

## Diplomarbeit

# Medizinisches VR Training

VR Simulation eines Tracheostomawechsels

Logik und Interaktivität einer VR Schulungs-App

Mihael Brsanovic                    5BHITM

Bewegung und Interaktion mit der virtuellen Realität

Alan Matykiewicz                    5BHIT

Graphical User Interface

Felix Hadinger                        5BHITM

3D Modellierung von Patientenzimmer und Props

Luca Bauer                            5BHITM

**Betreuer:** Mag. Dr. Jiresch Eugen

Ausgeführt im Schuljahr 2021/22

---

Abgabevermerk:

5. April 2022

Übernommen von:



# **Eidesstattliche Erklärung**

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

---

Ort, Datum

Mihael Brsanovic

---

Ort, Datum

Alan Matykiewicz

---

Ort, Datum

Felix Hadinger

---

Ort, Datum

Luca Bauer



# Kurzfassung

Der Auftraggeber KWP erhält regelmäßig neues Pflegepersonal, welches für notwendige medizinische Prozesse wie z.B. einen Tracheostomawechsel ausgebildet werden muss. Diese können nicht oder nur schwer in der Praxis geübt werden, da dies Risiken für Pflegebedürftige birgt. Daher wird nach einer flexiblen, wiederholbaren Trainingsumgebung in Virtual Reality (VR) gesucht. Das Projekt Medizinisches VR Training nimmt sich vor, eine dementsprechende Lösung für medizinisches Fachpersonal zu erstellen. Dafür entwickelt wir eine VR-Applikation für die Oculus Quest 2, mit welcher man den Tracheostoma-wechsel wo und wann man will durchführen kann.



# **Abstract**

Our client KWP regularly receives new nursing staff who need to be trained for necessary medical processes such as a tracheostomy change. The training for these processes cannot be practiced in a real environment, as this involves risks for patients. Therefore, a flexible, repeatable virtual reality training environment in is sought. The “Medizinisches VR Training” project takes on the task of developing a corresponding solution for medical professionals. For this purpose, we are developing a VR application for the Oculus Quest 2, with which one can perform the tracheostomy change everywhere anytime.



# Inhaltsverzeichnis

<b>1</b>	<b>Gendererklärung</b>	<b>13</b>
<b>2</b>	<b>Danksagung</b>	<b>15</b>
<b>3</b>	<b>Einleitung</b>	<b>17</b>
3.1	Ausgangslage . . . . .	17
3.2	Nutzen . . . . .	17
3.3	Grenzen . . . . .	18
3.4	Planung und Umsetzung . . . . .	18
<b>4</b>	<b>Studie</b>	<b>19</b>
4.1	Logik und Interaktivität einer Schulungs- und VR-App . . . . .	19
4.1.1	VR-Schulungs-Applikationen . . . . .	19
4.1.2	Wie wird VR heute eingesetzt? . . . . .	20
4.1.3	Entwurfsmuster . . . . .	22
4.1.4	Individuelle Aufgabenstellung . . . . .	23
4.2	Bewegung und Interaktion mit der VR Welt . . . . .	24
4.2.1	Entwicklung von VR-Applikationen . . . . .	24
4.2.2	Bewegung in VR . . . . .	26
4.2.3	Interaktion mit Objekten in VR . . . . .	31
4.3	Graphical User Interface (GUI) . . . . .	33
4.3.1	Was ist eigentlich ein GUI? . . . . .	33
4.3.2	Head-up Display und World Space . . . . .	34
4.3.3	Richtlinien und Empfehlungen . . . . .	35
4.3.4	Individuelle Aufgabenstellung . . . . .	37
4.4	3D Modellierung . . . . .	39
4.4.1	Entwerfen digitaler Modelle . . . . .	39
4.4.2	3D-Modellierungssoftware und Modellierungstechniken . . . . .	41
4.4.3	Texturierung . . . . .	49
4.4.4	Verhalten: Rigging, Animation und Physik . . . . .	50
4.4.5	Individuelle Aufgabenstellung . . . . .	51
<b>5</b>	<b>Konzept</b>	<b>53</b>
5.1	Konzept: Logik und Interaktivität einer Schulungs- und VR-App . . . . .	53
5.1.1	Ausgangslage . . . . .	53
5.1.2	Programmsteuerung . . . . .	53
5.1.3	Tasks . . . . .	56

5.1.4	Erweiterbare Entwicklung . . . . .	59
5.2	Konzept für die Fortbewegung und Interaktion mit der virtuellen Welt . . . . .	61
5.2.1	Entwicklungsumgebung . . . . .	61
5.2.2	Fortbewegung . . . . .	62
5.2.3	Interaktion mit Objekten . . . . .	64
5.3	Konzept eines Graphical User Interfaces . . . . .	66
5.3.1	Statusanzeige und weitere Darstellungen von Informationen . . . . .	66
5.3.2	Hauptmenü und Loading Screen . . . . .	67
5.3.3	Implementierungen von Guidelines . . . . .	69
5.4	Konzept für die Props und die Umgebung . . . . .	71
5.4.1	Modellierungstechnik . . . . .	71
5.4.2	Zimmer . . . . .	71
5.4.3	Props . . . . .	71
5.4.4	Hände . . . . .	78
<b>6</b>	<b>Implementierung</b>	<b>81</b>
6.1	Logik und Interaktivität einer Schulungs-App . . . . .	81
6.1.1	Programmsteuerung . . . . .	81
6.1.2	Tasks . . . . .	85
6.1.3	Daten . . . . .	95
6.2	Bewegung und Interaktion mit der virtuellen Realität . . . . .	98
6.2.1	Szene . . . . .	98
6.2.2	Room-scale based . . . . .	100
6.2.3	Interaktion . . . . .	100
6.3	Graphical User Interface . . . . .	109
6.3.1	Statusanzeige . . . . .	109
6.3.2	Hauptmenü und Loading Screen . . . . .	116
6.3.3	Weitere Implementierungen . . . . .	118
6.4	3D Modellierung von Patientenzimmer und Props . . . . .	122
6.4.1	Behandlungszimmer . . . . .	122
6.4.2	Modellierung Props . . . . .	122
6.4.3	Spritze . . . . .	123
6.4.4	HME (feuchte Nase) . . . . .	126
6.4.5	Manometer . . . . .	128
6.4.6	Trachealkanüle . . . . .	131
6.4.7	Trachealkompresse . . . . .	141
6.4.8	Kanülenbänder . . . . .	142
6.4.9	Gleitgeltuch . . . . .	143
6.4.10	Hände . . . . .	144
<b>7</b>	<b>Retrospektive</b>	<b>147</b>
7.1	Ziele . . . . .	147
7.1.1	Logik und Interaktivität einer Schulungs- und VR-App . . . . .	147
7.1.2	Bewegung und Interaktion mit der VR-Welt . . . . .	147
7.1.3	Graphical User Interface . . . . .	148
7.1.4	3D-Modellierung . . . . .	148
7.2	Verbesserung . . . . .	149

7.2.1	Logik und Interaktivität einer Schulungs- und VR-App . . . . .	149
7.2.2	Bewegung und Interaktion mit der VR-Welt . . . . .	149
7.2.3	Graphical User Interface . . . . .	150
7.2.4	3D-Modellierung . . . . .	150
7.3	Zukunft . . . . .	150
7.3.1	Logik und Interaktivität einer Schulungs- und VR-App . . . . .	150
7.3.2	Bewegung und Interaktion mit der VR-Welt . . . . .	151
7.3.3	Graphical User Interface . . . . .	151
7.3.4	3D-Modellierung der Props . . . . .	152
<b>8</b>	<b>Conclusio</b>	<b>153</b>
<b>Glossar</b>		<b>155</b>
<b>Literaturverzeichnis</b>		<b>157</b>



# **Kapitel 1**

## **Gendererklärung**

Aus Gründen der besseren Lesbarkeit wird in dieser Diplomarbeit die Sprachform des generischen Masculinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.



## **Kapitel 2**

### **Danksagung**

Wir danken an dieser Stelle allen, die uns in diesem Projekt unterstützt und geholfen haben. Spezifisch bedanken wir uns bei unserem Projektbetreuer Prof. Eugen Jiresch, welcher uns sowohl technisch, als auch organisatorisch als Vermittler zwischen dem Team und dem Auftraggeber agiert hat. Durch sein hohes Engagement als Betreuers motivierte er das gesamte Projektteam.

Zudem danken wir allen Testpersonen, welche uns während des gesamten Projekts konstruktives Feedback geliefert haben, ohne der kein gutes Produkt möglich gewesen wäre. Ein besonderer Dank geht an unsere Mitschüler Luca Lengdorfer, Liam Baumann, Daniel Mladenov und an unseren Prof. Dennis Trenner, welche unser fertiges Produkt nochmals probiert und bewertet haben.

Abschließend bedanken wir uns an unseren Auftraggeber, der KWP, welche uns das Projekt ermöglicht und uns mit wichtigem Feedback und der notwendigen Hardware versorgt haben.



# Kapitel 3

## Einleitung

### 3.1 Ausgangslage

Der Auftraggeber KWP erhält regelmäßig neues Pflegepersonal, welches für notwendige medizinische Prozesse wie z.B. einen Tracheostomawechsel ausgebildet werden muss. Ein Tracheostoma stellt eine operativ geschaffene Verbindung zwischen dem äußeren Luftraum und der Luftröhre durch den Hals dar. Durch diese wird eine Trachealkanüle eingeführt, durch die nun, anstatt durch Mund und Nase, die Luft geatmet wird. Die Kanüle muss regelmäßig gewechselt werden, wobei das Pflegepersonal hierfür eine Vielzahl an Schritte beachten muss, um einerseits dem Patienten nicht zu schaden und es ihm gleichzeitig so komfortabel wie möglich zu machen.[55, 68]

Mit den Jahren werden immer mehr Pflegekräfte ausgebildet und somit steigt der Bedarf an weiteren Ressourcen, was wiederum eine Erhöhung der Kosten bedeuten würde. Ein weiteres Problem ist auch, dass der medizinische Prozess des Tracheostomawechselns nicht bzw. nur schwer in der Praxis geübt werden kann, da dies Risiken für den Pflegebedürftigen birgt. Daher wird nach einer flexiblen, wiederholbaren Trainingsumgebung in Virtual Reality (VR) gesucht, welche den Prozess simulieren kann. Das Projekt Medizinisches VR Training nimmt sich somit vor, eine dementsprechende Lösung für medizinisches Fachpersonal zu erstellen. Dafür entwickeln wir eine VR-Applikation für die Oculus Quest 2, mit der man den Tracheostomawechsel schulen kann, ohne der Notwendigkeit realer Personen, an denen trainiert werden muss.

### 3.2 Nutzen

Der Nutzen der Applikation ist es, dem Pflegepersonal eine Simulationsumgebung anzubieten, in der sie die notwendigen Schritte des Tracheostomawechsels üben und einarbeiten können. Dabei soll der Ablauf trainiert werden und eine Normalität zu diesem geschaffen werden. Damit soll nicht nur die Komfortabilität des Pflegers, sondern auch die des Patienten gesteigert werden. Ebenfalls soll durch den einfachen Aufbau der Simulationsumgebung auch die Nutzung von komplizierteren Übungen mit schwereren Vorbereitungsbedingungen verringern. Auch die Übung an realen Personen soll, durch die von der Applikation gegebene Sicherheit, verringert werden. Die Applikation zeigt sich im Vergleich zu den anderen Trainingsmöglichkeiten als kostengünstiger.

### 3.3 Grenzen

Die Technologie stößt jedoch trotz einiger Vorteile schnell an seine Grenzen. So ist es möglich, die Schritte des Ablaufs darzustellen und miteinander zu verbinden. Allerdings kommt es bei der Detailebene zu einigen Problemen. So können gewisse Bewegungen nicht dargestellt bzw. erkennbar gemacht werden, da diese einfach zu komplex für die Technologie sind. Auch Kollisionen, die der Benutzer mit den Eingabemöglichkeiten auslöst, lassen sich nur durch Vibrationen als Feedback an den Nutzer senden.

### 3.4 Planung und Umsetzung

Unsere Projektgruppe besteht insgesamt aus vier Personen: Luca Bauer, Felix Hadinger, Mihael Brsanovic und Alan Matykiewicz. Luca Bauer hat die Aufgabe, die benötigten Props und den Raum zu modellieren. Mihael Brsanovic beschäftigt sich hauptsächlich mit der Logik in unserer VR-App, wie die Erstellung von Aufgaben und die Steuerung dieser. Alan Matykiewicz übernimmt die Aufgabe, unsere VR-Simulation interaktiv zu machen um mit Objekten in der VR-Welt zu interagieren. Felix Hadinger ist für das Graphical User Interface (GUI) zuständig, was für die Darstellungen von wichtigen Informationen, wie die aktuelle Aufgabe, mögliche Optionsmöglichkeiten und Hinweise für den Benutzer zuständig ist.

Im Kapitel "Studie" wird das bearbeitete Thema detailliert analysiert und relevant Informationen darüber dokumentiert. Jene Informationen werden dann im Kapitel "Konzept" auf unser derzeitiges Projekt angewendet und mögliche Skizzen und Konzepte entworfen. Schlussendlich werden die implementierten Bereiche im Kapitel "Implementierung" dokumentiert und auf mögliche Probleme und deren Lösungen hingewiesen. Im Kapitel "Retrospektive" werden die Ziele unseres Projekts nochmal zusammengefasst und Erkenntnisse für die Zukunft daraus gezogen.

# Kapitel 4

## Studie

### 4.1 Logik und Interaktivität einer Schulungs- und VR-App

Damit eine App funktionieren, kann wird immer eine Logik-Komponente gebraucht, diese könnte man auch als die "Spielregeln" des Programms verstehen. Durch Logik kann erst Interaktivität erreicht werden, sie definiert, wie das Programm auf User-Input reagieren soll, und schafft die Grundlage für Funktionalität der App. In diesem Kapitel analysieren wir, welche Rolle Logik innerhalb von Schulungs- und VR-Apps erfüllt und welche Gedanken sich dabei gemacht werden müssen.

#### 4.1.1 VR-Schulungs-Applikationen

##### 4.1.1.1 Was ist eine Schulungs-Software?

Typischerweise werden Ausbildungen in Firmen vor Ort mit Spezialisten durchgeführt. Da dies nicht nur finanziell, sondern auch noch organisatorisch aufwendig ist, wird ständig nach Alternativen gesucht. Viele Unternehmen suchen daher digitale Abhilfe durch verschiedene Lerntechnologien, wie digitale Simulationen und auch Training-Systeme wie ein Learning Management System (LMS). Eine Schulungs-Software hat somit das Ziel digital Inhalte zu vermitteln, meist ohne die Hilfe eines Trainers. Bedingt durch die Covid-Pandemie ist die Nachfrage nach virtueller Ausbildung verstärkt. [18, 51, 16]

##### 4.1.1.2 Wofür eine Schulungssoftware?

Heutzutage entschließen sich Firmen immer öfter, eine digitale Variante ihres Trainings zu erstellen. Schulungs-Software besitzt einige Vor- und Nachteile gegenüber traditionellen Systemen, welche Sie für viele Use-Cases sinnvoll macht.

##### Vorteile einer Schulungssoftware:

- Ständige Verfügbarkeit
- Personalisierter Lernpfad
- Finanziell kaum aufwendig
- Kein Personalaufwand (Trainer, Trainingspartner)

**Nachteile einer Schulungssoftware:**

- Kleine Details meist nicht übertragbar
- Selten eigenständig sinnvoll, da es die Realität nicht perfekt abbilden kann

[18, 51]

**4.1.1.3 Wie kann eine virtuelle Schulung aufgebaut sein?**

Eine klassische Implementierung einer virtuellen Schulungssoftware sind Simulatoren. Simulatoren werden verwendet, um Auszubildende in Situationen zu versetzen, in welchen sie risikolos praktische Erfahrung sammeln können. Auch andere Methoden werden immer beliebter als Learning Management Systeme, mit welchen man Kurse zur Verfügung stellen kann, um theoretisches Wissen ansammeln zu können.

**4.1.1.4 Schulungssysteme im Kontext von VR**

Auch VR-Schulungssysteme werden bereits entwickelt und finden sich bereits in Form von Simulationen am Markt. Das Ziel einer Simulation, das Abbilden einer realen Situation, mit der Ausbildung des Nutzers als Ziel. Für viele Situationen ist jedoch eine traditionelle Simulationsumgebung nicht optimal, da diese für keine realistischen Verhältnisse sorgen können. Aufgrund dessen, dass VR-Simulatoren eine weitere Ebene an Realismus durch das Nutzen seiner Interaktionssysteme erreichen kann, eignen sich diese besonders für Situationen, wo diese Bedingungen notwendig sind.

**4.1.2 Wie wird VR heute eingesetzt?****4.1.2.1 Gesundheitswesen**

Besonders viel Interesse am Potenzial von VR-Systemen besteht im Gesundheitswesen. Hier erfüllt VR mehrere Aufgaben, beispielsweise wird es verwendet, um Auszubildende auf reale Prozesse wie Chirurgie vorzubereiten. Auch als Therapiewerkzeug selbst findet VR im Gesundheit-Sektor Verwendung, um Schmerz und Angst behandeln zu können. [35]

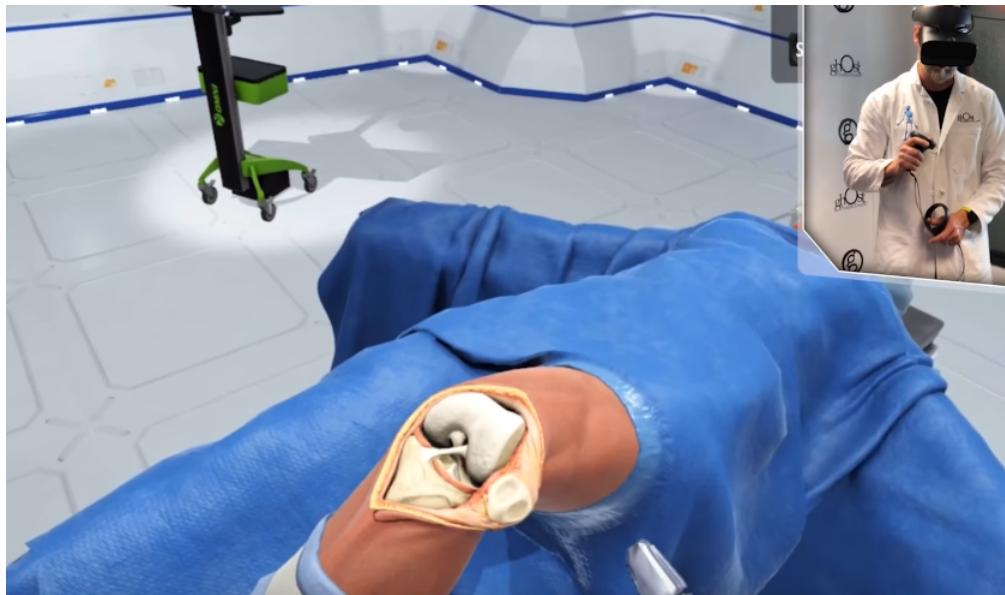


Abbildung 4.1: Eine VR-Simulation eines chirurgischen Prozesses

#### 4.1.2.2 Entertainment

Auch der Entertainmentsektor zeigt ein starkes Interesse an VR, besonders da VR-Hardware kostengünstiger geworden ist und somit kommerziell verfügbar für normale Nutzer ist. Hier sind sowohl Techgiganten wie Meta, Google als auch größere Spielentwickler wie Valve inkludiert und arbeiten an der Weiterverbreitung jener Technologien. [56]



Abbildung 4.2: Das VR-Spiel Beatsaber

#### 4.1.2.3 Bildung

Auch für Schulen und andere Bildungseinrichtungen ist die Verwendung von VR-Technologie für die Vermittlung des Stoffes am Horizont. Beschleunigt durch die Pandemie mussten sich viele Schulen einer Digitalisierung unterziehen, weshalb Projekte wie ClassVR an dieser Verwendung gefunden haben. Verwendet wird es hier, um Schülern eine besondere Perspektive zu geben, wie z.B. ein 3D-Modell des Herzens im Biologie Unterricht. [17]

### 4.1.3 Entwurfsmuster

#### 4.1.3.1 Was sind Entwurfsmuster?

Ein Entwurfsmuster definiert im generellen Lösungen für häufig wiederkehrende Software Design Probleme. Ein Entwurfsmuster ist jedoch keine fertige Vorlage, welche man einfach als Code nutzen kann. Entwurfsmuster lassen sich demnach auf mehrere Programmiersprachen übertragen und sind somit flexibel anwendbar. [69]

#### Gang of Four

Durch das Veröffentlichen des Buches "Design Patterns - Elements of Reusable Object-Oriented Software" verfasst von den 4 Autoren Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides, wurde das Konzept von Entwurfsmustern in der Softwareentwicklung verbreitet. Oft werden jene auch Gang of Four genannt. [73]

#### 4.1.3.2 Wozu benötigt man Entwurfsmuster?

Entwurfsmuster bieten sichere, schnelle und bewährte Methoden, um Design-Probleme zu lösen. Zusätzlich unterstützt die Bekanntheit dieser Muster bei der Kommunikation unter verschiedener Entwickler, indem sie klar definierte Terminologie und Problemstellungen bieten.

#### 4.1.3.3 Welche Entwurfsmuster gibt es?

##### Erzeugungsmuster

Erzeugungsmuster beschäftigen sich mit dem dynamischen Erzeugen von Klasseninstanzen.

**Factory Method Pattern:** Bei einem Factory-Pattern werden Objekte nicht wie üblicherweise durch Konstruktor-Aufrufe erzeugt, sondern stattdessen werden Methoden-Aufrufe verwendet. Ohne einem Factory-Pattern würde man wie folgend ein Objekt erzeugen.

```
1 SomeObject o = new SomeObject();
```

Im Vergleich nutzt man im Factory-Pattern Methoden, um Objekte zu erzeugen wie hier:

```
1 SomeObject o = SomeObjectFactory.createNewInstance();
```

Meist ist das zurückgegebene Objekt eine Unterklasse, hier von SomeObject. Die Vorteile des Factory Patterns sind eine leichte Erweiterung der Applikation. Für die Implementierung des Patterns sind jedoch viele Klassen notwendig.[46]

**Singleton Pattern:** Das Singleton Pattern ist ein Pattern, welches sicherstellt, dass nur ein Objekt einer bestimmten Klasse existiert und üblicherweise global verfügbar ist. Durch die Garantie und globale Verfügbarkeit des Objekts wird der Zugriff leicht gemacht. Das Singleton Pattern kann jedoch leicht zu schlechtem Code-Design führen.[64]

**Object Pool Pattern:** Das Object-Pattern ist ein Erzeugungsmuster, welches Objekte nach Erzeugung in einen Pool speichert, um diese wiederverwenden zu können. Besonders sinnvoll ist dieses Pattern, wenn häufig neue Objekte instantiiert werden. Dadurch kann man die Performance des Programms steigern und den Speicherverbrauch verringern. [49]

### Strukturmuster

Strukturmuster beschäftigen sich mit den Beziehungen zwischen einzelnen Klassen und wie diese flexibel gestalten werden kann.

**Decorator Pattern:** Das Decorator Pattern besteht aus einer Basisklasse, welche auch als Wrapper bezeichnet wird. Durch Sub-Klassen mit überschreibenden Methoden, welche auch die Super-Methode aufrufen, kann man diesen Wrapper "dekorieren". Dadurch kann man dynamisch eine Klasse mit allen notwendigen Funktionen dekorieren. Somit kann man während der Runtime einem Objekt Funktionen hinzufügen und entfernen. Ein Decorator Pattern kann jedoch unübersichtlich werden und die Reihenfolge der Dekorationen ist meist vorgegeben und relevant.[\[61\]](#)

**Adapter Pattern:** Ein Adapter-Pattern ermöglicht die Kommunikation zwischen zwei inkompatiblen Schnittstellen. Ein Beispiel wäre eine Schnittstelle welche XML an eine andere Schnittstelle die nur JSON akzeptiert weiterleiten will. Das Ändern einer der Schnittstellen könnte neue Probleme entstehen lassen. Durch eine Adapter-Klasse kann man eine Kommunikation der zwei Schnittstellen erlauben, z.B. indem es XML zu JSON übersetzt. [\[60\]](#)

### Verhaltensmuster

Verhaltensmuster beschäftigen sich spezifisch mit der Kommunikation zwischen Objekten. Relevante Beispiele für Verhaltensmuster sind:

**Observer Pattern:** Das Observer Pattern ermöglicht Objekten an einem Abonnement-System teilzunehmen, woraufhin sie von abonnierten Quellen Informationen erhalten. Somit erhalten sie für sich relevante Informationen und können dementsprechend reagieren. [\[63\]](#)

**Iterator Pattern:** Das Iterator Pattern soll den sequentiellen Durchlauf einer Collection ermöglichen, ohne Information über die Art der Collection zu aufzudecken. Durch die Implementierung eines Iterators kann die Implementierung neuer Datenstrukturen erleichtert werden, jedoch ist das Pattern meist nur bei komplexen Datenstrukturen sinnvoll. [\[62, 72\]](#)

#### 4.1.4 Individuelle Aufgabenstellung

- **Logische Infrastruktur:** Für das Projekt ist eine logische Komponente notwendig. Es soll eine logische Infrastruktur geschaffen werden, die es ermöglicht andere Komponenten und Aufgaben einzufügen zu können.
- **Prozess Management:** Ein Hauptmanager, welcher eine Schnittstelle zu den Bereichen UI und Sound bietet, ist von großer Relevanz.
- **Task System:** Eine der Hauptaufgabenstellung des Projekts ist die Integration eines Task-Systems in das Programm. Das System soll in der Lage sein dem User Aufgaben bieten und diese auch managen zu können. Eine Vielzahl an Aufgaben sollen unterstützt werden.
- **Prefabbing:** Für das Projekt werden Gegenstände, Charaktere und Räumlichkeiten benötigt. Für diese soll eine Vorlage gemacht werden, um eine mehrfache Integration so einfach wie möglich zu halten.

## 4.2 Bewegung und Interaktion mit der VR Welt

### 4.2.1 Entwicklung von VR-Applikationen

Seit der Geburt des ersten Computers war es der Traum von manchen, eine eigene Realität zu erstellen. Eine Welt mit eigenen Regeln und Erlebnissen. Das Unmögliche möglich machen. Im Jahr 2010 hat Palmer Luckey den ersten Prototypen der Oculus Rift entworfen, die erste VR-Brille. VR, oder virtuelle Realität, ist eine simulierte Erfahrung, die die reale Welt nachahmen kann oder auch ganz unterschiedlich sein kann. VR-Applikationen reichen von Unterhaltung (z.B. Video Spiele), Bildung (medizinische, wissenschaftliche, militärische Simulationen) bis zum Business (z.B. virtuelle Besprechungen). Andere Anwendungen der VR Technologie inkludieren augmented reality (AR) und mixed reality, oft zusammengefasst unter dem Begriff extended reality (XR).[11, 9, 12]

#### 4.2.1.1 Hardware-Anforderungen

VR Brillen sind neu und ressourcenhungrig und haben deswegen auch strikte Hardware Anforderungen. Diese müssen erfüllt werden, wenn man VR-Applikationen laufen will, und somit auch seine eigenen Applikationen testen will. Die Anforderungen sind für die diversen Brillen meist unterschiedlich, für den Vergleich werden hier nur die paar relevantesten Geräte erwähnt. Mit solchen **Mindestanforderungen** sollte man für die Entwicklung von VR-Applikationen rechnen:

##### HTC Vive[45]

- Grafikkarte: *NVIDIA GeForce GTX 1060, AMD Radeon RX 480 äquivalent oder besser*
- CPU: *Intel Core i5-4590 / AMD FX 8350 äquivalent oder besser*
- Arbeitsspeicher: *4 GB RAM oder mehr*

##### Valve Index[71]

- Grafikkarte: *Nvidia GeForce GTX 970 / AMD RX 480*
- CPU: *Dual Core mit Hyper-Threading*
- Arbeitsspeicher: *8 GB RAM*

##### Oculus Rift S[7]

- Grafikkarte: *NVIDIA GTX 1050 Ti / AMD Radeon RX 470 oder besser*
- CPU: *Intel i3-6100 / AMD Ryzen 3 1200, FX4350 oder besser*
- Arbeitsspeicher: *8 GB RAM oder mehr*

Unter den VR-Brillen gibt es auch neue Standalone-Geräte wie die **Oculus Quest**. Diese Brillen können als ein eigenes Gerät verwendet werden, mit der kompletten Hardware innerhalb der Brille verbaut. Neue Applikationen können darauf als **apk**-Dateien hochgeladen werden. Weiters besteht natürlich weiter die Möglichkeit, die Brille mit einem PC oder Laptop per Kabel zu verbinden.

#### 4.2.1.2 Entwicklungsumgebung

Die Software die zum Entwickeln von VR-Applikationen (Video Spiele oder Simulationen) meist benötigt wird, ist eine **Game Engine**. Game Engines gibt es auf dem Markt sehr viele, jedoch eignen sich nicht alle zur VR-Entwicklung. Von den Engines gibt es im Wesentlichen nur drei, die sich gut für die VR Entwicklung eignen.

#### Unreal Engine 4

Unreal Engine ist eine Spiel-Engine von Epic Games, die als das beste „*3D creation tool*“ vermarktet wird. Es ist für die Entwicklung von Konsolen- und Computerspielen bekannt, es wird jedoch auch in anderen Bereichen eingesetzt wie Film & TV, Architekturdesign in 3D, Simulationen und mehr. Die Engine wurde bereits im Jahr 1998 veröffentlicht, gemeinsam mit dem FPS (first-person shooter) Spiel *Unreal*. Seit der Veröffentlichung ist die Spiele-Engine aktuell bei ihrer vierten Generation, Unreal Engine 4, die fünfte Version wurde bereits im Mai 2020 vorgestellt. Unreal Engine ist unter anderem für ihre sehr fortgeschrittene Grafik-Leistung bekannt. [10, 30]

Die Unreal Engine bietet die Tools an, um lebensnahe Simulationen effizient zu entwickeln. Der Fokus liegt hier bei einem hohen Grad an Realismus, effizienter Datenverarbeitung sowie "Multi-platofrm deployment". Die Engine ist auch **XR-ready**. Es beinhaltet ein erweiterbares Framework, welches für langjährige Unterstützung von Open XR gebaut wurde. Unreal Engine unterstützt alle großen VR-Plattformen. [31, 32]

#### Unity

Unity ist eine der bekanntesten Laufzeit- und Entwicklungsumgebungen für Spiele. Es wurde von Unity Technologies im Jahr 2005 veröffentlicht und wird ständig weiterentwickelt. Die aktuelle Version 2021.2.7 wurde im Dezember 2021 veröffentlicht. Unity (auch als Unity3D bekannt) eignet sich vor allem für die Entwicklung von Computerspielen und anderen interaktiven 3D-Grafik-Anwendungen. Eines der wichtigsten Merkmale der Engine ist ihre Vielseitigkeit. Es können Anwendungen für PCs, Konsolen, mobile Geräte sowie Webbrowser entwickelt werden. Weiters ist die Haupt-Programmiersprache von Unity C# (C-Sharp), eine der gängigsten Sprachen für Spielentwicklung. Sie ist leicht zu erlernen, vor allem dadurch, dass sie viele Ähnlichkeiten mit Java teilt. Noch eine sehr wichtige Eigenschaft von Unity ist, dass jede Privatperson oder Kleinunternehmen ihre Anwendungen in Unity gratis entwickeln können. Eine Unity-Lizenz muss nur ab einem gewissen Umsatz gekauft werden. [83]

Unity ist eines der Führer der VR-Industrie. Die Plattform ist sehr gut für die VR-Entwicklung ausgelegt und hat sich durch viele erfolgreiche Titel auf dem Markt bewiesen. **Beat Saber**, eines der bekanntesten VR-Spiele, wurde mit Unity entwickelt. Die wichtigsten Tools, die Unity zur Verfügung stellt sind die High Definition Render Pipeline (HDRP) for VR, das XR Interaction Toolkit und mehr. Unity verfügt auch über eine Simulation-optimierte Version für die Entwicklung von skalierbaren Simulationen (Unity Simulations Pro). [82, 84]

#### CRYENGINE

CryEngine ist eine Spiel-Engine des deutschen Entwicklers Crytek. Die erste Version ist im Jahr

2002 erschienen, das erste Spiel mit der Engine war der FPS Far Cry (2004). Es wird ausschließlich für PC- und Konsolenspiele eingesetzt. Die aktuelle Version 5.6.5 erschien im Dezember 2019 raus. Die CryEngine verfügte für damalige Zeiten über herausragende Grafikqualität. Dadurch, und durch das in 2007 veröffentlichte Spiel Crysis, hat CryEngine ihren Ruhm gewonnen. Heutzutage ist die Spiel-Engine weniger im Vordergrund, wird trotzdem weiter in vielen großen Projekten eingesetzt. Für mit CryEngine entwickelte Spiele wurde eine 5-prozentige Abgabe (royalty) auf die mit dem Spiel erzielten Einnahmen eingeführt. [8, 21]

Die CryEngine ist nicht für die VR-Entwicklung bekannt, es ist aber trotzdem möglich VR-Anwendungen mit CryEngine zu entwickeln. Es werden folgende Plattformen unterstützt [22]:

- HTC Vive [45]
- Oculus Rift [9]
- OSVR open-source developer platform [67]

Es wurden zwei große VR-Spiele mit CryEngine entwickelt und veröffentlicht: **The Climb** (2016) und **Robinson: The Journey** (2016). [24, 23]

#### 4.2.2 Bewegung in VR

Fortbewegung in VR (engl. locomotion) ist sehr wichtig für eine gute VR Erfahrung. Fortbewegung ist die Fähigkeit, sich von einem Ort zu einem anderen in einem physischen Raum zu bewegen. In VR ist Fortbewegung die Technologie, die es dem Benutzer erlaubt, sich in der gesamten virtuellen Welt bewegen zu können. Meist rechnet man dabei mit sehr limitierten physischen Platz. [65]

Die virtuelle Realität imitiert die echte Welt, und damit die Illusion wahrhaft erscheint, muss es dem Nutzer. Dies ist in VR leider nicht so einfach möglich, ohne das Eintauchen zu ruinieren. Es existieren viele unterschiedliche Technologien für Fortbewegung in VR, alles beginnt aber bei der Hardware selbst. [65]

##### 4.2.2.1 Tracking

Ein grundlegendes Konzept hinter Fortbewegung in VR ist **Tracking**. Beim VR-Tracking werden reale Bewegungen des Nutzers von Sensoren erfasst und in die virtuelle Welt übertragen. Um die Kopfbewegungen zu erfassen, werden Sensoren innerhalb der VR-Brille benötigt. Das wird **Headtracking** genannt. Um die Position des Benutzers im physischen Raum zu ermitteln gibt es zwei (relevante) Varianten [33]:

**Externes Tracking** Beim externen Tracking (Outside-in tracking) werden unabhängig von der VR-Brille zusätzlich Kameras/Sensoren im Raum aufgestellt. Diese Geräte beobachten die Positionen von speziellen *Markern* die sich auf der Brille, VR-Controllern oder sogar auf dem Körper des Nutzers befinden können. Hier wird ein Netz von Infrarot-Lasern eingesetzt. Die Brille und die Software synchronisieren dann die positionellen Daten in Echtzeit, welche dann anschließend entsprechend verarbeitet werden. Diese Tracking-Variante wird zum Beispiel von Steam's **Valve Index** verwendet. [33, 52, 71]

**Vorteile**

- Sehr akkurate Lesungen, können durch mehr Kameras gesteigert werden
- Niedrigere Latenz als beim internen Tracking

**Nachteile**

- Okklusion, Kameras müssen direkte Sichtlinien haben
- Limitierter Nutzbereich

**Internes Tracking** Beim internen Tracking (Inside-out tracking) sind alle benötigten Sensoren in der VR-Brille verbaut. Das eliminiert die Abhängigkeit von externen Geräten. Bei dieser Variante werden mehrere Kameras in unterschiedliche Richtungen verbaut, die die Umgebung der Brille lesen. Die Methode kann mit oder ohne Markern funktionieren. HTC Vive's **Lighthouse** System benutzt beispielsweise aktive Marker. Tracking ohne Marker, wie bei der **Oculus Quest** benötigt gar keine externe Elemente. Hier wird ein Prozess namens **SLAM** verwendet; mehrere, auf der Brille verbauten Kameras bilden die Umgebung ab, aus dem dann eine 3D-Umgebung in Echtzeit generiert wird. [33, 52]

**Vorteile**

- Erlaubt größere Spielräume, die können an den Raum angepasst werden
- Adaptiv für neue Umgebungen, tragbar

**Nachteile**

- Mehr Verarbeitung an Bord benötigt
- Latenz kann höher sein

Weiters sind bei den Angaben zum Tracking auf einer VR-Brille eins der Begriffe zu finden: **6DOF** oder **3DOF**. Diese kennzeichnen die sogenannten Freiheitsgrade (engl. degrees of freedom), die das jeweilige Headset ermöglicht. Bei 3DOF ist nur das Neigen, Schwenken und Drehen möglich. Man kann nicht in die Tiefe des Raums gehen, die Brille ist logisch gesehen statisch. 6DOF ist es zusätzlich auch möglich, die Tiefe des virtuellen Raumes zu betreten. Die VR-Brille kann sich auf drei Achsen in der VR-Welt bewegen, Vor und Zurück, Rechts und Links sowie Hoch und Runter. Für ein natürliches VR-Erlebnis eignet sich nur 6DOF. Die meisten high-end VR-Brillen unterstützen heutzutage 6DOF. [33, 52]

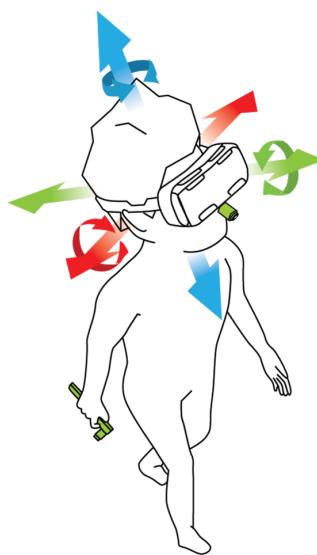


Abbildung 4.3: Sechs Freiheitsgrade

#### 4.2.2.2 Richtlinien für künstliche Fortbewegung

Bei der Entwicklung von Anwendungen mit künstlicher Fortbewegung sollte man einige wichtige Elemente bedenken. Diese Elemente werden laut den **Oculus for Developers guidelines** empfohlen. [65, 27]

- **Beschleunigung** - schnelle, unerwartete Bewegungen können für den Nutzer unangenehm sein, und in manchen Fällen zu Übelkeit führen; besondere Aufmerksamkeit soll auf Torso- und Kopf-Bewegungen gelegt werden
- **Geschwindigkeit** - um in der Reihe der menschlichen Wahrnehmung zu bleiben, empfiehlt Oculus Geschwindigkeiten der menschlichen Fortbewegung zwischen 1.4 und 3 Metern pro Sekunde
- **Richtung und Tracking** - bei der Verwendung von Motion-Controller setzt der Benutzer seinen Spielbereich (sog. guardian mode); Entwickler sollen verstehen, dass Rückwärts- und Lateral-Bewegungen selten sind
- **Nutzerkontrolle** - die Nutzer sollen so viel Kontrolle über die Bewegung wie möglich haben; das ist extrem wichtig für das Gameplay und das Vergnügen der Nutzer
- **Visuelle Qualität** - die Qualität der Grafik ist ebenfalls von zentraler Bedeutung; es gibt eine dünne Linie zwischen Qualität und Leistung. Abrupte Bewegungen wie Kopfschütteln können die immersive Erfahrung unterbrechen.

Anschließend muss man noch die Bildrate der entwickelten Anwendung bedenken. Das Ziel für VR-Entwickler ist es, jederzeit 90 FPS (*frames per second*) in ihrer Software zu erzielen.

*„One of the biggest technical challenges of VR is delivering content at a high enough frame rate to accurately ‚trick‘ the user into believing he or she is experiencing the external world.“*

*Studies have shown that in practice, any VR setup that generates frame rates below 90 frames per second (FPS) is likely to induce disorientation, nausea, and other negative user effects. The lower the frame rate, the worse the effects.“ [47]*

#### 4.2.2.3 Lösungen für Fortbewegung in VR

Es gibt verschiedene Methoden, wie sich der Benutzer innerhalb der virtuellen Umgebung fortbewegen kann. Jede Methode hat ihre eigenen Vorteile, Herausforderungen und Anwendungsfälle. Deswegen ist es sehr wichtig schon vor der Entwicklung zu entscheiden, welche Technik für die Anwendung am besten passt. [65]

Die Fortbewegungstechniken sind über Jahre ausführlich getestet und verfeinert worden, mit dem Ziel einer nahtlosen und benutzerfreundlichen Navigation einer virtuellen Umgebung. Als Erstes werden die Methoden in zwei Interaktionstypen aufgeteilt: **physisch** und **künstlich**. Bei den physischen Fortbewegungsmethoden hängt die Navigation von den physischen Bewegungen des Nutzers ab; die Schritte des Nutzers werden direkt in die Software als Bewegung des Avatars übersetzt. Künstliche Fortbewegung ist die meist eingesetzte Variante. Hier werden die echten Bewegungen des Benutzers ignoriert, stattdessen werden die von den Entwicklern vorgegebenen Methoden eingesetzt um die Welt zu navigieren. [65]

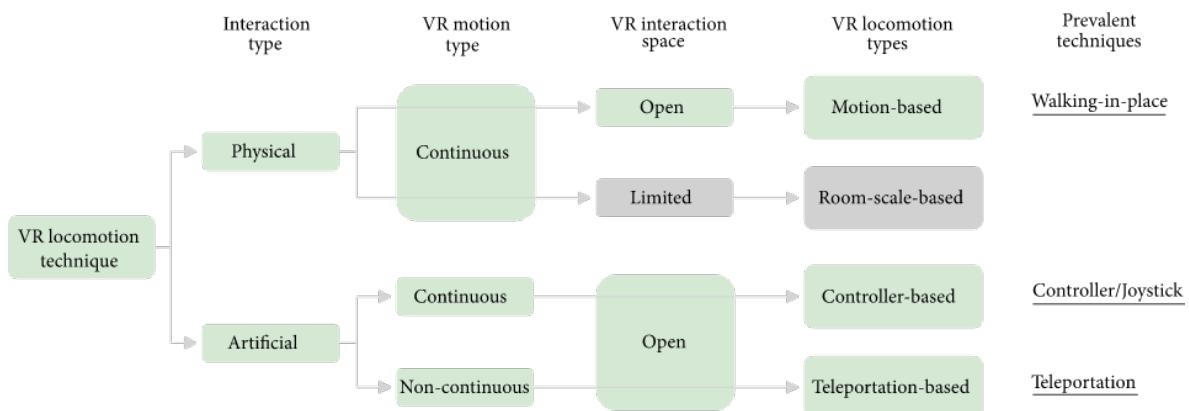


Abbildung 4.4: Fortbewegungstechniken für VR

**Room-Scale Based** Dieser Ansatz verwendet ausschließlich die physischen Bewegungen des Benutzers innerhalb seines physischen Spielbereichs. Das bedeutet, dass die Navigation durch die Ausmaß des Raumes limitiert ist. Anwendungen die diese Methode verwenden müssen selbstverständlich diese Bedingung bei der Entwicklung bedenken. **Beat Saber** ist ein gutes Beispiel für diesen Ansatz. Da werden die Bewegungen des Benutzers ausschließlich fürs Tanzen und Ausweichen der Wände gebraucht.

Auch wenn die entwickelte Applikation nicht für physische Fortbewegung gedacht wurde, ist es wichtig zu Durchdenken und Planen, was passiert, wenn sich der Benutzer in der echten Welt bewegt. Möglicherweise kann die virtuelle Kamera (und somit der Kopf des Nutzers) mit solider Geometrie kollidieren (Wände, **Props**, Charaktere). Solche Szenarien müssen eingeplant werden.

**Motion-Based** Der Ansatz benötigt zusätzliche externe Sensoren, um sämtliche physischen Bewegungen festzustellen und diese in VR-Bewegung zu übersetzen. Solche Hardware ist aber aufgrund der hohen Kosten bei keinem der großen Headsets vorhanden. Grundsätzlich wird diese Methode selten bei regulären Konsumenten in Verwendung kommen. Die bekanntesten derartigen Sensoren sind **VR-Treadmills**, die verhältnismäßig realistisches Laufen in sämtliche Richtungen innerhalb einer virtuellen Umgebung erlaubt. Diese sind aktuell noch teuer, und brauchen viel Platz, weswegen sie nicht viel Verwendung sehen.



