Name: Paul Dächert
Matrikelnummer: 5967009
Studienfach: Informatik BA
Semester: 11
E-Mail: daechert.p@gmail.com

**Bachelorarbeit**

# IsoSpace basierter Avatareditor in Virtual Reality als Teil des VAnnotatoR

Paul Dächert

Abgabedatum: 26.11.2020

angefertigt am
Lehrgebiet Texttechnologie der
Goethe-Universität Frankfurt am Main
Leiter: Prof. Dr. A. Mehler

Bitte dieses Formular zusammen mit der Abschlussarbeit abgeben!

# Erklärung zur Abschlussarbeit

**gemäß § 25, Abs. 11 der Ordnung für den Bachelorstudiengang Informatik vom 06. Dezember 2010:**

Hiermit erkläre ich Herr / ~~Frau~~

Paul Dachert

Die vorliegende Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst.

Frankfurt am Main, den 24.11.2020, P Dächt

Unterschrift der Studentin / des Studenten

# Zusammenfassung

In der aktuellen Zeit gibt es eine Vielzahl an annotierten Texten und anderen Medien. Genauso gibt es verschiedenste Möglichkeiten neue Texte zu annotieren, sowohl manuell als auch automatisch. Es gibt Systeme, die diese Annotationen in andere, visuell ansprechendere Medien umwandeln. Zu diesen Systemen gehören auch die *Text2Scene* Systeme, dort wird ein annotierter Text in eine dreidimensionale Szene umgewandelt. Ein Teil dieser *Text2Scene* Systeme können auch Personen durch Modelle von Menschen darstellen, aber bis jetzt gibt es noch kein System, dass Avatar Modelle selber synthetisieren kann.

Der Fokus dieser Arbeit liegt sowohl darauf eine Schnittstelle bereitzustellen, mit der Avatare mit bestimmten Parametern erstellt werden können, als auch die Möglichkeit diese Avatare in der virtuellen Realität anzuzeigen und zu bearbeiten. Man kann in einer virtuellen Szene die Eigenschaften bestimmter Körperteile anpassen und die Kleidung der Avatare auswählen.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Einleitung

## 1.1 Aufgabenstellung

Mit Hilfe dieser Arbeit soll es möglich sein, Avatare innerhalb der *VAnnotatoR* Applikation [27, 4, 18] zu erstellen, bearbeiten, speichern und laden. Bislang war es mit dem *VAnnotatoR* nur möglich 3D Objekte in virtuellen Räumen zu platzieren. Die Arbeit soll dem Benutzer ermöglichen Texte, in denen Personen vorkommen, in einer dreidimensionalen Szene nachzubauen und anschließend zu annotieren. Dabei soll es eine Vielzahl an Einstellungen geben, um den Avatar anzupassen. Die Avatare sollen basierend auf *ISO-Space* implementiert werden, damit sie später mit Textabschnitten verbunden werden können.

## 1.2 Motivation

In vielen Texten kommen Personen vor. Menschen werden häufig sehr genau beschrieben, da sie viele Merkmale haben. Wenn ein Stuhl in einem Text beschrieben wird, dann wird meistens nur erwähnt, dass das Objekt ein Stuhl ist, manchmal noch die Art oder das Material. Personen können genauso flüchtig erwähnt werden, aber damit der Leser eines Textes sich eine Person gut vorstellen kann muss die Beschreibung detailliert sein. Es gibt Menschen die besondere Körpermerkmale haben, sei es ihre Armlänge, Nasenlänge oder Position ihrer Ohren, keine zwei Menschen sind gleich. Selbst wenn manche Menschen sich ähnlich sehen, tragen sie eventuell komplett andere Kleidung. Es gibt unendlich viele Variationen.

Es gibt bereits *Text2Scene* Systeme, in denen Personen dargestellt werden können und das Projekt, das am weitesten entwickelt wurde ist *WordsEye* [11]. In *WordsEye* können Personen als Avatare dargestellt werden, dazu ein Modell aus der Datenbank geladen. Die Avatare können Posen einnehmen, um besser in die Szene zu passen, zum Beispiel eine Sitzposition. Das Problem ist, dass Avatare nicht generiert werden und deshalb die beschriebene unendliche Vielfalt an Variationen eines Menschen nicht dargestellt werden kann.

Für ein *Text2Scene* System, das maschinelles Lernen benutzt, müssen Trainingsdaten bereitgestellt werden und Resultate des Systems korrigiert werden. Aktuell gibt es noch kein System, dass dies ermöglicht: Korrektur von Annotationen in deren Text Avatare

vorkommen. Da das *Text2SceneVR* Projekt [1] ein maschinelles Lernen System sein soll, ist es notwendig ein Tool zu haben, mit dem Avatare in der Virtual Reality angepasst werden können, um das System zu trainieren.

# 2 Grundlagen

## 2.1 Text2Scene

Ein *Text2Scene* System generiert aus einem Text eine Szene. Zunächst wird der Text annotiert, um die Bedeutung und Rolle der einzelnen Worte und auch die Verbindung der Worte untereinander zu finden. Als nächstes müssen Objekte und Modelle gefunden oder synthetisiert werden für die Worte des Textes, die in einer Szene darstellbar sind. Zum Schluss werden die Objekte in der Szene platziert, dabei wird wieder auf den Text verwiesen, um die relative Position der Objekte untereinander und in der Szene richtig darzustellen.

### 2.1.1 Text2SceneVR

*Text2SceneVR* [1] ist ein Projekt des Texttechnologie Labs der Goethe Universität Frankfurt in der eine 3D Szene aus einem beliebigen Text erstellt werden soll. Dazu wird der Text zunächst mit Hilfe des *TextAnnotator* oder *VAnnotatoR* annotiert und dann daraus entsprechende *ISO-Space* Entitäten erzeugt. Als nächstes sollen die *ISO-Space* Entitäten in einer dreidimensionalen Szene platziert und angezeigt werden. Aktuell ist es bereits möglich, Hypertexte mit dem *VAnnotatoR* [15, 1] zu erstellen. Ein Benutzer kann innerhalb der Virtual Reality Anwendung des *VAnnotatoR* virtuelle Räume erstellen, bearbeiten und Objekte platzieren. Als nächstes können Annotationen an und zwischen den Objekten erzeugt werden.

Szenen sollen nicht wie bei anderen Projekten durch Zusammenstellung der Objekte von 1:1 Mapping erstellt werden, sondern durch ein System mit künstlicher Intelligenz synthetisiert werden. Ein künstliches System braucht für maschinelles Lernen außer Trainingsdaten noch Korrektur. Dies ist der Fokus des Projekts: Die Korrektur der automatisch erzeugten Annotationen zu ermöglichen und dadurch später die KI zu trainieren.

