text2City: Räumliche Visualisierung textueller Strukturen

Bachelorarbeit

im Studiengang Informatik

vorgelegt von

Attila Kett

am 10.07.2019
an der Goethe-Universität Frankfurt am Main

Erstprüfer: Prof. Dr. Alexander Mehler
Zweitprüfer: Prof. Dr. Visvanathan Ramesh

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Ebenso bestätige ich, dass diese Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde.

Ort, Datum

Unterschrift

Danksagung

An dieser Stelle möchte ich mich ganz herzlich bei Herrn Prof. Dr. Mehler für die Betreuung und großartige Anregungen bedanken.

Ein besonderer Dank geht an Herrn Giuseppe Abrami für die hilfreichen Gespräche und Unterstützung bei Fragen rund um den *TextAnnotator* [1].

Ein riesiges Dankeschön geht an meinen besten Freund, Oliver Krenz für das Korrekturlesen und an alle, die an der Evaluation teilgenommen haben.

Kurzfassung

Gegenstand der hier vorgestellten Arbeit ist eine Applikation für die virtuelle Realität (VR), die in der Lage ist, die Struktur eines beliebigen Textes als begehbare, interaktive Stadt zu visualisieren. Darüber hinaus bietet das Programm eine besondere Textsuche an, die so in anderen konventionellen Textverarbeitungsprogrammen nicht vorzufinden ist. Dank der strukturellen Analyse und der Verwendung einiger außergewöhnlicher Analysetools des *TextImager* [11], ermöglicht *text2City* nicht nur die Suche nach bestimmten Textmustern, sondern zum Beispiel auch die Bestimmung der Textebene (Wort, Satz, Absatz, etc.) und einiges mehr. Ein weiteres Feature ist die Kommunikationsverbindung zwischen dem *TextAnnotator*-Service [1] und *text2City*, die dem Benutzer die Möglichkeit zum Annotieren bietet, aber auch von anderen Personen durchgeführte Annotationen sofort sichtbar machen kann. Für die Ausführung des Programms ist eine der beiden VRBrillen, Oculus Rift oder HTC Vive, ein für VR geeigneter PC, sowie die Software *Unity* nötig.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

VR	Virtual Reality
POS	Part-of-speech
DDC	Dewey decimal classification [29]
DKPro	Darmstadt Knowledge Processing Repository [6] [25]
HUD	Head-up-Display
UIMA	Unstructured Information Management Architecture [5] [10]
XMI	XML Metadata Interchange
IDE	Integrated Development Environment
UMUX	Usability Metric for User Experience

1 Einleitung

Texte gehören zu den wichtigsten Informationsträgern in unserem Leben. Auch wenn zum Beispiel Hörbücher immer beliebter werden, werden Texte hauptsächlich, egal ob analog oder digital, in schriftlicher Form erstellt und aufbewahrt. Abgesehen von der geschätzten Länge eines Textes ist es unmöglich, sich ein Bild von seinem Inhalt oder gar von seiner Struktur zu machen, ohne ihn zumindest zu überfliegen. Es gibt mittlerweile eine Vielzahl an Software, die unter anderem die Visualisierung von Texten übernimmt. Genau dieses Ziel verfolgt auch *text2City*. Den entscheidenden Unterschied zwischen dieser und anderen Visualisierungen macht die Darstellungsumgebung aus. Während die meisten anderen aus Übersichtlichkeitsgründen zweidimensional arbeiten, versucht *text2City* auch die dritte Dimension mit einzubeziehen. Den großen Vorteil bringt hierbei der Einsatz der virtuellen Realität, denn durch ihre Verwendung kann die Tiefen- und Größenwahrnehmung, sowie die Bewegungsfreiheit genutzt werden, statt den Benutzer mit der Verwendung von Bildschirm und Tastatur einzuschränken.

Das Ziel dieser *text2City* ist erstens eine informationsreiche 3D-Visualisierung von Dokumenten, so dass der strukturelle Aufbau des Textes im Gesehenen bestmöglich wiedererkennbar ist. Der Benutzer der Applikation wird zweitens in der Lage sein, eine erweiterte Suche im Text einfach durchführen zu können. Drittens, durch eine Verbindung zum *TextAnnotator*-Service ermöglicht *text2City* auch das Annotieren eines visualisierten Textes in der virtuellen Umgebung

2 Ähnliche Projekte

Eine ausführliche Recherche nach Anwendungen, die Textstrukturen in VR visualisieren, war wenig ergiebig. Dennoch gibt es einige Projekte, die gleiche oder ähnliche Teilprobleme wie *text2City*bearbeiten:

- *text2City* verwendet das *resources2City*-Projekt [17] als Basis und sieht dementsprechend diesem zum Teil sehr ähnlich. Genaueres dazu in Unterkapitel 5.2.3.
- *LitViz* [30] verwendet, wie auch *text2City*, den Voronoi-Algorithmus (siehe 5.2.3) um Texte dreidimensional zu visualisieren. *LitViz*arbeitet aber ohne eine interaktive virtuelle Umgebung.
- *SoftwareCity* [3] ist ein Programm für VR um interaktive Städte zu erschaffen. Anders als *text2City*stellt es Software strukturell dar.
- *The Information Visualizer* [4] verwendet drei Dimensionen zur Darstellung von Informationen durch Objekte. Er nutzt Form und Position von den Objekten, um den Benutzern einen visuellen Bezug zu den Informationen, eine Art Erinnerungsbild, zu ermöglichen. Auch hier sind parallelen zu *text2City*erkennbar.
- Der Artikel „Design and Evaluation of Data Annotation Workflows for Cave-like Virtual Environments“ [21] beschäftigt sich, ähnlich wie *text2City*, mit der Problemstellung, wie die virtuelle Realität verwendet werden kann um Daten-Annotationen durchzuführen.
- Die Artikel „A City Metaphor to Support Navigation in Complex Information Spaces“ [8], „The social logic of space“ [12], „Constructing Virtual Cities by Using Panoramic Images“ [13], „Building virtual Cities: Applying Urban Planning Principles to the Design of Virtual Environments“ [14] und „Virtual Reality Smart City Based on WebVRGIS“ [18] beschäftigen sich mit der räumlichen Visualisierung von Daten.

3 Verwendete Hardware und Software

Das Programm wurde mit den VR-Headsets Oculus Rift und HTC Vive unter der Verwendung von *Unity* [28] entwickelt und getestet. *Unity* verwendet C# als Programmiersprache und stellt gleichzeitig *Microsoft Visual Studio* [20] als Entwicklungsumgebung (IDE) zur Verfügung.

Die ursprüngliche *Unity*-Implementation des Voronoi-Algorithmus [23] wurde für den Stadtalgorithmus modifiziert.

Alle komplexeren 3D-Modelle, die nicht dynamisch generiert werden, werden mit *Blender* [2] erstellt.

Der *Text Structure Analyzer* wurde in Java unter der Verwendung von *IntelliJ IDEA* [15] geschrieben.

Für die Bildverarbeitung wurde *GIMP* [24] benutzt.

text2City wurde innerhalb des Projekts *StolperwegeVR* entwickelt, von welchem einige Features verwendet werden: Der virtuelle Avatar, dessen Steuerung, die virtuelle Smartwatch, die Tastatur und der einfache Dateibrowser.

4 Allgemeine Steuerung und Begriffserklärung

Da im Gegensatz zu beispielsweise Smartphones VR-Brillen (noch) kein alltäglicher Bestandteil unseres Lebens sind, haben viele, so auch Teilnehmer der Evaluation, noch nie die virtuelle Realität erleben können. Aus diesem Grund wird in diesem Kapitel einerseits beschreiben, wie VR funktioniert. Anderseits erläutert dieses Kapitel, welche Basis-Aktionen mit dem Avatar ausgeführt werden können. Dabei tauchen einige spezielle Begriffe auf, die in der späteren Beschreibung verwendet werden.

Anmerkung: Unterkapitel 4.1 bis einschließlich 4.7 beschreibt nicht speziell für diese Bachelorarbeit, sondern für das *StolperwegeVR*-Projekt entwickelte Features. Unterkapitel 4.8 behandelt Schnittstellen und Fachbegriffe, die für die *text2City* ebenfalls relevant sind, aber nicht zu *StolperwegeVR* gehören.

4.1 Der virtuelle Avatar

Die für das *StolperwegeVR*-Projekt verwendeten Brillen benötigen einen leistungsstarken PC. Der Datenaustausch zwischen Brille und PC erfolgt über einen HDMI-Anschluss. Die Brillen sind so konstruiert, dass der Benutzer von der Außenwelt soweit abgeschottet wird, dass sie die Erlebnisse in der virtuellen Realität nicht stört. Mit Hilfe von Trackingsensoren wird die Kopfbewegung genau aufgezeichnet und an den PC übermittelt. Mit diesen Daten wird das Bild berechnet und an die Brille übertragen. Dadurch entsteht das Gefühl, dass man sich in einer virtuellen Welt befindet. Für Interaktionen existieren spezielle Controller, die in den Händen gehalten werden und ihre Bewegung an den PC übermitteln. Der virtuelle Avatar besteht aus zwei Hauptkomponenten: Aus einer linken und rechten Hand und aus dem Kopf mit einem Head-up-Display (HUD) für Informationsanzeige (siehe Abbildung 3 und Abbildung 4).

4.2 Die Steuerungstasten

Die Controller besitzen verschiedene Bedienelemente mit denen man sich in der virtuellen Welt bewegen und Aktionen auslösen kann. Oculus Rift hat speziell für die linke und für die rechte Hand geformte Controller (Abbildung 1); HTC Vive stellt zwei identische (Abbildung 2) Controller zur Verfügung. Da die Bedientasten gleiche Funktionen haben, aber diese teilweise durch unterschiedliche Tastentypen ausgelöst werden, wurde im *StolperwegeVR*-Projekt für alle verwendeten Tastentypen eine allgemeine Bezeichnung eingeführt:

- **Stick**: Ein in 2 Dimensionen frei bewegbares Thumbstick (bei Oculus Rift) oder ein Touchpad (bei HTC Vive). Diese sind auch als Taste benutzbar. Haptisch ist der Tastendruck mit einem Mausklick vergleichbar. Thumbstick und Touchpad werden mit dem Daumen bedient
- **Trigger**: Eine Drucktaste vorne (Oculus Rift), beziehungsweise unten am Controller (HTC Vive), die mit dem Zeigefinger aktiviert wird
- **Grab**: Eine Drucktaste (Oculus Rift), beziehungsweise Druckknopf (HTC Vive), bei beiden Herstellern seitlich am Controller platziert und für eine Benutzung mit dem Mittelfinger vorgesehen

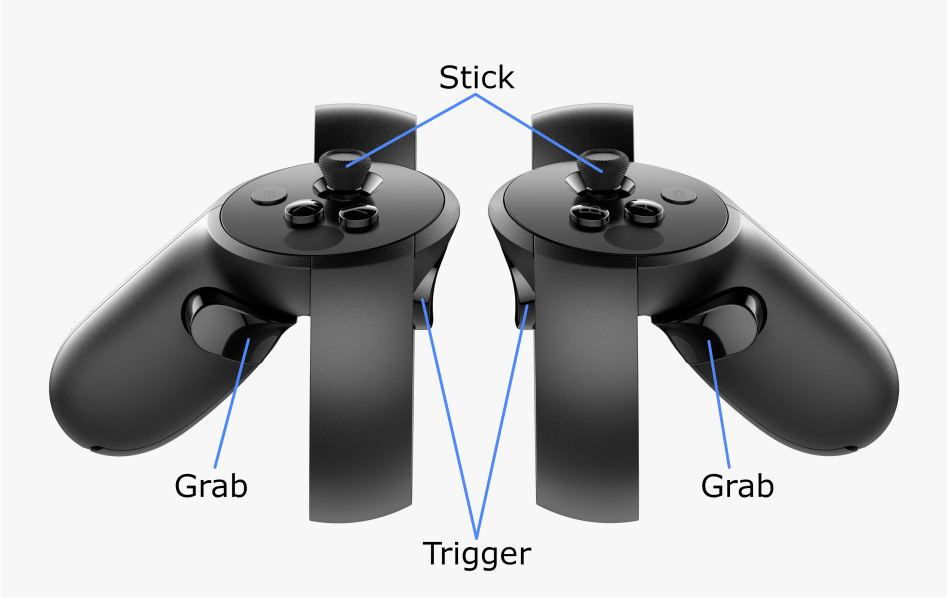

Abbildung 1: Die Oculus Rift-Controller und ihre in *text2City* verwendeten Bedientasten [16]

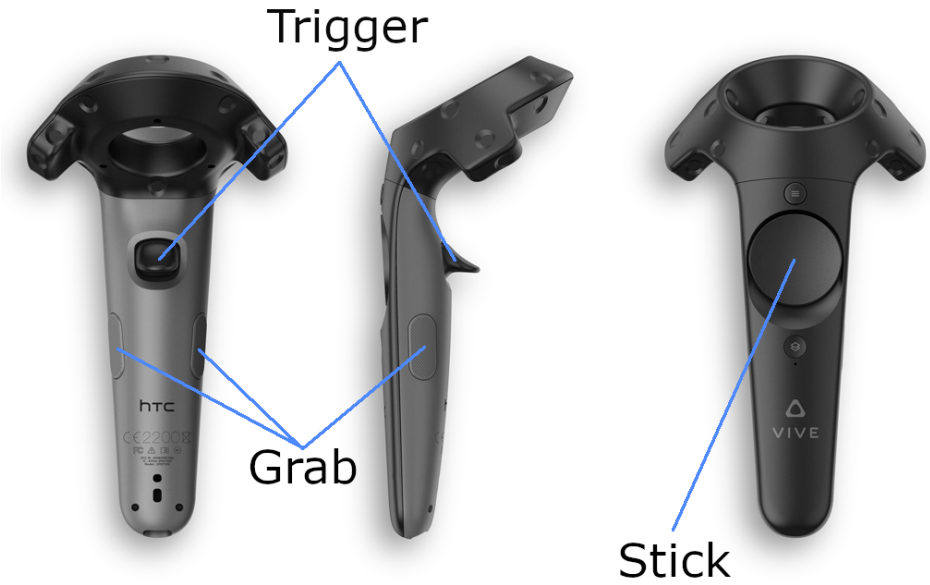

Abbildung 2: Die HTC Vive-Controller und ihre in *text2City* verwendeten Bedientasten [31]

4.3 Gesten mit den Händen

Die virtuellen Hände verfügen über zwei Gesten: Zeigen und Greifen (siehe Abbildung 3). Beim Greifen wird eine Faust gemacht, wobei ein sogenannter Collider um die Hand herum aktiviert. Trifft ein interaktives Objekt den Collider, so wird es in die virtuelle Hand genommen. Beim Zeigen wird der Zeigefinger ausgestreckt und der Druckpunkt in der Fingerspitze aktiviert. Befindet sich kein interaktives Objekt in der Nähe des Fingers, wird zusätzlich die Zeigerichtung mit einem Strahl visualisiert, vergleichbar mit einem Laserpointer.

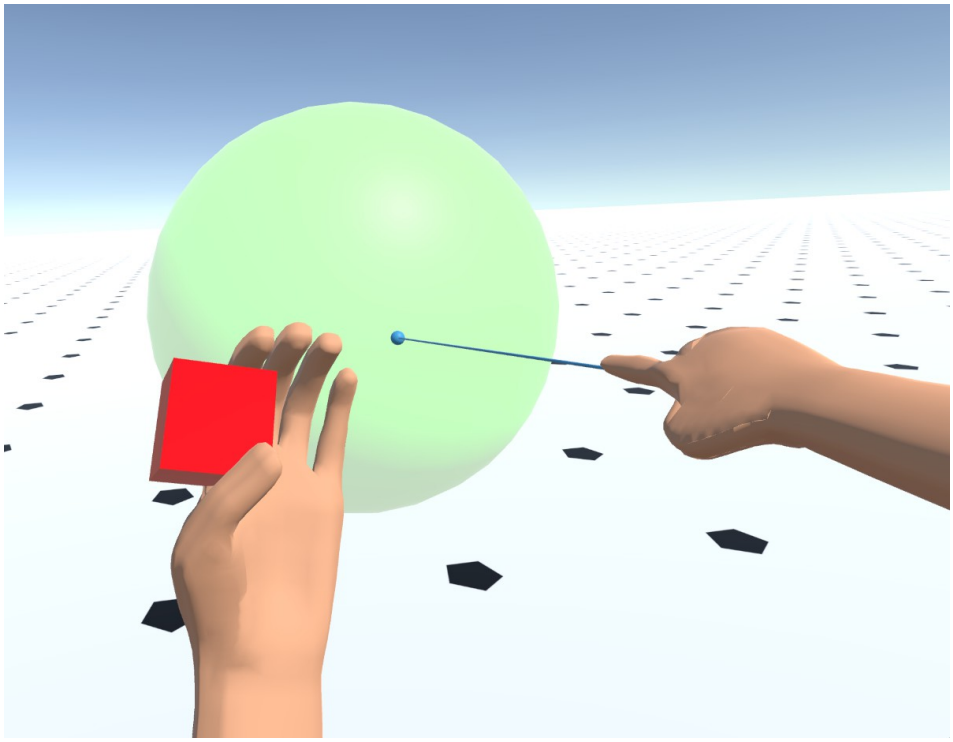

Abbildung 3: Zeigen und Greifen in *StolperwegeVR*

4.4 Interaktionen in *StolperwegeVR*

Nicht jedes Objekt in der virtuellen Welt von *StolperwegeVR* ist interaktiv. Welche Objekte interaktiv sind, ist leicht erkennen. Sobald sie anvisiert werden, oder auf sie gezeigt wird, werden sie „highlighted“ (wie die Kugel in Abbildung 3).