Abbildung 4.5: Ein VR-Treadmill

**Controller-Based** Üblicherweise haben VR-Controller auch eigene Joysticks. Somit werden bei diesem Ansatz diese Joysticks wie bei regulären Controllern für kontinuierliches Fortbewegen in der virtuellen Umgebung eingesetzt. Die Wahrnehmung von Bewegung in der virtuellen Welt, während man in der echten Welt stationär ist, kann zu Übelkeit führen, bzw. zu Vestibulären Fehlanpassung (engl. Vestibular Mismatch). Zusätzlich wird bei dieser Methode oft der rechte Joystick dazu verwendet, um den Avatar um 90 Grad links oder rechts zu drehen. Diese Drehung geschieht abrupt und wird bei den Meisten als sehr unangenehm empfunden. [vestibular\_mismatch]

**Teleportation Based** Bei dieser Variante wird die Fortbewegung durch sofortige Änderungen der Position innerhalb der virtuellen Spielwelt implementiert. Das Zielort wird durch den Nutzer mittels des Controllers festgelegt, das heißt der Nutzer kann sich seine neue Position präzise aussuchen. Diese Methode ist eine in Spielen häufig eingesetzte Fortbewegungstechnik. Es gibt auch weniger Bedingungen oder Bedenken, was die Entwicklung erleichtern soll. Da die sofortige Positionsänderung ohne Kontext für manche unangenehm sein kann, gibt es zwei Varianten für die Teleportation:

- **Blink** - hier wird nach der Auswahl des Zielortes die Sicht des Nutzers ausgeblendet, wonach der Avatar zu seiner neuen Position verlegt wird und die Sicht des Nutzers anschließend wieder eingeblendet wird; das Ganze soll für den Nutzer wie ein blitzen erscheinen
- **Dash** - die Blink-Variante wird öfter eingesetzt, kann jedoch die Immersion des Spielers hindern. Dafür gibt es die Dash-Variation, wo der Nutzer nicht sofort bei seinem Zielort landet, sondern dorthin mit Supergeschwindigkeit geeilt wird. Hier muss man sich wieder dem *Vestibular Mismatch* Problem bewusst sein

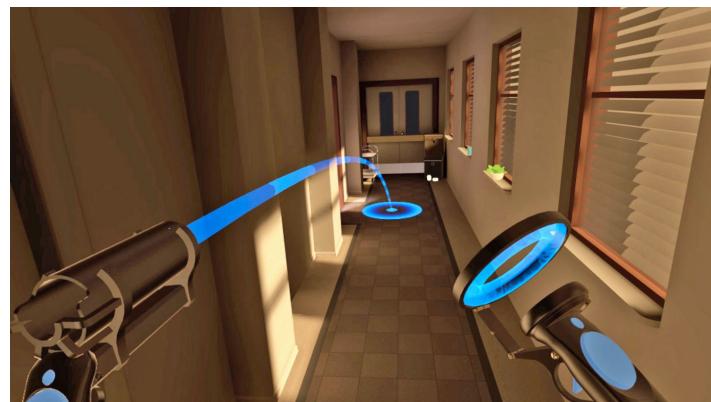


Abbildung 4.6: Beispiel einer Blink Anwendung

#### 4.2.3 Interaktion mit Objekten in VR

Beinahe jede gängige VR-Brille hat eigene VR-Controller, die an das jeweilige Trackingsystem angepasst sind. Sie simulieren Finger-, Hand- und Armbewegungen und ermöglichen auf Tastendruck oder über Analogsticks beziehungsweise Touchpads mit der virtuellen Realität zu interagieren.

Neben den VR-Controllern befinden sich bei der Hardware noch VR-Handschuhe. Diese werden momentan nur in kostenintensiven Prototypen verwendet. Solche Geräte sind für Privatanwender nicht sinnvoll und kommen nur bei größeren Unternehmen ins Spiel. Zuletzt gibt es noch die Alternative des Handtracking. Diese Technologie funktioniert mittlerweile für bestimmte Anwendungen gut und könnte in der Zukunft der Interaktionsstandard werden. Bis dann ist es nur bei wenigen Geräten zu finden, somit unflexibel für Anwendungen, mit denen ein Kunde interagieren soll. Weiter fehlt hier das **haptische Feedback** und das „handhaben“ von Objekten/Props kann für Manche unnatürlich vorkommen. Handtracking ist momentan nur eine Besonderheit für bestimmte Anwendungen. [33]

##### 4.2.3.1 Interaktion in der Entwicklung

Bei der Entwicklung von Interaktionssystemen für VR muss man mit den entsprechenden Schnittstellen arbeiten. Die VR-Brille ist die **Kamera** innerhalb der Software, und die Kopfbewegungen des Nutzers sollen so die Kamera auch mit bewegen. Die VR-Controller können dann ebenfalls leicht in die Szene inkludiert werden als einzelne Objekte. Für diese Zwecke muss man mit einem Setup arbeiten für die bestimmte Hardware. Für die Interaktion kann man sich entweder auf die zur Verfügung gestellten Libraries/Frameworks verlassen, oder die komplette Logik selbst entwickeln. Hier findet man beispielsweise in Unity das **XR Interaction Toolkit**, welches ein solides Gerüst für eine VR-Applikation bildet. Die restlichen benötigten Features müssen dann selbst entwickelt werden. [85]

Es ist möglich, sich auf Nutzer-erstellte Frameworks zu verlassen, welche ein komplettes Interaktionssystem bilden würden. Unity sowie Unreal Engine verfügen über ihren eigenen „Markt“, in dem Nutzer selbsterstellte Packages und Frameworks verkaufen können. Für beide Engines kann man gute Interaction Toolkits auf den Märkten finden. Der durchschnittliche Preis für so ein Framework liegt bei ca. 80€. [40, 29]

#### 4.2.3.2 Benötigte Features

Entwickelt wird eine VR-Schulungs-Software für medizinische Zwecken. Folgende Features werden von dem **Interaktionssystem** benötigt:

- **Greifen:** unterschiedliche Objekte (**Props**) müssen berührt und aufgehoben werden können. Das inkludiert auch das Loslassen von den Objekten. Das aufgehobene Objekt befindet sich in der entsprechenden Hand und bewegt sich mit der Hand mit
- **Interaktion zwischen unterschiedlichen Objekten:** sämtliche Objekte müssen miteinander interagieren können, zum Beispiel ein Prop muss mit einem anderen Prop berührt werden. Das soll mit einem in der Hand gehaltenen Objekt möglich sein. Die Objekte sollen miteinander kollidieren
- **Verbinden von Objekten:** für bestimmte Vorgänge müssen sämtliche Objekte miteinander verbunden werden. Verbundene Objekte bilden dann ein einzelnes Objekt, ein Gefüge von Objekten (**Compound Object**). Beim Verbinden von Objekten sollte es eine Vorschau bei dem Zielobjekt geben
- **Herausnehmen von Objekten aus Zusammenfügen von Objekten:** aus einem Compound Object sollte man bestimmte Objekte die Teil des Zusammenfüge sind herausnehmen können. Das Objekt welches entfernt werden soll muss in der Logik bestimmt werden, die anderen Teile des Gefüge bleiben weiter hängen und sollten nicht herausgenommen werden dürfen
- **Interaktion mit dem User Interface:** der Benutzer soll mit UI-Elementen interagieren können: Knöpfe, Schalter, Slider. Die Interaktion findet mit den VR-Controllern statt

## 4.3 Graphical User Interface (GUI)

Dieses Kapitel befasst sich mit dem Begriff Graphical User Interface (GUI) und deren Anwendungsfälle im Bereich VR. Ebenso werden hier die Richtlinien angesprochen, die für GUIs allgemein empfohlen werden.

### 4.3.1 Was ist eigentlich ein GUI?

Graphical User Interface oder Grafische Benutzeroberfläche bezeichnet im generellen eine Schnittstelle bei Computerprogrammen, durch die der Benutzer mit grafischen Elementen oder auch "Widgets" genannt, mit dem Programm interagieren kann. Grafische Elemente können z. B. Buttons, Text, Bilder, Slider usw. sein. Diese Kommunikation wird auch als human computer interaction (HCI) bezeichnet. Dies erleichtert das Benutzen eines Programms, da Befehle oder textbasierte Eingaben für die Kommunikation nicht mehr benötigt werden[28, 92, 15].

Ein einfaches Beispiel für ein GUI sind die Windows Einstellungen. Um z. B. die Zeitzone des Computers zu ändern ist es nicht mehr nötig, einen Befehl in der Kommandozeile einzugeben. Das Drücken ein paar Buttons genügt, um die gewünschten Änderungen durchzuführen:

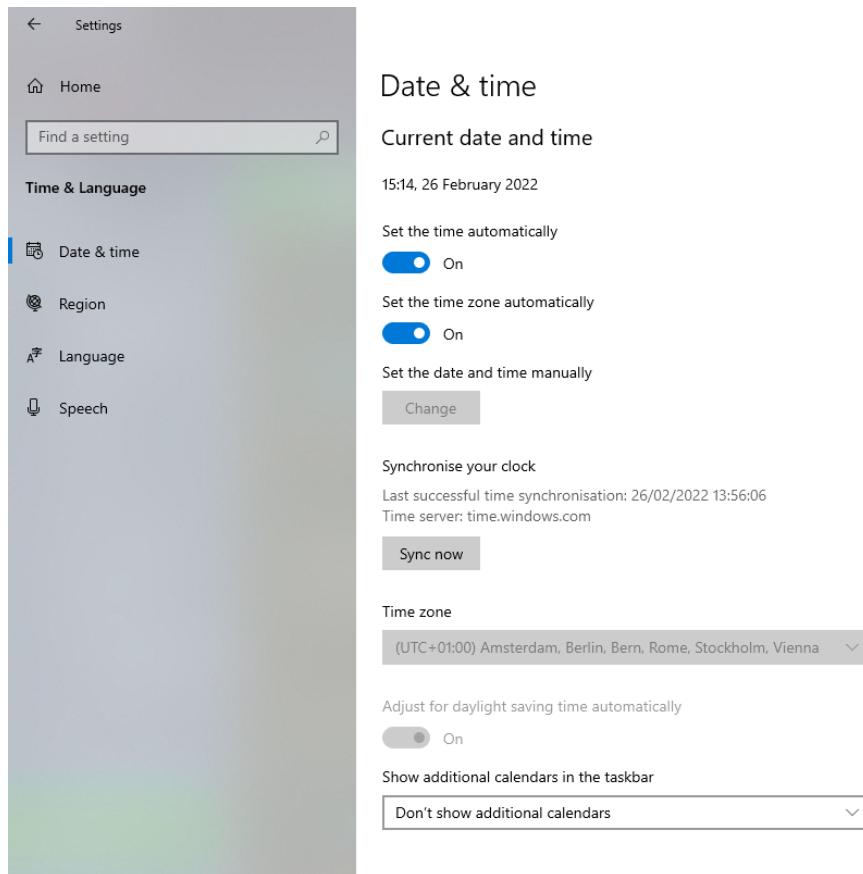


Abbildung 4.7: Windows Einstellungen

GUIs werden aber nicht nur in Programmen und Apps benutzt sondern auch in Videospielen

und in VR-Umgebungen. Mit ihnen ist es möglich, Einstellungen im Spiel vornehmen, seinen Fortschritt zu speichern/laden/löschen oder seinen derzeitigen Standort auf einer Landkarte zu überprüfen. Diese sind nur ein paar von vielen Anwendungen, die ein GUI übernimmt.

### 4.3.2 Head-up Display und World Space

Generell sind GUIs in vier Klassen unterteilt. Da aber für dieses Projekt eine Trainings-Applikation erstellt wird, haben nur zwei Klassen für uns Relevanz:

- Head-up Display (HUD) bzw. Non-Diegetic Components
- World Space bzw. Spatial Components

Beide Klassen haben ihre Vor- und Nachteile und es hängt immer von den Anforderungen ab, welche benutzt werden. Genaue Infos dazu werden in den nächsten Kapiteln erläutert[14].

#### 4.3.2.1 Head-up Display in virtual Reality

Für einen schnellen Zugriff auf wichtige Einstellungen oder Informationen gibt es sogenannte HUDs.

„Das Head-up-Display (HUD; wörtlich: „Kopf-oben-Anzeige“) ist ein Anzeigesystem, bei dem der Nutzer seine Blickrichtung und damit seine Kopfhaltung beibehalten kann, weil die Informationen in sein Sichtfeld projiziert werden [93].“

In normalen Spielen bezieht sich die Blickrichtung auf die Kamera, aber in VR-Applikationen auf die Kopfhaltung des Benutzers. In Abbildung 4.8 ist das HUD System von dem Spiel Overwatch rot umrahmt.



Abbildung 4.8: Screenshot vom Spiel “Overwatch”

Hier werden relevante Informationen gezeigt, die während des Spiels wichtig sind wie z.B: die Lebensanzeige des Spielers (links unten).

#### 4.3.2.2 User Interface Widgets im World Space

Anders als bei HUDs sind die Elemente im World Space (wie der Name schon sagt) in der Spielwelt dargestellt. Dies ist dann sinnvoll, wenn Informationen von einem bestimmten Objekt abgerufen oder dargestellt werden. Dies könnte eine Sprechblase einer Person sein oder einen Button an der Wand, der eine gewisse Aktion durchführt.

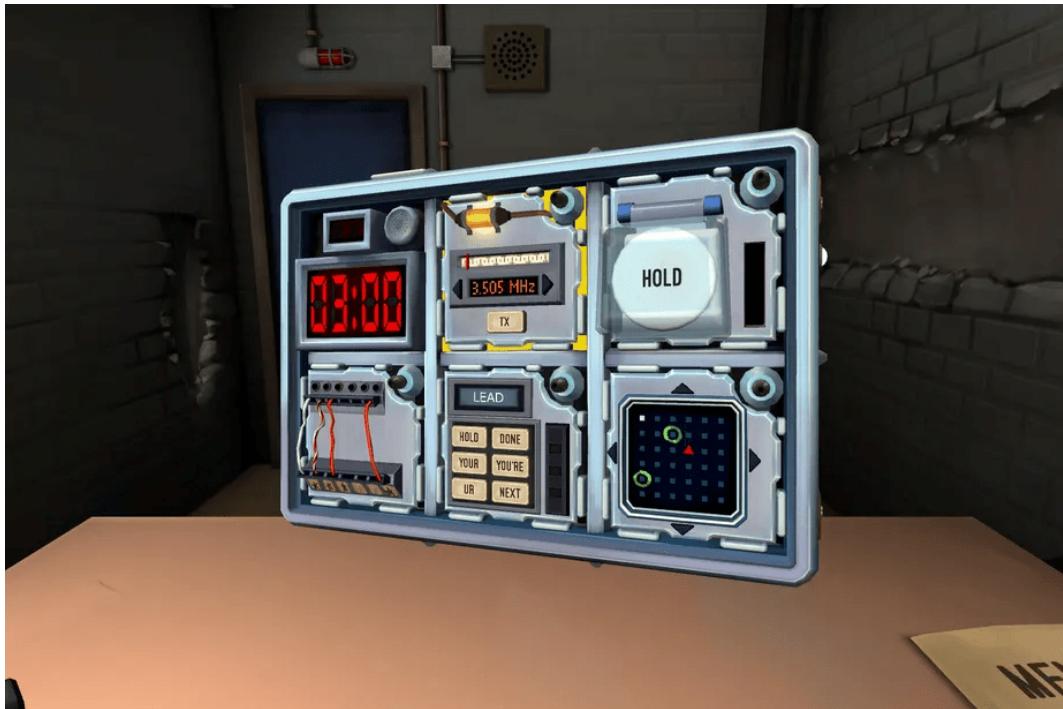


Abbildung 4.9: Screenshot vom VR Spiel "Keep Talking and Nobody Explodes" [50]

Bei Abbildung 4.9 haben die Widgets im World Space die Aufgabe bestimmte Informationen an den Spieler zu senden. Ein Beispiel dafür ist die große rote Zeitanzeige oben links an der Box. Diese könnte z. B. für einen Countdown stehen.

#### 4.3.3 Richtlinien und Empfehlungen

Über GUIs kann der Benutzer auf wichtige Informationen zugreifen und Einstellungen ändern. Damit dies auch reibungslos funktioniert müssen diese intuitiv, simpel und einfach zu benutzen sein. Da GUIs im Grunde genommen alle aus demselben Grundgerüst bestehen, haben sich Entwickler über die Jahre auf bestimmte Grundsätze geeinigt um die oben genannten Ziele zu erreichen. Die nächsten Unterkapitel werden ein paar der wichtigsten Grundsätze erläutern und dazu passende Beispiele auflisten.

##### 4.3.3.1 Information architecture (IA)

Mit einem gut strukturierten GUI findet sich der Benutzer auch ohne Anleitung und ohne das Programm jemals benutzt zu haben, zurecht. Diesen Bereich nennt man auch Information Architecture (IA) und ist eines der wichtigsten Bausteine eines guten GUIs. Dies kann durch eine

klare, systematische Darstellungen der vorhandenen Informationen und Optionen erreicht werden. Konsistenz im Bereich Strukturierung von Inhalten kann hier auch ein wichtiger Faktor im Bereich klarer Darstellung sein [37].

Ein Beispiel dafür sind Änderungen von Lautstärke-Einstellungen. Diese sollten immer schnell zu finden sein und haben meist diese Reihenfolge:

1. Öffnen der Haupteinstellungen
2. Auswahl des Unterpunktes namens "Lautstärkeeinstellungen"
3. Lautstärke wird durch einen Slider oder ähnlichen Möglichkeiten verändert.

#### 4.3.3.2 Farben und Formen

Mit Farben ist es nicht nur möglich, bestimmte Elemente hervorzuheben, sondern auch, Widgets eine zusätzlich emotionale Bedeutung zuzuweisen. Dies kann durch die Benutzung von Farbsymbolik und Farbtheorie erreicht werden. Rot steht z. B. für Gefahr, Kraft und Blau für Ruhe und Intelligenz. [53, 37].

Aber nicht nur Farben beeinflussen die Wahrnehmung von grafischen Elementen, sondern auch Formen. Sei es die Größe oder das scheinbare Material des Widgets, alles beeinflusst die Darstellung des Elements und somit auch die Reaktion des Benutzers darauf. Ebenso sollte darauf geachtet werden, dass bestimmte Formen auch das machen, was von ihnen erwartet wird. Ein Button sollte integrierbar und ein Slider (Schieberegler) verschiebbar sein. [48, 37].



Abbildung 4.10: Unterschiedliche Buttons

#### 4.3.3.3 Clean design

Detailliertere Informationen scheinen immer gut zu sein, aber manchmal ist weniger mehr. Zu viele Informationen können den Benutzer überfordern oder die Ansicht unübersichtlich machen. Dies kann vermieden werden, indem die wichtigsten Anwendungen sofort sichtbar sind und die anderen in zunächst versteckten Menüpunkten liegen. Die Nutzung von Icons kann auch vor einem Überfluss an Informationen schützen, da diese simpler zu erkennen sind und weniger Platz verbrauchen. Es wird auch öfters eine Methode namens "Progressive disclosure" angewendet, bei der nicht alle Optionen von Anfang an dem Benutzer zu Verfügung stehen, sondern sich mit der Zeit erweitern [37].

#### 4.3.3.4 VR spezifische Empfehlungen

Alle oben aufgelisteten Guidelines bzw. Empfehlungen sind universal auf alle GUIs anwendbar. Es gibt aber auch VR-spezifische Empfehlungen von Oculus, die sich auf die Benutzung von Widgets spezialisieren.

- **Buttons:** Es gibt viele states (=Zustände), die ein Button haben sollte, wie der default state (Ausgangszustand), pressed state und **Hover state**. Die Farben jeder dieser states sollte eindeutig sein, damit klar ist, welcher gerade aktiv ist. Dies ist besonders wichtig, wenn versucht wird, mit einer Hand einen Button zu drücken. Ebenso sollten Buttons nicht kleiner als 6 cm und der Abstand zwischen ihnen nicht kleiner als 1 cm sein (Die Maßangaben beziehen sich auf die Distanz des Controllers, die benötigt wird zwischen z. B. zwei Buttons zu wechseln). Dadurch wird verhindert, dass der Benutzer einen falschen Button drückt, oder einen gar übersieht [57].

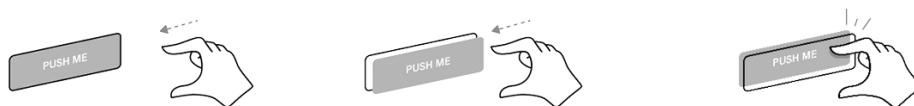


Abbildung 4.11: Ein Beispiel für die 3 Buttons States von Oculus

- **Pinch-and-Pull (= einklemmen und ziehen) Komponenten:** Um eine Liste an Elementen in VR durchzugehen ist die "Pinch-and-Pull" Variante eine bessere Möglichkeit als eine Scrollbar, da sie wesentlich präziser mit den Controllern zu benutzen ist [57].

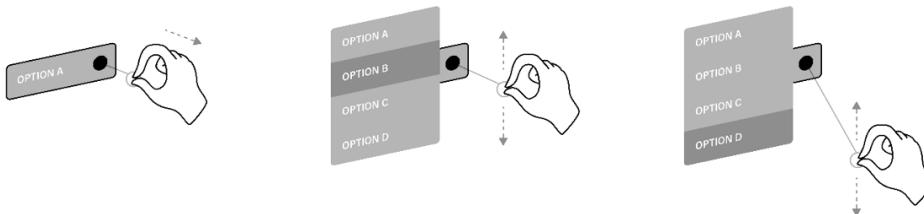


Abbildung 4.12: Pinch-and-Pull Komponenten von Oculus

- **Pointer (= Zeiger):** Oculus empfiehlt auch einen Pointer zu benutzen, statt einen Raycast, da dadurch die Präzision der Auswahl, bzw. Bestätigung von Elementen/Buttons erhöht wird.[57]



Abbildung 4.13: Pointers von Oculus

#### 4.3.4 Individuelle Aufgabenstellung

Da nun die wichtigsten Aspekte von GUIs bekannt sind, stellt sich nur noch die Frage, welche Aufgaben diese in unserem Projekt hat. Diese sind folgende:

- **Anzeige der einzelnen Schritte des Tracheostomawechsels:** Da es mehrere Teilschritte für den Tracheostomawechsel gibt, die dem Benutzer evtl. noch nicht bekannt sind, wird ein Interface benötigt, indem diese Aufgaben systematisch aufgelistet werden.
- **Darstellungen von User Feedback:** Der Benutzer soll informiert werden, wenn eine Aufgabe abgeschlossen wird, oder wenn andere wichtigen Informationen während einer Aufgabe angezeigt werden sollen.
- **Hauptmenü mit Einstellungen:** Es soll ein Hauptmenü bereitgestellt werden, indem es möglich sein soll das Training zu starten und einzelne Einstellungen, wie Lautstärke oder Steuerung einzusehen und zu verändern.
- **Loading Screen:** Möglicherweise wird beim Übergang vom Menü zum Training ein **Loading Screen** benötigt, da Files und Objekte geladen werden müssen.

## 4.4 3D Modellierung

Die 3D-Modellierung befasst sich mit der Darstellung der Umgebung im virtuellen Raum. Sie gehört zu den Teilen der Simulation, die der Benutzer im Endeffekt sehen und mit interagieren muss.

Dieses Kapitel befasst sich somit, wie man diese virtuellen Modelle dazu verwenden kann, komplexe und detaillierte Szenen oder Gegenstände in der virtuellen Welt zu erschaffen. Dabei werden die verschiedenen Techniken und Tools evaluiert und aber auch auf die Herausforderungen, die aus der 3D-Modellierung resultieren, wird hier eingegangen

### 4.4.1 Entwerfen digitaler Modelle

Die Idee dieser Modelagerungsart ist es ein 3 dimensionales virtuelles Modell eines Objektes zu erstellen. Diese können sowohl Ebenbilder realer Objekte oder Lebewesen sein, aber auch der Fantasie entspringen. Während sie bereits beliebt in Spielen und Simulationsumgebungen sind, finden sie immer mehr im Design, Ingenieurswesen, Druck und im Webdesign eine Verwendung.

3D Modelle stellen einen geometrischen Körper, durch eine Sammlung an Punkten (**Vertices**) im 3D-Raum dar, die je nach Software mit geometrischen Formen (meist Dreiecke und Vierecke) miteinander verbunden werden (**Mesh**). Dies soll eine komplette Oberfläche des Modells darstellen. Während sie meist eher die Größe und Form der Objekte darstellen, kommen mittlerweile auch oft die Texturen des Objektes mithinein. So können moderne Computer bereits aufwendigere Oberflächen generieren und darstellen, wodurch diese nicht mehr nur wie eine Illusion über die Farben auf das Modell gebracht werden müssen.[\[43\]](#)

#### 4.4.1.1 Modellierungsmethoden

Es lassen sich auf zwei Arten 3D-Modelle erstellen:

- **3D-Modellierung**
- **3D-Scan**

In der **3D-Modellierung** werden die Objekte von Beginn in einem 3D-Modellierungs-Tool erstellt, auch wenn es dieses in der Realität gibt. Dabei gibt es unterschiedliche Tools, welche aber alle mit einer simplen Basis anfangen und dementsprechend weiter gebaut werden. Diese ist die beste Möglichkeit fiktive Objekte zu erstellen und bietet im weiteren Verlauf auch mehr Freiraum, wenn man diese Modelle dann auch manipulieren möchte und ihnen ein Verhalten einbauen will.

Im Gegensatz dazu gibt es noch den **3D-Scan**. Hier muss man auf ein bereits existierendes Objekt vertrauen, denn man scannt dieses nur und erstellt es nicht von Grund auf. Für einen Scan braucht man viele Bilder aus vielen Blickwinkel, aus denen dann die Oberfläche des digitalen Modells berechnet wird. Der große Vorteil liegt hier in dem bereits existenten Objekt. Weiters kann man so einfacher Detailierungen dem Objekt anhängen, wenn dafür das Verhalten - also Animation oder Physik - in den Hintergrund rückt.

#### 4.4.1.2 Vorbereitung und Entwürfe

Wie bei jedem System sollte man auch hier nicht ganz planlos hineingehen und einfach drauflos modellieren. So muss man sich erst einmal überlegen, welche Methode nun angewendet werden soll und auf welche Details, im weiteren Verlauf, geachtet werden muss.

##### Vorlagen

Eine Möglichkeit sich ein Konzept zu bilden, ist es sich Vorlagen zu beschaffen. Dazu zählen reale Beispiele, Bilder aus Print oder Internet. Diese Methodik hilft, wenn reale Objekte modelliert werden sollen und daher mehr Zeit in die Richtigkeit der Darstellung gehen muss.

##### Concept Arts und Skizzen

Eine ebenfalls beliebte Entwurfsmöglichkeit ist das Gestalten von Skizzen oder besonders bei Spielen das Gestalten sogenannter "Concept Arts". Diese Concept Arts sollen so das zu modellierende Objekt oder Lebewesen aus mehreren Blickwinkeln betrachten, da so meist komplexere Modelle angegeben werden. Meist werden Farben auch in den Concept Arts mitgegeben, sodass der Modellierer ein klares Bild vor Augen hat. [91]

Diese Concept Arts geben meist auch ein klares Bild bei der Animation von Modellen. Diese sind so leicht vorzubereiten und festzulegen. Überhaupt bieten Concept Arts ein sehr gutes Konzept, da sie ein klares Bild zeigen.



Abbildung 4.14: Beispiel einer Concept Art eines professionellen Künstlers (Konstantin Maystrenko )[19]

## 4.4.2 3D-Modellierungssoftware und Modellierungstechniken

### 4.4.2.1 Modellierungstechniken

Für die 3D-Modellierung gibt es verschiedene Software sowohl kostenlos als auch kostenpflichtig. Die für Spiele und Animationsfilme meist bekannten sind **Blender** und **Maya**, welche auch die stärksten und umfangreichsten Tools darstellen, da sie für jeden Bereich der Modellierung nutzbar sind. Überhaupt gibt es für die 3D-Modellierung 3 unterschiedliche Modellierungstechniken:

- **Splines** (Kurven)
- **Box-Modelling** (Polygone)
- **Poly-Modelling** (Polygone)

**Splines** sind Kurven im 3-dimensionalen Raum, welche aus zumindest 2 Kontrollpunkten besteht. Diese ist die wohl älteste und traditionellste Form der 3D-Modellierung, welche momentan existiert. Hier wird ein sogenannter Spline-Käfig erstellt, der das Skelett des Modells darstellen soll, welchen das Programm dann verwendet, um einen Polygonkörper daraus zu bilden. Allerdings stellt diese Modellierungsform auch eine hohe Komplexität in ihrem Umgang dar und ist auch nicht für Animationen und Manipulation ungeeignet. Trotzdem lassen sich abgerundete statische Objekte, wie Kelche und Teller, in schneller Zeit aufbauen[1] [86].

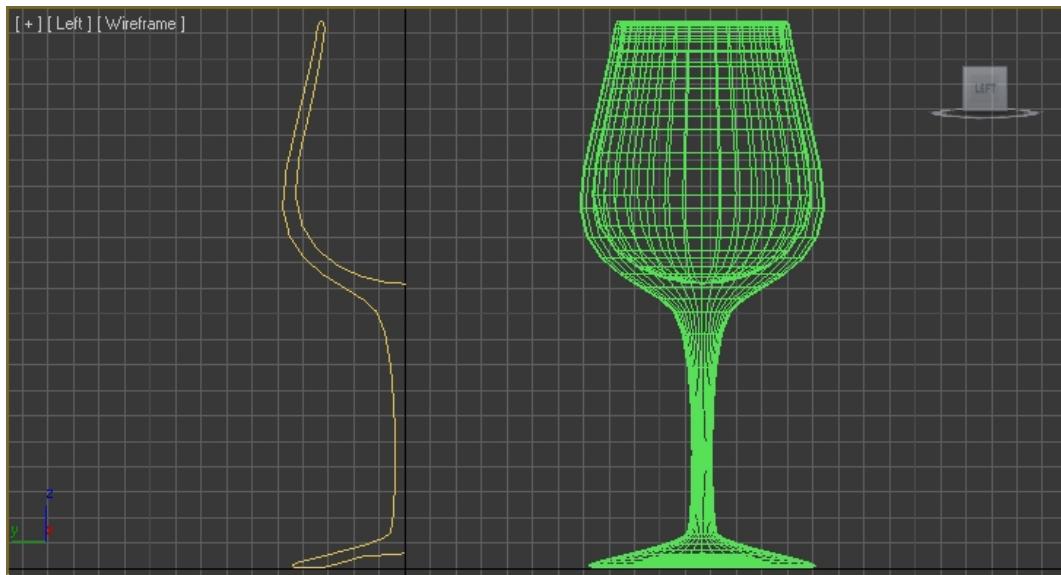


Abbildung 4.15: Spline-Modelling für ein Weinglas (links Skelletkurve, rechts fertiges Modell)

Dabei stellen die wohl gebräuchlichsten Tools die **Bezierkurven** und **NURBS** dar.

- **Bezierkurven:** Diese wurden vom französischen Ingenieur **Pierre Bézier** erfunden. Dabei wird die Kurve mittels Punkten generiert. Bei der simplen Bezierkurve gibt es beispielsweise nur 2 Punkte zwischen denen die Kurve verläuft. Allerdings wird die Kurve durch

Kontrollpunkte, welche sich immer an einer Tangente der Kurve befinden, die Kurve mehr und mehr manipuliert werden.[74]

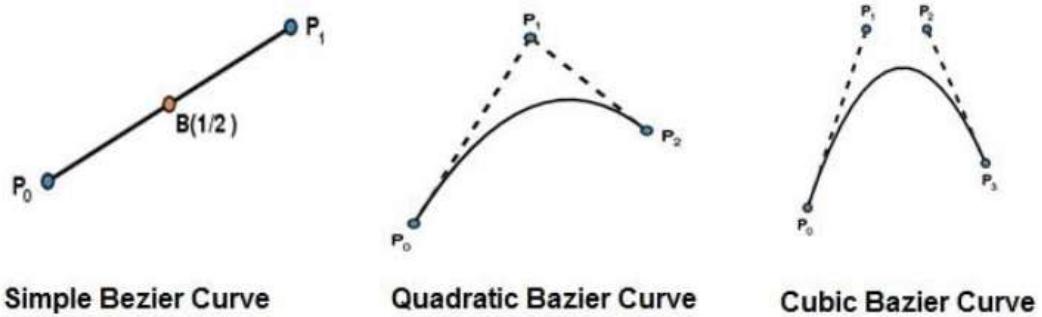


Abbildung 4.16: Darstellung unterschiedlicher Bézierkurven

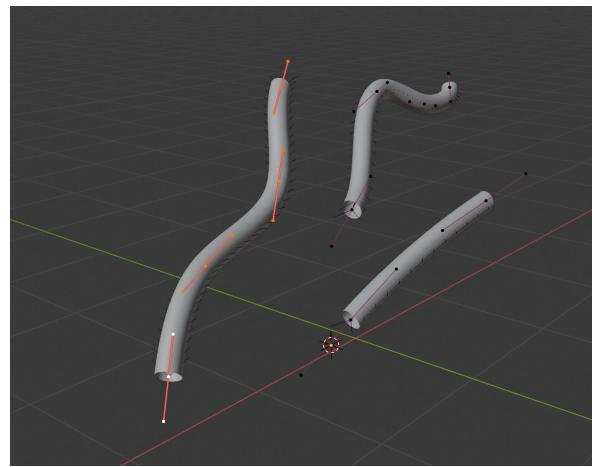


Abbildung 4.17: Darstellung eines Modells mittels Bezierkurve in Blender

Neben der Nutzung in der 3D-Modellierung werden die Bézierkurven mittlerweile auch im Videoschnitt und Animation gefunden. Besonders im Bereich Animation findet sich so eine weitere Nutzungsmöglichkeit für die 3D-Modellierung. Die Kurven sollen die Bearbeitung einer Animation, wie zum Beispiel schneller oder langsamer werden, automatisieren und somit realistischere Animationen hervorbringen(**Keyframe Interpolation**), aber auch für Bewegungen bzw. Wegvorgaben werden sie verwendet. [41]

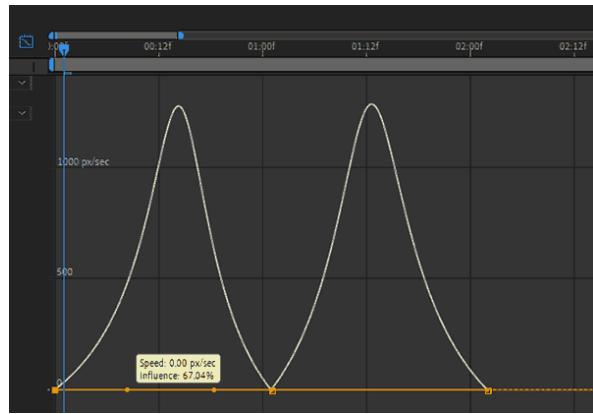


Abbildung 4.18: Keyframe Interpolation in Adobe After Effects

- **NURBS:** Non-Uniform Rational B-Splines beschreiben mathematische Kurven und Flächen. Diese kann jeden nicht verzweigten Linienzug darstellen, wozu auch Formen wie Kreise zählen. Die Bearbeitung von NURBS stellt sich dabei als sehr intuitiv und vorhersehbar heraus. Die Kontrollpunkte sind stets mit der Kurve oder Fläche verbunden. Der Unterschied zur Bézierkurve ist, dass NURBS eine Weiterentwicklung darstellen, da durch die Bézierkurve die Darstellung bspw. eines Kreises nicht möglich ist. [54]

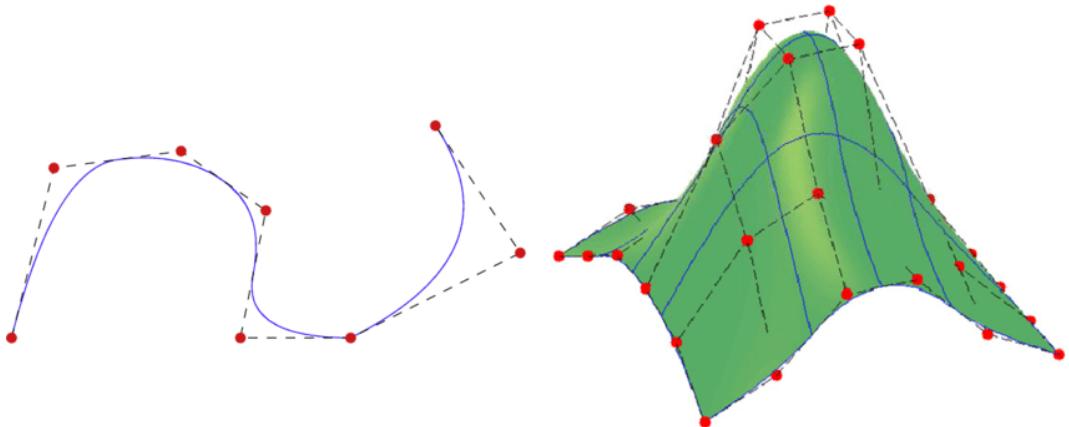


Abbildung 4.19: Darstellung einer NURBS als Kurve und als Fläche

Neben der Modellierung mit Kurven hat sich ebenfalls die Modellierung mit Polygonen durchgesetzt. Der große Unterschied ist, dass bei den Polygonen nicht aus einer mathematischen Funktion das Objekt berechnet wird, sondern dieses bereits aus einem mathematisch beschriebenen Objekt besteht. Dieses Polygon besteht bereits aus einem Mesh und so bieten die Werkzeuge der Polygon-Modellierung die Manipulation dieses Meshes an. So kann dieses zugeschnitten, erweitert, abgerundet, abgespitzt und noch vieles mehr werden.[70]

**Box-Modeling** stellt die momentan einfachste und populärste Methode des Modellierens dar. Diese ähnelt etwas dem Prozess der Bildhauerei. Dabei beginnt man mit einem einfachen Primitiv (meist einem Würfel, deshalb auch Box) und richtet sich mit dieser eine Grundform ein. Von dieser wird das Mesh dann geschnitten beziehungsweise ausgedehnt, um zum gewünschten Ergebnis zu kommen. [86]

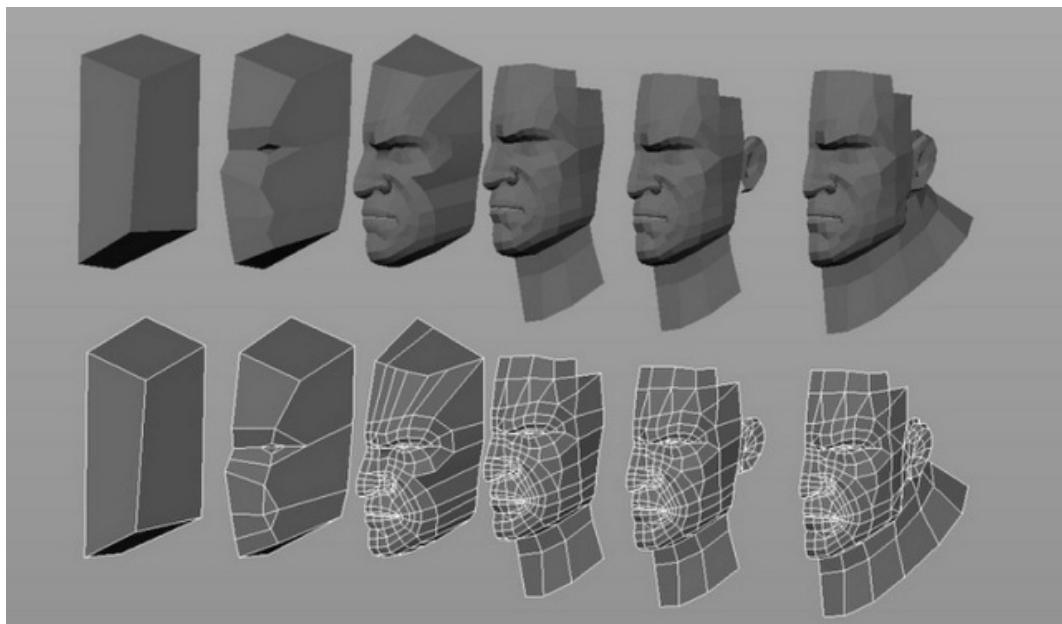


Abbildung 4.20: Modellierung einer Gesichtshälfte mittels Box-Modeling

Diese Art der Modellierung ist am besten für feste Gegenstände geeignet. Dazu zählen beispielsweise ein Fenster, ein Tisch, Möbel und gesamte Gebäude. Es ist ebenfalls möglich Lebewesen zu erstellen, allerdings könnte dies dann beim Animieren zu Problemen führen, wenn man nicht auf die **Topologie** aufpasst.

Die Topologie beschreibt den Aufbau des Meshes bzw. die Verbindungen zwischen den einzelnen Punkten des Körpers und wird somit dann auch bei der Animation des Meshes wichtig, wenn man dieses manipulieren will. Dabei muss man auch beachten, dass die Anzahl der Punkte nicht unbedingt die Qualität bzw. die Details eines Meshes beschreiben. Vielmehr ist es wichtig die Topologie so genau wie möglich zu machen, besonders dann, wenn man plant eine Animation einzubauen. Generell ist es empfohlen immer so zu modellieren, als ob man das Objekt später noch manipulieren könnte.

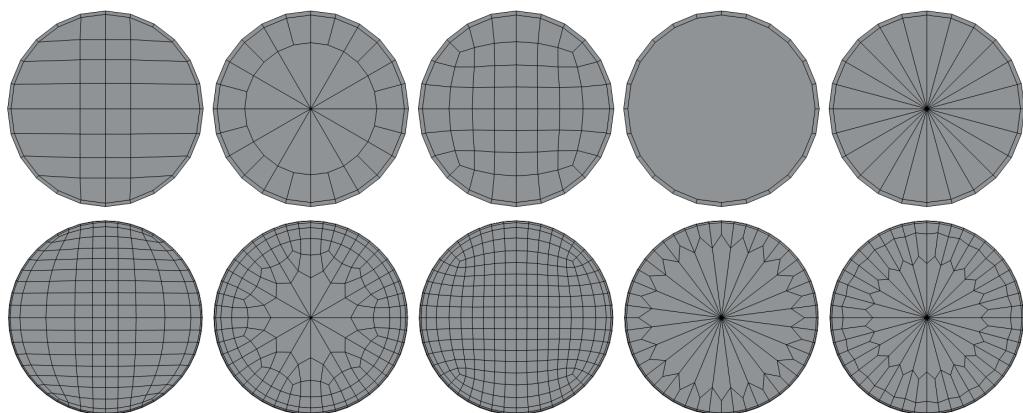


Abbildung 4.21: Topologien eines Kreises (z.B. Seite eines Zylinders, Kugel von der Seite)

Ist die Topologie nicht ganz sauber, kann es dann bei der Manipulation zu ungewollten Problemen führen, was meist nicht gerade wenig Arbeit ist, wieder in Ordnung zu bringen. Eine der wohl besten Möglichkeiten die Topologie zu überprüfen, ist zu schauen ob die einzelnen **faces** entlang der geplanten Animation liegen. Faces sind die einzelnen Flächen zwischen den Punkten. Die Topologie ist überhaupt wichtig, wenn man mit Polygonen arbeitet, somit auch bei der Poly-Modellierung. Mittlerweile bieten die meisten Programme auch schon ein eigenes Tool für die Topologie an.

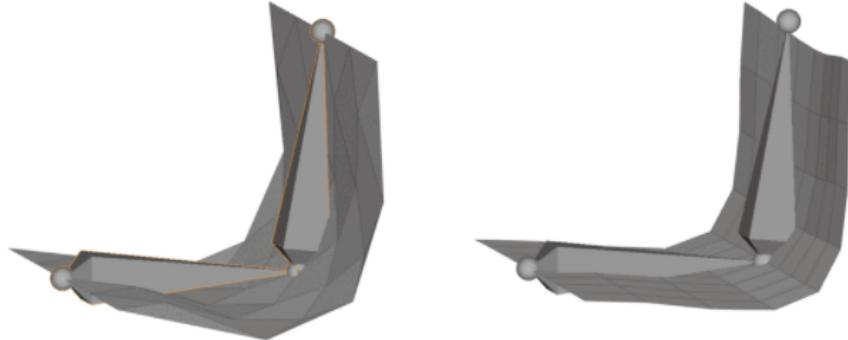


Abbildung 4.22: Links:falsche Topologie, Rechts: richtige Topologie

Das **Poly-Modeling** ist zwar ebenfalls eine Methode, die Polygone benutzt, sich aber sehr im **Workflow** zum Box-Modeling unterscheidet. Es bietet zwar nicht denselben leichten Einstieg an, aber dafür sehr hohe Genauigkeit und Effektivität. Ebenfalls ist es einfacher, die Topologie des fertigen Meshes zu kontrollieren, da man mit diesem arbeiten muss, um das Objekt zu erstellen. Diese Methode erweist sich ganz besonders bei organischen Modellen (Charaktere und Lebewesen) als hilfsbereit. Eine besondere Hilfestellung dieser Variante ist es Bilder von dem Objekt oder Concept Arts aus zumindest 2 Perspektiven (Front und Seitlich) zu nutzen, damit das Modell sozusagen nur noch abgezeichnet "werden muss und man immer die Vorlage vor Augen hat.<sup>[86, 87]</sup>

Den Prozess beginnt man meist mit einem **Quad**, was eine einfache 2-dimensionale Fläche im 3D Raum darstellt(Step 1).