Der Avatareditor baut direkt auf dem Projekt *Text2SceneVR* auf und erweitert die Funktionalitäten durch die Möglichkeit Avatare zu erzeugen und bearbeiten.

### 2.1.2 WordsEye

Das Projekt *WordsEye* [11] ist eines der ersten Systeme zu *Text2Scene* und kann mithilfe von weiteren Arbeiten [10, 26] 3D Szenen aus natürlicher Sprache generieren. Jedoch

ist es nötig, dass die Platzierung der Objekte manuell zur Sprache gemapped werden. Die Bindung ist sehr strikt und erfordert deshalb sehr spezifische Kommandos. *WordsEye* unterstützt auch Avatare, so werden Personen durch beispielsweise einen Namen im Text erkannt und dann entsprechend ein männliches oder weibliches Modell aus der Datenbank benutzt. Die Avatare können in unterschiedlichen Posen mit *Inverse Kinematics* dargestellt werden, solange die Modelle eine Skelett Struktur (Rig) besitzen. Die Posen und auch Gesichtsausdrücke werden dort als *Depictors* gespeichert, darunter würden auch andere Details von Objekten fallen wie zum Beispiel gefärbte Flächen oder Transparenz.

### 2.1.3 Stanford Text2Scene

In den Arbeiten der Stanford Natural Language Processing (NLP) Group [7, 8, 6] liegt der Fokus darauf die Makel von *WordsEye* auszubessern. Menschen lassen oft kleinere Details aus wenn Sie eine Szene beschreiben. Die Arbeiten versuchen diese fehlenden Entitäten und Verbindungen zu identifizieren und dann in den Prozess einzubinden, sodass auch mit nicht vollständiger Informationsgrundlage ein adäquates Ergebnis geliefert werden kann. Aktuell unterstützt das *Text2Scene* System der Stanford NLP Group jedoch keine Avatare.

### 2.1.4 PiGraphs

Einen größeren Fokus auf Interaktion von Menschen und der Umgebung legt die Arbeit PiGraphs [25]. Mit der Arbeit ist es möglich Sätze darzustellen, die eine Relation von einer Person und einem Objekt der Umgebung ausdrücken.

### 2.1.5 Learning the Visual Interpretation of Sentences

Bei der Arbeit von Microsoft [30] geht es hauptsächlich darum räumliche und semantische Relationen in Sätzen visuell darzustellen. Die generierten Szenen sind jedoch nur in 2D und erinnern an Bilder eines Bilderbuches. In den Beispielen der Arbeit werden auch zwei Personen dargestellt ”Jenny” und ”Mike”, jedoch liegt der Fokus der Arbeit auf der Relation der beiden Personen im Satz.

### 2.1.6 Language-Driven Synthesis of 3D Scenes from Scene Databases

Das Projekt *”Language-Driven Synthesis of 3D Scenes from Scene Databases”* [17] geht anders an die Erstellung einer Szene von Text heran. Es werden mehrere Datenbanken von Subszenen benutzt. Erst wird der Text geparsed und in einen semantischen Szene Graph umgewandelt, dann werden passende Subszenen aus den Datenbanken geladen, daraufhin werden die Szenen noch um Objekte erweitert, die vom Kontext des Textes

abhängig sind und zum Schluss werden die Subszenen zu einer großen Szene zusammengesetzt. Die erstellten Szenen bestehen aber ausschließlich aus Objekten und es können keine humanoiden Avatare damit erstellt werden.

## 2.2 Unity

Die Unity[1] Spiel-Engine wird von Unity Technologies entwickelt, die ihren Hauptsitz in San Francisco haben. Mit der Laufzeit- und Entwicklungsumgebung ist es möglich Computerspiele oder andere Anwendungen mit 3D Grafik zu entwickeln. Unity bietet viele Features wie eine eigene Rendering Engine und Nutzung der nVidia PhysX Engine, um physikalische Kräfte zu simulieren. Entwickler können durch Drag & Drop Szenen mit Objekten befüllen, deren Eigenschaften anpassen oder eigene Skripte schreiben und den Objekten zuordnen. Genauso können auch Skripte Objekte in den Szenen platzieren und modifizieren.

Der Avatareditor wurde in der Entwicklungsumgebung der Unity-Engine entwickelt.

## 2.3 UMA

Unity Multipurpose Avatar (UMA) [28] wird von *UMA steering group* entwickelt und ist ein Asset im Unity Assert Store. *UMA* ermöglicht es Avatare zur Laufzeit zu erstellen und zu bearbeiten. Im Asset sind ebenfalls einige Beispielmodelle und -ressourcen enthalten. Die Lizenz des *UMA* Assets ist nach dem Eintrag des GitHub Repositories[2] die MIT Lizenz.

*UMA* wird vom Avatareditor benutzt, um Avatare anzuzeigen und zu modifizieren. Außerdem werden die mitgelieferten Modelle und Ressourcen benutzt.

[1] https://unity.com/
[2] https://github.com/umasteeringgroup/UMA

# 3 Stand der Wissenschaft und Technik

## 3.1 ShapeNet

In den Arbeiten zu *ShapeNet* [9, 24] wird eine stark annotierte Datenbank bestehend aus 3D CAD Modellen von Objekten vorgestellt. Die Modelle stammen aus zwei öffentlichen Archiven: Trimble 3D Warehouse[1] und Yobi3D (seit 31.01.2019 geschlossen). Alle Modelle sind kategorisiert und organisiert nach den *WordNet* Klassifizierungen der Universität Princeton [29, 12]. Die Worte in *WordNet* werden in Synonyme Sets (*Synsets*) gruppiert und die 117000 *Synsets* sind mit anderen *Synsets* über begriffliche Relationen verbunden. *Synsets* haben außerdem einen Namen und meistens noch eine kurze Beschreibung, wie die Wörter des *Synsets* benutzt werden.

## 3.2 ISO-Space

*ISO-Space* [22, 23, 21] stellt ein Framework für Annotierung zur Verfügung. Mit *ISO-Space* können räumliche und temporäre Informationen aus Texten gewonnen werden. Aus einem Text können *Spatial Entities* erstellt werden, diese Entitäten sind durch Links verbunden, welche die Relation zweier oder mehrerer Entitäten beschreibt. Die Links können unterschiedliche Informationen darstellen:

**Qualitive Spatial Links (QSLink)** beschreiben topologische Relationen wie zum Beispiel den Pfad den eine Person geht, dabei wäre der Link die Verbindung von Start und Endpunkt. *QSLinks* haben das Attribut `relType`, um den Typ der Relation zu spezifizieren. Die möglichen Typen sind die `RCC8+` Typen (siehe Tabelle 3.1).