Ein interaktives Objekt besitzt folgende elementare Interaktionsmöglichkeiten: Es kann in die Hand genommen, angeklickt (Aktion Click) oder lang angeklickt (Aktion LongClick, siehe Abbildung 4) werden (siehe Tabelle 1). In-die-Hand-nehmen funktioniert wie folgt: Man streckt die Hand nach dem Objekt aus, betätigt die dafür vorgesehenen Tasten und umfasst das Objekt dadurch mit den Fingern. Um die Aktion Click eines Objekts auszulösen, muss es anvisiert und die Taste für Click kurz gedrückt werden. LongClick funktioniert nach dem gleichen Prinzip wie Click, mit dem Unterschied, dass man die Click-Taste so lange gedrückt halten muss, bis der erscheinende Ladebalken vollständig aufgefüllt wurde. Bei einem Abbruch wird nicht sofort der komplette Ladestatus zurückgesetzt, sondern der Ladebalken in der gleichen Geschwindigkeit wieder entladen. Click und Long-Click können bei den meisten Objekten auch durch Berührung mit der Fingerspitze (Antippen) ausgelöst werden, wenn das Objekt in greifbarer Nähe liegt.

Aktion	**Controllerbelegung**
Bewegung	Stick bewegen (links)
Zeigen	Grab (links)
Greifen	Grab + Trigger
Click / Long-Click	Stick drücken / gedrückt halten (rechts)
Schreibkabine aktivieren	Trigger gedrückt halten (links)

Tabelle 1: Belegung der Controller für die Avatar-Steuerung

4.5 Die virtuelle Smartwatch

StolperwegeVR besitzt verschiedene Features, wie zum Beispiel *VAnnotatoR* [19], den *resources2City* Explorer [17] und andere. Um diese Features schnell und einfach benutzen zu können, wurde der virtuelle Avatar mit einer Smartwatch ausgestattet (siehe Abbildung 5). Die Smartwatch besitzt drei verschiedene Modi:

- Ruhezustand (symbolisiert durch einen Kreis am Display)
- Menüaktivierungsmodus (symbolisiert durch Kacheln am Display)
- Dateibrowseraktivierungszustand (symbolisiert durch einen Ordner am Display)

Wird die Uhr anvisiert, oder darauf gezeigt, versetzt sie sich in den Menüaktivierungszustand. Wenn sie in diesem Zustand angeklickt wird, öffnet sich das Hauptmenü, von wo aus sich die meisten Features starten lassen. Wenn der Benutzer die andere Hand zur Smartwatch führt, wird der Dateibrowseraktivierungszustand aktiv. Greift man nach dem Ordner-Symbol, erscheint in der greifenden Hand die Miniatur eines einfachen Dateibrowsers, die sich beim Loslassen über den ganzen Bildschirm entfaltet. Der einfache Dateibrowser kann dafür benutzt werden *text2City* zu starten (siehe weiter in 4.6).

Abbildung 4: Der Fortschritt der Long-Click Funktion wird mit einem weißen Ladebalken im HUD angezeigt

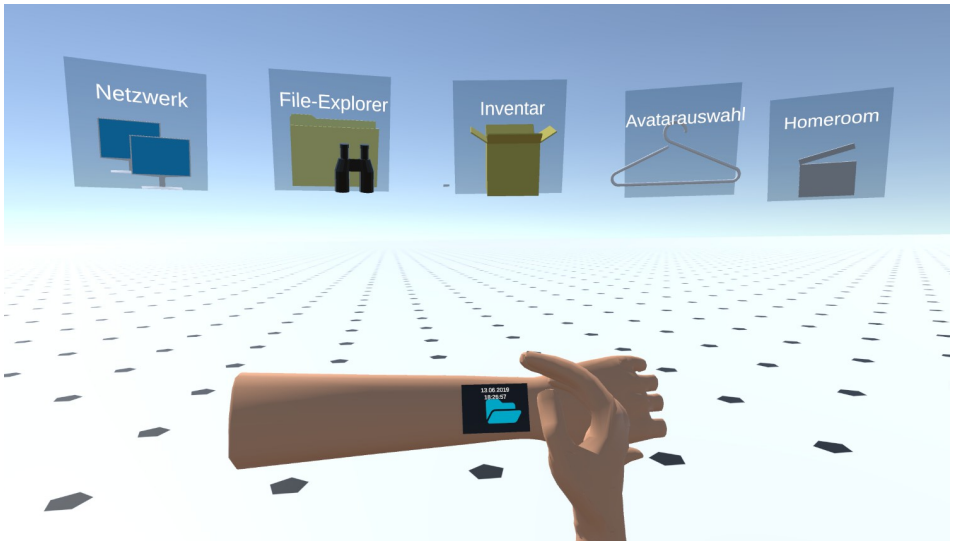

Abbildung 5: Die virtuelle Smartwatch mit geöffnetem Menü

4.6 Der einfache Dateibrowser

Der Dateibrowser besteht aus verschiedenen Komponenten. Im oberen Fenster hat man die Möglichkeit auszuwählen, ob man den lokalen Datenträger oder den ResourceManager durchsuchen möchte. Das Hauptpanel besitzt sogenannte Datencontainer, die bei Übergabe einer Datei ihre Informationen, wie Name, Typ und Format anzeigen. Wenn der Datencontainer einen Ordner beinhaltet und angeklickt wird, so wechselt das Panel in diesen Ordner und zeigt dessen Daten an. Bei einem Long-Click gibt das Panel die Datei als ein dreidimensionales, greifbares Objekt heraus.

Das linke Fenster ist der sogenannte Visualizer. Seine Aufgabe ist es anzuzeigen, ob und wie eine Datei visualisiert werden kann. Ob *resources2City* oder *text2City* aufgerufen werden, hängt davon ab, ob die Datei ein Ordner oder eine Textdatei ist. Nimmt der Benutzer eine Datei, die als virtuelle Stadt geladen werden kann, in die Hand, blinkt der Visualizer um den Benutzer darauf aufmerksam zu machen. Nachdem die Datei im Visualizer platziert wurde, kann der Stadtgenerierungsalgorithmus gestartet werden.

Abbildung 6: Der einfache Dateibrowser mit einer Textdatei im Visualizer

4.7 Das Schreiben in *StolperwegeVR*

Die Schreibkabine kann jederzeit manuell durch gedrückt halten des linken Triggers oder durch das Anklicken eines Eingabefeldes (interaktives Objekt) geöffnet werden. Sobald der Benutzer nach dem Schreiben den „Fertig“-Button tätigt, wird das Geschriebene als ein interaktiver Text-Würfel ausgegeben (nur beim manuellen Aufruf) oder in das entsprechende Eingabefeld automatisch eingetragen.

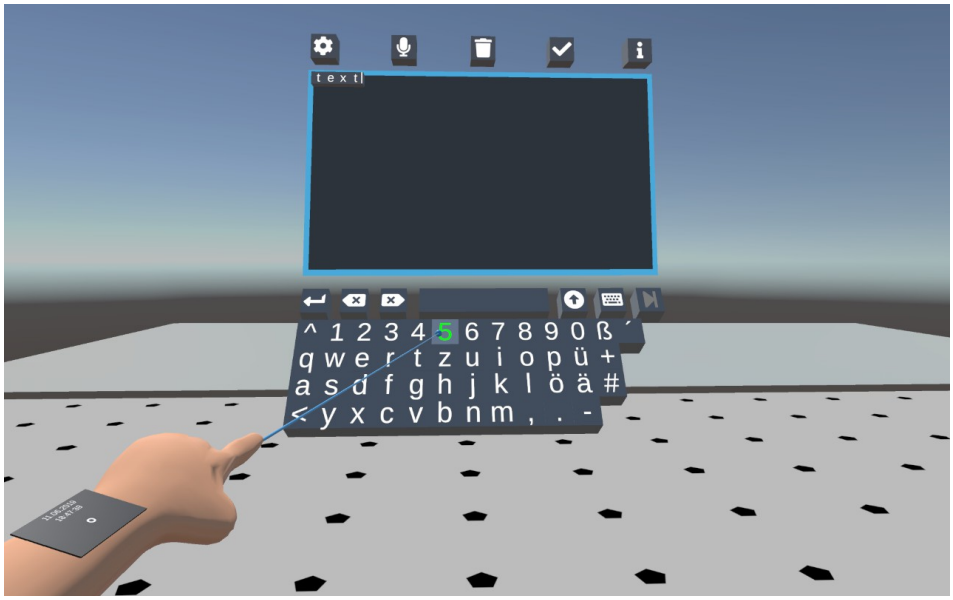

Abbildung 7: Das Schreiben in *StolperwegeVR*

4.8 *TextAnnotator*, *TextImager* und *Text Structure Analyzer*

Dieses Kapitel behandelt die Schnittstellen, welche die Hintergrundarbeit für *text2City* erledigen. Die Gemeinsamkeit dieser drei Plattformen ist, dass sie alle UIMA-basiert (Unstructured Information Management Architecture [5]) sind. UIMA ist ein weitverbreitetes Framework zur Extrahierung vom Wissen aus bestimmten Daten [10] und wird im Bereich Texttechnologie mittlerweile als Norm angesehen. Dabei wird der Text mit einer Liste von Tools, der sogenannten Pipeline, analysiert. Das Ergebnis wird in XMI-Format (XML Metadata Interchange) gespeichert und weitergegeben. Das Format XMI wird heutzutage häufig benutzt um zwischen Software-Entwicklungstools Informationen über Objekte auszutauschen.

Der *TextImager* [11] und *Text Structure Analyzer* sind genau solche UIMA-basierte Plattformen, wobei beim zuerst genannten der Benutzer selbst festlegen kann, mit welchem Tool der Text analysiert werden soll. Sollte die benutzerdefinierte Pipeline unvollständig oder in einer falschen Reihenfolge sein, wird sie vom *TextImager* automatisch ergänzt oder korrigiert. Der *Text Structure Analyzer* hat eine eigens für *text2City* festgelegte Pipeline, die sich zum größten Teil aus Tools vom *TextImager* und dem eigens für *text2City* entwickelten ChapterSplitter zusammensetzt. Der ChapterSplitter befindet sich noch in der Entwicklungsphase, weshalb der *Text Structure Analyzer*bis auf Weiteres als alleinstehendes Programm arbeitet.

Natürlich kann es vorkommen, dass die Analyse mit besagten Tools unrichtig oder unvollständig ist. Genau an dieser Stelle kommt der *TextAnnotator* zum Einsatz. Mit ihm sind Benutzer in der Lage, den Text manuell zu annotieren. Die genaue Funktionsweise des *TextAnnotator* wird noch in 5.3.4 erläutert. Ein Dokument kann von mehreren Benutzern gleichzeitig bearbeitet werden. Verändert ein Benutzer erfolgreich ein Dokument, so benachrichtigt der *TextAnnotator* alle Clients (Endgeräte), die das gleiche Dokument geöffnet haben darüber. *text2City* verwendet aktuell das QuickAnnotator-Tool des *TextAnnotators*. Tabelle 2 listet die Befehle des TextAnnotators, die *text2City* verwendet, mit ihren Funktionen auf.

Befehl	**Funktion**	**Parameter**
session	Verbindung herstellen	SessionID
create_cas_db	Datenbankeintrag für neues Dokument anlegen	XMI-Datei, Dateiname
open_cas	Öffnen eines existierenden Eintrags	ID des zu öffnenden Dokuments
open_tool	Wählt das Annotationswerkzeug aus	Toolname
work_batch	Änderung am Dokument, Client to Server	Dokument-ID, Veränderungsdaten
change_cas	Änderung am Dokument, Server to Clients	Dokument-ID, Veränderungsdaten

Tabelle 2: Die in *text2City* verwendeten *TextAnnotator*-Befehle

5 Eingeschlagener Realisierungsweg

Dieses Kapitel befasst sich mit der Implementierung des Programms. Sowohl die Funktionalität im Backend-Bereich als auch die visuelle, interaktive Seite der Benutzeroberfläche wird hier genauer erklärt. Tabelle 3 zeigt die drei Schnittstellen, auf denen das Programm läuft.

Schnittstelle	**Funktion**	**Komminiziert mit**
text2City	Benutzeroberfläche	*TextAnnotator*, *Text Structure Analyzer*
TextAnnotator	Annotieren des Textes	*text2City*
Text Structure Analyzer	Textanalyse	*text2City*

Tabelle 3: Die von *text2City* verwendeten Schnittstellen

Der Benutzer interagiert ausschließlich mit der virtuellen Welt in *Unity*, das für die visuelle Darstellung zuständig ist. Der Informationsaustausch zwischen den Schnittstellen erfolgt automatisch im Hintergrund. Zwischen dem Zeitpunkt, an dem der Benutzer eine Textdatei ausgewählt hat und dem an dem die Stadt generiert wird, findet folgender Prozess statt:

Zuerst muss *text2City* eine Verbindung zum *TextAnnotator* herstellen (siehe Tabelle 2, session). Für alle nichtöffentlichen Dokumente benötigt man Zugangsdaten und wird zum Login aufgefordert. Als nächstes wird festgestellt, ob für die ausgewählte Textdatei bereits ein Eintrag in Form einer Identifikationsnummer im lokalen Cache existiert. Diese Abfrage kann zwei Ausgänge haben:

- Wenn ja, schickt *text2City* einen Befehl zum Öffnen des Dokuments mit der herausgefundenen ID an den *TextAnnotator* (siehe Tabelle 2, open_cas).
- Wenn nein, stellt *text2City* eine Verbindung zum *Text Structure Analyzer* her und übersendet ihm den Inhalt der ausgewählten Datei zur Analyse.

Der *Text Structure Analyzer* wartet auf Daten und versucht diese dann zu analysieren. Wenn die empfangene Datei das XMI-Format besitzt, wurde sie möglicherweise bereits mit einigen, oder eventuell sogar allen, der benötigten Tools analysiert. Falls einige Tools fehlen, werden nur die fehlenden zur Analyse eingesetzt. Ansonsten wird die komplette Pipeline (siehe Tabelle 4) ausgeführt. Die Ergebnisse der Analyse werden an *text2City* als XMI zurückgeschickt. Nun kann *text2City* die analysierten Daten an den *TextAnnotator* weitergeben. Der *TextAnnotator* empfängt die Daten, legt ein neues Dokument in der Datenbank an (siehe Tabelle 2, create_db_cas), öffnet es und schickt das Ergebnis an *text2City* zurück (siehe Tabelle 2, open_cas). Nach dem beschriebenen Prozess beginnt die Stadtgenerierung. Der ganze Prozess wird in Abbildung 8 dargestellt. Unterkapitel 5.1 behandelt den Textvorverarbeitungsprozess, 5.2 die virtuelle Stadt und 5.3 den TextRaum, das begehbare innere eines Gebäudes.

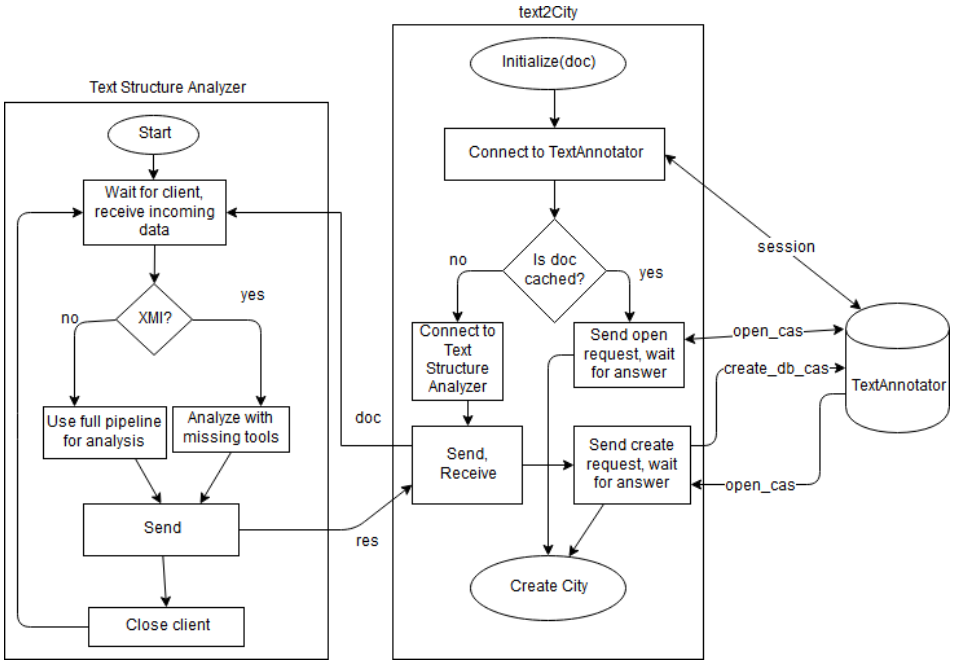

Abbildung 8: Der Datenaustausch der Schnittstellen vor der Stadtgenerierung

5.1 Die Textvorverarbeitung

Bevor ein Text zu einer virtuellen Stadt umgewandelt werden kann, muss er analysiert werden. Tabelle 4 zeigt die für die virtuelle Stadt benötigten Informationstypen und die verwendeten Tools, mit denen diese ausgelesen werden können.

