HAW HAMBURG

Masterarbeit

Jonathan Becker

Entwicklung eines Interaktionsframeworks zur Erstellung interaktiver Virtual Reality Szenarien

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Jonathan Becker

Entwicklung eines Interaktionsframeworks zur Erstellung interaktiver Virtual Reality Szenarien

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Dr. Susanne Draheim

Eingereicht am: 07. Juni 1954

Jonathan Becker

Thema der Arbeit

Entwicklung eines Interaktionsframeworks zur Erstellung interaktiver Virtual Reality Szenarien

Stichworte

Virtual Reality, Training, Interaktion

Kurzzusammenfassung

Die Entwicklung von interaktiven Virtual Reality Szenarien benötigt viele Interaktionen, die miteinander verknüpft werden, um komplexe Szenarien zu bilden. Dafür wird in dieser Arbeit ein Framework entworfen und implementiert, das die Erstellung beschleunigen und verbessern soll. Die Evaluation findet am realen Einsatz, Anforderungen und Beispielszenarien statt.

Jonathan Becker

Title of Thesis

Development of a interaction framework to create interactive Virtual Reality Szenarios

Keywords

Virtual Reality, Training, Interaction

Abstract

The development of interactive virtual reality scenarios requires many interactions that are linked together to form complex scenarios. For this purpose, a framework is designed and implemented in this thesis to accelerate and improve the creation. The evaluation is based on real-world applications, requirements and example scenarios.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Training für gefährliche Situationen, wie es Wartungstechniker brauchen, benötigt die Nachbildung oder das Stilllegen der Einsatzorte. Beides ist nicht immer machbar und kostenintensiv, zusätzlich ist es nicht immer möglich die Gefahren realistisch nachzubilden und auszuschließen. Als Beispiel dient das EnBW-Projekt, in dem drei Trainingsszenarien für Wartungstechniker von Windenergieanlagen entstanden sind: Eine Erstbegehung der Anlage, die Rettung eines Verletzten in 80 Meter Höhe und der Tausch eines Invertermoduls im Hochvoltbereich.

In dieser Arbeit wird untersucht, wie die Erstellung von Interaktiven Applikationen unterstützt werden kann. Der Schwerpunkt liegt dabei im Bereich von Schulung und Lehre. Das dafür erstellte Framework soll sich auf Virtual Reality spezialisieren, die seit der Definition des „Ultimate Displays“ von Sutherland [30] in den Möglichkeiten immer vielfältiger geworden ist. Dieses macht ein solches Framework attraktiv, da die Anzahl der möglichen Anwendungsszenarien steigt.

Dieses befähigt Virtual Reality dort für Serious Games einzusetzen, wo Probehandeln in der realen Welt schwierig ist oder schwerwiegende Konsequenzen haben kann. Dabei hilft virtuelles Training Sicherheits- und Gesundheitsrisiken vorzubeugen und Konsequenzen können gelebt und gefährliche Situationen geprobt werden [37]. Damit dieses aber genutzt werden kann, muss es möglich sein, für eine Vielzahl von Themen Szenarien erstellen zu können.

Heutige Game Engines nehmen dabei einen großen Teil der Entwicklung ab und unterstützen das Erstellen von Szenarien. Während sie so bereits Physik und Grafik Engine mitbringen, existieren in ihnen noch keine fertigen Interaktionen, die direkt für solche Szenarien verwendet werden können. Um dies zu beschleunigen und dadurch mehr Zeit für komplexe Abläufe zu haben, gilt es ein entsprechendes Framework zu entwickeln, das dieses unterstützt.

In vorherigen Arbeiten wurde dabei bereits der Grundstein für ein solches Framework gelegt [3] [2]. Dort wurde gezeigt, dass Interaktionen über Vererbung implementiert werden können und sich diese effizient nutzen und weiter verwerten lassen. Eine Erweiterung des Kontextes dieser Arbeit, die sich auf die Funktionen innerhalb der Game Engine fokussiert, bietet die Arbeit von Niklas Gerwens [9], die ein Hardware Abstraction Layer beschreibt, das sowohl Input als auch Feedback außerhalb der Virtual Reality unterstützt.

1.1 Zielsetzung

In dieser Arbeit soll ein Framework zur Unterstützung bei der Erstellung interaktiver Virtual Reality Szenarien konzipiert und entwickelt werden. In diesem Umfeld ist es wichtig, Nutzer aus unterschiedlichen Disziplinen zu unterstützen und auch für die Nutzer der Szenarien einheitliche Interaktionen anzubieten. Durch den Einsatz in diesem nicht homogenen Anwenderfeld ergeben sich Anforderungen, die analysiert und umgesetzt werden sollen.

Während für Unity schon eine Vielzahl an Erweiterungen existiert, sind diese nicht auf die schnelle Erstellung von Trainingsszenarien spezialisiert. So ist besonders zur schnellen Erstellung neuartiger Interaktionen ein erweiterbares Framework wichtig. Zusätzlich zur Abstraktion von Input müssen Interaktionen beliebiger Komplexität mit minimalem Aufwand erstellbar sein. Damit diese zu Szenarien werden, ist eine weitere darüber liegende Schicht notwendig, die diesen Prozess unterstützt.

Dieses soll durch ein Framework geschehen, das die benötigten Features enthält, um die Entwicklung von interaktiven Virtual Reality Szenarien zu beschleunigen. Das Ziel sind dabei hoch interaktive Environments, in denen die Interaktionen zu größeren Verbänden und Abläufen zusammen gekettet werden können. Damit soll es möglich sein, reale Umgebungen für den Einsatz in Trainings abzubilden. Zur Unterstützung von Trainings muss es einfach sein, Feedback und Kontrolle mit den Interaktionen in Einklang zu bringen. Auch die Anbindung externer Komponenten soll unterstützt werden, wobei das Framework für Unity und C# erstellt wird, wodurch die entsprechenden Schnittstellen vorgegeben sind.

1.2 Aufbau der Arbeit

Die Arbeit gliedert sich in insgesamt fünf Kapitel. Das erste Kapitel, die Einleitung, führt in die Arbeit ein, setzt Ziele und erläutert den Aufbau.

Im zweitem Kapitel wird die Zielumgebung der Arbeit analysiert. Dazu werden die Bereiche Virtual Reality, Interaktionen, Training in Virtual Reality und Parameter für Frameworks analysiert. Gefolgt wird dies durch die Aufschlüsselung von zwei Szenarien, die dem Anwendungszweck des Frameworks entsprechen. Danach werden die Anforderungen aufgelistet und das Kapitel endet in einer Zusammenfassung.

Das dritte Kapitel beschäftigt sich mit dem Design und der Erstellung des Frameworks. Als erstes wird dafür der allgemeine Aufbau beschrieben und wie dieser mit den Anforderungen zusammenhängt. Danach werden die Module des Frameworks beschrieben und ihre Features und Umsetzung vorgestellt.

Im viertem Kapitel wird das erstellte Framework anhand von drei Kategorien evaluiert. Als erstes anhand der Unterstützung bei der Implementierung der Beispielszenarien. Zweitens über die aufgestellten Anforderungen. Drittens durch Artefakte und die Gesamtwertung in der Verwendung im EnBW-Projekt.

Das fünfte Kapitel schließt die Arbeit ab, indem es sie zusammenfasst und einen Ausblick in die Zukunft bietet.

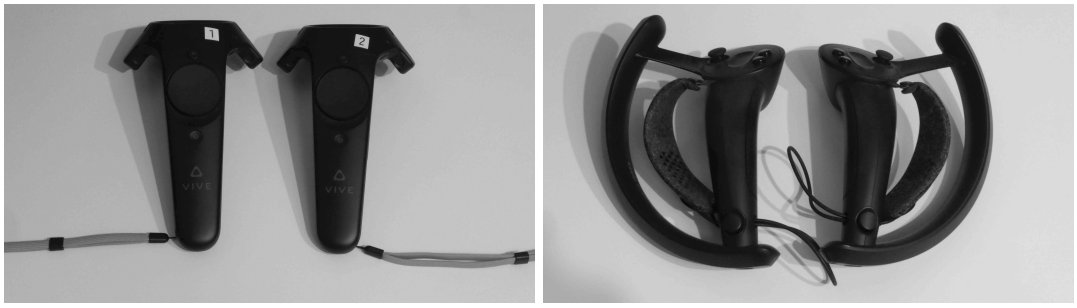
2 Analyse

Dieses Kapitel beschäftigt sich mit der Analyse des Einsatzgebietes des Frameworks. Das erste Thema ist Virtual Reality, siehe 2.1, der aktuelle Stand und welche Möglichkeiten Virtual Reality Nutzern und Entwicklern bietet. Das nächste Thema sind Interaktionen, siehe 2.2, erst sie erlauben es Virtuelle Welten interaktiv werden zu lassen und befähigen den Nutzer, die Welt zu beeinflussen. Mit diesen Grundlagen ist es möglich, die Virtuellen Welten für Training und Schulungen, siehe 2.3, zu nutzen. Zur Erstellung von erfolgreichen Trainingsszenarien sind Werkzeuge notwendig und diese müssen entsprechende Möglichkeiten bieten. Dazu werden vier Säulen im Kapitel: Parameter für Frameworks, siehe 2.4, vorgestellt. Darauf werden die Anforderungen an das Framework spezifiziert, siehe 2.6. Der letzte Abschnitt in diesem Kapitel, siehe 2.7, fasst die Aspekte zusammen.

2.1 Virtual Reality

Virtual Reality ist ein altes Konzept. Die ersten Entwicklungen gab es in den 1960er-Jahren mit der Definition des „Ultimate Displays“ von Sutherland [30]. Seitdem haben sich die Systeme rapide weiterentwickelt und das volle Ausschöpfen der Möglichkeiten ist nicht mehr notwendig. Das Thema „good enought“ beschreibt dafür den Punkt, den eine Anwendung erreichen muss um ihren Zweck ausreichend zu erfüllen. Für Navigation z. B. ist natürliche Bewegung wichtiger als hochwertige Grafik [26].

Mit der Weiterentwicklung der Systeme, siehe Kapitel 2.1.1, ist die Verwendung sowohl für Endnutzer als auch Entwickler besser geworden. Dieses begünstigt auch die Verwendung in neuen Bereichen, wie auch im Thema Schulung. Einen guten Überblick über den gesamten Themenbereich bietet „Virtual und Augmented Reality (VR/AR)“ [8].



(a) Die Wands Controller (b) Die Knuckel Controller

Abbildung 2.1: Unterschiedliche Virtual Reality Controller

2.1.1 Virtual Reality Systeme

Die Virtual Reality Systeme bieten inzwischen preiswerte und fortgeschrittene Möglichkeiten, in virtuelle Welten einzutauchen. Die benötige Hardware ist dabei leicht und vielfältig und lässt sich ohne Fachwissen nutzen. Die Präzision vieler Systeme erlaubt es, Bewegungen realistisch und in Echtzeit in die Welten zu übertragen.

Für diese Systeme stehen unterschiedliche Controller zur Auswahl. Die Entscheidung, welcher Typ eingesetzt wird, hat dabei Auswirkungen auf die Geschwindigkeit und Präzision, mit der Aufgaben ausgeführt werden. Es ist aber von den Aufgaben abhängig, welche Controller besser geeignet sind [20]. Wenn es darum geht Input zu vereinheitlichen, können Werkzeuge wie ein Hardware Abstraktion Layer [9] genutzt werden, um eine einheitliche Schnittstelle in die Virtual Reality zu haben.

Der generierte Input lässt sich dabei in mehrere Gruppen teilen. Die am meisten verbreitete ist 6dof (degre of freedom), diese überträgt Position und Rotation in die virtuelle Welt und kann so getrackte Objekte mit deren Bewegungen direkt wiedergeben. Zwei Controller, die dieses erlauben, sind die Wands, Abbildung 2.1a, und die Knukels, Abbildung 2.1b. Der größte Unterschied zwischen diesen beiden Controllern ist das die Knukels es ermöglichen die Handposition zu messen, um diese zusätzlich in die virtuelle Welt zu bringen. Zusätzlich können diese durch die Haltebänder losgelassen werden, ohne dass diese aus der Hand fallen.

2.1.2 Endnutzer

Der Endnutzer kann mit dem aktuellen Stand inzwischen nahezu fotorealistische virtuelle Welten in Echtzeit erleben. Dieses erlaubt einen hohen Grad der place illusion [17] ein Teil der Immersion. Die place illusion beschreibt wie sehr es möglich ist, sich vor Ort zu fühlen und die Virtuelle Welt als „Echt“ zu akzeptieren, obwohl bekannt ist, dass diese nicht real ist. Dieses ermöglicht konsistente Welten mit plausibility [17] zu schaffen, wenn der Nutzer mit diesen interagiert und die Welt reagiert. Ein Beispiel ist das korrekte Lernen des Verbeugens in sozialen Situationen [6], dafür ist dieses in Dialoge eingebaut, wo es für das Fortschreiten in der Anwendung notwendig ist.

2.1.3 Entwickler

Inzwischen unterstützen viele der 3D Engines, z.B. Unity und Unreal, Virtual Reality und haben damit die Notwendigkeit abgelöst, eigene Engines zu schreiben. Diese Generalisierung erlaubt es, Frameworks und Werkzeuge zu entwickeln, die längere Lebenszeiten haben und dabei helfen, Entwicklungsaufwand zu reduzieren und zu beschleunigen. Ein Beispiel hierfür ist das Tool FlowMatic [38], das es erlaubt, innerhalb der Virtual Reality das Verhalten von Inhalten durch interaktive Elemente, siehe Abbildung 2.2, zu spezifizieren. Ähnlich gibt es eine große Auswahl aus kommerziell erhältlichen Werkzeugen, die spezielle Aufgaben erleichtern können.

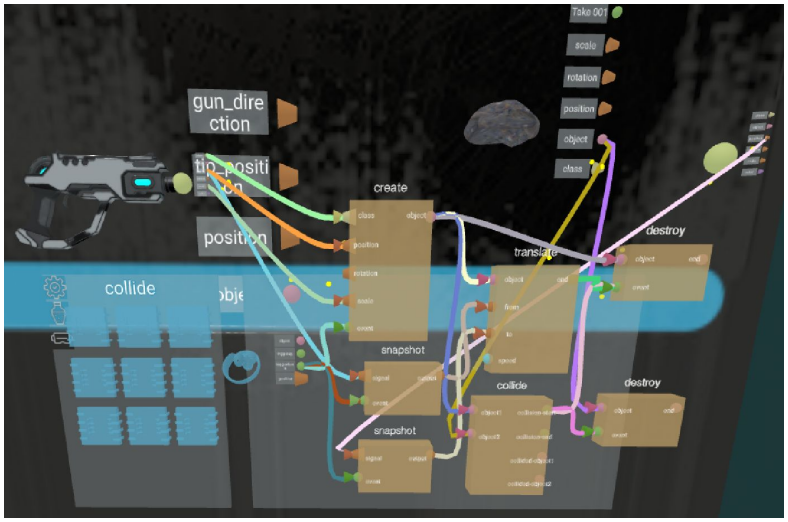


Abbildung 2.2: Ein einfaches Beispiel in FlowMatic [38, Figure 5].

2.2 Interaktionen

Interaktionen in der Virtual Reality fallen in den Bereich der Mensch-Computer Interaktion (MCI, engl. Human-Computer Interaction, HCI). Dieser Forschungsbereich beschäftigt sich mit dem Design, der Evaluierung und Umsetzung der Kommunikation zwischen Computer und Mensch. Im Mittelpunkt des Themas steht dabei die Usability der Schnittstellen.

Für klassische 2D Oberflächen gibt es Leitfäden für die benutzerfreundliche Gestaltung aufgrund von Erkenntnissen aus Design, Psychologie und Arbeitswissenschaft [13]. Modelle wie WIMP (Windows, Icons, Menus, Pointing) haben sich dabei in diesem Bereich etabliert. Für Virtual Reality lassen sie sich auf unterschiedliche Arten von 2D zu 3D umsetzten. 3D Aufgaben müssten dann in Einzelschritte zerlegt werden, die mit 2D Methoden erledigt werden können, was zu erhöhtem Arbeitsaufwand und niedrigerer Effizienz führt [18]. WIMP kann aufgrund der Bekanntheit dennoch eine gute Wahl für einzelne Elemente sein, wenn die Aufgaben nah genug an dem Schema liegen.

Dieses ist darin begründet, dass Virtual Reality andere Schnittstellen für Interaktionen bietet. Das wichtigste Beispiel hierfür ist Bewegung, die durch das Tracking direkt übertragen wird, was natürlichere Interaktionen der Positionsänderung erlaubt [10], anders als virtuelle Desktop Environments, wo dieses nur über Analogien funktioniert. Andere Teile des Körpers, die in die virtuelle Welt übertragen werden, haben den gleichen Vorteil, dass sie für natürliche Interaktionen genutzt werden können. Der Reality Gap wird so verringert und es entsteht eine bessere Übertragbarkeit und Orthogonalität. Dieses unterliegt neuen Richtlinien, für die es gilt, neue Konzepte zu finden. Für die Interaktionsfidelity ist es dabei nicht immer zielführend, Körperaktionen vollständig umzusetzen [24]. Die fehlenden Standards sorgen für inhomogene Interaktionen zwischen Anwendungen, es fehlt Orthogonalität, um übertragbar zu werden. So ist es besonders wichtig, dass Interaktionen innerhalb einer Anwendung einfach zu erlernen und zu nutzen sind. Eine Möglichkeit dieses zu evaluieren ist die Methode der Testbed Evaluations [4]. Dazu wurden Interaktionen in die drei Phasen: Selection, Manipulation und Release geteilt und in einer Taxonomie, siehe Abbildung 2.3, den Phasen unterschiedliche Elemente zugeteilt.

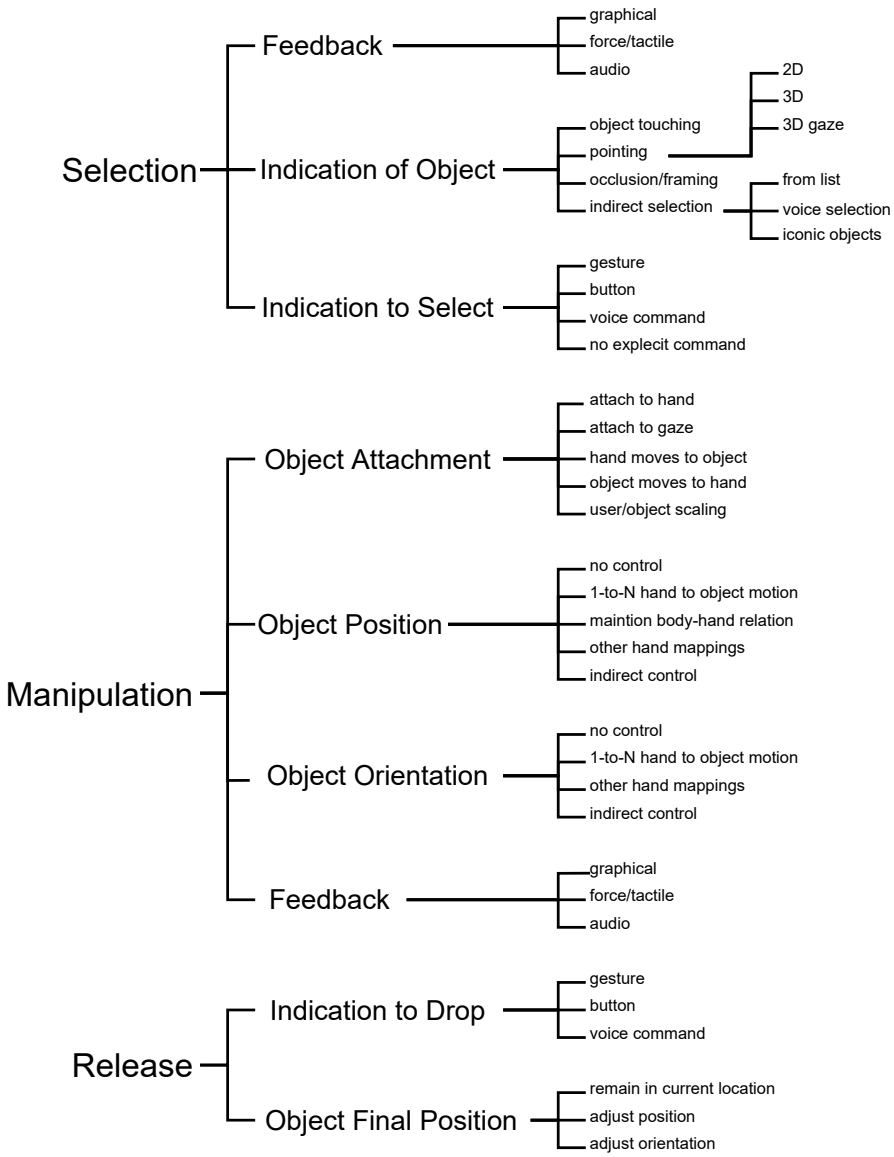


Abbildung 2.3: Taxonomie für Interaktionen, ähnlich [4].

2.2.1 Selektion und Manipulation

Interaktionstechniken lassen sich anhand vieler unterschiedlicher Merkmale gruppieren. Ein Merkmal sind die genutzten Grade der Freiheiten (dof). Dieses können Werte wie Position und Rotation sein, die in 3D jeweils drei Freiheitsgrade haben. Jeder weitere genutzte Input, wie Knöpfe am Controller, bildet dabei einen weiteren Freiheitsgrad, der genutzt werden kann. Für eine geringe Komplexität der Techniken ist dabei eine geringe Anzahl der Freiheiten wichtig [1, 4.4.1]. Dieses liegt auch daran, dass mit zunehmenden Freiheitsgraden die Komplexität der Verknüpfungen in die Anwendung für unterschiedliche Controller steigt.

In der Quelle [36] wird eine große Anzahl von Techniken verglichen und bietet einen guten Überblick über deren Eigenschaften. Viele dieser Techniken sind Hyper-natural Interaktionen, die es ermöglichen, die Reichweite des Nutzers zu vergrößern [5]. Diese können aber die Präzision reduzieren und sind nicht bei jeder Aufgabe hilfreich. Für Interaktionen mit der Umgebung sind virtual Hand Techniken besser als virtual pointing Techniken geeignet, da diese die natürlichere und direktere Möglichkeit bieten, die Umgebung zu beeinflussen [18].

Die Hände mit einer direkten Übertragung der Freiheitsgrade zu nutzen liegt entsprechend nahe, auch wenn der Aufwand von Rotationsaufgaben höher als in der realen Welt ist [28]. Andere Techniken umgehen das Problem wie die Handelbar Technik, die beide Hände nutzt, um das Objekt auf einer Stange zu halten und so präzisere Drehungen ermöglicht und zusätzlich die Skalierung anbietet. Hier sind aber beide Hände belegt und zur Manipulation deutlich größere Bewegungen notwendig, was zur schnelleren Ermüdung führt [1].

