLookedAtDesk** Klasse, welche alle Funktionen für das Berechnen der Position und Platzierung des Inventars beinhaltet. Diese Funktionen werden in den nächsten Abschnitten detailliert erläutert.

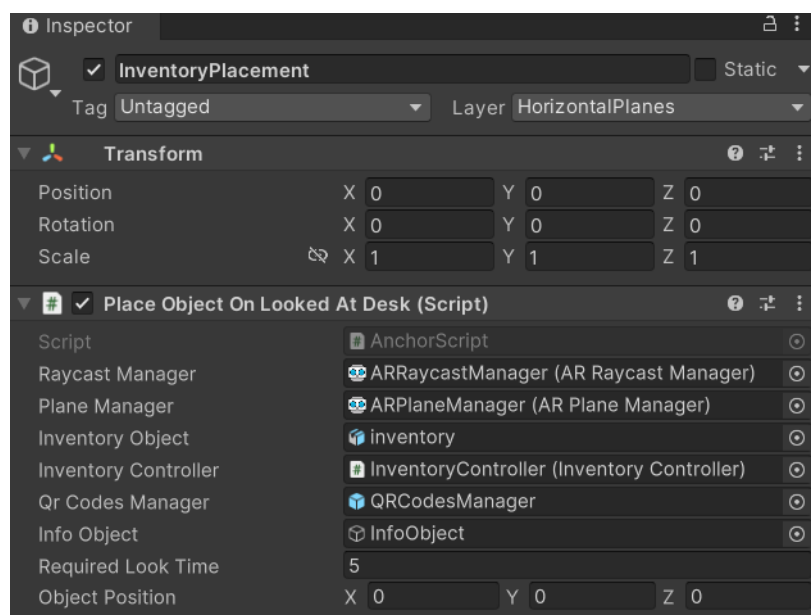


Abbildung 4.4: inventoryPlacement Objekt im Editor

Die Abbildung 4.4 zeigt das *inventoryPlacement*-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 *AnchorScript.cs*. Zudem sind vordefinierte Werte für bestimmte Variablen sichtbar. Diese Variablen umfassen:

- **Raycast Manager:** Das Game Objekt des ARRaycastManagers aus der Level 2 Szene.
- **Plane Manager:** Das Game Objekt des ARPlanesManagers aus der Level 2 Szene.
- **Inventory Object:** Das 3D-Modell des Inventars aus dem Prefab Ordner.
- **Inventory Controller:** Das Script des InventoryControllers.
- **QR Codes Manager:** Das Game Objekt des QRCodeManagers aus der Level 2 Szene.
- **Info Object:** Das Game Objekt des Info-Objekts aus der Level 2 Szene.
- **Required Look Time:** Die vorgeschriebene Zeit, die der Benutzer auf ein Plane schauen muss, damit das Inventar platziert wird.

Im Unity Editor können die Werte dieser Variablen direkt verändert werden, was Auswirkungen auf die Ausführung des Codes hat.

4.5.3.1 Klassenvariablen

```

1 public ARRaycastManager raycastManager;
2 public ARPlaneManager planeManager;
3 public GameObject inventoryObject;
4 public InventoryController inventoryController;
5 public GameObject qrCodesManager;
6 public GameObject infoObject;
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 4.2: Klassenvariablen der PlacedObjectOnLookedAtDesk Klasse

Die gezeigten Klassenvariablen in Codeabschnitt 4.2 sind Teil der *PlaceObjectOnLookedAtDesk* Klasse. Öffentliche (**public**) Variablen repräsentieren Objekte und Werte, die im Unity Editor festgelegt und übergeben werden. Dies ermöglicht einen direkten Zugriff auf diese Objekte im Code. Die Vektor-Variable *objectPosition* spielt eine entscheidende Rolle für die präzise Platzierung der perfekten Lösung. Die privaten (**private**) Variablen dienen der lokalen Speicherung von Werten, die von keiner anderen Klasse benötigt werden.

4.5.3.2 Startverzögerung

```

1 void Start()
2 {
3     StartCoroutine(DelayedStart());
```

4 }

Listing 4.3: Beginn des Anchor Scripts

Zu Beginn des Levels für das "Knapsack-Problem" wird die Lebenszyklusmethode *Start()* aufgerufen. Diese Funktion startet dann die Coroutine **DelayedStart()**. Die Startverzögerung wird hier verwendet, um dem *ARPlaneManager* ausreichend Zeit zu geben, um Ebenen in der Umgebung des Benutzers zu scannen und zu markieren. Dies ist von Bedeutung, um eine solide Grundlage für die spätere präzise Platzierung des Inventars zu gewährleisten.

```

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

```

Listing 4.4: Verzögerter Start

In der Funktion **DelayedStart()** wird eine Verzögerung von 3 Sekunden durchgeführt, bevor *canStartScript* auf true gesetzt wird. Diese Verzögerung ermöglicht es dem *ARPlaneManager*, ungestört die Umgebung zu scannen. Die Aktivierung von *canStartScript* markiert den Zeitpunkt, ab dem die *Update()* Methode ihre Ausführung fortsetzen kann. Dieser gestaffelte Startprozess gewährleistet eine reibungslose Erfassung von AR-Planes und legt somit einen soliden Grundstein für die präzise Platzierung von Inventargegenständen in der augmentierten Realität.

4.5.3.3 Frame-Aktualisierung zur Identifikation des gewünschten AR-Planes

Da in der Umgebung mehrere AR-Planes gescannt und markiert werden, ist es notwendig, ständig zu aktualisieren, welcher der markierten und gescannten Planes tatsächlich das beabsichtigte ist. Um dieses Verhalten zu erreichen, wird die Lebenszyklusmethode *Update()* verwendet. Diese Methode sorgt dafür, dass stets der gewünschte AR-Plane ausgewählt ist. Falls der Benutzer seinen Blick auf einen anderen, näher gelegenen Plane richtet, wird dieser als der neue ausgewählte Plane betrachtet.