Analyse-Tool	Zugehörigkeit	Ausgelesesene Information	C#-Klasse
ChapterSplitter	Eigene Entwicklung	Kapitel	Chapter
ParagraphSplitter	DKPro-Core [6] [25]	Absatz	Paragraph
LanguageToolSegmenter	DKPro-Core	Satz	Sentence
LanguageToolSegmenter	DKPro-Core	Zeichen(kette)	Token
MateLemmatizer	DKPro-Core	Stichwort	Lemma
StanfordPosTagger	DKPro-Core	Wortart	PartOfSpeech
DDCTool [29]	TextImager	DDC-Kategorie	DdcCategory
Sentiws [22]	TextImager	Stimmung	Sentiment

Tabelle 4: Die verwendeten Textanalyse-Tools

Der ChapterSplitter ist ein für *text2City* entwickeltes, rudimentäres Tool, das mit Hilfe von Textmustern, sogenannten regulären Ausdrücken, Kapitel eines Textes erkennt. Voraussetzung für die Ausführung dieses Tool ist eine bereits durchgeführte Absatzerkennung, da der ChapterSplitter die Absätze zur Erkennung von Kapiteln einsetzt. Als reguläre Ausdrücke, werden die Wörter „Kapitel“ und „Prolog“ für deutsche und „Chapter“ und „Prologue“ für englische Texte eingesetzt. Weitere Sprachen werden aktuell nicht unterstützt. Abbildung 9 zeigt die Funktionsweise des ChapterSplitter.

Kapitel, Absätze, Sätze und Tokens sind die strukturellen Textelemente, daher kann die Stadt ohne auch nur einen dieser Elemente nicht erstellt werden. Der StanfordPosTagger (siehe Tabelle 4) wird verwendet, um Wortarten (Part-of-speech, kurz POS) aus dem Text zu extrahieren. Damit der StanfordPosTagger eingesetzt werden kann, muss der Text zuerst mit dem MateLemmatizer analysiert werden. POS spielen in der virtuellen Stadt ebenfalls eine wichtige Rolle, dürfen von daher auf keinen Fall bei der Erstellung dieser fehlen.

Zur Veranschaulichung der Lemmatisierung sollte der folgende Beispielsatz betrachtet werden: *Peter geht morgen angeln.* Alle Zeichenfolgen in diesem Satz werden nach der Analyse sowohl als Token, als auch als Part-of-speech abgespeichert. Für die Zeichenfolge geht wird ein Token und eine Part-of-speech Instanz (Typ Verb) angelegt. Beide haben als Eigenschaft „Begin“ = 6, „End“ = 10 (der Inhaltstext beginnt mit dem 6. und hört mit 10. Zeichen des Textes auf. Zu beachten: Die Nummerierung der Charakter beginnt mit 0).

```
1  Initialize variables begin, end with 0;
2  Initialize the Set R of Regular Expressions;
3  Select all Paragraphs from the Document and copy them into List P;
4  foreach (Paragraph p in P)
5  {
6    if (p has more than 100 characters) skip p;
7    if (text of p contains one of the Expressions in R (ignore letter case))
8    {
9       end = begin of p;
10      if (begin not equals end)
11      {
12        Create new chapter with begin & end;
13        begin = end;
14      }
15   }
16 }
17 if (begin  Length of the Document)
18 {
19   Create new chapter with begin & length of document as end;
20 }
```

Abbildung 9: Die Funktionsweise des ChapterSplitter in Pseudocode

Die folgenden Textelemente werden in der Stadt nur dekorativ eingesetzt, haben keinen Einfluss auf die Stadtstruktur und sind auch nicht zwingend für die Erstellung der Stadt notwendig. Das DDCTool (Dewey Decimal Classification [7]; siehe Tabelle 4) erstellt nach Möglichkeit für die Textelemente Dokument, Kapitel, Absatz und Satz sogenannte DDC-Kategorien. Die DDC besteht aus 10 Hauptkategorien, denen weitere Unterkategorien zugeteilt sind. Jede Unterkategorie stellt ein Spezialgebiet der zugeordneten Hauptkategorie dar. Jede Kategorie hat eine dreistellige Zahl als Kennzeichen. Die Identifikationsnummer der Hauptkategorien endet immer auf zwei Nullen. Unterkategorien übernehmen die erste Stelle der ID der Hauptkategorie, der sie untergeordnet sind. Die zweite und dritte Stelle der Zahl definiert die eigene ID. In der Visualisierung verwendet *text2City* nur die Hauptkategorien und ordnet ihnen folgende Farben zu:

Kennzeichen	Farbe	Gebiet
000		Informatik, Informationswissenschaft, allgemeine Werke
100		Philosophie und Psychologie
200		Religion
300		Sozialwissenschaften
400		Sprache
500		Naturwissenschaften und Mathematik
600		Technik, Medizin, angewandte Wissenschaften
700		Künste und Unterhaltung
800		Literatur
900		Geschichte und Geografie

Tabelle 5: Die Hauptkategorien des Dewey Decimal Classification

Das Sentiws-Tool (siehe Tabelle 4) wurde entwickelt, um die Stimmung (negativ, neutral oder positiv) eines Textinhaltes bestimmen zu können. Es greift auf eine vordefinierte Datenbank von Begriffen zu, in der Wörter Stimmungswerten als Zahlen zugeordnet sind. Das Sentiws-Tool kann erst gestartet werden, nachdem die Tokens des Textes bekannt sind, denn diese werden durch das Tool mit einer reellen Zahl assoziiert. Die Stimmung eines Tokens ist positiv, wenn die reelle Zahl größer als Null und negativ, wenn sie kleiner als Null ist. Ein Null-Wert steht dementsprechend für einen neutralen Inhalt. Sentiws funktioniert ausschließlich bei deutschsprachigen Texten.

Nach dem ein neues XMI-File an den *TextAnnotator* gesendet wurde, ergänzt dieser das Dokument zusätzlich mit sogenannten Multi-Tokens. Sie dienen als Mittel zu Annotationen, damit die Informationen über dem ursprünglichen Token erhalten bleiben. MultiTokens können aus mehreren oder auch nur aus einem Token bestehen. Jeder Token muss mindestens mit einem anderen Token im Multi-Token benachbart sein (nebeneinanderliegenden Wörter). Eigennamen im Text werden durch Named Entities dargestellt. Letztere speichern zusätzliche Informationen über dem Textinhalt. Named Entities werden durch das QuickAnnotator-Tool des *TextAnnotators* beim Annotieren dem Dokument hinzugefügt. Folgendes Beispiel zeigt das Prinzip von Named Entities:

***Beispiel**: „Max Mustermann angelt.“. Dem Leser des Satzes wird schnell klar, dass Max Mustermann eine Person ist. Named Entities sind konzipiert worden, um solche Informationen abzuspeichern. Für die Erstellung eines Named Entity müssen zuerst die Tokens der Text festgestellt werden. Das geschieht mit dem LanguageToolSegmenter-Tool (siehe Tabelle 4). Die Tokenisierung des Beispielsatzes würde die folgenden Tokens liefern: „Max“, „Mustermann“, „angelt“ und „.“. Nun kann eine Named Entity-Instanz mit Startindex 0 (der erste Buchstabe des Satzes), dem Abschlussindex 13 (der letzte Buchstabe des Tokens „Mustermann“) und dem Typ Person erstellt werden.*

5.2 *text2City* - Die virtuelle Stadt

5.2.1 Die Textelemente-Klassen in *Unity*

Bevor die Implementierung der Textelementen beschrieben werden kann, wird zuerst der Aufbau eines Textes betrachtet. Tabelle 6 listet alle strukturellen Elemente eines Textes auf, die für *text2City* wichtig sind. Die hierarchische Reihenfolge wird durch die Rangnummern verdeutlicht:

Rang	Textelement	Eltern	Kinder
5	Dokument	-	Die zugehörigen Kapitel
4	Kapitel	Dokument	Die zugehörigen Absätze
3	Absatz	Das übergeordnete Kapitel	Die zugehörigen Sätze
2	Satz	Der übergeordnete Absatz	Die zugehörigen Tokens
1	Token (Zeichenkette)	Der übergeordnete Satz	-

Tabelle 6: Die Hierarchie der Textstruktur

Der hierarchische Aufbau des Textes wird auf die erschaffene virtuelle Umgebung übertragen. Anders gesagt: Ein Textelement, das Teil eines anderen größeren Elements ist, wird auch in der Stadt als ein Objektteil visualisiert. Das soll zum einen die Orientierung erleichtern und zum anderen dabei helfen, den Bezug zwischen dem Visuellen und dem Text herzustellen. Damit setzt sich ausführlicher Unterkapitel 5.2.2 auseinander.

In Unterkapitel 5.1 wurden bereits alle Textelemente erklärt, die, neben den strukturellen Elementen aus Tabelle 6, aus dem zu visualisierenden Text ausgelesen oder durch Textannotationen hinzugefügt werden. Diese Elemente haben zwar in den Hintergrundprozessen von *text2City* eine Funktion, aber nur manche werden visuell dargestellt (siehe Tabelle 7). Die meisten visualisierten Textelemente haben eine eigene 3D-Repräsentation, aber es gibt auch einige, die anderweitig dargestellt werden. Diese sind DDC-Kategorien, Sentiments und Named Entities.

Textelement	Strukturell	Visuelle Darstellung
Dokument	Ja	Die Stadt selbst
Kapitel	Ja	Stadtbezirk mit Bezirksverwaltungsgebäude
Absatzgruppe	Nein	In der Stadt: Etage Im Text-Raum: der Text-Raum selbst / Voronoi-Diagramm (bei 2 oder mehreren Sätzen)
Absatz	Ja	In der Stadt: Etage Im Text-Raum: der Text-Raum selbst / Voronoi-Diagramm (bei 2 oder mehreren Sätzen)
Satz	Ja	Textaufschrift an der Wand des Text-Raumes / VoronoiDiagramm-Balken (bei 2 oder mehreren Sätzen)
Token	Ja	Keine
Lemma	Nein	Keine
POS	Nein	Keine
Multi-Token	Nein	Ein greifbares Plättchen mit dem Textaufschrift drauf
DDC-Kategorie	Nein	Einfärbung der Gebäude-, Etagenfassade mit der zugewiesenen Farbe der Hauptkategorie
Sentiment	Nein	Wandfarbe des Text-Raumes
Named Entity	Nein	Einfärbung von Multi-Token-Plättchen

Tabelle 7: Die Textelemente und ihre visuelle Darstellung

Die folgende Beschreibung widmet sich der Implementierung der einzelnen Textelemente, wobei hier nur die wichtigsten Eigenschaften und Funktionen hervorgehoben werden. Abbildung 32 im Anhang zeigt ein vollständiges Klassendiagramm mit allen Eigenschaften und Methoden der Textelemente-Klassen. Die Klasse VRTextData implementiert das Textelement im Allgemeinen und ist die Basisklasse aller anderen Textelementen. VRTextData kann eine Liste von Kind-Textelement-Instanzen besitzen. Beispielsweise hat ein Kapitel alle in ihm befindlichen Absätze, ein Absatz alle in ihm befindlichen Sätze als Kinder und so weiter (siehe Tabelle 6).

Ein Textelement hat eine Reihe von essenziellen Eigenschaften. Eine Instanziierung ist deshalb nur durch die Übergabe bestimmter Parameter (siehe Tabelle 8) möglich. Abgesehen von diesen Initialisierungsparametern besitzt VRTextData einige andere Attribute, die entweder später eingetragen oder aus den vorhandenen Informationen berechnet werden können. Beispielsweise hat man in jedem Textelement die Möglichkeit abzufragen, wie viele andere untergeordnete Textelemente das Element beinhaltet (Absätze, Sätze, Token etc.). Allerdings sind diese Informationen erst verfügbar, sobald die untergeordneten Elemente ebenfalls erstellt und bei dem Element als Kinder eingetragen worden sind.

Parametername	Datentyp	Beschriebung
Parent	VRTextData	Das übergeordnete Textelement
ID	Ganze Zahl	Die Identifikationsnummer des Elements
Begin	Ganze Zahl	Der Anfangsindex des Elements im Text
End	Ganze Zahl	Der Schlussindex des Elements im Text
ClassType	Zeichenkette	Der Typ des Elements

Tabelle 8: Die essenziellen Attribute jedes Textelements

Die Textelemente-Klassen TextDocument, Chapter, Paragraph, Sentence, TextToken, QuickTreeUnit, NamedEntity, DdcCategory, Sentiment und PartOfSpeech sind die Implementierung der Textelemente im *Unity*-Backend-Bereich und werden aus den Daten erstellt, die der *TextAnnotator* über dem Websocket an *Unity* sendet.

Textdokument

Das größte Textelement, das Dokument selbst, wird als virtuelle Stadt dargestellt und durch die C#-Klasse TextDocument implementiert. Ein Textdokument besitzt als Kinder eine Liste von Kapiteln. Um einkommende Informationen über Annotationen schnell verarbeiten zu können, werden alle untergeordneten Textelemente in verschiedene Wörterbücher einsortiert (siehe Tabelle 9). Die TextDocument-Klasse besitzt außerdem unterschiedliche Abfrage-Funktionen, die durch Zugriff auf diese Wörterbücher gesuchte Textelemente zurückgeben können (siehe im Detail im Klassendiagramm in Abbildung 32).

Name	Schlüssel	Wert
ID_Map	ID des Textelements	Das dazugehörige Textelement
Type_Map	Typ des Textelements	Liste aller Textelemente des Typs
Begin_Map	Vorkommende Anfangsindexe	Alle Textelemente, wo Anfangsindex = Schlüssel
End_Map	Vorkommende Abschlussindexe	Alle Textelemente, wo Abschlussindex = Schlüssel

Tabelle 9: Die verschiedene Einsortierungsarten, um Textelemente im Dokument einfach zu finden

Kapitel

Kapitel, im Code als Chapter bezeichnet, stehen in der Textelemente-Hierarchie direkt unter dem Textdokument (siehe Tabelle 6). Sie sind Absätzen übergeordnet und enthalten diese wiederrum als Kinder. Eine essenzielle Methode in der Chapter-Klasse ist es, aus den Absätzen, die sich im Kapitel befinden, Gruppen zu bilden. Jede Gruppe enthält maximal 9 Absätze, wobei die Originalreihenfolge nicht verändert wird. Der Zweck der Gruppenbildung ist die Aufteilung der Absätze in mehrere Gebäude, was in Kapitel 5.2.2 ausführlicher beschrieben wird.

Absätze und Sätze

Die Paragraph-Klasse enthält die Liste der in den Absätzen vorkommenden Sätze. Die Sentence-Klasse implementiert Sätze aus dem Text; Ihre Kind-Elementliste besteht aus Tokens. Auch zu den Sätzen zugehörige Multi-Tokens werden im jeweiligen Satz abgespeichert. Für die Eintragung und Entfernung von Multi-Tokens besitzt die SentenceKlasse extra Funktionen, da nur die Tokens als Kinder erfasst werden.

Tokens

Tokens sind die kleinsten strukturellen Elemente im zu analysierenden Text und werden durch die TextToken-Klasse repräsentiert. Falls neben dem Token auch ein Part-of-Speech oder Sentiment mit dem gleichen Textinhalt existiert, wird die Verbindung zwischen ihnen mit den dafür vorgesehenen Methoden der TextToken-Klasse hergestellt. Das Aufrufen dieser Funktionen erfolgt aus der jeweiligen Klassen-Instanz. Darüber hinaus werden in der Token-Klasse auch alle Multi-Tokens, die den Textinhalt des Tokens beinhalten, in einer separaten Liste aufbewahrt.

Multi-Tokens

Multi-Tokens werden durch die QuickTreeUnit-Klasse implementiert und können auf eine oder mehrere TextToken-Instanzen verweisen. Genau wie Tokens die zugehörigen Instanzen der Multi-Token-Klasse verwalten, verwalten auch diese die zugehörigen TextToken-Klasseninstanzen. Multi-Tokens können auch mit einem oder mehreren NamedEntities verlinkt sein. Diese werden in einem Wörterbuch, nach Typ sortiert, abgespeichert.

Named Entities

Bei der Instanziierung der NamedEntity-Klasse wird eine Abfrage in der TextDocumentInstanz gestartet, um den Multi-Token zu finden, der den gleichen Start- und Abschlussindex besitzt. Die neu erstellte Named-Entity-Instanz speichert das Ergebnis der Abfrage und wird gleichzeitig im NamedEntities-Wörterbuch des zugehörigen Multi-Tokens eingefügt.

DDC-Kategorien

Dewey-Dezimalklassifikationen werden durch die Klasse DdcCategory umgesetzt. Bei der Instanziierung wird das Kennzeichen der Kategorie extrahiert und abgespeichert. Durch das Kennzeichen kann die Hauptkategorie bestimmt werden, dazu muss lediglich die zweite und dritte Stelle der 3-stelligen Kategorie-Identifikationsnummer durch Nullen ersetzt werden.