2.2.2 Feedback

Interaktionen können auf diverse Art Feedback geben. Dieses kann bei Selection, Release, Manipulation und Zustandsänderungen ausgelöst werden, um die aktuelle Interaktion zu verdeutlichen oder ihr Konsequenzen zu geben. Je nach Kontext kann Feedback in eine von drei Kategorien geteilt werden.

Die Erste davon ist das intrinsische Feedback, mit dem Interaktionsobjekte ihren Zustand verkörpern, es ist immer aktiv und an das Objekt gebunden. Dieses ist z.B bei einem Fenster der Öffnungsgrad. Wenn der Nutzer mit dem Fenster interagiert und dieses öffnet,

ändert sich zwangsweise der Öffnungsgrad. Intrinsisches Feedback wirkt damit immer direkt auf das Environment und die Konsequenzen können nicht abgeschaltet werden. Der Ball kann nur hindurch geworfen werden, wenn das Fenster offen ist.

Direktes Feedback ist die Änderung des Environments in Folge einer Zustandsänderung einer Interaktion. Direktes Feedback steht immer im Zusammenhang mit der Zustandsänderung einer Interaktion an anderer Stelle. Wird zum Beispiel ein Ventil geöffnet kann dies an anderer Stelle das Fließen von Wasser auslösen. Da hier ein Zwischenschritt notwendig ist und dieser nicht garantiert ist, handelt es sich nicht um Intrinsisches Feedback.

Entferntes Feedback beschäftigt sich mit Effekten, die nicht in der virtuellen, sondern in der realen Welt auftreten. Ein Ofen ist hierfür ein Beispiel, wenn das Anschalten von diesem über einen Heizkörper die reale Raumtemperatur erhöht.

2.3 Training

Aktuelle Virtual Reality Systeme erlauben einen vielseitigen Einsatz. Für Unterhaltungsmedien von Spiel bis Film. Auch als Werkzeug zur Arbeit, z. B. zur Unterstützung bei der Analyse von multidimensionalen Daten können sie helfen [19]. Ebenso für Lernmedien von Dokumentation, genereller Schullaufbahn [23] zu technischem Training [27] und Rehabilitation [25] können Virtual Reality Systeme eingesetzt werden.

Virtuelles Training bietet dabei viele Vorteile. Einfacher Skalierung von Trainingsumgebungen, die sonst nur kurz besucht werden können oder schwer zu erreichen sind. Auch sind so die Anforderungen an den vorhandenen Raum stark reduziert, da alle benötigten Umgebungen digital nachgebaut werden können [31]. So können diese Orte länger und vor dem eigentlichen Training besucht werden, dieses hilft, eine bessere räumliche Vorstellung aufzubauen und Suchaufgaben effizienter zu erledigen [16]. Zusätzlich entfällt das aufwendige Herrichten, Anpassen und Zurückbauen des Trainingsszenarios [25] und der Verbrauch von Material [27] [35]. Ein weiterer wichtiger Punkt ist das gefahrenfreie Trainieren von Aufgaben, die bei Fehlern in der Realität schwerwiegende Konsequenzen hätten. Der Umgang mit Hoch-Volt Schaltanlagen[22] ist hier ein Beispiel wie auch das Aufstellen von Konstruktionshilfsmitteln im Bau [12]. Das Gefühl, wirklich vor Ort zu sein, erlaubt es, das Training effizient zu gestalten. Sowohl physikalische als auch psychologische Aspekte im Training spielen dabei eine wichtige Rolle für die Immersion [21].

Mit den heutigen Möglichkeiten sind 6dof Interaktionen mit den Händen aufgrund der geübten Feinmotorik einfach und natürlich. Diese natürlichen Interaktionen sind gut übertragbar zwischen den Welten. Wenn nötig, lassen sie sich durch Hyper-natural Features anreichern, wie Objekte, die an Positionen einrasten und an Ort und Stelle in der Luft stehen bleiben [28].

2.4 Parameter für Frameworks

Abhängig von den Eigenschaften, nach denen ein Framework bewertet wird, werden die Anforderungen festgelegt. Dabei steht der Nutzen sowohl für die Entwickler von Szenarien als auch für deren Anwender im Vordergrund. Dazu gilt es Interaktionen einfach in die Virtual Reality zu bringen und diese für Szenarien zu verknüpfen.

Dazu können vier Kategorien von Eigenschaften [14] identifiziert werden: Einfache Nutzung der Funktionen; Unterstützung beliebiger Komplexität; Entwicklung wird beschleunigt; Vollständig konfigurierbar.

2.4.1 Einfache Nutzung

Die einfache Nutzung ist sowohl für den Endnutzer als auch Entwickler, von Einsteiger bis Experte, wichtig. In den vorherigen Arbeiten [3] [2] wurde die Wichtigkeit der Orthogonalität für Interaktionen, sowohl beim Erstellen als auch beim Nutzen, begründet. Sie erhöht die Übertragbarkeit von Wissen zwischen verwandten Komponenten und erleichtert die Einarbeitung.

Für diese Arbeit soll es drei Definitions-Stufen der Entwicklerbefähigung geben: Einsteiger, Fortgeschrittene und Experten. Die Anforderung, dass der Übergang zwischen unterschiedlichen Wissen-Stufen möglichst einfach sein muss, wurde in [14] aufgestellt. Dieses setzt voraus, dass die Komponenten der unterschiedlichen Stufen miteinander kompatibel sind, ohne einander vorauszusetzen, damit keine harten Abgrenzungen entstehen. Für die Entwicklung in Teams ist dieser Punk essentiell, da so Personen unterschiedlicher Spezialisierungen auf unterschiedlichen Stufen zusammenarbeiten können. Es ist dabei davon auszugehen, das der größte Teil der Nutzer sich zwischen den Stufen Einsteiger und Fortgeschrittene aufhält.

Einsteiger

Für Einsteiger sind grafische Oberflächen eine geeignete Schnittstelle, sie erlauben es, sich mit den Möglichkeiten des Frameworks vertraut zu machen, ohne den Code verstehen zu müssen. Die UI kann auch direkte Hilfestellungen zu Problemen und Konfiguration bieten sowie die Nutzungsmöglichkeiten zeigen, um die Nutzung zu erleichtern. Flow Diagramme sind Teil davon [38] und erlauben es, komplexes Verhalten ohne Code zu generieren, indem Komponenten visuell verknüpft werden können. Hierfür bieten sich ebenfalls Event basierte Systeme zur losen Kopplung an, da diese sich auch gut orthogonal durch die Stufen integrieren lassen.

Fortgeschrittene

Für Fortgeschrittene steht die Nutzung von Schnittstellen durch Code im Vordergrund. Zu dieser Umsetzung sind Vorlagen notwendig, die höhere Aufgaben übernehmen, um die Komplexität und Fehleranfälligkeit zu reduzieren. Dieses ermöglicht eigene Konzepte und komplexeres Verhalten zu implementieren. Fortgeschrittenen Nutzern sollen so mit Hilfe der Vorlagen die Möglichkeiten der Einsteiger zu erweitern können. Dazu sollen vorhandene Funktionen auf Code Basis neu kombinierbar sein, um weitere visuelle Komponenten mit ergänzendem Verhalten zu erschaffen.

Experten

Experten haben eine hohe Vertrautheit mit dem Framework und sind erfahrene Programmierer. Sie können alle Funktionen nutzen und überall Änderungen und Erweiterungen vornehmen. Mit dieser vollständigen Perspektive können sie für Einsteiger und Fortgeschrittene weitere Möglichkeiten durch neue Werkzeuge und Hilfsmittel schaffen.

2.4.2 Unterstützung beliebiger Komplexität

Zur Unterstützung beliebiger Komplexität darf das Framework nicht kompliziert aufzusetzen zu sein. Während Konfiguration bei großen Projekten notwendig ist, erschwert diese den Einstieg und macht den Einsatz in kleinen Projekten ineffizient. Dieses setzt auch Modularität voraus, da nicht jedes Projekt alle Aspekte eines Frameworks benötigt. Die nicht benötigten Teile sollten in diesem Fall ohne Konsequenzen zu ignoriert sein.

Wenn die Komplexität eines Projektes mit der Zeit wächst, kann dieses dazu führen, dass vorhandene Komponenten einschränkend werden. Daher ist es notwendig, dass diese erweitert, ersetzt oder umgangen werden können, um solche Projekte zu realisieren. Vorlagen sollen dabei helfen, dass neu erstellte Komponenten kompatibel mit den existierenden bleiben. So kann Komplexität, die über die vorhandenen Möglichkeiten des Frameworks geht, kompatibel gehalten werden.

Um die Bildung von Super-Ojekten zu verhindern, die zu viel Komplexität an einer Stelle vereinen, ist es nützlich diese so weit wie möglich auf einfache Objekte zu verteilen. Zum Beispiel in dem die Objekte von sich aus interaktiv und in ihrer Funktion geschlossen sind, so dass auf diese nur noch abstrakt zugegriffen wird. Dieses erleichtert besonders bei komplexen Environments die Konfiguration und verbessert die Übersicht [29].

2.4.3 Entwicklung wird beschleunigt

Durch die Verbreitung von heutigen Game Engine und anderen Werkzeugen wird die Entwicklung bereits stark beschleunigt. Um dieses auch auf ein Framework zu übertragen, ist eine Plug and Play Architektur mit einfacher Konfiguration hilfreich. Es sollte bei der Integration nicht notwendig sein, den Code des Frameworks anzupassen, wobei das Erweitern oder Implementieren von Schnittstellen akzeptabel ist. Diese Module müssen entsprechend über sinnvolle Konfigurationsmöglichkeiten verfügen, die in der Lage sind, die notwendigen Anpassungen vorzunehmen.

Konstruktive Fehlermeldungen sind ein weiterer Bestandteil der Beschleunigung in der Entwicklung. Fehler können dabei an unterschiedlichen Stellen entstehen und müssen unterschiedlich behandelt werden. Im besten Fall wird die Entstehung des Fehlers verhindert oder eine Unterstützung zum Beheben des Fehlers angeboten [35]. Diese sollten zusätzlich global einsehbar sein und für das gesamte Projekt prüfbar sein, da es möglich ist, dass bereits konfigurierte Objekte aufgrund von Änderungen an anderer Stelle ungültig werden. Dieses kann z.B der Fall sein, wenn Objekte gelöscht oder verschoben werden und die Referenzen zu diesen fehlen.

Ein weiter wichtiger Punkt ist der Umfang des Frameworks. Wenn dieses nicht genug Komplexität mitbringt oder diese sich nicht einfach erweitern lässt, wird die Entwicklung stark ausgebremst [11]. Die Komplexität muss aber übersichtlich gegliedert sein, dass Nutzer nicht überfordert werden.

2.4.4 Vollständig konfigurierbar

Die vollständige Konfigurierbarkeit ist notwendig, um Unabhängigkeit von der Hardware zu erlangen. Der größte Teil wird inzwischen von der Game Engine und den entsprechenden Virtual Reality-Frameworks übernommen. Für die Einbindung unterschiedlicher Controller ist eine Schnittstelle notwendig. Dazu ist es hilfreich, den Input auf einen einheitlichen Actor zu übertragen [15]. Abstrakter Input, wie z. B. Sprachsteuerung sollte auch einzubinden sein. Wenn der Input bereits abstrahiert ist, sollte dies im besten Fall mit geringer Komplexität und ohne Code möglich sein. Zu diesem Zweck ist Modularität hilfreich, da sie die Konfigurationsmöglichkeiten aufteilt und strukturiert.

2.5 Vorstellung der Szenarien

Um benötigte Features eines Frameworks zu identifizieren ist eine Analyse des Einsatzes notwendig. Die hierfür zugrundeliegenden Szenarien sind Ausschnitte aus realen Trainingsszenarien für Wartungstechniker von Windenergieanlagen. Sie wurden ausgewählt, da sie eine große Spanne an Interaktionen und Feedback abdecken und repräsentativ für vollständige Trainingseinheiten sind. Die Nutzung einer Hardware-Abstraktionsschicht [9] kann dabei eine höhere Vielfalt an Funktionen für die Szenarien ermöglichen. Dafür werden mithilfe der Szenarien auch benötigte Schnittstellen identifiziert und beschrieben.

2.5.1 Vorstellung Szenario 1

Das erste Szenario beschreibt das Öffnen der Dachluken einer Windenergieanlage. Dieses ist ein essenzieller Schritt, um genug Platz für die Arbeit zu schaffen und eine Notwendigkeit um Material mit dem Kran nach oben zu befördern. Es beinhaltet mehrere in Abhängigkeit stehende Interaktionen und einen Abschnitt, in dem mit beiden Händen unterschiedliche Interaktionen gleichzeitig getätigt werden müssen.

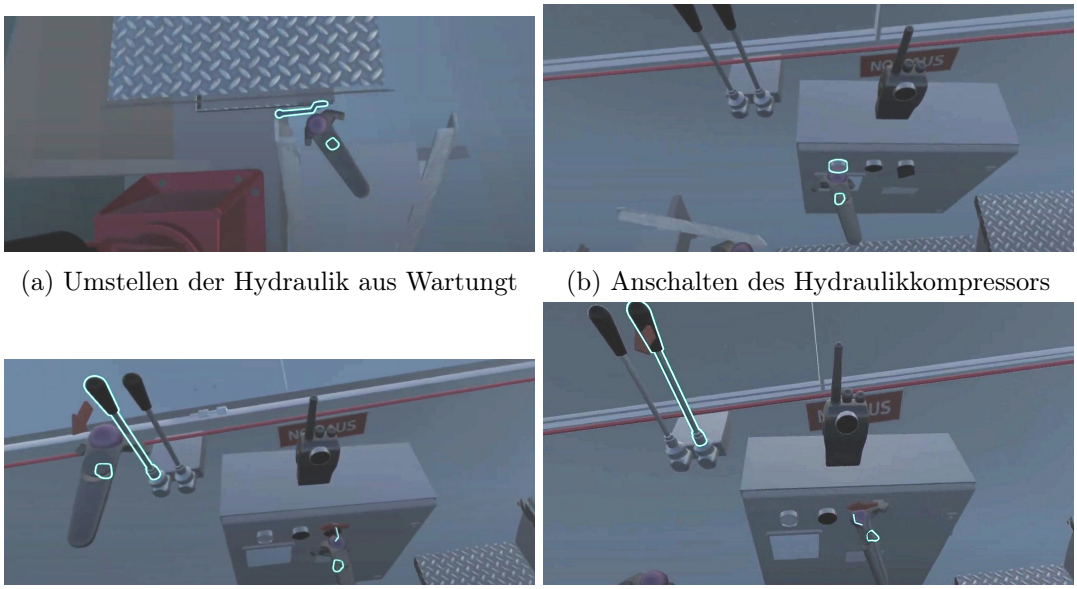
Die notwendigen Interaktionen und ihre Konsequenzen für diese Prozedur werden in der folgenden Umsetzung des Szenarios mit unterschiedlichen Arten von Feedback beschrieben. Die endgültige Version des Szenarios ist in folgendem Video zu sehen [7].

Das Szenario findet im hinteren Teil der Anlage statt. Es startet am Hydraulikkompressor mit einem Hebel, der um 90 Grad umgestellt werden muss 2.4a, dieser erzeugt dabei Audio-Feedback, wenn er eine seiner Endpositionen erreicht. Der zweite Schritt ist das Einschalten des Hydraulikkompressors mithilfe eines Knopfes 2.4b, dieser gibt Audio-Feedback für die Interaktion. Solange der Hydraulikkompressors aktiv ist läuft seine Tonspur. Das folgende Öffnen der Dachluken muss zweihändig vorgenommen werden, mit der rechten Hand muss ein Drehschalter nach rechts gedreht und gehalten werden. Er kehrt automatisch in seine Ausgangsposition zurück und hat Audio-Feedback beim Erreichen seiner Endpositionen. Mit der linken Hand müssen währenddessen zwei Hebel nacheinander bedient werden. Als Erstes wird durch ziehen des rechten Hebels die rechte Dachklappe geöffnet 2.4c, da diese die linke Dachklappe überlappt. Solange sich eine der Dachklappen in Bewegung befindet läuft eine Rückmeldung gebende Tonspur. Weiteres Feedback, das durch das Öffnen der Dachklappen ausgelöst wird, ist die Umstellung der Umgebungsgeräusche von Drinnen auf Draußen. Danach kann mit dem linken Hebel die Linke Dachklappe geöffnet werden, 2.4d was das Szenario abschließt. Weiterhin kann das Öffnen der Dachklappen durch Externes Feedback wie einen Wärmestrahler, der die Sonne simuliert, oder einen Ventilator, der Wind erzeugt, unterstützt werden.

Während alle Interaktionen des Szenarios jederzeit auf Input reagieren, müssen bestimmte Bedingungen erfüllt sein, damit die Dachluken sich als Reaktion öffnen können. In diesem Fall hat die zweihändige Interaktion nur Konsequenzen, wenn vorher sowohl der Hebel am Hydraulikkompressor umgestellt, als dieser auch eingeschaltet wurde. Dieses gilt es entsprechend in einer Umsetzung darzustellen.

Die Umsetzung der beschriebenen Interaktionen enthält unterschiedliche Elemente. Der Hebel am Hydraulikkompressor und die Hebel zur Dachklappen-Steuerung sind Hebel, die sich zwischen einstellbaren Limits bewegen lassen. Ein Knopf mit zwei Positionen, der sich umschalten lässt, sowie ein Drehknopf, der sich zwischen zwei Positionen drehen lässt. Die Dachklappen selber sind Elemente, die bewegungsfähig sein müssen. Dazu ist ein Element notwendig, das Audiofeedback mit den Interaktionen verknüpft und eines, das Abhängigkeiten und Reaktionen umsetzt.

Bei der Interaktion mit diesen Elementen gibt es unterschiedliche Feedback Kategorien: Direkt, Indirekt und Extern. Direktes Feedback ist die unmittelbare Veränderung des Interaktions-Gegenstandes, das den Zustand des Objekts beschreibt, z.B. die Rotation eines Hebels. Indirektes Feedback beschreibt Reaktionen, die an anderen Einheiten des Frameworks hängen aber durch Verknüpfungen ausgelöst werden. Ein Beispiel ist das



(a) Umstellen der Hydraulik aus Wartungt

(b) Anschalten des Hydraulikkompressors

(c) Offenen der Rechten Dachluke

(d) Offenen der Linken Dachluke

Abbildung 2.4: Öffnen der Dachluken

Geräusch des Hydraulikkompressors, wenn dieser per Knopf eingeschaltet wird. Das Zurückkehren der Hebel oder des Drehknopfes in ihre Ausgangsposition ist ebenfalls eine Art von Feedback, die durch eine weitere Komponente realisiert werden kann. Externes Feedback beschreiben Reaktionen, die außerhalb des Frameworks liegen, dies wäre der Wärmestrahler, der nach dem Öffnen der Dachluken die Sonnenstrahlung simuliert oder ein Ventilator, der dieses für den Wind übernimmt.

Um dieses Feedback anzubieten wird eine Möglichkeit unterschiedliche Einheiten zu verbinden benötigt. Direktes Feedback kommt ohne weitere Komponenten aus, da es direkt mit dem Status der Interaktion verknüpft ist. Indirektes Feedback benötigt entsprechende Schnittstellen und Möglichkeiten, die auf diesen aufbauen um die Komponenten zu verknüpfen. Externes Feedback benötigen eine Komponente, die zwischen den Schnittellen und externen Modulen vermittelt.

2.5.2 Vorstellung Szenario 2

Im zweiten Szenario handelt es sich um die Prozedur, die es erlaubt den Aufzug in der Position oben zu betreten. Dieser Ablauf ist interessant, da er eine Vielzahl von Einzelschritten enthält, die in direktem Zusammenhang stehen.

Dieses Szenario startet mit dem Nutzer vor dem Aufzug 2.5. Der erste Schritt ist das Öffnen des Rolltores, das den Aufzug verschießt. Dieses erzeugt ein entsprechendes Geräusch als Feedback. Danach muss ein Sicherungsstift, der an einer Kette befestigt ist, aus seiner Fassung, im Aufzug, gezogen werden und in das Gegenstück an dem Tor vor dem Aufzug gesteckt werden. Das Aufnehmen und Ablegen aus den Fassungen wird durch ein Audiofeedback quittiert. Dieses entriegelt einen Bolzen, der das Tor vor dem Aufzug verriegelt. Wenn dieser zur Seite geschoben wird, was ebenso mit Audiofeedback verbunden ist, verriegelt dieses den Steckschlüssel. Dann kann das Tor geöffnet und der Aufzug betreten werden. Bevor der Aufzug nun genutzt werden kann ist es notwendig die zum Betreten ausgeführten Schritte in umgekehrter Reihenfolge wieder rückgängig zu machen, da sowohl der Zustand des Rolltores als auch der des Steckschlüssels elektronisch überprüft werden, bevor der Aufzug sich in Bewegung setzten kann.

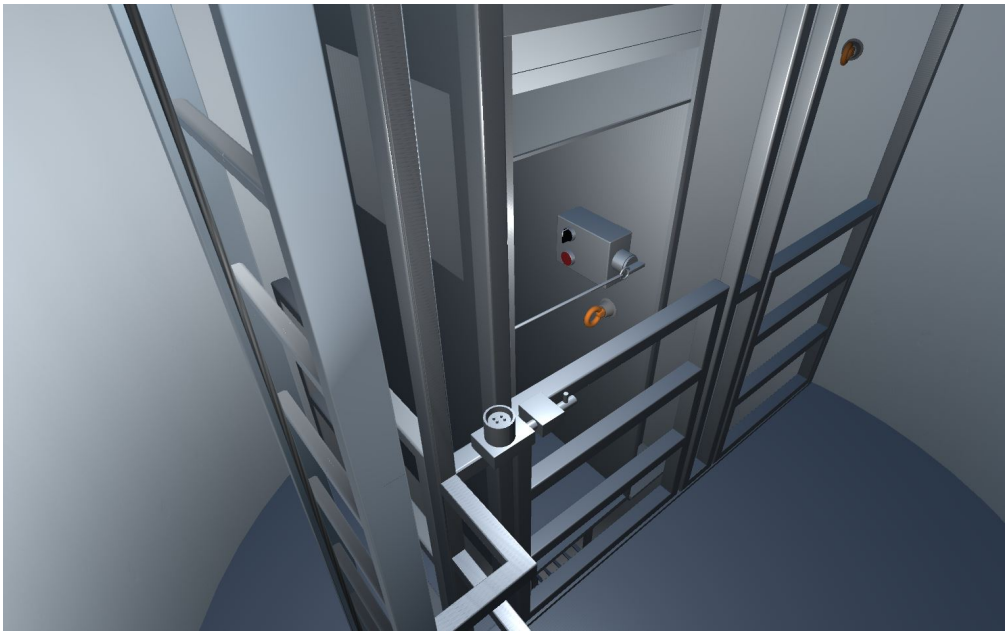


Abbildung 2.5: Der Aufzug in der oberen Position mit offenem Rolltor

Zu den Interaktionstypen von Szenario 1 kommen Interaktionen für das Rolltor und den Sicherungsstift hinzu. Die Absperrung vor dem Aufzug ist ein Hebel und der Bolzen kann wie ein Knopf behandelt werden. Hier ist besonders, dass einige der Interaktionen nur dann verfügbar sind, wenn andere in bestimmten Zuständen sind.