```

1 void Update()
2 {
3     if (!objectPlaced && canStartScript)
4     {
5         List<ARRaycastHit> hits = new List<ARRaycastHit>();
6         % Use Camera.main.transform.forward as the ray direction
7         if (raycastManager.Raycast(new Ray(Camera.main.transform.position,
8             Camera.main.transform.forward), hits, TrackableType.Planes))
9         {
10             ARPlane closestPlane = FindClosestPlane(hits);
11             if (closestPlane != null)
12             {
13                 if (selectedDeskPlane == null || selectedDeskPlane !=
14                     closestPlane)
15                 {
16                     selectedDeskPlane = closestPlane;
17                     lookStartTime = Time.time; % Start the timer when a new
18                     plane is selected.
19                 }
20             }
21         }
22     }
23 }

```

```

16         }
17         float timeLookedAtPlane = Time.time - lookStartTime;
18         if (timeLookedAtPlane >= requiredLookTime)
19         {
20             PlaceObjectOnDesk(selectedDeskPlane);
21             objectPlaced = true;
22         }
23     }
24     else
25     {
26         selectedDeskPlane = null;
27     }
28 }
29 else
30 {
31     selectedDeskPlane = null;
32 }
33 }
34 }

```

Listing 4.5: Update Funktion

Die *Update()* Methode bildet das Herzstück des AR-Anker-Skripts, das die kontinuierliche Interaktion zwischen der realen und augmentierten Realität orchestriert. Der Ablauf beginnt mit einer sorgfältigen Prüfung, ob das Inventarobjekt noch nicht platziert wurde (*!objectPlaced*) und ob das Skript gestartet werden kann (*canStartScript*). Diese Bedingungen dienen dazu sicherzustellen, dass der Platzierungsprozess nur startet, wenn die Voraussetzungen erfüllt sind.

```

1 List<ARRaycastHit> hits = new List<ARRaycastHit>();
2 if (raycastManager.Raycast(new Ray(Camera.main.transform.position, Camera.main
   .transform.forward), hits, TrackableType.Planes))
3 {
4     % Weitere Verarbeitung der Treffer...
5 }

```

Listing 4.6: Raycasting

Sobald die Startbedingungen erfüllt sind, wird ein Raycasting durchgeführt, um die AR-Planes in der Umgebung zu identifizieren. Der Richtungsvektor des Rays wird durch *Camera.main.transform.forward* bestimmt.

```

1 ARPlane closestPlane = FindClosestPlane(hits);

```

Listing 4.7: Kürzest entfernte Plane

Der nächste Schritt beinhaltet die Bestimmung des am kürzesten entfernten AR-Planes durch Aufruf der Funktion **FindClosestPlane(hits)**. Diese Funktion vergleicht die Distanzen aller getroffenen AR-Planes und gibt den nächstgelegenen zurück.

```

1 if (selectedDeskPlane == null || selectedDeskPlane != closestPlane)
2 {
3     selectedDeskPlane = closestPlane;
4     lookStartTime = Time.time; % Starten des Timers, wenn ein neues Plane
   ausgewählt wird.

```



```
5 }
```

Listing 4.8: Plane auswählen und timer starten

Die ausgewählten AR-Plane werden daraufhin miteinander verglichen, und wenn ein neues AR-Plane ausgewählt wird oder der aktuelle AR-Plane sich geändert hat, wird die Zeitmessung (*lookStartTime*) gestartet. Diese Zeitmessung erfasst die Dauer, für die der Benutzer auf den aktuellen AR-Plane blickt.

```
1 float timeLookedAtPlane = Time.time - lookStartTime;
```

Listing 4.9: Blickzeit messen

Die gemessene Blickzeit (*timeLookedAtPlane*) wird in jedem Frame aktualisiert, indem die aktuelle verstrichene Zeit seit dem Start des Blicks auf den AR-Plane gemessen wird.

```
1 if (timeLookedAtPlane >= requiredLookTime)
2 {
3     PlaceObjectOnDesk(selectedDeskPlane);
4     objectPlaced = true;
5 }
```

Listing 4.10: Platzier-Funktion aufrufen

Wenn die gemessene Blickzeit die erforderliche Blickzeit überschreitet, wird die Funktion **PlaceObjectOnDesk(selectedDeskPlane)** aufgerufen, um das Inventarobjekt auf dem ausgewählten AR-Plane zu platzieren. Zusätzlich wird *objectPlaced* auf true gesetzt, um zu signalisieren, dass das Objekt platziert wurde und weitere Überprüfungen gestoppt werden sollen.

```
1 else
2 {
3     selectedDeskPlane = null;
4 }
```

Listing 4.11: Plane null setzen

Falls kein AR-Plane getroffen wurde, wird *selectedDeskPlane* auf null gesetzt, um sicherzustellen, dass keine vorherigen Auswahlinformationen beibehalten werden.

Dieser Ablauf setzt sich in jedem Frame fort, bis die vorgegebene Blickzeit erreicht ist und somit das Inventarobjekt erfolgreich auf dem AR-Plane platziert wurde.

4.5.3.4 Das am kürzesten entfernte Plane finden

```
1 ARPlane FindClosestPlane(List<ARRaycastHit> hits)
2 {
3     ARPlane closestPlane = null;
4     float closestDistance = float.MaxValue;
5     foreach (var hit in hits)
6     {
7         ARPlane plane = planeManager.GetPlane(hit.trackableId);
8         if (plane != null)
```

```

 9      {
10          float distanceToPlane = Vector3.Distance(Camera.main.transform.
position, hit.pose.position);
11          if (distanceToPlane < closestDistance)
12          {
13              closestPlane = plane;
14              closestDistance = distanceToPlane;
15          }
16      }
17  }
18  return closestPlane;
19 }

```

Listing 4.12: Kürzest entfernte Plane - Funktion

Die Funktion *FindClosestPlane* durchläuft eine Liste von *ARRaycastHit*-Objekten, die durch den Raycasting-Prozess erstellt wurden. Für jedes Trefferobjekt wird der zugehörige AR-Plane über den *planeManager* abgerufen. Dann wird die Entfernung von der Kameraposition zum Trefferpunkt des Rays auf dem AR-Plane berechnet.

Die Schleife sucht nach dem am nächsten gelegenen AR-Plane, indem sie die Distanz für jeden Treffer vergleicht. Das am nächsten gelegene AR-Plane (*closestPlane*) und die entsprechende Distanz (*closestDistance*) werden aktualisiert, wenn ein näherer Treffer gefunden wird. Am Ende wird das am nächsten gelegene AR-Plane zurückgegeben.

Diese Funktion wird in der Regel in AR-Anwendungen verwendet, um das AR-Plane zu identifizieren, auf das der Benutzer am längsten schaut, bevor eine Aktion ausgeführt wird.

Beispiel: Angenommen, es existieren drei AR-Planes in der Umgebung des Benutzers und die Liste *hits* enthält folgende Trefferpunkte:

- 1. Trefferpunkt auf Plane A: Entfernung = 2 Meter
- 2. Trefferpunkt auf Plane B: Entfernung = 1 Meter
- 3. Trefferpunkt auf Plane C: Entfernung = 3 Meter

Während des Durchlaufs durch die Trefferpunkte würde die Funktion *FindClosestPlane* in diesem Beispiel das AR-Plane B zurückgeben, da es die kürzeste Distanz von 1 Meter zur Kameraposition hat.

4.5.3.5 Berechnung der Position und Aktivierung/Deaktivierung sämtlicher Objekte

