



Bringing Your Hands Into Virtual Reality

Schlussbericht

Studiengang: Informatik
Autoren: Simon Meer
Betreuer: Prof. Urs Künzler
Experten: Yves Petitpierre
Datum: 08.06.2015

Versionen

Version	Datum	Status	Bemerkungen
0.1	14.04.2015	Entwurf	Outline erstellt
0.9	08.06.2015	Entwurf	Draft zum Durchlesen erstellt

Inhaltsverzeichnis

1 Einleitung	1
2 Aufgabenstellung und Zielformulierung	2
2.1 Ursprüngliche Zielformulierung	2
2.2 Änderungen an der Zielformulierung	2
2.3 Hilfsmittel & Hardware	3
3 Design	4
3.1 Systemübersicht	4
3.2 Systemarchitektur	5
3.3 Systemdesign	7
4 Implementation des Indexers	23
4.1 Aufbau	23
4.2 Datenquellen	23
4.3 Datenstruktur	24
4.4 Herausforderungen	26
5 Implementation von IMVR	28
5.1 Unity 5	28
5.2 Aufbau	29
5.3 Interaktionskonzept	29
5.4 Visual Design	33
5.5 Herausforderungen	41
6 Testergebnisse	42
7 Projektmanagement	43
7.1 Soll-Ist	43
7.2 Zeitplan	43
8 Schlussbetrachtung	44
8.1 Ergebnis	44
8.2 Reflexion	44
8.3 Ausblick	44
Selbständigkeitserklärung	45
Glossar	46
Abbildungsverzeichnis	48

1 Einleitung

Eines der faszinierendsten Entwicklungen unserer Zeit ist zweifelsohne das Aufkommen von Virtual Reality (VR) zu einem erschwinglichen Preis. Passend zu dieser Entwicklung finden auch zunehmend neue Peripherie-Geräte ihren Weg in den Markt. Mitunter zu den offensichtlich natürlichssten Methoden des Inputs gehören ganz klar die eigenen zwei Hände. Es scheint deshalb logisch, diese als Peripherie zu verwenden.

In dieser Arbeit geht es darum, ein Head-Mounted Display (HMD), die Oculus Rift, mit einem Hand-Sensor, der Leap Motion, zu koppeln, und eine funktionierende Applikation damit zu entwickeln. Ein gewisses Know-How wurde bereits mit einer vorhergehenden Arbeit aufgebaut.

2 Aufgabenstellung und Zielformulierung

Jedes Projekt startet mit einer Aufgabenstellung. Im Falle dieses Projektes, hat sich diese im Laufe des Projekts ein wenig geändert. Darauf soll ebenfalls eingegangen werden.

2.1 Ursprüngliche Zielformulierung

Es soll eine Applikation namens "IMVR" entwickelt werden, welche Gebrauch von der Oculus Rift macht, um die Bilder- und Musiksammlung des Anwenders ansprechend darzustellen, z.B. in Form eines 3D-Karussells. Die zusätzliche "Tiefe", die durch den Einsatz eines stereoskopischen HMD entsteht, soll dem Anwender helfen, sich in seiner Medienbibliothek schneller zurechtzufinden.

Zusätzlich dazu soll die Leap Motion dazu verwendet werden, um vollständige Handfreiheit zu gewähren: Der Anwender soll komplett ohne Maus und Tastatur imstande sein, sich durch seine Bilder zu navigieren.

Kurz zusammengefasst muss die Applikation:

- Die Bild- und Musikbibliothek des Benutzers in stereoskopischem 3D darstellen.
- Diese freihändig durchsuchbar machen mit Sortier- und evtl. Gruppierfunktion.
- Die Bilder betrachtbar und die Musik abspielbar machen.
- Metainformationen darstellen (z.B. in Form von Diagrammen).

Zusätzlich zur Applikation selbst soll noch ein zusätzliches Tool entwickelt werden, welches im Voraus die Dateien auf dem Host-System indexiert und für die visuelle Applikation bereitstellt.

2.2 Änderungen an der Zielformulierung

Aufgrund der begrenzten Zeit und eines vermehrten Interesse in die Darstellung von Musik, wurde die Zielformulierung während des Projekts leicht abgeändert. Neu wird die Bilderbibliothek des Anwenders komplett ignoriert, dafür das Augenmerk auf seine Musikbibliothek gelegt.