Dieses macht den besonderen Teil des Szenarios aus, denn wie geben Interaktionen Rückmeldung, wenn sie nicht verfügbar sind? Möglichkeiten sind Controller-Vibration

oder Audio um eine nicht erfolgreiche Interaktion zu signalisieren. Dieses ist eine weitere Schnittelle, an der Externes Feedback benötigt wird, das außerhalb des Frameworks liegt. Dafür wird hier wie auch beim ersten Szenario eine Komponente zur Umsetzung des globalen Verhaltens benötigt, während für die Aspekte des Audio Feedbacks sich eine wiederverwendbare Komponente anbietet.

2.6 Anforderungen

Aufbauend auf der Analyse können die Anforderungen für die Entwicklung eines Frameworks zur Unterstützung bei der Erstellung interaktiver Virtual Reality Szenarien aufgestellt werden. Dazu kommen die Ziele aus den vorherigen Arbeiten: Toolchain für 3D-Objektmanipulation in VR-Trainingsumgebungen [3] und Evaluation der Toolchain für 3D-Objektmanipulation in VR-Trainingsumgebungen [2]. Als erstes werden Anforderungen von der technischen Seite vorgestellt, gefolgt von den Anforderungen für die Seite der Nutzer.

1. **Vielseitige Interaktionen** - Für interaktive Environments sind die unterschiedlichsten Interaktionen notwendig, die alle unterstützt werden müssen. Das Framework muss diese unterstützen und Möglichkeiten bieten um fehlende Interaktionen mit minimalem Aufwand zu ergänzen.

2. **Szenarien** - Trainings benötigen gerichtete Abläufe, deren Erstellung unterstützt werden muss. Dazu muss es Hilfsmittel geben, die helfen Interaktionen zu Szenarien zu verknüpfen und Interaktionen müssen dieses durch Schnittstellen unterstützen.

3. **Skalierbarkeit** - Das Framework muss den Einsatz einer großen Anzahl von Interaktionen erlauben, da diese den Hauptbestandteil von Szenarien darstellen. Dafür müssen diese einen kleinen Footprint in der Datenstruktur und Rechenleistung aufweisen. Auch darf der Kommunikationsaufwand zwischen den Komponenten nicht zu groß sein, damit die Verbindungen für Feedback kein Problem werden.

4. **Erweiterbarkeit** - Offene Schnittstellen und ein System, das einfach erweitert werden kann, sind ein weiterer Beitrag für die schnelle Entwicklung. Möglichkeiten über Schnittstellen externe Software einzubinden werden ebenfalls benötigt. Zur Beschleunigung der Entwicklung sollen viel genutzte Interaktionen und Features bereits im Framework vorhanden sein.

5. **Unabhängigkeit** - Langlebigkeit erfordert eine möglichst hohe Unabhängigkeit von der Hardware. Hierzu zählt die Möglichkeit der Einbindung beliebiger Controller. Weitere Abhängigkeiten gilt es zu diesem Zweck zu vermeiden oder zu isolieren.

Nun folgen die Anforderungen für die Nutzer.

1. **Einrichtung und Konfiguration** - Projekte stehen oft unter Zeitdruck. Um dieses Problem nicht zu verschärfen, ist es wichtig, dass die Einrichtung und Konfiguration einfach und schnell für alle Elemente möglich ist und diese vernünftige Standardkonfigurationen besitzen.

2. **Schnittstellen** - Offene Schnittstellen und ein System, das einfach erweitert werden kann, sind ein weiterer Beitrag für die schnelle Entwicklung. Die Möglichkeit, über Schnittstellen externe Software einzubinden, ist ein weiterer Aspekt.

3. **Modularität** - Nicht jedes Projekt hat die gleichen Anforderungen und benötigt dieselben Features. Um die Verwendung des Frameworks hier zu unterstützen, sollen Funktionen modular getrennt sein, so dass eine Auswahl über die verwendeten getroffen werden kann

4. **Orthogonalität** - Die von dem Framework zu unterstützenden Funktionen und Interaktionen sind weit gestreut. Um die Verwendung zu erleichtern und den Lernaufwand zu reduzieren, soll sich die Bedienung aller Elemente orthogonal verhalten.

5. **Nutzergruppen** - Die unterschiedlichen Nutzer benötigen verschiedene Sicht- und Nutzungsweisen, um effektiv arbeiten zu können. Dieses erfordert ein Angebot von GUIs und Code-Schnittstellen, um die Nutzung zu erleichtern und dass sich die Features einheitlich verhalten, um den Lernaufwand zu reduzieren. Dafür sollten einzelne Komponenten unabhängig voneinander einzusetzen sein.

2.7 Resümee der Analyse

In diesem Kapitel wurde ausgehend von der Zielsetzung der Einsatzbereich analysiert und Anforderungen für ein Interaktionsframework festgelegt. Für Virtual Reality wurde festgestellt, dass die heutigen Game Engines den Endnutzern und Entwicklern vieles ermöglichen und erleichtern. Danach wurden die Selektions- und Manipulations-Phase von Interaktionen untersucht und die Bedeutung der genutzten Freiheitsgrade aufgezeigt. Zusätzlich wurden drei Kategorien von Feedback analysiert und vorgestellt: Intrinsisches, Direktes und Entferntes. Folgend wurde gezeigt, wie Virtual Reality für Training und Schulung Gefahren mindert, Ressourcen schont und diese zugänglicher macht. Zur Bewertung des entstehenden Frameworks wurden vier Kategorien mit ihren Eigenschaften vorgestellt. Diese ermöglichen die Entwicklung zu lenken um ein nutzerorientiertes Framework zu erstellen. Im Abschnitt danach wurden zwei Szenarien vorgestellt und ihre Forderungen an das Framework analysiert. Zur genaueren Definition des Frameworks wurden Anforderungen aufgestellt, die von der Technischen- und der Nutzer-Seite kommen. Sie bilden den Kern für die Implementation, die im nächsten Kapitel folgt.

3 Design

Dieses Kapitel beschreibt die Umsetzung des Frameworks aufbauenden auf der Analyse. Dazu werden als erstes Entscheidungen für den Aufbau aufgrund der Anforderungen getroffen und die Aufgaben der Module beschrieben. Danach folgt die Beschreibung der Umsetzung der einzelnen Komponenten zusammen mit einer Erklärung der Features und Tools, die den Entwicklungsprozess von interaktiven Szenarien unterstützen sollen.

3.1 Die Module

Mithilfe der Anforderungen lassen sich Design-Entscheidungen für das Framework festlegen.

1. Das Framework soll in Module geteilt sein, die sich jeweils auf eine Aufgabe spezialisieren.
2. Die Verbindungen zwischen den Modulen sollen auf minimale Schnittstellen begrenzt sein, um Abhängigkeiten zu vermeiden.
3. Die einzelnen Features der Module sollen möglichst klein und unabhängig sein, damit Nutzer diese schnell lernen können, ohne Verständnis über die anderen zu benötigen.
4. Die einzelnen Features sollen auf Vorlagen aufgebaut sein, von denen aus Erweiterungen vorgenommen werden können.

Da das Framework in Kombination mit Unity erstellt wurde, gibt es Features und Module, die von Unity genutzt werden können. Die Grafik und Physik Engine von Unity nehmen dabei grundlegende Arbeit ab und benötigen keine eigene Implementation. Zusätzlich gibt ein Eventsystem, das in Unity integriert ist und eine grafische Oberfläche bietet. Die Erstellung eigener UIs wird durch Unity unterstützt, hierfür teilt sich die

Funktionalität in zwei Gruppen. Die erste enthält den Teil, mit dem die Anwender der fertigen Szenarien in Berührung kommen. In der zweiten sind alle Funktionen, die zur Erstellung und Konfiguration benötigt werden. Unity unterstützt diese Teilung, um die zweite Gruppe aus dem Endprodukt herauszuhalten und den Overhead, der mit Editor-Funktionen einhergeht, zu eliminieren. Die unterstützenden Features zur Konfiguration können so aufwendiger ausfallen, ohne dass diese das Endprodukt beeinflussen.

Die Grafik 3.1 bietet eine Übersicht über alle Module des Frameworks, die Verknüpfungen und die Anbindung an das Hardware Abstraction Layer [9]. Die Grafik 3.2 beschreibt die Schnittstellen zu diesem genauer.

Das erste Modul enthält Werkzeuge für den Umgang mit Fehlern und Kontrolle, es soll bei der Nutzung des Frameworks und dem Erstellen von Szenarien unterstützen. Die Features sollen bei der Fehlersuche, der Vermeidung von Fehlern und dem Erstellen von Szenen helfen. Diese Features sollen nur im Editor verfügbar sein, um nicht in die fertige Anwendung einzugehen.

Das Controller Modul ist die Schnittstelle zwischen dem Input und den Interaktionen. In den vorherigen Arbeiten [3] [2] wurde es bereits genutzt, um die Unabhängigkeit von der Hardware zu schaffen, indem es unterschiedliche Controller abstrahiert. Im Rahmen dieser Arbeit soll das Controller Modul überarbeitet werden, um bessere Möglichkeiten zur Erweiterung und Abstraktion zu bieten.

Das Modul für die Interaktionen soll diese bereitstellen und Möglichkeiten bieten, diese zu erweitern. Für Szenarien sind Interaktionen der wichtigste Baustein und sie können in einer großen Anzahl auftreten. Entsprechend ist es für die Entwickler von Szenarien wichtig, dass diese einfach einzurichten sind und für die Performance, dass sie einen geringen Footprint aufweisen. Das Ziel ist, dass die Einzelnen eigenständig, klein und modular sind und in der Nutzung von einer überliegenden Schicht orchestriert werden können. Zusätzlich soll bei der Konfiguration durch eigene UIs und grafische Repräsentationen unterstützt werden. Zur Anpassbarkeit soll es einfach sein, neue Interaktionen hinzuzufügen.

Das Modul für Stories soll dabei helfen, die übergreifenden Zusammenhänge zu implementieren, die für die Kontrolle von Interaktionen notwendig sind. Dieses soll durch grafische Elemente möglich sein, die erweiterbar sind und Vorlagen, die bei der Implementierung komplexer Vorgänge helfen. Die Anbindungen an externe Komponenten sollen ebenfalls unterstützt werden.

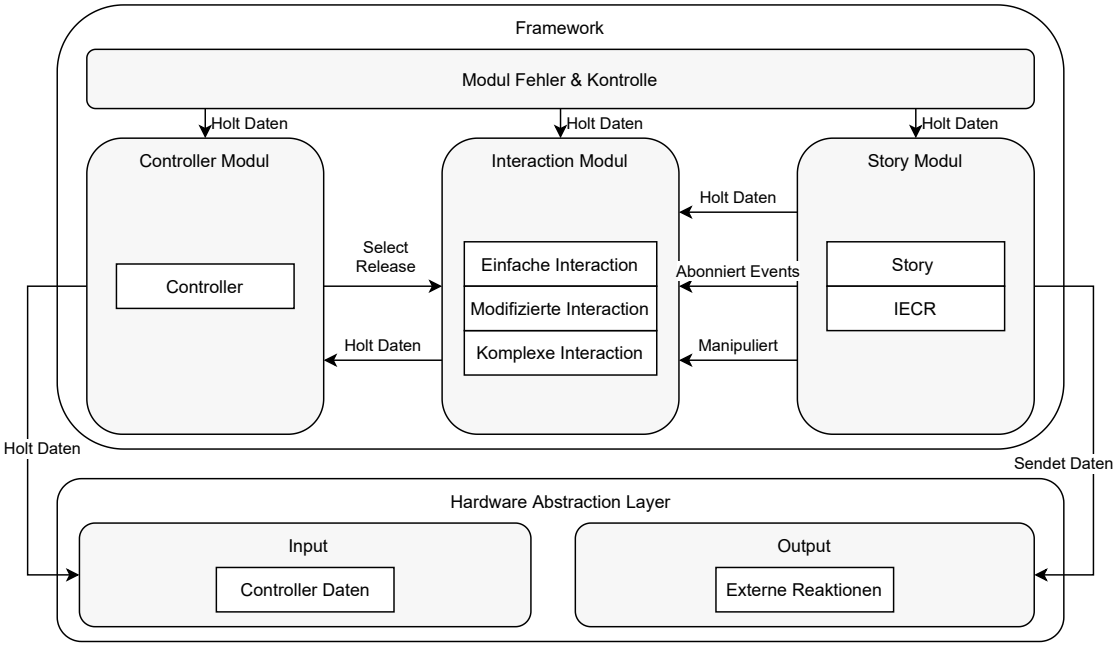


Abbildung 3.1: Übersicht über die Module des Frameworks

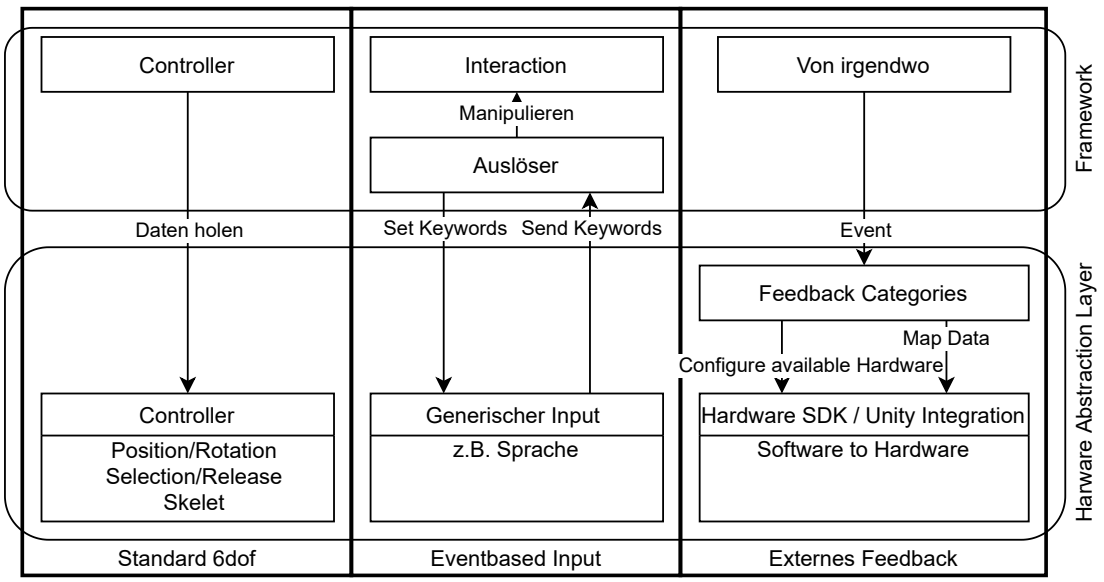


Abbildung 3.2: Schnittstelle des Frameworks zu anderen Modulen

3.2 Fehler & Kontrolle

Zur Erstellung von großen Szenarien sind Hilfsmittel notwendig, die Ordnung und Einheitlichkeit unterstützen und helfen, Fehler zu finden. In Unity gibt es viele unterschiedliche Wege, Projekte aufzubauen, weshalb es notwendig ist, dass die Hilfsmittel nicht durch Steifheit die Möglichkeiten einschränken. Um dieses Problem zu vermeiden, sollen die Werkzeuge keine Restriktionen auferlegen. Vielmehr ist es das Ziel, Werkzeuge zu erstellen, die gefilterte Rückmeldungen geben, bei denen der Nutzer selber entscheiden kann, was er sehen möchte und wie er mit ihnen umgeht. Wichtig ist auch, dass die Rückmeldungen möglichst früh verfügbar sind, damit die Probleme behoben werden können bevor sie auftreten.

Werkzeuge zur Konfiguration und Kontrolle haben dabei oftmals einen erheblichen Overhead und finden keine Anwendung im Compilierten Projekt. Dafür können die Werkzeuge im Editor Space liegen. Alternativ können dafür die von Unity verwendeten Platform directives mit der Platform Dependent Compilation [34] genutzt werden. Dieses ist bei Methoden hilfreich, die aufgrund ihrer Verwendung, zum Beispiel in OnValidate 3.2.1, nicht im Editor Space liegen können.

3.2.1 Error Log

In Unity gibt es unterschiedliche Fehlertypen die im Rahmen der Entwicklung auftauchen. Im Normalfall machen alle Fehler sich dadurch bemerkbar, dass sie in der Konsole ausgegeben werden. Fehler, die in der OnValidate Methode festgestellt werden 3.2.1, landen so jedes mal neu in der Konsole, was unübersichtlich ist und die Informationen nicht kompakt darstellt 3.3.

Das Error Log soll beim Beheben von Fehlern unterstützten und diese besser darstellen. Die Nützlichkeit des Werkzeugs soll sich darin zeigen, dass es duplizierte Fehler durch überschreiben reduziert, sowie die Anzeige von behobenen Fehlern entfernt werden können. Dieses soll die Konsole freihalten. Es soll zusätzlich die Möglichkeit geben, diese weiterhin in der Konsole anzuzeigen. Eine weitere Option ist das manuelle Starten einer Überprüfung, die für einen konfigurierbaren Bereich ausgeführt werden kann. Fehler, die nicht im aktiven Fokus liegen, sollen so gefunden werden können. Hierfür soll es möglich sein, eigene Überprüfungen hinzuzufügen.

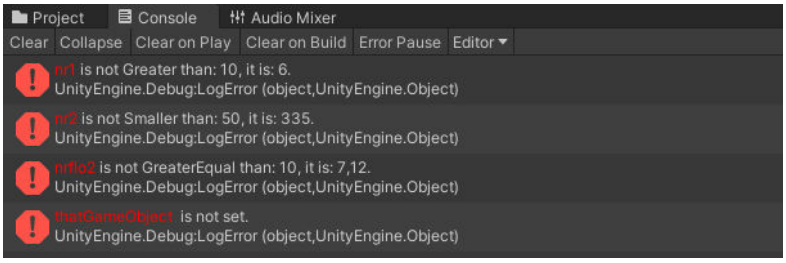


Abbildung 3.3: Die Standard Konsolen Ausgabe für festgestellte Konfigurationsfehler

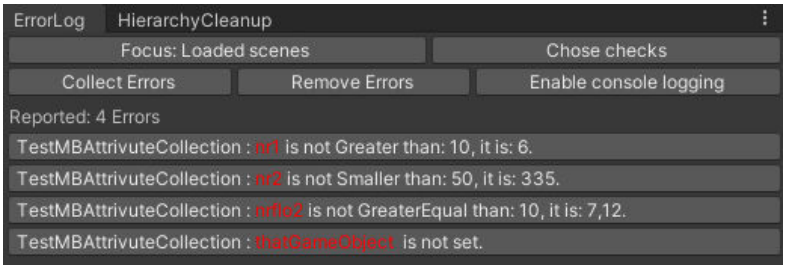


Abbildung 3.4: Das Error Log mit Beispiel Fehlern

Die Umsetzung findet komplett im Editor Space statt, es wird aber eine Brücke benötigt, damit das ErrorLog aus dem allgemeinen Kontext verfügbar ist. Die Brücke wird durch eine statische Referenz realisiert, die der Schnittstelle für das Error Log unterliegt. Das Registrieren wird durch Unitys InitializeOnLoadAttribute Attribute übernommen, das es erlaubt, nach jedem Kompilieren einen statischen Konstruktor auszuführen, der die Referenzen neu setzt. Über dieses Schnittstelle ist es möglich Fehler hinzuzufügen und zu entfernen. Damit diese eindeutig identifiziert werden können, wird die MonoBehavior zusammen mit einem Schlüssel übergeben, der den Typ des Fehlers spezifiziert. Beim Hinzufügen wird zusätzlich eine Nachricht übergeben, die im Log angezeigt werden soll.

Die manuelle Überprüfung wird direkt über das Editor Window ausgelöst, das auch zur Wiedergabe der Fehler dient. Für die manuelle Überprüfung kann als Erstes der Fokus der Überprüfung ausgewählt werden. Zu dem Zweck, dass die Art der Überprüfung konfigurierbar ist, können die Funktionen als Delegates eingehängt werden. Drei Typen werden dabei unterstützt: Einfache Ausführung, für alle GameObjects oder MonoBehaviors. Welche davon ausgeführt werden sollen, kann dann über die UI konfiguriert werden. Darunter werden die Fehler angezeigt 3.4 und mit einem Klick auf diese wird automatisch das zugehörige GameObject im Inspektor geöffnet.

Fehlende Referenzen

Unity bietet die Möglichkeit Referenzen durch Drag & Drop zu setzen. Die Anzahl der Drag & Drops, die in einem Projekt auftauchen, lässt dieses Verfahren schnell unübersichtlich werden. Fehlende Referenzen können dabei Fehler erzeugen, die erst spät in einem Szenario auffallen und somit erheblichen Zeitverlust beim Testen verursachen. Durch ein zusätzliches Modul für das Error Log wird dieses Problem minimiert. Dazu wurde eine Delegate eingehängt, die auf allen MonoBehavior Instanzen ausgeführt wird. Diese fragt ab, ob die Klasse entsprechend annotiert ist, und überprüft dann alle Felder, die serialisierte Referenzen enthalten, darauf ob diese gesetzt sind. Durch Nutzung von Vererbung ist diese Überprüfung für eine ganze Kategorie von Klassen implementiert, ohne in den Klassen selber zusätzlichen Code zu benötigen. Wenn ein Feld erst zur Laufzeit gesetzt werden soll, kann dieses über eine Annotation von der Überprüfung ausgenommen werden.

OnValidate

Unity bietet die Methode OnValidate() [33] an, diese erlaubt es automatisiert Eingaben im Inspektor zu überprüfen. Aufgerufen wird sie immer, wenn sich Werte im Inspektor geändert haben oder das Objekt neu geladen wird, z.B beim Öffnen einer Szene oder dem Rekompilieren von Code. Während diese Methode im Build zwar vorhanden ist, wird sie dort von Unity aber nicht länger aufgerufen.

Dieses bietet den optimalen Platz zur Implementation für die Konsistenz-Überprüfung der Felder, ob sie richtig ausgefüllt wurden und um dieses entweder zu korrigieren oder den Nutzer darüber zu informieren, z.B mit Hilfe des Error Logs.

3.2.2 Suche / Statistik

Das parametrisierte Suchen nach Objekten ist ein wiederkehrender Schritt in der Entwicklung. Während Unity die Möglichkeit besitzt, nach bestimmten Parametern in den Szenen zu suchen, ist das Angebot der Parameter nicht vollständig oder einfach erweiterbar. Kombinierte Suchen sind ein zusätzlicher Punkt, der für eine Erweiterung geeignet ist, um spezifischere Suchen zu ermöglichen.