```

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

```

```

12     inventoryController.gameObject.SetActive(true);
13     planeManager.planePrefab.SetActive(false);
14     gameObject.SetActive(false);
15 }

```

Listing 4.13: Inventar platzier - Funktion

Diese Funktion übernimmt mehrere wichtige Aufgaben im Kontext der Augmented Reality-Anwendung die den sicheren weiteren Ablauf der Applikation versichert.

1. **Aktivierung des QR-Code Managers:** Die Funktion startet damit, den QR-Code Manager zu aktivieren, was das Tracken von QR-Codes ermöglicht.
2. **Berechnung der Objektposition und Rotation:** Anschließend wird die Position des Inventar-Objekts berechnet. Dazu wird die Flächenmitte des ausgewählten AR-Planes genutzt, und die Rotation des Objekts wird auf der x-Achse festgelegt.
3. **Instantiierung des Objekts:** Das Inventar-Objekt wird dann durch die Verwendung der **Instantiate**-Funktion erstellt und auf eine bestimmte Größe skaliert.
4. **Platzierung des Info-Objects:** Die Position des Info-Objects, das aus TextMeshes und Buttons besteht, wird ebenfalls berechnet und aktiviert, um für den Benutzer sichtbar zu sein.
5. **Setzen des Inventar-Objekts im InventoryController:** Das platzierte Inventar-Objekt wird an den *InventoryController* weitergeleitet und dort als aktuelles Inventarobjekt festgelegt um sicherzustellen, dass hier dann in Folge auf das selbe Objekt zugegriffen wird.
6. **Aktivierung des InventoryControllers:** Nach der erfolgreichen Platzierung des Inventar-Objekts und Aktivierung des Info-Objects wird der *InventoryController* aktiviert, um die Inventarverwaltung zu starten.
7. **Sichtbarkeit der Planes ausschalten:** Da die AR-Planes nach erfolgreicher Platzierung der Objekte nicht mehr sichtbar sein sollen, weil sie nichtmehr gebraucht werden, werden sie mit *planeManager.planePrefab.SetActive(false)* deaktiviert.
8. **Deaktivierung des Skripts:** Schließlich wird das *AnchorScript.cs* deaktiviert, um weitere ressourcenintensive Aktualisierungen zu stoppen.

Dieser Ablauf stellt sicher, dass nach der Erkennung und Auswahl eines AR-Planes das Inventar-Objekt präzise platziert wird und alle notwendigen Elemente aktiviert bzw. deaktiviert werden, um eine nahtlose Integration von realer und augmentierter Realität zu gewährleisten.

4.5.4 Inventory Controller Game Objekt

Das *Inventory Controller* Game Objekt spielt eine entscheidende Rolle bei der Überwachung und Verwaltung des Inventars in der Augmented Reality (AR)-Anwendung. Durch das angehängte *InventoryController.cs* Script wird die **InventoryController** Klasse realisiert. Diese Klasse ist verantwortlich für die Erkennung neuer Gegenstände im Inventar, die Überprüfung auf verfügbaren Platz im Inventar, die Berechnung der Zellenposition anhand der QR-Code-Koordinaten und die Speicherung der Positionen dieser Items in einem *2D Array*.

Der Zugriff auf die Begrenzungen (*Bounds*) des Inventarmodells ermöglicht es, ständig zu überwachen, 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 *InventoryController.cs* Script interagiert dabei aktiv mit den AR-Elementen, insbesondere den QR-Codes, um den Platzbedarf der Gegenstände zu überwachen und ihre

Positionen im Inventar festzulegen. Zusätzlich interagiert der *InventoryController* auch mit dem *KnapsackAlgo* Game Objekt, in dem es das verspeicherte *2D Array* immer weitergibt um damit rechnen zu können.

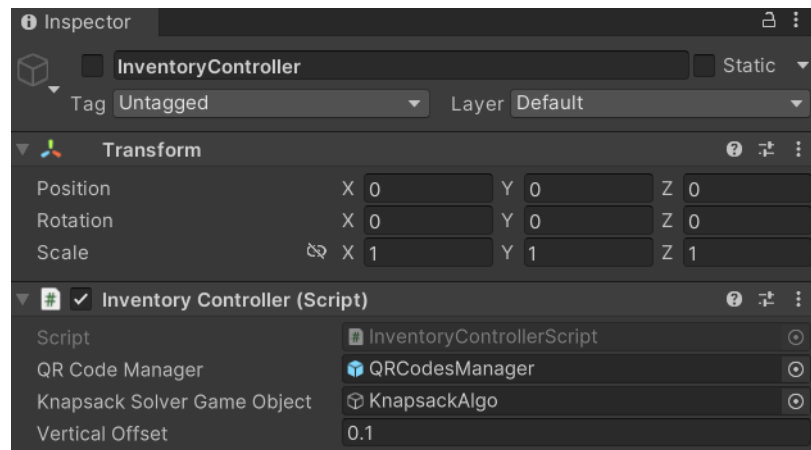


Abbildung 4.5: InventoryController Objekt im Editor

Die Abbildung 4.5 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:

- **QRCodeManager:** Das QRCodeManager Game Objekt aus der Level 2 Szene.
- **Knapsack Solver Game Objekt:** Das KnapsackAlgo Game Objekt aus der Level 2 Szene.
- **Vertical Offset:** Float Wert um den die Bounds⁸ des Inventory Objekts auf der X-Achse erweitert werden.

4.5.4.1 InventoryController Klassenvariablen

```

1 void PlaceObjectOnDesk(ARPlane deskPlane)
2 {
3     public GameObject QRCodeManager;
4     public GameObject knapsackSolverGameObject;
5
6     private SortedDictionary<System.Guid, GameObject> activeQRObjects;
7
8     private GameObject inventoryObject;
9     public float verticalOffset = 0.1f;
10    private Bounds inventoryBounds;
11    private int numRows = 3;
12    private int numColumns = 3;
13    private int[,] idGrid;
14    private KnapsackScript knapsackScript;
15    private int cap;

```

⁸Unity Bounds