Sentiments

Die „Connect“-Methode der Sentiment-Klasse sorgt für die Verlinkung zwischen dem zugehörigen TextToken-Klasse und dem Sentiment (siehe Tokens).

Part-of-speech (POS)

Tokens werden immer vor den POS ausgelesen, weshalb die Verlinkung zwischen einem Token und einem POS bei der Erstellung der POS-Instanz mit ihrer „Connect“-Methode erfolgt (siehe Tokens). Da die Anzahl der Wörter pro Wortart eine wichtige Rolle bei der Erstellung der Stadt spielt (siehe Unterkapitel 5.2.3), existiert eine Methode in der VRTextData-Klasse, mit der die Wortarten gezählt werden.

5.2.2 Das Stadtkonzept

Eines der Hauptziele dieser Bachelorarbeit ist es eine Visualisierung zu erschaffen, in der sich die Struktur des Textes möglichst gut erkennbar widerspiegelt. Das Wichtigste ist, dass der Benutzer durch das Gesehene einen Bezug zum Aufbau des Textes herstellen kann. Da in *text2City* das Dokument das größte Textelement ist, liegt es nahe, dieses als Planvorlage der Stadt auszuwählen. Dokumente lassen sich meistens in Kapitel unterteilen. Dementsprechend werden sie als Stadtteile (Bezirke) dargestellt. Ist das Dokument kurz und/oder besitzt es keine Kapitel, so wird auch die Stadt eine kleinere Größe annehmen und nur aus einem Stadtteil bestehen. Stadtteile setzen sich aus einem oder mehreren Grundstücken zusammen, auf denen sich Gebäude befinden. Bei der Frage, wie ein Kapitel durch einen Stadtteil dargestellt werden kann, taucht ein Übersichtlichkeitsproblem in der Visualisierung auf: Da Kapitel teilweise aus sehr vielen Absätzen bestehen können, wären die Gebäude, die die Absätze darstellen so hoch, dass die Inhalte der oberen Etagen von unten nicht mehr lesbar wären. Aus diesem Grund werden in *text2City* Gebäude auf zehn Etagen limitiert, wobei das Erdgeschoss den Eingangsbereich und die weiteren maximal neun Stockwerke Absätze darstellen. Alle verbleibenden Absätze des Kapitels werden auf weitere Gebäude im selben Stadtteil aufgeteilt.

Beispiel: *Bei einem Kapitel mit 15 Absätzen würden zwei Hochhäuser entstehen: Hochhaus 1 besteht aus dem Eingangsbereich und 9 Etagen für die Absätze 1 bis 9. Die verbleibenden 6 Absätze lassen ein zweites Hochhaus entstehen, das somit aus 7 Stockwerken besteht: Eingangsbereich und 6 Etagen für Absätze 10-15.*

Für die Visualisierung und das Annotieren von Absatzinhalten sind Stockwerke zwar praktisch, wenn man aber das ganze Kapitel lesen möchte, ist es oft umständlich das Gebäude zu wechseln. Aus diesem Grund existiert ein Gebäude, das nur für diesen Zweck erschaffen wurde: Das Bezirksverwaltungsgebäude. Letzteres besteht nur aus dem Erdgeschoss und ist nicht betretbar, zeigt aber den gesamten Text des Kapitels an. Die Visualisierung kleinerer Textelemente wird in Unterkapitel 5.3.1 beschrieben, da diese nicht mehr Teil des Stadtbildes sind, sondern sich im inneren der Gebäude befinden.

5.2.3 Der Stadtgenerierungsalgorithmus

In Kapitel 2 wurde *resources2City* bereits als Basis von *text2City* beschrieben. Der entscheidende Unterschied zwischen *resources2City* und *text2City* liegt einerseits bei der Datenquelle, denn als solche wird statt einem Speicherort eine Textdatei verwendet. Anderseits sind es, entsprechend der Datenquelle, auch andere Informationsarten, die die Anordnung der Gebäude bestimmen. Bevor auf die Anordnung im nächsten Absatz genauer eingegangen wird, müssen noch einige Eigenschaften der Stadt aufzeigt werden: Die Y-Achse des Koordinatensystems ist für die Höhe reserviert, also werden die Stadtgebäude auf der X-Z-Ebene angeordnet. Da Absätze eines Kapitels auf mehrere Gebäude aufgeteilt werden können, sollen letztere aus Übersichtlichkeitsgründen möglichst gruppiert stehen. Deswegen wird jedem Kapitel eine eigene, im Endergebnis aber unsichtbare, Parzelle in der Stadt zugewiesen. Alle Parzellen sind gleich groß und entsprechen der Fläche, die für die Gebäude des längsten Kapitels benötigt werden. Nachdem der Algorithmus die Parzellengröße ermittelt hat, können innerhalb der Parzellen die Gebäudepositionen errechnet werden. Dabei sollten die Gebäude aus Platzgründen innerhalb der Parzelle möglichst gut verteilt liegen.

Um die Gebäude zu positionieren, muss der Algorithmus für jedes einzelne eine X- und eine Z-Koordinate festlegen. Beide Koordinaten werden von jeweils einer Texteigenschaft abhängig gemacht und anhand ihrer berechnet. Beim Ausprobieren verschiedener Eigenschaftspaare hat sich die Textlänge in der Kombination mit der Anzahl der Verben im Text für eine gleichmäßige Verteilung am besten bewährt. Umso länger der Text (bei Absatzgruppen die Gesamtlänge), desto größer die X-Koordinate und umso mehr Verben im entsprechenden Text vorkommen (bei Absatzgruppen die Summe der Verben), desto größer fällt die Z-Koordinate der Gebäudeposition aus.

text2City verwendet, genau wie auch *resources2City*, den Voronoi-Algorithmus für das Stadtbild. Im Anfangsstadium der Entwicklung von *resources2City* wurde zuerst ein Gitteralgorithmus für die Anordnung der Gebäude entwickelt. Aus Übersichts-, Platzgründen und durchaus auch aus ästhetischer Hinsicht hat sich im späteren Verlauf der Voronoi-Algorithmus als die bessere Wahl fürs Anordnen erwiesen. Der Voronoi-Algorithmus benötigt eine beliebige Menge an Punkten auf einer Fläche. Bei *text2City* sind die errechnete Gebäudepositionen die Punkte für den Voronoi-Algorithmus. Aus der Punktemenge werden die Zellen (die Grundstücke der Stadt) mit ihren Grenzlinien (Straßen) berechnet.

Die Form einer Zelle bestimmt die Form des zugehörigen Gebäudes, jedoch ist das Erdgeschoss um 60% kleiner. Der Etagenumfang kann von Stockwerk zu Stockwerk variieren. Umso länger der Text eines Absatzes im Vergleich zu den anderen Absätzen eines Hochhauses, desto größer der Umfang der Etage. Sobald das 3D-Modell eines Gebäudes gezeichnet wurde, wird jede Etage an den Außenfassaden mit dem zugehörigen Absatztext dekoriert. Dabei wird die Schriftgröße vom Algorithmus so festgelegt, dass der Text auch bei höher gelegenen Stockwerken noch lesbar ist. Fassadenflächen, die die Mindestlänge von einem Meter nicht erfüllen, zeigen ebenfalls aus Lesbarkeitsgründen keinen Text an. Eine Außenfassade wird neben dem Text mit der Farbe der festgestellten DDCKategorie dekoriert. Mehrstöckige Gebäude erhalten eine Eingangstür im Erdgeschoss. Damit sind alle Prozesse der Stadterstellung vollständig durchgeführt, der Benutzer kann in die Stadt „teleportiert“ werden.

Abbildung 31 im Anhang zeigt den Gesamtprozess der Stadterstellung in der Übersicht. Abbildung 10 zeigt *text2City* nach der Erstellung aus der Sicht des Benutzers. Die folgenden Unterkapitel beschäftigen sich mit den Interaktionsmöglichkeiten der virtuellen Stadt.

Abbildung 10: Das Stadtbild von *text2City* aus der Sicht des Benutzers

5.2.4 Orientierung und Bewegung in der virtuellen Stadt

Um zu sehen, welche Grundstücke zum selben Stadtteil gehören, werden alle Bereiche eines Stadtteils blau markiert, sobald man eins von ihnen betritt. Auf der Minimap sind diese markierten Areale, neben dem Namen des Kapitels, ebenfalls sichtbar. Die Minimap kann durch Betätigen des rechten Triggers eingeblendet werden. Durch Scrollen mit dem rechten Stick lässt sich das Bild in der Minimap verschieben. Wird der Stick beim Verschieben zusätzlich gedrückt gehalten, so wird hinaus- oder hineingezoomt. Die Minimap kann den Benutzer zu einem beliebigen Ort der Stadt versetzen. Dazu muss man auf einen bestimmten Punkt der Minimap zeigen und die Click-Aktion auslösen. Die Long-ClickFunktion der Minimap zentriert das Minimap-Bild über dem Benutzer. Abbildung 11 zeigt die aktivierte Minimap und den blau markierten Stadtteil, in dem sich der Benutzer aktuell aufhält.

Neben der Minimap-Teleportation und der normalen Bewegung, die mit dem rechten Stick durchgeführt wird, gibt es zwei weitere Möglichkeiten um in *text2City* die Position zu wechseln: Teleportieren zu einem sichtbaren Ort und zur nächsten Straßenkreuzung. Um zu einem sichtbaren Ort versetzt zu werden, zeigt man zuerst auf die gewünschte Stelle am Boden und betätigt die Click-Taste. Die Straßen der Stadt besitzen ab einer bestimmten Länge interaktive Pfeile an den Enden, die jeweils auf die sich am anderen Ende der Straße befindende Kreuzung zeigen. Ihre Click-Aktion versetzt den Benutzer in diese Kreuzung.

Abbildung 11: Die Minimap

5.2.5 Die Kategorie-Legende und erweiterte Suche von *text2City*

Die Tastenkombinationen der VR-Controller sind deutlich begrenzter, als die von einer PC-Tastatur. Aus diesem Grund sind nur oft benutzte Funktionen, wie zum Beispiel die Minimap, mit dem Controller aufrufbar. Die Kategorie-Legende und die erweiterte Suche in *text2City* gehören nicht zu den häufig verwendeten Funktionen und können deshalb nur über das Smartwatch-Menü aufgerufen werden. Beide Elemente erscheinen nach dem Öffnen mit einer Animation direkt vor dem Auge des Benutzers.

Die Legende dient als Übersichtstabelle für die zehn DDC-Hauptkategorien und der zugeordneten Farben. Dem Benutzer wird eine Möglichkeit angeboten Informationen zu filtern, ohne suchen zu müssen: Ein Click auf ein jeweiliges Element in der Legende schaltet die Verfärbung an allen Fassaden mit der entsprechenden Kategorie ein oder aus. Weitere Kriterien zum Filtern bestimmter Informationen bietet die erweiterte Textsuche (siehe Abbildung 12).

Das Suchmodul besteht aus drei Komponenten: Ein Panel zur Einstellung der Suchparameter, eine DDC-Übersicht und ein Anzeigefenster für die Suchergebnisse. Frühere Suchergebnisse bleiben erhalten. Die Folgende Auflistung zeigt alle einstellbare Suchparameter: Der Suchbegriff, die Suchebene (Dokument, Kapitel, Absatz, Satz oder Token), ein Sentiment-Wert und die Relation der Ergebnisse zu diesem Wert (kleiner, kleinergleich, gleich, größer-gleich oder größer), die DDC-Kategorie(n) und die Anzahl der Ergebnisse. Wenn die Suche erledigt wurde, erscheint ein neues Anzeigefenster, in dem alle Ergebnisse als Vorschau aufgelistet werden. Jeder Vorschau ist herausnehmbar und entfaltet sich beim Loslassen zur bildschirmfüllenden Normalansicht. War ein Suchbegriff angegeben, so wird dieser im Text mit orangener Farbe hervorgehoben.

Abbildung 12: Die erweiterte Textsuche

5.2.6 Gebäude-Interaktionen

Alle Gebäudeteile, die einen Text visualisieren, zeigen ihn an ihren Fassaden an. Es kann bei längeren Textabschnitten öfters vorkommen, dass sie auf der zugewiesenen Fläche nicht vollständig angezeigt werden können. Für solche Fälle wird die Möglichkeit zum Durchscrollen des Textes an jedem betroffenen Gebäudeteil angeboten. Dazu fokussiert man den Gebäudeteil, an dem gescrollt werden soll und bewegt gleichzeitig den rechten Stick in die gewünschte Richtung.

Es ist möglich, dass die DDC-Kategorie des Textinhalts vom *Text Structure Analyzer* nicht festgestellt werden konnte oder verändert werden muss. Etagen und Bezirksverwaltungsgebäude bieten dem Benutzer die Möglichkeit die DDC-Kategorie des Textes, den sie visualisieren, anzupassen. Durch einen Click auf einen solchen Gebäudeteil öffnet sich ein Panel, der zum einen den Text anzeigt und zum anderen über einen AnnotationsButton verfügt. Die Aktivierung dieses Buttons lässt ein rundes Auswahlmenü erscheinen, in welchem alle zehn Hauptkategorien in eingefärbten Segmenten aufgelistet sind (siehe Abbildung 13). Ein Cursor, der sich zum Anfang in der Mitte des Auswahlmenüs befindet, lässt sich mit dem rechten Stick in jede beliebige Richtung bewegen. Durch einen Doppel-Click auf demselben Touchpad wird die Kategorie ausgewählt, über dessen Segment der Cursor schwebt. Die Fassadenfarbe des Gebäudeteils wird entsprechend der neu ausgewählten Kategorie verändert.

Die letzte Gebäude-Interaktion ist das Betreten eines Gebäudes. Die Eingangstür eines Hochhauses (siehe Unterkapitel 5.2.3) lässt sich per Click aktivieren. So wird der TextRaum mit den Absätzen des Hochhauses dekoriert und der Benutzer in den ersten Stock des Gebäudes versetzt.

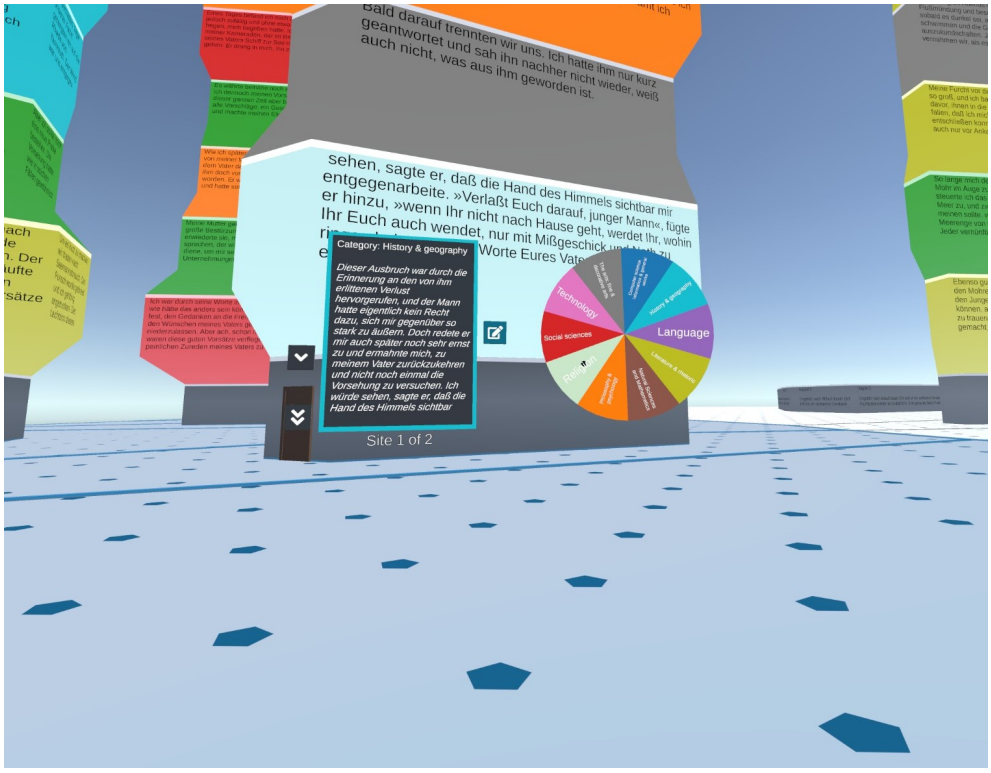

Abbildung 13: Das Annotieren eines Absatzes

5.3 Im Inneren eines Gebäudes

5.3.1 Das Text-Raum-Konzept

Im Gegensatz zu der dynamisch generierten Stadt, besitzt der Text-Raum eine vordefinierte Form. In Unterkapitel 5.2.2 wurde die Visualisierung des Dokuments, Kapitels und Absatzes beschrieben. Absätze werden in der Stadt als Stockwerke dargestellt und sind betretbar. Dementsprechend ist für jede Etage auch der Innenraum definiert.