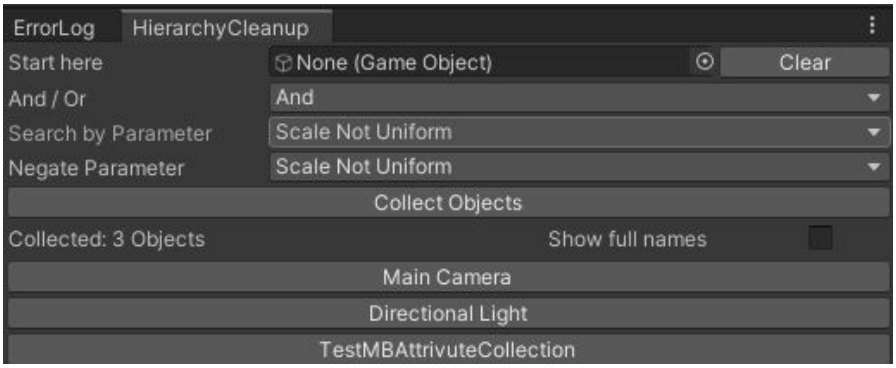


Abbildung 3.5: Das Hierachy Cleanup Fenster

Besonders wenn es darum geht, herauszufinden, ob eine bestimmte Kategorie von Objekten z.B in dem richtigem Physik-Layer liegt. Ein einfacher Fehler, der schnell passiert und dann erst beim Testen auffällt, wenn ein Objekt nicht reagiert. Dies ist schwierig zu debuggen, da ein solcher Fehler eine Vielzahl von Ursachen haben kann.

Das entstandene Werkzeug HierachyCleanup 3.5 kann in diesem Fall helfen. Der Nutzer kann mit ihm in einem selbst festgelegten Bereich mit konfigurierbaren Suchoptionen suchen. Für die Suchoperatoren kann festgelegt werden, ob diese genutzt werden sollen und ob sie wahr oder falsch sein sollen. Um die Auswahl dieser nicht einzuschränken, können mit Code weitere Suchoperatoren an die Kollektion übergeben werden. Diese bestehen aus dem Namen für den Suchoperator und einer Methode, die eine MonoBehavior oder ein GameObject entgegennimmt und einen Wahrheitswert zurückgibt. Suchoperatoren, die Parameter akzeptieren, brauchen zusätzlich noch eine Methode, die im Editor Window die UI für die Parameter anzeigt und diese in einer übergebenen Key-Value Datenstruktur speichert.

Die Performance für das Suchen nach Objekten skaliert mit der Größe der Szenen. Sollten diese zu groß werden und zu viele Suchoptionen auf einmal verwendet werden, kann der Prozess längere Zeit in Anspruch nehmen. Da es sich hier aber um einen Prozess handelt, der nicht kontinuierlich läuft, ist die Laufzeit meist zu vernachlässigen und es wäre möglich, die Suche über einen Zeitraum auszudehnen.

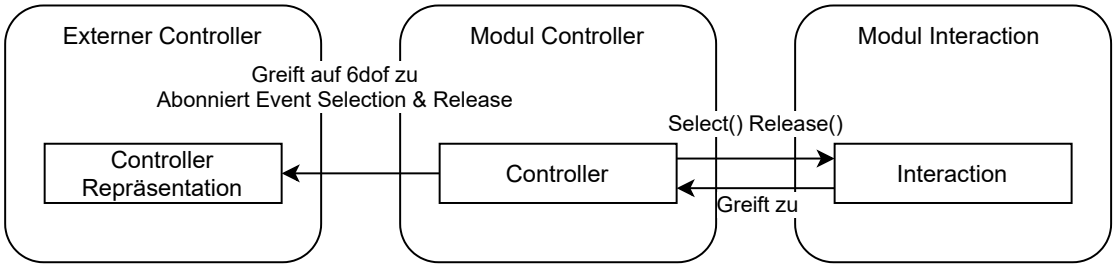


Abbildung 3.6: Modul Controller

3.3 Controller Modul

Das Controller Modul bildet die Schnittstelle zwischen Input und Interaktionen. Während für viele Controller Schnittstellen zur Einbindung in Unity existierten, sind diese in ihrer Beschaffenheit sehr unterschiedlich. Zur Erfüllung der Anforderungen an die Hardwareunabhängigkeit sind so Strukturen zum Einbinden unterschiedlicher Input Modelle notwendig, die mit beliebigen Systemen kompatibel sind.

Die Abstraktionsaufgaben des Moduls sind in zwei Kategorien geteilt. Die Erste ist für Input in der Form 6dof mit Selection- und Release-Event, z. B. bei den meisten in der Hand gehaltenen Controllern. Input dieser Art ist bevorzugt und kann direkt vom Adapter weiter verarbeitet werden. Entsprechend ist es einfach, den Adapter um andere Controller mit den gleichen Voraussetzungen zu erweitern 3.3.1.

Die zweite Kategorie behandelt allen anderen Input. Dieses wäre ein Controller mit 3dof, eine Mischung zwischen Sprach- und Blick-Steuerung oder die Bedienung über einen Bildschirm mit 2dof Input. In dieser Kategorie ist es nicht zu vermeiden, individuelle Lösungen für alle unterschiedlichen Typen von Input zu erstellen 3.3.2. Auch wenn die Möglichkeit zur Einbindung existieren soll, ist dies nicht der Fokus des Moduls.

3.3.1 6dof Controller

Der häufigste Controller Typ für Virtual Reality Anwendungen besitzt 6dof, da dieses es erlaubt, die Position und Rotation der Hände wiederzugeben. Zusätzlich verfügen die Controller abhängig von ihrem Typ über eine weitere Anzahl von Input Möglichkeiten 2.1.1.

Für Controller dieses Typs 6dof bietet sich eine allgemeine Abstraktion an, von der aus durch Vererbung spezifische Implementationen abgeleitet werden 3.7. Dies sind zum Beispiel der Standard Controller 3.3.1, der allgemein SteamVR Controller abbildet und der Fake Controller 3.3.1, der sich zum Debuggen eignet oder zur Nutzung durch Code gedacht ist.

Die abstrakte Klasse für Controller enthält die notwendigen Eigenschaften und Methoden. Folgende Parameter zur Konfiguration sind vorhanden: AllowUse kann die Verarbeitung von Events unterbinden. Zum Beispiel um zu verhindern, dass während Überblendungen Gegenstände losgelassen werden können und verloren gehen. Die LayerMask für die Konfiguration der Physik-Layer zur Selektion, TriggerDown um den Zustand der Selektion zu speichern und eine Liste zur Bereithaltung der gegriffenen Elemente.

Diese Klasse erbt von MonoBehavior, damit sie in der Unity Szene auf einem GameObject platziert werden kann und die Parameter über den Inspektor konfiguriert werden können. Jene Interaktionen, die Position und/oder Rotation als Input benötigen, können diese so direkt abgreifen. Dieses setzt voraus, dass sich das GameObject in der Hierarchie der Szene unter einer Repräsentation des Controllers befindet oder eine zusätzliche Methode zum kontinuierlichen Setzten der Position und Rotation des GameObjectes implementiert wird.

Die Methoden für Selection und Release sind vorhanden und rufen die entsprechende Select- und Release-Methode mit dem Controller als Parameter der Interaktionen auf. Für die Implementation von weiteren Funktionen wird bei Selection und Release für jedes Element eine Methode aufgerufen, die von den erbenden Klassen zu implementieren ist. Ebenso implementiert diese Klasse die Standardselection. Diese besteht aus einem mehrstufigen Verfahren. Dazu werden eine konfigurierbare Anzahl von sphärischen Suchen mit größer werdendem Radius gemacht. Dieser Prozess bricht ab, sobald mindestens ein GameObject gefunden wurde, das von der abstrakten Klasse der Einfachen Interaktionen erbt. Dies erleichtert die Selektion von Gegenständen und erlaubt eine hohe Präzision und Flexibilität, da so die Gefahr der mehrfachen Selektion reduziert wird.

Zur einfacheren Nutzung aller Klassen, die von der abstrakten Controller Klasse erben, werden zum Inspektor dieser Klassen Knöpfe zum Aufrufen der Methoden für Selection und Release hinzugefügt 3.8.

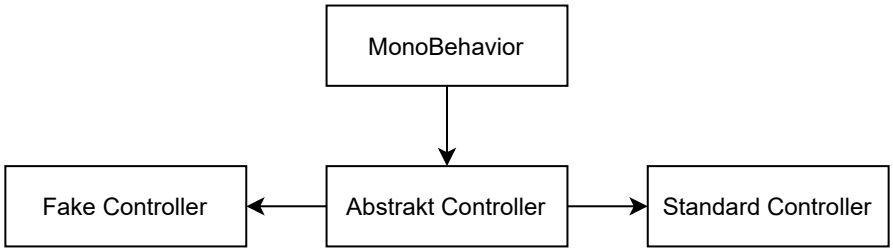


Abbildung 3.7: Übersicht über die Vererbung der Controller

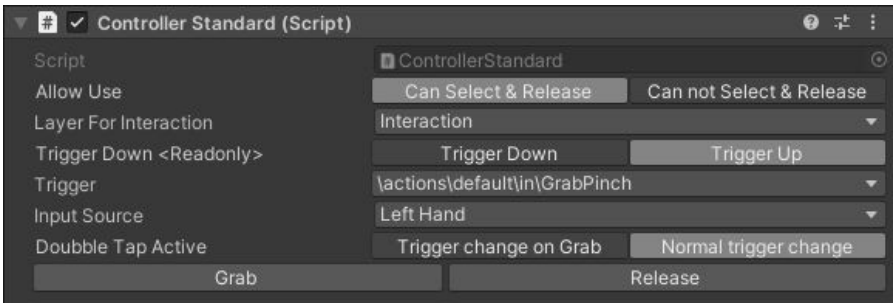


Abbildung 3.8: Inspektor des Standard Controllers

Standard Controller

Auf Basis der abstrakten Klassen kann nun ein Standard Controller für SteamVR implementiert werden 3.8. Zur Konfiguration sind zusätzliche Felder notwendig: Eines zur Auswahl des Input Gerätes und ein anderes zur Selektion des auslösenden Events. Notwendig ist es dazu die Selection- und Release-Methode an die konfigurierten Events zu knüpfen, damit diese aufgerufen werden können. Die Methoden für zusätzliche Funktionen können leer gelassen werden.

Fake Controller

Ein Fake Controller ist ein weiteres Beispiel, das beim Testen hilfreich sein kann. Er ist nicht dynamisch an eine Controller Repräsentation angebunden und kann nur manuell im SzeneView von Unity genutzt werden. Dazu müssen die Methoden für Extrafunktionen bei Selection und Release können aber leer bleiben, da kein zusätzliches Verhalten notwendig ist.

3.3.2 Andere Controller

Controller, die nicht 6dof unterstützen, benötigen eine kontextabhängige Übersetzung. Das Problem dieser Controller ist, dass mit zunehmender Anzahl der Interaktionen auch die der notwendigen Übersetzungen mitwächst. Dieses verhindert eine allgemeingültige Implementation für alle Interaktionen. Im Rahmen einzelner Szenarios ist der Einsatz eines Speziellen Controllers aber dennoch möglich.

Eine Lösung hierfür ist die Verwendung von Event-Aktion Paaren. Im Falle einer gemischten Sprach-Blick-Steuerung könnte Folgendes implementiert werden: Bei der Erkennung eines Befehls z.B. „Öffne Tür“ wird entlang des Blickes ein Raycast ausgeführt. Wenn dieser eine Tür trifft, wird versucht, diese direkt über ihre Methoden zu öffnen. Für Event-Aktion Paare könnte eine Oberfläche, die den IECRn 3.5.1 ähnlich ist, genutzt werden, um die Events in Selektion und Aktion zu übersetzen.

Zusätzliches Vorgehen ist notwendig, wenn die Select und Release Events 3.4.2 der Interaktion im Szenario gebraucht werden. Dieses lässt sich umgehen, in dem der Controller darum erweitert wird, dass er vor der Manipulation mit einem Fake Controller die Interaktion selektiert und hinterher loslässt.

Bei einem anderen Beispiel, der Steuerung über den Bildschirm, kann dieses Problem umgangen werden. Der Aufwand steigt für die Implementation dabei, da so für jede Interaktion der Input aus dem 2D auf in den 3D Raum übertragen werden muss. Dieses Prinzip lässt sich mit entsprechendem Aufwand auch auf andere Input-Verfahren übertragen.

Das Ergebnis ist, dass zur Verwendung eines solchen Systems viel Konfiguration notwendig ist. So muss kontextsensitiv mit den Interaktionen umgegangen werden und die Auswahl der verwendeten Interaktionen für das Szenario eingeschränkt werden.

3.4 Modul Interaktionen

Interaktionen und wie sie genutzt werden, sind vielfältig und benötigen eine entsprechende Grundlage. Diese soll das Modul für die Interaktionen schaffen. Vorlagen sollen dabei helfen, vorhandene Interaktionen zu nutzen und neue zu erstellen. Dazu geht es als Erstes um die Grundlagen 3.4.1 aus den vorherigen Projekten [3] und [2] und die Weiterentwicklung. Die verwendete Eventstruktur ist der nächste Punkt 3.4.2, zusammen mit

ihrer Überarbeitung. Dieses wirkt sich auch auf die Einfachen Interaktionen und deren Aufbau aus, was in 3.4.3 behandelt wird. Danach geht es um Modifikationen für Interaktionen 3.4.4, die keine neue Interaktion oder Erweiterung rechtfertigen. Der letzte Punkt beschäftigt sich mit Komplexen Interaktionen 3.4.5, die sich nicht mehr mit nur einer Interaktion beschreiben lassen.

3.4.1 Grundlagen und Weiterentwicklung

In den vorherigen Projekten [3] und [2] wurde bereits gezeigt, dass es möglich ist, Interaktionen effektiv mit Vererbung zu implementieren. Dieses ist durch die hohe Ähnlichkeit vieler Interaktionen gegeben. Um diesen Zustand zu verstärken, wurden sie in drei Kategorien geteilt. Dieses erlaubt eine stärkere Bündelung von Funktion, die auch in der Grafik A.3, die einen Ausschnitt der Interaktionen zeigt, zu sehen ist. Interaktionen blieben dadurch klein und modular und ließen sich erfolgreich für Szenarien nutzen. Auch wurden Eigenschaften wie die Orthogonalität und Abstraktion aufgestellt und hatten sich in der Nutzung als effektiv gezeigt, um Interaktionen in Virtual Reality nutzen zu können. Wichtig ist auch die Vereinheitlichung der Begriffe zu Selection und Release die vorher auch Greifen/Grab und Loslassen/Deselektion hießen.

3.4.2 Events und Zustände

Für die Weiterentwicklung des Frameworks haben bei den Events und Zuständen folgende Änderungen stattgefunden. Ursprünglich war das Senden von Events an die fünf Zustände: Disabled, Inactive, Grabbed, Running und Service geknüpft. So sendeten sie in den Zuständen Disabled und Service keine Events, außer TrySelect und TryRelease und ließen sich nur im Service Mode direkt über Code manipulieren.

Dieses hat sich als nicht praktikabel herausgestellt, da so Verkettungen von Events durch Code nicht möglich waren. Ebenso hat es den Aufwand erhört, Interaktionen direkt zu beeinflussen, da erst zu Service und dann wieder zurück gewechselt werden musste. Problematisch ist daran, dass so unklar ist, dass die Zustandsräume der Interaktionen inkonsistent sind. Bei der Erstellung neuer Interaktionen kann dies zu unerwünschten Zuständen führen, wenn dies nicht bedacht wird. Mit der neuen Auslegung ist es nun eindeutig, da diese Problematik einheitlich geregelt ist. Ebenso ist es nun eindeutig, wann

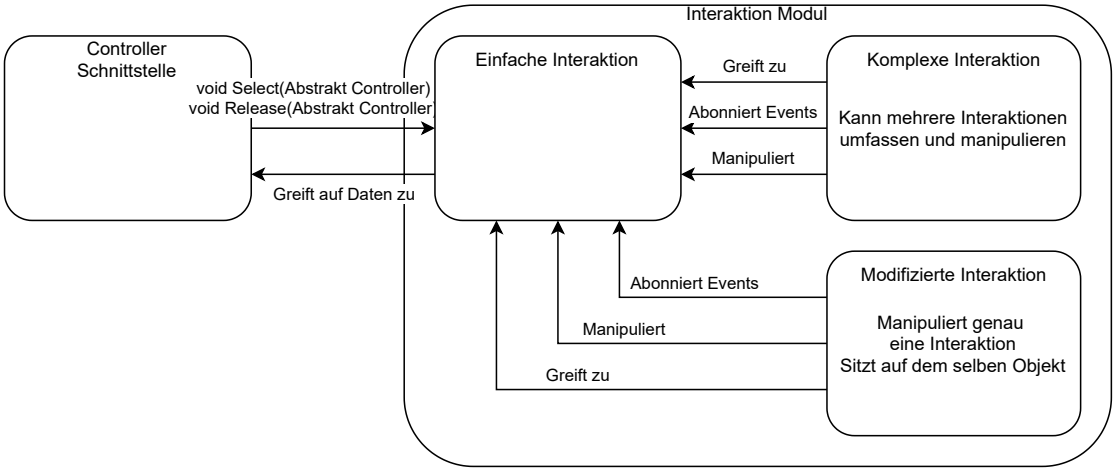


Abbildung 3.9: Aufbau des Modul Interaktionen

Events ausgelöst werden. Die Implementationen können als Konsequenz übersichtlicher gestaltet werden.

Die Änderungen sehen dabei wie folgt aus. Das Senden von Events wurde vollständig an eine eigene Option, SendEvents, gebunden so dass sich nun alle gleich verhalten. Dieses zieht die Option aus den Zuständen heraus, so dass sie nun wie folgt definiert sind.

- **Disabled** Die Interaktion kann nicht beeinflusst werden.
- **Inactive** Die Interaktion wartet auf Input.
- **Grabbed** Die Interaktion ist gerade gegriffen und reagiert auf den verlinkten Controller.
- **Running** Die Interaktion ist nicht gegriffen und ändert sich nach internen Regeln.
- **Service** Die Interaktion reagiert nur auf Input durch Programmcode.

Auch bei den Events gab es entsprechende Änderungen. In der vorherigen Version war es notwendig jedes Selektiert- oder Released-Event zu untersuchen um manuell festzustellen ob es erfolgreich war. Zusammen mit der nun allgemeingültigen Definition erleichtert dieses das Verständnis über der Funktionsweise. Durch die Erweiterung um die Events SelectFail und CanSelect sowie das Speichern des versuchenden Controllers kann einfacher mit dem Selektionsprozess interagiert werden. Dieses kann für die Auswertung oder um

neue Funktionen zu schaffen interessant sein. Die Übersicht über die Änderungen sind in der Grafik 3.10 zu sehen.

Ein Beispiel für den Nutzen ist die Tür in Szenario 2, beschrieben in 2.5.2, die sich nur dann öffnen lässt, wenn der Bolzen gelöst ist. Vorher war es notwendig bei jeder Selektion zu prüfen, in welchem Zustand das Tor war. Nun ist dieses nicht mehr notwendig und es kann mit dem SelectFail Event direkt Feedback ausgelöst werden, um die nicht erfolgreiche Selektion zu verdeutlichen.

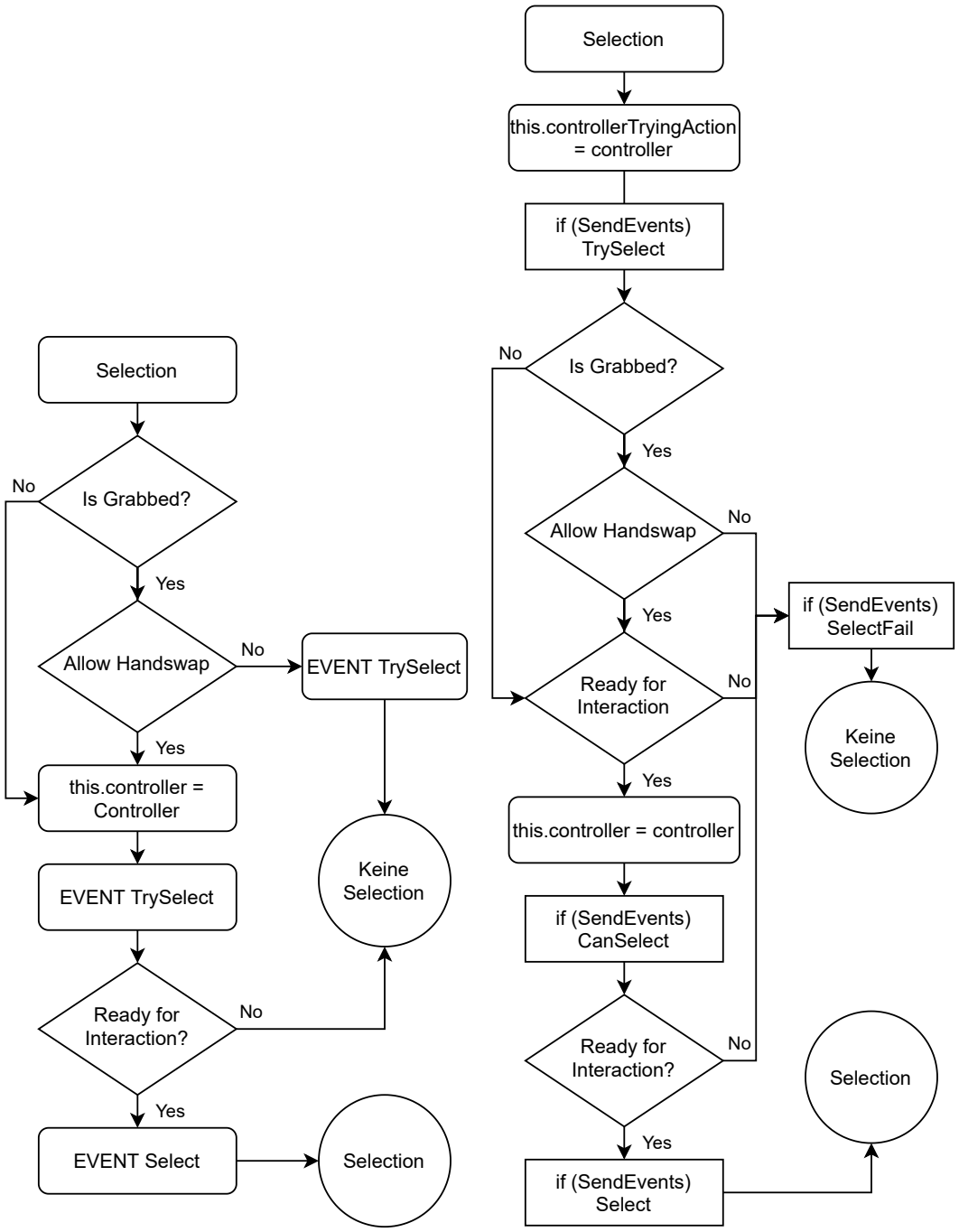
3.4.3 Einfache Interaktionen

Einfache Interaktionen bilden den Hauptbestandteil der Szenerien. Damit diese entsprechend skalieren, sind sie wie in vorherigen Projekten [3] und [2] beschrieben möglichst klein und unabhängig. Dies erlaubt das schnelle Einrichten und Anwenden auf Objekten und reduziert den Konfigurationsaufwand. Sie können durch Klassen, die vom abstraktem Controller erben, in 3.3.1 beschrieben, Selektiert und Released werden und bilden elementare Interaktionen ab, wie Hebel, Drehköpfe, Schieben und Tragen. Trotz der vielseitigen Interaktionen haben diese vieles gemeinsam. Ihre Lebenszyklen sind unendlich, sie besitzen keinen Zustand, der das Ende ihrer Interaktivität bedeutet. Die Zustände teilen sie sich ebenso und die Standardevents für Selektion und Loslassen, für die im nächsten Kapitel ein Beispiel aufgelistet ist.