```
16     private int currWeight = 0;
17     private string message;
18     private HashSet<int> processedItems;
19 }
```

Listing 4.14: Klassenvariablen der InventoryController Klasse

Die gezeigten Klassenvariablen in Codeabschnitt 4.14 sind Teil des *InventoryController* Klasse. Öffentliche(**public**) Variablen repräsentieren Objekte und Werte, die im Unity Editor festgelegt und übergeben werden. Dies ermöglicht einen direkten Zugriff auf diese Objekte im Code. Die float-Variable *verticalOffset* spielt hier eine wichtige Rolle, für die Erweiterung der *Bounds* von dem Inventar Modell. Die privaten (**private**) Variablen dienen der lokalen Speicherung von Werten, die von keiner anderen Klasse benötigt werden.

4.5.4.2 Start des InventoryControllers

```
1 void Start()
2 {
3     activeQRObjects = QRCodeManager.GetComponent<QRCodesVisualizer>().
qrCodesObjectsList;
4     knapsackScript = knapsackSolverGameObject.GetComponent<KnapsackScript>();
5     cap = knapsackScript.capacity;
6     processedItems = new HashSet<int>();
7     UpdateInventoryBounds();
8     InitializeIDGrid();
9 }
```

Listing 4.15: Start Funktion des InventoryControllers

Die **Start()** Funktion wird zu Beginn des Inventory Controllers aufgerufen und aufgeführt. Hier werden die wichtigen Objekte die für den weiteren Verlauf des Programms notwendig sind deklariert beziehungsweise erstellt und ebenfalls werden die beiden Funktionen **UpdateInventoryBounds()** und **InitializeIDGrid()** aufgerufen.

4.5.4.3 Neue Bounds setzen

```
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 4.16: Funktion um Inventar Bounds zu erweitern

Um zu erreichen, dass man ein Item wirklich in die Bounds des Inventars legen kann wird in der **UpdateInventoryBounds()** Funktion genau dies ausgeführt. Es werden die aktuellen Bounds des Inventars gespeichert und diese werden dann mittels der **ExtendBounds()** Funktion erweitert und anschließend werden diese Bounds als neue Bounds gespeichert.

4.5.4.4 Bounds des Inventars ermitteln

```

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

```

Listing 4.17: Funktion um Bounds zu ermitteln

Die von 4.16 aufgerufene Funktion **GetBounds** greift auf den **Renderer**⁹ des Inventar Modells zu um dadurch auf die Bounds zuzugreifen zu können, denn der **Renderer** ist dafür verantwortlich, dass das Modell überhaupt sichtbar ist. Wenn dieser **Renderer** ungleich **null** ist, gibt er die neuenn Bounds zurück und speichert sie in *localBounds* von Codeabschnitt 4.16.

4.5.4.5 Inventory Bounds erweitern

```

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

```

Listing 4.18: Funktion um Bounds zu erweitern

Um anschließend diese Bounds zu erweitern um zu erkennen ob QR-Koordinaten innerhalb von diesen Bounds ist wird die von 4.16 aufgerufene Funktion **ExtendBounds()** ausgeführt. Diese Funktion erzielt, dass die Bounds von dem Inventar Modell auf der X-Achse um das *Vertical Offset* erweitert werden. Nach Abschluss dieser Funktion sehen die Bounds von diesem Modell dann folgendermaßen aus:

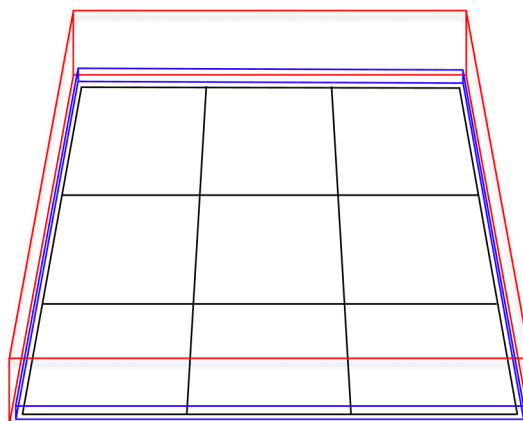


Abbildung 4.6: Erweiterte Bounds des Inventar Modells

Auf der Abbildung 4.6 ist eine einfache schematische Zeichnung des zweidimensionalen Inventars zu sehen. Die blaue Umrandung repräsentiert die Begrenzungen (Bounds) vor

⁹Unity **Renderer**

der Ausführung der Funktion **UpdateInventoryBounds()** aus dem Codeabschnitt 4.16. Nachdem die Funktion ausgeführt wurde, wird die rote Umrandung dargestellt. Durch die neuen Bounds wird deutlich, dass insbesondere die Begrenzung entlang der X-Achse erweitert wurde.

Die Erweiterung der Bounds ist von entscheidender Bedeutung, da nicht erweiterte Bounds die Höhe der Bauklötze nicht berücksichtigen würden. Dies könnte dazu führen, dass ein QR-Code, der sich tatsächlich innerhalb der Bounds befindet, als außerhalb betrachtet wird, wenn die Höhe der Bauklötze nicht angemessen einbezogen 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.

4.5.4.6 ID Grid Initialisierung

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

Listing 4.19: Initialisierung des idGrids

Die in Codeabschnitt 4.15 aufgerufene Funktion initialisiert das idGrid mit *numRows* und *numColumns* also wird das idGrid als *int [3,3]* initialisiert.

4.5.4.7 Update Funktion

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

Listing 4.20: Initialisierung des idGrids

Nach Abschluss des Codes aus Codeabschnitt 4.15 wird anschließend die *Lebenszyklusmethode* **Update()** ausgeführt. Diese Funktion ruft jeden Frame die **UpdateGrid()** Funktion auf, auf die im nächsten Abschnitt genau eingegangen wird.