Des Weiteren wurden auch Änderungen am Konzept der Applikation selbst vorgenommen. Dies ist auf die experimentelle Natur des Projekts zurückzuführen, da anfangs nicht klar ist, was funktioniert und was nicht. Konkret wurden zwei Modi eingeführt, und auf eine Sortier- und Gruppierfunktion verzichtet. Diese könnten allerdings in einem nächsten Schritt eingebaut werden.

2.3 Hilfsmittel & Hardware

Verschieden Hilfsmittel werden für die Durchführung des Projektes gebraucht. Seitens Software sind diese:

Tabelle 2.1: Übersicht der eingesetzten Software.

Name	Beschreibung
Unity3D	Spielengine und Entwicklungsumgebung
Visual Studio 2012 & 2013	Entwicklungsumgebung von Microsoft
Blender	Open-Source 3D Modelling-Tool
Krita	Open-Source Grafikbearbeitungs-Tool
GIMP	Open-Source Grafikbearbeitungs-Tool
LaTeX	Hilfsmittel für die Erstellung wissenschaftlicher Dokumente
Microsoft Visio	Tool zur Erstellung von Diagrammen

Im Hardware-Departement wurde folgendes eingesetzt:

Tabelle 2.2: Übersicht der eingesetzten Hardware.

Name	Beschreibung
Oculus Rift	HMD von Oculus VR
Leap Motion	Hand- und Fingererkennungsgerät von Leap Motion, Inc.

Für eine Erklärung der Oculus Rift und der Leap Motion sei an dieser Stelle auf das Pflichtenheft verwiesen.

3 Design

Bei der Entwicklung wurden diverse Design-Entscheidungen gefällt, welche zum Teil bereits in einem Vorprojekt analysiert wurden. Diese sollen in diesem Kapitel aufgelistet und erläutert werden.

3.1 Systemübersicht

Um die Applikation zu bedienen, setzt der Anwender die Oculus Rift auf und bewegt sich im von der mitgelieferten Kamera erkennbaren Bereich.

Die Leap Motion wird prinzipiell so verwendet, wie von der Herstellerfirma vorgesehen. Das heisst, diese wird mit dem offiziellen Aufsatz [1] an der Oculus Rift befestigt, und deckt so den frontalen Sichtbereich des Anwenders ab. Dies lässt sich gut auf Abbildung 3.1 erkennen.

Ebenfalls erkennbar ist, dass beide Geräte per USB mit dem Host-Computer verbunden sind und Daten an die jeweiligen Services schicken. Diese Services werden durch die in Unity verwendeten Plugins angesteuert, und die ausgewerteten Daten in IMVR verwendet.