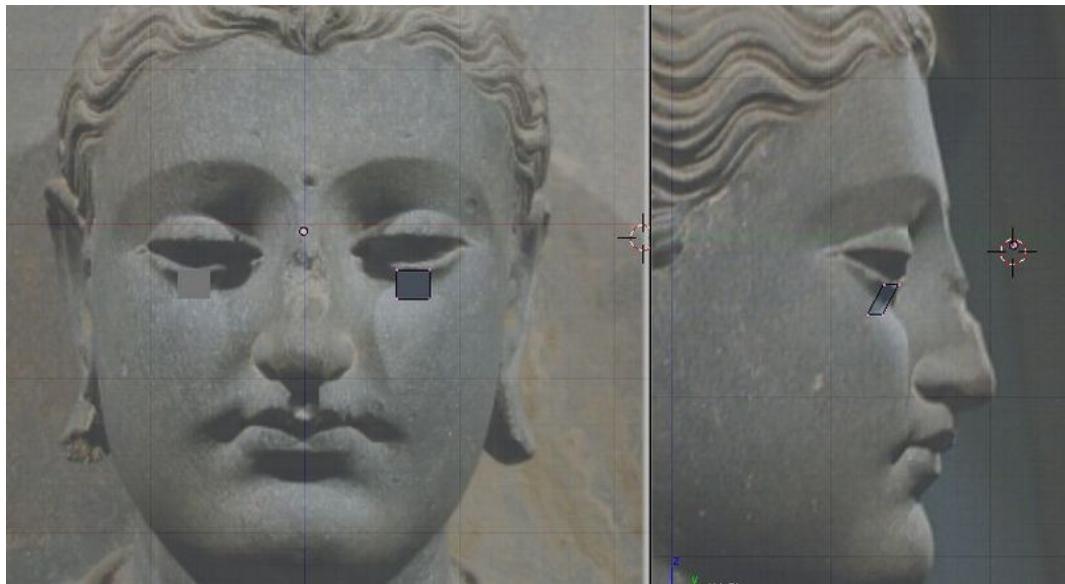


Abbildung 4.23: Step 1: Aller Anfang ist ein Quad

Von diesem Quad geht man dann Punkt zu Punkt und Kante zu Kante weiter und baut so die Struktur des Modells. Besonders für den Anfang sollte man sich erstmal auf die groben Konturen des Modells konzentrieren und diese mit mit vorerst noch eckigen Umrandungen darzustellen.

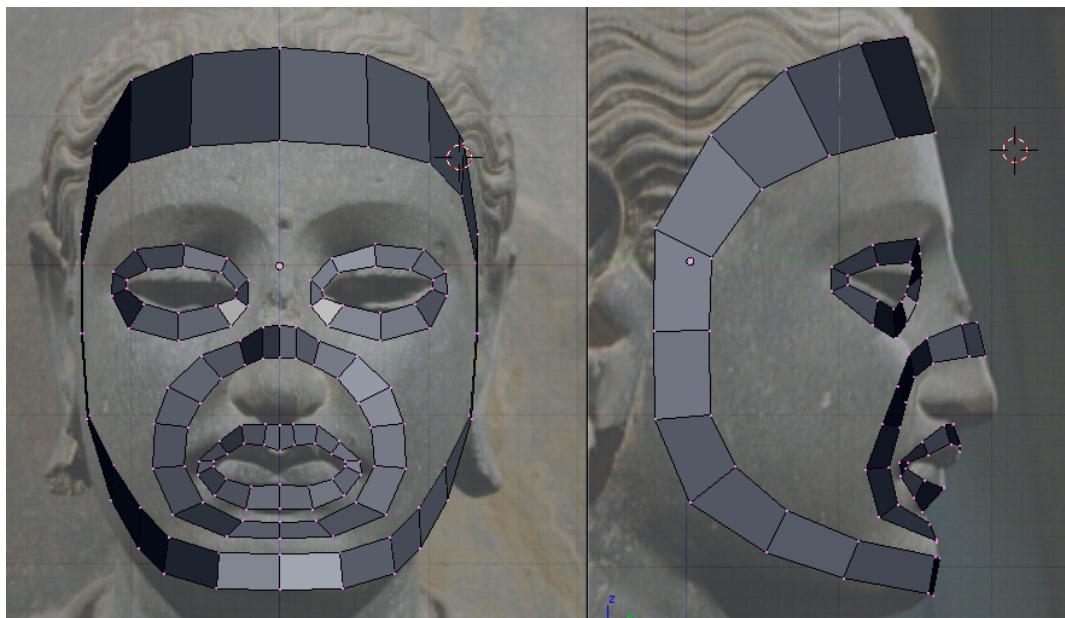


Abbildung 4.24: Step 2: Grobe Konturen mit weitern großen Quads

Im weitern Verlauf werden dann die Details weiter angefügt und die restlichen Lücken geschlossen. Hat man den fertigen Grundriss des Modells, kann man das Mesh noch weiter anpassen um etwaige Verschönerungen noch anzubringen.



Abbildung 4.25: Step 3: Einfügen der Details. Quads werden immer kleiner

Was bei allen dieser Modellierungsmethoden wichtig ist, ist dass man das Objekt auch **unterteilt**. Besonders im Hinblick auf Animation, muss man sich überlegen, welcher Teil des Objektes, sich wie verhalten soll. Bei einer Spritze beispielsweise soll sich der Kolben unabhängig vom Zylinder bewegen. Dazu lässt es sich im realen Leben abnehmen. Somit vereinfacht es den Modellierungsprozess sowie den Animationsprozess, wenn die Spritze nicht aus einem einzigen Objekt besteht, sondern aus einzelnen Teilen. Besonders beim Benutzen dieser Objekte außerhalb der Modellierungssoftware, muss man für ein sehr dynamisches Verhalten auf dieses Detail aufpassen. Diese lassen sich im Endeffekt auch **gruppieren**, um es als ein Objekt zu sehen.

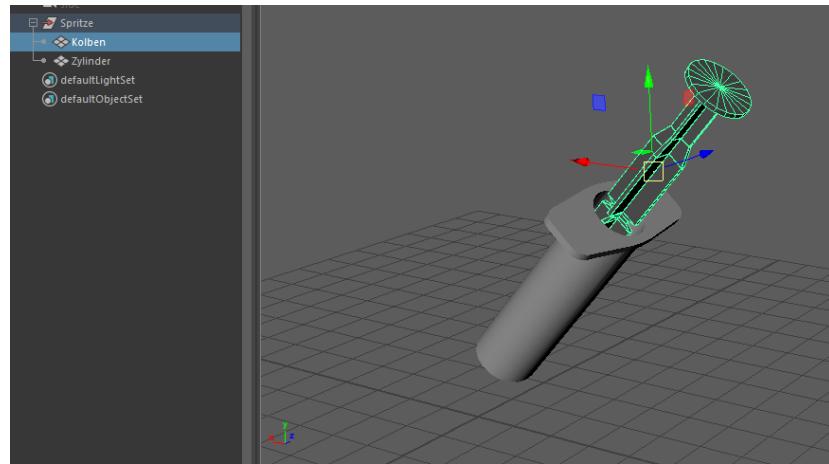


Abbildung 4.26: Modell einer fertigen Spritze. Aufgeteilt in Kolben und Zylinder

Ein weiterer wichtiger Teil der Polygon-Modellierung ist das **Sculpting**, welches besonders beim Design von Charakteren und Lebewesen oder aber auch anderweitige Skulpturen, welche ein hoch komplexes Mesh haben. Durch Sculpting wird es möglich größere Details einzubauen, welche mit den normalen Polygon-Tools für die Umsetzung mehr Komplexität erfordern.



Abbildung 4.27: Sculpting Prozess eines Gesichts in ZBrush

#### 4.4.2.2 Modellierungssoftware

Software zum 3D-Modellieren gibt es eine Vielzahl, egal ob nun kostenpflichtig oder kostenlos. Autodesk ist beispielsweise einer der führenden Hersteller von 3D-Modellierungssoftwaren. Diese bieten verschiedene Produkte in 3 Sparten an [4]:

- Spiele, Film und VFX (Bsp.: Maya und 3DS MAX)
- Industriedesign und Produktentwicklung (Bsp.: Inventor und Fusion 360)
- Architektur und Bauwesen (Bsp.: AUTOCAD und Revit)

Aufgrund des Anwendungszwecks, werden hier nur die für den Bereich Spiele und VFX angesehen.

**Autodesk Maya** **Maya** ist die mitunter bekannteste Software im Bereich der 3D-Modellierung. So kommt sie in den berühmtesten Animationsstudios weltweit (bspw. Disney Pixar) zum Einsatz und hat sich mit ihren Tools einen Namen gemacht. Mit einem Preis von 274 € / Monat bzw. 2.208 € / jährlich ist es aber auch eines der teuersten Arbeitsumgebungen. Maya umfasst zahlreiche Tools, von der einfachen Polygonmanipulation und NURBS-Technologien bis hin zu komplexen Physik- und Animationstools, um die Modelle so realistisch, wie möglich darstellen zu können. Dazu kommen sie mittlerweile auch schon mit eigenen Export-Möglichkeiten für die Game-Engines Unity und Unreal Engine, um den Workflow noch einfacher zu gestalten.[89]

**Blender** Das wohl beste und bekannteste, kostenlose Gegenstück zu Maya ist **Blender**. Neben Maya hat sich auch Blender mittlerweile bei großen Projekten bewährt und gibt mit jeder Version eine Vielzahl an Tools hinzu, welche den Workflow des Modellierers verbessern soll. Neben den Modellierungs- und Simulationsumgebungen haben die Entwickler einige weitere Phasen des Modellierprozesses in ihre Software gegeben. So kann man mit dem Grease Pencil bereits die notwendigen Skizzen oder Story-boards in Blender realisieren aber auch mit dem integrierten Videoschnittprogramm eine fertige Animation gleich in Blender bearbeiten.[90]

**Autodesk 3DS MAX** 3DS MAX ist ebenfalls von Autodesk und kann auch für die Modellierung für Spiele und Film eingesetzt werden. Durch die Wiederverwendung einiger Tools, welche auch Maya verwendet, wirkt das Programm auch sehr ähnlich. Tatsächlich ist der Workflow aber eher für die schnelle Umsetzung komplexer Szenen und Simulationen besser geeignet, als für den tatsächlichen Modellierungsprozess.[88]

**ZBrush** ZBrush stellt im Vergleich zu Maya und Blender ein Sculpting-Tool dar und ist somit eher für diesen Prozess geeignet. Die meisten 3D-Modellierer benutzen für ihren Workflow Maya oder Blender, um das grundlegende Mesh zu erstellen. Während Maya und Blender ebenfalls Sculpting-Tools anbieten, besitzt ZBrush noch einmal mehr Tools um den Prozess zu vereinfachen. ZBrush stellt sich ebenfalls als sehr hilfreich mit der Texturierung der Objekte dar.[95]

#### 4.4.3 Texturierung

Ein einfärbiges 3D-Modell reicht unserem Anwendungsfall - der Modellierung von medizinischen Arbeitsgeräten - allerdings nicht aus. Neben den Details im Mesh stellen auch die verwendeten Farben, Musterungen und andere Auffälligkeiten ein wichtiges Detail eines Modelles dar. Diese nennt man in der Fachsprache **Texturen**. Die grundlegende Idee einer Textur ist dabei, ein zweidimensionales Bild sozusagen über das Objekt zu "kleben". Für die Texturen gibt es einige Arten und Technologien diese umzusetzen. [26]

##### 4.4.3.1 Materials und Texturen

Allgemein unterscheidet man 2 Typen für die Texturierung:

- **Texturen** sind wie bereits gesagt Bilder, welche man dann über das Objekt klebt. Nehmen wir beispielsweise eine Ziegelwand, dann könnte man ein Bild einer realen Wand nehmen und dem Objekt zuweisen. Wir hätten zwar vorerst nur 2 Seiten, die richtig dargestellt werden, aber eine Ziegelwand. Mit Texturen kann man so Körnungen oder Farbwechsel darstellen. Dafür sind diese Texturen dann auch statisch und werden nicht wirklich vom Programm verändert.
- **Materials** definieren Attribute für die Darstellung eines Objektes. Zu diesen gehören Farbe, Reflexion und Durchsichtigkeit. Mit Materials ist möglich, dem Aussehen des Objektes eine gewisse Dynamik zu geben. So werden Lichter das Objekt mehr beeinflussen, durch Reflexionen. Das größte Problem an Materials ist die Kompatibilität zwischen den Programmen. So kann man Maya Materials nicht in Unity benutzen oder umgekehrt Unity Materials in Maya. In Unity werden die Texturen allerdings in Form eines Unity Materials an das Objekt gegeben, wodurch es auch möglich wird diese beiden Eigenschaften zu vereinen.

Wie vorhin erwähnt reicht jedoch nicht einfach nur ein Bild als Textur, es sei denn wir haben einen Würfel. Man muss das 3D-Modell irgendwie auf eine 2-dimensionale Fläche bekommen. Für diesen Anwendungszweck gibt es **UV-Mapping**.

#### 4.4.3.2 UV-Mapping

Eine UV-Map stellt ein 3-dimensionales Objekt im 2-dimensionalen Raum dar. U und V beschreiben dabei die horizontale und vertikale Achse der 2D Fläche, nachdem X, Y und Z bereits für das Modell genutzt werden. Der Prozess die UV-Map aus dem 3D-Modell herauszubekommen nennt man **UV-Unwrapping**. Maya sowie Blender unterstützen mittlerweile vorgefertigte UV-Mapping Tools, um die Map eines Objektes zu erstellen. Durch UV-Maps wird es möglich einem Objekt relativ einfach eine Textur zu geben und diese auch in anderen Plattformen einsetzen zu können. [25]

Während Materials Softwareabhängig sind, kann man Texturen auf unterschiedlichste Art herstellen und dann auch in jeder Applikation ohne Probleme nutzen.

Die erste Möglichkeit ist das Wiederverwenden von Assets. Es gibt eine große Menge an Internetseiten, welche kostenlose Texturen anbieten, was sich besonders bei monotonen Texturen, wie Papier oder Stoff anbietet, was mehr auf die Textur selbst Wert legt, als die Musterung oder Färbung.

Maya und Blender bieten beide ein Painting Tool an, was man genau für diesen Zweck verwenden kann. Besonders bei Farben und Musterungen bietet sich dieses Tool an, da man einfach auf dem 3D-Modell zeichnen kann, und die UV-Map mit diesem exportieren kann.

Neben diesen 2 Möglichkeiten, gibt es auch die Möglichkeit externe Software für diesen Anwendungszweck zu verwenden. Die UV-Maps lassen sich in Bildbearbeitungs und Designsoftwares anpassen. Beispiele für diese sind Adobe Photoshop und Illustrator bzw. als kostenlose Alternative GIMP und InkScape. Photoshop bietet mit der Substance Collection (Substance Painter und Designer) eine Möglichkeit an, über das 3D-Modell Texturen für dieses zu erstellen. Auch ZBrush hat sich bei der Texturierung, besonders im Cartoon-Stil sehr bewährt. [26]

#### 4.4.4 Verhalten: Rigging, Animation und Physik

In Spielen ist es wichtig, dass die Objekte, mit denen man interagiert, auch ein gewisses Verhalten aufweist. Dabei können es einfache Physik-Simulationen sein oder aber auch komplexe Animationen oder natürliche Bewegung durch Rigs.

##### 4.4.4.1 Key-Frame Animation

Die wohl einfachste Art der Animation ist die **Key-Frame Animation**. Bei dieser Animationsart werden sogenannte **Key-Frames** - also besondere Zeitstempel - gesetzt, in welchen man die wichtigsten Attribute der Animation setzt. Meist handelt es sich dabei um die Position, Rotation und Skalierung, wobei in Maya auch Deformer Attribute besitzen, welche mit dieser Animationsart animiert werden können.

Durch die Änderung dieser Attribute über die Key-Frames, berechnet sich das Programm dann die Änderungen für jedes Bild. In einem 30 Sekunden Clip könnte man alle 5 Bilder einen Key-Frame setzen, in welchen man die Änderungen macht. [13]

##### 4.4.4.2 Rigging

Ein **Rig** ist wie ein Skelett, welches man dem Mesh zuweist. Ein Rig besteht aus mehreren **Bones**, die das Skelett ausmachen. Diese haben eine Parent-Child-Beziehung, was bedeutet, dass es einen Wurzel-Bone gibt, von welchem sich diese dann wie ein Baum verzweigt. Durch diese

Beziehung werden bei der Manipulation eines Bones alle Child-bones ebenfalls anhand des Parents manipuliert. Meist funktionieren diese Rigs nur mit Rotationen, da dadurch die Position des Childs beeinflusst wird.[\[3\]](#)

#### 4.4.4.3 Physik

Die Möglichkeiten von **Physik-Simulationen** sind stark abhängig von der Software. So sind die **Maya-Physics** eher wie Key-Frame Animationen, welche der Computer anhand der vorhandenen Objekte und deren Attribute berechnet. Ist eine Animation mal berechnet, wird sie immer auf dieselbe Art abgespielt, auch wenn in der Berechnung ein gewisser Zufall eingebaut ist. Dazu lassen sich diese Physiks auch nur als Animationen exportieren, weshalb diese im Endeffekt sich nicht mehr dynamisch an die Spielszene anpassen lassen. [\[5, 6\]](#)

Durch die **Unity-Physik** lassen sich diese Physiken, dann während der Laufzeit benutzen, wodurch auch unerwartete Ereignisse passieren können, was manchmal mehr, manchmal weniger erwünscht ist. Dabei muss man allerdings sehr auf die Bauweise der Szene achten, nachdem die Maßeinheiten in Unity relativ zu der realen Welt ist. So zeigt das Physiksystem oftmals Probleme bei kleiner Skalierung der Szene. In Maya gehören aber auch Cloth und Soft Bodies zu der Physik der Software, wobei hier dann dasselbe gilt, wie vorhin erwähnt: Die Physik ist mehr ein Animationsclip, der einfach nur abgespielt wird. [\[78\]](#)

Ansonsten müssen Physiken in Unity auch selber umgesetzt werden bzw. man holt sich die Assets aus dem Unity Asset Store, wo es sowohl kostenpflichtige sowie kostenlose Assets gibt. [\[79\]](#)

#### 4.4.5 Individuelle Aufgabenstellung

Nachdem nun unterschiedlichste Methoden ein 3D-Modell zu erstellen, animieren und texturieren, stellt sich nun die Frage, welche Rolle diese in dem Projekt spielt. Die 3D-Modellierung hat folgende Aufgaben in diesem Projekt:

- **Detailgetreue Darstellung medizinischer Geräte:** Nachdem die VR-Applikation dafür da ist, die wichtigsten Schritte des Tracheostomawechsels mitzunehmen, sind auch die dazu benötigten Gegenstände, von großer Bedeutung. Diese sollen für den User einen Wiedererkennungswert haben, weshalb sie sehr an das reale Original angelehnt sein sollten, insbesondere in Form und Farbe.
- **Animation der Gegenstände und Hände:** Nachdem Applikation interaktiv sein soll, sollten die meisten Gegenstände auch etwas tun, wenn man mit diesen interagiert. Dabei muss herausgefunden werden, wie sich diese Objekte in der Realität verhalten und zu welchem Umfang man dies im virtuellen Raum umsetzen kann.
- **Zusammenstellung der VR-Szene:** Nachdem man in einer VR-Simulation meist mit sehr wenig Platz limitiert ist, muss auch die Szene auch dementsprechend vorbereitet werden. So dürfen die Gegenstände, als auch der Patient nicht zu weit weg sein. Genauso wenig sollte der Spieler aber in der Lage sein, Grenzen der Szene zu umgehen. Ebenfalls sollte die Szene an den Anwendungsfall orientiert werden und daher in Stil eines Krankenzimmers aufgebaut werden.



# Kapitel 5

## Konzept

### 5.1 Konzept: Logik und Interaktivität einer Schulungs- und VR-App

#### 5.1.1 Ausgangslage

Bevor sich ein Konzept überlegt werden kann, muss vorerst definiert werden, was im Projekt zu machen ist. Das Projekt "Medizinisches VR Training" nimmt sich als Ziel, den Prozess eines Tracheostomawechsels als VR-Simulators abzubilden. Der Tracheostomawchsel beschreibt ein medizinischer Vorgang, in welchem man eine Kanüle aus dem Hals des Patienten austauschen muss. Hier soll ein linearer Ablauf an Aufgaben (Tasks) vom Endbenutzer durchgeführt werden. Das Projekt nutzt die neuartige Technologie VR, um sich abzugrenzen und eine interaktive Erfahrung zu ermöglichen. Dieses Kapitel beschäftigt sich damit, wie die Logik-Komponente gestalten werden kann, um den Prozess des Tracheostomawechsels erfolgreich abbilden zu können.

#### Begriffserklärung

Der Tracheostomawchsel beinhaltet zwei Vorgänge, das Herausnehmen der Kanüle und das Hineingeben der Kanüle. Die Vorgänge haben wir als einzelne Prozesse definiert, zwischen welchen man wechseln können soll. Im Programm soll der User einen der zwei Prozesse durcharbeiten, indem dieser definierte Aufgaben löst. Diese Aufgaben haben wir Tasks genannt und sie stellen den größten Teilbereich der Logik dar. Eine Auflistung der Tasks ist im Unterkapitel "Tasks" vorhanden. Damit Tasks gelöst werden können, müssen bestimmte Werkzeuge, wie zum Beispiel die Kanüle verwendet werden. Diese Werkzeuge haben wir im Projekt Tools genannt.

#### 5.1.2 Programmsteuerung

##### 5.1.2.1 Konzept

Um den Ablauf der Tasks zu steuern und somit eine gute Simulation aufbauen zu können, ist es sinnvoll Klassen zu erstellen, die sich jeweils abgekapselt um die Durchführung ihrer eigenen Aufgaben kümmern.

#### Entwurf

Folgendes Diagramm stellt eine mögliche Klassenstruktur dar:

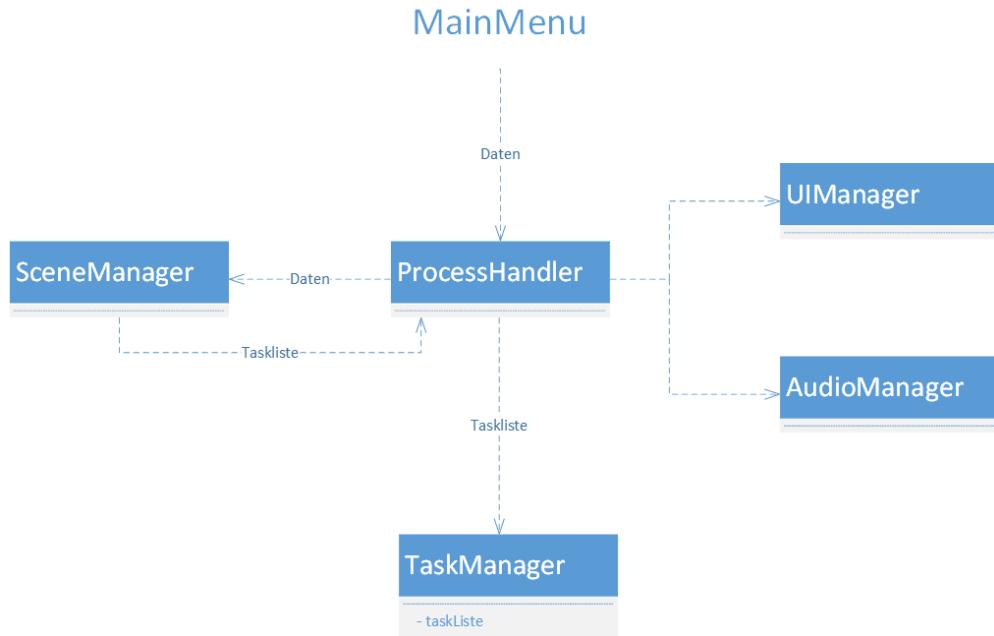


Abbildung 5.1: Konzept einer Programmstruktur

### 5.1.2.2 Process-Handler

#### Aufgabe:

Damit verschiedene Teilbereiche miteinander kommunizieren können und die Klassen so isoliert wie möglich sind, wird eine Klasse, welche als Schnittstelle dient. Diese Klasse ist die ProcessHandler-Klasse und dient als Schnittstelle für die Kommunikation zu den einzelnen Manager-Klassen.

#### Funktionsweise

Um den Prozess zu starten, erhält der ProcessHandler die für ihn benötigten Daten in einem jeweils definierten Format. Diese Daten werden an den Scene-Manager weitergegeben werden und die von ihm verarbeiteten Daten werden zurückgegeben, um diese an den TaskManager weiterzuleiten. Als relevante Daten wurden diese beschlossen:

- **ProcessID:** Hier wird spezifiziert, welcher Prozess gestartet werden soll, falls es mehrere gibt.
- **Tool-Liste:** Eine Liste von Tools, die für das Abschließen der Tasks benötigt wird.
- **Task-Liste:** Eine Liste an den jeweils abzuschließenden Tasks soll in beliebiger Form übergeben werden.

Wichtig ist es, dass der ProcessHandler global angesprochen werden kann, weshalb das Singleton-Pattern sinnvoll erscheint. Durch dieses Pattern kann garantiert werden, dass stets nur eine Instanz des ProcessHandlers existiert. Nachteilhaft an diesem Pattern ist, dass es schlechtes Design verstecken kann und globale Zugriffe die Sichtbarkeit des Codes reduzieren können. [64] Deshalb ist es wichtig, globale Instanzen nur selten und in sinnvollen Szenarien zu verwenden.

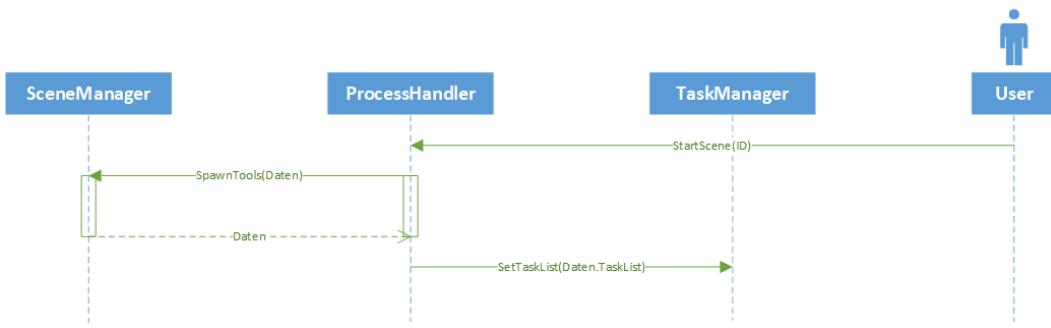


Abbildung 5.2: Sequenzdiagramm: Darstellung eines Programmstarts

### 5.1.2.3 Scene-Manager

#### Aufgabe:

Der SceneManager besitzt die Aufgabe, alle Vorbereitungen zu treffen, um den erfolgreichen Programmablauf zu erlauben. Er übernimmt vom ProcessHandler zur Initialisierung notwendige und verpackte Daten. Diese verarbeitet er und erhält Daten, wie zu instantiierende Tools und speichert diese in eine Datenstruktur ab. Zusätzliche erhält dieser auch noch eine Liste an Tasks. Die verarbeiteten Daten soll er zurück an den ProcessHandler weiterleiten.

#### Funktionsweise:

In dem Daten-Objekt enthaltene Tools sollen als **Prefab** abgespeichert werden, woraufhin der SceneManager eine Instanz dieser an vorher definierte Orte platzieren soll. Eine sinnvolle Datenstruktur zur Speicherung der Referenzen dieser instantiierten Objekte wäre eine Liste, welche sich durch ihre dynamische Art von Arrays abgrenzt. Die Task-Liste wird entpackt und in eine Queue umgewandelt, da diese performanter[66] ist.

### 5.1.2.4 Task-Manager

#### Aufgabe:

Es muss eine Entität im Programm geben, welche für die Steuerung der Tasks verantwortlich ist. im Diagramm 5.1 ist diese als der TaskManager definiert worden. Neben dem Ablauen von Tasks soll er während der Ausführung Task-relevante Informationen an den ProcessHandler kommunizieren, damit er diese, wenn nötig weiterleiten kann.

#### Funktionsweise:

Die Task-Liste erhält der Task-Manager über den ProcessHandler und somit kann dieser den Task-Ablauf starten. Bei erfolgreicher Durchführung einer Task startet dieser die im Queue nächste Task. Informationen über die nächste Task, wie z.B. die Beschreibung, wird dem ProcessHandler übergeben. Der Task-Manager ist nicht für die Erstellung der notwendigen Bedingungen für das Vervollständigen des Tasks zuständig.

### 5.1.2.5 User-Interface Manager

#### Aufgabe:

Nun benötigen wir eine Klasse, die das UI auf die derzeitige Task reagieren lässt. Die definierten

Aufgaben inkludieren:

- **Anzeigen spezifischer HUD Elemente:** Falls HUD-Elemente für die Lösung von Tasks benötigt werden, übergibt der UI-Manager diese Information an angesprochene UI Elemente.
- **Task Liste:** Daten über die derzeitige Task sollen, an einem UI-Element dargestellt werden können.

#### Funktionsweise:

Der UI-Manager besitzt Verbindungen zu anzulegenden UI-Elementen und leitet vom ProcessHandler erhältene Informationen an diese weiter.

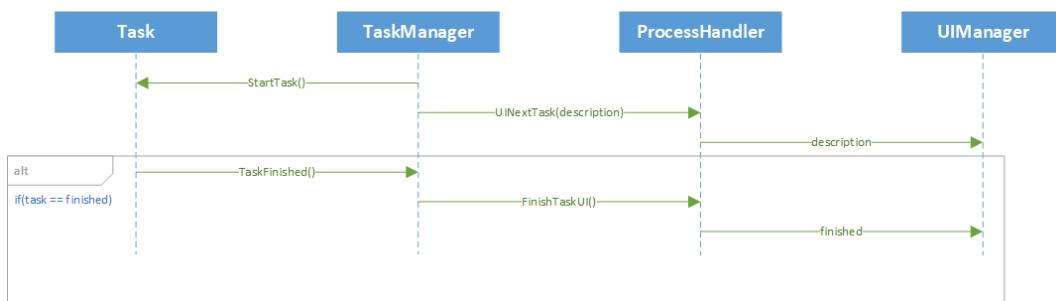


Abbildung 5.3: Sequenzdiagramm: Kommunikation zwischen Klassen bei Taskstart

#### 5.1.2.6 AudioManager

Der Audiomanager ist für das Abspielen von Sounds zuständig, seine Funktionsweise ist praktisch identisch zum UI-Manager.

### 5.1.3 Tasks

#### 5.1.3.1 Definition

Tasks beschreiben Aufgaben, welche vom Benutzer erfüllt werden müssen, damit dieser den Prozess beenden kann, ein Beispiel wäre das Platzieren der Kanüle in den Hals der Person. Alle Tasks sollen eine klare Endbedingung besitzen und auf diese reagieren können. Tasks sind somit einfach eine Darstellung einer Aufgabe in Form eines Unity GameObjects. [75]

#### 5.1.3.2 Konzept

Da eine Vielzahl an verschiedener Tasks definiert werden muss, ist es sinnvoll eine abstrakte Klasse zu definieren welche, notwendige Rahmenbedingungen für Tasks schafft.

#### Variablen

Jede Task benötigt bestimmte Informationen, um zu funktionieren. Während jede Task-Art andere Informationen benötigt, sind manche universell notwendig. Diese Variablen sind somit in die abstrakte Klasse zu inkludieren:

- **taskName:** Eine Variable, die den Namen der Task angeben soll.
- **description:** Soll wichtige Informationen über das Abschließen der Task beinhalten. Diese werden dem User über UI-Elemente angezeigt.
- **toolListe:** Eine Liste von instantiierten Tools, um auf diese direkt zugreifen zu können.

### Methoden

Auch manche universellen Methoden sind für das Inkludieren in die abstrakte Klasse sinnvoll. Diese Methoden sind:

- **StartTask():** StartTask() ist eine Methode, welche von jeder Task implementiert werden muss. Ihr Aufruf erfolgt durch den Task-Manager und signalisiert den Start einer Task. In dieser Methode werden somit alle notwendigen Vorbereitungen getroffen, damit die Task durchgeführt werden kann. Diese Methode als virtual zu markieren ist sinnvoll, damit erbende Klassen dynamisch damit arbeiten können.
- **FindTool(Text):** Eine Methode, welche anhand eines Texts ein Tool für eine jeweilige Task finden kann. Dafür durchsucht es die übergebene toolListe und ist somit performant, da er alternativ sonst die gesamte Szene nach dem Tool durchsuchen müsste.
- **FinishTask():** Wie StartTask() soll auch eine Methode definiert werden, welche am Ende der Task aufgerufen wird, z.B. um nicht benötigte Tools zu entfernen.

#### 5.1.3.3 Tasks

Für die Simulation des Tracheostomawechsels müssen viele verschiedenen Arten von Tasks erstellt werden. Das Ziel ist es, möglichst wenige Sub-Typen zu erstellen, indem man flexible Tasks hinzufügt, die für möglichst viele Aufgaben nutzbar sind. Um diese definieren zu können, ist eine Schritt-Liste für ein Tracheostomawchsel notwendig. Nachdem Analysieren des Vorgangs wurden folgende notwendigen Schritte/Tasks festgelegt:

##### Kanüle herausnehmen:

- **Feuchte Nase abnehmen:** Die feuchte Nase (auch HME genannt), die auf der Kanüle darauft steckt, soll entfernt werden.
- **Luft aus der Cuffline entfernen:** Die Cuffline soll mit der Spritze verbunden werden, wonach man die Luft aus dem Ballon mit der Spritze entfernen soll.
- **Band öffnen:** Das Band, welches die Kanüle an den Hals der Person befestigt, soll entfernt werden.
- **Kanüle herausnehmen:** Die Kanüle soll aus dem Hals genommen werden.
- **Müll entsorgen:** Der entstandene Müll soll entsorgt werden.

**Kanuele reingeben:**

- **Cuff mit der Spritze kontrollieren:** Der Cuff soll mit der Spritze verbunden werden und auf Fehler getestet werden (durch Auf und Abpumpen des Ballons).
- **Bänder befestigen:** Die Bänder sollen links und rechts von der Kanüle befestigt werden.
- **Trachealkompresse befestigen:** Die Trachealkompresse soll an der Kanüle befestigt werden.
- **Kanüle mit Gleitgel einschmieren:** Die Kanüle soll mit Gleitgel eingeschmiert werden.
- **Kanüle befestigen:** Die Kanüle soll mit einer drehenden Bewegung in den Hals der Person befestigt werden.
- **Manometer mit Cuff verbinden:** Das Manometer soll mit dem Cuff verbunden werden.
- **Ballon aufpumpen:** Der Ballon soll nun mit dem Manometer aufgepumpt werden.
- **Feuchte Nase aufsetzen:** Die feuchte Nase soll auf die Kanüle platziert werden.

**5.1.3.4 Task-Arten**

Da nun alle Tasks definiert wurden, können wir diese gruppieren, um Task-Arten definieren zu können. Um jede Task erstellen zu können, benötigen wir folgende Task-Arten:

**ConnectionTask**

Die meisten Aufgaben beinhalten das Verbinden von verschiedenen Objekten, weshalb eine dafür dedizierte Task-Art notwendig ist. Sie ist dafür zuständig, dass zwei angegebene Objekte sich verbinden können und die Task bei Verbindung sich abschließt.

**PressTask**

Objekte wie die Spritze und das Manometer benötigen die Betätigung eines Knopfes, um die Task zu beenden. Durch eine PressTask soll das logisch ermöglicht werden.

**Remove Task**

Besonders im "Kanuele herausnehmen"Workflow ist es notwendig Objekte zu entfernen. Dieser Vorgang benötigt auch eine Task, die das Entfernen ermöglicht/bei Entfernung Task abschließt.

**Rotation Task**

Für das Befestigen der Kanüle ist eine besondere Art einer Connection-Task notwendig, die eine Verbindung nur erlaubt, wenn das Tool entsprechend rotiert wird.

**TouchTask**

Eine Task, die bei Berührung von zwei Objekten abgeschlossen wird, ist ebenfalls sinnvoll für Tasks wie "Kanüle mit Gleitgel beschmieren".

### 5.1.3.5 Events

Das Abschließen verschiedener Tasks soll über Events geschehen, welche involvierte Objekte bei z.B. Berührung senden. Diese empfängt der Task-Manager und leitet es der Task weiter. Durch dieses System sollen Tasks dynamisch auf Events reagieren können.

### 5.1.3.6 Entwurf

In einem UML-Klassendiagramm sieht unsere Task-Struktur folgendermaßen aus:

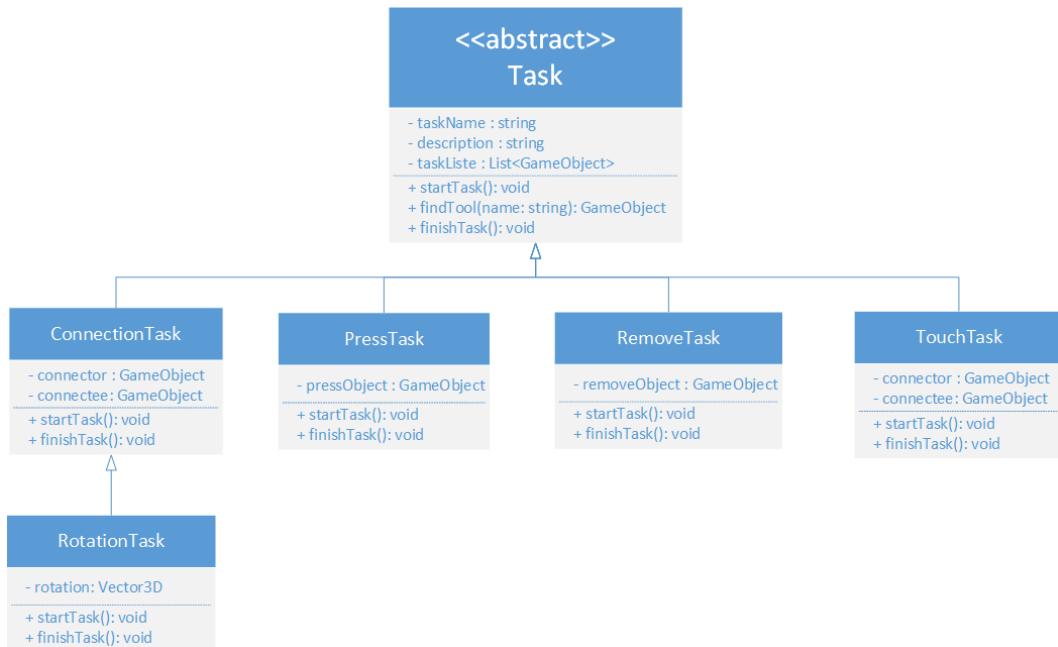


Abbildung 5.4: Klassendiagramm: Vererbungshierarchie der Tasks

## 5.1.4 Erweiterbare Entwicklung

### 5.1.4.1 Datenformat

Wie in den verschiedenen Diagrammen 5.1, 5.2, 5.3, 5.4 gesehen werden kann, benötigen fast alle Komponenten des Prozesses Daten. Bei der Überlegung des Formats der Daten ist eine simple und leicht zu erweiterndes Format sinnvoll. Auch für unerfahrene Nutzer soll das System leicht veränderbar und anpassbar sein. Durch das Verwenden des Drag-and-drop Features im Inspektor Fensters und einer klar definierten Struktur kann das leicht umgesetzt werden. Als Daten-Objekt eignet sich somit die Verwendung eines Scriptable Objects, welcher eine Task und ToolList für einen beliebigen Prozess übernimmt. Beispiel:

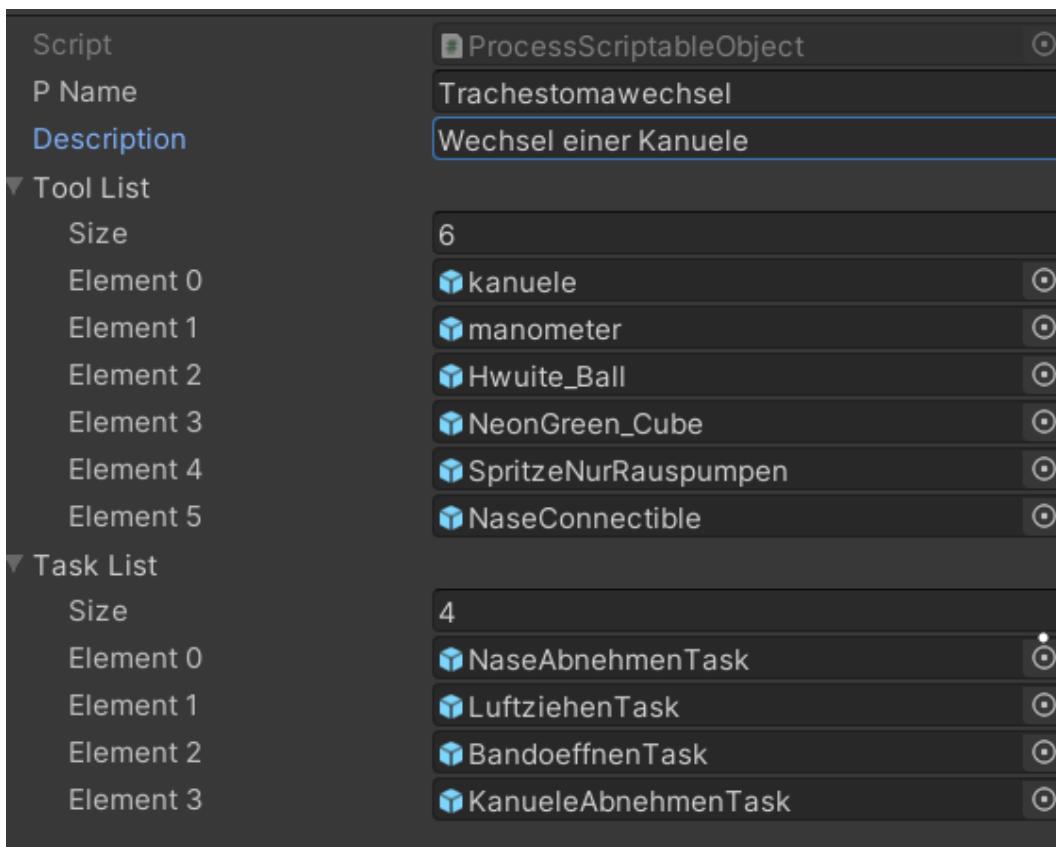


Abbildung 5.5: Unity Inspektor Fenster mit definierter Struktur

#### 5.1.4.2 Tools & Tasks

Tools und Tasks sollen wie oben im Bild angegeben in Form eines Prefabs erstellt werden. Durch das können diese problemlos wiederverwendet und erweitert werden.

## 5.2 Konzept für die Fortbewegung und Interaktion mit der virtuellen Welt

### 5.2.1 Entwicklungsumgebung

**VR-Headset** Bei der Auswahl der Hardware war die bereits verfügbare Hardware entscheidend. Der Auftraggeber KWP hat uns eine **Oculus Quest 2** VR-Brille zur Verfügung gestellt. Diese Brille wird für die Entwicklung und auch für das Testen verwendet. Um die Applikation auf dem Headset zu laufen, um es zu testen, muss man die Brille per USB-C Kabel mit dem Rechner verbinden und **Oculus Link** installiert haben. Im Menü innerhalb der Brille kann man sich dann per Oculus Link mit dem Gerät verbinden.

**Engine** Die Auswahl der Engine ist die wichtigste Entscheidung vor der Entwicklung eines Projektes. Ein Umsteigen auf eine andere Spiel-Engine während der Entwicklung kostet viel, dauert lange und wird deswegen nicht empfohlen. Für das Projekt wird eine robuste Engine benötigt, aber auch eine, mit der sich die Projektmitglieder auskennen. Um die richtige Entscheidung zu treffen, wird eine **Nutzwertanalyse** erstellt. Folgende Eigenschaften sind für die Analyse relevant:

- **XR-Support** - bewertet wie gut die out-of-the-box VR-Unterstützung ist (ohne zusätzliche kostenpflichtige Frameworks)
- **Leistung** - kennzeichnet die Leistung der Engine während der Entwicklung, sowie der fertigen Software, in Bezug auf VR Entwicklung
- **Kosten** - bewertet die Lizenzvoraussetzungen der jeweiligen Spiele-Engine, in Bezug auf ein internes nicht-kommerzielles Produkt
- **Simplizität** - Einfachheit der Umsetzung, in Bezug auf Codezeilen, Anzahl an benötigte Entities usw.
- **Vorwissen** - stellt das Vorwissen des Teams gegenüber dem benötigten Wissen für die jeweilige Engine, das bedeutet vorherige Erfahrungen mit der Engine selbst, sowie die Programmiersprachen, Workflow usw.

### 5.2.1.1 Nutzwertanalyse

		Unreal Engine		Unity		CryEngine	
Kriterium	Gewichtung	Bewertung (ungewichtet)	Bewertung (gewichtet)	Bewertung (ungewichtet)	Bewertung (gewichtet)	Bewertung (ungewichtet)	Bewertung (gewichtet)
XR-Support	30%	7	2,1	9	2,7	5	1,5
Leistung	15%	10	1,5	7	1,05	8	1,2
Kosten	10%	9	0,9	10	1	9	0,9
Simplizität	20%	7	1,4	8	1,6	5	1
Vorwissen	25%	7	1,75	9	2,25	5	1,25
Summe	100%	40	7,65	43	8,6	32	5,85

Abbildung 5.6: Spiel-Engine Nutzwertanalyse

Die Nutzwertanalyse betrachtet die drei in der Studie besprochenen Spiel-Engines: Unreal Engine, Unity und CryEngine. Die Bewertung erfolgte auf einer Skala von 1 bis 10. Aus der Nutzwertanalyse hat sich ergeben, dass **Unity** für das Projekt am besten passt. Von höchster Bedeutung war hier das Vorwissen des Projektteams. Alle Mitglieder kennen sich bereits mit der Engine aus und haben bereits Erfahrung mit anderen Projekten in Unity. Ebenfalls wichtig ist der gute XR-Support von Unity, was bei der Umsetzung des Projektes von hoher Bedeutung sein wird.

### 5.2.2 Fortbewegung

Das Fortbewegungssystem ist für ein VR-Projekt höchst relevant. Aus den im Kapitel [Lösungen für Fortbewegung in VR](#) erwähnten Techniken kommen für dieses Projekt nur Room-scale based, Controller based und Teleportation based in Betracht. Alle drei wurden überlegt und ausführlich getestet, um die richtige Entscheidung zu treffen.