Das nächstkleinere Textelement nach dem Absatz ist der Satz. Zum einen wird der Text jedes Satzes an der Wand des Text-Raums angezeigt. Zum anderen, wenn der Absatz aus mehreren Sätzen zusammengesetzt ist, werden letztere auf dem in der Mitte des Raumes stehenden Tisch in einem Voronoi-Diagramm-Modell angezeigt. Statt Gebäude besitzt dieses Voronoi-Diagramm-Modell Balken, die die Sätze repräsentieren. Die Höhe der Balken zeigt die zueinander relativen Satzlängen und die Farben der Balken die DDCKategorien der Sätze an. Die Anordnung der Balken erfolgt nach dem gleichen Prinzip, wie die der Gebäude in der Stadt (siehe Unterkapitel 5.2.3).

Das Token gilt in *text2City* als das kleinste strukturelle Textelement. Wie in Tabelle 7 ersichtlich, besitzen Tokens keine visuelle Darstellung. Alle Tokens werden vom *TextAnnotator* dem Dokument zusätzlich als Multi-Tokens hinzugefügt (siehe Unterkapitel 5.1). Da Multi-Tokens auch annotiert werden können, ergibt es aus Übersichtlichkeitsgründen mehr Sinn, nur diese visuell anzuzeigen. Multi-Tokens werden ebenfalls auf dem Tisch des Raumes präsentiert und durch Plättchen mit dem Text darauf visualisiert. Obwohl sie visualisiert werden, gelten sie als nicht-strukturelle Elemente (siehe Tabelle 7). Der Grund dafür ist erstens, dass sie erst nach der strukturellen Analyse durch den *TextAnnotator* in das Dokument eingefügt werden und zweitens sind sie durch ihre Veränderbarkeit (durch zusammenfügen, löschen, etc.) keine festen Bestandteile des Textes. Die Wandfarbe des Raumes zeigt die Stimmung des Textes an: Bei den Sentiments wird rot als negative, weiß als neutrale und grün als positive Farbe verwendet. Je größer der absolute Sentiment-Wert, umso gesättigter ist das Grün bei positivem, oder das Rot bei negativem Vorzeichen (zu sehen in Abbildung 14, Abbildung 16 und Abbildung). In den folgenden Unterkapiteln werden die Interaktionsmöglichkeiten des Text-Raums behandelt.

5.3.2 Das Verlassen des Text-Raums und der Wechsel zwischen Etagen

Der Text-Raum verfügt über eine Tür, die als Schnellausgang in die Stadt benutzt werden kann (siehe Abbildung 14 links). Durch die Click-Aktion dieser Tür wird der Text-Raum geschlossen und der Benutzer auf die Position zurückversetzt, auf der er beim Betreten des Raumes der Stadt stand.

Für den Wechsel zwischen Etagen ist der Text-Raum mit einem interaktiven Aufzug ausgestattet. In der Praxis existiert tatsächlich nur ein Text-Raum, dessen Einrichtung bei der Zuweisung eines anderen Absatzes verändert wird. Der interaktive Aufzug sollte lediglich den Etagenwechsel simulieren. Durch Aktivierung des Knopfes Auf der linken Seite des Aufzuges öffnen sich die Aufzugstüren. Im Aufzugsinneren befindet sich ein Nummernblock mit den Zahlen 1 bis 9. Die Taste des aktiven Stockwerks leuchtet grün, alle anderen aktivierbaren Knöpfe blau. Alle nicht ansteuerbare Ziffern bei einem Hochhaus mit weniger als 9 Etagen werden ausgegraut. Durch Click auf einen der Knöpfe schließen sich die Aufzugstüren. An dieser Stelle wird der Text-Raum mit dem dazugehörigen Absatz des Hochhauses neu dekoriert und die Türen öffnen sich wieder.

Abbildung 14: Im Text-Raum

5.3.3 Der Annotationstisch

In Unterkapitel 5.3.1 wurde bereits der in der Mitte des Text-Raumes stehende Tisch erwähnt. Der Tisch kann sowohl das Voronoi-Diagramm-Modell aller Sätze, als auch die Multi-Tokens einzelner Sätze anzeigen. Dementsprechend verfügt der Tisch über zwei Ansichtsmodi: Der „Voronoi"- (siehe in Abbildung 14) und der „Sentence"-Modus (siehe in Abbildung 16). Ein Button auf dem Tisch ermöglicht per Click den Wechsel zwischen den beiden Modi. Im „Vorono"-Modus sind die Balken, die die Sätze darstellen, interaktiv. Sobald ein Balken fokussiert wird, wird der passende Text an der Wand farbig hervorgehoben (siehe in Abbildung 14 im Hintergrund). Ein Click auf den gewünschten Balken öffnet das Annotationspanel, mit dem die Kategorie des Satzes festgelegt werden kann. Das Annotationspanel sieht aus und funktioniert, wie das Annotationspanel von Gebäudeteilen (siehe in Unterkapitel 5.2.6 und in Abbildung 13).

5.3.4 Die Annotation von Multi-Tokens

Das Annotieren von Multi-Tokens ist ein komplexerer Prozess und wird in diesem Unterkapitel separat behandelt. *text2City* implementiert nur die wichtigsten Funktionen des QuickAnnotator. Der QuickAnnotator ist eines der auswählbaren Tools des *TextAnnotator*. Das Tool bietet dem Benutzer zum einen die Möglichkeit zum Zusammenbinden und Aufsplitten von Multi-Tokens. Zum anderen können letztere durch Erstellung von Named Entities mit Tags verbunden werden (siehe das Beispiel am Ende von Unterkapitel 5.1). Da das Annotieren in *text2City* dem auf der *TextAnnotator*-Weboberfläche nachempfunden wurde, wird an dieser Stelle kurz auf die QuickAnnotator-Weboberfläche eingegangen

Nach Auswahl des zu annotierenden Dokuments werden dem Benutzer die vorhandenen Bearbeitungstools aufgelistet. Nachdem aus der Liste der QuickAnnotator ausgewählt wurde, kann die Bearbeitung begonnen werden. Auf der linken Seite der Weboberfläche werden die aktuell geladenen Multi-Tokens angezeigt. Weitere können durch Angabe der gewünschten Seitenzahl angezeigt werden (siehe Abbildung 15 rechts). Durch das gedrückt halten der Umschalt-Taste können Multi-Tokens zusammengefügt werden, in dem der Benutzer mit der Maus über dem ersten Multi-Token die linke Maustaste gedrückt hält, auf den zweiten Multi-Token zieht (angezeigt durch eine blaue Verbindungslinie, siehe Abbildung 15 oben links) und dort die Maustaste loslässt. So entsteht ein neuer Multi-Token der das Start-, Ziel- und allen dazwischen liegenden Multi-Tokens beinhaltet. Um ein Multi-Token aufzusplitten muss ebenfalls die Umschalt-Taste gedrückt gehalten und ein einfacher Linksklick auf das gewünschte Multi-Token getätigt werden. Um Multi-Tokens zu taggen, gibt es zwei unterschiedliche Möglichkeiten: Zum einen kann zuerst die passende Entität rechts ausgewählt (siehe Abbildung 15, rot eingerahmter Teil) und anschließend das Ziel-Multi-Token angeklickt werden. Zum anderen kann die Auswahlliste auch direkt am Ziel-Multi-Token durch Anklicken angezeigt und dort die passende Entität gewählt werden.

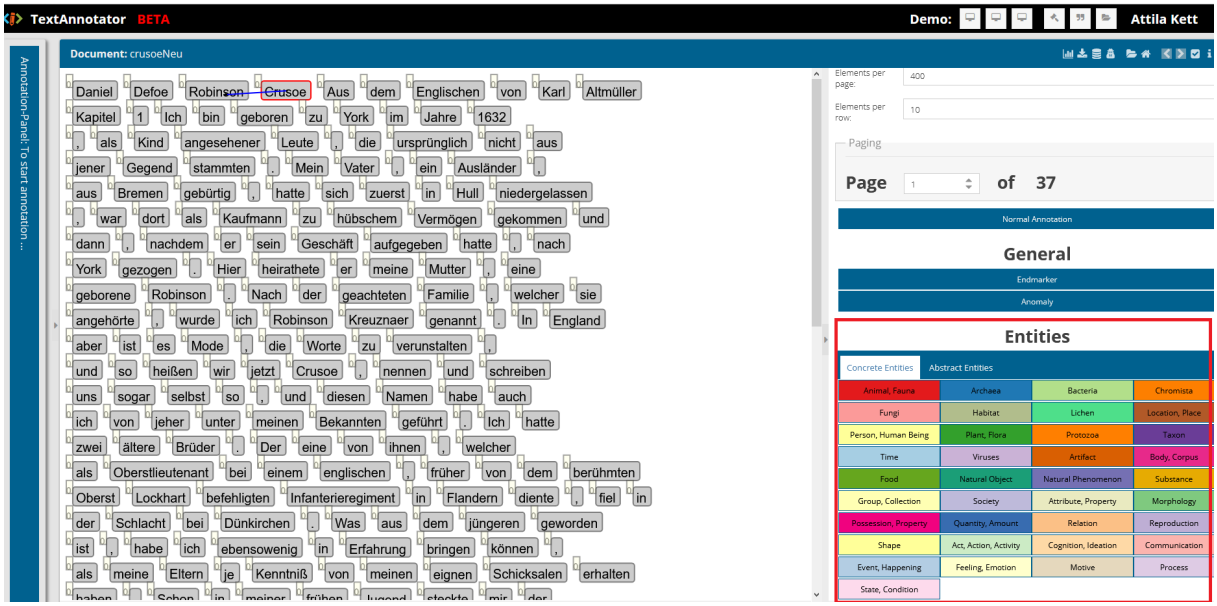

Abbildung 15: Die *TextAnnotator*-Weboberfläche [1]

Um in *text2City* Multi-Tokens annotieren zu können, muss zuerst der Ansichtsmodus des Tisches auf „Sentence“ gestellt werden (siehe Abbildung 16 unten). In diesem Zustand erscheint auf dem Tisch unter dem Modus-Button nun eine zusätzliche Option. Damit kann der Benutzer zwischen den Sätzen des Absatzes wechseln. Die Multi-Tokens des Satzes werden als graue Plättchen dargestellt und auf dem Tisch sortiert aufgelistet. Jedes Plättchen ist mit dem Text des Multi-Tokens beschriftet und besitzt zwei verschiedene Interaktionsmöglichkeiten:

- Wird der rechte Stick gedrückt gehalten, erscheint ein Pfeil mit der Position des ersten Multi-Tokens als Start und ein beliebiger aktuell fokussierter Punkt als Ende (siehe Abbildung 17). Die Farbe des Pfeils indiziert, ob beim Loslassen des Sticks eine Operation ausgeführt werden würde (grün), oder nicht (rot). Es gibt, wie auf der *TextAnnotator*-Weboberfläche zwei verschiedene Operationen: Zeigt der Pfeil auf das gleiche Plättchen, von dem aus er auch startet, so wird ein SplitBefehl für diesen Multi-Token an den *TextAnnotator* gesendet. Ansonsten wird ein Befehl zum Zusammenfügen von allen betroffenen Multi-Tokens abgeschickt. Es spielt keine Rolle ob das erstgewählte Multi-Token im Text vor oder nach dem zweitgewählten steht, um die Sortierung kümmert sich *text2City* automatisch.
- Durch einen Click auf ein Multi-Token öffnet sich die Auswahlliste aller Named Entity Typen. Das Kontrollkästchen vor einem Element der Liste zeigt an, ob für das Multi-Token ein Named Entity des entsprechenden Typen bereits angelegt ist (siehe in Abbildung 16). Die Named Entity Farben werden von der Weboberfläche übernommen.

In dieser Bachelorarbeit unterstützt *text2City* nur das online Bearbeiten von Dokumenten. Auch durch andere Benutzer gemachte Änderungen am gleichen Dokument werden durch die Websocket-Verbindung vom *TextAnnotator*-Server („change_cas“-Befehl, siehe Tabelle 2) sofort empfangen und umgesetzt. Das gilt natürlich auch visuell, sofern das zu verändernde Multi-Token aktuell angezeigt wird. In *text2City* durchgeführte Multi-Token-Annotationen werden als „work_batch“-Befehl (siehe Tabelle 2) zum Server geschickt.

Erst wenn der Server die Veränderung akzeptiert und alle Clients darüber benachrichtigt hat, wird die visuelle Veränderung in *text2City* vorgenommen. Diesen Ablauf zeigt Abbildung 30 im Anhand in der Übersicht.

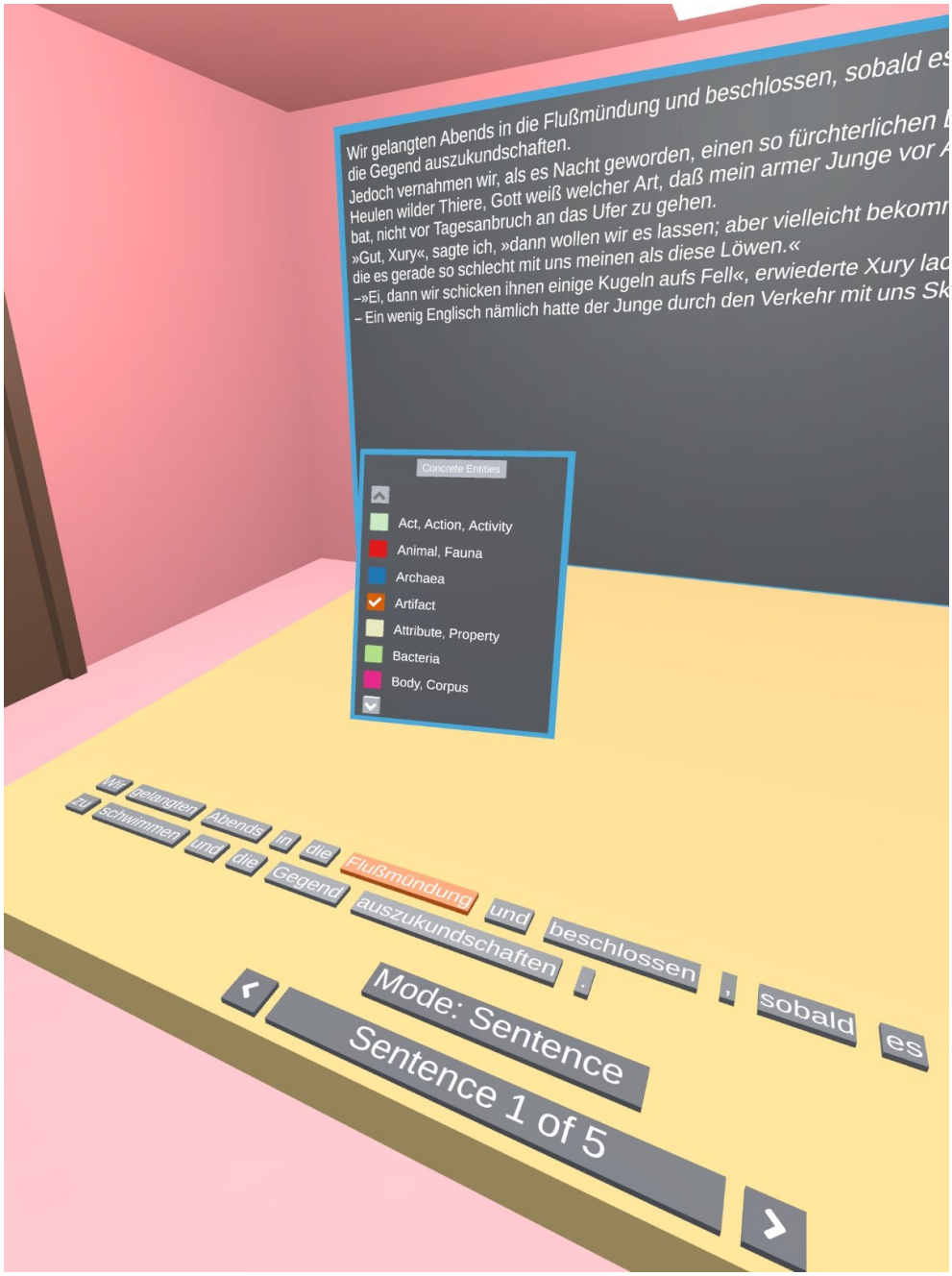

Abbildung 16: Named Entity erstellen

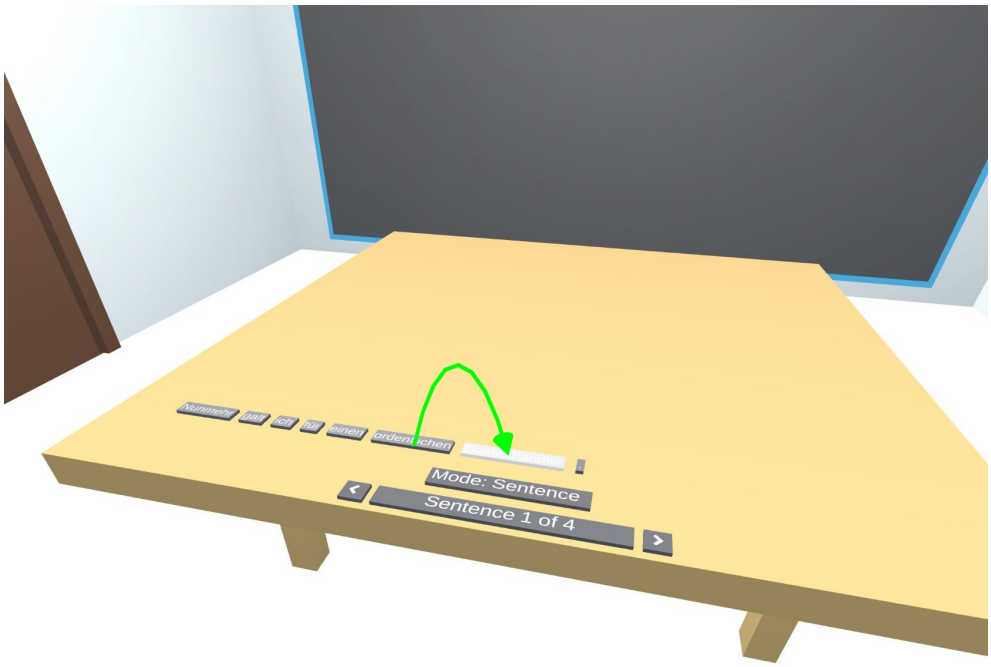

Abbildung 17: Multi-Tokens zusammenfügen

6 Evaluation

Die Evaluierung von *text2City* besteht aus zwei Teilen. Der erste Teil findet in *StolperwegeVR* statt und beschäftigt sich damit, wie gut die drei Hauptziele dieser Bachelorarbeit umgesetzt werden konnten. Die zweite Hälfte der Evaluation ist eine UMUX-Umfrage [9] rund um die Usability von *text2City*. An der Evaluation haben 17 Personen teilgenommen