Leben einer Tür

Unter diesen Bedingungen lässt sich das Leben einer Interaktion an Hand einer Tür, einem Hebel-Objekt, wie folgt beschreiben. Das Ziel ist eine Tür, wie sie in Szenario 2 2.5.2 beschrieben ist, die erst nach dem Ausführen einer anderen Interaktion selektierbar wird. Dabei werden Funktionen aufgerufen, deren Management auf der Unity Seite liegt. Die Übersicht über die Reihenfolge ist in der Grafik Unity Execution Order A.4 zu sehen.

Beim Laden der Szene wird einmalig die Methode Awake() aufgerufen, wo die UnityEvents initialisiert werden. Im nächsten Schritt wird von der Start() Methode eines Feedback Scripts, sie wird immer nach Awake() aufgerufen, das Event SelectFail abonniert, um in diesem Fall ein Rüttelaudio abzuspielen. Danach ist die Initialisierung der Komponenten abgeschlossen.



(a) Alter Ablauf (b) Neuer Ablauf

Abbildung 3.10: Ablauf der Selektion. Links alt ([3, nach Abb. 7]), rechts neu. Der Code dazu A.2

Wird dann versucht, die Tür im Zustand Disabled zu Greifen, in dem sie konfiguriert wurde, löst erst das TrySelect und dann das SelectFail Event aus und der Vorgang bricht ab. Der nächste Schritt, das Ändern des Zustandes von Disabled zu Inaktiv, wird ausgeführt, wenn die Vorbedingung für das Öffnen des Tores erfüllt wird.

Wird die Tür nun gegriffen, geschieht dieses erfolgreich mit einem TrySelect, CanSelect und dann Select Event, wonach der Zustand zu Grabbed geändert wird. Nun reagiert die Tür auf die Position des Controllers, bis die Tür losgelassen wird. Hierbei wird der Zustand zu Inactive geändert und TryRelease eine CanRelease und ein Release Event ausgelöst.

Implementation einer Interaktion

Zur Erstellung neuer Interaktionen bieten sich folgende Schritte zur Erleichterung, die Checkliste dazu in Abbildung 3.11. Bei der Implementation einer neuen Interaktion ist das gewünschte Verhalten der Ausgangspunkt für die Implementation. Als Erstes bietet es sich an, die Parameter der Interaktion zu definieren und implementieren, gefolgt von den Konsistenzchecks, um fehlerhafte Konfigurationen zu vermeiden. Es gilt festzulegen wann und wodurch sich die Eigenschaften einer Interaktion ändern. Möglich ist dies in der Selections und/oder Release Phase, explizit über Methoden oder im Update mit den Zuständen Grabbed und Running. Für die notwendigen Wege der Manipulation ist es dabei wichtig, dass diese gegeneinander abgesichert sind und es nicht durch eine beliebige Reihenfolge von Aufrufen zu Problemen kommen kann. Wenn die Interaktion Zustandsübergänge besitzt, ist es nützlich, diese als Events auszugeben. Als Letztes bietet es sich an, noch Methoden für die Benutzung durch die IECR zu implementieren, siehe 3.5.1.

3.4.4 Modifizierte Interaktionen

Es gibt bestimmte Modifikationen, die wiederholt für unterschiedliche Interaktionen vorkommen, z.B für Hebel das Zurückkehren zur Ausgangsposition. Diese Funktion wird aber nicht von allen Hebeln benötigt und würde, wenn direkt implementiert, beim Erben unnötigen Ballast erzeugen und weitere Implementation erschweren. Eine weitere Klasse zu erstellen, die nur um diese Funktion erweitert ist, führt ebenso zu Unübersichtlichkeit, besonders wenn weitere Objekte dieselbe Funktion benötigen. Eine Lösung für dieses

1. Gewünschtes Verhalten ermitteln
2. Aufstellung der Parameter
3. Automatische Konsistenzchecks der Parameter implementieren
4. Implementation der Manipulation
5. Erstellung der Events für die Zustandsübergänge
6. Erweiterung um IECR Methoden (siehe 3.5.1)

Abbildung 3.11: Checkliste zur Implementation einer Interaktion

Problem sind Modifikationen, die Interaktionsobjekte von außen beeinflussen. Die Implementation kann dann zusätzlich für ein Objekt weiter oben in der Hierarchie stattfinden, so dass diese möglichst breit einzusetzen ist. Selbst wenn die Funktion für Objekte ohne gemeinsame Basisklasse implementiert werden soll, ist es möglich, die Modifikation um entsprechende Wrapper zu erweitern. Zusätzlich können diese nicht selektiert werden und müssen so durch Events über anstehende Aktionen informiert werden.

Als Vorlage gibt es die Klasse Abstract Interaction Modification Object, diese bringt den Check mit, ob alle Referenzen, die das Objekt enthält, gesetzt sind und das entsprechende Feld um dieses anzuzeigen. Sie erbt dabei von der Klasse AOBJWithState, die die Zustände der Interaktionen implementiert, damit diese sich identisch mit den Einfachen Interaktionen steuern lassen. Die Modifikations Scripte sollten dabei für die Übersichtlichkeit immer auf dem Objekt liegen, das sie modifizieren. Sie dürfen dabei nur ein einziges Objekt modifizieren, können aber von beliebigen lesen. Dieses kann genutzt werden, um Input auf Einfache Interaktion zu übertragen oder Objekte wieder in ihre Ausgangsposition zurückkehren zu lassen.

Modifikation ReturnToPosition

Mit der Hilfe der Vorlage kann eine mögliche Modifikation erstellt werden. Für die Erstellung kann auch die Checkliste im Kapitel Implementation einer Interaktion 3.4.3 verwendet werden.

Diese Modifikation soll es einem Hebel-Objekt ermöglichen, in die Ausgangsposition zurückzukehren. Dieses soll immer dann stattfinden, wenn das Objekt nicht gegriffen ist.

Abbildung 3.12: Inspektor der Interaktions-Modifikation Return to Position

Hier kann die Verallgemeinerung geplant werden, die Modifikation für alle Objekte, die von AbstractRotation erben (Klassen Übersicht Abbildung A.3) gültig zu machen.

Die notwendigen Parameter definieren, zu welcher Position zurückgekehrt werden soll und wie schnell dies geschehen soll, siehe Abbildung des Inspektors 3.12. Der Konsistenzscheck in OnValidate muss nur prüfen, ob die Geschwindigkeit positiv ist, da das Feld für die Ausgangsposition ein Auswahlfeld ist und nicht falsch konfiguriert werden kann. Aktiv wird dieses Script immer, wenn das Release-Event der verlinkten AbstractRotation ausgelöst wird. Die Bedingung ist dabei, dass das Objekt erfolgreich initialisiert wurde (Sucsessfull Load) ist und der Zustand der Modifikation nicht Disabled ist. In diesem Fall wird der Zustand auf Running gesetzt und das Objekt bewegt sich mit der eingestellten Geschwindigkeit in die eingestellte Position. Dort angekommen geht das Script wieder in den Zustand Inaktive.

3.4.5 Komplexe Interaktionen

Komplexe Interaktionen werden darüber definiert, dass sie mehrere einfache Interaktionen zu einer neuen kombinieren und so nicht als eine Erweiterung einer Einfachen Interaktion definiert werden können. Im Gegensatz zu den Modifikationen sind die Komplexen Interaktionen nicht einer einzigen Interaktion zugeordnet.

Sie enthalten die Zustandsmaschine für Übergänge und Verhalten zwischen den Interaktionen sowie die Möglichkeit, die verbundenen Interaktionen um zusätzliches Verhalten zu erweitern. Entsprechend sind diese genau wie Einfache Interaktionen für spezielle Ziele optimiert und sind spezifisch einzusetzen. Während diese wie die Modifikationen nicht

über Controller ausgelöst werden, können die Events der verlinkten Interaktionen genutzt werden, um auch den internen Zustand aktiv zu setzten. Die Vorlage gleicht den Modifizierten Interaktionen abgesehen vom Namen.

Messgerät

Die Komplexen Interaktionen ermöglichen nun Umsetzungen wie ein beidhändig zu bedienendes Messgerät für elektrische Spannungen, das auf vorhandenen Interaktionen aufbaut. Die beiden Teile des Messgerätes sind Einfache Interaktionsobjekte, die einzeln gegriffen werden können und dann dem Controller folgen. Die für die Funktion des Messgerätes nötigen Teile werden von der Komplexen Interaktion Messgerät verwaltet. Notwendig sind hierfür Referenzen auf die beiden zu haltenden Teile sowie Objekte, die beschreiben, wo die Messspitzen sind. Zur Anzeige des Ergebnisses ist ein Textfeld referenziert und zur Verdeutlichung, ob grade gemessen wird, Farben für die unterschiedlichen Zustände.

Im Inneren befindet sich dann ein Zustandsautomat. Im Zustand Inactive werden keine Aktionen ausgeführt. Der Übergang zu Running findet statt, sobald einer der beiden Teile des Messgerätes gegriffen wird und das Script nicht im Zustand Disabled oder Service ist. Die Schleife, die im Running Zustand läuft, prüft zuerst, ob beide Teile gegriffen sind, wenn dies nicht der Fall ist, geht das Script direkt wieder zu dem Zustand Inactive über. Der Grund für die Prüfung in der Schleife liegt daran, das zu dem Zeitpunkt des Select-Events die Zustände noch nicht geändert sind. Folgend wird an beiden Messpunkten nach GameObjects mit passenden Komponenten gesucht. Wenn für beide Messpunkte ein Kontakt gefunden wurde, werden die Werte aus dafür angelegten Objekten ausgelesen und die Werte auf der Anzeige und die Displayfarbe aktualisiert. Wenn nicht für beide welche gefunden wurden, wird das Display auf Off gesetzt und ein konfigurierbarer Text angezeigt.

3.5 Modul Story Entwicklung

Das Modul Story hilft dort, wo die Möglichkeiten der Interaktionen aufhören Zusammenhänge zu schaffen. Keiner der Interaktionstypen kann über seine internen Möglichkeiten hinaus agieren und Zustände speichern. Es ist zwar möglich, für komplexes Verhalten Interaktionen beliebig weit zusammen zufassen, doch reduziert dieses die Übersichtlichkeit.

Ebenso sind Interaktionen ab einer bestimmten Abstraktion nur noch lose verbunden und nicht jede Reaktionen betrifft eine Interaktion im selben Verbund. Übergreifende Abläufe sind dabei von sich aus oft linear, was der Wiederholbarkeit der Interaktionen widerspricht. Oder sie benötigen spezielle Zustände, was der Wiederverwendbarkeit der Interaktionen entgegensteht. Dasselbe gilt für die Einbindung von externem Code, wenn die Interaktionen mit anderen Frameworks zusammenspielen sollen.

Um Nutzern unterschiedlicher Erfahrungsgrade diese Möglichkeit zu bieten, sind unterschiedliche Werkzeuge nötig. Damit die Zusammenarbeit den zwischen verschiedenen Erfahrungsstufen funktioniert und sich das Lernen neuer Features lohnt, ist es notwendig, dass die einzelnen Komponenten dabei miteinander kompatibel sind.

Für Einsteiger eignen sich visuelle Mittel, um dieses zu ermöglichen. Der gewählte Weg sind in diesem Fall vorgefertigte IECR (Interaktion Event Controll Reaktion), siehe 3.5.1. Sie erlauben das Verknüpfen von Events mit Funktionen und das Aufstellen von Bedingungen.

Fortgeschrittenen ist es möglich, eigene IECR aus vorgefertigten Teilen zusammenzustellen. Auch in ihrer Befähigung ist es einfache Stories 3.5.2 mit der Vorlage der abstrakten Story Klasse zu schreiben, indem die Schnittstellen zu den Interaktionen genutzt werden.

Experten können mit Hilfe der Vorlagen komplexe IECR bauen oder diese für andere Systeme anpassen, um neue Möglichkeiten und Erleichterungen zu schaffen.

3.5.1 IECR

IECR (Interaction Event Control Reaction) sind die schnellste Art, Interaktionen miteinander zu verknüpfen. Basierend auf der Vererbungsstruktur der Interaktionen, bieten diese eine Vielzahl von Events an, siehe Kapitel 3.4.2. Aus Gründen der Übersichtlichkeit und Performance zeigen Interaktionen ihre Events aber nicht im Inspektor, sind aber durch Code verfügbar. Die IECR erlauben es, mit diesen visuell zu interagieren. Sie bestehen dabei aus drei grundlegenden Elementen. Das erste Element spezifiziert das auslösende Event, das zweite die Möglichkeit, eine Bedingung zu setzten und zu überprüfen. Das dritte Element spezifiziert eine Reaktion. Ein Beispiel hierfür ist der IECR trigger with condition, siehe Abbildung 3.13, welcher alle drei Kategorien benutzt. Damit diese auch mit anderen Werkzeugen und sich selbst kombinierbar sind, verfügen sie über

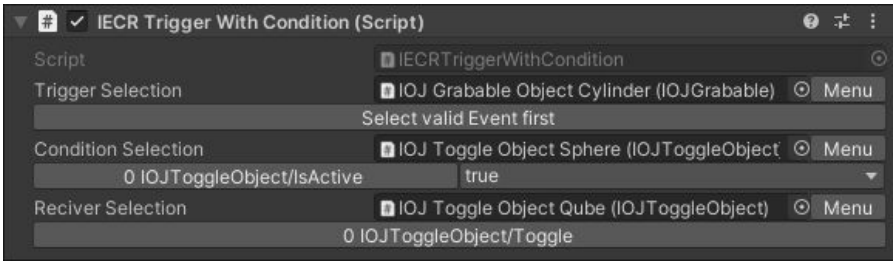


Abbildung 3.13: Bild des Inspektors von IECR trigger with condition

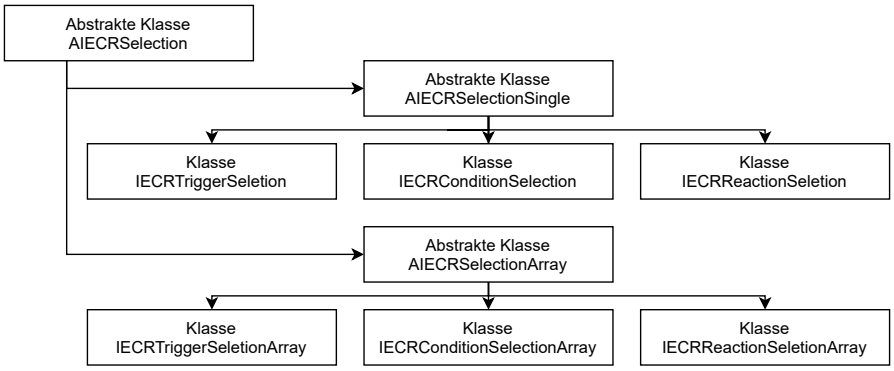


Abbildung 3.14: Vererbung bei den IECRn

Schnittstellen nach außen. Über diese können sie zu größeren Konstrukten zusammengebaut werden.

Alle drei Kategorien sind auf der selben abstrakten Klasse (AIECRSelection) implementiert, siehe Abbildung 3.14. So verfügen sie über Methoden zur Prüfung der Gültigkeit und PropertyDrawer, um die Bedienung über den Inspektor zu erleichtern.

Event

Für die Selektion des auslösenden Events kann ein GameObjekt verlinkt werden, von dem dann eine public Methode die UnityActions akzeptiert oder ein verfügbares UnityEvent ausgewählt werden kann. Dafür werden die verfügbaren Elemente herausgesucht und in einem Menü angeboten, siehe Abbildung 3.15. Dabei ist es möglich, nach mit dem IECRTrigger Attribut ausgewiesenen Elementen zu suchen oder sich alle verfügbaren

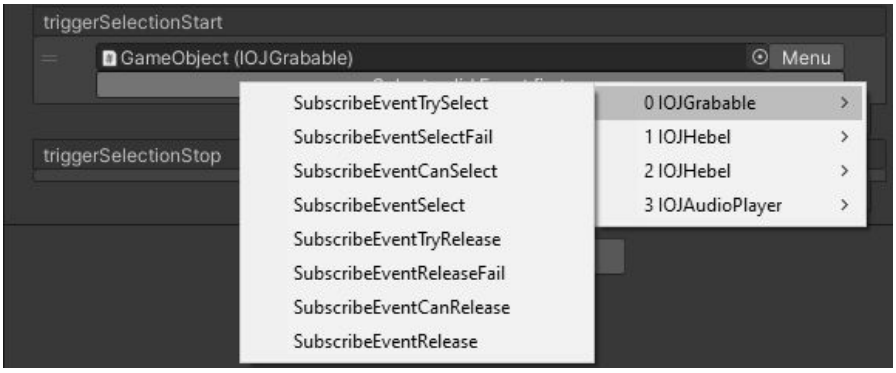


Abbildung 3.15: Menü zum Auswählen eines neuen Trigges

anzeigen zu lassen. Zur Kapselung des Abonnierens wird eine Methode bereitgestellt, die eine UnityAction akzeptiert.

Control

Control erlaubt es, Methoden und Felder mit Rückgabewerten zu wählen und diese auszuwerten. Dazu gibt es zwei Möglichkeiten. Es können entweder alle Methoden und Felder, deren Rückgabewerte unterstützt sind, gezeigt werden oder nur jene, die mit dem IECRCondition Attribut ausgewiesen sind. Zusätzlich kann abhängig vom Rückgabewert eine Bedingung spezifiziert werden. Zur Kapselung der Evaluation gibt es die Methode Evaluate, die die Bedingung auswertet und einen Wahrheitswert zurückgibt.

Reaction

Die Reaction erlaubt es, Methoden auszuwählen, die aufgerufen werden sollen. Hier gibt die Möglichkeit, nach allen oder nach als IECRReaction ausgewiesenen Methoden zu suchen. Gezeigt werden dann alle, deren Parameter unterstützt sind. Für die unterstützten Typen können dann die Parameter für den Aufruf spezifiziert werden. Hier wird eine Methode InvokeReaction zum Auslösen der gewählten Methoden bereitgestellt.

Standard IECR

Aus den vorgestellten Typen lassen sich eine Reihe vorgefertigter Möglichkeiten zusammenbauen. Zur Vielseitigkeit dieser Typen gehört es, dass ein IECR mehrere Felder einer der Kategorien unterstützen kann. Dieses kann über ein Array geschehen oder die dafür vorgesehenen Klassen, siehe Abbildung 3.14. Sie verfügen über einen Property drawer zur besseren Ansicht und ummanteln die Methoden der eingebundenen Klasse. Zum Bilden von komplexem Verhalten ist es möglich, die vorgefertigten IECR auch untereinander zu verbinden. Die Standardversionen besitzen dazu ausgewiesene Methoden.

Neue IECR

Mithilfe der drei Typen können fortgeschrittene Nutzer eigene spezialisierte IECR zusammenstellen. Ermöglicht wird dies durch die vorhandene Abstraktion der einzelnen Komponenten. Für eine spezifische Lösung können Arrays für die Start- und Stopp-Trigger deklariert werden sowie die Methoden, die von diesen aufgerufen werden. Mit der Konditionskomponente können an dieser Stelle einfach weitere Bedingungen geprüft werden, bevor eventuell die Reaktionskomponente ausgelöst wird. Diese Komponenten können dabei auch in anderen Zusammenhängen genutzt werden, z.B um die Berechnung und Aktualisierung von Statuswerten auszulösen oder Objekte ein und aus zu schalten.

3.5.2 Story

Stories dienen dazu, auf höchster Ebene den Zustand von Abläufen in einem Szenario zu beschreiben. Zur Unterstützung bei der Implementation von Stories soll eine Vorlage mit hilfreichen Features geschaffen werden. Mit den Events der Interaktionen können sie übergreifende Interaktions-Ketten und Feedback erzeugen. Auch sind sie der Ort für Kommunikation nach außen, die nicht mit IECRn abgebildet werden kann.

Die erstellte Vorlage bringt und fördert dabei folgende Punkte. Stories haben einen allgemeinen übergreifenden Zustand: Nicht bereit, Kann starten, Läuft und Abgeschlossen. Wenn Stories zu groß sind, können sie so zur Übersicht aufgespalten werden. Hierzu verfügen die Stories über die Methoden für das Zurücksetzen, Starten, Beenden oder Pausieren. Jede dieser vier Methoden verfügt auch über ein UnityEvent, das abonniert werden kann, um eigene Funktionen einzuklinken. So kann beim Beenden des einen Abschnitts der nächste gestartet werden. Zur besseren Übersicht für das Abonnieren von

Events gibt es die zu überschreibende Methode Subscribe() welche in der Basisklasse in Start() aufgerufen wird.

Stories beschreiben oft eine Vielzahl von unterschiedlichen Abschnitten und Zuständen, die in ihrer Masse unübersichtlich werden. Dieselben Zustände, die die Story beschreiben, können genutzt werden, um dieses Problem zu vermeiden. Mit ihnen können dazu im Inspektor sichtbar die einzelnen Abschnitte der Story beschrieben werden, um die Übersicht zu verbessern. Dieses erleichtert besonders bei bedingten Übergängen das Finden von Fehlern und die Lesbarkeit.

Größere Stories besitzen auch viele Referenzen, bei denen geprüft werden muss, ob diese gesetzt sind. Die so entstehenden Fehler können zum Teil erst sehr spät oder selten auftauchen, was sie sehr störend macht. Die einzelne Überprüfung durch OnValidate, siehe 3.2.1, ist dabei nicht praktikabel. An dieser Stelle können die in dem Kapitel: Fehlende Referenzen, siehe 3.2.1, beschriebenen Maßnahmen genutzt werden, um diese automatisch zu überprüfen und Nullpointer Exeptions zu vermeiden. Zusätzlich können in OnValidate Überprüfungen für die Zustände der Interaktionen hinterlegt werden oder wenn es in das Konzept des Szenarios passt, diese in der Methode für Reset oder Start hergestellt werden.