#### 5.2.2.1 Fortbewegungstechniken im Vergleich

**Controller based** Dieser Ansatz benutzt die Joysticks auf den VR-Controllern. Es ist wahrscheinlich die einfachste Methode zum Implementieren, da die Joystick-Inputs einfach an das **Input-System** von Unity angehängt werden können. Danach kann man mit denen wie mit einem gewöhnlichen Controller arbeiten. Der linke Joystick ist für das Bewegen in vier Richtungen zuständig und der rechte hat die Funktion, die Kamera rapid um 90 Grad nach links oder rechts zu drehen. Diese Methode bietet viele Möglichkeiten. Nach dem Testen hat sich leider ergeben, dass sich diese Fortbewegung sehr unnatürlich anfühlt und bei manchen fast Übelkeit hervorruft. Vor allem das Drehen mit dem rechten Joystick hat sich als übel und gar unbrauchbar ergeben.

**Teleportation based** Bei diesem Ansatz sieht der Benutzer eine Anzeige, wo er sich dann nach Drücken eines bestimmten Knopfes auf dem Controller hin teleportieren kann. Von der Umsetzung her ist das die schwierigere Variante; die Anzeige muss gut sichtbar sein und sich intuitiv verhalten, die Teleportation sollte nicht mit der UI-Interaktion kollidieren und es sollte der **Blink**-Effekt implementiert werden. Die Testungen haben ergeben, dass diese Fortbewegungsmethode angenehmer als Controller based ist, jedoch trotzdem nicht perfekt, da die Teleporta-

tion so abrupt ist. Der Benutzer kann sich oft verloren oder verwirrt fühlen. Hier sollte aber der Blink-Effekt aushelfen.

**Room-scale based** Hier muss gar keine Fortbewegung umgesetzt werden, da die ganze Applikation für einen limitierten Spielbereich gestaltet wird. Für die Bewegung in der VR-Welt werden ausschließlich die Bewegungen des Benutzers in der realen Welt betrachtet. Dies eliminiert natürlich den Aufwand des Implementierens eines vollständigen Fortbewegungssystems. Die Arbeit ist hier bei dem Gestalten der Anwendung. Alles, was der Benutzer benötigt, muss sich innerhalb eines sehr limitierten Bereiches befinden. Es sollten auch Probleme gelöst werden, die entstehen würden, wenn der Benutzer beispielsweise die virtuelle Szene verlassen würde oder mit der Geometrie kollidieren würde (**Head collisions**).

#### 5.2.2.2 Die Entscheidung und ihre Implikationen

Für das Projekt wurde **Room-scale based** als die bevorzugte Fortbewegungsmethode entschieden. Controller based führt zu Übelkeit und Teleportation based sprengt den Rahmen des Projektes. Es ist nicht nötig, dass sich der Benutzer groß während der Schulung bewegen kann. Die medizinische Schulung passiert auf einem Patienten, mit wenigen benötigten Tools. Das bedeutet, der benötigte Bereich ist nicht groß und statisch. Wenn man sich bei der Entwicklung dieser Bedingung bewusst ist, sollte es keine Probleme geben.

Es sollten Head collision implementiert werden, damit der Nutzer nicht innerhalb der virtuellen Geometrie schauen kann. Zusätzlich sollte es verhindert werden, dass der Nutzer durch die Wände geht und die Szene verlässt.

Anhand der Voraussetzung, dass alles was benötigt wird sich in Griffnähe des Nutzers befindet, muss die Szene entsprechend gestaltet werden. Die folgende Skizze zeigt das bevorzugte **Layout** des Raumes:

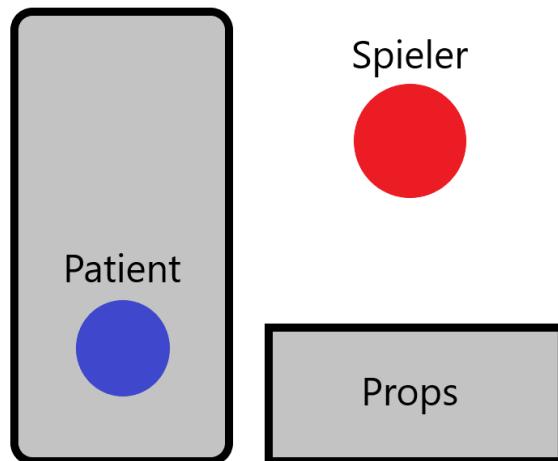


Abbildung 5.7: Skizze des Raum-Layouts

### 5.2.3 Interaktion mit Objekten

Die Interaktion mit Objekten ist der Grundbaustein für eine interaktive VR-App. Nur mit einem funktionsfähigen Interaktionssystem kann der Nutzer die virtuelle Welt beeinflussen und somit die Schulung durchführen. In Unity gibt es das **XR Interaction Toolkit**, welches in diesem Projekt eingesetzt wird.

#### 5.2.3.1 XR Interaction Toolkit

*„The XR Interaction Toolkit package is a high-level, component-based, interaction system. It provides a framework that makes 3D and UI interactions available from Unity input events. The core of this system is a set of base Interactor and Interactable components, and an Interaction Manager that ties these two types of components together. It also contains helper components that you can use to extend functionality for drawing visuals and hooking in your own interaction events.“ [85]*

Das Toolkit erstellt erstmals das **VR Rig** welches das VR-Headset in der Szene repräsentiert. In dem VR Rig befindet sich dann die Kamera. Das XR Interaction Toolkit bildet das Grundgerüst für die komplette Interaktion der Anwendung. Es bietet cross-platform Support für XR-Controller Inputs, haptisches Feedback bei den Controllern, einfache UI-Interaktion und vieles mehr. Das Implementieren von Greifen, Aufhaben und Werfen Mechaniken wird durch das Toolkit sehr einfach gemacht. Folgende Klassen aus dem Toolkit sind für das Projekt relevant:

- **XR Rig**
- **XR Controller** - repräsentiert die VR-Controller. Es werden zwei Instanzen dieser Klasse benötigt, eins für die linke und eins für die rechte Hand
- **XR Direct Interactor** - ermöglicht direkte Interaktion mit entsprechenden Objekten. Wird für das Grabbing benötigt
- **XR Ray Interactor** - neben dem Direct Interactor wird ein Ray Interactor benötigt, um mit UI Elementen zu interagieren (Canvas). Dieser Interactor arbeitet mit Raycasts, die auch visualisiert werden können
- **XR Grab Interactable** - Klasse für Objekte, die interaktiv sein sollen. Diese Objekte können dann mit anderen XR Elementen interagieren (z.B. mit den Händen aufgehoben werden)

#### 5.2.3.2 Interactable Object

Alle interaktiven Objekte in der App werden zu Interactable Objects. Sie müssen alle nach einem bestimmten Muster entwickelt werden. Es soll die Klasse **InteractableObject** erstellt werden, die alle interaktiven Objekte kennzeichnet und von der geerbt werden kann, um noch komplexe Objekte zu erstellen. Die Objekte sollten auch alle in der **Interactable** Layer sein. Folgende Struktur wird von interaktiven Objekten erwartet:

- **Rigidbody** - Element der Unity Engine. RigidBodies sind Physikobjekte, ermöglicht Transformation, Rotation und Einfluss von Gravität. RigidBodies können auch kinematisch gestellt werden, damit sie ihre Position nicht willkürlich ändern

- **Collider** - Collider sind unsichtbare Rahmen eines Objekts. Sie ermöglichen Kollisionen mit anderen Collidern. Wird meistens in Kombination mit RigidBody eingesetzt. Collider ist nur die Superklasse, es wird ein spezifischer Collider benötigt: z.B. **Capsule Collider**
- **XR Grab Interactable** - Teil des XR Interaction Toolkits, wird für Grabbing benötigt. Mit der Interaction Layer Mask kann man einstellen, mit welchen Objekten dieses Objekt interagieren soll. Eine leere Interaction Layer Mask bedeutet, dass dieses Objekt mit keinem anderen Objekt interagiert, und somit nicht mehr z.B. aufgehoben werden kann
- **InteractableObject** - eigene interne Klasse für alle interaktive Objekte. Kann vererbt werden, um komplexere Objekte zu entwickeln

**Movement Type** Ein **XR Grab Interactable** Objekt hat die Option eines Movement Types. Im Endeffekt bedeutet das, wie sich das Objekt verhält, wenn es gegriffen wird (also Bewegung während es in der Hand ist). Es gibt drei Möglichkeiten zur Auswahl: **Velocity Tracking**, **Kinematic** und **Instantaneous**. Bei Kinematic wird das gegriffene Objekt kinematisch gestellt, es kann zu ungewöhnlichem Verhalten führen, wenn dieses Objekt mit anderen frei liegenden Objekten kollidiert. Bei Instantaneous ist das Objekt nicht kinematisch, folgt aber der Position der Hand auf sofortige Weise, was bei abrupten Bewegungen der Hand krass aussehen kann. Bei Velocity Tracking kriegt das gegriffene Objekt die Beschleunigung der Hand mit, was zu scheinbar glatteren Bewegungen führt, während das Objekt auch gewöhnlich mit anderen frei liegenden Objekten interagieren kann. Velocity Tracking bringt auch den Vorteil, dass das Objekt geworfen werden kann, ohne zusätzlichen Code. Das passiert dadurch, weil beim Loslassen das Objekt weiterhin die Beschleunigung des Controllers behält. Für dieses Projekt haben wir uns für das Velocity Tracking Movement Type entschieden.

### 5.2.3.3 Compound Object und Connections

Es werden logische Zusammenfüge von Objekten benötigt. Zuerst braucht man ein **Compound Object**, mehrere Objekte, die ein einzelnes Objekt bilden, aus dem dann Teile entfernt werden können (z.B. feuchte Nase aus der Kanüle herausnehmen). Die einzelnen Teile sind kinematisch, solange sie nicht zu entfernen sind. Das Teil welches entfernt werden muss wird logisch aufgerufen und greifbar gemacht. Diesen Vorgang beginnt die entsprechende **Task**.

Für das Verbinden von Teilen in ein großes Objekt braucht man wiederum **Connections**. Es sollte möglich sein, dass mehrere Teilobjekte an ein Hauptobjekt angehängt werden. Dies wird mit **Anchorpoints** ausgeführt, die entsprechend angereiht sind, um die richtige Reihenfolge der Verbindungen sicherzustellen. Benötigt wird eine **ConnectorObject** und eine **Connectible** Klasse, die beiden von **InteractableObject** erben. Das Connectible wird vom Nutzer gegriffen, um auf das ConnectorObject zu verbinden, welches hier als ein „Zielobjekt“ dient. Das Freigeben der Anchorpoints und somit die Möglichkeit der Verbindung wird von den entsprechenden Tasks durchgeführt.

## 5.3 Konzept eines Graphical User Interfaces

Da es eine große Vielfalt von GUIs gibt, existieren auch mehrere Möglichkeiten eine zu implementieren. Im nächsten Kapitel werden die einzelnen Aufgaben des Interfaces im Bereich VR nochmals besprochen und die verschiedenen Möglichkeiten für das Umsetzen dieser Bereiche evaluiert.

### 5.3.1 Statusanzeige und weitere Darstellungen von Informationen

#### 5.3.1.1 Entwicklungsbereich

Wie in den vorherigen Kapiteln besprochen, werden wir die Game-Engine Unity für dieses Projekt benutzen. Somit befinden sich alle UI Elemente in einem "Canvas". Dieser ist sozusagen der Behälter für Widgets, durch die auch generelle Einstellungen gesetzt werden können. Eine der wichtigsten sind die Render-Einstellungen. Hier gibt es insgesamt drei:

- **Screen Space - Overlay:** UI-Elemente werden innerhalb des Canvas über alle anderen Elemente der Szene (UI und nicht UI Objekte) dargestellt. Diesen Modus ist mit VR Applikationen nicht kompatibel (es benötigt immer eine Kamera) und ist somit nicht relevant für unser Projekt.
- **Screen Space - Camera:** Ziemlich ähnlich zu Screen Space - Overlay. Der einzige Unterschied besteht darin, dass es sich auf eine gewisse Kamera bezieht (Sichtfeld des Benutzers) und es möglich ist, den Abstand zwischen Kamera und UI-Elementen einzustellen.
- **World Space:** Hier verhalten sich die UI Elemente wie ganz normale Objekte und überdecken keine anderen Elemente mehr [77].

#### 5.3.1.2 Konzept einer Statusanzeige

Als Nächstes müssen wir uns entscheiden, welche Art von Statusanzeige benutzt wird und welche Informationen überhaupt relevant sind. In Abbildung 5.8 werden zwei Ideen in Betracht gezogen:

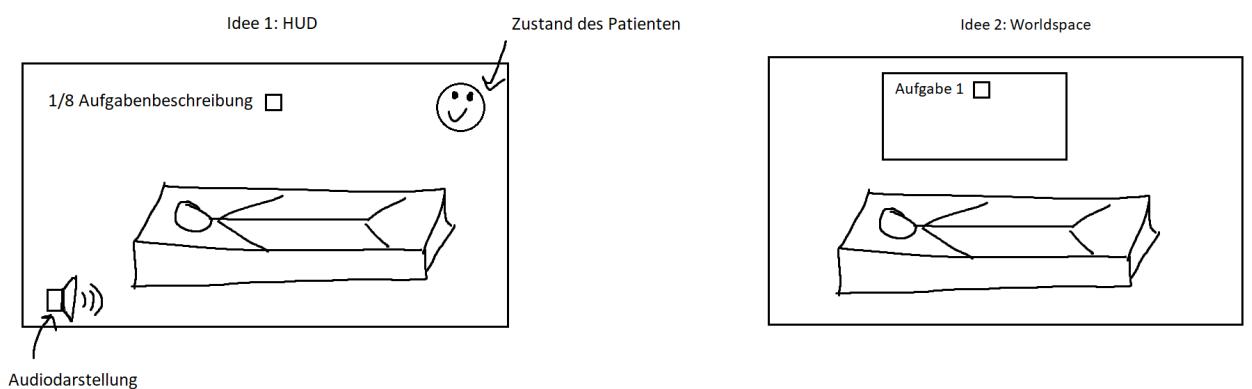


Abbildung 5.8: Ideen für Statusanzeige

Auf der linken Seite werden alle Informationen in einem HUD (in Unity: Screen Space - Camera) dargestellt. Links oben befindet sich die derzeitige Aufgabenstellung, links unten die Lautstärke und rechts oben der derzeitige Zustand des Patienten. Der Zustand würde sich abhängig von der Schnelligkeit und der Menge an Fehlern, die der Benutzer macht, verändern. In der anderen Skizze zeigen wir die Darstellung der Aufgaben im World Space auf einer Tafel. Die anderen Informationen würden im HUD-Bereich bleiben.

#### Idee 1: HUD

##### Vorteile:

- Schnellen Zugriff auf alle Informationen
- Ständig sichtbar
- Alle Informationen am selben Ort

##### Nachteile:

- Sichtfeld eingeschränkt
- Mögliche Überforderung des Benutzers
- überdeckt evtl. wichtige Teile der Szene

#### Idee 2: World Space

##### Vorteile:

- Sichtfeld wird nicht wie bei HUD eingeschränkt
- Mehr Platz für Aufgabenbeschreibung

##### Nachteile:

- Informationen aufgeteilt (mögliche Verwirrung des Benutzers)
- Kann übersehen werden

Wir haben uns dann schließlich für Idee 2 entschieden, da viel Sichtfeld benötigt wird und die Aufgabenbeschreibungen relativ lang sein können.

#### 5.3.2 Hauptmenü und Loading Screen

Als nächstes auf der Agenda ist das Hauptmenü mit Optionseinstellungen und der Loading Screen. Diese sind unabhängig von der Hauptszene und beinhalten nur UI-Elemente. Das Menü soll aus drei Buttons bestehen, namens "Start", "Optionen" und "Exit".

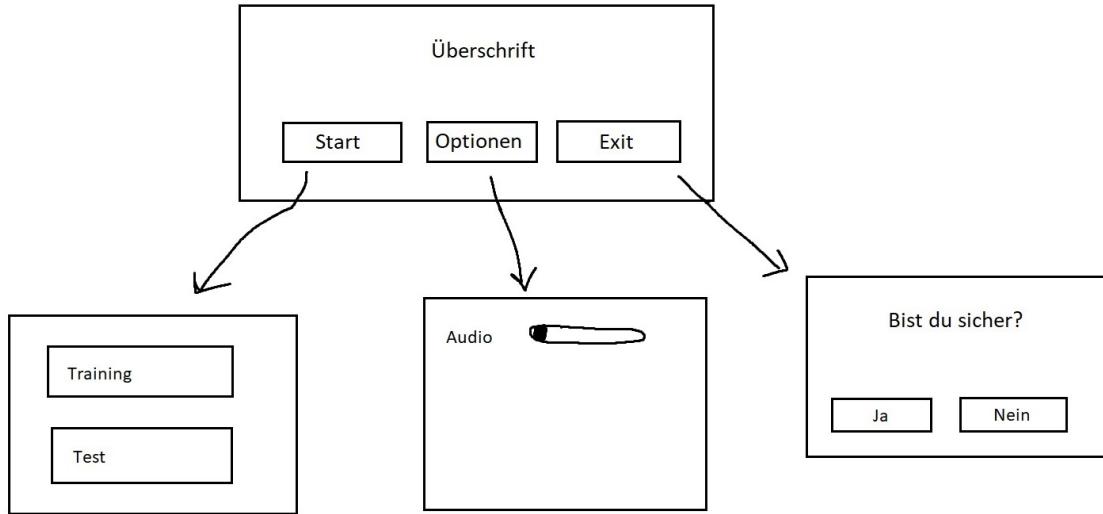


Abbildung 5.9: Funktionalität des Menüs

Jeder dieser Buttons verweist auf ein weiteres Fenster, in dem die genaueren Maßnahmen ausgewählt werden. Beim Start-Button erscheint die Auswahl zwischen Trainings-Modus und Test-Modus. Der Training-Modus zeigt alle Hinweise und Aufgabenstellungen an. Beim Test-Modus sieht der Benutzer nur noch die benötigten Werkzeuge. Damit ist das Training für Übungszwecke gedacht und mit dem Test wird das Wissen überprüft.

Da wir mit der funktionalen Idee des Menüs zufrieden waren, stand nur noch zu Diskussion, wie die Elemente in der Szene angeordnet werden und womit der Rest befüllt wird. Hierfür entwarfen wir wieder zwei Konzepte:

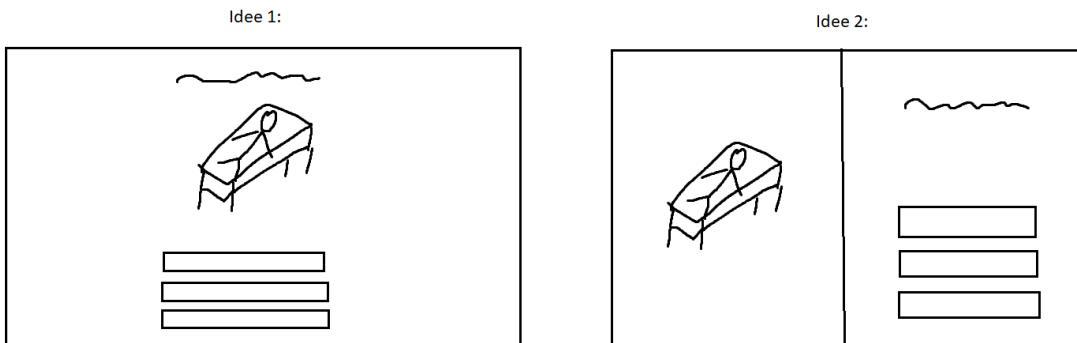


Abbildung 5.10: Darstellung des Menüs

Damit das Menü weniger leer wirkt, planten wir, das Bett und den Patienten schon im Menü darzustellen. Nun zur Anordnung: Bei Idee 1 zentrieren wir die gesamten Elemente und listen sie vertikal auf. Für dieses Konzept spricht die einfache Durchführung. Es muss nur auf den richtigen Abstand zwischen den einzelnen Widgets geschaut werden, doch dies bringt auch Nachteile.

Die Bereiche links und rechts bleiben immer noch leer was ein wenig eigenartig wirken könnte. Ebenso besteht die Gefahr, dass die Widgets insgesamt zu hoch sind und somit verkleinert werden müssen, falls der Platz auf dem Bildschirm nicht reicht. Dies würde grafisch nicht nur schlechter wirken, sondern auch das Problem mit den freien Bereichen verschlimmern, da diese noch größer werden. Eine andere Möglichkeit wäre auch, das Bett und die Buttons zu trennen, wie Idee 2 veranschaulicht. Dadurch würden die Probleme von Idee 1 nicht auftreten, doch dafür ist die Umsetzung wesentlich aufwändiger, weil beide Bereiche unterschiedlich aussehen müssen und doch farblich zueinander passen müssen.

Vorab können wir nicht entscheiden, welche der beiden Konzepte besser für unser Projekt ist. Zunächst werden wir Idee 2 umsetzen, falls die Qualität der Lösung nicht ausreicht, wechseln wir zu Idee 1.

Zum Schluss kommt noch der Loading Screen. Dieser besteht aus einem Text "Bitte warten" und einem Ladebalken, der den Fortschritt des Ladens anzeigt:

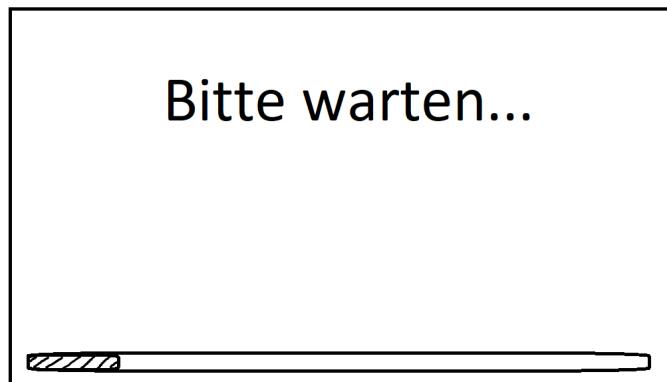


Abbildung 5.11: Loading Screen Skizze

### 5.3.3 Implementierungen von Guidelines

Mit den neu erlernten Guidelines und Richtlinien können wir jetzt auch diese auf alle oben angesprochenen Aufgabenbereiche anwenden.

#### 5.3.3.1 Anwendung: Information architecture (IA)

- **Statusanzeige:** Alle Aufgabenbeschreibungen werden untereinander angeordnet und die neueste Aufgabe steht immer oben. Ebenso ist es klar, was die zwei Icons (Lautsprecher und Smiley) bedeuten und wo diese zu sehen sind (HUD).
- **Hauptmenü:** Alle 3 Buttons haben einen klar definierten Zweck und sind auch so beschrieben, dass dieser von Anfang an klar ist.
- **Loading Screen:** Informiert den Benutzer, dass er warten muss. Somit weiß er was passiert und verhindert somit, dass er z. B. das Programm neu startet. Ebenso wird der Fortschritt an den Benutzer kommuniziert.

### 5.3.3.2 Anwendung: Farben und Formen

- **Statusanzeige:** Wenn eine Aufgabe abgeschlossen wird, befindet sich ein grüner Haken neben der Aufgabenbeschreibung. Lautstärke wird durch einen Lautsprecher symbolisiert und der Zustand des Patienten durch ein Smiley.
- **Loading Screen:** Der Ladebalken und der Text werden farblich hervorgehoben, damit diese Elemente sofort erkannt werden.

### 5.3.3.3 Anwendung: Clean Design

- **Statusanzeige:** Es gibt eine maximale Anzahl an Aufgabenstellungen, die gleichzeitig dargestellt werden. Ebenso bestehen die Aufgabenbeschreibungen nur aus einem Satz. Dadurch wird verhindert, dass der Benutzer zu viel Text auf einmal sieht und somit überfordert wird.
- **Hauptmenü:** Beinhaltet nur die notwendigsten Einstellungen in den Optionen.

### 5.3.3.4 Anwendung: VR-Spezifische Empfehlungen

- **Hauptmenü:** Buttons beinhalten unterschiedliche Zustände, damit erkannt werden kann, wenn sie gedrückt werden. Es wird auch ein Pointer statt einem Raycast benutzt.

## 5.4 Konzept für die **Props** und die Umgebung

In der Simulationsumgebung soll der Benutzer die Umgebung und sowie benutzten Tools so gut wie möglich wiedererkennen können. Umso wichtiger ist es, dass wichtige Details eingearbeitet werden, dabei aber auch auf die Simplizität des Ganzen geachtet wird.

Ein Konzept für die Erstellung der Modelle lässt sich daher am besten mit der realen Darstellung abnehmen. Bilder aus mehreren Perspektiven und unterschiedlichen Zuständen helfen zu erkennen, ob eine Animation gebraucht wird oder man das reale Objekt in mehrere 3D-Modelle aufteilen sollte. Auch grobe Skizzen vereinfachen den Prozess der Modellierung

### 5.4.1 Modellierungstechnik

Die Objekte werden mittels des **Box-Modelings** umgesetzt. Da es sich bei den Modellen um anorganische Objekte handelt, bietet sich diese Methode am besten an. Es wird dabei mit einem Polygon-Primitiv (Würfel, Kugel, Zylinder, Kapsel) gestartet und aus diesen dann, mittels der verfügbaren Polygon-Werkzeugen, die Einzelteile des fertigen Modells erstellt.

Die Objekte werden sich im Low bis Mid-Poly Bereich befinden, was bedeutet, dass das Mesh aus einer geringeren Anzahl an Dreiecken besteht, was das Rendering in der Engine erleichtert und somit der Performance Platz gibt.

### 5.4.2 Zimmer

Der Raum, in dem das Training stattfindet, soll den Räumen einer Pflegeeinrichtung (Altersheim, Krankenhaus) nachempfunden werden. Wichtig sind hier aber erstmals die benötigten Einrichtungsgegenstände, wie auch in der Abbildung 5.7 gezeigt wird:

- Krankenhausbett für die Patientin
- Regal auf Hüfthöhe für die **Props**

Andere Details, wie weitere Möbel oder Dekorationen, sind von sekundärer Priorität.

### 5.4.3 **Props**

Umso wichtiger für den Prozess gestalten sich die benötigten **Props**. Über die Vorgaben des Auftraggebers mussten wir zuerst herausfinden, welche für den Tracheostomawechsel notwendig sind. Folgende **Props** kommen dabei infrage:

- Spritze
- Manometer
- HME (feuchte Nase)
- Trachealkanüle
- Trachealkompresse
- Kanuelenbänder

- Gleitgeltuch

Im Folgenden beschreiben wir, wie der Prozess der Modellierung aussehen soll, um das gewünschte Ziel zu erreichen. Weiters wird in einem Satz beschrieben, welchen Zweck die **Props** erfüllen und, welches Verhalten eingebaut wird.

#### 5.4.3.1 Spritze

Die Spritze wird aus 2 Einzelteilen bestehen: dem **Zylinder** und dem **Kolben**. Die Eigenschaften dieser werden sein, dass sie sich unabhängig voneinander bewegen können, wodurch die Trennung benötigt wird. Auf eine Nadel oder Abdeckung wird aufgrund des Anwendungszweckes sowie der notwendigen Detailgebung verzichtet.

Die Form soll eine etwas größere Spritze darstellen. Also keine, welche beispielsweise bei Impfungen oder Blutabnahmen zum Einsatz kommen, sondern mit hoher Kapazität, da diese Luft einsaugen soll.

Als Animation wird sowohl eine Ein und Auszug Animation des Kolbens benötigt. Dies wird in Unity mittels Key-Frame Animation umgesetzt werden, da die Einzelteile getrennt wurden.

Für den Zylinder wird ein durchsichtiges Unity-Material erstellt, da vorerst keine Beschriftungen oder Abbildungen benötigt werden. Hier muss auf eine passende Farbgebung, Transparenz und Reflexion geachtet werden, um das Material so gut wie möglich zu treffen. Der Kolben wird mit einer Spritze typischen Türkisen Textur versehen. Diese wird in Maya mittels des Texturierungstools und UV-Maps erstellt und in Unity importiert.



Abbildung 5.12: Vorlage der Spritze [44]

#### 5.4.3.2 Manometer

Das Manometer ist vom Typen eines Cuffdruckmessgerätes. Dieses hat einen Griff, welcher sich zusammendrücken lässt, einen mittleren Körper, der das Gestell zusammenhält, ein Ziffernblatt und in dessen Mitte eine Nadel. Auf den seitlichen Knopf wird hier verzichtet, da er in der Anwendung nicht benötigt wird.

Wichtig für Animation ist hier die Trennung der einzelnen Teile. So werden diese in Griff, Körper und Nadel aufgeteilt, nachdem sowohl die Nadel als auch der Griff ein Verhalten haben sollen.

Für den Griff wird eine blau-lila Textur verwendet, für den Körper eine dunkelblaue und für die Nadel eine schwarze. Die Halterung der Nadel wird vergoldet. Da das Objekt in 3 Teile aufgeteilt wurde, werden hier auch 3 UV-Maps benötigt. Das Ziffernblatt wird erst in Illustrator erstellt und dann mittels Photoshop in die UV-Map eingefügt.



Abbildung 5.13: Vorlage des Manometers[2]

#### 5.4.3.3 HME-Filter (feuchte Nase)

Der Heat Moisture Exchanger (HME) oder auch feuchte Nase ist dazu da die, durch die Trachealkanüle eingehende, Luft warm und feucht zu machen, so wie es die Nase selbst auch tut.

Die Vorlage des HMEs wird hierbei ein Modell mit einem O<sub>2</sub>-Port sowie einer Abdeckung. Den Sauerstoff-Port sieht man dabei an der Seite, nämlich das dünne Rohr, mit welchem in der Praxis Luft direkt eingefügt und entnommen werden kann. Die Abdeckung kommt auf die Seite, welche nicht mit der Kanüle verbunden wird.

Das Modell wird in 5 Teile aufgeteilt, wobei der O<sub>2</sub>-Port und das Gehäuse im Endeffekt einen Körper darstellen und die 2 Filter, welche aus Papier oder Schaumstoff gemacht sind. Diese zwei Filter sind ebenfalls zwei eigene Objekte, die aber im Gehäuse stecken. Dazu kommt noch die Abdeckung, welche auf eine Seite des HMEs kommt

Die Filter und der O<sub>2</sub>-Port bestehen aus einfachen Zylindern, wobei der Port in der Mitte einen Durchgang hat und oben enger wird. Die Abdeckung wird ebenfalls aus einem Zylinder gemacht, wobei eine Kante zum drüberstecken hinzugefügt wird und die Öffnung, durch 6 Pyramiden dargestellt wird. In der realen Welt kann man diese bewegen, wird aber für das Modell nicht beachtet. Der Körper besteht aus 2 Zylindern, welche wieder in der Mitte leer sind. Im größeren Stecken dann die Filter und in der Mitte geht dann der 2 Zylinder im 90° Winkel durch. Der zweite Zylinder bekommt ebenfalls eine Öffnung zum Filterraum, wobei diese beiden Zylinderhälften leicht verbunden werden.

Das Gehäuse wird wieder aus einem durchsichtigen Kunststoff sein, wodurch es möglich ist die Filter zu sehen. Die Filter bekommen wie in der Vorlage eine gelbe Schaumstoff-textur. Auch der O<sub>2</sub>-Port bekommt eine durchsichtige Kunststofftextur.



Abbildung 5.14: Vorlage des HME(feuchte Nase)[42]

#### 5.4.3.4 Trachealkanüle

Die Trachealkanüle wird das Herzstück des gesamten Trainings und besteht aus den meisten Einzelteilen.

- **Kanülenrohr**, das um 90° gebogen wird.
- **Cuff**, der auf einer Seite des Rohrs platziert wird
- **Konnektor**, der auf der anderen Seite des Rohrs platziert wird. Dieser ist zum Aufsetzen von HMEs und Sprachventilen
- **Kanülen­schild**, das auf der Seite des Konnektors um das Rohr platziert ist. Die Bänder werden mit dem Schild verbunden, um die Kanüle zu befestigen
- **Kontrollballon**, der über einen dünnen Schlauch mit dem Cuff verbunden wird. Dieser Ballon lässt Luft in den Cuff und wieder heraus
- **dünner Schlauch**, der den Ballon und das Cuff verbindet.

Das Kanülenrohr wird aus einem Zylinder gebaut. Mittels des Boolean-Tools wird in der Mitte ein Durchgang geschaffen. Mittels des Bend-Deformers wird das Rohr dann gebogen. Das Kanülenrohr wird ebenfalls aus einem durchsichtigen Kunststoff bestehen, wobei dieses nicht ganz so stark transparent ist.

Der Cuff wird im selben Verfahren, wie das Rohr gemacht, nur das dieses etwas breiter und um einiges niedriger ist. Das Innere sollte etwas breiter sein, als das Rohr, damit das Rohr durch die Biegung den Cuff nicht berührt. Durch das Bevel-Tool werden die Kanten um einiges abgerundet, um die Ballon-Form zu erzeugen. Der Cuff soll ebenfalls leicht transparent sein, aber etwas mehr reflektieren.

Auch der Konnektor wird aus einem hohlen Zylinder gemacht, welcher am Ende ein Stück nach oben geschoben bekommt. Der Konnektor wird in einem einfachen Weiß dargestellt.

Ausgangspunkt für das Schild ist ebenfalls ein Würfel, wobei die Seiten gestrafft werden, damit er dünn aber weit geht. Mit Subdivisions wird dann durch Ecken die grundlegende Form vorgegeben. Mittels des Bevel-Tools werden alle eckigen Kanten auf der Seite abgerundet. Als letztes werden dann mittels des Boolean-Tools zwei Kapselförmige Löcher für die Bänder in das Schild gegeben. Das Schild bekommt ebenfalls eine weiße Farbe.

Der Kontrollballon wird wieder mit einem Würfel gestartet und in die Form eines Ballons ge cutet. Mittels der Edges wird die Form vorgefertigt und mittels des Bevel-Tools diese abgerundet. Der Ballon bekommt ebenfalls einen kleinen Konnektor an das Ende, welches nicht durch den Schlauch mit der Kanüle verbunden ist.

Der Schlauch wird in Unity mittels Zylinder und Hinge-Joints gebaut, wodurch der Ballon an dieser hängt und sich dieser auch wie ein Schlauch verhältet.

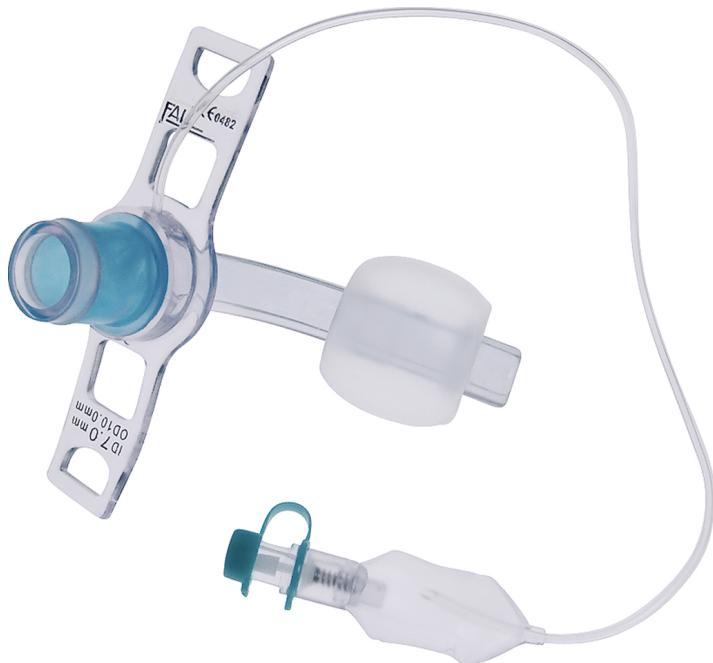


Abbildung 5.15: Vorlage der Trachealkanüle[36]

#### 5.4.3.5 Trachealkompresse

Der Sinn der Trachealkompresse ist die Aufnahme des abgesonderten Sekrets, das auf die Kanüle mit dem Tuch aufgetragen wird. Es handelt sich dabei um einen Körper aus Stoff und polstert den Bereich zwischen Hals und Trachealkanüle auf.

Als Beispiel wird hier eine 3-lagige Trachealkompresse verwendet, wodurch diese auch etwas höher dargestellt werden muss. Das Objekt wird dieses Mal nur aus einem Teil bestehen. Mit den Poly-Werkzeugen muss ein Loch und eine Trennung zwischen den Wänden vom Loch zu einer Seite der Kompresse ausgeschnitten werden. Ebenfalls, wird die Kompresse ein wenig gebogen, wodurch es vorkommt, als würde sie sich an die Form des Halses anpassen.

Animation benötigt die Kompresse keine. Die Trachealkompresse bekommt eine weiße Stoff-textur, die man im Internet leicht finden kann.



Abbildung 5.16: Vorlage der Trachealkomresse[39]

#### 5.4.3.6 Kanülenbänder

Die Kanülenbänder bestehen auch wieder aus einem gepolsterten Stoff, wodurch das Modell wie ein Polster erstellt werden soll. Die Idee ist es die in Unity nutzbare Cloth Physik mit einem Constraint zu verwenden und die beiden Bänder so an die Kanüle anzubringen, dass man sie mit beiden Händen verbinden kann.

Wir erstellen nur eines der beiden Bänder und spiegeln dieses, um es auch auf der anderen Seite richtig darzustellen



Abbildung 5.17: Vorlage der Kanülenbänder[20]

#### 5.4.3.7 Gleitgeltuch

Das Gleitgeltuch wird aufgrund der Einschränkungen der Cloth-Komponente in Unity vorerst nur durch eine Cloth-Animation als einzelnes Mesh dargestellt, welches etwas gewellt ist. Dabei wird das Tuch als Plane gebaut und durch die Animation dann verformt. Dieses Tuch bekommt eine weiße Stofftextur.

#### 5.4.4 Hände

Der Spieler der VR-Welt hat aus seiner Sicht nur Hände und auch diese sind neben der Brille auch alles was da ist um mit der Umgebung zu interagieren. Diese Hände werden an die Bewegungen des Controllers in der echten Welt angepasst.

Um einen gewissen Realismus in die VR-Simulation zu bringen, sollen die Hände eine Greifanimation bekommen, wenn man nach den Gegenständen in der Umgebung greift. Diese soll mittels des Rigging-Tools in Maya erstellt und als Animation nach Unity exportiert werden.

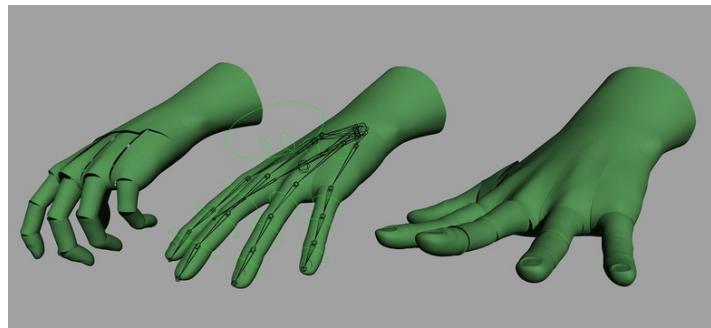


Abbildung 5.18: Vorlage der Greifanimation und Rigs[3]



# Kapitel 6

## Implementierung

Da nun alle notwendigen Konzepte erstellt wurden, können diese jetzt implementiert werden. Die Methoden oder Funktionen von Unity, die in den unteren Kapiteln benutzt werden, befinden sich alle in der Unity Dokumentation [80].

### 6.1 Logik und Interaktivität einer Schulungs-App

#### 6.1.1 Programmsteuerung

Wie bereits im Konzept beschrieben, ist die Steuerung des Programms eines der Hauptaufgaben der Logik-Komponente des Programms. Für die Implementierung ist die Struktur des Programms durch die oben angegeben Klassendiagramme definiert. Diese Klassen sollen nun absteigend nach Relevanz implementiert werden.

##### 6.1.1.1 Process-Handler

Angefangen wird die Implementierung mit dem zentralen Process-Handler. Dieser besitzt die Rolle, eine Schnittstelle zwischen mehreren Managern aufzubauen. Um diese Rolle effizient ermöglichen zu können, haben wir uns für die Anwendung des Singleton Patterns entschieden, welches den Zugriff auf den Process-Handler leicht ermöglicht. Damit der Process-Handler funktionieren kann, benötigt er jeweils eine Referenz für den SceneLoader, TaskManager und UIManager durch den Inspektor. Diese Referenzen bieten dem Process-Handler eine von außen unzugängliche Verbindung an alle notwendigen Akteure. Zudem sollen universell notwendige Referenzen und Prefabs durch den Process-Handler erreichbar sein.

##### Code

Der kommentierte, abgekürzte Code des Process-Handlers sieht folgendermaßen aus:

```

1  public class ProcessHandler : MonoBehaviour
2  {
3      // Notwendige Manager
4      [SerializeField] private SceneLoader sceneLoader;
5      [SerializeField] private TaskManager taskManager;
6      [SerializeField] private UserInterfaceManager uiManager;
7      // Notwendige Referenzen/Prefabs
8      [SerializeField] private GameObject tracheostomaCO, woman;
9      // Anwendung des Singleton Patterns
10     private static ProcessHandler _instance;
11     public static ProcessHandler Instance { get { return _instance; } }
12
13     private void Awake()
14     {
15         if(_instance != null && _instance != this) Destroy(this.gameObject);
16         else _instance = this;
17     }
18
19     void Start()
20     {
21         // [...] Richten Prozess auswählen
22     }
23     // Startet die nächste Task
24     public void NextTask() => taskManager.NextTask(false);
25
26     // Wird aufgerufen, wenn alle Tasks vorbei sind
27     public void EndOfTasks() => uiManager.EndOfTasks();
28
29     // Ladet eine Szene, definiert über eine ID
30     void LoadScene(string pid)
31     {
32         sceneLoader.LoadProcess(pid);
33         taskManager.SetToolList(sceneLoader.GetToolList());
34         taskManager.SetTaskList(sceneLoader.GetTaskList());
35     }
36     // Übergibt dem UI-Manager die neue Task-Beschreibung
37     public void UINextTask(string desc) => uiManager.NewTask(desc);
38 }
```

Auflistung 6.1: Code des ProcessHandlers

### 6.1.1.2 Scene-Manager

Damit alle notwendigen Objekte geladen und die Szene präpariert werden kann, ist als Nächstes der Scene-Manager notwendig. Dieser soll für den Process-Handler eine Möglichkeit bereitstellen, einen ausgewählten Prozess zu laden. Um das Laden eines Prozesses erstmals zu ermöglichen, muss der Scene-Manager eine Liste aller Prozesse erhalten. Durch eine vom Process-Handler übergebene ID kann der Scene-Manager den zugehörigen Prozess aus der Liste finden. Aus dem Prozess soll eine Tool- und Task-Liste entpackt werden können. Die entpackten Tools sollen instantiiert und gespeichert werden, wonach diese dem Process-Handler bereitge-

stellt werden. Der Process-Handler übernimmt nach der Ladung des Prozesses die Task- und Tool-Liste.

### Code

Der kommentierte, abgekürzte Code des SceneManagers sieht folgendermaßen aus:

```

1  public class SceneManager : MonoBehaviour
2  {
3      // Vorhandene Prozesse
4      [SerializeField] private ProcessScriptableObject[] processes;
5      private ProcessScriptableObject selectedProcess;
6      public List<GameObject> toolList = new List<GameObject>();
7
8      // Ladt anhand einer ID einen Prozess
9      public void LoadProcess(string pid)
10     {
11         // [...] Richtiger Prozess wird in selectedProcess geladen
12         GameObject[] tools = selectedProcess.toolList;
13         int i = 1;
14         // Alle notwendigen Tools werden instantiiert
15         foreach (GameObject p in tools)
16         {
17             toolList.Add(Instantiate(p, spawnPoints[i].position ,
18             Quaternion.identity, this.gameObject.transform));
19             i++;
20         }
21         // Gibt notwendige Objekte im Listenformat zurück
22         public List<GameObject> GetTaskList() => selectedProcess.GetTasklist();
23         public List<GameObject> GetToolList() => return toolList;
24     }

```

Auflistung 6.2: Code des SceneManagers

#### 6.1.1.3 Task-Manager

Nachdem alle erforderlichen Daten entpackt und instantiiert wurden, können die Tasks nun gestartet und bearbeitet werden. Damit der Task-Durchlauf gesteuert werden kann, wird der Task-Manager benötigt. Dieser verwendet die Task- und Tool-Liste, welche vom Process-Handler übergeben werden. Die Tasks sind in einer Liste gespeichert - eine Datenstruktur, welche das Entfernen von Elementen leicht ermöglichen soll. Signalisieren Tasks ihren Abschluss, soll der Task-Manager die vorherige Task beenden und aus der Liste entfernen. Darauf-folgend wird die nächste Task gestartet, indem diese instantiiert wird.

### Code

Der kommentierte, abgekürzte Code der Task-Manager sieht folgendermaßen aus:

```

1  public class TaskManager : MonoBehaviour
2  {
3      private List<GameObject> taskList;
4      public List<GameObject> toolList;
5      [SerializeField] private Task currentTask;
6      // Startet die nächste Task
7      public Task NextTask(bool isFirst)
8      {
9          if (taskList.Count <= 0)
10         {
11             ProcessHandler.Instance.EndOfTasks();
12             return null;
13         }
14         // [...] Spielt den Task-Anfang Sound ab
15
16         Task task = StartNextTask();
17         ProcessHandler.Instance.UINextTask(task.description, isFirst);
18         return task;
19     }
20     public Task StartNextTask()
21     {
22         // [...] Instantiiert die nächste Task
23         taskList.RemoveAt(0);
24         if (task != null)
25         {
26             task.SetSpawnTools(toolList.ToArray());
27             task.StartTask();
28             if (currentTask != null) currentTask.FinishTask();
29             currentTask = task;
30         }
31         return task;
32     }
33
34     // [...] Setter-Methoden für Attribute
35
36 }

```

Auflistung 6.3: Code des TaskManagers

#### 6.1.1.4 User-Interface Manager

Damit der User über den Verlauf des Programms informiert werden kann, ist nun die Einbindung von UI-Elementen notwendig. Der logische Teil dieser Aufgabe ist das Liefern notwendiger Informationen über die derzeitige Task. Die wichtigste Information ist hier die Task-Beschreibung, welche auf einem UI-Element angezeigt werden soll. Zusätzlich soll das UI-Element auf den Abschluss aller Tasks reagieren können. Zusatzinformationen wie benötigte Warn-schilder sollen ebenfalls an die UI übergeben werden.