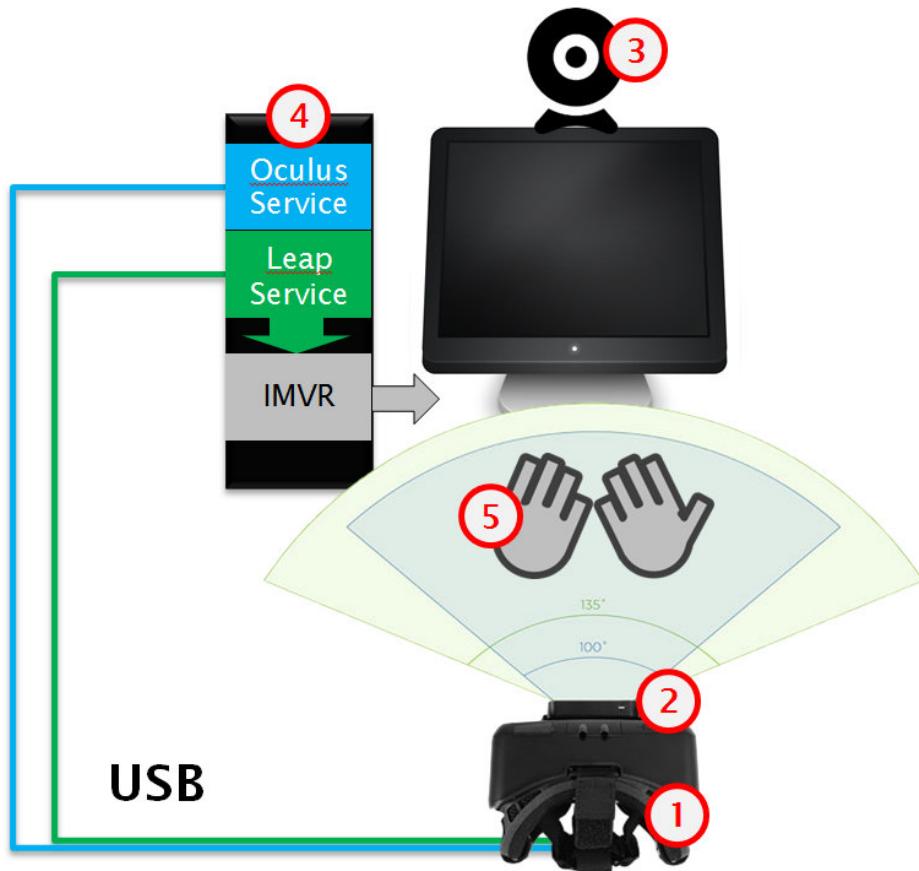


Abbildung 3.1: Eine Übersicht der Technologien und wie sie verbunden sind.

Nr.	Komponente	Beschreibung
1.	Oculus Rift DK2	HMD für den grafischen Output.
2.	Leap Motion	Gerät, welches Hände erkennt und ihre Koordinaten an den Computer sendet.
3.	Oculus Rift Kamera	Kamera, welche seit dem DK2 für das örtliche Tracking zuständig ist.
4.	Computer	Host-System für IMVR.
5.	Benutzer	Benutzer, der die Oculus Rift trägt und mit seinen Händen das Programm steuert.

3.2 Systemarchitektur

Zuerst ist es wichtig, zu verstehen, wie die Applikation grob aufgebaut ist. In Abbildung 3.2 wird dies in zwei Schritten illustriert.