**Orientation Links (OLinks)** beschreiben nicht-topologische Relationen wie zum Beispiel die Relation eines Objektes mit einem Tisch *Das Buch liegt **auf** dem Tisch.*

**MoveLinks** beschreiben Bewegungen von Entitäten wie zum Beispiel *Der Mann **verließ** den Laden.*

[1] https://3dwarehouse.sketchup.com/

| Typ | Beschreibung | Beispiel |
|---|---|---|
| DC | getrennt | der *Geldautomat* außerhalb des *Supermarktes* |
| EC | äußerlich vebunden | das *Buch* auf dem *Schrank* |
| PO | teilweise überlappend | *Deutschland* und *Europa* |
| EQ | gleich | *Das Weiße Haus* und *Pennsylvania Avenue 1600* |
| TPP | tangentialer teil von | die *Küste* von *Frankreich* |
| TPPi | tangentialer teil invers | (invers zu TPP) |
| NTPP | nicht-tangentialer teil | eine *Insel* auf dem *See* |
| NTPPi | nicht-tangentialer teil invers | (invers zu NTPP) |
| IN | Disjunktion von TTP und NTTP | das *Bücherregal* im *Raum* |
| OUT | Disjunktion von EC und DC | - |

Tabelle 3.1: RCC8+ Linktypen

*ISO-Space* ist Teil des *Semantic Annotation Framework* (SemAF) [16, chap. 4.2] und beinhaltet einen weiteren Link Typ, der besonders für diese Arbeit sehr wichtig ist:

**Metalink** beschreibt keine räumliche oder zeitliche Relation und gehört deshalb nicht zum *ISO-Space*. Ein Metalink beschreibt eine Relation zwischen *Spatial Entities* und hat dabei einen von drei Typen:

- `COREFERENCE`: Beide *Spatial Entities* referenzieren das selbe Objekt
- `SUBCOREFERENCE`: Eine *Spatial Entity* 1 ist Teil eines größeren Subsets *Spatial Entity* 2
- `SPLITCOREFERENCE`: Zwei *Spatial Entities* werden mit einer *Spatial Entity* zusammengefasst. Der Link zwischen der zusammenfassenden *Spatial Entity* und der erstellten Hilfs-*Spatial Entity* ist dann vom Typ `SPLITCOREFERENCE`

## 3.3 TextAnnotator

Der *TextAnnotator* [2, 3, 5] ist ein browserbasiertes, multi-annotations System des Texttechnologie Labs der Goethe Universität Frankfurt. Schon vor der Entwicklung des *TextAnnotator* gab es Annotationssystem, jedoch haben sie alle unterschiedlichen Fokus gehabt und nicht alle Features vereint. Es gab noch kein System, dass die Features Multi-Authoring, Multi-Document, Multi-Annotation, Multi-Plattform, Multi-Analysis und simultane Kollaboration vereint hat. Der *TextAnnotator* erlaubt es mehreren Benutzern unabhängig voneinander an unterschiedlichen oder auch den selben Dokumenten zu arbeiten. Dokumente können in Repositories nach verschiedenen Merkmalen strukturiert werden und werden in einer Datenbank abgespeichert. Es können

eine Vielzahl von Annotationsarten kombiniert und durchgeführt werden. Der *TextAnnotator* stellt eine API zur Verfügung durch die es möglich ist, das System auf einer beliebigen Plattform zu nutzen. Es gibt die Möglichkeit Annotationen nachträglich zu analysieren, zum Beispiel auf die Qualität der Annotation mit dem *inter-annotation agreement* [19]. Mehrere Benutzer können gleichzeitig ein Dokument gemeinsam annotieren, diese Funktion kann bei Bedarf auch ausgeschaltet werden.

Die Dokumente des TextAnnotators werden von dem *ResourceManager* verwaltet und bereitgestellt. Der *TextAnnotator* führt Annotationen basierend auf Unstructured Information Management Architecture (UIMA) [13] aus, daher müssen Texte im UIMA Format vorliegen oder zum UIMA Format vom *TextImager* [14] konvertiert werden.

## 3.4 VAnnotatoR

Der *VAnnotatoR* [27, 4, 18] ist eine Virtual Reality Applikation, die mit Unity entwickelt wurde. Mit dem System ist es möglich die Funktionen des *TextAnnotator* in einer virtuellen Umgebung zu benutzen und vor allem darzustellen. Texte können normal als mehrere Wörter angezeigt werden, aber auch als Graphen, um Annotationen besser darzustellen und zu bearbeiten. Mit Hilfe des *VAnnotatoR* können auch Objekte aus dem *ResourceManager* geladen und dargestellt werden. Der Avatareditor baut auf dem Framework des *VAnnotatoR* auf und stellt neue Funktionalitäten und Schnittstellen bereit.

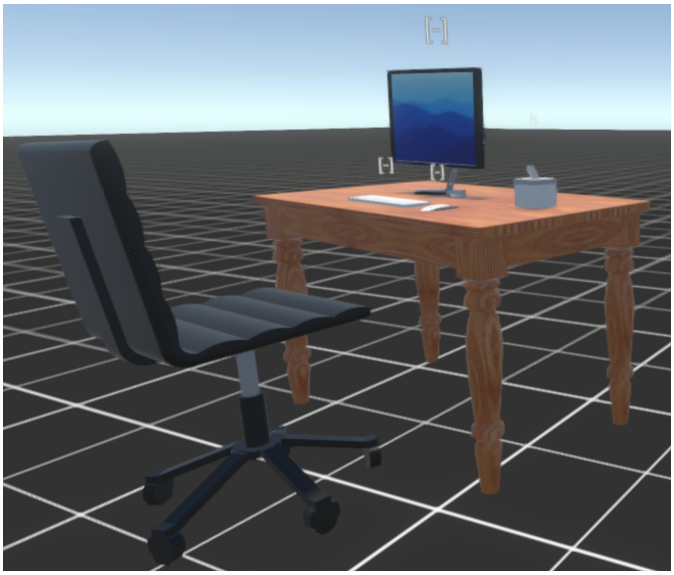


Abbildung 3.1: Beispiel *VAnnotatoR*

# 4 Konzeption

## 4.1 Datenstruktur

Die benutzte Datenstruktur ist die vorgestellte Datenstruktur von *ISO-Space*. Ein Avatar ist auf mehrere *ISO-Space* Entitäten aufgeteilt. Eine *Spatial Entity* ist dabei immer mit einer anderen *Spatial Entity* durch einen *MetaLink* des Typs `SUBCOREFERENCE` verbunden. Die Attribute der *Spatial Entities* werden als `IsoObjectAttribute` abgespeichert und mit den Entitäten verbunden. Ein Avatar besteht dabei aus 44 verschiedenen Attributen, die den Körper verändern und 4/9 (weibliches/männliches Modell) Slots für Kleidungsstücke. Die Datenstruktur kann in der Abbildung 4.1 auf Seite 11 betrachtet werden.