##### Code

Der kommentierte, abgekürzte Code des UI-Managers sieht folgendermaßen aus:

```

1  public class UserInterfaceManager : MonoBehaviour
2  {
3      // [...] Referenzen zu UI Elementen
4
5      // Zeigt neue Tasks in den UI-Elementen an
6      public void NewTask(string taskdescription, bool isFirst)
7      {
8          // [...] Übergibt notwendige Informationen an UI-Elemente
9      }
10     // Aufruf, wenn alle Tasks vorbei sind
11     public void EndOfTasks() => StartCoroutine(wbh.ShowEndMessage_2());
12     // Gibt dem User eine Warnung an
13     public void ShowWarning(int warningIndex) => hh.Warning(warningIndex);
14 }
```

Auflistung 6.4: Code des UI-Managers

## 6.1.2 Tasks

Da alle notwendigen Manager jetzt vorhanden sind und somit die logische Infrastruktur gegeben ist, wird das Implementieren von den spezifischen Tasks notwendig.

### 6.1.2.1 Task

Wie im Konzept angeschrieben ist es sinnvoll zuerst eine Basis-Klasse für alle Tasks zu erstellen. Diese soll allgemeine Funktionalitäten für alle Tasks bereitstellen und eine klare Definition einer Task bieten. Eine Task beinhaltet:

- **taskName:** Beinhaltet den Namen der Task.
- **description:** Eine Beschreibung der Ausführung jener Task, soll im UI angezeigt werden.
- **spawnedTools:** Eine Liste aller instantiierten Tools, welche mithilfe der Methode "FindTools(string)" durchsucht werden kann.

Wichtig sind die Methoden StartTask() und FinishTask() welche in jeder Task jeweils am Anfang und am Ende aufgerufen werden sollen. Indem diese Methoden als "virtual" markiert werden, soll das Erweitern leicht ermöglicht werden. Da bei manchen Tasks die Anzeige von Warnschildern möglich sein soll, wurde die Klasse nach diesen Anforderungen erweitert.

```

1  public abstract class Task : MonoBehaviour
2  {
3      public string tName, description;
4      public GameObject[] spawnedTools;
5      public bool warning_Message = false;
6      // Wird beim Start der Task aufgerufen
7      public virtual void StartTask()
8      {
9          // [...] Zeigt angegebene Warn-Schilder an
10     }
11    // Wird beim Beenden der Task aufgerufen
12    public virtual void FinishTask() => Destroy(this);
13    // Findet ein Tool aus der Tool-Liste
14    public GameObject FindTool(string prefabName)
15    {
16        for (int i = 0; i < spawnedTools.Length; i++)
17        {
18            // Instantiierte Tools haben ein zugehörigen String
19            if ((prefabName + "(Clone)") == spawnedTools[i].name) return
20            ↪   spawnedTools[i];
21        }
22        // [...] Zusätzliche Elemente, welche durchsucht werden
23        return null;
24    }
}

```

Auflistung 6.5: Code der abstrakten Klasse Task

### 6.1.2.2 Erste Idee

Beim Implementieren der Tasks war die erste Idee ein Event System einzubauen, welches verschiedene Events (z.B. ein CollisionEvent) an die derzeit aktive Task sendet. Jedoch stellte dieser Ansatz sich als schwer erweiterbar und schlecht behandelbar heraus, weshalb diese Idee verworfen wurde.

```

1  public class SimpleCollisionEvent
2  {
3      public GameObject FirstObject { get; set; }
4      public GameObject SecondObject { get; set; }
5
6      public SimpleCollisionEvent(GameObject firstObject, GameObject secondObject)
7      {
8          FirstObject = firstObject;
9          SecondObject = secondObject;
10     }
11    // Berichtet über Kollisionen
12    public void ReportCollision() => ProcessHandler.Instance.ReportCollision(this);
13
14    public override GameObject[] GetEventData()
15    {
16        GameObject[] daten = { FirstObject, SecondObject };
17        return daten;
18    }
19 }
```

Auflistung 6.6: Beispielcode eines Events

**Beispielcode:****6.1.2.3 InteractableObjects**

Nachdem die Event-Lösung nicht durchführbar war und diese verworfen wurde, sind weitere Ideen dazugekommen. Entschlossen wurde, dass Objekte selbst für das Überprüfen der Task-Vervollständigung zuständig sein sollen. Dafür wird das InteractableObject Script erstellt, welche als Basis-Klasse für Task Objekte dienen soll. InteractableObjects sind beispielsweise verwendete Tools.

**6.1.2.4 TouchTask**

Für Aufgaben, welche die Berührung zweier Objekte benötigen, ist eine eigene Task-Art notwendig, die TouchTask. Diese ist sehr simpel aufgebaut und benötigt nur die zwei zu berührenden Objekte aus dem Inspektor Fenster. Die Aktivierung des TouchObjects wird von der Task übernommen.

```

1  public class TouchTask : Task
2  {
3      public GameObject touchObject, touchTarget;
4      public override void StartTask()
5      {
6          base.StartTask();
7          touchObject = base.FindTool(touchObject.name);
8          touchTarget = base.FindTool(touchTarget.name);
9          // Setzt das Target im Tool
10         touchObject.GetComponent<TouchObject>().SetTouchTarget(touchTarget);
11     }
12 }

```

Auflistung 6.7: Code der Touchtask

**TouchTask-Code:****TouchObject:**

Das dazugehörige Objekt ist ebenfalls simpel aufgebaut. Das TouchObject nutzt Unity Methoden, um bei Kollision das touchTarget mit dem Kollisionsobjekt zu vergleichen.

```

1  public class TouchObject : InteractableObject
2  {
3      private GameObject touchTarget;
4      // Bei Berührung des touchTarget wird die Task beendet
5      private void OnCollisionExit(Collision collision) {
6          if (collision.gameObject.Equals(touchTarget))
7          {
8              ProcessHandler.Instance.NextTask();
9          }
10     }
11 }

```

Auflistung 6.8: Code eines TouchObject

**6.1.2.5 ConnectionTask**

Die meisten Tasks lassen sich auf das Verbinden von Gegenständen abstrahieren, weshalb eine zugehörige Task, die ConnectionTask unerlässlich ist. Da die ConnectionTask zu besonders vielen Tasks passen muss, ist dessen Flexibilität sehr wichtig. Die ConnectionTask selbst besitzt einen ähnlichen Aufbau zu anderen Tasks, indem diese zwei Parameter übernimmt: dem Connector und dem Connectible.

**Connectible:**

Das Connectible beschreibt ein Objekt, welches zu einem Connector verbunden wird. Bei der Kanüle wären das beispielsweise die Bänder oder die feuchte Nase. In Tasks, welche die Cuffline involvieren, stellt die Cuffline ebenfalls ein Connectible dar. Ein Connectible beinhaltet viele verschiedene Methoden, um die Verbindung zwischen Objekten zu ermöglichen. Für uns ist hier

jedoch in diesem Kapitel die Logik hinter wie die Verbindung passiert irrelevant, dies wird näher im Kapitel [Compound Object und Connections](#) besprochen. Wichtig ist, dass die Task dem Connectible eine Referenz für das zugehörige ConnectorObject über gibt.

### Connector:

Der Connector beschreibt das Objekt, an welches ein Connectible gebunden wird, im Programm stellt das beispielsweise die Kanüle dar. Wie das Connectible beinhaltet ein ConnectorObject viel Verbindungslogik, welches wiederum im Interaktionskapitel näher angesprochen wird. Die Task diktiert dem ConnectorObject, wann dieser aktiviert und somit verbunden werden können soll. Bei der Verbindung mit dem Connectible wird die nächste Task geöffnet und das ConnectorObject wieder inaktiviert.

```

1  public class ConnectorObject : InteractableObject
2  {
3      public bool connectorActive = false;
4      // [...] Verbindungsrelevante Informationen
5
6      public virtual void Connect(GameObject connectible)
7      {
8          if (!connectible.GetComponent<InteractableObject>().GetIsGrabbed() &&
9              !connectorActive && this.GetIsGrabbed())
10         {
11             // [...] Ist für die Verbindung zuständig
12             ProcessHandler.Instance.NextTask();
13         }
14     }
15 }
```

Auflistung 6.9: Code eines ConnectorObjects

**ConnectionTask-Code:**

```

1  public class ConnectionTask : Task
2  {
3      public GameObject connector, connectible;
4      public override void StartTask()
5      {
6          base.StartTask();
7          connector = base.FindTool(connector.name);
8          connectible = base.FindTool(connectible.name);
9          if (connector != null && connectible != null)
10         {
11             connector.GetComponent<ConnectorObject>().connectorActive = true;
12             connectible.GetComponent<Connectible>().SetConnector(connector.GetComponent<ConnectorObject>());
13         }
14     }
15 }
```

Auflistung 6.10: Code einer ConnectionTask

**RotationTask** Eine Erweiterung der ConnectionTask war geplant, welche die Verbindung der Objekte nur bei entsprechender Rotierung des Objekts erlaubt. Diese Task soll das Platzieren der Kanüle realistischer machen, da diese beim Hineinschieben in der Realität eine Drehbewegung benötigt. Nachdem wir jedoch unsere Prioritäten neu evaluiert haben, sind wir zum Entschluss gekommen andere Task-Arten zu priorisieren. Ersetzt wurde diese Task durch eine simple ConnectionTask.

**6.1.2.6 PressTask**

Manche Tasks benötigen das Betätigen eines Tools, wie z.B. das Manometer oder die Spritze. Die Task ist ähnlich zu der TouchTask und übernimmt ein Objekt über dem Inspektor. Das Interagieren mit dem Objekt wird über die Task geregelt.

**PressTask-Code:**

```

1  public class PressTask : Task
2  {
3      public GameObject pressObject;
4      public override void StartTask()
5      {
6          pressObject = base.FindTool(pressObject.name);
7          // Ermöglicht das Drücken des Objekts
8          pressObject.GetComponent<PressObject>().SetPressable(true);
9      }
10 }
```

Auflistung 6.11: Code einer PressTask

**Object-Code:**

Ein PressObject überprüft, ob dieses gedrückt werden kann und wird bei Knopfdruck aktiviert.

Es stellt eine Basis-Klasse dar und kann durch Unterklassen erweitert werden.

```

1  public class PressObject : MonoBehaviour
2  {
3      public bool pressable = false;
4      // Wird bei Knopfdruck aufgerufen
5      public virtual void Press()
6      {
7          if (pressable) ProcessHandler.Instance.NextTask();
8      }
9
10     public void SetPressable(bool pressable) => this.pressable = pressable;
11 }
```

Auflistung 6.12: Code eines PressObjects

#### 6.1.2.7 RemoveTask

Für die "Kanüle Herausnehmen" Task Durchlauf benötigen wir noch die Möglichkeit ein zusammengesetztes Objekt in Einzelteile zu zerlegen. Die Remove-Task bildet das Gegenteil zu der ConnectionTask und kann bei Kanüle verwendet werden. Es findet das zu entfernende Objekt und aktiviert die Aufhebfunktion dieser. Dafür benötigen wir wiederum eine spezielle Art von Objekt.

**CompoundObject:** Ein CompoundObject speichert Referenzen zu seinen Bestandteilen samt ihrer Reihenfolge. Wichtig ist, dass die RemoveTask somit weiß welche das nächste zu entfernende Objekt ist. Das in der Szene benötigte CompoundObject soll im ProcessHandler gespeichert werden.

**CompoundPart:** Ein CompoundPart stellt ein Bestandteil eines CompoundObjects dar. Bei Aktivierung dieser, durch die Task soll, dieses beim Loslassen die Task beenden.

```

1  public class CompoundPart : InteractableObject
2  {
3      private bool taskFocus;
4      // Wird beim Loslassen aufgerufen
5      public override void OnDrop()
6      {
7          base.OnDrop();
8          if (taskFocus)
9          {
10              ProcessHandler.Instance.NextTask();
11              taskFocus = false;
12          }
13      }
14 }
```

Auflistung 6.13: Code eines CompoundParts

```

1  public class RemoveFromCompoundObjectTask : Task
2  {
3      [SerializeField] private GameObject compoundObject;
4      private GameObject objectToRemove;
5      public override void StartTask()
6      {
7          base.StartTask();
8          CompoundObject c0 = ProcessHandler.Instance.GetCompoundObject().GetComponent<CompoundObject>();
9          if (c0) compoundObject = c0.gameObject;
10         objectToRemove = compoundObject.GetComponent<CompoundObject>().GetPart();
11         if (objectToRemove)
12         {
13             CompoundPart compP = objectToRemove.GetComponent<CompoundPart>();
14             compP.SetGrabbable(true);
15             compP.SetTaskFocus(true);
16         }
17     }
18 }
```

Auflistung 6.14: Code der RemoveTask

**RemoveTask-Code:****Object-Code:**

Ein PressObject überprüft, ob dieses gedrückt werden kann und wird bei Knopfdruck aktiviert. Es stellt eine Basis-Klasse dar und kann durch Unterklassen erweitert werden.

```

1  public class PressObject : MonoBehaviour
2  {
3      public bool pressable = false;
4      // Wird bei Knopfdruck aufgerufen
5      public virtual void Press()
6      {
7          // Ruft die nächste Task auf
8          if (pressable) ProcessHandler.Instance.NextTask();
9      }
10
11     public void SetPressable(bool pressable) => this.
12     pressable = pressable;
13 }
```

Auflistung 6.15: Code eines ProessObjects

**6.1.2.8 BandControlTask**

Mit den bisherigen Task-Arten können wir fast alle erforderlichen Tasks implementieren. Die letzte nun notwendige Task ist das Anbinden des Bandes am Hals der Person. Für die Anbindung der Bänder wurde eine Verkrümmung-Animation erstellt, welche im Kapitel (LUCA) detailliert

wird. Die Überlegungen waren, wie man den Prozess so realistisch wie möglich gestalten kann, trotz der Limitationen von VR.

### **Überlegungen**

Da hier die Verwendung beider Hände ausschlaggebend ist, muss die Task mit beiden Händen geschehen. Das kann durchgeführt werden, indem wir beim Drücken des Interaktionsknopfs die Mitte der zwei Hände finden und kontrollieren, ob die Mitte in einem definierten Radius ist. Diese Methode hat zwar ebenfalls Nachteile, da sie auch alternative Lösungen erlaubt. Jedoch wurde beschlossen, dass das kein großes Problem darstellt und innerhalb einer VR-Welt eine gute Lösung darstellt.

**BandControlTask-Code:**

```

1  public class BandControlTask : Task
2  {
3      [SerializeField] private GameObject bandLinks, bandRechts;
4      // Der Toleranzradius
5      [SerializeField] private float range = 0.1f;
6      // Falls die Animation rückwärts gespielt werden muss
7      [SerializeField] private bool inReverse = false;
8      private Animator[] animators;
9      private Vector3 centerVector;
10     private GameObject[] hands;
11     public override void StartTask()
12     {
13         base.StartTask();
14         bandLinks = base.FindTool(bandLinks.name);
15         bandRechts = base.FindTool(bandRechts.name);
16         // Berechnet sich den zentralen Vektor der Bänder
17         centerVector = (bandLinks.transform.position +
18             ↳ bandRechts.transform.position) / 2;
19         // [...] Referenzen werden gesetzt
20     }
21     // Wird bei Knopfdruck ausgeführt
22     public void CheckDistance(InputAction.CallbackContext context)
23     {
24         Vector3 handsCenter = new Vector3();
25         // Berechnet sich den Mittelpunkt der Hände
26         foreach(GameObject hand in hands)
27         {
28             handsCenter += hand.transform.position;
29         }
30         handsCenter /= hands.Length;
31         // Schaut, ob der Mittelpunkt im Toleranzradius ist
32         if (Vector3.Distance(handsCenter, centerVector) <= range)
33         {
34             foreach (Animator anime in animators) anime.SetTrigger("biegen");
35             ProcessHandler.Instance.NextTask();
36         }
37     }
38 }
39

```

Auflistung 6.16: Code einer BandControlTask

**6.1.2.9 Task-Aufteilung**

Da alle notwendigen Task-Arten nun implementiert sind, müssen wir nur noch jeder spezifischen Task eine der Arten zuweisen. Geeinigt haben wir uns auf folgende Zuweisungen geeinigt:

**Kanuele rausnehmen:**

- **Feuchte Nase abnehmen** -> Remove Task
- **Luft aus der Cuffline entfernen** -> ConnectionTask + PressTask
- **Band öffnen** -> Remove Task
- **Kanüle herausnehmen** -> Remove Task

**Kanuele reingeben:**

- **Cuff mit der Spritze kontrollieren** -> ConnectionTask + PressTask
- **Bänder befestigen** -> ConnectionTask
- **Trachealkompresse befestigen** -> ConnectionTask
- **Kanüle mit Gleitgel beschmieren:** -> TouchTask
- **Kanüle befestigen** -> ConnectionTask
- **Manometer mit Cuff verbinden:** -> ConnectionTask
- **Ballon aufpumpen:** -> PressTask
- **Feuchte Nase aufsetzen:** -> ConnectionTask

### 6.1.3 Daten

#### 6.1.3.1 ScriptableObject

Um einen Prozess zu erstellen, benötigen wir eine Art Datenbank, welche zu verwendenden Tools und Tasks speichert. Entschieden haben wir uns für die Verwendung eines ScriptableObjects, welcher unsere Prefabs speichern kann. Somit können wir verschiedene Prozesse abbilden.

##### Code

ScriptableObjects sind ähnlich zu normalen Scripts und beinhalten normalen Code. Genannt haben wir unser ScriptableObject "ProcessScriptableObject" und dessen Aufbau ist wie folgt:

```

1 // Ermöglicht die Erstellung eines Objekts im AssetMenu
2 [CreateAssetMenu(fileName = "ProcessScriptableObject", menuName =
3     "ScriptableObjects/Process", order = 1)]
4 public class ProcessScriptableObject : ScriptableObject
5 {
6     // Prozessname und deren Beschreibung
7     public string pName;
8     public string description;
9     // Liste an benötigten Tools
10    public GameObject[] toolList;
11    // Liste von allen Tasks
12    [SerializeField] private GameObject[] taskList;
13
14    // Gibt die Länge der Task-Liste zurück
15    public int RemainingTasks()
16    {
17        return taskList.Length;
18    }
19
20    // Gibt die Task-Liste zurück
21    public List<GameObject> GetTasklist()
22    {
23        return new List<GameObject>(taskList);
24    }

```

Auflistung 6.17: Code eines Prozesses als ScriptableObject

## Verwendung

Nun können wir ein Prozess wie im Diagramm dargestellt erstellen:

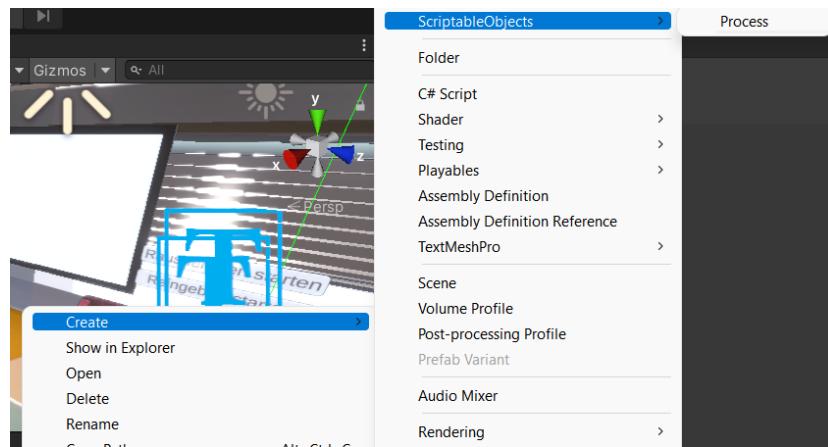


Abbildung 6.1: Erstellung eines Prozesses

Um nun diesem, Tasks und Tools zuweisen zu können wenden wir uns erneut an das Inspektor-Fenster und ziehen benötigte Prefabs, wie im Diagramm dargestellt einfach hinein.

### 6.1.3.2 Task-Prefabbing

Zuletzt müssen wir noch alle Tasks als Prefabs implementieren, indem wir diese erstellen, ihnen das richtige Skript zuweisen und alle Parameter im Inspektor setzen. Im Diagramm sieht man beispielsweise eine ConnectionTask als Prefab, welche die Kanüle mit dem linken Band verbindet.

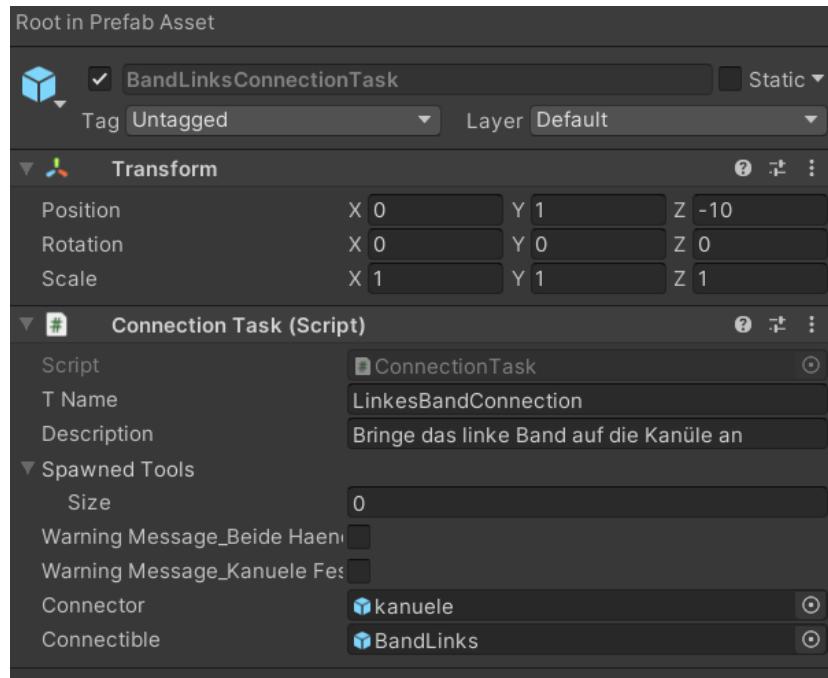


Abbildung 6.2: Erstellen einer Task

## 6.2 Bewegung und Interaktion mit der virtuellen Realität

### 6.2.1 Szene

Die Arbeit hat mit der allerersten Testszene begonnen. Da haben wir das komplette XR Rig konfiguriert, sowie die XR Controller. Es wurden die ersten Hand-Models in die Szene hereingebracht. Es wurde gleich auch der **ProcessHandler** inkludiert. Es wurden auch schon die ersten Tests mit der UI Interaktion durchgeführt.

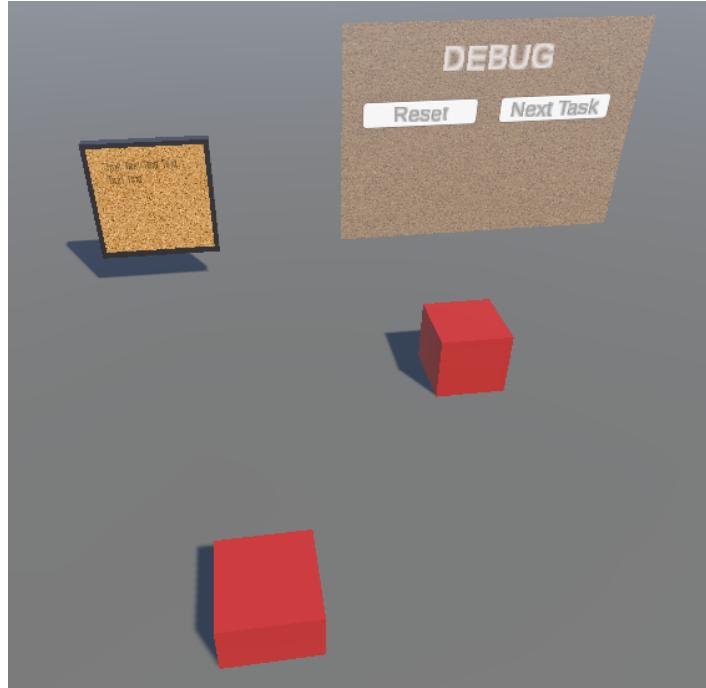


Abbildung 6.3: Erste Test Szene

Die erste Erkenntnis gab es bei den **XR Interactor** Klassen. XR Direct Interactor wird für Grabbing benötigt und XR Ray Interactor für die Interaktion mit dem User Interface. Da ein XR Controller nur eine Interactor Instanz haben darf, mussten wir für jede Hand jeweils zwei XR Controller erstellen, einen für Direct Interactor und einen für Ray Interactor. Das eigentliche Handmodell befindet sich dann nur auf den jeweiligen Direct Interactors. Dieser Ansatz hat einwandfrei funktioniert.

**Spawnpoints** Ein wichtiger Teil der Szene für die dynamische Behandlung von Workflows sind die Spawnpoints. In der Szene befinden sich einige **leere GameObjects** für die nur ihre Position (Transform) relevant ist. Diese Spawnpoints müssen somit auf die korrekten Positionen platziert werden. Die Spawnpoints werden dem ProcessHandler übergeben, der anschließend die benötigten **Tools und Props** anhand der **Spawn Locations** instanziert. Die Spawnpoints sind für die dynamischen Prozesse von höchster Bedeutung. Der folgende Codeausschnitt ist für das instanziieren der Tools anhand der Spawnpoints zuständig:

```
1  GameObject[] tools = selectedProcess.toolList;
2  Transform[] spawnPoints = ProcessHandler.Instance.GetSpawnPoints();
3
4  if(tools.Length >= spawnPoints.Length)
5  {
6      return;
7  }
8
9  int i = 1;
10
11 foreach (GameObject p in tools)
12 {
13     toolList.Add(Instantiate(p, spawnPoints[i].position , Quaternion.identity,
14     ↳ this.gameObject.transform));
15     i++;
16 }
```

Auflistung 6.18: Codeausschnitt für das instanziieren der Tools

In der Testszene befinden sich auch neben den Spawnpoints auch sogenannte **Snap Zones**. Diese verwenden die **XR Socket Interactor** Klasse aus dem XR Interaction Toolkit. Objekte können in solche Snap Zones einrasten und im Platz bleiben. Diese Snap Zones wurden später im Projekt nicht mehr übernommen, aufgrund der Notwendigkeit von multiplen Verbindungen.

**Finale Szene** Die weitere (finale) Szene ist entstanden, nachdem das erste Raummodell und mit ihm auch das erste Patientenmodell verfügbar gemacht wurde. Die Spawn Points wurden erweitert, um Platz für mehr Tools zu machen. Die relevanten Elemente der Geometrie wurden mit entsprechenden Collidern versehen, wie zum Beispiel der Kasten, auf dem die Tools liegen. Ebenfalls hat die Patientin einen entsprechenden Collider bekommen.

### 6.2.2 Room-scale based

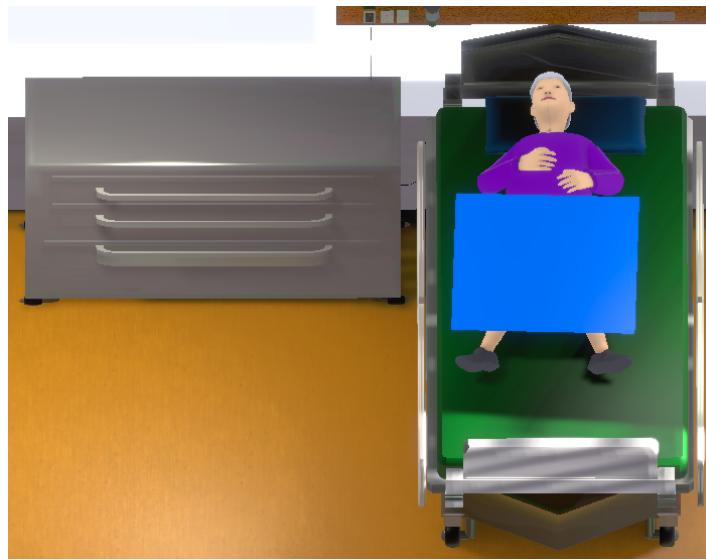


Abbildung 6.4: Abschnitt des Raumes

Für dieses Projekt wurde Room-scale based als Fortbewegungsvariante entschieden. Das bedeutet natürlich, dass kein wirkliches Fortbewegungssystem entwickelt werden musste. Stattdessen musste die Anwendung so gestaltet werden, dass alles Relevante für den Nutzer in Griffnähe ist. Das Layout der finalen Szene basiert auf dem Layout aus Abbildung 5.7. In der oberen Abbildung 6.4 sieht man, einen Kasten und das Bett auf dem die Patientin liegt. Der Nutzer würde sich dazwischen befinden, links unten auf der Abbildung. Auf dem Kasten befinden sich die vorher erwähnten Spawnpoints, somit sind die entsprechenden Tools immer in Griffnähe bereit. Ebenfalls ist die Patientin direkt nebenan.

Nach ausführlichen Testungen hat das Layout für alle Testpersonen perfekt gepasst. Der komplette Workflow findet in diesem kleinen Bereich statt, deswegen ist kein Fortbewegungssystem nötig.

### 6.2.3 Interaktion

Das Erste, was benötigt wird, sind XR Controller mit einem XR Direct Interactor drauf. Diese sind für die Interaktion mit Objekten zuständig. Objekte, mit denen interagiert werden soll, werden zu **InteractableObjects**.

#### 6.2.3.1 InteractableObject

Ein interaktives Objekt bekommt einen Collider, ein RigidBody, das XR Grab Interactable und das InteractableObject Skript. XR Grab Interactable ist im Zusammenhang mit dem XR Direct Interactor für das **Grabbing** zuständig. Der Fokus hier liegt dann bei dem InteractableObject Skript, der alle interaktiven Objekte kennzeichnet. So würde beispielsweise die Struktur für ein simples interaktives Objekt aussehen, welches gehoben werden kann, aber keine weitere Funktionalität besitzt:

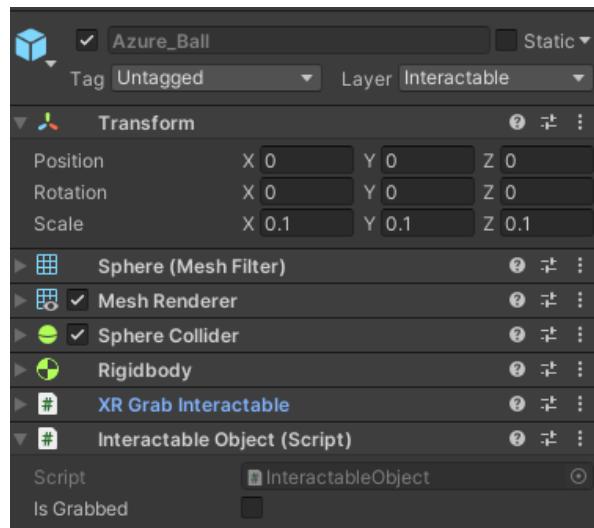


Abbildung 6.5: Beispiel eines InteractableObjects

Im Endeffekt ist `InteractableObject` mehr eine Basis-Klasse für komplexere Konzepte, abhängig vom Anwendungsfall. Somit können Objekte spezifische `InteractableObjects` besitzen, um komplexe Interaktion durchzuführen. Logisch gesehen können trotzdem alle Objekte mit `InteractableObject` aufgerufen werden. Die Klasse selbst hat folgenden Aufbau:

```

1  public class InteractableObject : MonoBehaviour
2  {
3      [SerializeField] private bool isGrabbed = false;
4
5      private LayerMask lmNotGrabbable = 0;
6      private LayerMask lmGrabbable = ~0;
7
8      //Setzt ob das Objekt aufgehoben werden kann,
9      //in dem es die LayerMask seines XRGrabInteractable Skripts ändert
10     public void SetGrabbable(bool grab)
11     {
12         XRGrabInteractable xrObject = gameObject.GetComponent<XRGrabInteractable>();
13         xrObject.interactionLayerMask = grab ? lmGrabbable : lmNotGrabbable;
14     }
15
16     public virtual void OnDrop() { }
17
18     public bool GetIsGrabbed()
19     {
20         return isGrabbed;
21     }
22
23     public virtual void SetIsGrabbed(bool isg)
24     {
25         this.isGrabbed = isg;
26         if (!isg) OnDrop();
27     }
28 }
```

Auflistung 6.19: Code für die InteractableObject Klasse

### 6.2.3.2 TouchObject

Das TouchObject erbt von InteractableObject und ist die simpelste Implementierung eines interaktiven Objekts. Das Ziel ist, ein anderes Objekt (*touchTarget*) mit diesem Objekt zu berühren. Anschließend wird die Task beendet, dies passiert mit dem Aufruf der *NextTask()* Funktion aus dem **ProcessHandler**. Es wird *OnCollisionExit* und nicht *OnCollisionEnter* verwendet, damit die Task erst beendet wird, nachdem das Zielobjekt nicht mehr berührt wird. Das Ziel muss zuerst mit der Setter-Methode gesetzt werden, damit die Funktionalität überhaupt stattfinden kann. Diese Methode wird von dem entsprechenden Task aufgerufen.

```

1  public class TouchObject : InteractableObject
2  {
3      private GameObject touchTarget;
4      private void OnCollisionExit(Collision collision) {
5          if (collision.gameObject.Equals(touchTarget))
6          {
7              ProcessHandler.Instance.NextTask();
8              touchTarget = null;
9          }
10     }
11     // Setzt das Ziel des TouchObjects
12     public void SetTouchTarget(GameObject touchTarger) => touchTarget = touchTarger;
13 }
```

Auflistung 6.20: Code für die TouchObject Klasse

### 6.2.3.3 CompoundObject

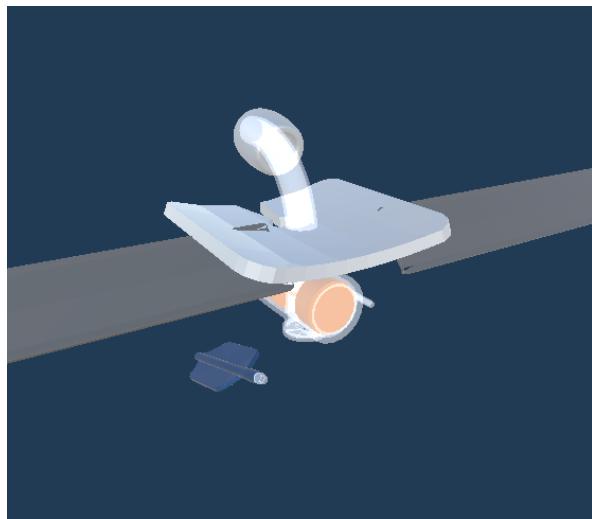


Abbildung 6.6: Ein Compound Object

Für das Zusammenfügen von Objekten wurden zwei Klassen geschrieben. Das CompoundObject ist kein InteractableObject. Es dient nur für die logische Zuordnung mehrerer Teile eines Objekts. Es speichert die einzelnen Teile in einer **Queue**. Wenn ein Objekt benötigt wird, wird es aus der Queue entfernt (*Dequeue*). Dieses Skript ist nur für die Speicherung zuständig, die Task behandelt die Einzelteile. In der oberen Abbildung 6.6 sieht man das CompoundObject für die Kanüle mit allen Elementen dran. Mit den entsprechenden Tasks können dann die einzelnen Teile entfernt werden.

```

1  public class CompoundObject : MonoBehaviour
2  {
3      [SerializeField] private GameObject[] parts;
4      [SerializeField] private GameObject grabbablePart;
5      private Queue<GameObject> objectQueue = new Queue<GameObject>();
6      void Awake()
7      {
8          foreach (GameObject go in parts) {
9              go.GetComponent<InteractableObject>().SetGrabbable(false);
10             objectQueue.Enqueue(go);
11         }
12     }
13     public GameObject GetPart()
14     {
15         if(objectQueue.Count>0) return objectQueue.Dequeue();
16         return null;
17     }
18     [...]
19 }
```

Auflistung 6.21: Code für die CompoundObject Klasse

#### 6.2.3.4 CompoundPart

Ein CompoundPart erbt von InteractableObject und ist Teil eines **CompoundObjects**. CompoundPart Objekte werden in einem CompoundObject gespeichert und wenn der entsprechende Task ein Teil entfernen soll, wird das **taskFocus** Attribut auf *true* gesetzt. Der Task gilt als beendet, wenn das CompoundPart Objekt losgelassen wird. Da wird natürlich impliziert, dass damit ein Objekt losgelassen wird, muss es zuerst gehoben werden. Nachdem das Objekt losgelassen wird, wird es auch nicht mehr kinematisch. Das Skript sieht wie folgt aus:

```

1  public class CompoundPart : InteractableObject
2  {
3      [SerializeField] private bool taskFocus;
4      void Awake() => taskFocus = false;
5      // Wird aufgerufen wenn das Objekt losgelassen wird
6      public override void OnDrop()
7      {
8          base.OnDrop();
9          if (taskFocus)
10          {
11              ProcessHandler.Instance.NextTask();
12              taskFocus = false;
13              GetComponent<Rigidbody>().isKinematic = false;
14          }
15      }
16      public void SetTaskFocus(bool focus) => taskFocus = focus;
17 }
```

Auflistung 6.22: Code für die CompoundPart Klasse

### 6.2.3.5 ConnectorObject

Um mehrere Teilobjekte zu einem Objekt zusammenzufügen, braucht man ein ConnectorObject. Diese Klasse erbt von InteractableObject und gilt als „Hauptobjekt“ oder „Zielobjekt“ bei einer Verbindung. Ein ConnectorObject beinhaltet einen oder mehrere **Anchorpoints**, welche einfache leere GameObjects sind, mit einer vordefinierten Position. Wie schon mal zuvor ist hier nur das Transform relevant. Die Anchorpoints werden in eine **Queue** zusammengetan, deswegen muss man beim Erstellen eines ConnectorObjects auf die Reihenfolge der Anchorpoints zu achten!

Beim Verbinden in der *Connect* Funktion wird das **Connectible** übernommen. Das Objekt wird kinematisch gestellt und seine Collider deaktiviert. Weiters wird das Parent des Teilobjektes auf den obersten Anchorpoint in der Queue gesetzt. Anschließend wird das Transform des Connectible Objektes zurückgesetzt und die Task beendet.

Noch bevor das Connectible verbunden wird, haben wir ein **Preview** hinzugefügt. Das heißt, während der Nutzer das Connectible Objekt in der Hand hält, sieht man auf der richtigen Position auf dem ConnectorObject (Zielobjekt) ein Preview. Das Preview ist im Wesentlichen eine Kopie des Connectibles welches auf der richtigen Anchorpoint Position instanziert wird und mit einem entsprechenden Material versehen wird. Wenn sich das Objekt außerhalb der Verbindungsreichweite befindet, ist das Preview rot, ansonsten wird es grün. So kann man leicht feststellen, wo man ein Objekt verbinden muss, und wann man es loslassen kann.

**Wichtig!** Das ConnectorObject kann nur funktionieren, wenn es aktiviert wird. Das passiert durch das Setzen des *connectorActive* Attributs. Das wird der entsprechende Task dann durchführen.

```

1  public class ConnectorObject : InteractableObject
2  {
3      public bool connectorActive = false;
4      public GameObject preview = null;
5      public GameObject[] anchorPoints;
6      public Queue<GameObject> anchorQueue;
7      public void Awake()
8      {
9          anchorQueue = new Queue<GameObject>();
10         foreach (GameObject ap in anchorPoints) anchorQueue.Enqueue(ap);
11     }
12     public virtual void Connect(GameObject connectible)
13     {
14         if (!connectible.GetComponent<InteractableObject>().GetIsGrabbed() &&
15             connectorActive && this.GetIsGrabbed())
16         {
17             connectible.transform.parent = anchorQueue.Dequeue().transform;
18             connectible.GetComponent<Rigidbody>().isKinematic = true;
19             foreach (Collider collider in
20                 connectible.GetComponentsInChildren<Collider>()) collider.enabled =
21                 false;
22             connectible.GetComponent<InteractableObject>().SetGrabbable(false);
23             connectible.transform.localPosition = new Vector3(0, 0, 0);
24             connectible.transform.localEulerAngles = new Vector3(0, 0, 0);
25             connectible.GetComponent<Connectible>().SetConnected(true);
26             this.connectorActive = false;
27         }
28     }
29 }
```

```

24         ProcessHandler.Instance.NextTask();
25         DestroyPreview();
26     }
27 }
28 [...]
29 // Startet den Preview (Rote Vorzeige)
30 public virtual void StartPreview(GameObject prefab)
31 {
32     if (preview == null)
33     {
34         preview = Instantiate(prefab);
35         preview.GetComponent<InteractableObject>().SetGrabbable(false);
36         preview.GetComponent<Rigidbody>().isKinematic = true;
37         foreach (Component comp in preview.GetComponents<Component>())
38         {
39             // Rigidbody must not be destroyed, else crashes
40             if (!(comp is Transform) && !(comp is Rigidbody))
41             {
42                 Destroy(comp);
43             }
44         }
45         foreach (Collider collider in
46             preview.GetComponentsInChildren<Collider>())
47             Destroy(collider);
48         PreviewFar();
49         preview.transform.parent = anchorQueue.Peek().transform;
50         preview.transform.localPosition = new Vector3(0, 0, 0);
51         preview.transform.localEulerAngles = new Vector3(0, 0, 0);
52     }
53 }
54 [...]
55 }
```

### 6.2.3.6 Connectible

Anschließend gibt es das Connectible Objekt, welches auch von InteractableObject erbt. Das Objekt dient als Teilobjekt und ist für eine Verbindung zu einem **ConnectorObject** gedacht. Wenn dieses Objekt gehoben wird, startet es eine *Coroutine*, welche regelmäßig die Distanz zum Zielobjekt misst. Dieser Vorgang dient für das Erstellen des Previews sowie das Setzen des korrekten Materials (rot wenn nicht in Reichweite, und grün wenn in Reichweite). Der Connector (das Zilobjekt) wird mit der *SetConnector* Methode von dem dazugehörigen Task gesetzt.

Wenn dieses Connectible Objekt losgelassen wird (und der Connector gesetzt ist) und sich innerhalb der Verbindungsreichweite (gespeichert im *range* Attribut) befindet, wird die *Connect* Funktion des ConnectorObjects aufgerufen. Das bedeutet, dass das ConnectorObject endgültig für das Abschließen des Tasks zuständig ist.

```

1  public class Connectible : InteractableObject
2  {
3      [...]
4      public override void OnDrop()
5      {
6          base.OnDrop();
```

```

7         if (connector != null)
8     {
9         if ((connector.GetAnchorPosition() -
10            → this.gameObject.transform.position).magnitude < range)
11         {
12             connector.Connect(this.gameObject);
13         }
14     }
15     public override void SetIsGrabbed(bool isg)
16     {
17         base.SetIsGrabbed(isg);
18         if (connector != null)
19         {
20             if (isg)
21             {
22                 StopCoroutine(CheckDistance());
23                 StartCoroutine(CheckDistance());
24                 connector.StartPreview(this.gameObject);
25                 numGrabbed++;
26             }
27             else
28             {
29                 numGrabbed--;
30                 if (numGrabbed == 0)
31                 {
32                     StopCoroutine(CheckDistance());
33                     connector.DestroyPreview();
34                 }
35             }
36         }
37     }
38     public void SetConnector(ConnectorObject connector)
39     {
40         this.connector = connector;
41         this.SetGrabbable(true);
42         if (GetIsGrabbed() && connector != null)
43         {
44             connector.StartPreview(this.gameObject);
45             StartCoroutine(CheckDistance());
46         }
47     }
48     [...]
49     // On each Frame check if the Connectible Object is in Range of the
50     → ConnectorObject
51     IEnumerator CheckDistance()
52     {
53         for(; GetIsGrabbed();)
54         {
55             if (connector != null)
56             {
57                 if ((connector.GetAnchorPosition() -
58                    → this.gameObject.transform.position).magnitude < range)

```

```
57     {
58         if (!inReach)
59         {
60             connector.PreviewClose();
61             inReach = true;
62         }
63     }
64     else
65     {
66         if (inReach)
67         {
68             connector.PreviewFar();
69             inReach = false;
70         }
71     }
72 }
73 yield return new WaitForEndOfFrame();
74 }
75 [...]
76 }
77 }
```

## 6.3 Graphical User Interface

In diesem Kapitel geht es um die Implementierungen der GUI und mögliche Probleme, Lösungen und Änderungen, die während der Entwicklung aufgetreten sind.