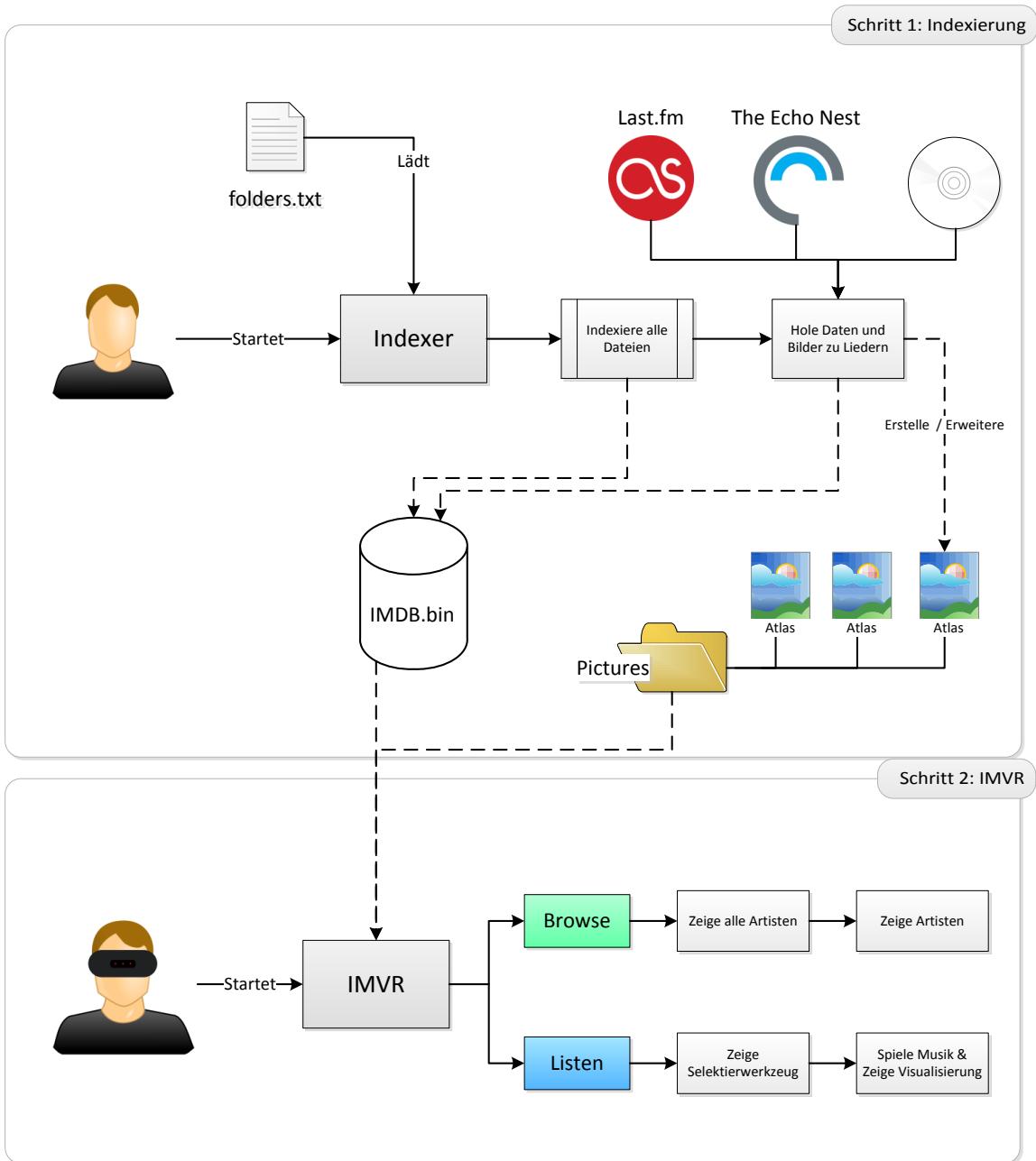


Abbildung 3.2: Grober Überblick des Programmablaufs

Man erkennt wie der Benutzer zuerst ausserhalb der Unity-Applikation den Indexer startet und durchlaufen lässt. Dieser durchläuft alle Ordner, die er in einer *folders.txt* findet und schreibt diese in die Datenbank. Im nächsten Schritt werden aus diversen Quellen weitere Daten abgerufen.

Was ebenfalls auf der Grafik zu sehen ist, sind die Atlassse. Um Ressourcen zu sparen (siehe Kapitel 5.5.1), werden alle Bilder in sogenannten Atlassen, sprich Bildersammlungen, gespeichert. Die Bilder, von denen hier die Rede ist, sind Fotos der Artisten und das Artwork der indexierten Alben.

Sobald die Datenbank im ersten Schritt erstellt wurde, setzt der Anwender, wie in Kapitel 3.1 beschrieben, seine Oculus Rift mit dem Leap Motion Aufsatz auf und startet IMVR. Er erhält dann die Auswahl, welchen Modus er beschreiten will und je nach Wahl entsprechende weitere Optionen.

3.3 Systemdesign

Jetzt, wo das grobe Zusammenspiel der Elemente im System klar geworden ist, soll ein bisschen näher auf die Unterelemente eingegangen werden. Aufgrund der Brückenfunktion und deshalb der globalen Relevanz, wird zuerst ein Blick auf die Klassenstruktur der Datenbank geworfen. Danach wird der Aufbau des Indexers untersucht, und zuletzt schliesslich die eigentliche Applikation.

Zu beachten: Die meisten Klassen in den folgenden UML-Diagrammen sind Unity-Komponenten und erben somit von der Klasse MonoBehaviour. Um die Diagramme lesbar zu halten, wurde diese Beziehung oft ignoriert. Weiterhin wurden manche C#-Properties als "get_Property" und "set_Property" ausgeschrieben, wo zusätzliche Logik vorhanden ist.

3.3.1 Commons (Datenbank)

Um Daten zwischen den zwei Programmteilen zu transportieren, wurde noch ein weiteres Projekt erstellt mit einer eigenen Datenstruktur, welche in Abbildung 3.3 ersichtlich ist.