### 4.1.1 Root

Die Wurzel des Avatars hält grundlegende Informationen über den Avatar:

**Model** entspricht der ”race” von *UMA*. Aktuell gibt es nur zwei verschiedene Modelle, die in der Anwendung eingebunden sind, ein Mann und eine Frau.

**SkinColor** ist die Hautfarbe des Avatars als Hexadezimal String.

**Upper-/LowerMuscles & Weight** beeinflussen die Statur des Avatars.

### 4.1.2 Wardrobe

Ein Avatare hat mehrere Slots für Kleidung, dabei werden Frisuren, Augen und Gesichtsbehaarung ebenfalls als Kleidung gespeichert. Es gibt verschiedene Oberteile, Unterteile und Unterwäsche die ausgewählt werden kann.

### 4.1.3 Head

Beim Kopf kann man die Breite und Größe einstellen sowie die Stirn in Größe und Breite anpassen.

#### 4.1.4 Torso

Der Torso beinhaltet Variablen für die Breite des Nackens, Brustgröße und die Größe des großen Gesäßmuskels.

#### 4.1.5 Legs

An den Beinen kann man die Länge, den Beinabstand und die Fußgröße einstellen.

#### 4.1.6 Ears

Die Ohren kann man in Größe, Rotation und Position anpassen.

#### 4.1.7 Face

Das Gesicht umfasst Wangengröße und Position, sowie Position der unteren Wange.

#### 4.1.8 Arms

Bei den Armen kann die Länge und Breite verändert werden, an den Unterarmen ebenso Länge und Breite, bei den Händen die Größe.

#### 4.1.9 Eyes

Augen haben Variablen für Größe, Rotation und Position.

#### 4.1.10 Nose

Die Nase hat mehrere Einstellmöglichkeiten. Veränderbar sind Größe, Plattheit, Position, Krümmung, Steigung, Ausprägung und Breite.

#### 4.1.11 Mouth

Beim Mund können die Mundbreite, die Lippengröße, Kieferbreite und -größe, Kinnbreite, Kinngröße und Ausprägung des Kinns, sowie Unterkiefergröße eingestellt werden

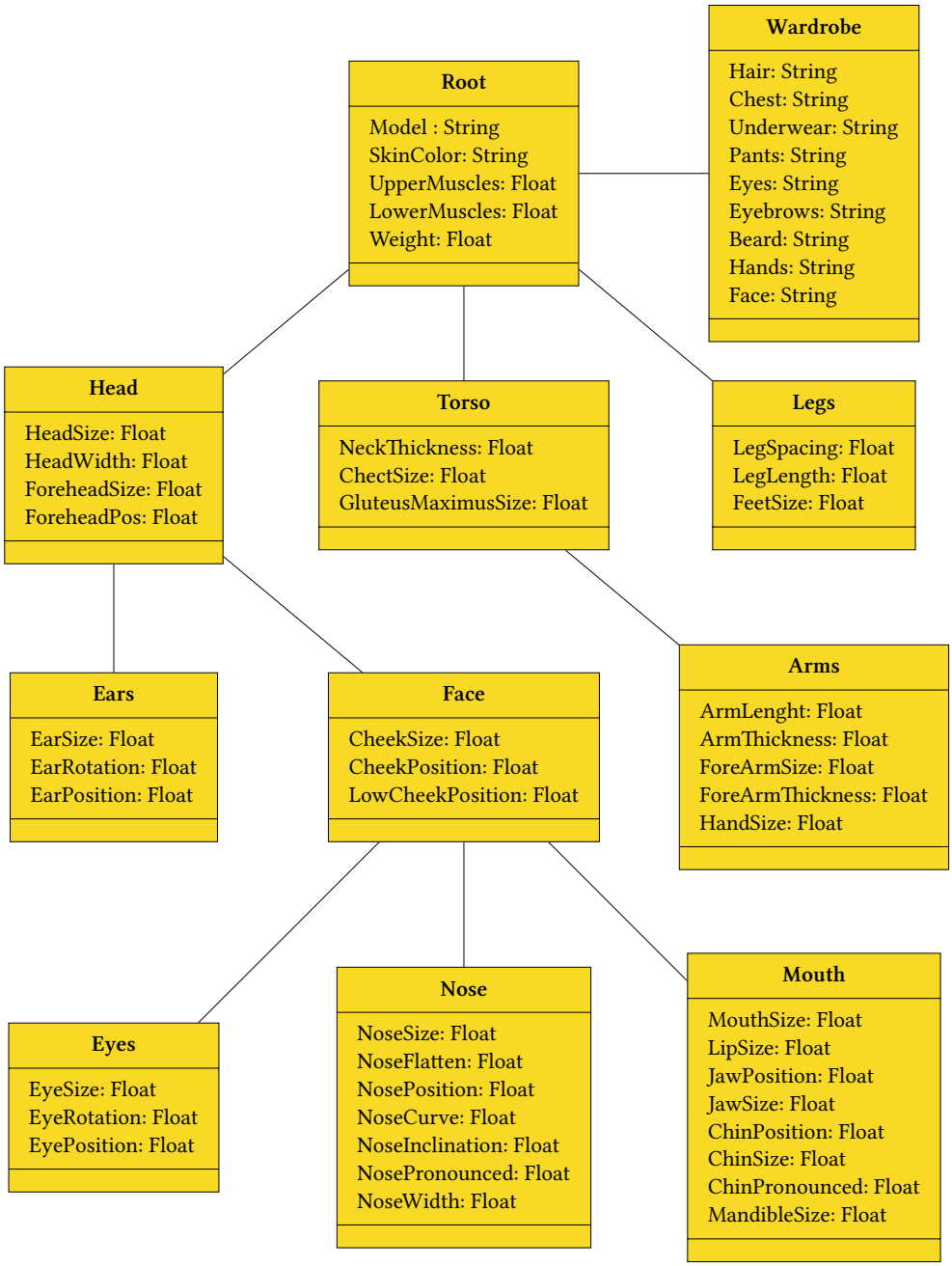



Abbildung 4.1: Datenstruktur
Diese Abbildung zeigt die Datenstruktur, die Kästen sind dabei *Spatial Entities* und die Attribute sind die *Object Features* der *Spatial Entities*

## 4.2 Benutzeroberfläche

Die Benutzeroberfläche sollte ähnlich zu der bereits existierenden Benutzeroberfläche des *Text2SceneVR* Projektes sein. Für *ShapeNet* Objekte gibt es ein Panel, mit dem sich Informationen über das Objekt auslesen lassen, außerdem gibt es die Möglichkeit die Position, Rotation und Größe des Objektes anzupassen. Zunächst wurde versucht das sogenannte *AnnotationObjectPanel* um einen Tab zu erweitern mit dem man Avatare bearbeiten kann, aber relativ bald bemerkt, dass es besser wäre ein eigenes Panel zu erstellen. Das Aussehen sollte so wie beim *AnnotationObjectPanel* sein. Die Hierarchie des Avatars sollte dabei beibehalten werden mit mehreren Tabs und Subtabs. Die Variablen sollen als Eingabefeld mit dem aktuellen Wert dargestellt und mit zwei Pfeilen dekrementiert und inkrementiert werden.