### 6.3.1 Statusanzeige

#### 6.3.1.1 Board

Da die Darstellung der Aufgaben für den Anfang (z. B. zum Testen des Aufgabenablaufs) sehr wichtig ist, haben wir mit dem Board begonnen. Die Ursprungsidee war es, einen flachen Quader mit einem Corkboard-Material [38] (Pinnwand) zu erstellen. Der Aufbau des Codes sah so aus:

```

1  public class CorkboardHandler : MonoBehaviour
2  {
3      public TextMeshProUGUI task_text; //text
4      public GameObject checkmark; //bild
5
6      // Setzt Text der Aufgabe
7      public void FirstTask(string taskdescription) {}
8
9      // Führt FinishTask und SwitchTask aus
10     public void NewTask(string taskdescription) {}
11
12     // Übergang von Tasks
13     IEnumerator SwitchTask(string taskdescription)
14     {
15         // Alter Task wird hier deaktiviert, der Text wird zum neuen Task geändert
16         → und wieder aktiviert
17     }
18
19     // Stellt einen Task als fertig dar
20     public void FinishTask()
21     {
22         // Änderung der Textfarbe auf Grün und aktivierung des Hakens
23     }

```

Auflistung 6.23: Anfangscode der Pinnwand

Im oben dargestellten Code wird immer nur die aktuelle Aufgabe angezeigt. Wenn diese erledigt wird, ändert sich nur der Inhalt des Textes und nicht das Objekt selbst. Wir entschieden uns für diese Art der Implementierung, da es sehr einfach umzusetzen ist und wenig Zeit benötigt. Mit der Zeit fanden wir aber heraus, dass unser Auftraggeber KWP gerne auch ältere Aufgaben sehen würde und somit musste der Code bearbeitet werden. Bis jetzt gab es immer nur einen Text und ein Bild (grüner Haken) auf der Pinnwand, doch nun werden mehrere Aufgabenstellungen benötigt. Zur Bewältigung dieses Problems gibt es nun ein Prefab für Aufgabenbeschreibungen. Jedes Mal, wenn die derzeitige Aufgabe erledigt wird, bewegt sie sich auf der Pinnwand nach unten und die neue Aufgabe (neues Prefab) erscheint oben. Somit ist die aktuelle Aufgabe immer oben und ist somit einfacher zu erkennen. Die Grundlage des Codes blieb aber gleich:

- eine Methode für die Erstellung der ersten Aufgabe (*FirstTask*)
- eine Methode für die Erstellung aller anderen Aufgaben (*NewTask*)
- Änderung der Farbe des Textes und Aktivierung des grünen Hakens (*FinishTask*)
- Änderung und Verschiebung von Aufgaben (*Taskrotation*) (früher: Änderung der Textbeschreibung -> *SwitchTask*)

```

1  public class WhiteboardHandler : MonoBehaviour
2  {
3      public List<GameObject> tasklist; // Liste der Aufgaben
4      public GameObject task_prefab; // Prefab einer Aufgabe
5      private GameObject task_current; // derzeitiger Task
6      private TextMeshProUGUI task_text_current; //text
7      private SpriteRenderer checkmark_current; //bild
8      private float y_gap = 0.5f; // Y-Abstand zwischen den Tasks
9      private float maxTaskShown = 4; // Max anzahl an Tasks
10     // [...]
11     // Verschiebung der Tasks und das Anzeigen des neuen Tasks.
12     IEnumerator TaskRotation(string taskdescription)
13     {
14         yield return new WaitForSeconds(0.4f);
15         for (int i = tasklist.Count - 1; i >= 0; i--) // Verschiebung der Tasks auf
16             // der Y-Achse
17             {
18                 tasklist[i].transform.position -= new Vector3(0f, y_gap, 0f);
19             }
20             if(tasklist.Count >= maxTaskShown) // Wenn das maximum an Tasks erreicht
21             // ist, wird das älteste Element gelöscht
22             {
23                 Destroy(tasklist[0]);
24                 tasklist.Remove(tasklist[0]);
25             }
26             yield return new WaitForSeconds(0.4f);
27             // Erstellung eines neuen Tasks
28             task_current = Instantiate(task_prefab, transform);
29             tasklist.Add(task_current);
30             task_text_current = tasklist[tasklist.Count -
31             // 1].GetComponentInChildren<TextMeshProUGUI>();
32             checkmark_current = tasklist[tasklist.Count -
33             // 1].GetComponentInChildren<SpriteRenderer>();
34             task_text_current.SetText(taskdescription);
35         }
36         // [...]
37     }

```

Auflistung 6.24: Neuer Whiteboard Code mit mehreren Aufgaben

Im oben aufgelisteten Code gibt es jetzt eine Liste namens *tasklist*, die alle erstellten Prefabs beinhaltet. Somit kann leichter auf alle erstellen Prefabs zugegriffen und verschoben werden. Die

Verschiebungen werden noch manuell getätigt, da wir von keiner besseren Alternative zu diesem Zeitpunkt gewusst haben. Aber nicht nur der Code hat sich geändert, sondern auch das Aussehen der Pinnwand. Sie ist nun komplett weiß, damit die schwarze Schrift besser lesbar ist und heißt nun Whiteboard.

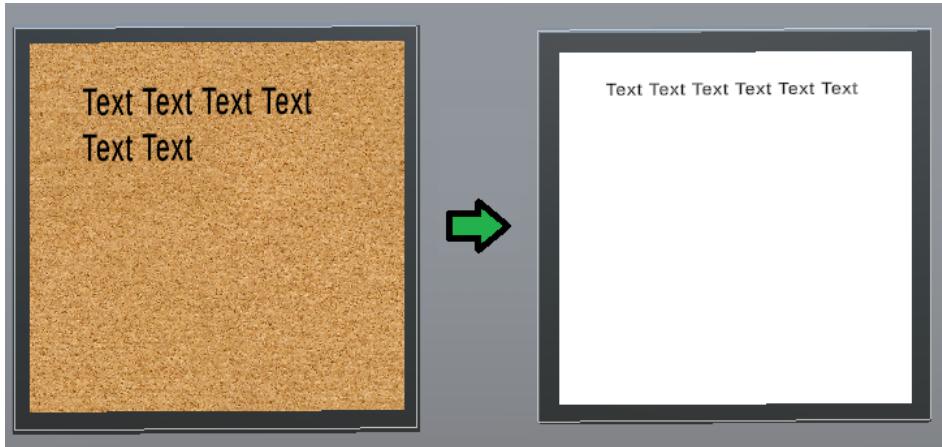


Abbildung 6.7: Alte Pinnwand (links) und neues Whiteboard (rechts)

Mit der Zeit fanden wir aber heraus, dass es auch eine Möglichkeit gibt, die Auflistung der Aufgaben dynamisch darzustellen. Unity bietet eine Funktion namens "Vertical Layout Group", die es ermöglicht, Auflistungen von Elementen dynamisch und einfach zu implementieren. Ein weiteres Problem, dass eine Funktion von Unity gelöst hat, ist der "Content Size Fitter". "Vertical Layout Group" gibt zwar einen Abstand zwischen den einzelnen Objekten an, aber nicht zwischen den Texten. Es kann somit möglich sein, dass ein zu langer Text über die Größe des Objekt-Feldes hinausgeht und innerhalb des nächsten Textes steht. Durch Content Size Fitter passt sich das Objekt an den Text an und verhindert somit dieses Problem.

Schlussendlich wollte unser Auftraggeber die Reihenfolge der Aufgaben umgekehrt haben (von oben nach unten - aktuelle Aufgabe befindet sich unten) und ebenso eine Möglichkeit zwischen den unterschiedlichen Arbeitsabläufen (wie in [5.1.1](#) besprochen) zu wechseln. Um dieser Anfrage nachzukommen, gibt es nun zwei Buttons beim Whiteboard, die für die Auswahl der Arbeitsabläufe zuständig sind. Wir haben schlussendlich doch einen Raycast statt einen Pointer benutzt, da dies einfacher zu implementieren und zeitsparender war. Ebenso war die Genauigkeitsverlust (wie im Studie Kapitel [4.3.3.4](#) besprochen) für diese Benutzungsanwendung nicht relevant. Die Hauptmethode *NewTask* sieht nun so aus:

```

1  public class WhiteboardHandler : MonoBehaviour
2  {
3      // [...]
4      public void NewTask(string taskdescription, bool first) // Neuen Task erstellen
5      {
6          task_current = Instantiate(task_prefab, transform.GetChild(1));
7          task_current.SetActive(false);
8          tasklist.Add(task_current);
9          if (first)
10         {
11             task_text_current =
12                 tasklist[0].GetComponentInChildren<TextMeshProUGUI>();
13             checkmark_current =
14                 tasklist[0].GetComponentInChildren<SpriteRenderer>();
15         }
16         else
17         {
18             task_text_current = tasklist[tasklist.Count -
19                 1].GetComponentInChildren<TextMeshProUGUI>();
20             checkmark_current = tasklist[tasklist.Count -
21                 1].GetComponentInChildren<SpriteRenderer>();
22         }
23         task_text_current.SetText(taskdescription);
24
25         Color tmp = checkmark_current.GetComponent<SpriteRenderer>().color; // Änderung der opasity auf 0 -> 0%
26         tmp.a = 0f;
27         checkmark_current.GetComponent<SpriteRenderer>().color = tmp;
28
29         if (tasklist.Count > maxTaskShown) // Wenn das maximum an Tasks erreicht
30             ist, wir das älteste Element gelöscht
31         {
32             Destroy(tasklist[0]);
33             tasklist.Remove(tasklist[0]);
34         }
35         task_number++;
36     }
37     // [...]
38 }
```

Auflistung 6.25: Endversion des Whiteboard Codes

Alle Methoden vom *WhiteboardHandler* werden im Script *UserInterfaceManager* aufgerufen, wie in Kapitel [User-Interface Manager](#) erläutert. Nun schaut das Whiteboard so aus:

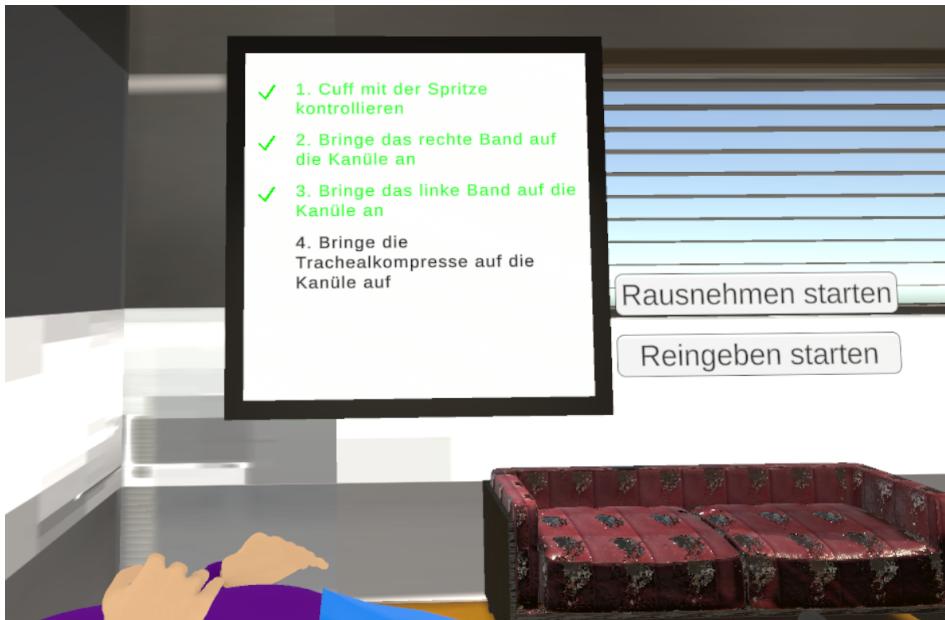


Abbildung 6.8: Whiteboard - Endstand

### 6.3.1.2 Darstellung von User Feedback

In der Planungsphase haben wir die Darstellung des Patienten-Zustandes und die Audio-Darstellungen (wie im Kapitel 5.3.1.2 erwähnt) besprochen, doch diese Features sind nicht implementiert worden. Der Zustand des Patienten macht erst dann einen Sinn, wenn der Patient auch Emotionen zeigen kann. Da aber das Modellieren schon mehr Zeit benötigt als erwartet, werden fürs Erste auch keine Emotionen implementiert und somit verfällt diese Idee. Für die Audiodarstellung gibt es ebenfalls keinen Bedarf mehr, da außer ein paar Soundeffekte sonst keine Sounds oder Geräusche in der Applikation existieren.

Nach einigen Testläufen des Aufgabensystems fiel uns auf, dass eine visuelle Benachrichtigung für den Benutzer hilfreich wäre, wenn die derzeitige Aufgabe abgeschlossen wird. Nun erscheint ein Hinweis im HUD und verschwindet nach kurzer Zeit wieder [81]. Der Code für diese Darstellung sieht so aus:

```

1  public class HUDHandler : MonoBehaviour
2  {
3      public TextMeshProUGUI hud_text; // Angezeigte Text
4
5      public void ShowText() // Darstellung des grünen Textes "Aufgabe
6          ↪ abgeschlossen". Wird immer dann ausgeführt, wenn eine Aufgabe abgeschlossen
7          ↪ wird.
8      {
9          StopAllCoroutines();
10         StartCoroutine(Fade_text(0.05f, 0.12f));
11     }
12     IEnumerator Fade_text(float wait, float value) // Fade Darstellung des Textes
13     {
14         while (hud_text.color.a < 1.0f) // Fade in
15         {
16             hud_text.color = new Color(hud_text.color.r, hud_text.color.g,
17                 ↪ hud_text.color.b, hud_text.color.a + value);
18             yield return new WaitForSeconds(wait);
19         }
20         yield return new WaitForSeconds(1.3f);
21         while (hud_text.color.a > 0.0f) // Fade out
22         {
23             hud_text.color = new Color(hud_text.color.r, hud_text.color.g,
24                 ↪ hud_text.color.b, hud_text.color.a - value);
25             yield return new WaitForSeconds(wait);
26         }
27     }
28 }

```

Auflistung 6.26: Code für Darstellung des Textes "Aufgabe abgeschlossen"

Ebenso ist uns bewusst geworden, dass bestimmte Aufgaben nicht detailgetreu implementieren werden können, wie z. B. das Angreifen eines Objektes mit beiden Händen. Damit aber der Benutzer trotzdem weiß, auf was dieser achten muss, haben wir bei solchen Aufgaben einen Hinweis hinzugefügt. Dieser erscheint auf der linken Hand und erklärt, wie bestimmte Aufgaben ausgeführt werden sollten, wie z. B. "Beide Hände verwenden". Für dieses Feature gibt es ein Script und dieses beinhaltet folgende Codezeilen:

```

1  public class HandHandler : MonoBehaviour
2  {
3      // [...]
4      public GameObject warning; // on default disabled
5      public TextMeshProUGUI[] warning_texts;
6      Image warning_sign;
7      Image warning_image;
8
9      public void Warning(int warningIndex) // Wird aufgerufen, wenn UI 'Beide Hände
10     ↪ benutzen' anzeigen soll
11     {
12         StartCoroutine(ShowWarning(warningIndex, 0.1f, 0.12f));
13     }
14     /**
15      * warningIndex: Welche Warning dargestellt wird
16      * wait: Länge der Wartezeit zwischen fade
17      * value: Opasity Änderung per Durchgang
18      */
19     IEnumerator ShowWarning(int warningIndex, float wait, float value)
20     {
21         warning_texts[warningIndex].gameObject.SetActive(true);
22         for (int i = 0; i < 3; i++)
23         {
24             while (warning_image.color.a < 1.0f &&
25                 ↪ warning_texts[warningIndex].color.a < 1.0f) // fade in
26             {
27                 // Änderung der Opasity auf 1 von warning_text, warning_sign und
28                 ↪ warning_image
29             }
30             while (warning.GetComponent<Image>().color.a > 0.0f &&
31                 ↪ warning_texts[warningIndex].color.a > 0.0f) // fade out
32             {
33                 // Änderung der Opasity auf 0 von warning_text, warning_sign und
34                 ↪ warning_image
35             }
36             yield return new WaitForSeconds(1f);
37         }
38     }
39 }

```

Auflistung 6.27: Code zur Darstellung eines gewählten Hinweises



Abbildung 6.9: Hinweis - Grafische Darstellung

### 6.3.2 Hauptmenü und Loading Screen

Für das Hauptmenü sind viele unterschiedliche Anordnungen und Möglichkeiten ausprobiert worden und am Ende entschieden wir uns für eine einfache Gliederung mit einer Überschrift und zwei Buttons: "Start" und "Exit". Dies kommt der Idee 1 nahe, was im Kapitel 5.3.2 ange- sprochen wird. Die Möglichkeit, bestimmte Optionen einzustellen und die visuelle Darstellung des Patienten wurde entfernt, da diese für das Benutzen der Applikation nicht notwendig sind und nur einen weiteren Aufwand bedeutet hätte. In Abbildung 6.10 ist das simple Hauptmenü dargestellt:



Abbildung 6.10: Hauptmenü

Damit aber trotzdem die Controller-Belegung angesehen werden kann, wird dies innerhalb der Hauptszene implementiert. Dieser ist mit dem Sekundär-Button aktivierbar und befindet sich auf derselben Stelle, wie die Hinweise. Somit ist der Code auch im selben Script:

```
1 public class HandHandler : MonoBehaviour
2 {
3     public GameObject controls; // on default disabled
4     public GameObject right_ray_interactor; // on default enabled
5     public GameObject left_ray_interactor; // on default enabled
6     public InputActionReference toggleReferenceControl_right = null;
7     // [...]
8     private void Awake()
9     {
10         // [...]
11         toggleReferenceControl_right.action.started += ShowControls;
12     }
13     private void OnDestroy()
14     {
15         toggleReferenceControl_right.action.started -= ShowControls;
16     }
17     public void ShowControls(InputAction.CallbackContext context) // Wird
18     ↵ aufgerufen, wenn der Button für toggleReferenceControl_right gedrückt wird
19     ↵ -> siehe Samples/Default Input Actions/XRI Default Input Actions
20     {
21         bool isActive = !controls.activeSelf;
22         controls.SetActive(isActive);
23
24         var right_hand_ray = right_ray_interactor.GetComponent<UnityEngine.XR.Interaction.Toolkit.XRInteractorLineVisual>(); // Deaktiviert die Rays beim
25         ↵ Aktivieren des Control-Help-Panels
26         right_hand_ray.gameObject.SetActive(!isActive);
27     }
28 }
```

Auflistung 6.28: Code zur Anzeige von der Controllerbelegung

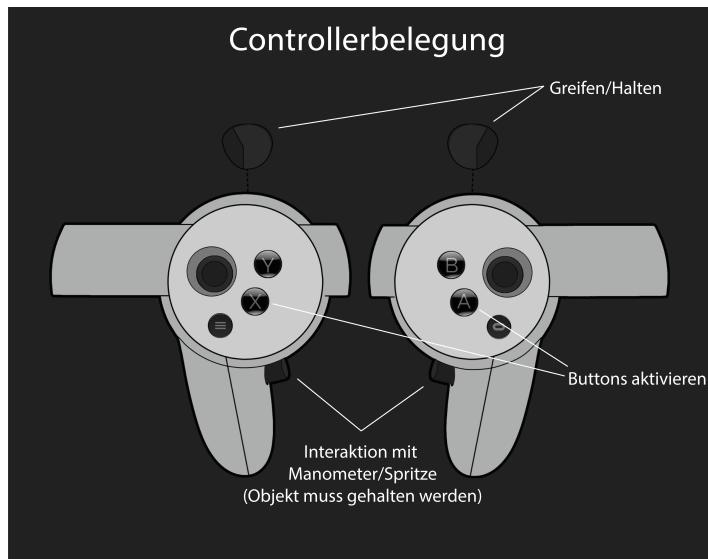


Abbildung 6.11: Controllerbelegung [59]

Damit dies funktioniert, haben wir das “XR Interaction Toolkit” von Unity benutzt. Weitere Informationen darüber wird im Kapitel [6.3.3.1](#) erläutert.

### 6.3.3 Weitere Implementierungen

#### 6.3.3.1 Input System

Da wir mit VR-Controllern von Oculus 2 arbeiten, können wir deren Inputs nicht so einfach empfangen und mit diesen arbeiten, wie z. B. mit `Input.getButtonDown()` bei Tastaturinputs. Einfache Tasten, wie der Greif-Button (Hinterer Button des Controllers) funktionieren zwar automatisch mit dem “XR Rig” (= Default Objekte der Camera, Linke Hand, Rechte Hand usw.), doch für spezielle Buttoninputs wird eine andere Möglichkeit benötigt. Hierfür wird ein “XR Interaction Toolkit” benötigt, was von Unity zur Verfügung gestellt wird:

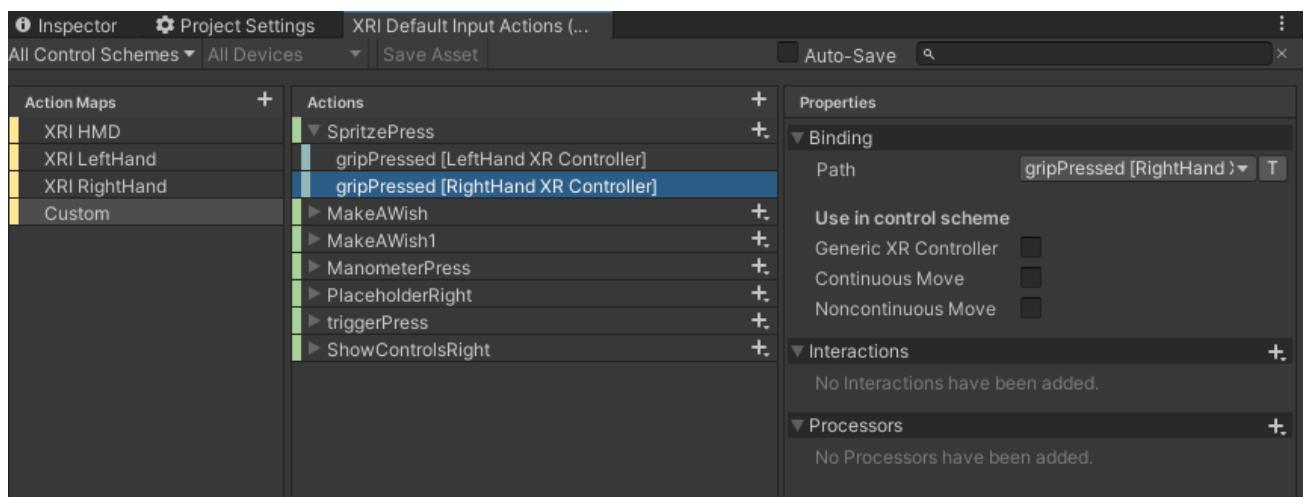


Abbildung 6.12: XRI Default Input Actions in Unity

“Action Maps” sind Ordner, die fürs strukturierte Organisieren unterschiedlicher “Actions” zuständig sind, welche wiederum unterschiedlich viele “Bindings” enthalten. In einer “Action” kann z. B. die Art des Inputtyps ausgewählt werden, wie “Value” (mehrere Werte möglich) oder “Button” (On - Off: benutzerter Inputtyp in diesem Projekt). Jedes “Binding” beinhaltet mehreren Einstellungen, doch die wichtigste ist die der Button Auswahl. In Abbildung 6.12 wird hier der Button “gripPressed” von der rechten Hand ausgewählt (Seiten-Button des Controllers).

### 6.3.3.2 Animation von Manometer und Spritze

Da wir jetzt eine Möglichkeit haben, auf alle Inputs vom Oculus 2 Controller zuzugreifen, können wir auch mit dem Manometer und der Spritze interagieren. Um diese nämlich zu benutzen, musste ein weiterer Button während des Haltens gedrückt werden.

Für das Manometer musste eine Animation implementiert werden, wo die Nadel des Ziffernblattes bewegt wird. Der Code dafür schaut wie folgt aus:

```

1  public class ManometerMovement : PressObject
2  {
3      public GameObject nadel;
4      public InputActionReference toggleReference = null; // Action des Seiten-Buttons
        ↪ vom Controllers
5
6      private void Awake()
7      {
8          toggleReference.action.started += Toggle;
9      }
10     private void OnDestroy()
11     {
12         toggleReference.action.started -= Toggle;
13     }
14     private void Toggle(InputAction.CallbackContext context) // Wird aufgerufen,
        ↪ wenn der Button für toggleReference gedrückt wird -> siehe Samples/Default
        ↪ Input Actions/XRI Default Input Actions
15     {
16         StopAllCoroutines();
17         StartCoroutine(Druecken());
18     }
19     // [...]
20 }
```

Auflistung 6.29: Code einer Implementierung des Input Systems

In der *Awake*-Methode wird ausgewählt, welche Methode aufgerufen wird, wenn der Button gedrückt wird. Dieser startet dann die “Coroutine” (=Abk. Methode mit der Möglichkeit, den Code für eine gewisse Zeit zu pausieren) *Druecken*:

```

1  public class ManometerMovement : PressObject
2  {
3      // [...]
4      private void Update()
5      {
6          if (nadel.transform.localRotation.eulerAngles.y > 32f)
7          {
8              nadel.transform.Rotate(new Vector3(0, -23f * Time.deltaTime, 0));    // 
9                  ↪ Druck entlassen
10         }
11     }
12     IEnumerator Druecken() // Bewegung der Nadel
13     {
14         if (this.GetComponent<InteractableObject>().GetIsGrabbed())
15         {
16             enabled = false; // stoppt Script, damit update-function gestoppt wird
17                 ↪ -> Druecken wird aber weiter ausgeführt (warum auch immer - bissl
18                 ↪ russisch)
19             for (int i = 0; i < 50; i++)
20             {
21                 if (nadel.transform.localRotation.eulerAngles.y >= 330) { break; }
22                 if (nadel.transform.localRotation.eulerAngles.y >= 140)
23                 {
24                     Press();
25                     GetComponent<ConnectorObject>().Disconnect();
26                     break;
27                 }
28                 nadel.transform.Rotate(new Vector3(0, 1, 0));    // Druck hinzufügen
29                 yield return new WaitForSeconds(0.007f);
30             }
31             enabled = true; // aktiviert Script wieder
32         }
33     }
34 }

```

Auflistung 6.30: Code für die Manometer-Animation

Bei der Spritze musste ebenfalls eine Animation werden und bis auf ein paar Änderungen agiert sie so, wie das Manometer:

```

1  public class SpritzePressObject : PressObject
2  {
3      public InputActionReference toggleReference = null;
4      public bool reingepumpt = false;
5      public bool nurRauspumpen = false;
6      public GameObject kolben;
7      private Animator anim;
8
9      // [...]
10
11     public override void Press()
12     {
13         if (!pressable) return;
14         GetComponent<ConnectorObject>().Disconnect();    //disconnect object
15         if (!nurRauspumpen)
16         {
17             GameObject temp = GameObject.FindGameObjectWithTag("CompoundGrabbablePa ]
18             ↵ rt").transform.parent.parent.gameObject;
19             temp.GetComponent<InteractableObject>().SetGrabbable(true);
20             temp.GetComponent<Rigidbody>().isKinematic = false;
21         }
22         base.Press();    //next task
23     }
24     IEnumerator Reinpumpen() // Start der "Reinpumpen" Animation
25     {
26         anim.SetTrigger("reinpumpen");
27         yield return new WaitForSeconds(1.1f);
28         reingepumpt = true;
29     }
30     IEnumerator Rauspumpen() // Start der "Rauspumpen" Animation
31     {
32         anim.SetTrigger("rauspumpen");
33         yield return new WaitForSeconds(1.3f);
34         Press();
35     }
}

```

Auflistung 6.31: Code für die Spritze-Animation

Es wird die Bewegung nicht manuell (wie beim Manometer) geändert, sondern mit einer Animation ausgeführt, wo der Kolben nach unten bzw. nach oben geschoben wird. Diese werden mit *anim.SetTrigger()* ausgeführt.

## 6.4 3D Modellierung von Patientenzimmer und Props

### 6.4.1 Behandlungszimmer

Es wurde eine Vorlage des Raumes erstellt. Dabei stellen Würfel im Inneren das Innenleben dar und gebender Position und Rotation an. Fenster und andere Details werden ebenfalls angegeben.

Aufgrund der steigenden Komplexität der **Props**, sowie deren Priorität, suchten wir nun nach einem Modell, das sowohl Möbel als auch Texturen enthält. Mindestanforderungen sind hierbei ein Bett und ein Regal, auf dem die Props platziert werden können.

Das gefundene Modell wird so angepasst, dass das Regal und das Bett beisammen stehen und das VR-Rig so platziert werden kann, das dieses beides in Reichweite hat.

### 6.4.2 Modellierung **Props**

Die **Props** stellen das Wichtigste des Prozesses momentan dar. Diese sollen so gut wie möglich erkennbar sein. Dazu gehören natürlich nicht nur die eigentliche Modellierung, sondern auch Texturierung und Animation. Die Props stellen die höchste Priorität im Bereich der Modellierung dar..

Die Modelle werden in der 3D-Modellierungssoftware Maya erstellt und werden, sofern keine komplexen Texturen verwendet, auch dort texturiert. Ansonsten werden dazu noch die Unity-Materials verwendet. Exportiert werden die Modelle bevorzugt über das FBX-Format.

### 6.4.3 Spritze

#### 6.4.3.1 Aufbau des Meshes

##### Zylinder

Begonnen wird mit dem Außenzyylinder der Spritze. Dabei wird ein längerer Zylinder erstellt, dupliziert, auf der X und Z Achse kleiner skaliert und in die Mitte des ersten Zylinders gesteckt. Dann muss zuerst der äußere und dann der kleinere ausgewählt werden und mittels *Mesh > Booleans > Difference* ein Durchgang geschaffen. Dabei wird aber darauf geachtet, dass eine kleine Fläche am Boden übrig bleibt.

Als nächsten Schritt wird dann die Öffnung am Boden gemacht. Hierfür wird der oben benutzte Vorgang wiederholt. Dieses Objekt wird dann so klein skaliert, dass es die richtige Proportion zur Spritze bekommt. Dieses wird dann mit der *Wireframe-Ansicht* gut platziert und dem Spritzen Mesh mit *Mesh > Booleans > Union* angefügt. Mit einem Zylinder, welcher ca. so groß wie der Durchgang dieser Öffnung ist, wird durch den Boden ein weiterer Durchgang geschaffen.

Mit einem Würfel wird dann der Griff, der an die große Öffnung für den Kolben an angebracht wird, modelliert. Dieser wird so in den Seiten skaliert, dass es ein dünnes Rechteck wird. Mit einem Cut in der Mitte können dann die Kanten an den Seiten enger zusammengezogen werden. Mit dem Bevel-Tool werden die Kanten alle abgerundet. Bei den Seiten werden dabei zwölf neue Divisions eingearbeitet, um die Kanten gut abzurunden. In der Mitte wird es auf fünf ausgeweitet. In der Mitte wird nochmals ein Zylinder mit dem Difference-Boolean ein Loch in das Objekt zu geben. Mit dem Union-Boolean wird das Mesh des Griffes an das Mesh der Spritze angehängt. (Abb. 6.13)

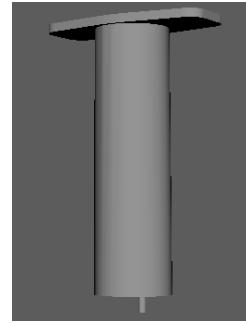


Abbildung 6.13: Mesh des Zylinders

## Kolben

Als nächsten Schritt wird dann der Kolben der Spritze gebaut. Hier wird mit einem Würfel gestartet, wobei dieser vorerst in die Länge skaliert wird. Mit dem *Extrude-tool* macht man dann aus dem Quader eine Kreuzform, indem man vier Seiten auswählt und diese dann herauszieht. Dabei muss man achten, dass man bei dem Tool die verbundenen Kanten ausschaltet, damit auch wirklich nur die Kreuzform übrigbleibt. Die Flächen an den Seiten werden dann breiter gemacht, damit eine gewisse Schräge in die Kreuzform gegeben wird.

Mit dem *Multicuttool* werden dann an ein paar Stellen Subdivisions eingeschnitten. Drückt man mit dem Tool den *Shift-Key*, kann das Programm bei einfachen Strukturen von selbst Subdivisions cutten, wodurch man sich komplexes Schneiden ersparen kann. Dann wählt man die Kanten dieser Subdivisions und verschiebt diese. Mit dem *Bevel-Tool* gibt man dem ganzen dann noch abgerundete Kanten, wodurch es nochmal etwas realistischer aussieht.

Als letzten Schritt erstellt man nun 2 dünne Zylinder und gibt diese an die beiden Enden des vorher bearbeiteten Meshes. Dabei achtet man darauf, dass der Zylinder, welcher in die Spritze fährt etwas kleiner ist, als der andere. Diese verknüpft man mit dem *Union-Tool* zu einem Mesh zusammen. (Abb. 6.14)

Um das Ganze nun als ein Objekt ansprechen zu können, wählt man beide Meshes aus und drückt *STRG + G* oder unter *Edit > Group*. Folgend ist auch die Darstellung des gesamten *Props*.

### 6.4.3.2 Texturierung

#### Zylinder

Der Zylinder bekommt ein Unity-Material, um die Transparenz darzustellen. Da kein metallischer Shader erstellt werden soll, stellt man den Workflow Mode auf Specular. Um die Transparenz nun einzubauen, wird der Surface type auf Transparent und der Blending Mode auf Additive gestellt. Die Base-Map bleibt weiß da, kein farbiges Material benötigt wird.

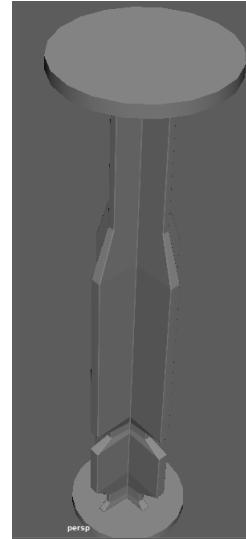


Abbildung 6.14: Mesh des Kolbens

## Kolben

Für den Kolben wird in Maya ein UV-Sheet erstellt. Dieses kann Maya automatisch erstellen lassen, durch Auswahl des Objektes und dem integrierten Tool unter *UV > Automatic*. Dieses kann man sich auch unter dem Workspace *UV Editing* ansehen und bearbeiten. Maya unterstützt ein eigenes Zeichentool (*3dPaintTool*), mit dem es möglich ist, Texturen an dem Objekt zu zeichnen. Der Kolben bekommt ein für die Spritze übliches Türkis.

Die Map sollte sich meistens mit dem FBX-Export mitabspeichern und somit an Unity übergeben werden können. Dies hat aus unbekanntem Grund nicht funktioniert, weshalb die Map direkt aus dem Workspace Ordner (*sourceimages > 3dPaintTextures*) herausgenommen werden musste und dann in Unity als Textur importiert werden muss. Diese Textur gibt man dann dem GameObject des Kolbens, der diese als Unity-Material übernimmt. Einstellungen müssen nicht geändert werden, da die Darstellung bereits passt. (Abb. 6.15)

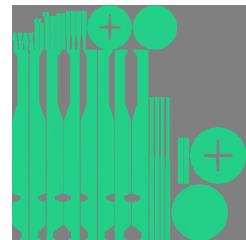


Abbildung 6.15: Map des Kolbens

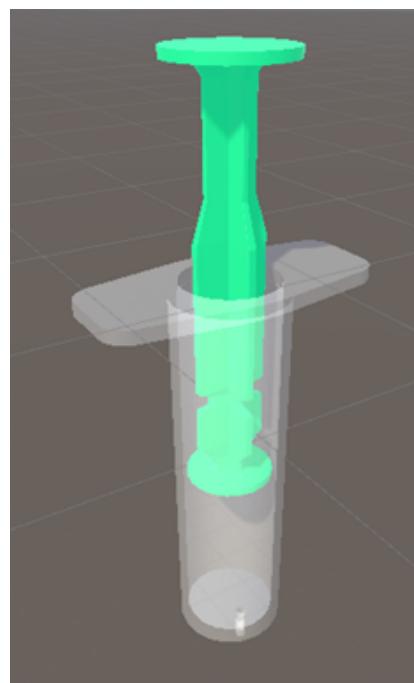


Abbildung 6.16: Fertiges Modell der Spritze

#### 6.4.4 HME (feuchte Nase)

##### 6.4.4.1 Aufbau des Meshes

###### Hülle

Die Außenhülle der feuchten Nase wird mit einem Zylinder-Primitiv begonnen. Diesem geben wir eine angenehme Größe und setzen die Proportionen so, wie sie in der Vorlage gezeigt sind. Dann wird dieses dupliziert und etwas herunterskaliert, sodass nur eine kleine Wand zwischen beiden übrigbleibt. Mit dem *Difference-Tool* setzt man dann einen Hohlraum in den Zylinder.

Als nächsten Schritt wird wieder ein Zylinder genommen. Dieser soll im 90° Winkel durch die Mitte des ersten großen Zylinders gehen. Um die beiden Meshes zusammenzufügen, benutzen wir das *Union-Tool*. Der kleine Zylinder soll wieder einen Durchgang bekommen, was bedeutet, dass wir diesen wieder duplizieren und herunterskalieren und dann mit dem *Difference-Tool* zusammensetzen.

Als nächsten Schritt nehmen wir wieder einen Zylinder mit der Breite des Durchgangs der Außenhülle und setzen es so, dass wir einen Teil des Durchgangs wegnehmen. Dafür benutzen wir wieder das *Difference-Tool*. Mit dem *Extrude-Tool* nimmt man dann die obere und untere Seite des Durchgangs und verbindet beide Seiten, sodass eine Art Brücke entsteht.

Für den O<sub>2</sub>-Port wird wieder ein Zylinder mit Durchgang erstellt, welcher dieses Mal länger und dünner wird. Auf der oberen Seite machen wir einen Cut mit dem *Multi-Cut-Tool* und verengen den Eingang. Mit dem Union-Tool wird der O<sub>2</sub>-Port an das Hauptmesh angefügt. (Abb. 6.17)

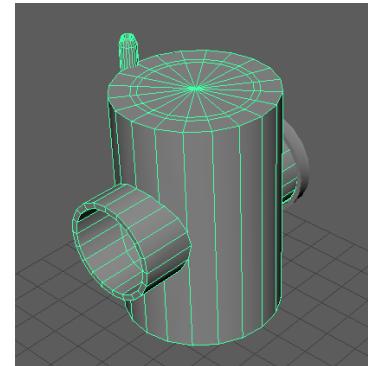


Abbildung 6.17: Mesh der Hülle

###### Kappe und Filter

Die Filter sind sehr einfach, da wir als Form nur zwei identische Zylinder brauchen. Dazu skaliert man diese so, dass ein Filter über dem Durchgang liegen kann und der andere darunter.

Die Kappe wird auch mit einem Zylinder begonnen. Dieser ist etwas breiter als der Durchgang der Hülle, aber sehr dünn. Man baut eine weitere Subdivision ein und verschiebt diese weiter an den äußeren Rand. Mit dem *Extrude-Tool* wird dann die kleinere Subdivision herausgezogen und die gegenüberliegende Fläche nach oben skaliert. Dadurch entsteht eine Schräge. (Abb. 6.18)

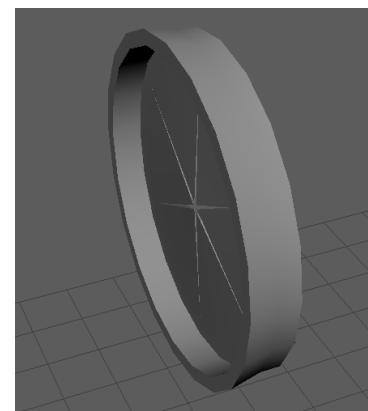


Abbildung 6.18: Mesh der Kappe

#### 6.4.4.2 Texturierung

##### Hülle und Kappe

Für die Hülle und Kappe wird dasselbe durchsichtige Material genommen, wie für den Zylinder der Spritze. Dafür muss man auch keine Änderungen vornehmen, da dieses bereits sehr realitätsgerecht aussieht.

##### Filter

Für den Filter wird ein eigenes Unity-Material erstellt. Dieses Material bekommt eine gelbe Base Map, die ca. die helle gelbe Farbe haben soll, wie der echte Schaumstofffilter. Da Schaumstofftexturen schwierig darzustellen sind und der HME, durch die Größe, gar nicht so viel Detail benötigt, wird ein einfaches farbiges Unity-Material verwendet.

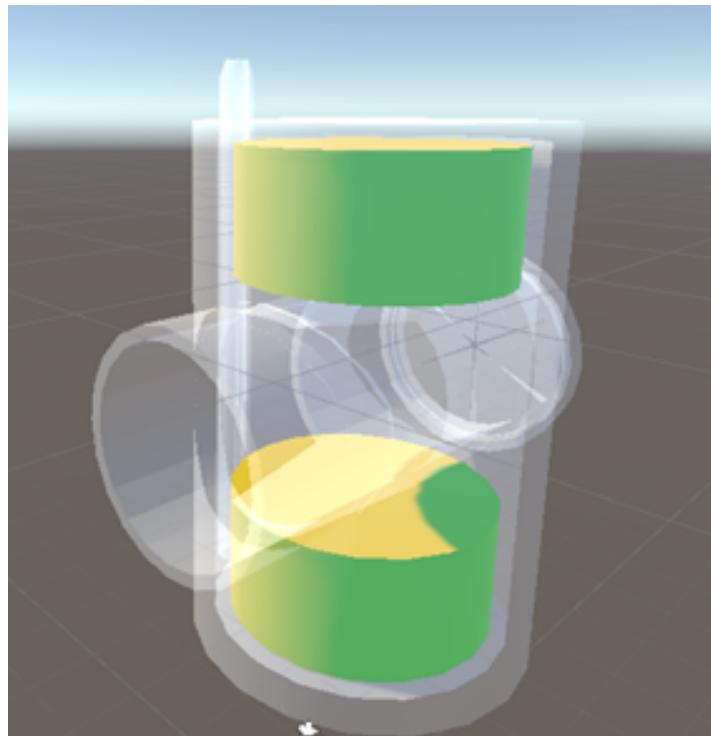


Abbildung 6.19: Fertiges Modell der feuchten Nase

### 6.4.5 Manometer

#### Mittel-Körper

Die Halbkugel als MittelKörper wird wieder mit einem Zylinder gebaut. Die Form muss dabei sehr breit sein. Durch Subdivisions kann man die Trennwand und dann die Vertiefung für das Ziffernblatt erstellen. Dabei wird das *Extrude-Tool* verwendet, um einerseits Teile des Meshes herauszuziehen, aber auch um Vertiefungen im Mesh zu erstellen. Durch weitere Subdivisions kann man dann auch den Halter für die Nadel bauen. (Abb. 6.20)

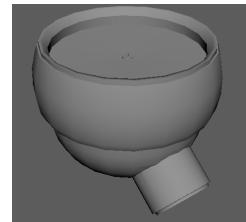


Abbildung 6.20: Mesh des Mittel-Körpers

#### Druckkörper

Dieser wird ebenfalls mit einem Zylinder begonnen. Da wir diesen in einer Ovalform haben wollen, werden mehrere Subdivisions erstellt und diese dann so skaliert, dass eine glatte aber abgerundete Oberfläche entsteht.

Auf einer Seite wird mit dem *Extrude-Tool* dann eine Verbindung geschaffen, mit der der Druckkörper und der Mittel-Körper verbunden wird. Auf der anderen Seite benutzen wir selbiges Tool, um eine kleine Anhebung und in deren Mitte ein Loch in das Mesh zu kreieren.

Als letzten Schritt werden die scharfen Kanten mit einem *Bevel* abgerundet, wobei 3 weitere Divisions verwendet werden. (Abb. 6.21)

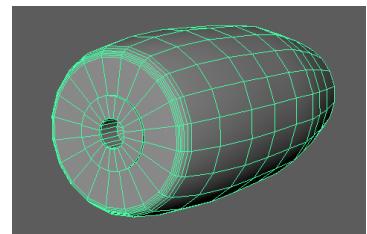


Abbildung 6.21: Mesh des Druck-Körpers

#### Nadel

Die Nadel wird mit einem flachen Zylinder begonnen. Als nächstes Objekt wird dann ein langer Quader aus einem Würfel gebaut und mit *Boolean > Union* an den Zylinder angehängt. Derselben Schritt wird dann mit einem weiteren Zylinder gemacht, der etwas größer als der erste sein muss. Und zuletzt wird wieder ein Quader gebaut, welcher in der Mitte eine Subdivision geschnitten bekommt und eine Seite wird zu einer Spitz verengt. Die breite Seite kommt an den zweiten Zylinder gegenüber des ersten Quaders. (Abb. 6.22)

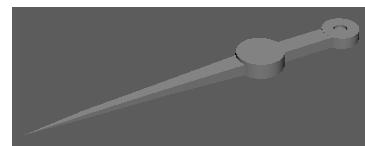


Abbildung 6.22: Mesh der Nadel

### **Erster Versuch: Alles in einem Mesh**

Beim ersten Versuch wurde einen Fehler beim Modellieren gemacht. Dabei wurde das gesamte Manometer durch das *Union-Tool* zu einem Mesh zusammengefügt. Dieses war für Animationen nicht verwendbar, da man weder die Nadel unabhängig rotieren konnte, noch den Druckkörper so manipulieren kann, dass er den Rest des Meshes nicht beeinflusst.

### **Zweiter Versuch: Trennung in 3 Objekte**

Um also das gesamte Mesh zu verbessern, mussten die Einzelteile neu angefangen werden. Dabei musste man auf den Einsatz der Tools achten, da die Extrudes an der Oberseite von außen nach innen gemacht werden mussten. Für die Halbkugel wurde dieses mal allerdings nicht nur ein Bevel, sondern auch das Tool *Edit Mesh > Merge to Center* für die Vertices der unteren zylinderfläche genutzt, um diese an einem Punkt in der Mitte zusammenzufügen. Mit dem Multicuttool und Skalieren dieser Kanten wird der erschaffene Kegel nun in die Form einer Halbkugel geformt. Danach wird der Bevel verwendet, um die Halbkugel abzurunden.