3.6 Anbindung externer Komponenten

Die Art der Anbindung externer Komponenten hängt davon ab, womit diese verknüpft werden sollen. Input des Nutzers, der Interaktionen beeinflussen soll, fällt in den Bereich des Controller Moduls, siehe 3.3, das als Schnittstelle dient. Dort muss der Input aufgearbeitet werden. Anderer Input, der Interaktionen direkt beeinflussen soll, muss anderweitig abstrahiert werden. Fällt dieser in den Rahmen des Fortschrittes, kann dieses auch praktisch in den Stories, siehe 3.5.2, geschehen. Für andere Arten von Input, der nicht in direkte Berührung mit dem Framework tritt, sind keine Vorlagen vorhanden und es ist notwendig, neue Adapter zu erstellen. Feedback kann über die IECR angebunden werden, wenn die entfernten Komponenten über unterstützte Events und Methoden verfügen. Alternativ können die IECR erweitert werden oder das Feedback über die Stories abgehandelt werden.

4 Evaluation

Die Evaluation des Frameworks besteht aus drei Bereichen:

Erstens durch die Qualität der Unterstützung bei der Implementierung der Beispielszenarien.

Zweitens durch die funktionelle Bewertung anhand der Anforderungen, die sich aus den Parametern und Beispielszenarien ergeben.

Drittens mit Artefakten aus dem EnBW-Projekt und der Retrospektive durch die Verwendung im EnBW-Projekt.

4.1 Erstellung der Szenarien

Das Ziel des Frameworks ist, die Entwicklung von Szenarien zu unterstützten. Eine Evaluation anhand eines solchen Einsatzes liefert entsprechend Ergebnisse für die Bewertung. Dieses Kapitel beschreibt den Entwicklungsprozess der im Kapitel Technische Anforderungen 2.6 vorgestellten Szenarien 1 2.5.1 und 2 2.5.2 mit Hilfe des Frameworks. Die Erstellung ist in mehrere Abschnitte geteilt, die sich auf unterschiedliche Teile des Frameworks beziehen.

Der Schwerpunkt der folgenden Beschreibung bezieht sich auf den Einsatz des Frameworks. Zusätzlich zu dem Framework werden für die Szenarien Assets benötigt. Dazu zählt das statische Environment, die interaktiven Objekte und Audiodateien für das Feedback. Die visuellen Assetes können dabei mit Programmen zum Modellieren von dreidimensionalen Objekten hergestellt werden, wie Blender oder Maya, um dann dem Projekt hinzugefügt zu werden.

4.1.1 Vorbereitung des Environments

Der erste Schritt ist die Erstellung des Environments. Hierzu werden die statischen und dynamischen Objekte platziert. Nachdem diese an den richtigen Stellen sind, gilt es die Interaktionsobjekte herauszutrennen, damit diese später interaktive gemacht werden können. Dieser Schritt findet in der Hierarchie der Szenen statt und es werden auch nicht benötige Objekte entfernt, z. B. Collider die automatisch generiert wurden. Das Resultat ist, das die Meshes der interaktiven Objekte dabei jeweils unter einem eigenem leerem GameObject liegen, das auch um als Pivot zu dient. Hierbei hilft das Werkzeug HierachyCleanup 4.2.2 die gewünschten Einstellungen zu kontrollieren, da es das Finden von falsch konfigurierten Objekten oder unerwünschten Komponenten unterstützt.

Zur weiteren Vorbereitung gehört auch das Hinzufügen der Controller Scripte zum Abstrahieren des Inputs. Für dieses Szenario wird SteamVR mit den Wands 2.1.1 als Controller verwendet. Dazu wird das Entsprechende Prefab der Controller in die Szene gebracht. In der Hierarchie unter den Controllern werden dann die Skripte für den Standardcontroller ausgerollt 4.3 und für die richtigen Events und Physik-Layer konfiguriert.

4.1.2 Interaktivität herstellen

Zur Herstellung der Interaktivität gilt es auf die vorbereiteten Objekte die Interaktions-Scripte auszurollen und zu konfigurieren 4.4. Dieser Schritt muss für jedes einzigartige Objekt in der Szene ausgeführt werden. Arbeit kann gespart werden, wenn Interaktionsobjekte in ihrer Konfiguration doppelt vorkommen, dann können sie wie bei den Hebeln zur Dachklappensteuerung als Prefab definiert werden, was 50% der Arbeit spart. Es gilt auch die Physik-Layer einzustellen und die Collider zu den Objekten einzurichten. Zusätzlich müssen die Modifikations-Scripte auf den Hebeln zur Dachklappensteuerung und dem Drehknopf hinzugefügt werden, damit diese wieder zur Ausgangsposition zurückkehren.

Für die Dachklappen wird eine neue Modifikation AutoRotate erstellt. Die Anforderung an die Modifikation ist, dass sie von AbstractRotation erbende Objekte automatisch rotieren kann. Dazu soll sie befehle für Öffnen, Schließen und Stopp entgegennehmen können. Sie soll nur aktiv sein, solange noch eine Änderung in der Rotation stattfindet. Die Bewegungsgeschwindigkeit soll sich konfigurieren lassen. Zusätzlich soll sie über Events darüber informieren können, wenn sich der Zustand ändert. Daraus ergibt sich

```
[RequireComponent(typeof(AbstractRotation))]
void InteraktionsModifikationAutoRotate()
{
 public enum RotationGoals { None, Open, Close }
 public RotationGoals rotationGoal;
 public float degreeSpeed;
 private AbstractRotation abstractRotation;

 public void SetOpening()
 {
  rotationGoal = RotationGoals.Open;
  State = InteractionState.Running;
  if(SendEvents) eventNowOpening.Invoke();
 }
 public void SetClosing()...
 public void StopMotion()...

 void Awake() Events initalisieren
 void Start() abstractRotation initalisieren
 void Update()
 {
  Dachklappe bewegen bis Ende,
  dann mit Event stoppen.
 }
}
```

Abbildung 4.1: Der Pseudocode für die Modifikation die die Dachklappen Rotation steuert

dann die Implementation 4.1 mit dem Framework, die dann direkt aus der Spezifikation abgeleitet werden kann. In gleicher Art gilt es die Interaktionen im zweiten Szenario auszurollen.

4.1.3 Audioquellen hinzufügen

Die erste Art von explizit zu konfigurierendem Feedback ist das Audio-Feedback unter Einsatz des IECRAudioPlayer A.1. Hierzu wird unter den Objekten, die Geräusche machen, ein weiteres Objekt mit dem IECRAudioPlayer angelegt. Auf diesem wird dann über die Referenzen und Events eingestellt, wann diese Audioquelle abgespielt werden soll. Für den Hydraulikhebel ist dieses beim Erreichen der konfigurierten Endpositionen

der Fall. Das Audio für die Dachklappen wird über die Events der Modifikationen für die Bewegung gesteuert, um beim Beginn der Bewegung das Audio zu starten und am Ende zu stoppen. In dieser Art wird für alle im Szenario beschriebenen Audioquellen ein IECRAudioPlayer eingerichtet.

In Szenario 2 2.5.2 kann dieser Prozess für die Objekte übernommen werden. Die meisten IECRAudioPlayer abonnieren Events für das erfolgreiche Greifen. Für die Objekte bei denen Feedback für das nicht erfolgreiche Greifen notwendig ist, muss dann abweichend das Event für das nicht erfolgreiche Greifen abonniert werden.

4.1.4 Abläufe programmieren

Nach dem Einrichten der Interaktionen gilt es diese zusammenzufügen. Hierfür wird eine Story Klasse angelegt, die von der vorhandenen Vorlage erbt. Als Erstes werden alle für die Story benötigten Referenzen angelegt. Die automatische Überprüfung weist danach darauf hin, dass diese nicht gesetzt sind, bis dieser Zustand behoben ist.

Für die Möglichen Hooks, die für Zurücksetzen, Starten, Beenden oder Pausieren gibt es keinen bedarf, da die Unteraktionsobjekte so konfiguriert wurden, dass sie bereits in den richtigen Zuständen sind und für dieses Szenario keine besonderen Optionen benötigt werden.

Der nächste Schritt ist entsprechend die Implementierung der eigentlichen Story, die Dachklappen zu steuern. Dieses geschieht, wenn alle Vorbedingungen gegeben sind und einer der Hebel sein äußeres Limit erreicht. Zusätzlich gilt, dass die zweite Dachluke sich erst dann öffnen lässt, wenn die erste nicht mehr geschlossen ist, da diese sich überlappen. Damit diese Bedingungen ausgeführt werden, wird die Methode 4.2 an alle relevante Events in der zu Implementierenden Subscribe Methode übergeben.

Zusätzlich gibt es eine Methode, die an die Dachklappen Übergeben wird, wenn diese Ihren Öffnungszustand ändern. Von dieser aus werden die entsprechenden Zustände für das Environment gesetzt, da die Umgebungsgeräusche abhängig davon sind, ob die Klappen offen sind. In dieser Methode können dann auch die anderen Reaktionen, die außerhalb des Frameworks liegen, untergebracht werden.

Szenario 2 ist bereits in der Beschreibung eine Final State Machine 2.5.2 und lässt sich entsprechend auch implementieren. In der Umsetzung werden alle Übergänge als Metho-

den implementiert und an die entsprechenden Events der Interaktionsobjekte übergeben. Die Zustände werden dann direkt von den Interaktionsobjekten gehalten.

4.1.5 Bewertung des Erstellungsprozesses

Bei der Erstellung der Szenarien waren unterschiedliche Aufgaben zu bewältigen, die durch das Framework unterstützt wurden. Dieses fing an beim Erstellen des Environments mit der Unterstützung des HierachyCleanup Werkzeugs und des Controller Moduls, das bei der Controller-Einrichtung unterstützte. Mithilfe der vorgefertigten Interaktionen und den Erweiterungsmöglichkeiten konnte die Interaktivität der Szenarien einfach hergestellt werden. Insgesamt besteht das Szenario 1 damit aus 13 und das Szenario 2 aus 6 Elementen. Audiofeedback wurde gleichfalls unterstützt, so dass ohne Probleme in beiden Szenarien je 7 Audio IECR verwendet wurden. Die übergreifenden Abläufe konnten durch die Vorlage für die Stories implementiert werden. Damit hat das Framework gut die schnelle Entwicklung der Szenarien ermöglicht und unterstützt.

4.2 Fehler & Kontrolle

Im Rahmen des Projektes ist eine Sammlung von Werkzeugen zur Unterstützung entstanden. Im Folgenden werden diese anhand der vier Kriterien: Nützlichkeit, Komplexität, Erweiterbarkeit und Wiederverwendbarkeit bewertet. Diese Kriterien führen zu Werkzeugen, die langlebig, klein und vielseitig einsetzbar sind.

4.2.1 Error Log

Das Modul beschleunigt die Entwicklung durch eine einheitliche Art Fehler anzuzeigen, Dubletten zu vermeiden und behobene zu entfernen. Die grafische Schnittstelle erlaubt eine einfache Nutzung der Features und dient zur Konfiguration. Beliebige Komplexität wird unterstützt durch die Einbindung von Delegates, die es von anderen Modulen entkoppeln und eine einfache Wiederverwendung zulassen.

Dieses hat das Modul in der Verwendung zur Erstellung der Szenarien unter Beweis gestellt. Die verbesserte Übersicht über aktuelle Fehler hat die Arbeit beschleunigt 3.4.

```
void UpdateDachklappenBewegung()
{
 if (StoryIsActive && HebelHydraulik.Open &&
     KnopfHydraulik.IsActivated && DrehschalterHydraulik.Open)
 {
  if (HydraulicHebelHydraulicSeite.Open)
   AutoRotateDachklappeHydraulicSeite.Oeffene();
  else if (HydraulicHebelHydraulicSeite.Closed)
   AutoRotateDachklappeHydraulicSeite.Schliessen();
  else
   AutoRotateDachklappeHydraulicSeite.Stop();
  end

  if (HydraulicHebelA3Seite.Open &&
      (!DachklappeHydraulicSeite.Closed ||
      !DachklappeA3Seite.Closed))
   AutoRotateDachklappeA3Seite.Oeffene();
  else if (HydraulicHebelA3Seite.Closed)
   AutoRotateDachklappeA3Seite.Schliessen();
  else
   AutoRotateDachklappeA3Seite.Stop();
  end
 }
 else
 {
  AutoRotateDachklappeHydraulicSeite.StopMotion();
  AutoRotateDachklappeA3Seite.StopMotion();
 }
}
```

Abbildung 4.2: Der Pseudocode für die Bewegung der Dachklappen zu steuern

Dazu zählt, dass nicht die Konsole sondern ein eigenes Fenster für die Ausgabe von Konfigurationsfehlern genutzt wurde, was die allgemeine Übersichtlichkeit stark verbessert hat.

Im Zusammenhang mit dem Finden von nicht gesetzten Referenzen war das Error Log ebenso hilfreich. Bei dieser Erweiterung handelt es sich um eine einzelne Methode. Die Komplexität ist somit gering und die Funktion einfach anzupassen. Ein Beispiel ist der erweiterte Einsatz in der OnValidate 3.2.1 Methode der MonoBehaviors. So erhält der Nutzer Rückmeldung über den Stand der Referenzen mit jeder dort gemachten Änderung. Für die Wiederverwendbarkeit in einem anderen Kontext können Anpassungen notwendig werden, da Referenzen auf das Error Log enthalten sind. Sehr hilfreich hat sich dieses bei den Stories 4.6 gezeigt, die aus vielen Referenzen bestehen und wo diese einzeln und explizit zu überprüfen schnell Fehler erzeugt.

4.2.2 Hierarchy Cleanup

Hierarchy Cleanup ist ein Werkzeug, das dabei hilft, in großen Szenen den Überblick zu behalten. Es bietet die Möglichkeit, Objekte über Suchoperatoren zu finden, die nicht im Hierarchie Fenster verfügbar sind. Dazu kann der Nutzer die Suche aus den Suchoperatoren, die dynamisch an das Werkzeug übergeben werden, konfigurieren. So komplementiert es die Suche mithilfe des ErrorLogs nach Fehlern durch eine dynamischere Möglichkeit. Zur Erweiterung der Hierarchy Cleanup Klasse um weitere Parameter sind allerdings rudimentäre Programmierkenntnisse notwendig. In der Entwicklung beschleunigt das Modul die Suche nach komplexen Parametern und Konfigurationsfehlern 3.5.

Im Projekt war dieses nützlich, um Objekte eines bestimmten Typs zu überprüfen, ob diese in der richtigen Physik-Layer liegen oder die Skalierung von Objekten einheitlich war. Durch die Möglichkeit, Parameter zu kombinieren waren komplexe Suchen machbar. Das Erweitern um neue Parameter ist dabei komplexer, da sowohl die Suche als auch wenn nötig das visuelle Element für die Parameterkonfiguration erstellt werden muss. Dieses erlaubt es, das Werkzeug in beliebigen weiteren Projekten zu verwenden.

4.3 Controller Modul

Das Controller Modul ist die Schnittstelle zwischen Input 2.1.1 und Interaktionen. Dieses schafft die Unabhängigkeit von der Hardware und erlaubt eine einheitliche Implementation der Interaktionen. Für beliebige Controller, die dem 6dof Schema folgen, existiert eine Vorlage, die eine einfache Integration erlaubt, wenn zusätzlich ein Event für Selection und Release vorhanden ist. Eine Implementation für SteamVR 2.0 ist bereits im EnBW-Projekt zum Einsatz gekommen und hat sich dort bewährt. Die vorhandenen Parameter ließen dabei eine einfache Konfiguration für Testen und Nutzung zu 3.8. Das Einbinden unterschiedlicher Controller wird durch einen lokalen Offset vereinfacht, so dass Interaktionen präzise selektiert werden können.

Ein weiterer Teil der hohen Präzision ist das gestaffelte Suchen nach Interaktionsobjekten. Besonders bei dicht zusammen liegenden Objekten hat sich dieses von Vorteil erwiesen. So konnten unbeabsichtigte Mehrfach-Selektionen, unabhängig von der Objektgröße, vermieden werden und die Präzision bleibt konstant.

4.3.1 Erweiterung um Controller

Die Erweiterung wird durch die abstrakte Klasse AController erleichtert. Die Vervollständigung dieser wird durch abstrakte Methoden unterstützt, die die Implementation der Kernmethoden erfordern.

Dieses wurde genutzt, um das vorhandene System um eine Knukel Controller 2.1.1 Version zu erweitern. Hier haben sich die modularen Teile des Frameworks von Vorteil gezeigt. Der neue Controller konnte von der StandardController Klasse abgeleitet werden und um die notwendigen Referenzen erweitert werden. In den Methoden für zusätzliches Verhalten werden beim Selektieren die Objekte auf die Verfügbarkeit von Handposen geprüft und die Handposen zur Anwendung an die Referenzen weiter gegeben. In der Methode für zusätzliches Verhalten beim Release wird anschließend über die Referenzen die Handpose wieder zurückgesetzt. Damit ein tragbares Objekt auch den spezifischen Offset der Handpose nutzen kann, wird für diese eine Interaktions Modifikation 3.4.4 benötigt, die den Offset kontinuierlich setzt, während das Objekt gehalten wird. Damit war das Erweitern möglich und das Framework erfüllt auch diese Anforderung. Durch die so entstandene Abstraktion war der Wechsel zwischen den Controller Typen möglich, ohne die Interaktionen zu beeinflussen.

Abbildung 4.3: Geteilte Zustände aller interaktiven Objekte

Auf gleichem Wege ermöglicht dies die Anbindung an andere Hardware Abstraktion Layer, wenn diese 6dof Controller bieten, wie das von Gerwens [9]. Die Erweiterung um Controller, die anderweitigen Input produzieren, wird nicht generalisiert vom Framework abgefangen. Es können aber Controller-Transformation oder Event Action Paare als Lösung genutzt werden.

4.4 Interaktionen

Im Laufe der Entwicklung des Frameworks und EnBW-Projektes sind eine Vielzahl von Einfachen und Komplexen Interaktionen sowie Modifikationen entstanden. Die Menge an unterschiedlichen Interaktionen wurde dabei ohne Probleme durch die Struktur des Frameworks bewältigt. Dieses war da durch möglich, dass jedes Element auf eine spezifische Aufgabe ausgerichtet ist, die mit den geringsten möglichen Abhängigkeiten auskommt. Das Resultat ist eine hohe Spezifität zusammen mit der Kapselung der Komponenten und einem kleinen Footprint.

4.4.1 Spezifität der Unterklassen

Auch bei einer hohen Spezifität der Interaktionen wird für die Orthogonalität eine einheitliche Struktur benötigt. Die Gemeinsamkeiten aller Varianten sind durch das Erben von der selben Grundklasse AOBJWithState 4.3 und durch eine Erweiterung mit einer Typ spezifischen abstrakten Klasse gewährleistet. Die einfache Integration der Interaktionen über einheitliche Schnittstellen ist so möglich. Die ableitenden abstrakten Klassen implementieren dann weitere Verallgemeinerungen. Dieses erfüllt die Anforderung an Spezifität, Kapselung und Vollständigkeit der Interaktionen.

Die Aufteilung in die drei Arten von Interaktionsobjekten ist wichtig für die Spezifität. Einfache Interaktionen sind nur auf sich selbst bezogen und können nur sich selbst beeinflussen. Modifizierte Interaktionen verändern das Verhalten von genau einer Interaktion und verhindern, dass Interaktionen selber erweitert oder abgeändert werden müssen. Da diese auf derselben Ebene wie die Interaktion existieren, bleiben die Interaktionen übersichtlich und müssen nicht anders behandelt werden. So kann für eine Interaktion eine Vielzahl von unterschiedlichen Verhaltensänderungen erstellt werden, die sich einfach wiederverwenden lassen. Komplexe Interaktionen übernehmen den Teil der Interaktionen, die sich aus mehreren Einfachen zusammensetzen. Das Pattern erlaubt es, eine Anzahl Interaktionen zu einer Größeren zusammenzufügen und so komplexes Verhalten darzustellen, ohne die darunterliegenden Interaktionen zu verändern. Zu diesem Zweck enthalten sie weitere Zustände, Übergänge, Bedingungen und Verhalten, um die beteiligten Interaktionen zu organisieren.

4.4.2 Konfigurieren

Für das Konfigurieren interaktiver Objekte wird der Inspektor genutzt, der durch entsprechende Visualisierungen der Werte unterstützt wird. So benötigt das Konfigurieren einen geringen Anteil der Zeit im Projekt. Es ist möglich Interaktionen ausschließlich über die UI zu konfigurieren, so sind sie für alle Nutzergruppen zugänglich.

Der Einfache Interaktions Button zum Beispiel bietet folgende Optionen zur Konfiguration an Abbildung 4.5. Dieses ist ein gutes Beispiel für die Unterstützung durch den Inspektor. Durch die Beschriftung der Felder, die um Tooltips erweitert sind wenn die Maus über diese gehalten wird, werden auch komplexe Zusammenhänge verständlich. Zusätzlich wird das Objekt als Wiremesh 4.4 in beiden Konfigurationen visualisiert.