## 4.3 Verknüpfung von Avatar & Text

Damit das Tool auch komplett in den *VAnnotatoR* eingebunden ist, muss es die Möglichkeit geben Avatare und deren Körperteile mit Textstellen zu verbinden. Da die Avatare auf der *ISO-Space* Datenstruktur aufbauen sollen ist das ohne großen Aufwand möglich. Die *Spatial Entities* werden bereits durch die Vorarbeit an *Text2SceneVR* [1] im *VAnnotatoR* angezeigt, es muss nur noch ein passendes Label gewählt werden, durch welches die Körperteile in der Konsole identifizierbar sind.

# 5 Technische Realisierung

### 5.0.1 Avatar Klassen

Ich habe das Datenmodell so entwickelt, dass es einfach ist neue Attribute, Kleidung oder sogar Modelle einzuführen.

#### UMAISOEntity

Die Klasse `UMAISOEntity` repräsentiert eine `IsoSpatialEntity` also zum Beispiel die Entität "Root". Die Attribute werden in einem Dictionary von `UMAProperties` und deren Wert als String gespeichert. `UMAProperty` ist eine Klasse, die eine statische Instanz aller möglichen Attribute hält wie zum Beispiel "model".

#### UMAISOObject

Ein Avatar Objekt in der Szene hat immer eine `UMAISOObject` Komponente. In der Klasse werden alle `UMAISOEntities` des Avatars gehalten und sind für andere Komponenten verfügbar. Die Komponente kümmert sich außerdem um die Initialisierung der Attribute und das Erstellen der Spatial Entities auf dem Server, wenn ein neuer Avatar erstellt wird.

#### UMAController

Der `UMAController` ist die Komponente, die den *UMA* anpassen kann. Wenn ein Kleidungsstück angepasst oder die Ohrengröße verändert werden soll, dann wird eine Funktion des `UMAControllers` aufgerufen und der Avatar wird direkt neu geladen.

#### UMAManagerInteface

Das `UMAManagerInterface` hält alle Avatare, die aktuell im View des Dokumentes sind. Außerdem ist es dafür da, die Initialisierung der `UMAISOObjects` auszuführen, wenn Avatare geladen werden.

#### UMAWardrobe & UMAWardrobeItem

Mit der `UMAWardrobe` lässt sich festlegen, welche Kleidungsstücke (`UMAWardrobeItem`) in einem Slot verfügbar sind. Das Modell des Avatars ist auch relevant, da die Kleidungsstücke eines *UMA* für ein bestimmtes Modell gemacht sind.

### UMAModel

Die Klasse für die verschiedenen Modelle (bei *UMA*"race" genannt) ist `UMAModell`. Dort werden alle aktuell verfügbaren Modelle festgehalten und können zum Beispiel dem `UMAController` übergeben werden, um das Modell des Avatars zu ändern.

Abbildung 5.1: Beispiel Avatare

### 5.0.2 Avatar Object Panel

Für das Bearbeiten der Avatare wurde eine neue Konsole entwickelt die mehrere Tabs für die vielen Attribute des Avatars hat. Die Datenstruktur der Tabs ist wie die Datenstruktur der Avatare, abstrakt und leicht skalierbar. Die Tab Komponenten erben alle von einer Superklasse `AvatarObjectPanelTab` in der gemeinsame Attribute wie der Tab Button, die Subtabs, der aktive Subtab, der Parenttab und die `InteractieShapeNetObject` Komponente des Avatars definiert sind. Die Tabs haben alle eine Funktion um sie aktiv zu setzten, die Werte der Slider in dem Avatarobjekt zu speichern und sich zu Initialisieren. Der ganze Avatar kann durch einen Speicher Knopf jederzeit im Dokument gespeichert werden.


Abbildung 5.2: Avatar Object Panel

### 5.0.3 Interactive Slider & Interactive Selection Menu

Um eine intuitive Bedienung des Avatars zu ermöglichen wurden zwei neue Komponenten und Prefabs erstellt:

**Interactive Slider** ist die Komponente mit der man den Werte einer `UMAISOProperty` verändert und den Zahlenwert speichert. Die Komponente kann mit einem aktuellen, minimalen und maximalen Wert, sowie einer Schrittgröße initialisiert werden. Zusätzlich hat die Komponente noch ein delegate `OnSelectionChangeEvent`, dieses wird aufgerufen, wenn der gespeicherte Wert verändert wird. Das Prefab besteht aus einem Display, welches den aktuellen Wert anzeigt und zwei Buttons zum Erhöhen und Verringern des Wertes. Das Display kann auch ausgewählt werden, darauf öffnet sich eine Tastatur mit der ein Wert direkt eingegeben werden kann.


Abbildung 5.3: Interactive Slider

**Interactive Selection Menu** funktioniert ähnlich zum `InteractiveSlider`. Die Komponente wird mit einer Liste von Auswahlmöglichkeiten als String und dem aktuell ausgewählten Objekt initialisiert. Die Bedienung ist sonst wie bei `InteractiveSlider` ohne die Displayauswahl.


Abbildung 5.4: Interactive Selection Menu

# 6 Ergebnisse und Evaluation

## 6.1 Anbindung Text2Scene Konsole

In der Builderconsole habe ich einen neuen Tab angelegt, den `AvatarTab`. Es gibt aktuell nur einen Button, der einen neuen Avatar erstellt.

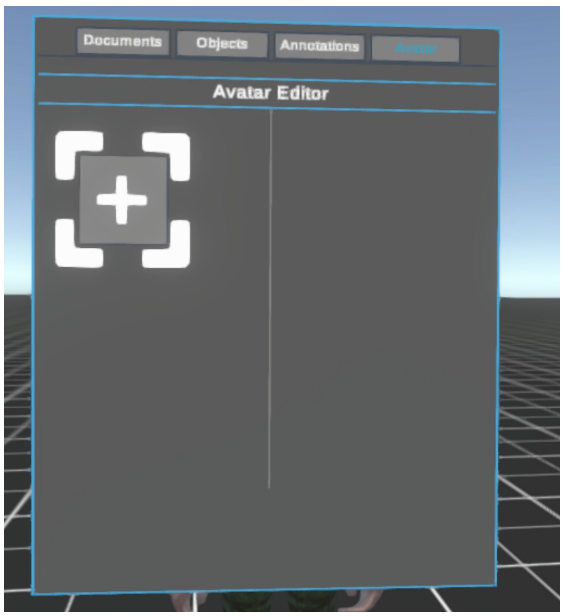



Abbildung 6.1: Builderconsole Avatartab

## 6.2 Avatareditor

Der Avatareditor benutzt die beschriebene Komponente `AvatarObjectPanel` und besteht aus sechs Haupttabs. Die Tabs sind in mehrere Kategorien aufgeteilt, damit die Konsole nicht zu groß wird und die Slider einfacher auffindbar sind. Die Kategorien sind wie folgt:

**Orientation** Der Tab Orientation ist hauptsächlich vom `AnnotationObjectPanel` übernommen und war ursprünglich dazu entwickelt die Position, Rotation und Skalierung von Avataren zu ändern. Im Laufe des Avatareditor Projektes wurde das Grundsystem verändert und die Funktion für die *ShapeNet* Objekte überarbeitet. Die Funkionalitäten des Orientation Tabs werden bei nächster Gelegenheit ebenfalls überarbeitet und angepasst.