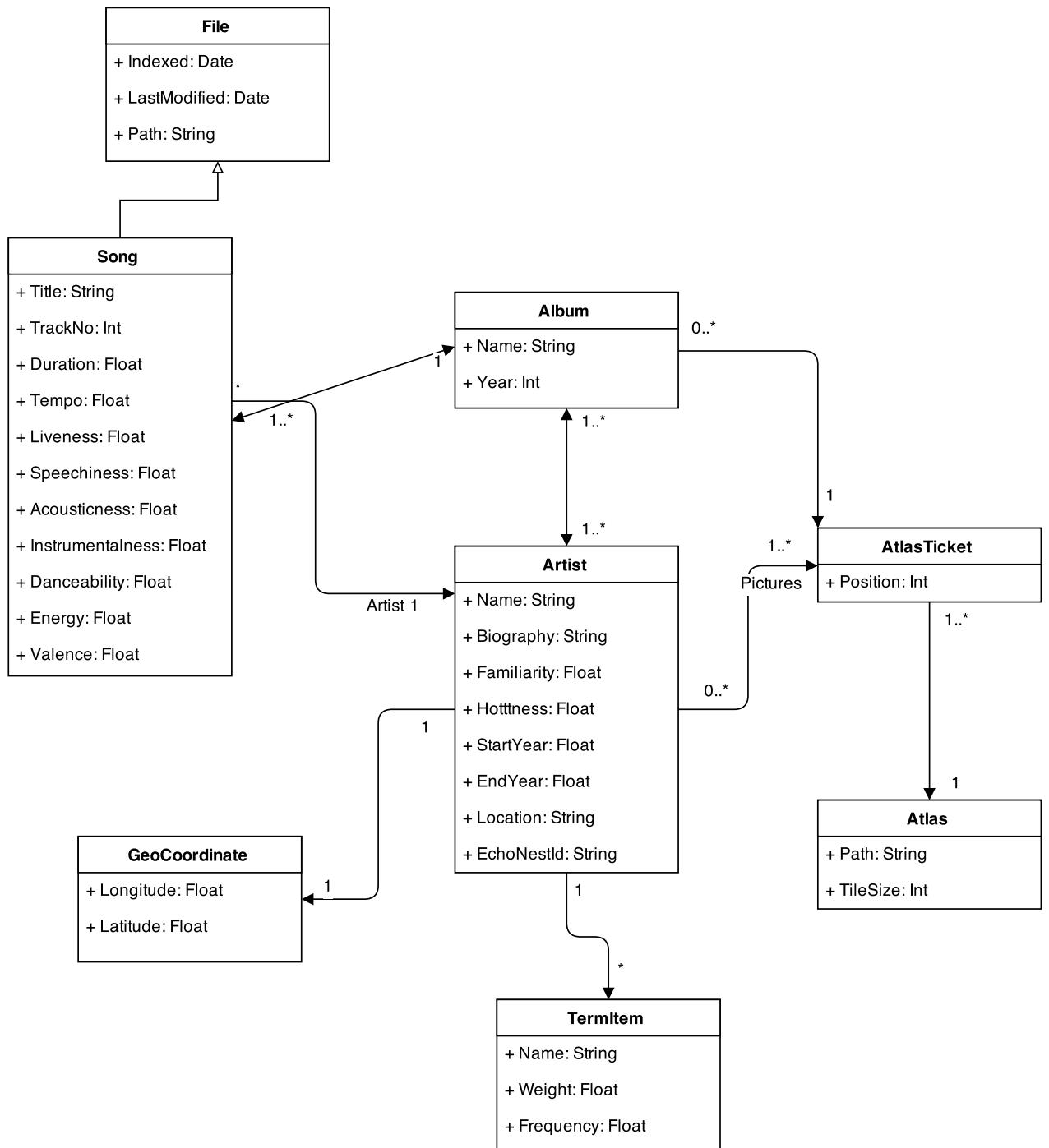


Abbildung 3.3: Klassendiagramm für die Klassen im IMVR.Commons Package

Klasse	Beschreibung
Artist	Klasse für einen einzelnen Artisten mit Biografie und ein paar Kennwerten sowie Koordinaten und Aktivitätsdaten.
Album	Jeder Artist hat ein oder mehrere Alben, an denen er beteiligt ist. Das Album selbst hat nur einen Namen und ein Jahr und kann zu mehreren Künstlern gehören.
Song	Repräsentiert einen Song und gehört genau einem Künstler. Jeder Song hat einen Pfad und ist somit von einer reellen Datei gestützt. Ein Song erhält zusätzlich verschiedene Kennwerte, die von Services heruntergeladen werden.
TermItem	Ein Begriff (z.B. Genre), der einem Künstler mit einem Gewicht und einer Frequenz zugeordnet wird. Könnte noch weiter normalisiert werden, wurde aber der Einfachheit halber so gelassen.
AtlasTicket	Repräsentiert eine Position bzw. ein Bild in einem Atlas. Mit der Position ist gemeint, an welcher Stelle das Bild im Atlas erscheint.
IMDB	Zentraler Zugangsknoten zu den Daten. Hält Referenzen auf die Artisten, Songs und Atlassse.