4.5.4.8 Funktion für neue Items innerhalb des Bounds

```
1 void UpdateGrid()
2 {
3     lock (activeQRObjects)
4     {
5         foreach (var item in activeQRObjects.Values)
6         {
7             QRCode qrCode = item.GetComponent<QRCode>();
8             Vector3 worldPosition = item.transform.TransformPoint(qrCode.item.
qrData.position);
9
10            if (item != null && inventoryBounds.Contains(worldPosition))
```

```

11         {
12             int itemId = qrCode.item.qrData.id;
13
14             if (!processedItems.Contains(itemId))
15             {
16                 if (currWeight + qrCode.item.qrData.weight <= cap)
17                 {
18                     processedItems.Add(itemId);
19                     message = " ";
20                     Vector2 startGridPosition = CalculateGridPosition(
worldPosition);
21                     idGrid[(int)startGridPosition.x, (int)
startGridPosition.y] = itemId;
22                     knapsackScript?.UpdateInfoMesh(message);
23                     currWeight += qrCode.item.qrData.weight;
24                     EventManager.GridUpdate(idGrid);
25                 }
26                 else
27                 {
28                     message = "Item hat zu viel Gewicht!";
29                     knapsackScript?.UpdateInfoMesh(message);
30                 }
31             }
32         }
33         else if (!inventoryBounds.Contains(worldPosition) &&
processedItems.Contains(qrCode.item.qrData.id) && ContainsId(qrCode.item.
qrData.id))
34         {
35             int itemId = qrCode.item.qrData.id;
36             processedItems.Remove(itemId);
37             RemoveItem(itemId);
38             currWeight -= qrCode.item.qrData.weight;
39             EventManager.GridUpdate(idGrid);
40         }
41     }
42     PrintGrid();
43 }
44 }

```

Listing 4.21: Code fuer ueberpruefen neuer Items im Inventar

Die vorliegende Funktion, `UpdateGrid()`, ist maßgeblich für die Überwachung und Verwaltung von neu hinzugefügten oder entfernten Items im Inventar verantwortlich. Zu Beginn eines jeden Frames wird das `activeQRObjects`-Dictionary gesperrt, um Thread-Sicherheit zu gewährleisten. Anschließend erfolgt eine Iteration durch jedes Element in diesem Dictionary mittels einer `foreach`-Schleife.

Innerhalb dieser Schleife wird für jedes Objekt die zugehörige `QRCode`-Komponente extrahiert, und die Weltposition des Objekts wird durch Transformation der lokalen Position des QR-Codes unter Berücksichtigung der Transformation des übergeordneten Objekts berechnet.

Die Hauptüberprüfung erfolgt daraufhin, ob das aktuelle Item nicht null ist und ob seine Weltposition innerhalb der durch `inventoryBounds` definierten Grenzen liegt. Wenn diese Bedingungen erfüllt sind, wird das Item als "hinzugefügt" betrachtet.

Im weiteren Verlauf wird überprüft, ob die ID des Items bereits in der Menge der verarbeiteten Items (**processedItems**) vorhanden ist. Falls nicht, wird überprüft, ob das Hinzufügen des Items das aktuelle Gesamtgewicht (**currWeight**) zusammen mit dem Gewicht des Items unter die maximale Kapazität (**cap**) bringt.

Wenn diese Bedingungen erfüllt sind, wird das Item in die Menge der verarbeiteten Items aufgenommen, eine Nachricht wird auf Leer gesetzt, und die Position des Items im zweidimensionalen Array **idGrid** wird aktualisiert. Zusätzlich wird die Anzeige des Inventars durch den Aufruf der Methode **UpdateInfoMesh** des **knapsackScript** aktualisiert, und das aktuelle Gewicht des Inventars wird entsprechend angepasst. Schließlich wird ein Ereignis (**GridUpdate**) ausgelöst, um andere Teile des Systems über die Aktualisierung des Grids zu informieren.

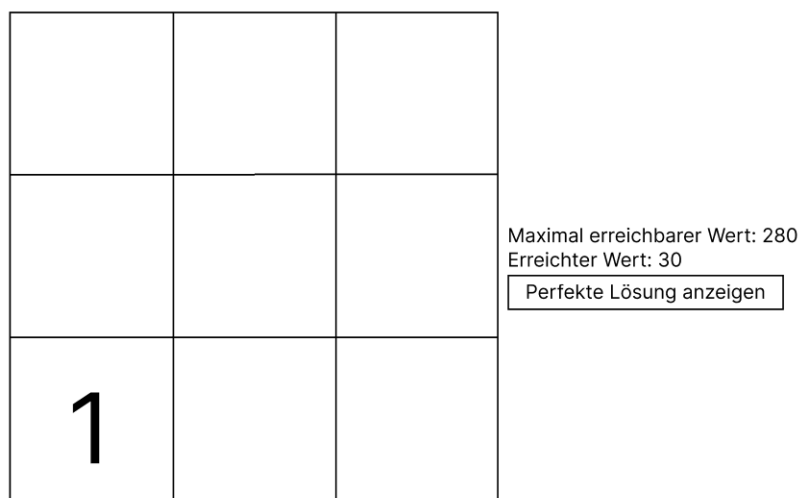


Abbildung 4.7: Item zu Inventar hinzugefügt

In Abbildung 4.7 ist der Zustand nach dem Hinzufügen eines Items dargestellt. In Zelle 7 wurde das Item mit der ID: 1 hinzugefügt. Nach dem Hinzufügen des Items wird der *Knapsack Algorithmus* ausgelöst und anschließend die beiden TextMeshes aktualisiert, indem in dem oberen TextMesh der maximal erreichbare Wert angezeigt wird und in dem unteren der Wert des selbst zusammengestellten Inventars.

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

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

Listing 4.22: ID gespeichert

Das Hashset *processedItems* sieht nach platzieren des Items dann so aus:

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

Listing 4.23: ID gespeichert

Sollte das Gewicht des hinzugefügten Items die Kapazität überschreiten, wird eine Fehlermeldung generiert, und die Aktualisierung des Grids wird unterbunden. Die Fehlermeldung und der resultierende Zustand des Grids sind in Abbildung 4.8 dargestellt.



Abbildung 4.8: Item zu schwer

In Abbildung 4.8 sind vier verschiedene Items im Inventar enthalten, mit den IDs 1, 3, 5, 9. Die Reihenfolge ihres Hinzufügens zum Inventar war 1, 5, 9, 3. Das Item mit der ID 9 wurde jedoch aufgrund seines zu hohen Gewichts nicht zu *idGrid* und *processedItems* hinzugefügt. Gleichzeitig wird eine Fehlermeldung angezeigt, um dem Benutzer zu signalisieren, dass das zuletzt hinzugefügte Item 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 int idGrid = [[0, 0, 0],
2               [5, 0, 0],
3               [1, 3, 0]];
```

Listing 4.24: Item zu schwer für das Inventar

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