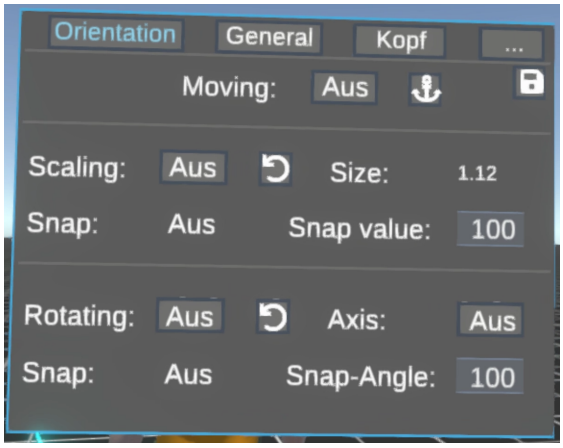



Abbildung 6.2: Orientationtab

**Allgemein** Im ersten Tab lassen sich allgemeine Attribute des Avatars einstellen. Beim Modell kann man zwischen dem männlichen und dem weiblichen Modell wählen, Muskeln und Gewicht können verändert werden und zusätzlich noch die Hautfarbe. Aktuell gibt es fünf verschiedene Hautfarben zur Auswahl.

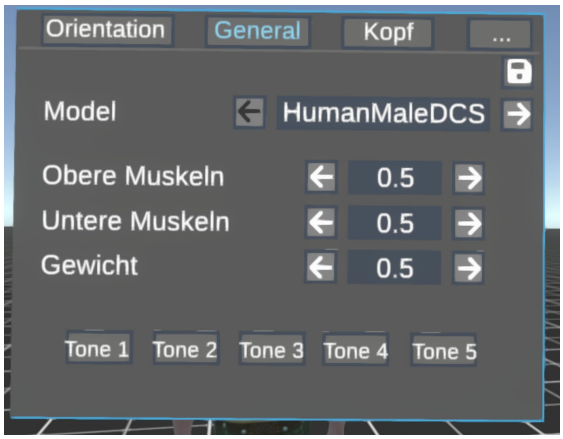
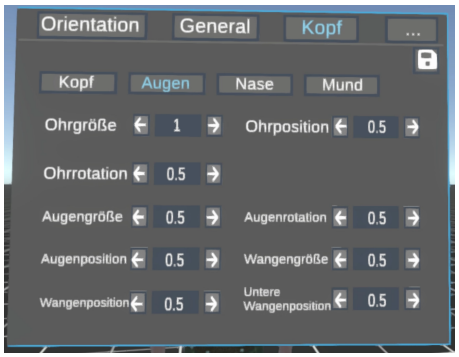



Abbildung 6.3: Generaltab

**Kopf** Der Kopf Tab hat vier Subtabs: Der erste ist für generelle Kopf Attribute wie die Kopfgröße und -breite. Es gibt einen Tab für die Augen und Ohren, einen für die Nase und der letzte ist für den Mund.


(a) Kopf Allgemein




(b) Augen&Ohren


(c) Nase


(d) Mund

Abbildung 6.4: Kopftabs

**Oberkörper** Der Tab für den Oberkörper beinhaltet Einstellungen für Nacken, die Brust und die Arme.


Abbildung 6.5: Oberkörper

**Unterkörper** Beim Unterkörper Tab lassen sich der große Gesäßmuskel, Beine und Füße bearbeiten.


Abbildung 6.6: Unterkörper

**Kleidung** Im letzten Tab kann man die Kleidung des Avatar verändern. Der Tab hat beim weiblichen Modell eine Seite mit Sliedern, bei Auswahl des männlichen Modells gibt es eine zusätzliche Seite, da es für das männliche Modell mehr Slots gibt bei denen Kleidung von *UMA* bereitgestellt wird.


(a) Tab 1 (b) Tab 2

Abbildung 6.7: Kleidungstabs

## 6.3 Annotation der ISO-Space Entities

Die `UMAISOEntities` des Avatars werden im *VAnnotatoR* wie andere `IsoSpatialEntities` angezeigt und können wie gewohnt zu Annotation des Textes benutzt werden.

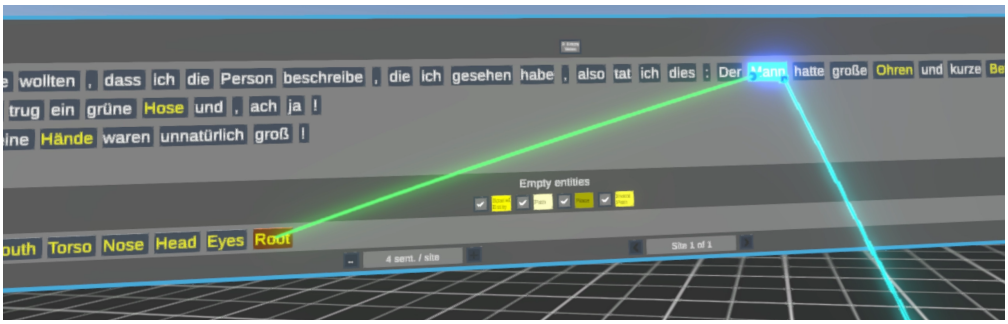



Abbildung 6.8: Beispiel Annotation

## 6.4 Evaluation

Um den Avatareditor zu evaluieren wurde der *Usability Metric for User Experience* (UMUX) (Finstad, 2010) Test mit 7 Probanden durchgeführt. Die Aufgaben des Tests bestanden daraus einen Avatar zu erstellen, nach einem Text anzupassen und dann den Text entsprechend zu annotieren. Der Text bestand aus einem kurzen Satz der eine Person beschreibt: *Sie wollten, dass ich die Person beschreibe, die ich gesehen habe, also tat ich dies: "Der Mann hatte große Ohren und kurze Beine. Er trug ein grüne Hose und, ach ja! Seine Hände waren unnatürlich groß!"*. Als nächstes sollten die Probanden die folgenden Fragen mit einer Skala von 1-7 bewerten, wobei 7 volle Zustimmung bedeutete:

1. Die Möglichkeiten des Avatareditors entsprechen meinen Anforderungen.
2. Den Avatareditor zu benutzen ist eine frustrierende Erfahrung.
3. Der Avatareditor ist einfach zu benutzen.
4. Ich brauche zu viel Zeit um bestimmte Einstellung im Avatareditor zu finden.