Tabelle 3.1: Erklärung der wichtigsten allgemeinen Klassen

Es handelt sich um ein bewusst sehr minimalistisch gehaltenes Schema, um eine Musiksammlung darzustellen. Zu den Problemen, die nicht abgedeckt werden gehören zum Beispiel:

- Jede Ausgabe eines Liedes gilt als ein separates Lied
- Für kombinierte Artisten (z.B. A feat. B) wird jeweils ein neuer Artist erstellt
- Mehrere Aktivitätsperioden eines Künstlers können nicht abgebildet werden

Es gäbe keine Grenzen, wenn man eine perfekte Struktur erreichen wollte, und das liegt ausserhalb des Bereichs dieses Projektes.

3.3.2 Indexer

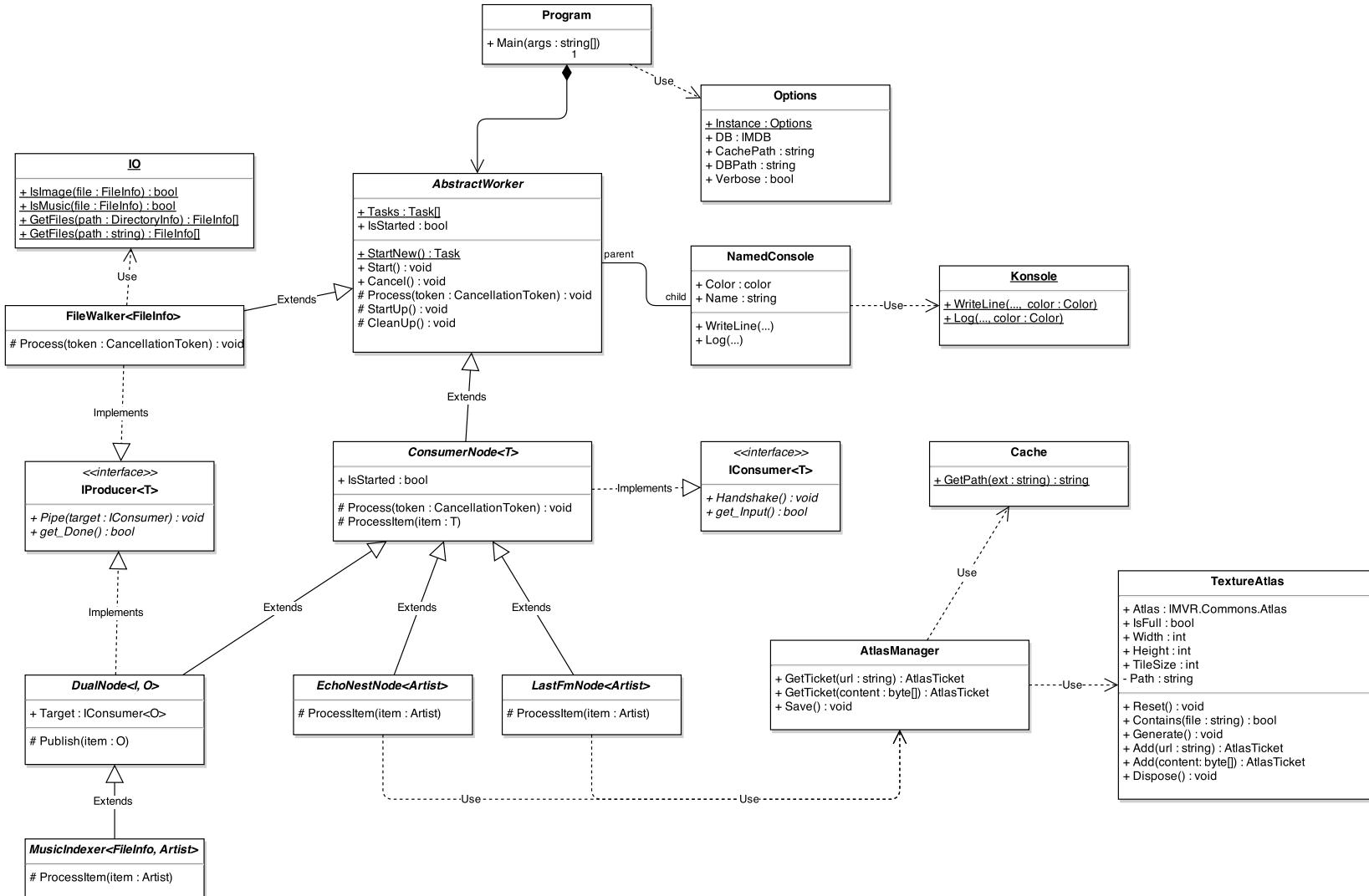


Abbildung 3.4: Klassendiagramm des Indexers

Der Indexer basiert auf Nodes. Das Konzept wird dabei mithilfe eines Producer-Consumer Patterns abgebildet.