#### **6.4.5.1 Texturierung**

##### **Druckkörper**

Die Textur des Druckkörpers besteht aus einer UV-Map. Diese wurde in Maya mit dem Generierungstool automatisch erstellt. Mit dem Paint Effects Tool wird diese Map in einem Purpurviolett bemalt. Diese Map wird dann mit dem Mesh nach Unity exportiert und dort als Material an den Druckkörper angebracht. (Abb. 6.23)

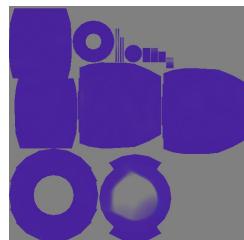


Abbildung 6.23: Map des Druckkörpers

##### **Mittel-Körper**

Für den Mittelkörper wurde eine UV-Map mittels des automatischen Generierungstools (*UV > Automatic*). Diese wird dann mit dem Paint Effects Tool bemalt. Dabei bekommt dieser einen dunklen Blauton, wobei die Nadelhalterung vergoldet wird und der Bereich des Ziffernblattes leer gelassen wird.

Wurde das Ziffernblatt erstellt, wird dieses in die Datei der Textur mit Photoshop eingefügt. Das Ziffernblatt wird dann im nächsten Schritt als PNG exportiert, um dieses dann in Unity als Textur hinzuzufügen. Die Textur wird dann dem importieren Mittelkörper als Material angefügt. (Abb. 6.24)

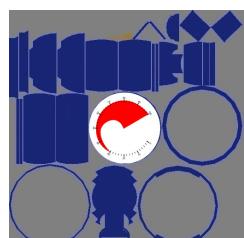


Abbildung 6.24: Map des Mittelkörpers mit Ziffernblatt

### Ziffernblatt

Das Ziffernblatt wurde in Adobe Illustrator erstellt. Dafür wurde zum Großteil das *Line-tool* zum Erstellen von 2 dimensionalen Linien verwendet. Mit dem Rotationspunktwerkzeug und dem Duplizieren dieser Linien, ist es möglich die Linien um einen gesetzten Punkt rotiert duplizieren zu lassen. Die Ziffern wurden dann mit dem *Text-tool* erstellt und passend rotiert. Die rote Form in der Mitte entstand aus der Vorlage und wurde mit dem *Kurventool* erstellt und mit roter Farbe gefüllt. Das Ziffernblatt wurde dann als PNG exportiert, da wir einen transparenten Hintergrund benötigen.



Abbildung 6.25: Map der Nadel

### Nadel

Die Nadel wird ebenfalls mit einer UV-Map texturiert.

Die UV-Map bekommt, wie nach der Vorlage definiert, eine monochrome schwarze Textur. Auch die Map wird mit dem Mesh nach Unity exportiert und dort an die Nadel angehängt. (Abb. 6.25)

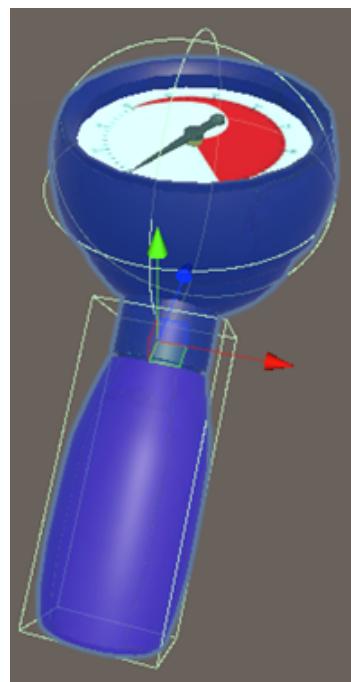


Abbildung 6.26: Fertiges Modell des Manometers

## 6.4.6 Trachealkanüle

### 6.4.6.1 Aufbau des Meshes

#### Kanülenrohr

Das Kanülenrohr wird mit einem Zylinder begonnen, der einen Durchgang in der Mitte bekommt. Davor werden für den Zylinder der Höhe entlang 20 Subdivisions gesetzt. Diese werden dann für den Bend-Deformer benötigt, der benutzt wird, um den Zylinder um 90° zu biegen. (Abb. 6.27)

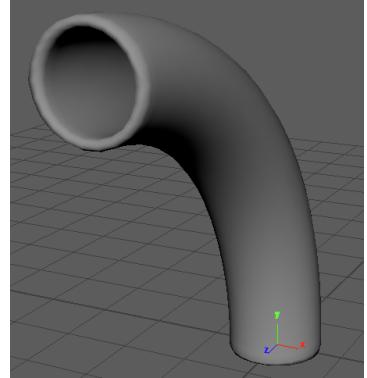


Abbildung 6.27: Mesh des Kanülenrohrs

#### Cuff

Den Cuff wird mit einem Zylinder begonnen, der ebenfalls einen Durchgang in der Mitte bekommt. Dieser Durchgang soll einen etwas größeren Durchmesser haben, als das Kanülenrohr. Durch das Verschieben der Subdivisions nach außen, wird der Cuff wie ein aufgeblasener Ballon dargestellt. Mit dem *Mesh > Retopologize* Tool, wird dem ganzen ein natürlicherer Look gegeben, da das Mesh runder wird. Dieses wird dann so positioniert, dass es an der unteren Seite um das Kanülenrohr angeklebt ist. (Abb. 6.28)

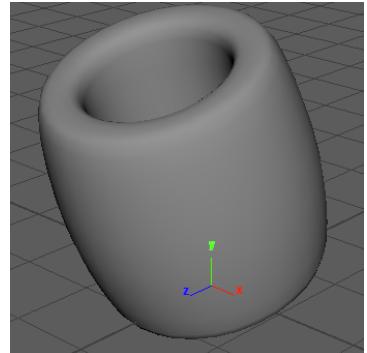


Abbildung 6.28: Mesh des Cuffs

### Konnektor

Der Konnektor wird mit einem Zylinder begonnen und wieder einen Durchgang gegeben. Auf der Außenseite wird dann mit dem *Multicuttool* eine weitere Subdivision in das Mesh eingeschnitten. Die Subdivision wird an den Rand einer Seite des Meshes gezogen und die entstandene Fläche mit dem *Extrude-Tool* herausgezogen. Von diesem neuen Meshstück, wird die am Rand des Meshes liegende Fläche genommen und wieder mit einem Extrude herausgezogen. Dadurch entsteht der Aufsatz, der dann sozusagen an das Kanülenrohr gesteckt wird. Um das Mesh ein realistischeres Aussehen zu geben, habe ich das *Retopologize-Tool* benutzt, wodurch die Kanten etwas weniger angewinkelt werden. (Abb. 6.29)

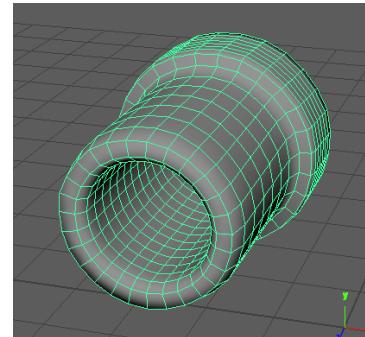


Abbildung 6.29: Mesh des Konnektors

### Schild

Das Schild wird anders als die anderen Meshes mit einem Würfel begonnen, aus dem im ersten Schritt ein langer Quader gebaut wird. Als Nächstes wird auf der langen Seite in der Mitte eine weitere Kante, mit dem *Multicuttool*, eingeschnitten. Diese wird dann aus dem Mesh herausgezogen. Mit dem *Bevel-tool* habe ich dann die Kanten des Meshes abgerundet.

Als Nächstes kommen die Löcher für die Kanülenbänder. Dafür wird einen Würfel genommen und dieser in einen Quader skaliert. Der Quader wird dann mit einem *Bevel* auf beiden Seiten abgerundet. Diese Form wird dann dupliziert und an die andere Seite des Meshes platziert. Mit dem *Difference-Tool* werden die beiden Formen genommen und aus dem Mesh des Schildes ausgeschnitten.

Als letzten Schritt wird auch das Schild einmal *Retopologized*, damit dieses glatter und abgerundeter wirkt. (Abb. 6.30)

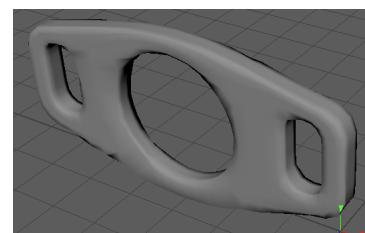


Abbildung 6.30: Mesh des Schild

### Ballon und Ballon-Konnektor

Für den Ballon wird wieder mit einem Würfel begonnen. Der Würfel wird zu einem langen Quader skaliert und dann mit dem *Cutting-tool* eine ungefähre Form des realen Ebenbilds gegeben. Mit einem *Bevel* werden die erstellten Kanten abgerundet. An der oberen und unteren Fläche habe ich mit dem *Cutting-Tool* die Kanten so gesetzt, dass dem ganzen eine Ballonform, durch Unebenheiten, gegeben wird

Um den Durchgang in der Mitte zu bilden, wird mit einem *Union* ein Zylinder dem Mesh angefügt und mittels der Bewegung der Kanten die gewünschte Form gebildet. Um nun den Durchgang zu kreieren, wurde der Zylinder vor dem Union dupliziert, in der Mitte des Originalmeshes herunterskaliert und mit einem *Difference* aus diesem herausgeschnitten.

Beim Ballon-Konnektor wird der bereits erstellte Konnektor der Kanüle kopiert und herunterskaliert, sodass der Konnektor an den Ballon-Durchgang passt. (Abb. 6.31)

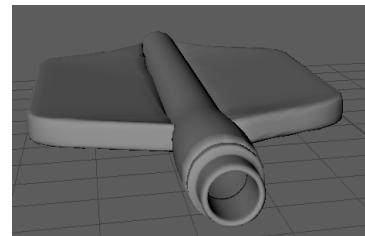


Abbildung 6.31: Mesh des Ballons

#### 6.4.6.2 Texturierung

##### Kanülenrohr, Cuff und Cuffröhre

Das Kanülenrohr, der Cuff und die Cuffröhre bekommen den transparenten Shader, der Spritze und des HMEs. Der Shader für den Cuff wird dupliziert und soll etwas mehr reflektieren und nicht ganz so transparent sein. Dafür wird der Blending Mode von Additive auf Alpha geändert und unter Surface Inputs die Smoothness auf 0.39 erhöht.

##### Konnektor und Schild

Der Konnektor und das Schild bekommen beide ein Unity-Material mit einer weißen *Base Map*. Sonst wird an dem Material nichts geändert.

**Ballon und Ballon-Konnektor** Der Ballon bekommt ebenfalls ein durchsichtiges Material. Dieses wird nun ebenfalls dupliziert und etwas abgeändert. Die Specular Map bekommt eine blaue Farbe und die Base Map wird gelöscht, wobei dafür der Blending Mode auf *Premultiply* geändert werden muss, damit die Farbe richtig dargestellt wird.

Der Konnektor des Ballons bekommt ein weißes Unity-Material.

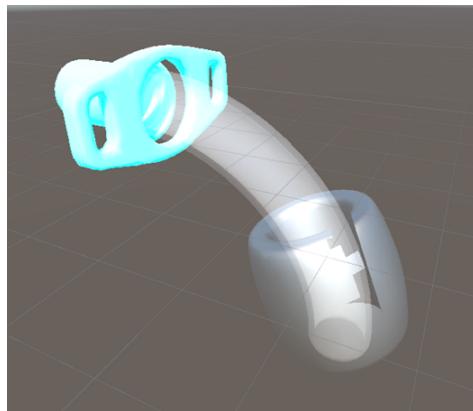


Abbildung 6.32: Fertiges Modell der Kanüle, nur ohne Cuffrohr

#### 6.4.6.3 Animation und Verhalten

##### Cuff

Der Cuff um die Röhre ist dafür da, dem Patienten Bequemlichkeit zu verschaffen. Da dieser in seinem aufgeblasenen Zustand nur schwer hinein- und herausgenommen werden kann, kann die Luft aus diesem ein- und ausgelassen werden. Dafür gibt es den Kontrollballon und das mit dem Cuff verbundene dünne Rohr.

##### Erster Versuch

Der erste Versuch war einfach nur diesen zu skalieren und etwas zu rotieren. Dieser hatte allerdings keine wirkliche Ähnlichkeit zur Realität.

##### Zweiter Versuch

Um die Animation um einiges realistischer zu machen, wird die Maya Cloth-Physik nCloth aus dem nDynamics-Packet verwendet. Dafür wird aus dem Cuff ein nCloth gemacht (*nCloth > Create nCloth*) und der dazu erstellte Nukleus so angepasst, dass der Cuff sich nach innen zusammenzieht. Damit der Cuff in Position bleibt, wird der Wert für die Schwerkraft auf 0 gesetzt und dessen *Direction* wird nicht verändert. Genau so wird Wind Noise und Wind Speed auf 0 gesetzt. Nur die Air Density wird auf 1 gesetzt, wodurch sich das Objekt dann zusammenzieht.

##### Export

Für die Animation wird eine Key-Frame-Animation von 30 Keyframes erstellt, da nach den 30 Frames das Programm das Mesh etwas unbrauchbar macht, da sich die Polygone ineinander erstrecken. Diese sollte erst als FBX exportiert werden, was nicht funktioniert hat. Nach weiterer Recherche kam heraus, dass man die Cloth-Animationen nicht als FBX, in einer für Unity verwendbaren Form exportieren kann. Die erste Möglichkeit, die angeboten wird, ist der Blend-Shape-Deformer. Für diesen musste das Mesh an den wichtigsten Frames der Animation dupliziert und die Cloth-Component von diesem gelöscht werden, sodass nur das Mesh übrig bleibt. Bei Maya muss man allerdings darauf achten, in welcher Reihenfolge man die Objekte auswählt, da das für Maya und das Tool relevant ist. So wird das letzte ausgewählte Objekt als Base-Objekt für die *Blend-Shapes* verwendet, welche dann diese Shapes als Attribute gespeichert hat. Durch

das Ändern der Attribute von 0 auf 1 können die Shapes gewechselt werden, wobei bei Aktivierung mehrerer Shapes, die Werte dieser in dem Base-Mesh zusammengesetzt werden. Mit einer Key-Frame Animation setzt man dann die Shapes des Objektes an den Frames, an denen man diese entnommen hat. Die Animation ist zwar nicht ganz so flüssig, aber als FBX exportierbar.

Allerdings kam es auch mit dieser Lösung zu einem Problem. Der genaue Grund wurde nie ganz herausgefunden, wobei einige Quellen davon ausgehen, dass die Shapes immer noch von den nCloth-Objekten gestört werden. Das Problem besteht, dass die Animation nicht als FBX exportiert wird.

Als nächsten Ansatz wurde dann der *Alembic Cache* gefunden, welcher genau für die nDynamics und Rigging-Animationen existiert. Die grundlegende Idee dieser Technologie ist, dass die Geometriedaten des Meshes für die Animation in einem Zwischenspeicher (Cache) gespeichert werden. In Fall des Alembic Caches ist es eine Datei, die diese Meshdaten an andere Programme, welche den Alembic Cache unterstützen, weitergeben kann.

Das Erstellen dieser Datei in Maya ist sehr einfach. Man geht in der Menüleiste zu *Cache > Alembic Cache > Export Selection (All) to Alembic*. Dort wird der erste Animationsclip von selbst gespeichert, es können aber auch mehrere Frames in mehreren Animationen gespeichert werden. Um in Unity nun diese Datei importieren zu müssen, muss man erst das Alembic Package installieren, falls man dies nicht bereits gemacht hat. Dann funktioniert die Datei ähnlich wie eine FBX-Datei. In dieser befinden sich bei einem Animationsclip zwei Unity-clips. Der mit dem Namen Time stellt dabei die Cloth-Animation dar, welche man dann auch in einem Animator verwenden kann.

Als letzten Schritt wird der animierte Cuff in die Kanüle eingefügt und der Kanüle einen Animator mit zwei States gegeben zum ein- und auslassen der Luft. Diese sind mit zwei Triggern verbunden, welche die Animation dann abspielen sollen. Für die Einlassen-Animation wird die Auslassen-Animation mit einer Geschwindigkeit von -1 - also rückwärts - abgespielt.

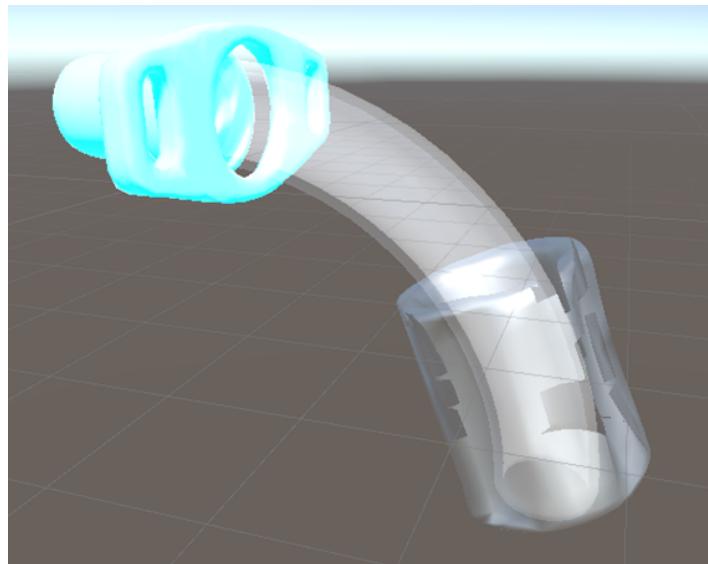


Abbildung 6.33: Animation des Cuffs

## Cuffrohr

Da wir ein dynamisches Rohr brauchen, das sich je nach Position der Kanüle und des Kontrollballons, anpassen soll, muss dieses in Unity gebaut werden. Wünschenswert wäre natürlich auch, dass sich dieses wie eine Art Seil, das gespannt wird, verhält und durchhängen kann.

Der **erste Ansatz** dafür waren die von Unity mitgelieferten Joints, welche für das Verbinden von Rigidbodies gedacht sind. Dabei gibt es einige, die genau für diesen Zweck angewendet werden können. Der erste Versuch der Joints war also mit den Hinge Joints, welche dem verbundenen Objekt, eine Rotation an dem anderen Objekt erlaubt, aber sonst zusammenhält. Die Hinge Joints haben im Test erst auch funktioniert, wobei hier 12 Zylinder und der Ballon an der Kanüle angehängt wurden.

Beim Einfügen in die Szene sind die einzelnen Teile, also die Rigidbodies, dann aufgrund der Joints so durcheinander geflogen, dass diese kein stabiles Seil mehr bilden konnten. Nach einiger Recherche zur Unity-Physik und Rigidbodies, hat sich dann ergeben, dass die Größe der Szene ein Problem war, da diese viel zu klein für die Physik-Engine ist und damit auch die verwendeten **Props**.

Um nun auszutesten, ob jede Joint-Art ein Problem mit der Größe hat, wurde die Kanüle in der Test-Szene herunterskaliert. Anstatt der Hinge Joints werden diese durch Spring Joints ausgetauscht, wodurch die einzelnen Rigidbodies sich verhalten als würden sie an einer Feder hängen. Diese Methodik hat jedoch nach vermehrten Austesten verschiedener Werte trotzdem Probleme gemacht, da entweder das Cuffrohr auseinandergebrochen ist oder sich viel zu sehr gestreckt hat.

Nachdem die Joints nicht funktioniert haben, wurde, bei weiterer Recherche, die **Verlet-Integration** gefunden. Diese basiert auf Grundlage von verbunden Punkten. Dabei nimmt man die Differenz der vorherigen Position und der momentanen Position, um die nächste Position zu berechnen. In diese Berechnung wird auch die Schwerkraft mit einberechnet.

Tatsächlich beruht die Unity-Physik eher auf den Prinzipien von Euler, die sich sehr gut für Festkörper anbietet. Die Verlet-Integration bietet sich hingegen besser bei den beweglichen Körpern an, wie Kleidung, Seile und weiche Objekte, wie ein Polster. Dazu wurde in den Foren sehr oft erwähnt, dass Joints viel mehr Probleme machen und oft auch die Performance unter diesen leidet.[\[58\]](#)

Verlet Integration bietet hingegen die Möglichkeit, diese bewegbare Röhre zu erstellen, ohne, dass die Performance darunter leidet. Ebenfalls sind Verlet-Seile um einiges stabiler. Im folgenden Code-Snippet, wird gezeigt, wie die Berechnung der einzelnen Punkte umgesetzt wurde. (Auflistung [6.32](#))

```

1 // Methode, welche die Punkte anhand der Position und der vorherigen Position
2   ↪ berechnet
3 private void UpdateRopeSimulation()
4 {
5     Vector3 gravity = new Vector3(0f, this.gravity, 0f); // Vektor für die
6       ↪ Schwerkraft definieren.
7     float t = Time.deltaTime;
8
9     for (int i = 1; i < nodes.Count; i++) // Iteration über die Nodes
10    {
11        RopeNode current = nodes[i]; // Zwischenspeicherung des momentanen Nodes
12        Vector3 vel = current.pos - current.oldPos; // Berechnung der Differenz der
13          ↪ momentanen und vorherigen Position
14        current.oldPos = current.pos; // momentane Position wird zur vorherigen
15        Vector3 newPos = current.pos + vel; // Differenz wird an die momentane
16          ↪ Position angerechnet.
17        newPos += gravity * t; // Schwerkraft wird mit Deltatime dazugerechnet.
18        current.pos = newPos; // Neue Position wird zur momentanen Position
19        nodes[i] = current; // Abspeichern des Nodes im array
20    }
21 }
22
23 // Stellt die Nodes des Ropes dar und speichert die momentane sowie die vorherige
24   ↪ Position
25 public struct RopeNode
26 {
27     public Vector3 pos;
28     public Vector3 oldPos;
29
30     public RopeNode(Vector3 pos)
31     {
32         this.pos = pos;
33         this.oldPos = pos;
34     }
35 }
```

Auflistung 6.32: Berechnung der Punkte

Momentan lässt sich das Rohr allerdings unendlich lang ziehen, ohne dass die Position, anhand der vorgegebenen Länge zwischen den Punkten, korrigiert wird. Darum kümmert sich das nächste Code-Snippet. (Auflistung 6.33)

```

1 // setzt die Länge der Positionen auf die angegebenen RopeLength
2 private void MaximumStretch()
3 {
4     for (int i = 0; i < nodes.Count - 1; i++) // Iteration über die Punkte
5     {
6         RopeNode top = nodes[i]; // Nimmt den momentanen Punkt
7         RopeNode bot = nodes[i + 1]; // Nimmt den nächsten Punkt
8         float dist = (top.pos - bot.pos).magnitude; // Entfernung beider Punkte
9         ↵ berechnen
10        float distError = Mathf.Abs(dist - ropeNodeLength); // Wie viel die
11         ↵ Entfernung von der vorgegebenen Länge abweicht.
12        Vector3 changeDir = Vector3.zero;
13
14        if (dist > ropeNodeLength)
15            // Ist die Distanz beider Punkte größer als die vorgegebene
16            {
17                changeDir = (top.pos - bot.pos).normalized; // wird der Richtungsvektor
18                ↵ von dem momentanen Punkt zum nächsten Punkt genommen
19            }
20        else if (dist < ropeNodeLength) // Ist die Distanz kleiner als die
21            ↵ vorgegebene Länge
22            {
23                changeDir = (bot.pos - top.pos).normalized; // wird der Richtungsvektor
24                ↵ von dem nächsten zum momentanen Punkt genommen
25            }
26        Vector3 change = changeDir * distError; // Änderung der Distanz zwischen
27        ↵ den Punkten
28        if (i != 0 && i + 1 != nodes.Count - 1)
29        {
30            bot.pos += change * ropeNodeLength;
31            nodes[i + 1] = bot;
32            top.pos -= change * ropeNodeLength;
33            nodes[i] = top;
34        }
35        // Beim ersten wird statt dem momentanen nur der nächste Punkt verändert,
36        ↵ da der erste ja vom Startpoint bearbeitet werden soll
37        else if (i == 0)
38        {
39            bot.pos += change;
40            nodes[i + 1] = bot;
41        }
42        // Umgekehrt auch beim letzten. Hier soll der vorherige Punkt nur richtig
43        ↵ gestellt werden.
44        else
45        {
46            top.pos -= change;
47            nodes[i] = top;
48        }
49    }
50}

```

Auflistung 6.33: Korrektur bei Überdehnung

Die Berechnung der Punkte ist vorerst abgeschlossen und überprüfen auch, dass diese die richtige Spannung haben. Um nun das Rohr zu zeichnen, nehmen wir eine *LineRenderer-Component* und setzen die Position der Punkte in diese. Der LineRenderer zeichnet zwischen den, in der Komponente gespeicherten, Punkten dann Linien, welche sich immer zur Kamera drehen, wodurch das Rohr gezeichnet wird. (Auflistung 6.34)

```

1 // Setzt die Punkte der LineRenderer Component und updatet die Positionen der Punkte
2 private void DisplayRope()
3 {
4     lineRenderer.startWidth = ropeWidth;
5     lineRenderer.endWidth = ropeWidth; // Setzen der Breite des LineRenderers
6
7     Vector3[] positions = new Vector3[nodes.Count];
8
9     for (int i = 0; i < nodes.Count; i++)
10    {
11        positions[i] = nodes[i].pos; // Positionen in einem Vector3 Array speichern
12    }
13
14    lineRenderer.positionCount = positions.Length; // Setzen der Anzahl an
15    ↪ Positionen
16    lineRenderer.SetPositions(positions); // Setzen der Positionen
}

```

Auflistung 6.34: Zeichnen der Quads zwischen den Punkten

Als letzten Schritt wird sich noch um das Finden und Behandeln von Kollisionen gekümmert. Dafür wurde an den Positionen der Punkte eine OverlapSphere errichtet, welche Unity in seiner Physik-Engine anbietet. Diese simuliert eine Kugel an der gegebenen Position mit gegebenem Radius und gibt alle Collider an, welche diese Sphere in dem Frame berührt. Um dann die Position der Punkte an den Rand des berührten Objektes zu legen, benutzt man von der Mitte des Objektes den Richtungsvektor. (Auflistung 6.35)

```

1 // Nimmt Kollisionen mit Collidern wahr und korrigiert die Punkt entsprechend
2 private void AdjustCollision()
3 {
4     for(int i = 2; i < nodes.Count; i++) // Iteration der Nodes ab Position 2,
5         → da der Startpunkt ignoriert wird.
6     {
7
8         if (i == nodes.Count - 2) // Die letzten 2 Punkte
9             break;
10
11
12         RopeNode current = nodes[i]; // Nimmt den momentanen Punkt
13
14         // Finden von Collidern mit OverlapSphere
15         Collider[] colliders = Physics.OverlapSphere(current.pos, ropeWidth /
16             → 2f);
17
18         // Iteration der Collider
19         foreach (Collider c in colliders)
20         {
21             Vector3 cCenter = c.transform.position; // Mitte des kollidierten
22             → Objekts
23             Vector3 cDirection = cCenter - current.pos; // Distanz von der
24             → Mitte zum momentanen Punkt
25             current.pos -= cDirection.normalized * Time.deltaTime; // → Berechnung der Korrektierung des Punktes mit dem
             → Richtungsvektor der Distanz
             nodes[i] = current; // Überschreibung des Punktes
             break; // Es wird nur der erst gefundene Collider an dem Punkt
             → berechnet
26         }
27     }
28 }

```

Auflistung 6.35: Kollisionserkennung und -behandlung

Diese Lösung ist zwar noch etwas unruhig, da gewisse stabilisierende Berechnungen fehlen. Allgemein werden die Punkte allerdings außerhalb des Colliders gehalten, was vorerst ausreicht.

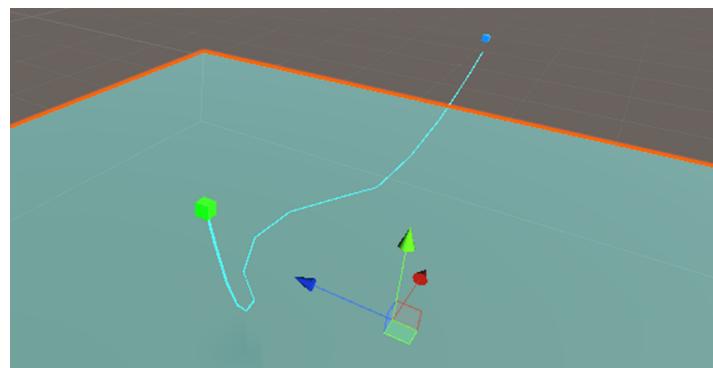


Abbildung 6.34: Demo der Physik des Cuffrohrs

## 6.4.7 Trachealkompresse

### 6.4.7.1 Aufbau des Meshes

Für den Aufbau der Trachealkompresse wird mit einem Würfel begonnen. Dieser wird in die Länge und Breite gezogen, aber flach gelassen. Dann setzt man die Subdivisions für die Länge und Breite auf fünf um, sich ein Bild machen zu können, wo das Loch gesetzt werden soll.

Dieses kann man wieder mit einem Zylinder und dem *Difference-Tool* angehen. Hat man nun das Loch, so schneidet man auf der Fläche eine Öfnung von dem Loch zum Rand des Meshes. Dies kann mit dem Multicuttool gelöst werden, wobei man darauf achten muss, dass diese Cuts auf der oberen und unteren Fläche identisch sind. Dann kann man die Flächen löschen, welche innerhalb der geschnittenen Grenze liegt.

Wählt man als nächsten Schritt die Kanten der freigelegten Flächen aus, kann man *Mesh > Fill Hole* benutzen, um an diesen Stellen, das Mesh zu schließen. Um die Kanten der Komresse abzurunden, benutzt man das *Bevel-Tool*.

Um die Komresse, noch etwas realistischer zu machen, wurde diese mit dem Bend-Deformer, welcher unter *Deform > Bend* zu finden ist, etwas manipuliert, um diese an die Form des Halses anzupassen. Dabei musste allerdings auf die richtige Rotation geachtet werden, damit auch die gewünschte Form erreicht werden kann.

### 6.4.7.2 Texturierung

Die Vorlage der Trachealkompresse besteht aus weißem Stoff. Somit wird dieses mit einem weißen Unity-Material eingefärbt.

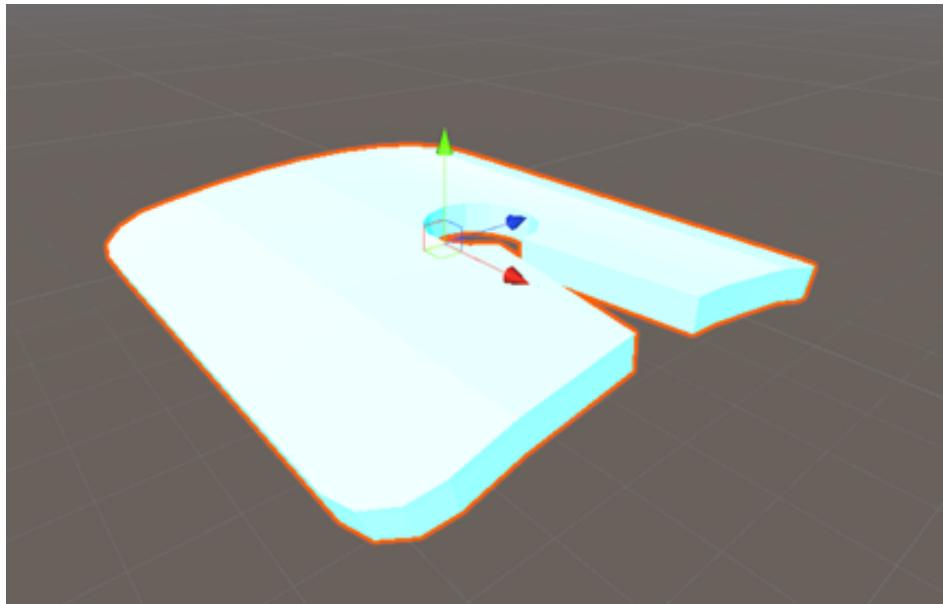


Abbildung 6.35: Fertiges Modell der Trachealkompresse

## 6.4.8 Kanülenbänder

### 6.4.8.1 Aufbau des Meshes

Für den Aufbau des Bandes wird mit einem Würfel angefangen und dieser zu einem langen flachen Quader skaliert. Als Nächstes werden dem Mesh dann einige Subdivisions entlang der Länge und Breite hinzugefügt, um später auch die Animation zu ermöglichen.

Mit dem *Bevel-Tool* werden dann die Kanten des Meshes abgerundet. Die Subdivisions der oberen und unteren Fläche des Meshes werden an die Seite gezogen, damit die Polster ähnliche Struktur der Bänder gebaut werden kann. Dafür wird dann nur noch das Verschieben dieser Subdivisions benötigt, um die Flächen abzurunden und als Polsterung darstellen zu können.

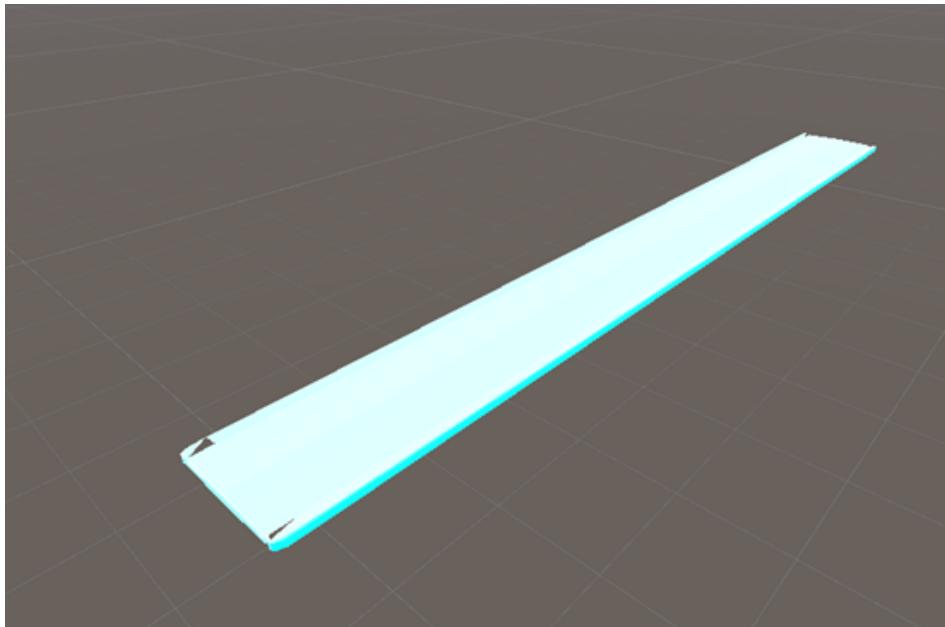


Abbildung 6.36: Fertiges Modell des Bandes

### 6.4.8.2 Animation

Da die Cloth-Component in Unity für diesen Anwendungszweck ungeeignet ist und sich das Verbinden durch den Spieler als äußerst schwierig erweist, wird die Animation in Maya vorgebaut. Dafür wird diese als Key-Frame Animation mit dem einem Bend-Deformer gebaut.

Um mit dem Deformer eine Animation machen zu können, muss dieser erst wieder richtig rotiert und dann so eingestellt werden, dass er sich in der Simulation von selbst um den Hals des Patienten legen kann. Dafür werden für beide Zustände die Keyframes gesetzt. Da wir allerdings diese Animation nicht mittels FBX exportieren können, müssen wir aus der Animation bei den wichtigsten Frames, das Mesh duplizieren.

Dann wählt man vom letzten zum ersten dieser Objekte das Mesh aus und setzt unter *Deform > Blend Shapes*, sodass die verschiedenen Meshes in einem Objekt gespeichert werden. Nun kann, durch das Setzen der richtigen Shapes in der richtigen Reihenfolge in den Keyframes, diese Blend-Shapes-Animation abgespielt werden. Diese Animation kann nun per FBX an Unity übergeben und dort als Clip abgespielt werden.

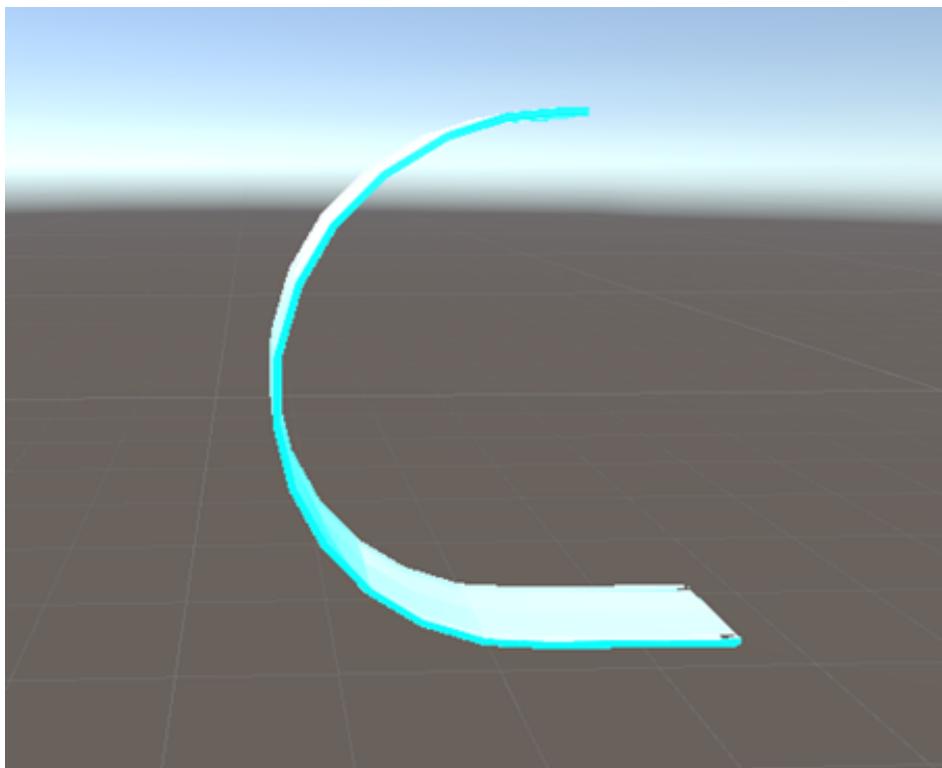


Abbildung 6.37: Bend-Animation des Bandes

## 6.4.9 Gleitgeltuch

### 6.4.9.1 Aufbau des Meshes

Das Mesh des Tuchs wird in Maya auf Grundlage einer 2-dimensionalen Plane erstellt. Diese Plane bekommt eine Cloth-animation mit dem *nDynamics-Tool nCloth*. Dafür wird die Plane als nCloth gesetzt. Um dies zu finden, müssen wir das Menü links auf *FX* stellen und dann wird oben in der Menüleiste die Option *nCloth* angezeigt. Da nur eine nCloth-Component benötigt wird, erstellt man diese mit *Create nCloth*.

Mit dem nun erstellten nCloth wird auch ein *Nucleus* erstellt, welcher dem Cloth-Objekt Physik gibt. Dieser gibt dem Objekt vor, wie sich die Schwerkraft, der Wind sowie weitere Einflüsse auf dieses auswirkt. Um nun ein Mesh mit Falten aus diesem herauszubekommen, benötigen wir Constraints. Mit diesen können wir die einzelnen Punkte des Meshes "befestigen", sodass beispielsweise der simulierte Wind das Objekt nicht davonträgt, sondern von dem Constraint, in Position gehalten wird.

Maya berechnet sich aus den vorgegebenen Werten eine Key-Frame Animation. Wie Maya mit Physik Animationen erstellt, wurde bereits in der Studie behandelt. Durch diese Key-Frame Animation kann dann an einer Stelle das Mesh genommen werden. Dafür muss man in der Hierarchie *Display > Shapes* aktivieren und das outputMesh duplizieren. Dadurch entfernt sich auch die Cloth-Komponent von der Kopie. Das Mesh wird dann ausgewählt, per Object-Datei exportiert und in Unity importiert.

#### 6.4.9.2 Texturierung

Für das Gleitgeltuch wurde eine weiße Papiertextur heruntergeladen, die dem Original sehr ähnlich wirkt, aus dem Internet heruntergeladen.[34] Eine Papiertextur aus dem Grund, da keine der gefundenen Stofftexturen, welche frei zur Nutzung sind, an das Original so gut herankommen. Diese ist genauso zu behandeln, wie die UV-Maps. Diese Textur soll etwas, durch etwaige Musterung oder Körnung, Realismus in das statische Objekt bringen.

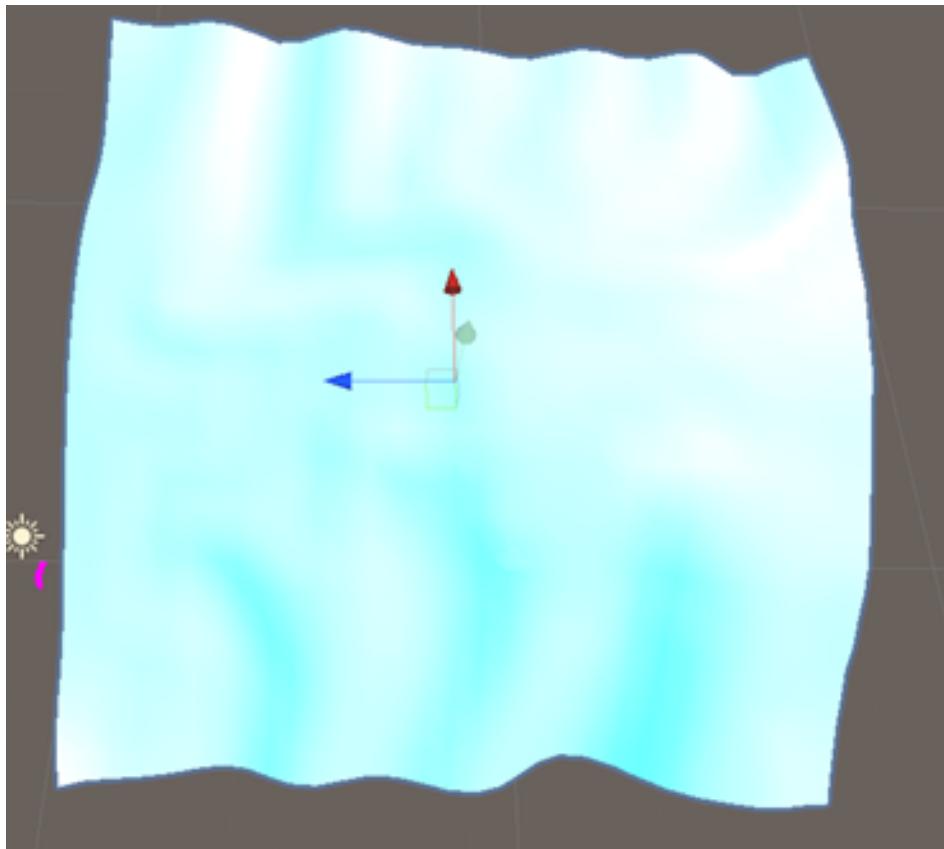


Abbildung 6.38: Fertiges Modell des Gleitgeltuches

#### 6.4.10 Hände

Bei den Händen wurde aus dem Internet zur Verfügung gestelltes Mesh verwendet. Dieses besteht allerdings nur aus einem statischen Mesh, somit fehlt jegliche Dynamik. Wir erstellen daher eine Animation der Hände erstellt und in Unity mit einem Animator zur Verfügung gestellt.

##### 6.4.10.1 Rigging

Um das Mesh der Hand zu animieren, wird ein Rig benötigt. Die Studie hat diese Methode bereits gut zusammengefasst. So stellt ein Rig eine Art Skelett des Mesches dar. Dieses Rig besitzt auch sogenannte *Bones*, welche eine Parent-Child Beziehung zueinander haben. Dabei hängt die Bewegung der Child-Bones von der Rotation der Parent-Bones ab.

Somit ist nach dem Importieren des Meshes der erste Schritt mittels des *Rigging-Tools* von Maya ein passendes Rig aufzubauen. Hierfür bietet es sich für die erste Positionierung an, die seitlichen Perspektiven zu nehmen, um die Rigs nicht zufällig in der Welt zu verteilen, da die freie Perspektive zum Setzen von Rigs einige Nachteile bringt.

In Maya kann man, zum Erstellen eines Rigs, mittels des *Create Joints-Tools* seine eigenen Bones setzen, um das Rig aufzubauen. Dabei muss man aber auch auf den vorgegebenen Radius der Bones achten, damit diese im Mesh bleiben, damit das Rig auch fehlerfrei funktionieren kann. Nachdem das Rig gesetzt wurde, muss die Position jedes Bones richtig in der Hand platziert werden. Hier muss man achten, dass man sich von oben nach unten arbeitet, da ja die Bewegung der Parent-Bones die Childs beeinflusst. Dabei hilft als Vorlage die eigene Hand, in dem man sich ansieht, an welcher Stelle sich die Gelenke wie bewegen.

Um nun das gesetzte Rig an das Mesh zu setzen, benutzt man das *Bind-Skin-Tool*, womit man dann das Mesh selber nicht mehr manipulieren kann, sondern dieses durch die Bones beeinflussen muss.