Ein weiteres Beispiel ist der Drehmomentschlüssel, eine Komplexe Interaktion, die im EnBW-Projekt zum Einsatz kam. Dieser besteht aus zwei Einfachen Interaktionen einem Tragbaren- und Hebel-Objekt, die durch ein Komplexe Interaktion Drehmomentschlüssel vereint werden. Zur Konfiguration dient der Inspektor 4.7. Dieser enthält eine Vielzahl an Werten, die konfiguriert werden können. Einige der Parameter werden durch Schieberegler dargestellt, die keine ungültigen Werte zulassen. Die anderen geben visuell und mithilfe des ErrorLogs darüber Auskunft, wenn eine ungültige Konfiguration vorliegt. Zusätzlich werden Teile der Konfiguration visuell im SceneView dargestellt, um den Zah-

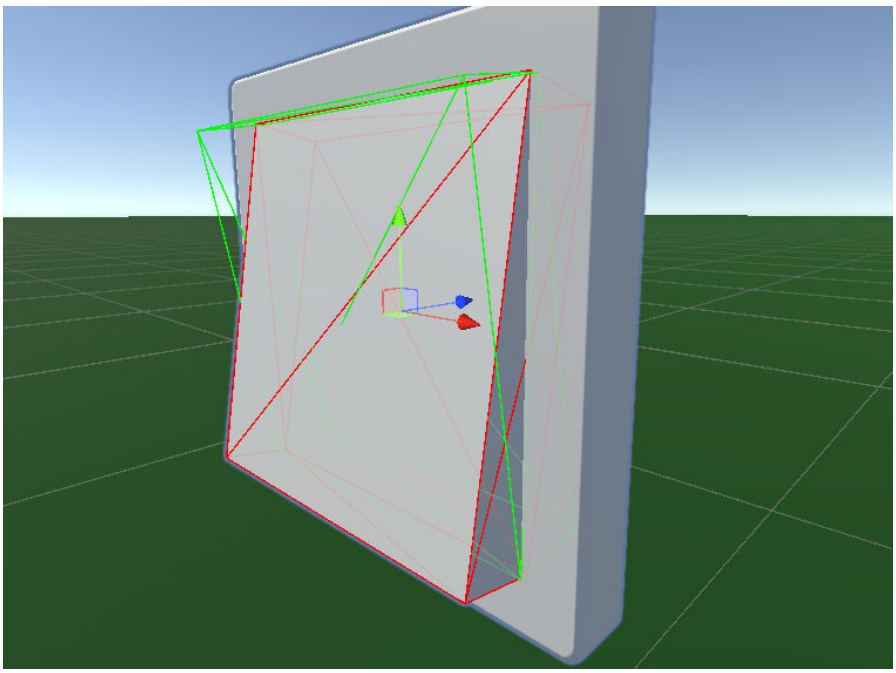


Abbildung 4.4: Die Gizmos des Button

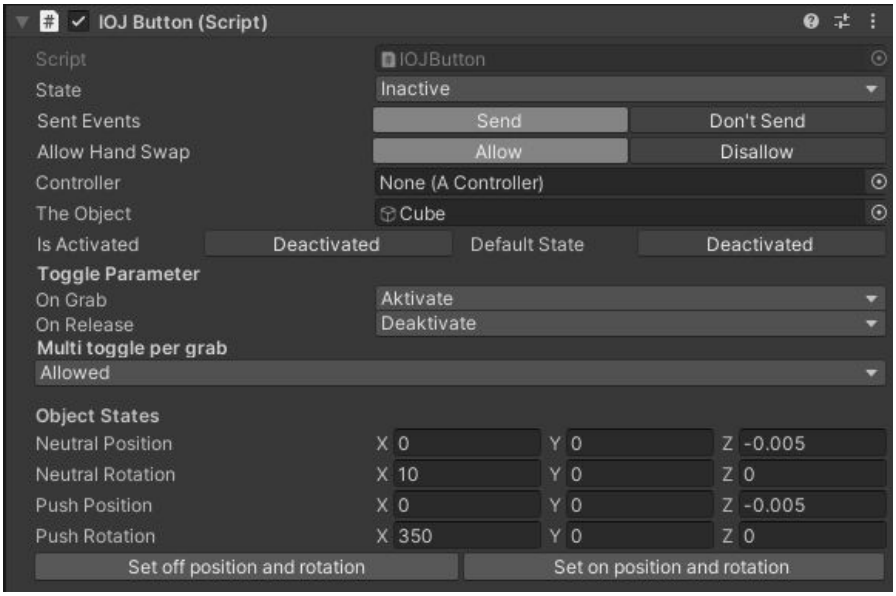


Abbildung 4.5: Der Inspector des Button

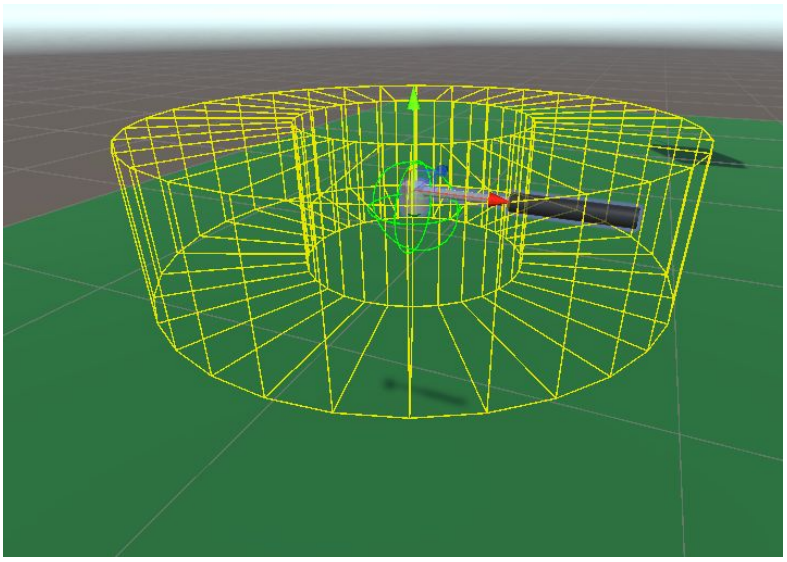


Abbildung 4.6: Die Gizmos des Drehmomentschlüssels. Die gelbe Markierung zeigt den Bereich, den die Hand verlassen muss, damit der Drehmomentschlüssel sich von der Schraube ablöst. Der grüne Bereich markiert, in welchem er sich mit Schrauben verbindet.

len Kontext zu verleihen 4.6. So ließen sich mit Mitteln des Frameworks auch komplexe Konfigurationen problemlos bewältigen.

4.4.3 Neue Erstellen / Erweitern

Das Erweitern und Erstellen von Interaktionen wurde durch die Vorlagen sinnvoll unterstützt. So wurde erzwungen, dass vor Beginn der Implementation einer Interaktion diese spezifiziert wurde um diese in eine der drei Kategorien einzuteilen oder auf diese aufzuteilen. Durch das Erben von einer der Vorlagen wurden große Teile des Codes ausgelagert, was die Erstellung stark beschleunigt hat. Auch öffnet dieses die Arbeit mit Interaktionen auf der Codeebene für die Fortgeschrittenen Nutzer, da so die Schnittellen nach außen klar abgegrenzt und abstrahiert sind. Die erzwungene Spezifität war dann auch beim Testen, Evaluieren und Konfigurieren der Interaktion hilfreich.

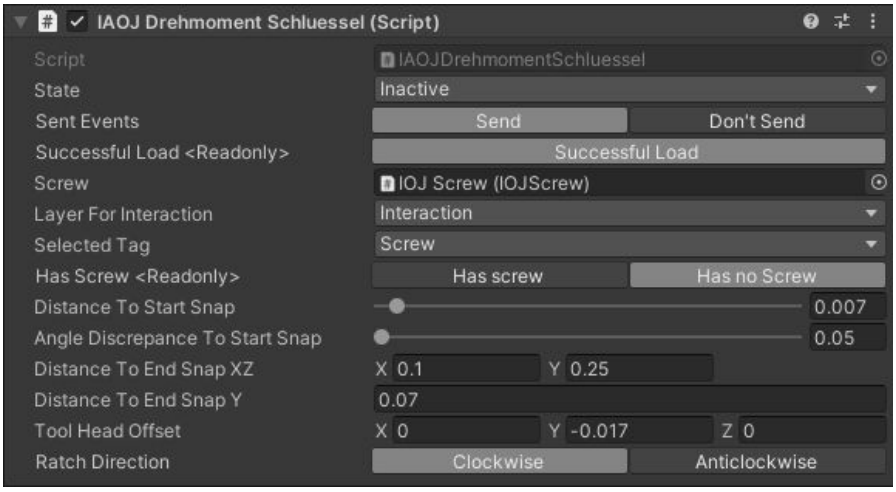


Abbildung 4.7: Der Inspector des Drehmomentschlüssels

4.4.4 Abschließend

Das Modul der Interaktionen erfüllt die aufgestellten Anforderungen. Sie sind spezifisch und haben einen geringen Footprint. Dieses erlaubt ihnen, in sich vollständig zu sein und ermöglicht das einfache Hinzufügen durch das Erben von vorhandenen Vorlagen. Über die Schnittstelle ist auch die Verbindung des Controllers mit den Interaktionen standardisiert. Zusätzlich bieten Interaktionen spezifische Möglichkeiten, direkt manipuliert zu werden. Mithilfe von Events, die bei allen wichtigen Zustandsänderungen ausgelöst werden, können beliebige andere Komponenten auf diese reagieren. Dieses erlaubt eine standardisierte Verknüpfung von Interaktionen.

In der Verwendung im EnBW-Projekt hat sich die Implementierung vorteilhaft gezeigt. Besonders die Anpassungs- und Erweiterungsmöglichkeiten haben sich als wertvoll erwiesen. Neue Interaktionen, die benötigt wurden, konnten einfach hinzugefügt werden. Durch die Möglichkeit, Komplexe Interaktionen oder Modifikationen einzubinden, konnten die originalen Interaktionen erhalten bleiben und es wurden keine Superinteraktionen benötigt. Die Interaktionen und ihre Grundlagen lassen sich so auch in Zukunft für weitere Projekte einsetzen.

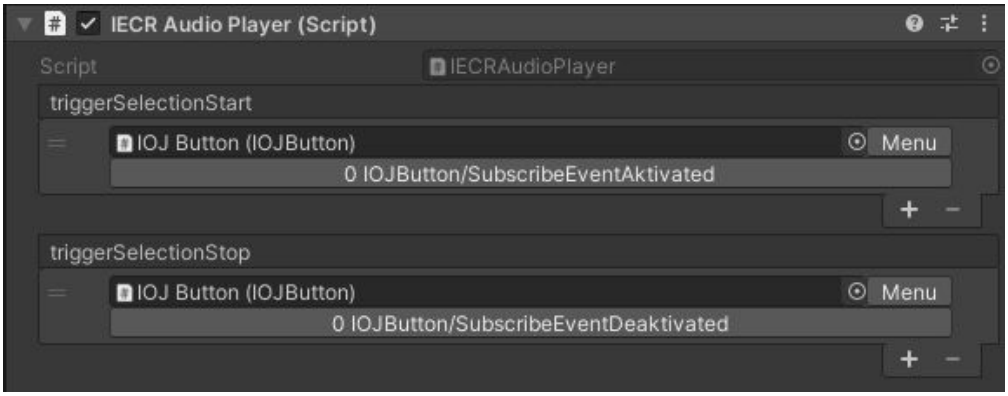


Abbildung 4.8: Der Inspektor des IECRAudioPlayer mit den Eventkategorien Start und Stopp Audio

4.5 IECR (Interaktion Event Control Reaktion)

Die Möglichkeit, Reaktionen ohne Code mit Hilfe des IECR Pattern zu spezifizieren, hat sich an vielen Stellen bewährt. Das generische IECR Pattern erlaubt dabei bereits eine hohe Komplexität des Verhaltens. Der Einsatz von Property Drawern erlaubt eine übersichtliche Konfiguration der Elemente für Nutzer ohne Programmierkenntnisse. Die Modularität der Event-, Control- und Reaktions-Komponenten erweitert dies durch die Möglichkeit, IECR für spezialisierte Aufgaben zu erstellen. Diese lassen sich bereits mit einfachem Verständnis der Programmiersprache und ohne Kenntnisse der darunterliegenden Komponenten zusammenbauen 4.8. So lassen sich diese auch in zukünftigen Projekten in neuen Konstellationen wiederverwenden oder auch in ein Modul für Visuelles Programmieren einbinden 4.7.2.

Dieses ist im EnBW-Projekt an den Audioquellen deutlich geworden 4.1.3, die in den meisten Fällen als Reaktion auf eine Zustandsänderung eines Interaktionsobjektes auslösen. Der Audio IECR ließ sich aus den Komponenten mit minimalem zusätzlichem Code erstellen A.1. Das Ergebnis erlaubt eine einfache Konfiguration vieler individueller Audio Quellen.

In den Szenarien des EnBW-Projekt kam dieses an vielen Stellen zum Einsatz: Wenn Hebel und Drehknöpfe ihre Endposition erreichen, Knöpfe ihren Zustand ändern, Objekte in Dropzones platziert oder von ihnen aufgenommen werden oder Objekte gegriffen werden.

Die Anbindung externer Komponenten ist auch mithilfe der IECR möglich, da die Optionen über die Reflexions API bereitgestellt werden. Dieses gilt für alle drei Komponenten und erlaubt eine einfache Schnittstelle zur Integration der Feedbackmöglichkeiten der Arbeit Gerwens [9].

4.6 Stories - Hilfe beim erstellen von Szenarien

Zusammenhängende Abläufe sind Ziel der Entwicklung von Trainingsszenarien. Diese Funktion wird im Framework von den Stories erfüllt. Die entwickelte Methodik erlaubt es, übergreifendes Verhalten beliebiger Komplexität zu erstellen. Die Interaktionen hierfür müssen selber nicht angepasst werden, da auf deren Schnittstellen implementiert werden kann, was diesen Vorgang auch Programmieranfängern ermöglicht. So können Interaktionen übersichtlich miteinander verbunden werden und komplexes Verhalten geschaffen werden.

Komplexes Verhalten wird durch die Implementierung von Verhalten als Automaten unterstützt. Zur Erweiterung dieses Schemas kann jede Implementation einer Story dabei selber Teil eines Automaten sein. Die automatische Überprüfung für das Füllen der Referenzen-Felder erleichtert dabei die Wartung 3.2.1.

Dieses wurde auch in den beschriebenen Szenarien verwendet. Für Szenario 1 und 2 wird die Erstellung der Stories im Kapitel Abläufe programmieren 4.1.4 beschrieben.

Mit diesen Vorteilen war die Verwendung der Stories ein essenzieller und hilfreicher Teil in den EnBW-Projekten. Die Abstraktion, Bündelung und Unterteilung in Abschnitte haben einen wichtigen Beitrag in der Erstellung und Weiterentwicklung der Szenarien geleistet, die immer wieder nach neuen Informationen angepasst werden mussten. Durch die Bündelung war es des Weiteren auch einfach, Fehlerfeedback in einem der Szenarien zu implementieren, da alle wichtigen Komponenten an einer Stelle versammelt waren. Da diese Komponente mehr Vorlage als Funktion ist, lässt sie sich ohne Probleme auch in zukünftigen Projekten einsetzten, selbst wenn diese eine andere Ausrichtung haben.

4.7 Was ist noch möglich: Vorbereitete Erweiterungen

Für ein solches Framework gibt es immer noch Features, die dieses Erweitern können. Während die folgenden Features den Rahmen der Arbeit gesprengt hätten, sind dennoch Vorbereitungen getroffen, damit Erweiterungen und Zusammenarbeit mit anderen Werkzeugen möglich sind.

4.7.1 Update Manager

In den bisherigen Projekten war die Performance des Frameworks kein Problem. Einige Interaktionen benötigen keine kontinuierlichen Änderungen und verursachen so keine Grundlast. Jene, die aktive Komponenten haben, führen diese nur in entsprechenden Zuständen aus. Sollte ein Projekt eine solche Zahl an Interaktionen verwenden, dass der Overhead in der Ausführung, bis diese abgebrochen wird, zu groß ist, könnte auf einen eigenen Update Manager ausgewichen werden. Dazu müsste die Methode, die das Ändern des Zustandes erlaubt, um das Registrieren und Deregistrieren in einem Update Manager erweitert werden.

4.7.2 Visueller Code

Während das Framework selber kein Modul für visuellen Code enthält, gibt es bereits Werkzeuge dafür. Für Untiy ist zum Beispiel das Werkzeug Bolt vorhanden. In ein solches kann das Framework integriert werden. Zu diesem Zweck können die vorgefertigten IECR Komponenten genutzt werden. Alternativ können als Hilfestellung die Attribute, die von den IECRn zur Markierung von Methoden genutzt werden, Verwendung finden. So existieren bereits Hilfestellungen, um eine zukünftige Umsetzung zu realisieren.

4.8 Fazit

In der Evaluation wurde anhand der Beispielszenarien und der funktionellen Bewertung gezeigt, dass das Framework viele unterschiedliche Elemente zusammenbringt und effektiv eingesetzt werden kann. Dieses Kapitel fasst die Ergebnisse mit Hilfe der Anforderungen zusammen und fügt Erkenntnisse durch die Retrospektive hinzu.

Vielseitige Interaktionen - Dass die Interaktionen vielseitig einsetzbar sind, hat sich in Design und Evaluation der Interaktionskomponente gezeigt. Dies wurde auch durch das EnBW-Projekt bestätigt, wo die Interaktionen vielseitig eingesetzt und dynamisch um neue erweitert wurden.

Szenarien - Das Framework hat sich auch für die Erstellung von Abläufen bewährt. Die Vorlagen für Stories, IECR und Schnittstellen zu den Interaktionen haben dieses ermöglicht, wie die Evaluation gezeigt hat. In gleicher Art waren diese auch im EnBW-Projekt hilfreich.

Skalierbarkeit - Die Skalierbarkeit des Frameworks ist gegeben. In den vollständigen Trainingsszenarien 1 & 2 kamen so ca. 120 Interaktionen zum Einsatz und in Trainingsszenario 3 ca. 230 Interaktionen. Auch bei den Audio-IECRn kamen entsprechend viele zur Verwendung, in 1 & 2 ca. 60 und in 3 waren es 160. Die Skalierung für das übergreifende Verhalten skalierte ebenso gut. Die benötigte Performance war auch ohne die vorbereiteten Optimierungen für einen Updatemanager kein Problem. Sollte die Anzahl der benötigten Interaktionen so groß werden, dass der Overhead der Update Methode zu groß wird, kann die Optimierung vorgenommen werden, worauf die Interaktionen nur noch durch die Anzahl der maximal gleichzeitig aktiven begrenzt wäre.

Erweiterbarkeit - Durch die Designentscheidung, die einzelnen Umsetzungen auf Vorlagen aufzubauen, war es einfach, das System zu erweitern. Da alles auf den Vorlagen aufgebaut ist, verhalten sich neue Komponenten orthogonal zu den Alten und es gibt keine Kompatibilitätsprobleme. Dieses hat zu einer schnellen Umsetzung neuer Szenarien geführt. Die bereits vorhandenen Komponenten für Interaktionen, Controller und IECR haben dies unterstützt. Zugleich führt die Nutzung der Vorlagen auch zu einer besseren Wartbarkeit, da die Features so kleiner bleiben.

Unabhängigkeit - Durch die Entscheidung, den Input für die Interaktionen durch das Controller Modul zu abstrahieren, wurde dieser Punkt erfüllt. So war es möglich, Controller mit unterschiedlichen Features zu verallgemeinern, ohne dass diese ihre speziellen Eigenschaften verloren haben. Nachträglich war es des Weiteren möglich, einen ausgewählten Ausschnitt aus dem dritten Trainingsszenario für eine 2D-Desktop Steuerung umzusetzen, indem eine entsprechende kontextsensitive Umwandlung des Inputs für die enthaltenen Interaktionen vorgenommen wurde.

Einrichtung und Konfiguration - Die angepassten Inspektor-Fenster und Szenen-Gizmos haben zu einer einfachen und sicheren Einrichtung und Konfiguration geführt.

Durch die aufgearbeitete Visualisierung können Komponenten erfolgreich genutzt werden. Dieses war besonders für die drei Trainingsszenarien ein wichtiges Hilfsmittel für die reibungslose Umsetzung.

Schnittstellen - Durch den Aufbau in einer modularen Struktur wurden Schnittstellen für alle Module und Komponenten erzwungen. Dieses war maßgeblich hilfreich, um die Interaktionen zu verknüpfen, sei es für Stories, IECR, Modifikationen oder komplexe Interaktionen.

Modularität - Die Modularität wurde als grundlegende Designentscheidung festgelegt und entsprechend umgesetzt. Die Komponenten wurden so getrennt, dass sie unabhängig voneinander einzusetzen sind, abgesehen von den implementierten Vorlagen, die Schnittstellen umsetzten.

Orthogonalität - Durch die starke Verwendung von Vorlagen sind die daraus entstanden Komponenten sehr ähnlich und es ist einfach, Funktionen auf verwandten Komponenten wiederzufinden. Dieses ist auch für die Interaktionen in der Virtual Reality gegeben. Durch das einheitliche Interface zu den Interaktionen und die einheitliche Programmierung waren diese orthogonal in Virtual Reality zu bedienen.

Nutzergruppen - Die unterschiedlichen Nutzergruppen sind durch die unterschiedlichen Abstraktionslevel der Komponenten gut angebunden. So bieten die IECR und Interaktionen viele Konfigurationsmöglichkeiten, ohne dass eine Zeile Code geschrieben werden muss. Wenn diese Möglichkeiten nicht ausreichen, ist es durch die Vorlagen und Schnittstellen fortgeschrittenen Nutzern möglich, neue zu schaffen. Die Experten Nutzer profitieren auch von den Schnittstellen, da diese das Framework strukturieren und zugänglicher machen. Dieses hatte sich auch in der Entwicklung der Trainingsszenarien 1 bis 3 gezeigt. Dort wurde das Audio-Feedback mithilfe der IECR eingerichtet, ohne dass dazu eine komplizierte Einweisung notwendig war.

Die Evaluation zeigt, dass die gestellten Anforderungen erfüllt wurden und die getroffenen Design Entscheidungen zu einem guten Ergebnis geführt haben. Dasselbe Ergebnis liefern die vorgestellten Artefakte, die im Zusammenhang der Trainingsszenarien entstanden sind, und die eingestreute Retrospektive. Insgesamt ist das entstandene Framework für die Erstellung dieser Art von Szenarien sehr hilfreich und erhöht deren Qualität.

5 Schluss

Dieses Kapitel fast die Arbeit zusammen und gibt einen Ausblick für Weiterentwicklung und Nutzung.

5.1 Zusammenfassung

Die Arbeit beschreibt die Entwicklung eines Frameworks zur Erstellung von interaktiven Virtuell Reality Szenarien. Die notwendige Unterstützung wird dabei durch das Design, die Vorlagen und die vorhandenen Features erreicht.

In der Analyse, siehe Kapitel 2, werden Parameter identifiziert, die für ein solches Framework relevant sind. Es wurde der aktuelle Stand von Virtual Reality aufgeführt, und wie Game Engines den Endnutzern und Entwicklern vieles ermöglichen und erleichtern. Interaktionen nehmen eine zentrale Stellung ein, da sie die virtuellen Welten interaktiv machen und damit erst Probehandeln möglich wird. Dafür wurden Interaktionen analysiert und festgestellt, dass eine direkte Übertragung und in diesem Kontext Nutzung der Handposition gut für die Selektions- und Manipulations-Phase ist. Feedback der Interaktionen wurde dabei in drei Kategorien aufgeteilt: Intrinsisches, Direktes und Entferntes.

Dann wurde die Vielseitigkeit von Training in Virtual Reality vorgestellt und die Vorteile von virtuellem Training aufgezeigt, z. B. einfache Skalierung, bessere Verfügbarkeit, reduzierte Kosten und keine Unfallgefahr. Folgend wurden Parameter für Frameworks, die in diesem Kontext eingesetzt werden, untersucht. Dabei wurden unterschiedliche Nutzergruppen für das Framework identifiziert sowie der fließende Wechsel und die Zusammenarbeit zwischen diesen für wichtig befunden. Danach wurden zwei Ausschnitte aus Trainingsszenarien vorgestellt und auf ihre benötigten Features untersucht. Aus den Features wurden dann die Anforderungen sowohl von der technischen Seite des Frameworks

als auch von der Seite der Nutzer aufgestellt. Wichtig ist dabei, dass Nutzer unterschiedlicher Erfahrungsstufen effizient zusammenarbeiten können. Dafür wurden Eigenschaften wie Skalierbarkeit, Erweiterbarkeit, Unabhängigkeit und Modularität aufgestellt.