```
1 processedItems = [1, 5, 9];
```

Listing 4.25: ID gespeichert

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

Im alternativen Zweig der Hauptbedingung wird überprüft, ob das Item außerhalb der Inventargrenzen liegt, aber zuvor als verarbeitet markiert wurde und weiterhin in *processedItems* und *idGrid* vorhanden ist. Dies deutet darauf hin, dass das Item aus dem Inventar entfernt wurde. In diesem Fall wird die ID des Items aus *processedItems* und *idGrid* entfernt, und das Gewicht des Inventars wird entsprechend angepasst. Auch hier wird ein *GridUpdate*-Ereignis ausgelöst.

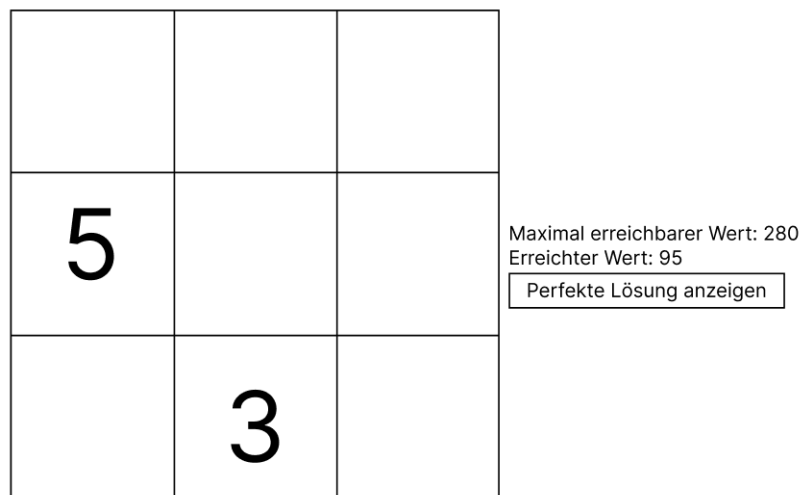


Abbildung 4.9: Item aus Inventar entfernt

In Abbildung 4.9 ist der aktuelle Zustand eines selbst zusammengestellten Inventars zu sehen. In diesem Inventar sind die beiden Items mit den IDs 5, 3 enthalten. Im Hintergrund sieht das *idGrid* jedoch so aus:

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

Listing 4.26: ID trotz nicht vorhandenem Item gespeichert

Und das HashSet *processedItems* sieht so aus:

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

Listing 4.27: ID trotz nicht vorhandenem Item gespeichert

Anhand der Abbildung 4.9 und den beiden Codeabschnitten 4.26 und 4.27 ist aber zu sehen, dass hier noch die ID 1 gespeichert ist. Dies bedeutet, dass das Item mit der ID 1 in einem vorherigen Zustand im Inventar platziert wurde aber jetzt außerhalb der Bounds des Inventars liegt. Anhand dieser Erkenntnis wird anschließend diese ID aus dem *idGrid* und *processedItems* entfernt und das *currWeight* angepasst. Diese sehen nach der Entfernung dieser ID folgendermaßen aus:

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

Listing 4.28: ID trotz nicht vorhandenem Item gespeichert

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

Listing 4.29: ID verspeichert

Schließlich erfolgt der Aufruf der Funktion **PrintGrid()**, die Testweise für das Debugging enthalten ist um aus den Logs herauslesen zu können, wie das *idGrid* aufgebaut ist und welche Items enthalten sind und ob die Berechnung und Platzierung richtig abgelaufen ist.

4.5.4.9 Berechnen der Position im *idGrid* anhand der Koordinaten

In Codeabschnitt 4.21 wird bei platzieren eines Items innerhalb des Inventars die Funktion **CalculateGridPosition()** aufgerufen. Diese Funktion kümmert sich darum, dass anhand der gegebenen Koordinaten des Items innerhalb des Inventar Modells ein Vektor berechnet wird an welcher Stelle dieses Item im *idGrid* liegt. Diese Funktion sieht folgendermaßen aus:

```

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) /
6     cellWidth);
7     int row = Mathf.FloorToInt((inventoryBounds.max.z - objectPosition.z) /
8     cellHeight);
9     return new Vector2(row, col);
10 }

```

Listing 4.30: Stelle im *idGrid* anhand Koordinaten berechnen

Die Funktion beginnt damit, dass sie sich anhand der Länge der Bounds in X- und Y-Achse die Länge einer einzelnen Zelle berechnet. Die Variable `cellWidth` repräsentiert die Breite jeder Zelle, und `cellHeight` repräsentiert die Höhe jeder Zelle im Inventar-Raster.

Anschließend werden die 2D-Koordinaten der Position des übergebenen Objekts im Inventar-Raster berechnet. Dazu wird zuerst die X-Koordinate des Objekts relativ zur minimalen X-Grenze der Inventar-Bounds genommen und durch die Breite einer Zelle (`cellWidth`) geteilt. Dieser Wert wird dann auf die nächste ganze Zahl abgerundet (`Mathf.FloorToInt`) und repräsentiert die Spalte (`col`) im Raster.

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

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

4.5.4.10 Hilfsfunktionen

Im Codeabschnitt 4.21 werden die Funktionen **RemoveItem** und **ContainsID** für die Entfernung eines Items aus dem *idGrid* und die Überprüfung, ob eine bestimmte Item-ID im *idGrid* vorhanden ist, verwendet. Die Funktion **RemoveItem** hat den Zweck, ein Item anhand seiner ID aus dem *idGrid* zu entfernen. Die Funktion **ContainsID** wird verwendet, um zu überprüfen, ob eine bestimmte Item-ID bereits im *idGrid* vorhanden ist. Die Implementierungen dieser Funktionen sind entsprechend:

```

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 }

```

```
12     }  
13 }  
14 }
```

Listing 4.31: ID aus idGrid 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 4.32: Ueberpruefen ob ID in idGrid enthalten ist

4.5.5 Knapsack Algorithmus Definition

Im folgenden Abschnitt wird der Knapsack-Algorithmus allgemein erklärt, die Problemstellung verdeutlicht und die Unterschiede der beiden möglichen Algorithmen erläutert. Außerdem wird darauf eingegangen, warum er so wichtig ist und in welchen Anwendungen er zum Einsatz kommt.

Der Knapsack-Algorithmus ist ein häufig verwendetes Werkzeug in der Informatik, um das Problem der optimalen Ressourcenallokation zu lösen. Dieses Problem tritt auf, wenn eine begrenzte Menge an Ressourcen so effizient wie möglich genutzt werden soll, um einen bestimmten Nutzen oder Gewinn zu maximieren. Der Begriff "Knapsack" leitet sich von der Idee ab, dass man versucht, einen Rucksack mit begrenztem Fassungsvermögen mit Gegenständen zu füllen, die unterschiedliche Werte und Gewichte haben.

4.5.5.1 Problemstellung

Gegeben sei ein Rucksack mit begrenzter Kapazität und eine Menge von Gegenständen, von denen jeder einen bestimmten Wert und ein bestimmtes Gewicht hat. Das Ziel besteht darin, die Gegenstände auszuwählen, die in den Rucksack passen und den Gesamtwert maximieren.

4.5.5.2 Algorithmus

Der Knapsack-Algorithmus kann in verschiedenen Varianten implementiert werden, darunter der dynamische Programmieransatz und der Greedy-Ansatz. Im dynamischen Programmieransatz wird eine Tabelle erstellt, um Teilprobleme zu lösen und die optimale Lösung zu berechnen. Der Greedy-Ansatz hingegen wählt Gegenstände basierend auf bestimmten Kriterien aus, um eine lokale Optimierung zu erreichen.

In unserem Kontext wurde der dynamische Programmieransatz implementiert, um den Knapsack-Algorithmus umzusetzen. Der Code dafür ist in der Datei *KnapsackScript.cs* zu finden, welche den maximal erreichbaren Wert und die optimale Lösung berechnet.

4.5.5.3 Anwendungen

Der Knapsack-Algorithmus findet Anwendung in verschiedenen Bereichen, darunter Logistik, Finanzplanung, Ressourcenmanagement und Netzwerkoptimierung. Beispielsweise kann er verwendet werden, um den effizientesten Transport von Gütern mit begrenzten Kapazitäten zu planen oder in der Finanzplanung, um das Portfolio von Investitionen zu optimieren.

4.5.6 Knapsack Algorithmus Implementierung

Der folgende Abschnitt beschreibt das *KnapsackScript.cs* und den darin geschriebenen Code für die Umsetzung des dynamischen Knapsack-Algorithmus in C-Sharp unter Verwendung von Unity. Die Implementierung umfasst den Funktionsaufruf, die Initialisierung der Variablen für den Algorithmus, den eigentlichen Algorithmus, das Backtracking zur Identifizierung der ausgewählten Gegenstände für die perfekte Lösung, die Berechnung der Gruppengröße, die Konvertierung der temporären Liste in ein 2D-Array, die Rückgabe des Gesamtwerts, sowie die Darstellung der perfekten Lösung und die Berechnung des eigenen Inventars.

4.5.6.1 Funktionsaufruf

```
1 int maxValue = KnapsackMaxValue(out usedItems, out int coveredCapacity);
2 int inventoryValue = -1;
3 maxMesh.text = "Maximal erreichbarer Wert: " + maxValue.ToString();
```

Erklärung: In diesem Codeabschnitt erfolgt der Aufruf der Funktion *KnapsackMaxValue*. Die beiden *out*-Parameter (*usedItems* und *coveredCapacity*) dienen dazu, Werte aus der Funktion herauszugeben, und werden nach dem Funktionsaufruf mit den berechneten Werten aktualisiert. Der Rückgabewert *maxValue* enthält den maximal erreichbaren Wert des Rucksacks. Dieser Wert wird dann im TextMesh *maxMesh* dargestellt, um dem Benutzer die Information über den maximal erreichbaren Wert anzuzeigen.

4.5.6.2 Initialisierung der Variablen für den Algorithmus

```
1 int n = items.Count;
2 int[,] dp = new int[n + 1, capacity + 1];
3 bool[,] selected = new bool[n + 1, capacity + 1];
```

Erklärung: Die Initialisierung erstellt die notwendigen Arrays für die dynamische Programmierung.

4.5.6.3 Dynamischer Knapsack Algorithmus

```
1 for (int i = 0; i <= n; i++)
2 {
3     for (int w = 0; w <= capacity; w++)
4     {
5         % Initialisierung der DP-Tabelle
6         if (i == 0 || w == 0)
7             dp[i, w] = 0;
8         else if (items[i].weight <= w && i <= maxItems)
```

```

9      {
10         % Berechnung des neuen Werts, wenn das Item ausgewählt wird
11         int newValue = items[i].value + dp[i - 1, w - items[i].weight];
12         if (newValue > dp[i - 1, w])
13         {
14             dp[i, w] = newValue;
15             selected[i, w] = true;
16         }
17         else
18         {
19             dp[i, w] = dp[i - 1, w];
20             selected[i, w] = false;
21         }
22     }
23     else
24     {
25         dp[i, w] = dp[i - 1, w];
26         selected[i, w] = false;
27     }
28 }
29 }

```

Erklärung:

1. **Iterationen durch die Items und Kapazität:** Die äußere Schleife durchläuft jedes Item (i) von 0 bis zur Gesamtanzahl der Items (n). Die innere Schleife durchläuft jede mögliche Kapazität (w) von 0 bis zur maximalen Kapazität des Rucksacks.
2. **Initialisierung der DP-Tabelle:** Wenn i oder w 0 ist, wird der Wert in der DP-Tabelle auf 0 gesetzt. Dies liegt daran, dass es keine Items gibt oder die Kapazität des Rucksacks 0 ist.
3. **Überprüfung, ob das Item ausgewählt werden kann:** Wenn das aktuelle Item ausgewählt werden kann (basierend auf Gewicht und maximaler Anzahl von Items), wird der neue Wert berechnet, wenn das Item ausgewählt wird.
4. **Vergleich und Auswahl:** Der neue Wert wird mit dem Wert verglichen, wenn das Item nicht ausgewählt wird. Wenn der neue Wert größer ist, wird das Item ausgewählt (`selected[i, w] = true`), andernfalls wird das vorherige Ergebnis beibehalten (`selected[i, w] = false`).
5. **Falls Item nicht ausgewählt wird:** Wenn das aktuelle Item nicht ausgewählt wird, wird der Wert in der DP-Tabelle auf den Wert gesetzt, den das Rucksack ohne das aktuelle Item in der vorherigen Iteration erreicht hat.

Bedeutung der DP-Tabelle: Die DP-Tabelle (`dp`) dient dazu, Teilprobleme zu lösen und die besten Ergebnisse für verschiedene Kombinationen von Items und Kapazitäten zu speichern. Jeder Eintrag in der Tabelle repräsentiert den maximal erreichbaren Wert für eine bestimmte Menge von ausgewählten Items und eine bestimmte Kapazität des Rucksacks.

Insgesamt ermöglicht dieser Algorithmus, die optimale Auswahl von Gegenständen zu finden, die in den Rucksack passt, um den Gesamtwert zu maximieren.

4.5.6.4 Backtracking, um ausgewählte Items und perfekt Lösung zu finden