## Animation

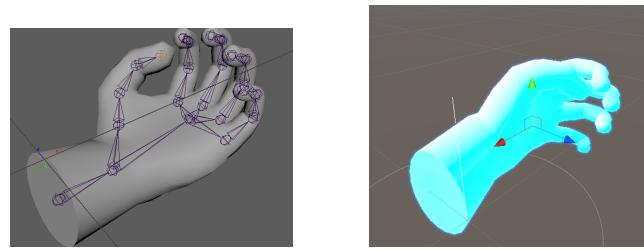
Die Animation wird mittels einer Keyframe-Animation realisiert. Dabei wird an Frame 0 die Standardrotation der Bones gesetzt und die Zeit-Leiste auf 30 Frames gesetzt, wodurch die Animation aus 30 Bildern besteht. An Frame 30 verändere ich nun an den wichtigsten Bones der Hand, also an den Fingern, die Rotation. Dabei muss man viel herumprobieren, um eine gute Einstellung der Hand zu finden (*Trial-and-Error*). (Abb. 6.39a)

Diese Animation wird mittels eines Alembic-Cache an Unity übergeben. Wie bereits erwähnt benötigt Unity dafür ein eigenes Package und muss vor der Benutzung installiert werden. Durch den Alembic-Cache ist es nun möglich, das Mesh sowie die Animation an Unity zu übergeben. Um nun die rechte Hand zu bekommen, kann man das Mesh einfach spiegeln.

In Unity kann man sich dann aus dem Alembic-Cache-File das Mesh der Hand holen und dieses in eine Szene hinzufügen. Allerdings kann man nicht das vor der Animation benutzte Prefab verwenden, da der Alembic-Cache auch nur sein eigenes Objekt manipulieren kann.

Durch die Animator-Component werden nun zwei States erstellt. Eine, welche die Hand zu macht und die andere, die sie aufmacht. Für die beiden States wird der TIME-Clip des Alembic-Cache verwendet, da dieser die Daten der Animation hat. Beim Schließen wird der Clip mit der Geschwindigkeit -1 abgespielt, damit diese Rückwärts läuft. Diese States werden aus per Trigger gestartet. Die Hand kann allerdings die Animation zum Öffnen nur abspielen, wenn diese vorher geschlossen wurde. (Abb. 6.39b)

Das ganze GameObject wird dann als Prefab zur weiteren Benutzung hinterlegt.



(a) Gerigge Hand in Maya (mit Rig)

(b) Gerigge Hand in Unity

Abbildung 6.39: Hand Animation

# Kapitel 7

## Retrospektive

### 7.1 Ziele

#### 7.1.1 Logik und Interaktivität einer Schulungs- und VR-App

##### 7.1.1.1 Programmsteuerung

Im Bereich der Programmsteuerung konnten alle Manager implementiert werden. Alle Must-Have Features im Bereich der Programmsteuerung wurden eingefügt. Auch das Anwenden des Singleton-Patterns, beim Erstellen des Processhandlers konnte sich als sinnvoll erweisen. Nice-To-Have Features wie die Erweiterbarkeit des Systems wurden im Projekt ebenfalls umgesetzt.

##### 7.1.1.2 Task-System

Alle definierten Tasks konnten durch das Anwenden der Task-Arten implementiert werden, so-mit sind alle Must-have-Anforderungen erfüllt. Manche Nice-to-haves wie die RotationsTask wurde aufgrund der Priorisierung anderer Tasks weggelassen und durch ConnectionTasks ersetzt.

#### 7.1.2 Bewegung und Interaktion mit der VR-Welt

##### 7.1.2.1 Entwicklungsumgebung und Workflow

Die Auswahl der Unity Engine hat sich als eine gute Entscheidung erwiesen. Das Vorwissen der Teammitglieder hat bei der Umsetzung viel Zeit gespart, die sonst mit Lernen und Recherchieren verbracht hätte sein können. Die Unterstützung für VR-Apps in der Unity Engine hat sich ebenfalls als ausreichend erwiesen. Das Testen während der Entwicklung hat den Workflow verbessert.

##### 7.1.2.2 Interaktion mit der virtuellen Welt

Alle Must-Have-Anforderungen im Bereich der Interaktion konnten erfüllt werden. Das Heben, Loslassen, Werfen und Verbinden von Objekten konnte erfolgreich umgesetzt werden. Das XR Interaction Toolkit war hier eine große Hilfe. Das Drehen von Objekten wie bei der Rotations-task wurde aufgrund der Zeitbegrenzungen weggelassen.

### 7.1.3 Graphical User Interface

#### 7.1.3.1 Darstellung von User Feedback und Aufgabenablauf

Da unser Auftraggeber KWP keine genauen Angaben bezüglich der Anzeige von Aufgaben oder dem User Feedback machte, kann nicht genau gesagt werden, ob alle notwendigen Features implementiert wurden. Manche Ideen, die wir uns im Konzept überlegt hatten, wurden entweder komplett verworfen oder stark geändert. Trotz des starken Kontrastes zwischen Geplanten und Umgesetzten scheint die KWP mit der Anzeige und dem User Feedback zufrieden zu sein. Alle Aufgaben werden dynamisch und intuitiv dargestellt und diverse Benachrichtigungen, wie Hinweise scheinen hilfreich zu sein.

#### 7.1.3.2 Hauptmenü, Loading Screen und Sonstiges

Im Bereich des Hauptmenüs wurde nur das Mindeste implementiert, da es aus zeitlichen Gründen nicht mehr möglich war ein komplexeres Menü zu erstellen und für uns dies auch nicht als sinnvoll erschien. Das Konzept des Loading Screens wurde komplett verworfen, da es nicht benötigt wurde. Stattdessen wurde eine Controller-Belegung implementiert und weitere Features, die unabhängig von der GUI sind, wie das Input System oder einzelne Animationen von Werkzeugen.

### 7.1.4 3D-Modellierung

#### 7.1.4.1 Modellierungsmethode und Modellierungssoftware

Die Auswahl des Box-Modelings hat sich für die [Props](#) als sehr effektiv herausgestellt. Nicht nur bot diese einen sehr einfachen Einstieg, sondern bewährte sich auch beim Einfügen von Details, als sehr nützlich. Im Nachhinein hätte an einigen Stellen mehr auf die Topologie geachtet werden sollen beziehungsweise auch das von Maya integrierte Retopologize-Tool genutzt werden sollen. Immerhin stellen Topologien eine wichtige Grundlage der 3D-Modellierung dar und wird besonders bei komplexeren Animationen wichtig zu meistern.

Maya als Modellierungssoftware bietet nicht nur die Tools für einen sehr angenehmen [Workflow](#) an, sondern auch ein gutes User-Interface, um komplexeste Technologien, so leicht wie möglich bedienen zu können. Auch wenn die grafische Benutzeroberfläche bei erstem Benutzen sehr überfüllt aussieht, ist es trotzdem nach erstem Einarbeiten einfach, die unterschiedlichen Tools und Plugins kennenzulernen und erste Workflows zu bilden. Ebenfalls ist es eine extrem gute Erfahrung, dieses Tool zu lernen, da es sich in vielen Bereichen der Entertainment-Industrie als Standard etabliert hat.

#### 7.1.4.2 Texturierung und Animation

Die Texturierung hat leider unter der Qualität der Meshes, sowie der Programmierung der Cuffröhre gelitten. Diese bestehen meist nur aus einfachen und monoton gefärbten Texturen oder einfach nur durchsichtigen Materials. Während die Objekte trotzdem sehr realistisch aussehen, könnten komplexere Texturen und Shader die Darstellung sogar noch besser gestalten.

Für die Objekte wurden einige sehr komplexe, aber realistische Animationen gebaut. Dabei wurden einfache Key-Frame Animation, die nur die Position, Rotation und Skalierung in Unity gemacht. Animationen, wie das Aufblasen des Ballons oder die Biegung des Kanülenbandes,

sind so durch die komplexeren Tools entstanden, die Maya zur Verfügung stellt, wobei diese den Objekten eine Dynamik geben. Auch das Bauen eines komplexeren Rigs, für die Hand, stellt eine gute Grundlage für Animationen der VR-App dar.

## 7.2 Verbesserung

### 7.2.1 Logik und Interaktivität einer Schulungs- und VR-App

#### 7.2.1.1 Programmsteuerung

Die Interaktion zwischen den einzelnen Managern konnte im Konzept gut definiert und implementiert werden. Das Verwenden eines Simpleton-Patterns für den Processhandler war für das Projekt aufgrund der Zeitbeschränkung sinnvoll, jedoch gibt es hier bessere Alternativen. Die Verantwortungen der Manager waren auch nicht perfekt definiert, weshalb manche Manager über ihren Bereich hinausgegangen sind.

#### 7.2.1.2 Task-System

Die Tasks sind grundsätzlich alle implementiert worden. Hier besteht jedoch einiges an Verbesserungspotential, da viele Tasks momentan nur annähernd die Realität darstellen können. Beispielsweise kann beim Hineingeben der Kanüle wie in der Realität die Rotation mitberücksichtigt werden. Momentan sind Task-Prefabs nur für die Vorbereitung und den Start der Task zuständig. Besser wäre hier, dass diese auch für das Beenden der Task verantwortlich sind.

### 7.2.2 Bewegung und Interaktion mit der VR-Welt

#### 7.2.2.1 Fortbewegung

Wir hätten uns mehr Zeit zum Durchtesten und Durchplanen des Fortbewegungssystems lassen können. Möglicherweise könnte man da eine bessere Lösung als Room-scale based finden. Ein vollständiges Fortbewegungssystem würde mehr Freiheit bei der Implementierung und auch mehr Möglichkeiten für sämtliche Erweiterungen geben. Auch in Sachen Room-scale based fehlt eine entsprechende Behandlung von Head Collisions.

#### 7.2.2.2 Zusammenfüge von Objekten

Bei dem Konzept von Compound Objects sollte die Technologie in beide Seiten gehen können. Im Projekt wird das CompoundObject nur für das Entfernen aus einem Zusammenfüge eingesetzt und ein komplett anderes System wird für Verbindungen verwendet. Das Verbinden und Entfernen hätte besser geplant und gestaltet werden können, damit nur ein einheitliches System für Zusammenfüge von Objekten entwickelt werden kann. Das derzeitige System ist funktionsfähig, jedoch unklar und ineffizient.

### 7.2.3 Graphical User Interface

#### 7.2.3.1 Darstellung von User Feedback und Aufgabenablauf

Alle wichtigen Implementierungen wurden zwar umgesetzt, doch sind nicht alle komplett fehlerfrei bzw. optimiert. Benachrichtigungen vom HUD "Aufgabe abgeschlossen" oder Hinweise, wie "Beide Hände verwenden" können von einem Objekt, wie z. b. der Wand teilweise oder komplett verdeckt werden, wenn sich der Benutzer zu nah an diesem aufhält. Für diesen Fehler wurde während der Implementierungsphase keine Lösung gefunden und somit wurde das Problem nicht weiter bearbeitet. Ebenso wurde ein Raycast für das Auswählen der Buttons verwendet, da ein Pointer wesentlich zeitintensiver wäre.

#### 7.2.3.2 Hauptmenü, Loading Screen und Sonstiges

Hier wurden wie beim User Feedback und Aufgabenablauf alle benötigten Features implementiert, doch hier wurde nur das mindeste implementiert. Es wäre noch möglich das Hauptmenü weiter auszubauen und mehr visuell ansprechender zu gestalten. Ebenso könnte man weitere Features, wie Optionen, hinzufügen. Ein Loading Screen würde nur dann Sinn ergeben, wenn die Wartezeit zwischen dem Hauptmenü und des Trainings lang genug ist. Ebenso existiert noch ein Fehler beim Input System, der manchmal Unity zum Abstürzen bringt. Dies passiert zum Glück nicht oft, doch eine Behebung dieses Problems wäre auf jeden Fall sinnvoll.

### 7.2.4 3D-Modellierung

#### 7.2.4.1 Texturierung

Wie bereits erwähnt, hat die Texturierung, sehr an der Modellierung komplexerer Meshes und Animationen gelitten. So wäre das Kennenlernen von Techniken, diese Modelle zu texturieren, in einigen Fällen zwar nicht unbedingt notwendig gewesen, hätte aber trotzdem zu sehr tollen "Nice-to-haves" für den Auftraggeber werden können.

#### 7.2.4.2 Darstellung der Szene

Durch die höhere Priorität der [Props](#) ist die Darstellung des Raumes sehr in den Hintergrund gerückt. Dabei wurde weniger darauf geschaut, diese für die Engine richtig einzustellen, sondern mehr darauf, das XR Rig so schnell wie möglich einsatzbereit zu machen. Besonders als Projektleiter wäre es wichtig gewesen, dem Team etwas über die Schulter zu schauen. Leider ist erst im Verlauf der Arbeit wirklich klar geworden, wie die Physik-Engine wirklich funktioniert und welche Probleme die zu kleinen Szene mit sich bringt.

## 7.3 Zukunft

### 7.3.1 Logik und Interaktivität einer Schulungs- und VR-App

#### 7.3.1.1 Planung

Durch dieses Projekt konnten wir viel Erfahrung im Planen von Systemen sammeln und diese Pläne auch angewendet sehen. Im Projekt haben wir oft Diagramme verwendet, um benötigte

Klassen zu definieren und die Interaktion zwischen dieser darzustellen. In zukünftigen Projekten soll bereits sollen solche Diagramme so früh wie möglich erstellt werden, damit man eine klare Idee von seinen Ideen hat.

### 7.3.1.2 Programmieren

Durch das Erstellen der Systeme konnten wir allgemeine Kenntnisse im Bereich Programmieren, besonders in C# erwerben. Da wir uns für die Implementierung der Funktionen neue Konzepte wie VR aneignen mussten, konnten wir uns auch in diesen Bereichen weiterbilden. Somit haben wir für zukünftige Projekte gelernt, wie man sich schnell neue Konzepte aneignen und anwenden kann.

## 7.3.2 Bewegung und Interaktion mit der VR-Welt

### 7.3.2.1 Planung

Das Projekt hat uns viele Probleme mit ungenauer Planung gezeigt. Wir haben gelernt, dass eine vorzeitige ausführliche Planung aller potenzieller Systeme und Technologien, viele Probleme löst, noch bevor sie auftreten. Zukünftige Projekte werden mit Klassendiagrammen, Sequenzdiagrammen und so weiter angefangen. Eine längere Planung spart Zeit bei der Umsetzung.

### 7.3.2.2 Entwickeln für VR

Aus diesem Projekt haben wir vieles über die Entwicklung von VR-Applikationen gelernt. Alles von Planung, Design, Interaktion bis hin zu der Bewegung in der virtuellen Welt. Potenzielle zukünftige VR Projekte werden von diesem erlernten Wissen profitieren. Ebenfalls war die Verwendung von neuen Unity Frameworks nützlich. Wir haben uns viele neue Konzepte aus Unity und C# angeeignet.

## 7.3.3 Graphical User Interface

### 7.3.3.1 Planung und Konzeptideen

Viele Konzepte wurden für die Funktionen der GUI erstellt, doch eine genauere Überlegung der Sinnhaftigkeit und dessen Nutzen wurde leider zu wenig in Betracht gezogen. In den nächsten Projekten sollte mehr Zeit für solche Implementierungskonzepte aufgewendet werden und genauer zwischen Must-have Features und Nice-to-have Features unterschieden werden.

### 7.3.3.2 Implementierung

Im Laufe der Umsetzung haben wir viele wichtige Methoden und Funktionen im Bereich GUI entdeckt und erlernt, die wir vorher noch nicht kannten, wie Vertical Layout Group, Content Size Fitter und viele mehr. Dadurch wurden viele Kenntnisse im Bereich Programmierung, aber auch im Unity Inspektor erlangt. Diesen Lernprozess sollte auch in die Zeitplanung für zukünftige Projekte in Betracht gezogen werden, besonders wenn es um ein unbekanntes Thema geht.

### 7.3.4 3D-Modellierung der Props

#### 7.3.4.1 Planung und Konzept

Das Projekt hat wieder einmal gezeigt, wie wichtig Planung in diesen Projekten ist. So hat beispielsweise die Änderung der Prioritäten auf die **Props** sehr wohl einen Einfluss auf, den bestehende Workflow gehabt und hätte mit der richtigen Planung besser umgesetzt werden können. Aber auch die Konzepte der einzelnen Props waren anfänglich etwas holprig, da Details nicht gleich klar waren oder auch darauf vergessen wurde. Dies hatte dann auch zur Folge, dass Props nicht alle Features haben konnten bzw. die Zeit für mehr "Nice-to-haves" nicht gereicht hat.

#### 7.3.4.2 Topologien

Wie bereits erwähnt, sind Topologien sehr wichtig in der 3D-Modellierung. So sollte auf dieses Detail besonders bei sehr komplexen Meshes und Animationen geachtet werden, selbst wenn dies in dem Projekt keine Probleme gemacht hat. In ähnlichen Projekten oder überhaupt für die 3D-Modellierung ist es allerdings wichtig, diese Technik zu verstehen und anwenden zu können.

# **Kapitel 8**

## **Conclusio**

Das Diplomprojekt “Medizinisches VR Training” hat dem gesamten Team einen Einblick in einen vorher teilweise unbekannten Bereich der VR-App Entwicklung gegeben. Somit sind wir nicht nur auf technische, aber auch auf Kommunikations-Probleme zwischen den einzelnen Teammitgliedern gestoßen. Trotz allen Stolpersteinen wurde das Projekt erfolgreich abgeschlossen. Es konnten nicht nur alle notwendigen Elemente (Tasks, usw.) rechtzeitig implementiert werden, sondern es gab auch genügend Zeit um Nice-to-Have Features wie z.B. die Cuffline zu integrieren. Dies wurde vor allem erreicht, da laufend eine große Menge an Feedback von sowohl unserem Auftraggeber, als auch unserer Testpersonen zugekommen ist. Zudem waren unsere Konzeptideen detailliert und gut genug durchgeplant, damit große Hindernisse im Projektfortschritt vermieden werden konnten.



# Glossar

**Hover state** Ein Button ist in einem Hover state, wenn z.B: der Mauszeiger oder andere pointer über z. b. einen Button “schweben”, aber nicht gedrückt wird.. [37](#)

**Loading Screen** Ein Loading Screen ist ein Bild, das gezeigt wird, während das Programm Inhalte ladet und initialisiert[\[94\]](#).. [38](#)

**Prefab** Ein Prefab ist im generellen eine Vorlage eines Objekts. Dies ist dann praktisch, wenn man mehrere Objekte von der selben Art erstellt werden müssen.[\[76\]](#). [55](#)

**Props** Ein Prop stellt in Spielen einen Gegenstand dar. Meist wird mit Props ein Tool verbunden, das benutzt werden muss, um ein bestimmtes Ziel zu erreichen.. [10](#), [29](#), [71](#), [72](#), [98](#), [122](#), [124](#), [136](#), [148](#), [150](#), [152](#)

**Workflow** Bezeichnung für die Ordnung oder Reihenfolge, in der die Arbeit absolviert wird. Dazu zählen in der 3D-Modellierung die benutzten Tools und Programme, welche zum Endergebnis führen sollen.. [45](#), [148](#)



# Literaturverzeichnis

- [1] Autoren von 3D-Ace. *Polygon and Spline Modeling: Know The Difference*. 04.04.2022. URL: <https://3d-ace.com/blog/polygon-and-spline-modeling-know-the-difference/>.
- [2] Autor des Amazon-Eintrags. *VBM Cuff Pressure Gauge Pocket*. 04.04.2022. URL: <https://www.amazon.co.uk/VBM-Cuff-Pressure-Gauge-Pocket/dp/B073Q1VNYP>.
- [3] Jahirul Amin. *ntroduction to rigging in Maya - Part 5 - Rigging the hands*. 04.04.2022. URL: <https://3dtotall.com/tutorials/t/introduction-to-rigging-in-maya-the-hands-jahirul-amin-animation-arms-shoulder-character>.
- [4] Autoren von Autodesk. *3D-MODELLIERUNGSSOFTWARE*. 04.04.2022. URL: <https://www.autodesk.de/solutions/3d-modeling-software>.
- [5] Autodesk Autoren. *Bullet Rigid and Soft Body Dynamics*. 04.04.2022. URL: <https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2019/ENU/Maya-SimulationEffects/files/GUID-CC0BDE04-468D-4383-9553-14157E23D171-.htm.html>.
- [6] Autodesk Autoren. *nDynamics Overview and Concepts*. 04.04.2022. URL: <https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-SimulationEffects/files/GUID-E1498F66-BD9D-4DB9-9BB7-EA123ABEB9E7-.htm.html>.
- [7] MetaQuest Autoren. *Oculus Rift S Anforderungen*. 10.03.2022. URL: <https://support.oculus.com/articles/getting-started/getting-started-with-rift/rift-s-minimum-requirements/>.
- [8] Wikipedia Autoren. *CryEngine*. 01.04.2022. URL: <https://de.wikipedia.org/wiki/CryEngine>.
- [9] Wikipedia Autoren. *Oculus Rift*. 10.03.2022. URL: [https://en.wikipedia.org/wiki/Oculus\\_Rift](https://en.wikipedia.org/wiki/Oculus_Rift).
- [10] Wikipedia Autoren. *Unreal Engine Wikipedia*. 01.04.2022. URL: [https://de.wikipedia.org/wiki/Unreal\\_Engine](https://de.wikipedia.org/wiki/Unreal_Engine).
- [11] Wikipedia Autoren. *Virtuelle Realität*. 10.03.2022. URL: [https://en.wikipedia.org/wiki/Virtual\\_reality](https://en.wikipedia.org/wiki/Virtual_reality).
- [12] Wikipedia Autoren. *XR*. 10.03.2022. URL: [https://en.wikipedia.org/wiki/Extended\\_reality](https://en.wikipedia.org/wiki/Extended_reality).
- [13] Martin John Baker. *3D Theory - Keyframe animation*. 04.04.2022. URL: <https://www.euclideanspace.com/threed/animation/keyframing/index.htm>.
- [14] Micah Bowers. *Level Up: A Guide to Game UI (with Infographic)*. 27.02.2022. URL: <https://www.toptal.com/designers/gui/game-ui>.

- [15] Conor Burke. *User interface*. 27.02.2022. URL: <https://ecampusontario.pressbooks.pub/gamedesigndevelopmenttextbook/chapter/user-interface/>.
- [16] Farah Lalani Cathaly Li. *How Covid has changed Education*. 26.03.2022. URL: <https://www.weforum.org/agenda/2020/04/coronavirus-education-global-covid19-online-digital-learning/>.
- [17] Autor von classVR. *VR im Bildungsbereich*. 21.03.2022. URL: <https://www.classvr.com/virtual-reality-in-education/>.
- [18] Autor von Clevis Consult. *TMS vs LMS, was passt am Besten zu ihren Bedürfnissen*. 20.03.2022. URL: <https://www.clevis.de/ratgeber/training-management-system-vs-learning-management-system/>.
- [19] Autoren von Concept Art Worlds. *Konstantin Maystrenko (Artist)*. 04.04.2022. URL: <https://conceptartworld.com/artists/konstantin-maystrenko/>.
- [20] Autoren von Consu-Med. *TRACOE care Kanülenhalteband max. Länge 43cm, 2-teilig mit Klettverschluss, REF 903-F*. 04.04.2022. URL: [https://www.consu-med.de/tracoe-care-kanuelenhalteband-ref-903-f?gclid=CjwKCAjw0a-SBhBkEiwApljUOsIna0g5sx5yaqbkr3F8qErasDbIRFQiXcmD4g-NnmYVApjA47WBHoC95AQAvD\\_BwE](https://www.consu-med.de/tracoe-care-kanuelenhalteband-ref-903-f?gclid=CjwKCAjw0a-SBhBkEiwApljUOsIna0g5sx5yaqbkr3F8qErasDbIRFQiXcmD4g-NnmYVApjA47WBHoC95AQAvD_BwE).
- [21] Autoren von Crytek. *CryEngine Homepage*. 01.04.2022. URL: <https://www.cryengine.com/>.
- [22] Autoren von Crytek. *CryEngine VR*. 01.04.2022. URL: <https://docs.cryengine.com/pages/viewpage.action?pageId=25536773>.
- [23] Autoren von Crytek. *Robinson: The Journey*. 01.04.2022. URL: <http://www.robinsonthegame.com/>.
- [24] Autoren von Crytek. *The Climb*. 01.04.2022. URL: <https://www.theclimbgame.com/>.
- [25] Thomas Denham. *What is UV Mapping and Unwrapping?* 04.04.2022. URL: <https://conceptartempire.com/uv-mapping-unwrapping>.
- [26] Bruno Deschatelets. *Understanding UV Mapping and Textures*. 04.04.2022. URL: [https://www.spiria.com/en/blog/desktop-software/understanding-uv-mapping-and-textures/](https://www.spiria.com/en/blog/desktop-software/understanding-uv-mapping-and-textures).
- [27] Autoren von Oculus for Developers. *VR Locomotion Design Guide*. 01.04.2022. URL: <https://developer.oculus.com/resources/bp-locomotion>.
- [28] Autoren von Educba. *What is GUI?* 25.02.2022. URL: <https://www.educba.com/what-is-gui>.
- [29] Autoren von Epic Games. *Advanced Framework - VR, Mobile & Desktop aus dem Unreal Engine Marketplace*. 02.04.2022. URL: <https://www.unrealengine.com/marketplace/en-US/product/advanced-vr-framework>.
- [30] Autoren von Epic Games. *Unreal Engine Homepage*. 01.04.2022. URL: <https://www.unrealengine.com/en-US/>.
- [31] Autoren von Epic Games. *Unreal Engine Simulations*. 01.04.2022. URL: <https://www.unrealengine.com/en-US/solutions/simulation>.
- [32] Autoren von Epic Games. *Unreal Engine XR*. 01.04.2022. URL: <https://www.unrealengine.com/en-US/xr>.
- [33] Josef Erl. *Virtual Reality: Alles, was ihr über VR wissen müsst*. 01.04.2022. URL: <https://mixed.de/virtual-reality-starter-guide>.

- [34] Autoren von Everytexture. *Muted High Resolution Paper Texture*. 24.04.2022. URL: <https://everytexture.com/everytexture-com-stock-paper-texture-00014/>.
- [35] Gunther Eysenbach. *Design Strategies for Virtual Reality Interventions for Managing Pain and Anxiety in Children and Adolescents*. 20.03.2022. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7055787/>.
- [36] Autoren von Fahl-Medizintechnik. *TRACHEOTEC® VARIO CUFF*. 04.04.2022. URL: <https://www.fahl-medizintechnik.de/produkte/trachealkanuelen/produktdetails/tracheotecr-vario-cuff/>.
- [37] Autoren von Fiverr Team. *A Guide to Game UI Design*. 05.03.2022. URL: <https://blog.fiverr.com/post/a-guide-to-game-ui-design>.
- [38] Autoren von freepbr. *Cork Wood PBR Material*. 29.03.2022. URL: <https://freepbr.com/materials/cork-wood-pbr-material/>.
- [39] Autoren von Fresenius Kabi. *Produktbilder Trachealkompressen*. 04.04.2022. URL: <https://www.fresenius-kabi.com/de/produktbilder-trachealkompressen>.
- [40] Bearded Ninja Games. *VR Interaction Framework vom Asset Store*. 02.04.2022. URL: <https://assetstore.unity.com/packages/templates/systems/vr-interaction-framework-161066>.
- [41] Salman Khan Ghauri. *What is Keyframe Interpolation in After Effects?* 04.04.2022. URL: <https://thesbit.com/keyframe-interpolation-in-after-effects/>.
- [42] Autoren von Globalhealthcare. *HME Tracheostomy Humidifying Filter*. 04.04.2022. URL: <https://globalhealthcare.net/usa/product/hme-tracheostomy-humidifying-filter/>.
- [43] Svetlana Golubeva. *Wie man 3D-Modelle erstellt*. 04.04.2022. URL: <https://www.artec3d.com/de/learning-center/how-make-3d-models>.
- [44] Autoren von Growland. *Disposable Syringe 2ml - 20ml*. 04.04.2022. URL: <https://www.growland.biz/Disposable-Syringe-2ml-20ml>.
- [45] Autoren der HTC Corporation. *HTC Vive Anforderungen*. 10.03.2022. URL: [https://www.vive.com/eu/support/vive/category\\_howto/what-are-the-system-requirements.html](https://www.vive.com/eu/support/vive/category_howto/what-are-the-system-requirements.html).
- [46] Author von Ionos. *Factory-Pattern: the key information*. 21.03.2022. URL: <https://www.ionos.com/digitalguide/websites/web-development/what-is-a-factory-method-pattern/>.
- [47] Autoren von IrisVR. *The Importance of Frame Rates*. 01.04.2022. URL: <https://help.irisvr.com/hc/en-us/articles/215884547-The-Importance-of-Frame-Rates/>.
- [48] Autoren von IronEqual. *THE ONLY ADVICE YOU WILL NEED TO MAKE A GREAT GAME UI/UX*. 05.03.2022. URL: <https://medium.com/ironequal/the-only-advice-you-will-need-to-make-a-great-game-ui-ux-74a0db8de642>.
- [49] Author von javatpoint. *Object Pool Pattern*. 22.03.2022. URL: <https://www.javatpoint.com/object-pool-pattern>.
- [50] Autoren von KeepTalking. *Keep Talking and Nobody Explodes*. 26.03.2022. URL: <https://keertalkinggame.com/>.
- [51] Monic Kraft. *Difference between TMS vs LMS*. 20.03.2022. URL: <https://trainingorchestra.com/training-management-system-and-learning-management-system-differences/>.

- [52] Hugh Langley. *Inside-out v Outside-in: How VR tracking works, and how it's going to change*. 01.04.2022. URL: <https://www.wearable.com/vr/inside-out-vs-outside-in-vr-tracking-343>.
- [53] Anna Lundberg. *Die Bedeutung der Farben und die Kunst, Farbsymbolik anzuwenden*. 05.03.2022. URL: <https://99designs.de/blog/design-tipps/bedeutung-der-farben/>.
- [54] Autoren von McNeel-wiki. *What does NURBS mean and why should I care?* 04.04.2022. URL: <https://wiki.mcneel.com/rhino/nurbs>.
- [55] Dr. med. Mikolaj Walensi. *Tracheostoma*. 24.04.2022. URL: <https://flexikon.doccheck.com/de/Tracheostoma>.
- [56] Anastasia Morozova. *Virtual Reality for the Entertainment Market*. 25.03.2022. URL: <https://jasoren.com/virtual-reality-for-the-entertainment/>.
- [57] Autoren von Oculus. *User Interface Components*. 12.03.2022. URL: <https://developer.oculus.com/resources/hands-design-ui/>.
- [58] Michael Palmos. *Verlet Rope in Games*. 04.04.2022. URL: <https://toqoz.fyi/game-rope.html>.
- [59] User von Reddit: Arc8ngel. *Oculus Touch controller diagrams*. 29.03.2022. URL: [https://www.reddit.com/r/oculus/comments/8ycgov/oculus\\_touch\\_controller\\_diagrams/](https://www.reddit.com/r/oculus/comments/8ycgov/oculus_touch_controller_diagrams/).
- [60] Author von Refactoring Guru. *Design Pattern - Adapter*. 22.03.2022. URL: <https://refactoring.guru/design-patterns/adapter>.
- [61] Author von Refactoring Guru. *Design Pattern - Decorator*. 22.03.2022. URL: <https://refactoring.guru/design-patterns/decorator>.
- [62] Author von Refactoring Guru. *Design Pattern - Iterator*. 22.03.2022. URL: <https://refactoring.guru/design-patterns/iterator>.
- [63] Author von Refactoring Guru. *Design Pattern - Observer*. 22.03.2022. URL: <https://refactoring.guru/design-patterns/observer>.
- [64] Author von Refactoring Guru. *Design Pattern - Singleton*. 21.03.2022. URL: <https://refactoring.guru/design-patterns/singleton>.
- [65] Arthur Ribeiro. *VR Locomotion: How to Move in VR Environment*. 01.04.2022. URL: <https://circuitstream.com/blog/vr-locomotion/>.
- [66] Chuck Robert. *Advantages and Disadvantages of a Queue*. 01.04.2022. URL: <https://www.techwalla.com/articles/advantages-and-disadvantages-of-electronic-filing-systems>.
- [67] Yuval Boger und Ryan A. Pavlik. OSVR. 01.04.2022. URL: [https://osvr.github.io/whitepapers/introduction\\_to\\_osvr/](https://osvr.github.io/whitepapers/introduction_to_osvr/).
- [68] Autoren der Seite Deutscherfachpflege. *TRACHEOSTOMA UND LUFTRÖHRENSCHNITT*. 24.04.2022. URL: <https://deutscherfachpflege.de/wiki/tracheostoma-und-luftroehrenschnitt/>.
- [69] Autor von Source Making. *Design Patterns*. 21.03.2022. URL: [https://sourcemaking.com/design\\_patterns](https://sourcemaking.com/design_patterns).
- [70] Autoren von Sparknotes. *Defining a Polygon*. 04.04.2022. URL: <https://www.sparknotes.com/math/geometry1/polygons/section1/>.
- [71] Autoren des Steam Supports. *Valve Index Systemanforderungen*. 10.03.2022. URL: <https://help.steampowered.com/en/faqs/view/105E-66E3-962A-1577>.

- [72] Author von tutorialspoint. *Design Pattern - Iterator Pattern*. 22.03.2022. URL: [https://www.tutorialspoint.com/design\\_pattern/iterator\\_pattern.htm](https://www.tutorialspoint.com/design_pattern/iterator_pattern.htm).
- [73] Author von tutorialspoint. *Design Pattern - Overview*. 21.03.2022. URL: [https://www.tutorialspoint.com/design\\_pattern/design\\_pattern\\_overview.htm](https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm).
- [74] Autoren von TutorialsPoint. *Computer Graphics Curves*. 04.04.2022. URL: [https://www.tutorialspoint.com/computer\\_graphics/computer\\_graphics\\_curves.htm](https://www.tutorialspoint.com/computer_graphics/computer_graphics_curves.htm).
- [75] Autor von Unity. *GameObjects*. 01.04.2022. URL: <https://docs.unity3d.com/ScriptReference/GameObject.html>.
- [76] Autor von Unity. *Prefabs*. 04.04.2022. URL: <https://docs.unity3d.com/Manual/Prefabs.html>.
- [77] Autoren von Unity. *Canvas*. 12.03.2022. URL: <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html>.
- [78] Autoren von Unity. *Intro to the Unity Physics Engine*. 04.04.2022. URL: <https://learn.unity.com/tutorial/intro-to-the-unity-physics-engine#5eb961abedbc2a0021ab8452>.
- [79] Autoren von unity. *Unity Asset Store Homepage*. 04.04.2022. URL: [https://assetstore.unity.com/?gclid=CjwKCAjw0a-SBhBkEiwApljU0j-ZZTztRl3KUfTURnA5\\_U09dMp9CaJcjFBg91IxN9wCFMuOIfuBwE&gclsrc=aw.ds](https://assetstore.unity.com/?gclid=CjwKCAjw0a-SBhBkEiwApljU0j-ZZTztRl3KUfTURnA5_U09dMp9CaJcjFBg91IxN9wCFMuOIfuBwE&gclsrc=aw.ds).
- [80] Autoren von Unity. *Unity User Manual 2020.3 (LTS)*. 29.03.2022. URL: <https://docs.unity3d.com/Manual/index.html>.
- [81] User vom Unity Forum: LeftyRighty. *Fading in/out GUI text with C Sharp?* 29.03.2022. URL: <https://forum.unity.com/threads/fading-in-out-gui-text-with-c-solved.380822/>.
- [82] Autoren von Unity Technologies. *Unity für VR*. 01.04.2022. URL: <https://unity.com/unity/features/vr>.
- [83] Autoren von Unity Technologies. *Unity Homepage*. 01.04.2022. URL: <https://unity.com/>.
- [84] Autoren von Unity Technologies. *Unity Simulations Pro*. 01.04.2022. URL: <https://unity.com/products/unity-simulation-pro>.
- [85] Autoren von Unity Technologies. *XR Interaction Toolkit*. 02.04.2022. URL: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.9/manual/index.html>.
- [86] Autoren von Viscircle. *Einstiegerguide: Einführung in die 3D Modellierung*. 04.04.2022. URL: <https://viscircle.de/einfuehrung-in-die-3d-modellierung/>.
- [87] Autoren von WikiBooks. *Blender 3D: Noob to Pro/Polygon by Polygon modeling*. 04.04.2022. URL: [https://en.wikibooks.org/wiki/Blender\\_3D:\\_Noob\\_to\\_Pro/Polygon\\_by\\_Polygon\\_modeling](https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Polygon_by_Polygon_modeling).
- [88] Autoren von Wikipedia. *Autodesk 3ds Max*. 04.04.2022. URL: [https://en.wikipedia.org/wiki/Autodesk\\_3ds\\_Max](https://en.wikipedia.org/wiki/Autodesk_3ds_Max).
- [89] Autoren von wikipedia. *Autodesk Maya*. 04.04.2022. URL: [https://en.wikipedia.org/wiki/Autodesk\\_Maya](https://en.wikipedia.org/wiki/Autodesk_Maya).
- [90] Autoren von Wikipedia. *Blender (software)*. 04.04.2022. URL: [https://en.wikipedia.org/wiki/Blender\\_\(software\)](https://en.wikipedia.org/wiki/Blender_(software)).
- [91] Autoren von Wikipedia. *Concept Art*. 04.04.2022. URL: [https://en.wikipedia.org/wiki/Concept\\_art](https://en.wikipedia.org/wiki/Concept_art).

- [92] Autoren von Wikipedia. *Grafische Benutzeroberfläche*. 25.02.2022. URL: [https://de.wikipedia.org/wiki/Grafische\\_Benutzeroberfl%C3%A4che](https://de.wikipedia.org/wiki/Grafische_Benutzeroberfl%C3%A4che).
- [93] Autoren von Wikipedia. *Head-up-Display*. 27.02.2022. URL: <https://de.wikipedia.org/wiki/Head-up-Display>.
- [94] Autoren von Wikipedia. *Loading screen*. 06.03.2022. URL: [https://en.wikipedia.org/wiki>Loading\\_screen](https://en.wikipedia.org/wiki>Loading_screen).
- [95] Autoren von Wikipedia. *ZBrush*. 04.04.2022. URL: <https://en.wikipedia.org/wiki/ZBrush>.

# Abbildungsverzeichnis

4.1	Eine VR-Simulation eines chirurgischen Prozesses . . . . .	21
4.2	Das VR-Spiel Beatsaber . . . . .	21
4.3	Sechs Freiheitsgrade . . . . .	28
4.4	Fortbewegungstechniken für VR . . . . .	29
4.5	Ein VR-Treadmill . . . . .	30
4.6	Beispiel einer Blink Anwendung . . . . .	31
4.7	Windows Einstellungen . . . . .	33
4.8	Screenshot vom Spiel “Overwatch” . . . . .	34
4.9	Screenshot vom VR Spiel “Keep Talking and Nobody Explodes” [50] . . . . .	35
4.10	Unterschiedliche Buttons . . . . .	36
4.11	Ein Beispiel für die 3 Buttons States von Oculus . . . . .	37
4.12	Pinch-and-Pull Komponenten von Oculus . . . . .	37
4.13	Pointers von Oculus . . . . .	37
4.14	Beispiel einer Concept Art eines professionellen Künstlers (Konstantin Maystrenko )[19]) . . . . .	40
4.15	Spline-Modelling für ein Weinglas (links Skelletkurve, rechts fertiges Modell) . . . . .	41
4.16	Darstellung unterschiedlicher Bézierkurven . . . . .	42
4.17	Darstellung eines Modells mittels Bezierkurve in Blender . . . . .	42
4.18	Keyframe Interpolation in Adobe After Effects . . . . .	43
4.19	Darstellung einer NURBS als Kurve und als Fläche . . . . .	43
4.20	Modellierung einer Gesichtshälfte mittels Box-Modeling . . . . .	44
4.21	Topologien eines Kreises (z.B: Seite eines Zylinders, Kugel von der Seite . . . . .	44
4.22	Links:falsche Topologie, Rechts: richtige Topologie . . . . .	45
4.23	Step 1: Aller Anfang ist ein Quad . . . . .	46
4.24	Step 2: Grobe Konturen mit weitern großen Quads . . . . .	46
4.25	Step 3: Einfügen der Details. Quads werden immer kleiner . . . . .	47
4.26	Modell einer fertigen Spritze. Aufgeteilt in Kolben und Zylinder . . . . .	47
4.27	Sculpting Prozess eines Gesichts in ZBrush . . . . .	48
5.1	Konzept einer Programmstruktur . . . . .	54
5.2	Sequenzdiagramm: Darstellung eines Programmstarts . . . . .	55
5.3	Sequenzdiagramm: Kommunikation zwischen Klassen bei Taskstart . . . . .	56
5.4	Klassendiagramm: Vererbungshierarchie der Tasks . . . . .	59
5.5	Unity Inspektor Fenster mit definierter Struktur . . . . .	60
5.6	Spiel-Engine Nutzwertanalyse . . . . .	62

5.7	Skizze des Raum-Layouts . . . . .	63
5.8	Ideen für Statusanzeige . . . . .	66
5.9	Funktionalität des Menüs . . . . .	68
5.10	Darstellung des Menüs . . . . .	68
5.11	Loading Screen Skizze . . . . .	69
5.12	Vorlage der Spritze [44] . . . . .	72
5.13	Vorlage des Manometers[2] . . . . .	73
5.14	Vorlage des HME(feuchte Nase)[42] . . . . .	74
5.15	Vorlage der Trachealkanüle[36] . . . . .	76
5.16	Vorlage der Trachealkompresse[39] . . . . .	77
5.17	Vorlage der Kaniülenbänder[20] . . . . .	78
5.18	Vorlage der Greifanimation und Rigs[3] . . . . .	79
6.1	Erstellung eines Prozesses . . . . .	96
6.2	Erstellen einer Task . . . . .	97
6.3	Erste Test Szene . . . . .	98
6.4	Abschnitt des Raumes . . . . .	100
6.5	Beispiel eines InteractableObjects . . . . .	101
6.6	Ein Compound Object . . . . .	103
6.7	Alte Pinnwand (links) und neues Whiteboard (rechts) . . . . .	111
6.8	Whiteboard - Endstand . . . . .	113
6.9	Hinweis - Grafische Darstellung . . . . .	116
6.10	Hauptmenü . . . . .	116
6.11	Controllerbelegung [59] . . . . .	118
6.12	XRI Default Input Actions in Unity . . . . .	118
6.13	Mesh des Zylinders . . . . .	123
6.14	Mesh des Kolbens . . . . .	124
6.15	Map des Kolbens . . . . .	125
6.16	Fertiges Modell der Spritze . . . . .	125
6.17	Mesh der Hülle . . . . .	126
6.18	Mesh der Kappe . . . . .	126
6.19	Fertiges Modell der feuchten Nase . . . . .	127
6.20	Mesh des Mittel-Körpers . . . . .	128
6.21	Mesh des Druck-Körpers . . . . .	128
6.22	Mesh der Nadel . . . . .	128
6.23	Map des Druckkörpers . . . . .	129
6.24	Map des Mittel-Körpers mit Ziffernblatt . . . . .	129
6.25	Map der Nadel . . . . .	130
6.26	Fertiges Modell des Manometers . . . . .	130
6.27	Mesh des Kanülenrohrs . . . . .	131
6.28	Mesh des Cuffs . . . . .	131
6.29	Mesh des Konnektors . . . . .	132
6.30	Mesh des Schild . . . . .	132
6.31	Mesh des Ballons . . . . .	133
6.32	Fertiges Modell der Kanüle, nur ohne Cuffrohr . . . . .	134

6.33 Animation des Cuffs . . . . .	135
6.34 Demo der Physik des Cuffrohrs . . . . .	140
6.35 Fertiges Modell der Trachealkomresse . . . . .	141
6.36 Fertiges Modell des Bandes . . . . .	142
6.37 Bend-Animation des Bandes . . . . .	143
6.38 Fertiges Modell des Gleitgeltuches . . . . .	144
6.39 Hand Animation . . . . .	146



# Auflistungsverzeichnis

6.1	Code des ProcessHandlers . . . . .	82
6.2	Code des SceneManagers . . . . .	83
6.3	Code des TaskManagers . . . . .	84
6.4	Code des UI-Managers . . . . .	85
6.5	Code der abstrakten Klasse Task . . . . .	86
6.6	Beispielcode eines Events . . . . .	87
6.7	Code der Touchtask . . . . .	88
6.8	Code eines TouchObject . . . . .	88
6.9	Code eines ConnectorObjects . . . . .	89
6.10	Code einer ConnectionTask . . . . .	90
6.11	Code einer PressTask . . . . .	90
6.12	Code eines PressObjects . . . . .	91
6.13	Code eines CompoundParts . . . . .	91
6.14	Code der RemoveTask . . . . .	92
6.15	Code eines ProessObjects . . . . .	92
6.16	Code einer BandControlTask . . . . .	94
6.17	Code eines Prozesses als ScriptableObject . . . . .	96
6.18	Codeausschnitt für das instanziieren der Tools . . . . .	99
6.19	Code für die InteractableObject Klasse . . . . .	102
6.20	Code für die TouchObject Klasse . . . . .	103
6.21	Code für die CompoundObject Klasse . . . . .	104
6.22	Code für die CompoundPart Klasse . . . . .	104
6.23	Anfangscode der Pinnwand . . . . .	109
6.24	Neuer Whiteboard Code mit mehreren Aufgaben . . . . .	110
6.25	Endversion des Whiteboard Codes . . . . .	112
6.26	Code für Darstellung des Textes "Aufgabe abgeschlossen" . . . . .	114
6.27	Code zur Darstellung eines gewählten Hinweises . . . . .	115
6.28	Code zur Anzeige von der Controllerbelegung . . . . .	117
6.29	Code einer Implementierung des Input Systems . . . . .	119
6.30	Code für die Manometer-Animation . . . . .	120
6.31	Code für die Spritze-Animation . . . . .	121
6.32	Berechnung der Punkte . . . . .	137
6.33	Korrektur bei Überdehnung . . . . .	138
6.34	Zeichnen der Quads zwischen den Punkten . . . . .	139
6.35	Kollisionserkennung und -behandlung . . . . .	140