Aufbauend auf den Anforderungen wurde in Kapitel 3 das Framework entworfen und implementiert. Dafür wurden als erstes Designentscheidungen für Modularität, lose Kopplungen, minimale Features und Erweiterbarkeit getroffen und auf diesen die entworfene Modul Struktur vorgestellt. Die Module übernehmen einzelne Aufgaben im Framework: Fehler-Bearbeitung/-Vermeidung und Kontrolle, Abstraktion der Controller, Bereitstellen der Interaktionen und Erstellen von Szenarien. Diese Module sind daraufhin in ihrem Design und der Implementation beschrieben worden.

In der Evaluation, siehe Kapitel 4, wurde das Framework anhand von drei Kategorien bewertet. Die erste war die Qualität der Unterstützung bei der Implementierung der Beispielszenarien. Anhand einer Beschreibung, mit welchen Schritten diese erstellt wurden, konnte festgemacht werden, dass das Framework dieses Kriterium erfüllt. Dasselbe wurde für die vollständigen Trainingsszenarien festgestellt. Zweitens hat die funktionelle Bewertung anhand der Anforderungen ergeben, dass diese durch die Eigenschaften und Module des Frameworks erfüllt werden. Das Arbeiten mit Vorlagen, auf denen alles aufgebaut ist, ist dabei maßgeblich entscheidend, da so auch neue Komponenten den Vorhandenen ähneln und kompatibel zu diesen sind. Anhand der eingebrachten Artefakte konnte gezeigt werden, dass auch Erweiterungen möglich sind, während die Retrospektive zeigt, dass das Framework sich auch für größere Einsätze eignet und dort die Entwicklung vereinfachen und beschleunigen kann.

5.2 Ausblick

Das Ergebnis dieser Arbeit bietet eine solide Plattform für Entwicklungen von Virtual Reality Trainingsszenarien. Weiterführend ist es möglich, dieses Framework für weitere Projekte in Unity einzusetzen. Solange das Projekt dabei von 3D Interaktionen profitiertet, kann das Framework für diesen Zweck angepasst werden. In den anderen Fällen können die Werkzuge des Frameworks immer noch einen Mehrwert bei der Erstellung bieten. Mit jedem Projekt kommen dabei neue Herausforderungen, für die es gilt Lösungen zu finden. Durch den modularen Ansatz würde das Framework so mit jedem Einsatz mitwachsen, wenn neue Interaktionen und Features hinzukommen. Das Framework ist dabei

in einem Zustand, der es anderen Entwicklern nach einer Einweisung ermöglicht, dieses selbst zu benutzten, sodass es auch in anderen Projekten verwendet werden kann.

Darüber hinaus gibt es einige allgemeine Erweiterungen, die ins Framework integriert werden könnten. Hierzu gehört die erwähnte Anbindung an ein Werkzeug für visuellen Code, um das Framework noch mehr für Nutzer, die des Programmierens nicht mächtig sind, zu öffnen. Auch der Update Manager zu Reduzierung der benötigten Performance ist eine sinnvolle Erweiterung, die vorgenommen werden kann. Zu diesem Punkt zählt auch eine Erweiterung des 2D zu 3D Adapters des Controller Moduls, das im Moment nur eine kleine Auswahl von Interaktionen unterstützt.

Auch wenn Interaktion für zukünftige Projekte noch Spekulationen sind, ist es bereits sinnvoll, die Werkzeugkollektion um Features zu erweitern. Denn Features, die bei der Einrichtung und Erstellung helfen, können bereits jetzt allgemeingültig implementiert werden. Dieses können die IECR sein, die um komplexeres Verhalten erweitert werden, wobei es als Herausforderung darauf zu achten gilt, dass diese in ein existierendes Tool wie Bolt integriert oder mit diesem verbunden werden können. Darüber hinaus könnten es weitere Tools für die Visualisierung von Interaktionen sein oder Hilfsmittel für die übersichtlichere Gestaltung von neuen und alten Interaktionen sein.

Jede Erweiterung um ein Werkzeug, das in Zusammenhang mit den bereits existieren Features zusammenarbeitet, schafft dabei eine Erweiterung, die in zukünftigen Projekten hilfreich sein kann. Wenn diese auch noch Nutzer unterschiedlicher Erfahrungsstufen unterstützen, vergrößern sie den Wert des Frameworks umso mehr.

Literaturverzeichnis

[1] Argelaguet Sanz, Ferran ; Andujar, Carlos: A Survey of 3D Object Selection Techniques for Virtual Environments. In: *Computers and Graphics* 37 (2013), Mai, Nr. 3, S. 121–136. – URL https://hal.archives-ouvertes.fr/hal-00907787

[2] Becker, Jonathan ; Gerwens, Niklas: *Evaluation der Toolchain für 3D-Objektmanipulation in VR-Trainingsumgebungen.*. – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2021-proj/beckerHP.pdf. – Zugriffsdatum: 2021-12-03

[3] Becker, Jonathan ; Gerwens, Niklas: *Toolchain für 3D-Objektmanipulation in VR-Trainingsumgebungen.* – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2021-proj/beckerGP.pdf. – Zugriffsdatum: 2021-12-03

[4] Bowman, Doug A. ; Johnson, Donald B. ; Hodges, Larry F.: Testbed Evaluation of Virtual Environment Interaction Techniques. In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*. New York, NY, USA : Association for Computing Machinery, 1999 (VRST '99), S. 26–33. – ISBN 1581131410

[5] Bowman, Doug A. ; McMahan, Ryan P. ; Ragan, Eric D.: Questioning Naturalism in 3D User Interfaces. In: *Commun. ACM* 55 (2012), September, Nr. 9, S. 78–88. – ISSN 0001-0782

[6] Cheng, Alan ; Yang, Lei ; Andersen, Erik: Teaching Language and Culture with a Virtual Reality Game. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA : Association for Computing Machinery, 2017 (CHI '17), S. 541–549. – ISBN 9781450346559

[7] CSTI: *VR Wind Turbine Training.* – URL https://www.youtube.com/watch?v=srsESt4AQU4. – Zugriffsdatum: 2022-01-20

[8] Dörner, R. ; Broll, W. ; Grimm, P. ; Jung, B.: *Virtual und Augmented Reality (VR/AR).* Berlin : Springer Vieweg, 2013 (eXamen.press). – ISBN 978-3-642-28902-6

[9] Gerwens, Niklas: *Eine Hardware-Abstraktionsschicht für Interaktionen in VR-Applikationen.* – URL `https://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/gerwens.pdf`

[10] Gillies, Marco: What is Movement Interaction in Virtual Reality for? In: *Proceedings of the 3rd International Symposium on Movement and Computing.* New York, NY, USA : ACM, 2016 (MOCO '16), S. 31:1–31:4. – ISBN 978-1-4503-4307-7

[11] Green, Mark: Towards Virtual Environment Authoring Tools for Content Developers. In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology.* New York, NY, USA : Association for Computing Machinery (VRST '03), S. 117–123. – URL `https://doi.org/10.1145/1008653.1008675`. – ISBN 1581135696

[12] Hafsia, Mehdi ; Monacelli, Eric ; Martin, Hugo: Virtual Reality Simulator for Construction Workers. In: *Proceedings of the Virtual Reality International Conference - Laval Virtual.* New York, NY, USA : ACM, 2018 (VRIC '18), S. 11:1–11:7. – ISBN 978-1-4503-5381-6

[13] Heinecke, Andreas: *Mensch-Computer-Interaktion : Basiswissen für Entwickler und Gestalter.* Springer, 2012. – ISBN 9783642135064

[14] Hendricks, Zayd ; Marsden, Gary ; Blake, Edwin: A Meta-Authoring Tool for Specifying Interactions in Virtual Reality Environments. In: *Proceedings of the 2nd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa.* New York, NY, USA : Association for Computing Machinery, 2003 (AFRIGRAPH '03), S. 171–180. – ISBN 1581136439

[15] Hess, Nicolai ; Wischweh, Jan D. S. ; Albrecht, Kirsten ; Blom, Kristopher J. ; Beckhaus, Steffi: ACTIF: An Interactor Centric Interaction Framework. In: *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology.* New York, NY, USA : Association for Computing Machinery (VRST '08), S. 39–42. – URL `https://doi.org/10.1145/1450579.1450587`. – ISBN 9781595939517

[16] Koutitas, George ; Smith, Kenneth S. ; Lawrence, Grayson ; Metsis, Vangelis ; Stamper, Clayton ; Trahan, Mark ; Lehr, Ted: A Virtual and Augmented Reality

Platform for the Training of First Responders of the Ambulance Bus. In: *Proceedings of the 12th ACM International Conference on PErvasive Technologies Related to Assistive Environments*. New York, NY, USA : ACM, 2019 (PETRA '19), S. 299–302. – ISBN 978-1-4503-6232-0

[17] M, Slater: Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments. In: *Philosophical transactions of the RoyalSociety of London Series B, Biological sciences 364* (2009)

[18] Mendes, D. ; Caputo, F. M. ; Giachetti, A. ; Ferreira, A. ; Jorge, J.: A Survey on 3D Virtual Object Manipulation: From the Desktop to Immersive Virtual Environments. In: *Computer Graphics Forum* 38 (2019), Nr. 1, S. 21–45

[19] Millais, Patrick ; Jones, Simon L. ; Kelly, Ryan: Exploring Data in Virtual Reality: Comparisons with 2D Data Visualizations. In: *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2018 (CHI EA '18), S. LBW007:1–LBW007:6. – ISBN 978-1-4503-5621-3

[20] Noblecourt, Sylvain ; Bourgoin, Geoffrey ; Havard, Vincent ; Baudry, David: Evaluating the Influence of Interaction Technology on Procedural Learning Using Virtual Reality. In: *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*. New York, NY, USA : Association for Computing Machinery (VRST '21). – URL `https://doi.org/10.1145/3489849.3489918`. – ISBN 9781450390927

[21] Pellett, Kerry ; Zaidi, Syed Fawad M.: A Framework for Virtual Reality Training to Improve Public Speaking. In: *25th ACM Symposium on Virtual Reality Software and Technology*. New York, NY, USA : Association for Computing Machinery, 2019 (VRST '19). – ISBN 9781450370011

[22] Petkov, Emiliyan ; Angelov, Vladislav: *Virtual Reality Training System for Specialists Who Operate on High-Voltage Switchgears in an Oil Plant in Russia*. S. 266–269. In: *Proceedings of the 21st International Conference on Computer Systems and Technologies '20*, Association for Computing Machinery, 2020. – URL `https://doi.org/10.1145/3407982.3408003`. – ISBN 9781450377683

[23] Pirker, Johanna ; Dengel, Andreas ; Holly, Michael ; Safikhani, Saeed: Virtual Reality in Computer Science Education: A Systematic Review. In: *26th ACM*

Symposium on Virtual Reality Software and Technology. New York, NY, USA : Association for Computing Machinery (VRST '20). – URL https://doi.org/10.1145/3385956.3418947. – ISBN 9781450376198

[24] Rogers, Katja ; Funke, Jana ; Frommel, Julian ; Stamm, Sven ; Weber, Michael: Exploring Interaction Fidelity in Virtual Reality: Object Manipulation and Whole-Body Movements. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2019 (CHI '19), S. 414:1–414:14. – ISBN 978-1-4503-5970-2

[25] Rossol, Nathaniel ; Cheng, Irene ; Bischof, Walter F. ; Basu, Anup: A Framework for Adaptive Training and Games in Virtual Reality Rehabilitation Environments. In: *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*. New York, NY, USA : Association for Computing Machinery, 2011 (VRCAI '11), S. 343–346. – ISBN 9781450310604

[26] Ruddle, Roy A. ; Lessels, Simon: For Efficient Navigational Search, Humans Require Full Physical Movement, but Not a Rich Visual Scene. In: *Psychological Science* 17 (2006), Nr. 6, S. 460–465. – PMID: 16771793

[27] Schwarz, Stephanie ; Regal, Georg ; Kempf, Marina ; Schatz, Raimund: *Learning Success in Immersive Virtual Reality Training Environments: Practical Evidence from Automotive Assembly*. In: *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society*, Association for Computing Machinery, 2020. – URL https://doi.org/10.1145/3419249.3420182. – ISBN 9781450375795

[28] Seo, Jinsil H. ; Smith, Brian M. ; Cook, Margaret E. ; Malone, Erica R. ; Pine, Michelle ; Leal, Steven ; Bai, Zhikun ; Suh, Jinkyo: Anatomy Builder VR: Embodied VR Anatomy Learning Program to Promote Constructionist Learning. In: *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA : Association for Computing Machinery (CHI EA '17), S. 2070–2075. – URL https://doi.org/10.1145/3027063.3053148. – ISBN 9781450346566

[29] Sokolowski, Jacek ; Walczak, Krzysztof: A Contextual Semantic Interaction Interface for Virtual Reality Environments. In: *Proceedings of the Virtual Reality International Conference - Laval Virtual*. New York, NY, USA : Association

for Computing Machinery (VRIC '18). – URL https://doi.org/10.1145/3234253.3234300. – ISBN 9781450353816

[30] Sutherland, Ivan E.: The Ultimate Display. In: *Proceedings of the IFIP Congress*, 1965, S. 506–508

[31] Tsai, Wan-Lun ; Chung, Ming-Fen ; Pan, Tse-Yu ; Hu, Min-Chun: Train in Virtual Court: Basketball Tactic Training via Virtual Reality. In: *Proceedings of the 2017 ACM Workshop on Multimedia-Based Educational and Knowledge Technologies for Personalized and Social Online Training*. New York, NY, USA : Association for Computing Machinery (MultiEdTech '17), S. 3–10. – URL https://doi.org/10.1145/3132390.3132394. – ISBN 9781450355087

[32] Unity: *Unity Execution Order*. Eingesehen 2017-12-14. – URL https://docs.unity3d.com/560/Documentation/Manual/ExecutionOrder.html. – Zugriffsdatum: 2017-12-14

[33] Unity: *Unity OnValidate*. Eingesehen 2017-12-14. – URL https://docs.unity3d.com/2019.4/Documentation/ScriptReference/MonoBehaviour.OnValidate.html. – Zugriffsdatum: 2022-01-12

[34] Unity: *Unity Platform Dependent Compilation*. Eingesehen 2017-12-14. – URL https://docs.unity3d.com/2019.4/Documentation/Manual/PlatformDependentCompilation.html. – Zugriffsdatum: 2022-01-12

[35] Wang, Xin ; Wang, Xiuyue: Virtual Reality Training System for Surgical Anatomy. In: *Proceedings of the 2018 International Conference on Artificial Intelligence and Virtual Reality*. New York, NY, USA : Association for Computing Machinery, 2018 (AIVR 2018), S. 30–34. – ISBN 9781450366410

[36] Weise, Matthias ; Zender, Raphael ; Lucke, Ulrike: A Comprehensive Classification of 3D Selection and Manipulation Techniques. In: *Proceedings of Mensch Und Computer 2019*. New York, NY, USA : Association for Computing Machinery, 2019 (MuC'19), S. 321–332. – ISBN 9781450371988

[37] Wyk, Etienne van ; Villiers, Ruth de: Virtual Reality Training Applications for the Mining Industry. In: *Proceedings of the 6th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*. New York, NY, USA : Association for Computing Machinery, 2009 (AFRIGRAPH '09), S. 53–63. – ISBN 9781605584287

[38] Zhang, Lei ; Oney, Steve: FlowMatic: An Immersive Authoring Tool for Creating Interactive Scenes in Virtual Reality. In: *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA : Association for Computing Machinery (UIST '20), S. 342–353. – URL `https://doi.org/10.1145/3379337.3415824`. – ISBN 9781450375146

A Anhang

```
[RequireComponent(typeof(EinfacheInteraktionAudioPlayer))]
class IECRAudioPlayer( )
{
  IECRTriggerSelectionArray triggerSelectionArrayStart;
  IECRTriggerSelectionArray triggerSelectionArrayStop;
  EinfacheInteraktionAudioPlayer reciverSelection;

  void Start()
  {
    triggerSelectionArrayStart.Subscribe(IECRStartSound);
    triggerSelectionArrayStop.Subscribe(IECRStopSound);
  }

  void IECRStartSound() => reciverSelection.Start();
  void IECRStopSound() => reciverSelection.Stop();
}
```

Abbildung A.1: Der Pseudocode für den Audio IECR

```
protected bool TryCanNotSelect(AController controller)
{

 ControllerTryingAction = controller;

 if(SendEvents) InvokeEventTrySelect();

 if (State == InteraktionsState.Grabbed && !allowHandSwap)
 {
  if (SendEvents) InvokeEventSelectFail();
  return true;
 }

 if (IsNotReadyForInteraction())
 {
  if (SendEvents) InvokeEventSelectFail();
  return true;
 }

 Controller = controller;

 if (SendEvents) InvokeEventCanSelect();

 if (IsNotReadyForInteraction())
 {
  if (SendEvents) InvokeEventSelectFail();
  return true;
 }

 if (SendEvents) InvokeEventSelect();

 return false;
}
```

Abbildung A.2: Der Code der genutzt wird um festzustellen ob ein Interaktions Objekt nicht gegriffen werden kann

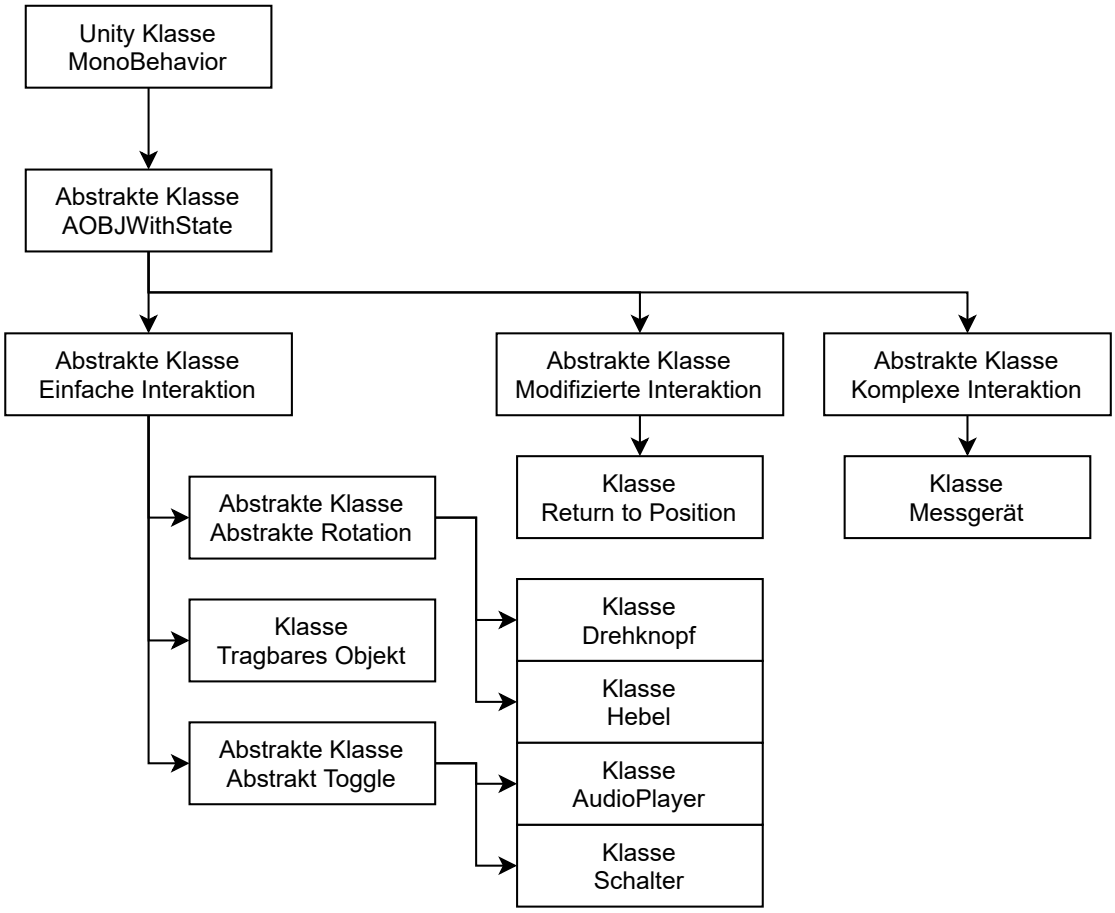


Abbildung A.3: Vererbung der unterschiedlichen Interaktionstypen

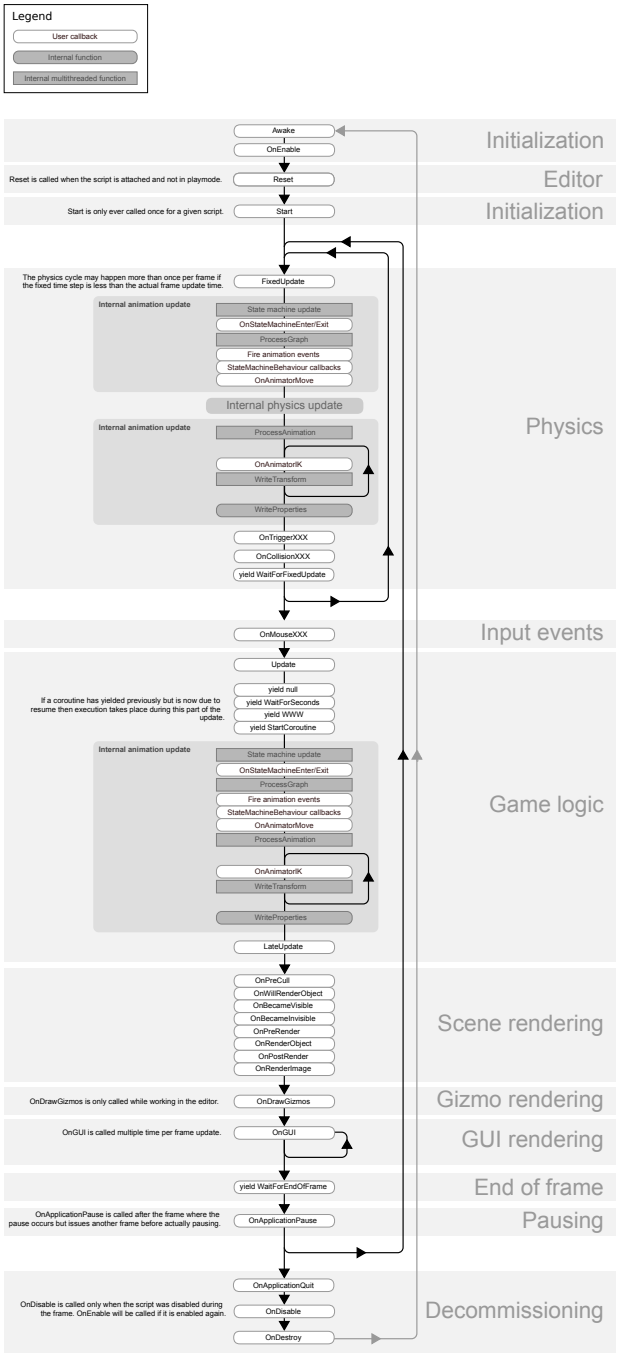


Abbildung A.4: Die Unity Execution Order der Gameloop [32]

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original