Alle Nodes im System führen eine Aufgabe auf einem oder mehreren Threads durch. Wenn diese einen Input haben, implementieren sie IConsumer oder eine der abstrakten Implementationen davon, und wenn diese einen Output haben, implementieren sie IProducer, um die Daten an den nächsten Node weiterzugeben.

Klasse	Beschreibung
Options	Singleton-Klasse, welche mit den Command-Args gebildet wird und die Konfigurationen beinhaltet.
IProducer	Ein Produzent von Daten (→ Output-Knoten)
IConsumer	Ein Konsument von Daten (→ Input-Knoten)
AbstractWorker	Jede Node erbt von dieser Klasse. Sie sorgt dafür, dass die Nodes auf eine einheitliche Weise initialisiert, durchgeführt und aufgeräumt werden können.
ConsumerNode	Hilfsklasse, welche IConsumer implementiert und mithilfe einer Methode ProcessItem den Einsatz erleichtert.
DualNode	Hat die gleiche Funktion wie eine ConsumerNode, aber implementiert zusätzlich IProducer und bietet mit Publish ebenfalls eine Hilfsmethode an.
EchoNestNode	Holt Daten aus der The Echo Nest API und schreibt diese in die Datenbank.
LastFmNode	Holt Daten aus der Last.fm API und schreibt diese in die Datenbank.
MusicIndexer	Konsumiert Files und extrahiert daraus Artisten, Alben und Songs.
FileWalker	Durchläuft die Medienordner und findet Musik und Bilder.

Tabelle 3.2: Erklärung der wichtigsten Klassen des Indexers

3.3.3 IMVR

Nun, da das Design des Indexers festgelegt ist, bedarf auch die Hauptapplikation einer Erklärung. Um das Design besser verständlich zu machen, wurde hierbei das Schema in verschiedene, logische Teilbereiche gegliedert, die auch den Namespaces des Projekts entsprechen.

Es wird empfohlen, diese Schemas zusammen mit den Erklärungen in Kapitel 5 zu verwenden.