6.1 Vorbereitungsphase

Der mit *text2City* für die Evaluation visualisierte Text liegt dem Teilnehmer ausgedruckt vor. Als Vorbereitung kann sich der Teilnehmer den ausgedruckten Text kurz ansehen, ohne ihn dabei vollständig durchlesen zu müssen. Im zweiten Schritt soll ein Satz aus dem Text auf der *TextAnnotator*-Weboberfläche vom Teilnehmer nach Vorgaben annotiert werden. Die Annotationsvorgaben liegen ebenfalls ausgedruckt vor. Die auf der *TextAnnotator*-Weboberfläche gewonnene Erfahrung soll als Vergleichsbasis für die letzte Evaluationsaufgabe in *text2City* dienen.
Die Benutzung von *text2City* setzt voraus, dass der Benutzer mit der grundlegenden Steuerung in *StolperwegeVR* vertraut ist. Da die meisten Evaluationsteilnehmer vor der Evaluation *StolperwegeVR*, oder gar VR noch nie erlebt haben, wird Ihnen für die Eingewöhnungsphase und das Kennenlernen der Steuerung ein kleines Tutorial zur Verfügung gestellt.

6.2 Evaluation der Hauptziele

Die Evaluation in *StolperwegeVR* besteht aus 3 Aufgaben, mit je einer Aufgabe für jedes Hauptziel.

6.2.1 Aufgabe 1 – Wiedererkennung der Textstruktur

Die erste Aufgabe soll zeigen, wie gut der Wiedererkennungswert der Textstruktur im Stadtbild von *text2City* ist. Der Teilnehmer wird darum gebeten, sich in der virtuellen Stadt umzuschauen, sich ein Bild über den Stadtaufbau zu machen und anschließend die Textelemente Dokument, Kapitel, Absatzgruppe und Absatz mit den seiner Meinung nach passenden 3D-Elementen der virtuellen Stadt zu verbinden (siehe Abbildung 23). Geprüft wird bei Aufgabe 1, ob die Elemente richtig und vollständig verbunden wurden. Wenn der Teilnehmer den „Fertig“-Button betätigt, gilt die Aufgabe als abgeschlossen.

Auswertung der Ergebnisse

Die Lösung der Aufgabe wird in Tabelle 10 veranschaulicht:

Textelement	3D-Repräsentation
Dokument	Stadt
Kapitel	Bezirk **und** Gebäude ohne Etage
Mehrere Absätze	Hochhaus
Absatz	Etage

Tabelle 10: Die Lösung von Evaluationsaufgabe 1

Tabelle 11 zeigt die Zuordnungshäufigkeit pro 3D-Element (Zeilen) für jedes Textelement (Spalten). *Anmerkung: Die Summe der Zuordnungen pro Textelement entspricht nicht immer die Anzahl der Teilnehmer. Das liegt daran, dass mehrfache Zuordnungen möglich sind.*

3D-Repr./Textelement	Dokument	Kapitel	Mehrere Absätze	Absatz
Stadt	15	1	0	0
Bezirk	1	13	0	0
Gebäude ohne Etage	1	5	0	3
Hochhaus	0	0	16	1
Etage	0	1	1	15

Tabelle 11: Die Ergebnistabelle von Evaluationsaufgabe 1

Der Wiedererkennungswert vom Dokument liegt mit 15 richtigen Antworten von 17 Teilnehmern bei etwa 88.2%.

Da das Kapitel durch zwei Objekte dargestellt wird, muss zuerst der Wiedererkennungswert pro Repräsentation berechnet werden. 13 Personen haben das Kapitel als Bezirk identifiziert, das entspricht etwa 76.4%. Lediglich 5 von 17 Personen haben das Kapitel als Gebäude ohne Etage erkannt, damit ergibt sich ein zweites Zwischenergebnis von ca. 29.4%. Der Durchschnitt und damit der Wiedererkennungswert des Kapitels liegen etwa bei 52.9%.

16 von 17 Teilnehmern haben Absatzgruppen als Hochhäuser identifiziert. Damit liegt der Wiedererkennungswert der Absatzgruppe bei ca. 94.1%.

Absätze wurden von 2 Teilnehmern auch als Gebäude ohne Etage erkannt. Eine mehrfache Zuordnung muss in diesem Fall als falsch angerechnet werden. Damit haben 13 von 17 Personen, die den Absatz richtig zugeordnet, das entspricht ungefähr 76.4%.

Mit dem Durchschnitt aller einzelnen Werte ergibt sich ein **Gesamtwiedererkennungswert von 77.9%.**

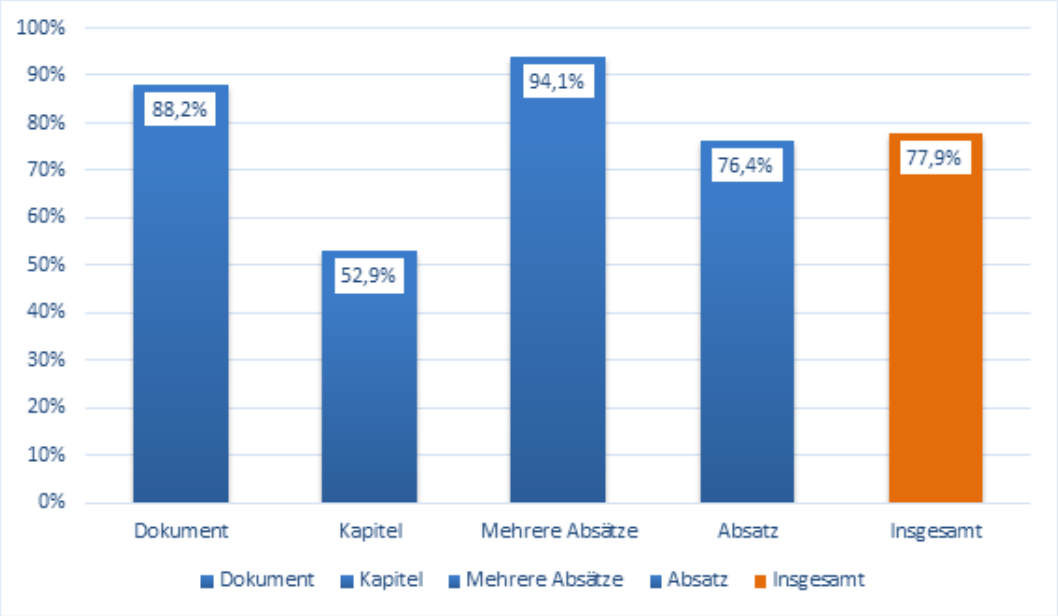

Abbildung 18: Der Wiedererkennungswert der virtuellen Textstadt

6.2.2 Aufgabe 2 - Textsuche

Die zweite Aufgabe befasst sich mit der erweiterten Suchfunktion von *text2City*. Der Teilnehmer soll den längsten Satz finden, in dem das Wort „Robinson“ vorkommt (siehe Abbildung 24). Nachdem ein Suchergebnis im dafür vorgesehenen Feld platziert wurde, sollen noch die folgenden zwei Fragen mit einer Zahl von 1 bis 10 bewertet werden:

Frage 1: Auf einer Skala von 1 bis 10, wie selbsterklärend finden Sie die Funktionen der Suche? (1: überhaupt nicht, 10: vollkommen)

Frage 2: Auf einer Skala von 1 bis 10, wie intuitiv finden Sie die Bedienung der erweiterten Suche von *text2City*? (1: überhaupt nicht, 10: vollkommen)

Auswertung der Ergebnisse und Fragen

Jeder Evaluationsteilnehmer hat das richtige Ergebnis gefunden, somit liegt die Erfolgsrate der Suche bei der vorgegebenen Aufgabe bei **100%**.

Tabelle 12 zeigt für welche Frage welche Punktzahl wie oft vergeben wurden.

Frage/Punkt	1	2	3	4	5	6	7	8	9	10
Frage 1	0	0	1	0	0	0	2	5	5	4
Frage 2	0	0	0	1	0	3	0	5	3	5

Tabelle 12: Die Punktvergabetabelle für die Fragen in Evaluationsaufgabe 2

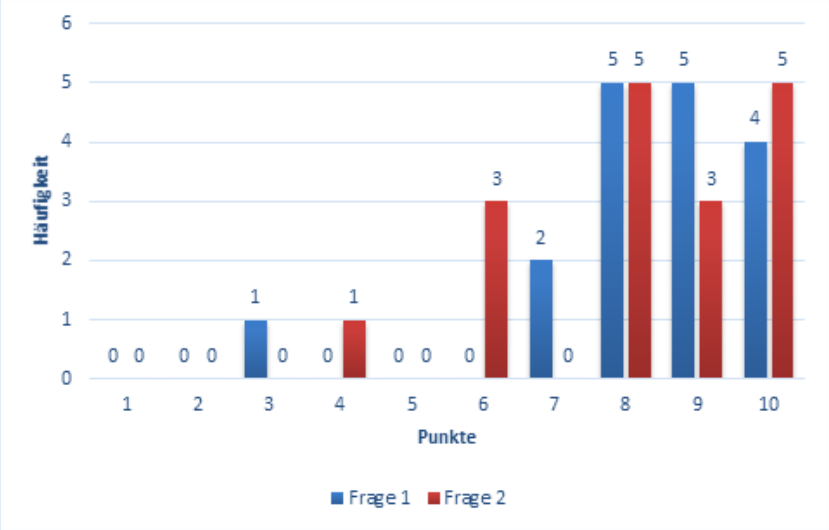

Abbildung 19: Das Punktvergabediagramm für die Fragen in Evaluationsaufgabe 2

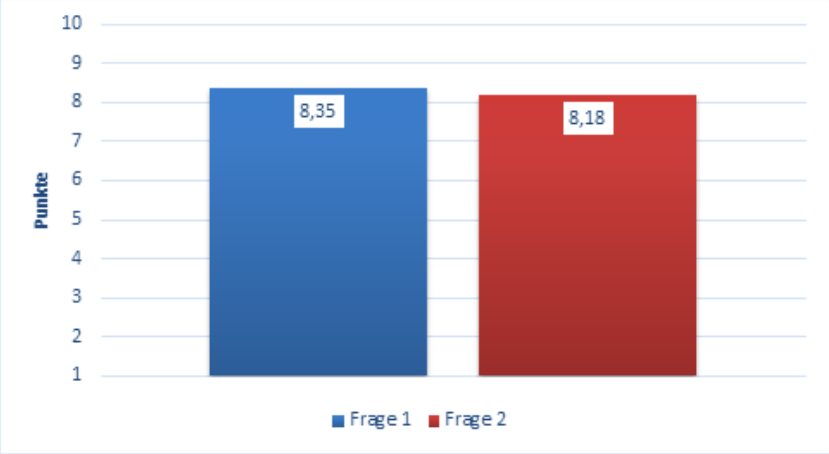

Abbildung 20: Durchschnittsbewertung der Fragen in Evaluationsaufgabe 2

6.2.3 Aufgabe 3 – Multi-Tokens Annotieren

Aufgabe 3 dient zur Evaluierung des Annotierens in *text2City*. Der Teilnehmer soll den gleichen Satz des Textes aus der Vorbereitungsphase mit den gleichen Vorgaben nun in *text2City* annotieren (siehe Abbildung 25). Anschließend sollen die folgenden zwei Fragen mit einer Zahl von 1 bis 10 bewerten:

Frage 1: Auf einer Skala von 1 bis 10, wie gut eignet sich die virtuelle Realität fürs Annotieren von Texten? (1: überhaupt nicht, 10: vollkommen)

Frage 2: Wie einfach ist das Annotieren in *text2City* im Vergleich zum *TextAnnotator*? (1: komplizierter, 10: genau so einfach)

Auswertung der Fragen

Tabelle 13 auf der nächsten Seite zeigt für welche Frage welche Punktzahl wie oft vergeben wurden.

Frage/Punkt	1	2	3	4	5	6	7	8	9	10
Frage 1	0	1	1	0	3	1	0	3	1	7
Frage 2	1	0	2	1	0	0	1	4	0	8

Tabelle 13: Die Punktvergabetabelle für die Fragen in Evaluationsaufgabe 3

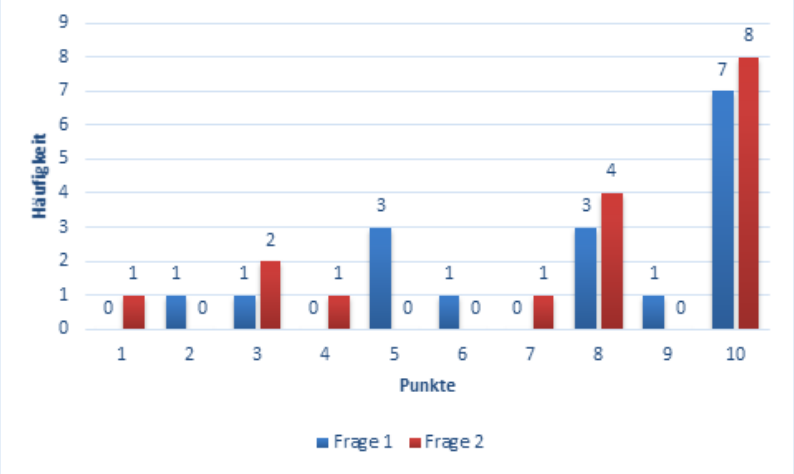

Abbildung 21: Das Punktvergabediagramm für die Fragen in Evaluationsaufgabe 3

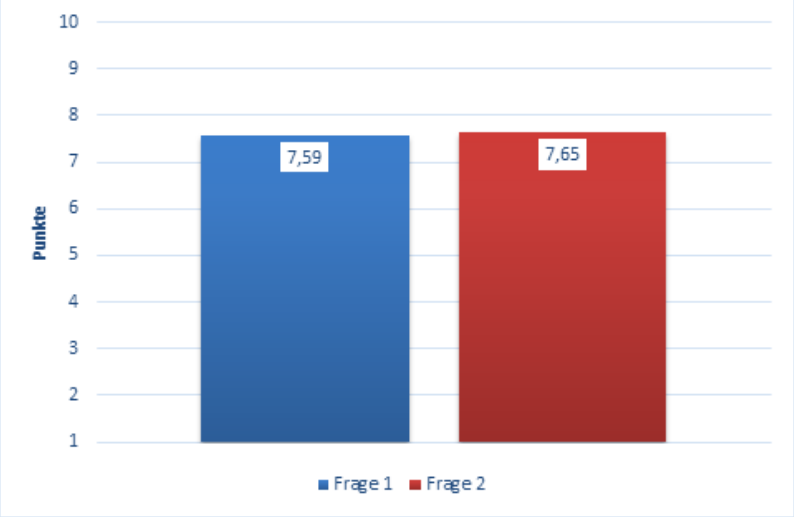

Abbildung 22: Durchschnittsbewertung der Fragen in Evaluationsaufgabe 3

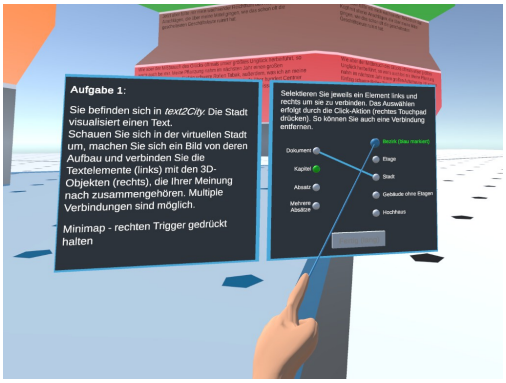

Abbildung 23: Screenshot der ersten Evaluationsaufgabe

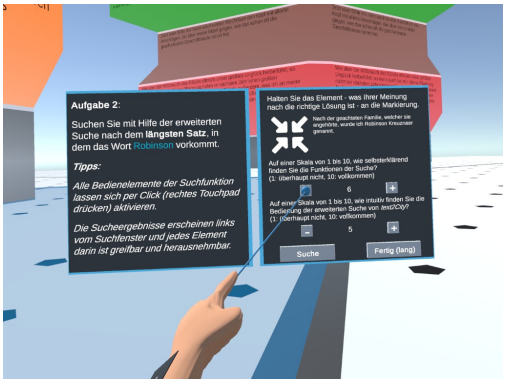

Abbildung 24: Screenshot der zweiten Evaluationsaufgabe

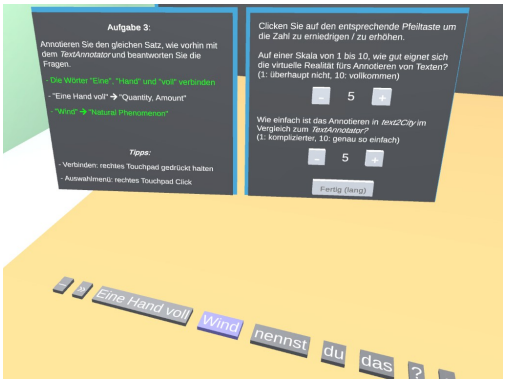

Abbildung 25: Screenshot der dritten Evaluationsaufgabe

6.3 Die Benutzerfreundlichkeit von *text2City*

Der UMUX-Fragebogen umfasst fünf Fragen, die jeweils mit 1 (Strongly Disagree) bis 7 (Strongly Agree) bewertet werden können. Das Ziel eines UMUX-Fragebogens ist es herauszufinden, wie gut die Verwendbarkeit von Softwares ist. Für *text2City* wurden folgende Fragen formuliert:

Frage 1: Ist die Benutzung von *text2City* ein frustrierendes Erlebnis?