Die Ergebnisse (Abbildung 6.9 auf Seite 23) der Evaluation sind überwiegend positiv. Die Bedienung des Avatareditors scheint nicht allen einfach zu fallen, dies könnte aber damit zusammenhängen, dass manche Teilnehmer wenig bis gar keine Erfahrung mit Virtual Reality hatten und auch nur manche den *VAnnotatoR* vorher benutzt haben. Die Ergebnisse von Frage 4 lassen darauf schließen, dass manche Einstellungen in Kategorien aufgeteilt wurden, die sich dem Benutzer nicht spontan erschlossen haben, wenn man danach sucht.

Da die ganze Welt aktuell mit der Bewältigung der Corona Pandemie beschäftigt ist und Versammlungen mehrerer Menschen stark beschränkt wird, konnten nur Personen zur Evaluation eingeladen werden, die eine Virtual Reality Brille besitzen und von zu Hause an der Evaluation teilnehmen können. Deshalb ist die Anzahl der Teilnehmer der Evaluation entsprechend gering.

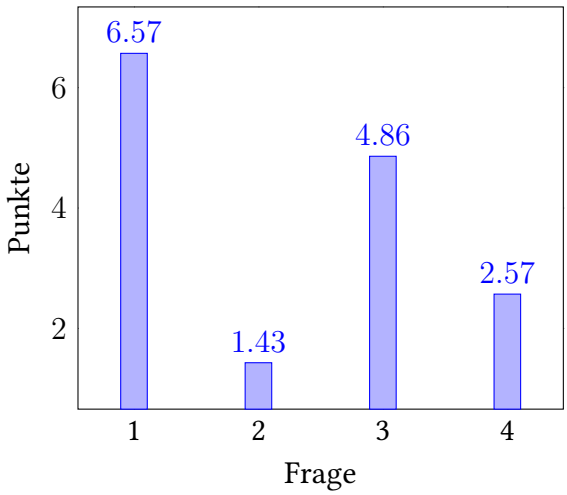



Abbildung 6.9: Fragebogen Ergebnisse
Die Ergebnisse des UMUX Tests mit 7 Teilnehmern

Abbildung 6.10: Beispiel Avatar der Evaluation

# 7 Schlussfolgerungen und Ausblick

## 7.1 Fazit

Im Laufe der Arbeit wurde ein funktionsfähiger Avatareditor in Virtual Reality erstellt und die Avatare sind kompatibel mit dem *VAnnotatoR*. Avatare können innerhalb eines Dokuments und Views mit Hilfe des Avatartabs in der Builderconsole erstellt werden (siehe Kapitel 6.1). Mit dem neu entwickelten Avatareditor kann der Körper eines Avatar mit 44 verschiedenen Einstellungen verändert werden und Bekleidung in 4/9 (weibliches/männliches Modell) verschiedene Kleidungsslots ausgewählt werden (siehe Kapitel 4.1 und Kapitel 6.2). Die erstellten und bearbeiteten Avatare können außerdem gespeichert und auch wieder geladen werden. Damit ist ein weiterer Grundbaustein des *Text2SceneVR* Projektes gelegt.

## 7.2 Offene Probleme

### 7.2.1 Anzahl an Auswahlmöglichkeiten

Da nur die Kleidungsassets benutzt werden, die von *UMA* als Beispiele bereitgestellt werden, gibt es leider nicht viele Auswahlmöglichkeiten. Manche Kleidungsslots haben nur eine Auswahlmöglichkeit, manchmal nur für ein Modell. Es gibt im Unity Assetstore jedoch mehrere Assets die neue Kleidungsstücke speziell für *UMA* zur Verfügung stellen, leider ist der größte Teil davon kostenpflichtig. Man könnte jedoch für die zukünftige Entwicklung eine Person beauftragen spezifische Kleidungsstücke zu erstellen, falls noch mehr Auswahl gewünscht wird. Man kann leider nicht einfach irgendwelche Kleidungsstücke benutzen, die als 3D Modell verfügbar sind. Die Assets müssen extra für *UMA* angepasst werden, dazu gibt es jedoch Guides auf der Wiki Webseite von *UMA*[1].

### 7.2.2 Automatisierung

Die Erstellung der Avatare geschieht zur Zeit nur manuell. Es könnten jedoch auch Avatare erstellt werden indem die nötigen *Spatial Entities* mit ihren *Object Features* und Links in einem Dokument angelegt werden. Die Avatare würden dann bei Aufruf des Dokumentes in dem *VAnnotatoR* geladen werden. Wenn die *Text2SceneVR* Applikation

[1] http://umawiki.secretanorak.com/Main_Page

ihre künstliche Intelligenz bekommt, dann müsste das System noch lernen, wie man Avatare erstellt und bei Bedarf könnte man noch eine einfachere Schnittstelle anlegen.

### 7.2.3 Zeitliche Veränderung

Das Projekt *Text2SceneVR* plant in der Zukunft auch ISOTimeML [20] zu benutzen, deshalb muss man überlegen, wie und in welchem Kontext die Zeit Avatare beeinflusst. Ein Beispiel dafür wäre die Evolution eines Menschen mit der Zeit: Ein Baby wird geboren, wächst mit der Zeit zu einem Kind heran und entwickelt sich immer weiter zu einem Jugendlichen, Erwachsenen und schließlich zu einem alten Menschen weiter. Eine solche Evolution eines Avatars kann man leicht in einem Text finden, die Frage stellt sich nun: *Wie stelle ich Evolution dar?*. Es gibt mehrere Lösungen die mir dazu einfallen, aber das Problem der Zeit bleibt vorerst bestehen.

## 7.3 Anregungen für weiterführende Arbeiten

### 7.3.1 Farben

*UMA* ermöglicht es Farben zu bearbeiten. Es gibt mehrere Slots für Farben und aktuell wird nur einer benutzt, die Hautfarbe. Die Kleidungsstücke sind in einfärbbare Flächen unterteilt zum Beispiel kann bei einem gepunkteten Tanktop die Punkte und der Basisteil unabhängig voneinander eingefärbt werden. Die Datenstruktur könnte um weitere Entitäten und Attribute erweitert werden, um die Farben der Kleidung abzuspeichern. Dann könnte man im Avatar Panel für die Kleidungsslots ein Farbauswahlwerkzeug bereitstellen. Man könnte auch ein unabhängiges Werkzeug (zum Beispiel einen Pinsel oder Farbeimer) einführen, mit dem man die Kleidungsstücke einzeln auswählen und dann mit einem extra Fenster einfärben kann.