```

1 List<List<int>>> tempUsedItems = new List<List<int>>>();

```

```

2 int row = n;
3 int col = capacity;
4 while (row > 0 && col > 0)
5 {
6     List<int> group = new List<int>();
7     while (row > 0 && col > 0 && selected[row, col])
8     {
9         // Item zu ausgewählter Gruppe hinzufügen
10        group.Add(items[row].id);
11        col -= items[row].weight;
12        row--;
13    }
14
15    // Hinzufügen der Gruppe zur Liste
16    if (group.Count > 0)
17    {
18        tempUsedItems.Add(group);
19    }
20
21    // Fortfahren mit dem Backtracking
22    if (row > 0 && col > 0)
23    {
24        row--;
25    }
26 }

```

Erklärung:

1. **Initialisierung von Zeilen- und Spaltenindizes:** Die Variablen `row` und `col` werden auf die letzte Zeile und Spalte der DP-Tabelle gesetzt.
2. **Hauptschleife für Backtracking:** Die äußere Schleife wird durchlaufen, solange die Zeile und die Spalte größer als 0 sind.
3. **Innere Schleife für Gruppenbildung:** Solange die Zeile und die Spalte größer als 0 sind und das Item in der DP-Tabelle ausgewählt wurde (`selected[row, col]`), wird das Item zur aktuellen Gruppe hinzugefügt, das Gewicht von der aktuellen Kapazität abgezogen, und die Zeile wird dekrementiert.
4. **Hinzufügen der Gruppe zur Liste:** Wenn die Gruppe mindestens ein Item enthält, wird die Gruppe zur temporären Liste `tempUsedItems` hinzugefügt.
5. **Fortfahren mit dem Backtracking:** Die Zeile wird dekrementiert, um das nächste ausgewählte Item zu finden.

Diese Schritte werden wiederholt, bis alle ausgewählten Items gefunden sind und die temporäre Liste `tempUsedItems` alle ausgewählten Gruppen enthält.

4.5.6.5 Gruppengröße berechnen

```

1 int maxGroupSize = 0;
2 foreach (var group in tempUsedItems)
3 {
4     if (group.Count > maxGroupSize)
5         maxGroupSize = group.Count;
6 }

```


Erklärung: Findet die maximale Gruppengröße.

4.5.6.6 Temporäres Array umwandeln und das usedItems Array befüllen

```

1 usedItems = new int[tempUsedItems.Count, maxGroupSize];
2 for (int i = 0; i < tempUsedItems.Count; i++)
3 {
4     for (int j = 0; j < tempUsedItems[i].Count; j++)
5     {
6         usedItems[i, j] = tempUsedItems[i][j];
7     }
8 }

```

Erklärung: Konvertiert die temporäre Liste von Gruppen in ein 2D-Array, um die ausgewählten Items also die perfekt Lösung zu speichern.

4.5.6.7 Rückgabewert

```

1 return dp[n, capacity];

```

Erklärung: Gibt den Gesamtwert des besten Rucksacks zurück.

4.5.6.8 Darstellung der perfekten Lösung

Um die perfekt Lösung nach der Berechnung erfolgreich darzustellen wird in diesem Abschnitt auf die *PLACEHOLDES* Funktion des *KnapsackScript.cs* eingegangen um die errechnete Lösung nach dem drücken des *SolveButton* anzuzeigen

4.5.6.9 Berechnen des eigenen Inventars

In diesem Abschnitt wird auf die Funktion *KnapsackInventoryValue* des *KnapsackScript.cs* eingegangen. Diese Funktion berechnet das Inventar Value des selbst gebauten Inventars.

4.5.6.10 Funktionsaufruf

```

1 inventoryValue = KnapsackInventoryValue(inventory);
2 if (maxValue == inventoryValue)
3 {
4     infoMesh.color = Color.green;
5     infoMesh.text = "Maximale Punktzahl erreicht";
6 }
7 else
8 {
9     infoMesh.text = "";
10 }
11 ownMesh.text = "Erreichter Wert: " + inventoryValue.ToString();

```

Erklärung:

1. **Inventar Wert berechnen:** Die Funktion *KnapsackInventoryValue* wird mit dem übergebenem inventory aufgerufen und der Inventar Wert berechnet.

2. **Ergebnis Handling:** Wenn der eigene Wert gleich dem Maximalen errechnetem Wert entspricht hat der Benutzer gewonnen. Anderenfalls wird dem infoMesh der aktuelle Wert zugewiesen.

4.5.6.11 Code für die Berechnung des eigenen Inventars

```
1 int KnapsackInventoryValue(int[,] inventory)
2 {
3     if(inventory == null)
4     {
5         throw new System.Exception("Inventory is null");
6     }
7
8     int totalValue = 0;
9
10    foreach (var item in items.Values)
11    {
12        int itemId = item.id;
13        int itemValue = item.value;
14
15        for (int j = 0; j < inventory.GetLength(0); j++)
16        {
17            for (int k = 0; k < inventory.GetLength(1); k++)
18            {
19                if (inventory[j, k] == itemId)
20                {
21                    totalValue += itemValue;
22                }
23            }
24        }
25    }
26
27    return totalValue;
28 }
```

Erklärung:

1. **Überprüfung auf Null:** Die Funktion überprüft, ob das übergebene Inventar null ist. Falls ja, wird eine Ausnahme ausgelöst.
2. **Initialisierung des Gesamtwerts:** Die Variable `totalValue` wird auf 0 initialisiert. Hier wird der Gesamtwert des Rucksackinventars gespeichert.
3. **Durchlauf aller Gegenstände:** Die Funktion durchläuft alle Gegenstände im globalen `items`-Dictionary.
4. **Extrahierung von Informationen:** Für jeden Gegenstand werden die ID (`itemId`) und der Wert (`itemValue`) extrahiert.
5. **Durchlauf des Inventars:** Die doppelte Schleife durchläuft das gesamte Inventar (`inventory`) und prüft, ob der aktuelle Gegenstand vorhanden ist.
6. **Berechnung des Gesamtwerts:** Wenn der Gegenstand im Inventar gefunden wird, wird sein Wert zum Gesamtwert (`totalValue`) addiert.
7. **Rückgabe des Gesamtwerts:** Am Ende wird der Gesamtwert des Rucksackinventars zurückgegeben.

4.5.7 Unit-Tests

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

4.6 Performance

Performance-Messung

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 Knappsack-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://sneakydaggersgames.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.foundation@4.1/manual/plane-manager.html> (besucht am 05.11.2023).
- Unity. AR Raycast Manager. URL: <https://docs.unity.cn/Packages/com.unity.xr.foundation@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. Loop Tools. URL: <https://docs.blender.org/manual/en/latest/addons/mesh/looptools.html> (besucht am 20.11.2023).
- 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-qr-code-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.