Frage 2: Erfüllen die Werkzeuge und Hinweise zur Orientierung in *text2City* Ihre Erwartungen?

Frage 3: Ist *text2City* einfach zu verwenden?

Frage 4: Ist die erweiterte Suche in *text2City* einfach zu verwenden?

Frage 5: Sind Annotationen in *text2City* einfach durchzuführen?

Auswertung des Fragebogens

Tabelle 14 zeigt welcher Frage (Zeile) mit welcher Häufigkeit die Punkte vergeben wurden (Spalte).

Frage/Punkt	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Frage 1	10	3	1	2	1	0	0
Frage 2	0	0	0	3	3	7	4
Frage 3	0	0	1	3	4	4	5
Frage 4	0	0	1	1	2	3	10
Frage 5	0	0	3	0	3	6	5

Tabelle 14: Die Punktvergabetabelle für die Fragen des UMUX-Fragebogens

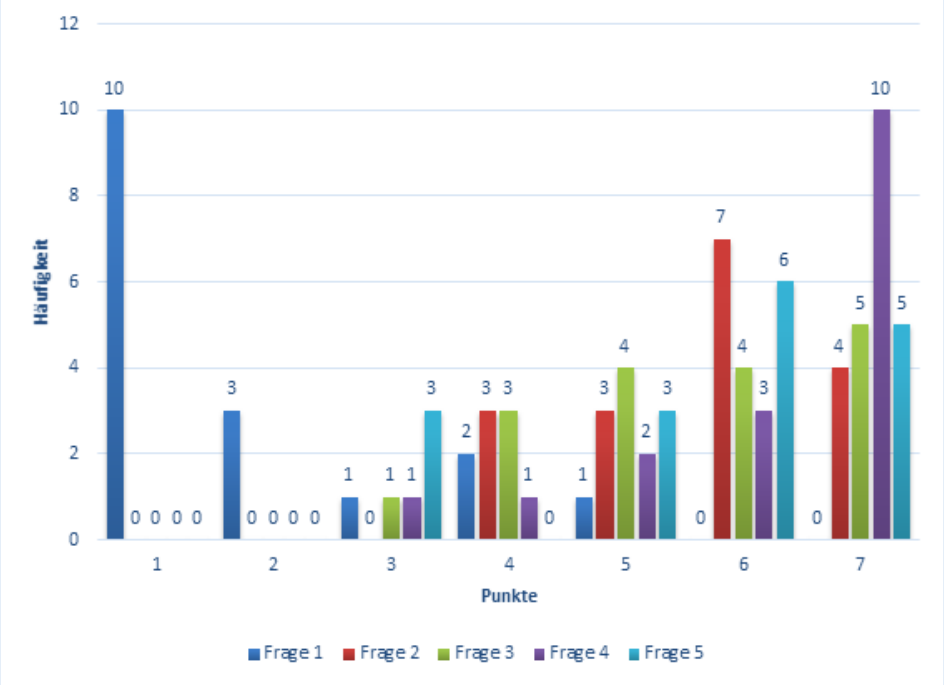

Abbildung 26: Das Punktvergabediagramm für die Fragen des Fragebogens

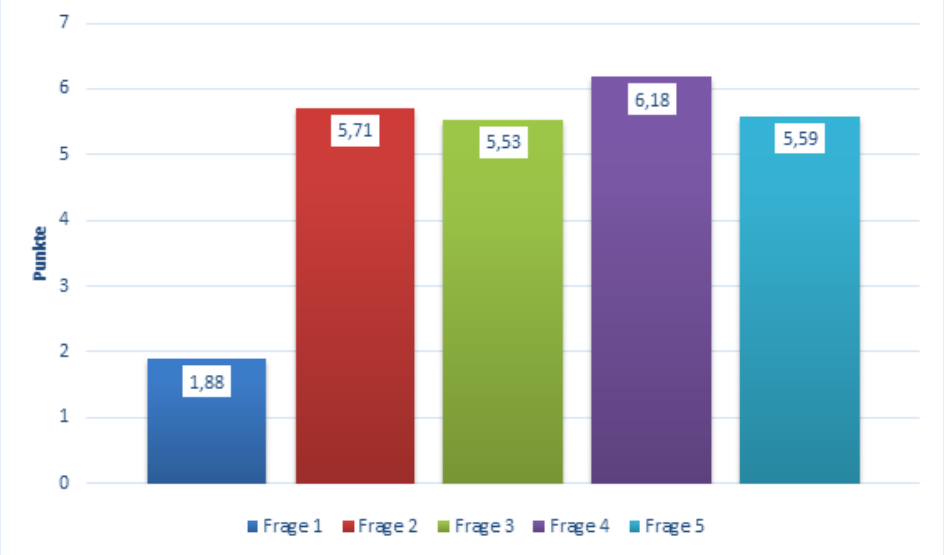

Abbildung 27: Durchschnittsbewertung der Fragen des Fragebogens

7 Zusammenfassung und Ausblick

Das Hauptziel dieser Arbeit ist es eine interaktive 3D-Visualisierung für Texte in der virtuellen Realität zu erschaffen, sodass die Textstruktur im Stadtbild bestmöglich widerspiegelt wird. Mit einem Gesamtwiedererkennungswert von 77,9% ist dieses Ziel durchaus gelungen. Den schlechtesten Wert mit 52.9% erzielte die Visualisierung des Kapitels in der Stadt. Eine bessere visuelle Veranschaulichung des Konzepts „Bezirk mit Bezirksverwaltungsgebäude“ könnte diesen Wert erheblich und damit den Gesamtwiedererkennungswert vermutlich noch stark erhöhen. Diese Erkenntnis sollte bei der Verbesserung von *text2City* unbedingt umgesetzt werden.

Ein weiteres Ziel von *text2City* ist es, eine intuitive Textsuche zu implementieren, die dem Benutzer spezifische Suchanfragen ermöglicht. Diese Funktion hat sowohl im virtuellen Teil (100% Erfolgsrate), als auch im UMUX-Fragebogen am besten abgeschnitten. Für die Zukunft könnte es interessant sein, die Suche mit anderen Suchparametern zu erweitern. Obwohl 16 von 17 Teilnehmer die Selbstverständlichkeit der Suchfunktionen mit 7/10 oder mehr Punkten bewertet haben, wäre es besonders für andere Einstellungsmöglichkeiten (wie zum Beispiel Sentiment-Werte) wichtig, die Suche mit einem Hilfestellungsfenster zu ergänzen.

Die dritte Intention der Arbeit ist das Ermöglichen des Annotierens in der virtuellen Stadt. Ein Durchschnitt von 5,59 von 7 Punkten zeigt, dass auch die Annotationsfunktion in *text2City* über eine annehmbare Usability verfügt. Dennoch haben mehr als die Hälfte der Teilnehmer (9 von 17) das Annotieren in *text2City* als komplizierter eingestuft, als das Annotieren auf der *TextAnnotator*-Weboberfläche. Das kann daran liegen, dass VR für die meisten Benutzer noch völliges Neuland ist. Das sollte aber nicht von der Verbesserung des Annotationstools in *text2City* abhalten.

Für die Weiterentwicklung von *text2City* würde es in erster Linie von hohem Nutzen sein, wenn das in der Entwicklungsphase befindliche ChapterSplitter-Analysetool des *Text Structure Analyzer* in den *TextImager* eingebettet werden würde. Durch die Benutzung des *TextImager* würden eine Menge neuer Tools zur Verfügung stehen, was zahlreiche Visualisierungsmöglichkeiten mit sich bringen würde.

Darüber hinaus sollte *text2City* mit weiteren Features des *TextAnnotator* erweitert werden. Zum Beispiel neben dem QuickAnnotator mit anderen Annotationstools des *TextAnnotator* erweitert werden. Auch andere Features, wie zum Beispiel offline Bearbeitung von Dokumenten aus dem *TextAnnotator* zu übernehmen.

8 Anhang

8.1 *text2City* - Startanleitung

Vorbereitung:

Für die Ausführung des Programms ist Oculus Rift, oder HTC Vive und ein VR-kompatibles PC erforderlich. Nach dem die Brille angeschlossen und eingerichtet wurde (Anleitung gibt es dazu von den Herstellern) muss die richtige *Unity*-Version auf dem PC installiert werden. Für diese Bachelorarbeit wird *Unity 2018.3.11* verwendet, dieses Release kann aus dem *Unity*-Archive unter https://unity3d.com/get-unity/download/archive [27] heruntergeladen werden. Nach der Installation kann direkt *Unity* gestartet werden, für ihre Verwendung muss man angemeldet sein (mehr dazu unter https://id.unity.com [26]). Schließlich kann auf dem angehängten Datenträger befindliche „*StolperwegeVR*-Ordner“ als Projekt ausgewählt werden.

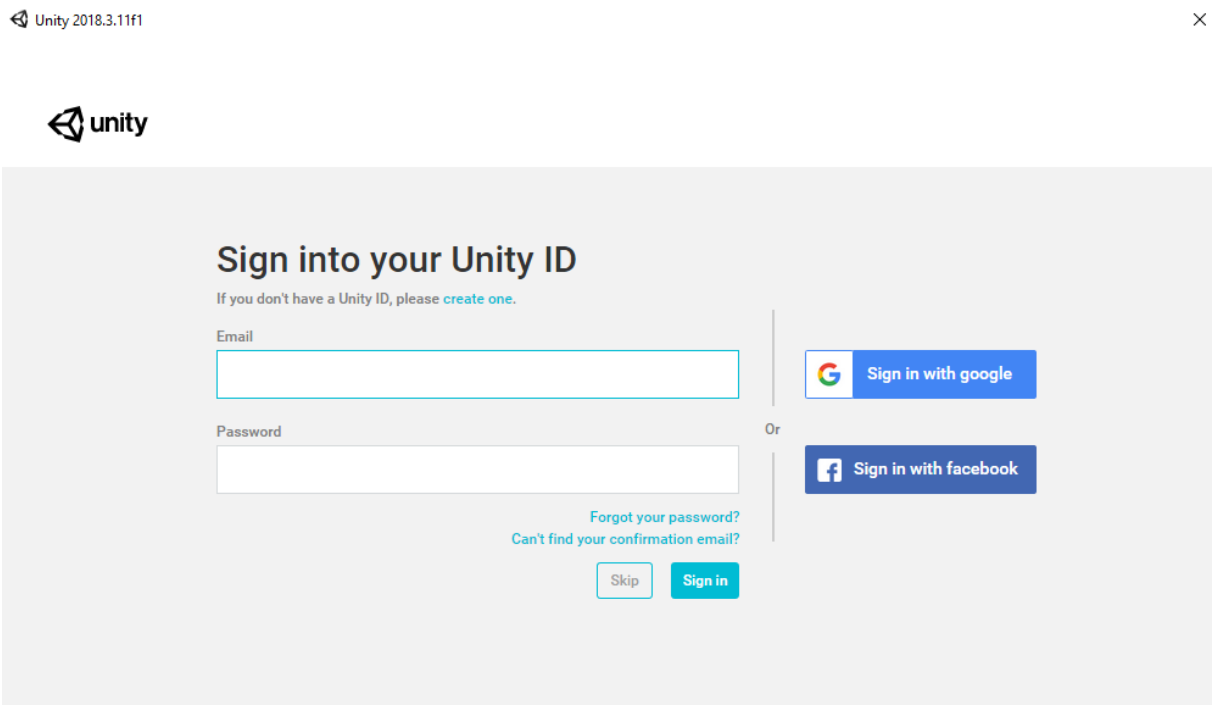

Abbildung 28: Das Login-Fenster von *Unity*

StolperwegeVR starten:

Sobald das Projekt erfolgreich mit *Unity* geöffnet wurde, muss vor dem Start noch die richtige Umgebung ausgewählt werden. Unten im Reiter „Project“ sieht man die komplette Struktur, dort muss nach „Assets/Text2City/Text2CityEvaluation.unity“ gesucht und durch einen Doppelklick auf diese Datei die Szene geöffnet werden. Mit dem „Play“-Button oben in der Mitte startet das Programm (siehe Abbildung 29): Auswahl der richtigen Szene und Start).

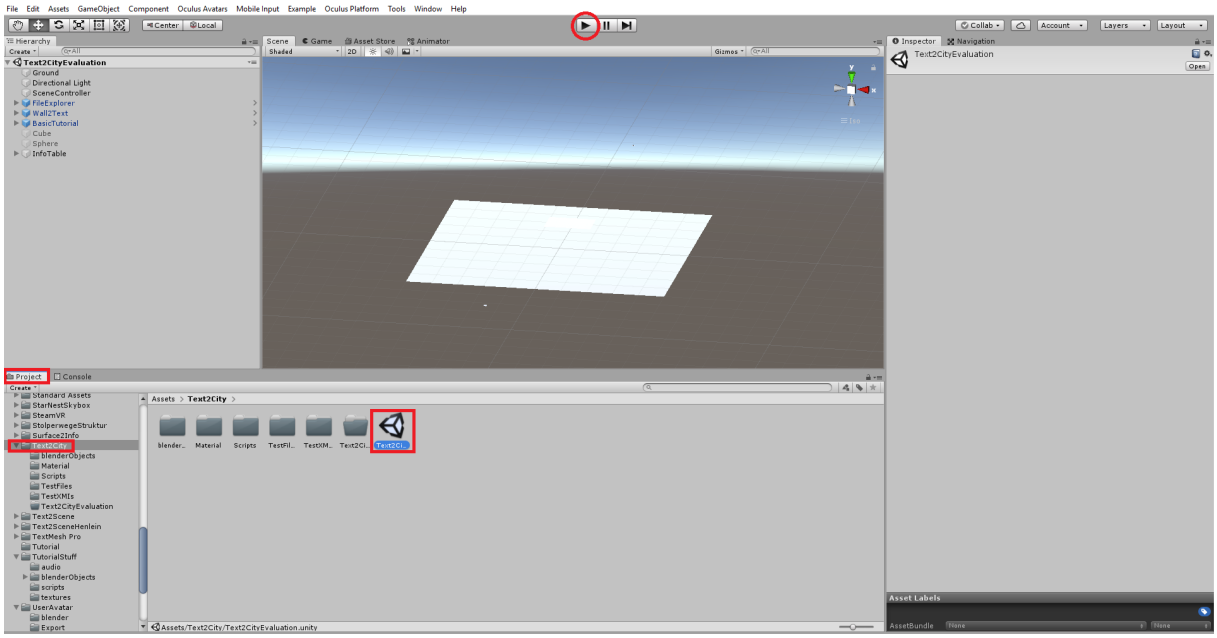

Abbildung 29: Auswahl der richtigen Szene und Start

text2City starten:

Die ausgewählte Szene wurde auch für die Evaluation verwendet. Nachdem der Countdown abgelaufen ist, wird automatisch das Basis-Tutorial gestartet. Wenn die Steuerung bereits bekannt ist, kann der Countdown auf Wunsch mit dem „Start!“-Button unterbrochen und damit gleichzeitig auch *text2City* gestartet werden.