### 7.3.2 Posen

Damit die Avatare in eine beliebige Szene passen, muss es möglich sein, ihre Pose zu verändern. Aktuell haben alle Avatare eine Standardpose. Man könnte entweder mehrere einfache Posen erstellen und bei Bedarf auf die Avatare anwenden, zum Beispiel eine Pose für Sitzen, Stehen, Laufen, usw. oder man macht die Posen dynamisch. Bei der zweiten Variante könnte man die Position und Rotation der Gelenke des Avatars in der Datenbank speichern.

# Literaturverzeichnis


[1] Giuseppe Abrami, Alexander Henlein, Attila Kett, and Alexander Mehler. Text2scenevr: Generating hypertexts with vannotator as a pre-processing step for text2scene systems. In *Proceedings of the 31st ACM Conference on Hypertext and Social Media*, HT '20, page 177–186, New York, NY, USA, 2020. Association for Computing Machinery.

[2] Giuseppe Abrami, A Mehler, P Helfrich, and E Rieb. Textannotator: A browser-based framework for annotating textual data in digital humanities. *Proceedings of the Digital Humanities Austria*, 2018.

[3] Giuseppe Abrami, Alexander Mehler, Andy Lücking, Elias Rieb, and Philipp Helfrich. Textannotator: A flexible framework for semantic annotations. In *Workshop on Interoperable Semantic Annotation (ISA-15)*, page 1, 2019.

[4] Giuseppe Abrami, Christian Spiekermann, and Alexander Mehler. Vannotator: Ein werkzeug zur annotation multimodaler netzwerke in dreidimensionalen virtuellen umgebungen. In *Proceedings of the 6th Digital Humanities Conference in the German-speaking Countries, DHd 2019*, 2019.

[5] Giuseppe Abrami, Manuel Stoeckel, and Alexander Mehler. Textannotator: A uima based tool for the simultaneous and collaborative annotation of texts. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 891–900, 2020.

[6] Angel Chang, Will Monroe, Manolis Savva, Christopher Potts, and Christopher D Manning. Text to 3d scene generation with rich lexical grounding. *arXiv preprint arXiv:1505.06289*, 2015.

[7] Angel Chang, Manolis Savva, and Christopher D Manning. Interactive learning of spatial knowledge for text to 3d scene generation. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 14–21, 2014.

[8] Angel Chang, Manolis Savva, and Christopher D Manning. Learning spatial knowledge for text to 3d scene generation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2028–2038, 2014.

[9] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[10] Bob Coyne, Alex Klapheke, Masoud Rouhizadeh, Richard Sproat, and Daniel Bauer. Annotation tools and knowledge representation for a text-to-scene system. In *Proceedings of COLING 2012*, pages 679–694, 2012.

[11] Bob Coyne and Richard Sproat. Wordseye: an automatic text-to-scene conversion system. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 487–496, 2001.

[12] Christiane Fellbaum. Wordnet. *The encyclopedia of applied linguistics*, 2012.

[13] David Ferrucci, Adam Lally, Karin Verspoor, and Eoric Nyberg. Unstructured information management architecture (uima) version 1.0. oasis standard. *Standard, OASIS*, 2009.

[14] Wahed Hemati, Tolga Uslu, and Alexander Mehler. Textimager: a distributed uima-based system for nlp. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pages 59–63, 2016.

[15] Alexander Henlein, Giuseppe Abrami, Attila Kett, and Alexander Mehler. Transfer of ISOSpace into a 3D environment for annotations and applications. In *16th Joint ACL - ISO Workshop on Interoperable Semantic Annotation PROCEEDINGS*, pages 32–35, Marseille, May 2020. European Language Resources Association.

[16] Nancy Ide and James Pustejovsky. *Handbook of linguistic annotation*. Springer, 2017.

[17] Rui Ma, Akshay Gadi Patil, Matthew Fisher, Manyi Li, Sören Pirk, Binh-Son Hua, Sai-Kit Yeung, Xin Tong, Leonidas Guibas, and Hao Zhang. Language-driven synthesis of 3d scenes from scene databases. *ACM Transactions on Graphics (TOG)*, 37(6):1–16, 2018.

[18] Alexander Mehler, Giuseppe Abrami, Christian Spiekermann, and Matthias Jostock. Vannotator: A framework for generating multimodal hypertexts. In *Proceedings of the 29th on Hypertext and Social Media*, pages 150–154. 2018.

[19] Christian M Meyer, Margot Mieskes, Christian Stab, and Iryna Gurevych. Dkpro agreement: An open-source java library for measuring inter-rater agreement. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*, pages 105–109, 2014.

[20] James Pustejovsky, Kiyong Lee, Harry Bunt, and Laurent Romary. Iso-timeml: An international standard for semantic annotation. In *LREC*, volume 10, pages 394–397, 2010.

[21] James Pustejovsky, Jessica Moszkowicz, and Zachary Yocum. Spaceeval annotation guidelines. 2014.

[22] James Pustejovsky, Jessica L Moszkowicz, and Marc Verhagen. Iso-space: The annotation of spatial information in language. In *Proceedings of the Sixth Joint ISO-ACL SIGSEM Workshop on Interoperable Semantic Annotation*, volume 6, pages 1–9, 2011.

[23] James Pustejovsky, Jessica L Moszkowicz, and Marc Verhagen. Using iso-space for annotating spatial information. In *Proceedings of the International Conference on Spatial Information Theory*, 2011.

[24] Manolis Savva, Angel X Chang, and Pat Hanrahan. Semantically-enriched 3d models for common-sense knowledge. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 24–31, 2015.

[25] Manolis Savva, Angel X Chang, Pat Hanrahan, Matthew Fisher, and Matthias Nießner. Pigraphs: learning interaction snapshots from observations. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

[26] Lee M Seversky and Lijun Yin. Real-time automatic 3d scene generation from natural language voice and text descriptions. In *Proceedings of the 14th ACM international conference on Multimedia*, pages 61–64, 2006.

[27] Christian Spiekermann, Giuseppe Abrami, and Alexander Mehler. Vannotator: a gesture-driven annotation framework for linguistic and multimodal annotation. *Proc. of AREA*, 2018.

[28] Unity Asset Store. Uma-unity multipurpose avatar. `https://assetstore.unity.com/packages/3d/characters/uma-2-unity-multipurpose-avatar-35611`, 2017. [Online; accessed 10.11.2020].

[29] Carlo Strapparava, Alessandro Valitutti, et al. Wordnet affect: an affective extension of wordnet. In *Lrec*, volume 4, page 40. Citeseer, 2004.

[30] C Lawrence Zitnick, Devi Parikh, and Lucy Vanderwende. Learning the visual interpretation of sentences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1681–1688, 2013.