8.2 Diagramme

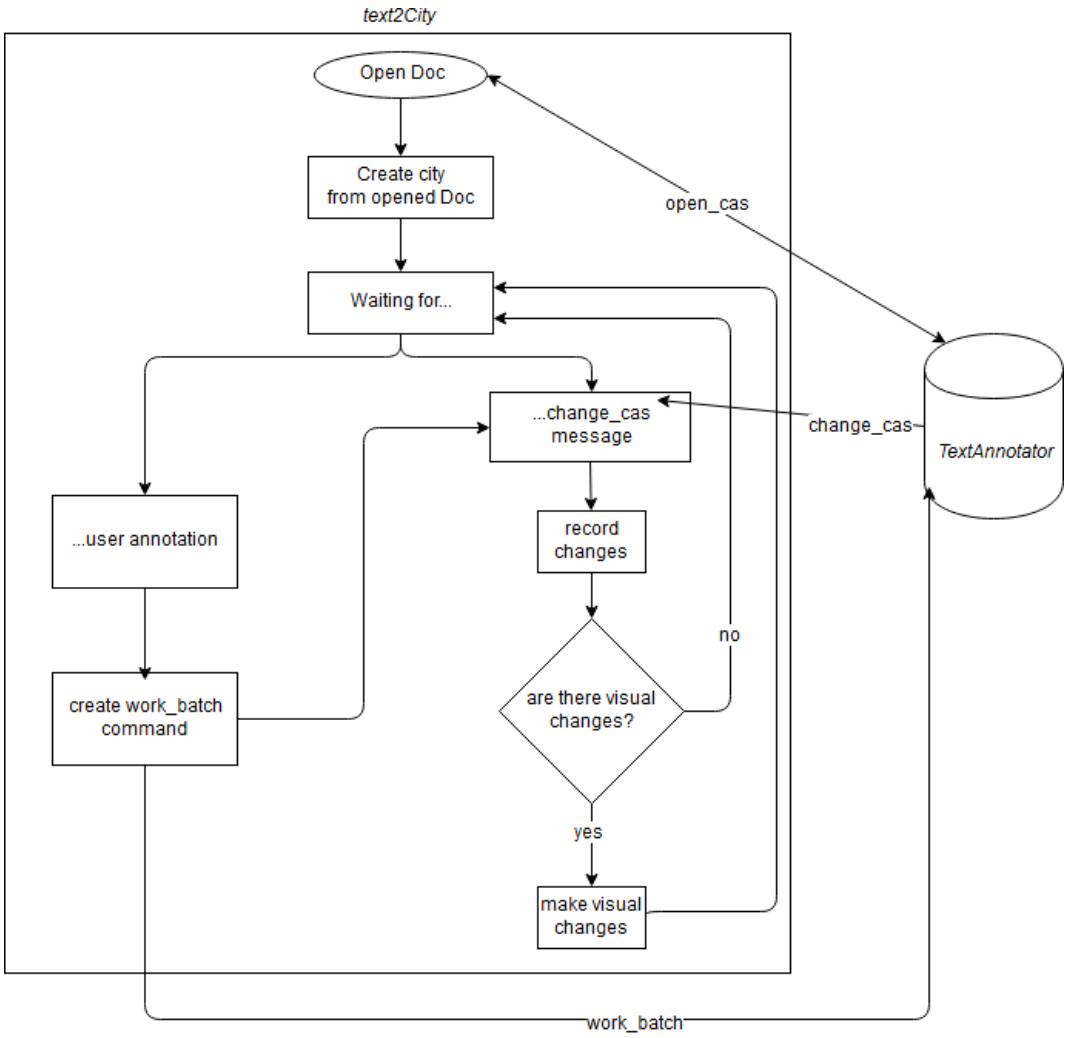

Abbildung 30: Die Kommunikation zwischen *text2City* und *TextAnnotator* beim Annotieren eines Dokuments

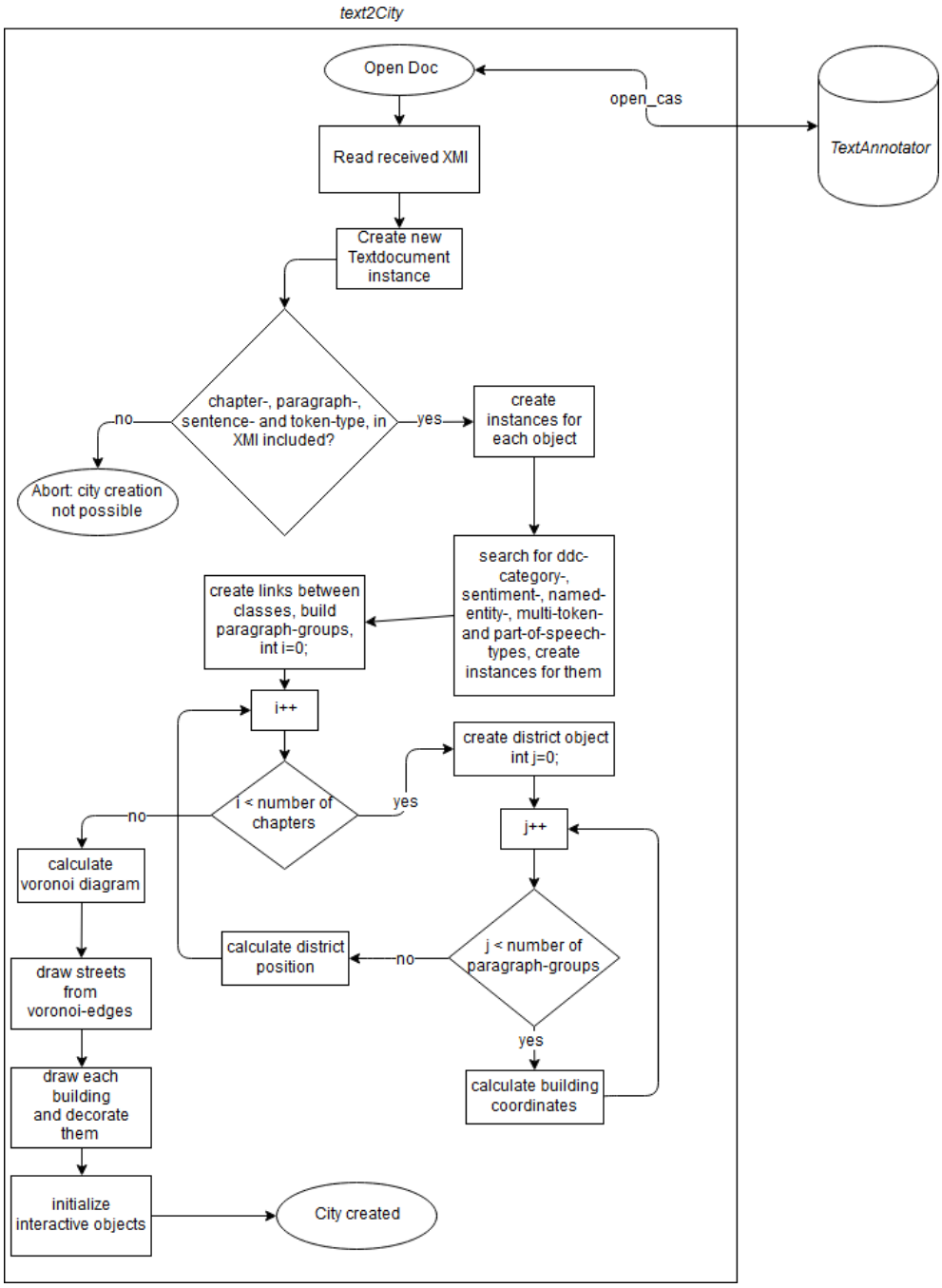

Abbildung 31: Der Ablauf des Stadtgenerierungsalgorithmus

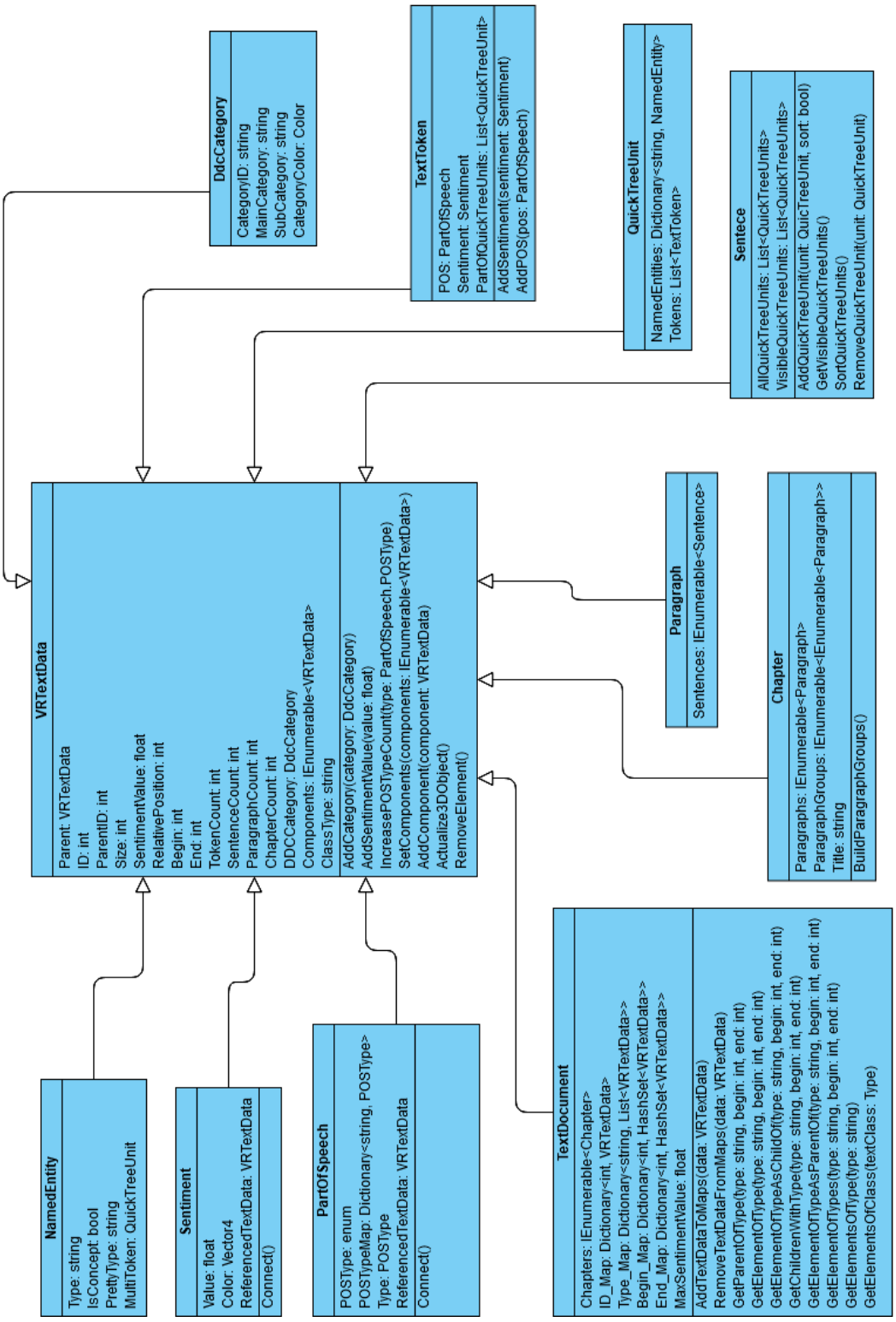

Abbildung 32: Klassendiagramm der Textelemente-Klassen

References

[1] Giuseppe Abrami, Alexander Mehler, Andy Lücking, Elias Rieb, and Philipp Helfrich. TextAnnotator: A flexible framework for semantic annotations. In *Proceedings of the Fifteenth Joint ACL - ISO Workshop on Interoperable Semantic Annotation, (ISA-15)*, ISA-15, May 2019.

[2] Blender. Blender. " https://www.blender.org/", 2019. Zugriff: 26. Juni 2019.

[3] C. Langenmair, S. Kassal. Mit AR und VR durch SoftwareCity. "https://www.maibornwolff.de/blog/ar-vr-stadtrundgang-insoftwarecity/", 2017. Zugriff: 16. Juni 2019.

[4] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 181–186, New York, NY, USA, 1991. ACM.

[5] D. Ferrucci, A. Lally, K. Verspoor and E. Nyberg. Unstructured information management architecture (uima) version 1.0. "https://docs.oasis-open.org/uima/v1.0/uima-v1.0.html/", 2009. Zugriff: 29. Juni 2019.

[6] Richard Eckart de Castilho and Iryna Gurevych. A broad-coverage collection of portable nlp components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, pages 1–11, 2014.

[7] Melvil Dewey. *A classification and subject index, for cataloguing and arranging the books and pamphlets of a library.* Brick row book shop, Incorporated, 1876.

[8] Andreas Dieberger and Andrew U. Frank. A city metaphor to support navigation in complex information spaces. *Journal of Visual Languages Computing*, 9(6):597 – 622, 1998.

[9] Kraig Finstad. The usability metric for user experience. *Interacting with Computers*, 22(5):323 – 327, 2010. Modelling user experience - An agenda for research and practice.

[10] Thilo Gotz and Oliver Suhre. Design and implementation of the uima common analysis system. *IBM Systems Journal*, 43(3):476–489, 2004.

[11] Wahed Hemati, Tolga Uslu, and Alexander Mehler. Textimager: a distributed uima-based system for nlp. In *Proceedings of the COLING 2016 System Demonstrations*. Federated Conference on Computer Science and Information Systems, 2016.

[12] Bill Hillier and Julienne Hanson. *The social logic of space*. Cambridge university press, 1989.

[13] Katsushi Ikeuchi, Masao Sakauchi, Hiroshi Kawasaki, and Imari Sato. Constructing virtual cities by using panoramic images. *International Journal of Computer Vision*, 58(3):237–247, 2004.

[14] Rob Ingram, Steve Benford, and John Bowers. Building virtual cities: applying urban planning principles to the design of virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 83–91. ACM, 1996.

[15] JetBrains. IntelliJ IDEA. "https://www.jetbrains.com/idea/", 2019. Zugriff: 26. Juni 2019.

[16] K. Joyce, VRFocus. A Guide to the Oculus Rift's touch controllers. "https://www.vrfocus.com/2017/07/a-guide-to-the-oculus-rifts-touch-controllers/", 2019. Bild überarbeitet von A. Kett, Zugriff: 16. Juni 2019.

[17] Attila Kett, Giuseppe Abrami, Alexander Mehler, and Christian Spiekermann. Resources2City Explorer: A system for generating interactive walkable virtual cities out of file systems. In *Proceedings of the 31st ACM User Interface Software and Technology Symposium*, 2018.

[18] Zhihan Lv, Tengfei Yin, Xiaolei Zhang, Houbing Song, and Ge Chen. Virtual reality smart city based on webvrgis. *IEEE Internet of Things Journal*, 3(6):1015–1024, 2016.

[19] Alexander Mehler, Giuseppe Abrami, Christian Spiekermann, and Matthias Jostock. VAnnotatoR: A framework for generating multimodal hypertexts. In *Proceedings of the 29th ACM Conference on Hypertext and Social Media*, Proceedings of the 29th ACM Conference on Hypertext and Social Media (HT '18), New York, NY, USA, 2018. ACM.

[20] Microsoft. Visual Studio - Tools der Spitzenklasse für jeden Entwickler. "https://visualstudio.microsoft.com/de/", 2019. Zugriff: 26. Juni 2019.

[21] S. Pick, B. Weyers, B. Hentschel, and T. W. Kuhlen. Design and evaluation of data annotation workflows for cave-like virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, 22(4):1452–1461, April 2016.

[22] Robert Remus, Uwe Quasthoff, and Gerhard Heyer. Sentiws-a publicly available german-language resource for sentiment analysis. In *LREC*. Citeseer, 2010.

[23] S. Fortune. Voronoi: Very old, but fast and lightweight C program to compute 2d Voronoi diagrams and Delaunay triangulations. "https://github.com/jceipek/Unity-delaunay/", 2014. Übersetzung in C für Unity von J. Ceipek (2014), Überarbeitet von A. Kett, Zugriff: 14. Juni 2019.

[24] The GIMP Team. GIMP - GNU Image Manipualtion Program. " https://www.gimp.org/", 2019. Zugriff: 26. Juni 2019.

[25] Ubiquitous Knowledge Processing Lab. Ukp. https://dkpro.github.io/, 2014. Zugriff: 29. Juni 2019.

[26] Unity Technologies. Sign into your Unity ID. "https://id.unity.com/", 2019. Zugriff: 6. Juli 2019.

[27] Unity Technologies. Unity download archive. "https://unity3d.com/get-unity/download/archive", 2019. Zugriff: 6. Juli 2019.

[28] Unity Technologies. Unity für alle. "https://unity.com/de/", 2019. Zugriff: 26. Juni 2019.

[29] Tolga Uslu, Alexander Mehler, and Daniel Baumartz. Computing Classifier-based Embeddings with the Help of text2ddc. In *Proceedings of the 20th International Conference on Computational Linguistics and Intelligent Text Processing, (CICLing 2019)*, CICLing 2019, 2019.

[30] Tolga Uslu, Alexander Mehler, and Dirk Meyer. LitViz: Visualizing Literary Data by Means of text2voronoi. In *Proceedings of the Digital Humanities 2018*, DH2018, 2018.

[31] VRQuest. Oculus Touch vs Vive Controller: Der große Vergleich. "https://www.vr-quest.de/oculustouch-vs-vive-controller-der-grosse-vergleich/", 2019. Bild überarbeitet von A. Kett, Zugriff: 13. Juni 2019.