Interface

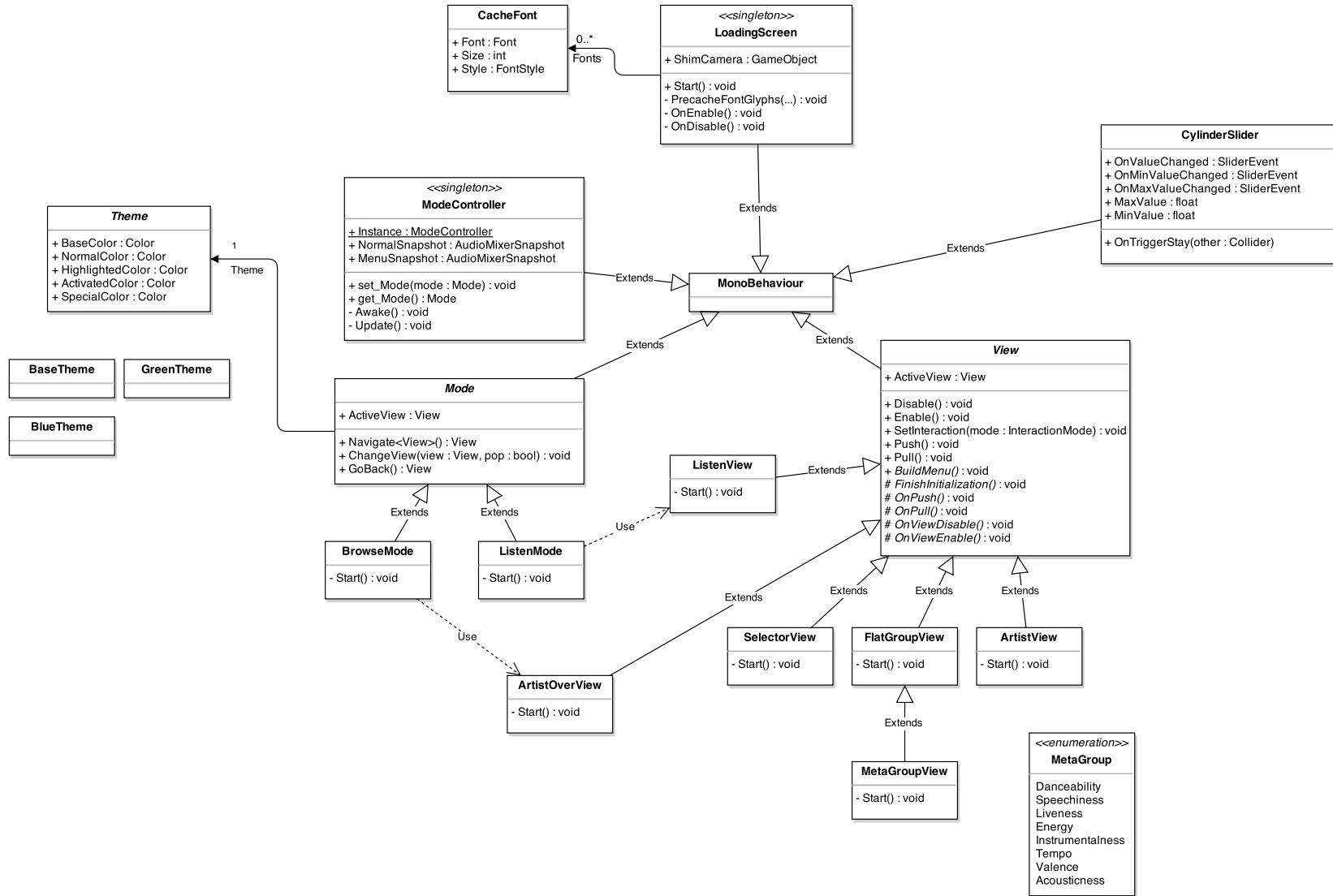


Abbildung 3.5: Klassendiagramm des Interfaces der Applikation (View-Behavior)

Das Interface von IMVR basiert auf sogenannten *Views*. Zu jedem Zeitpunkt der Applikation ist irgendeine View aktiv, die verantwortlich für die momentane Darstellung der Szene ist.

Die Views gehören und werden kontrolliert vom momentan aktiven *Mode*, wovon zwei existieren: der *BrowseMode* und der *ListenMode*. Der aktuelle Modus wird verwaltet durch die Klasse *ModeController*. Das Wechseln des Modus ist Aufgabe des Ring-Panels und wird in einem separaten Kapitel behandelt.

Ein weiteres relevantes Detail ist das *Theme*, welches den Modi zugeordnet wird. Dieses Theme enthält eine Anzahl von Farben, die später von anderen GUI-Elementen verwendet werden und für eine leichtere Unterscheidung des momentanen Modus sorgen sollen.

Klasse	Beschreibung
ModeController	Führt Buch über den momentan aktiven Modus.
Mode	Verfügt über einen eigenen Stack von Views, durch die der Anwender navigiert.
BrowseMode	Modus, in dem der Anwender gezielt durch seine Musiksammlung navigiert, indem ihm eine Übersicht aller Künstler angezeigt wird. Beginnt mit der View <i>ArtistOverView</i> .
ListenMode	Modus, in dem der Anwender nur die Art der Musik angibt, die er hören will. Die Applikation übernimmt den Rest. Beginnt mit der View <i>SelectorView</i> .
Theme	Eine Sammlung von Farben, die einem Modus zugeordnet wird.
View	Eine "Ansicht", die aktiviert und deaktiviert sowie nach vorne und nach hinten geschoben werden kann.
ArtistOverView	Stellt einen gruppierten Zylinder mit den Künstlern in der Musiksammlung dar.
ArtistView	Stellt einen einzelnen Artisten und seine Alben dar.
SelectorView	Stellt ein Auswahlwerkzeug dar, mit dem der Anwender eine Musikart selektieren kann.
ListView	Verwaltet die visuellen Effekte beim Hören von Musik im Listen-Mode.

Tabelle 3.3: Erklärung der wichtigsten Interface-Klassen

Music Arm

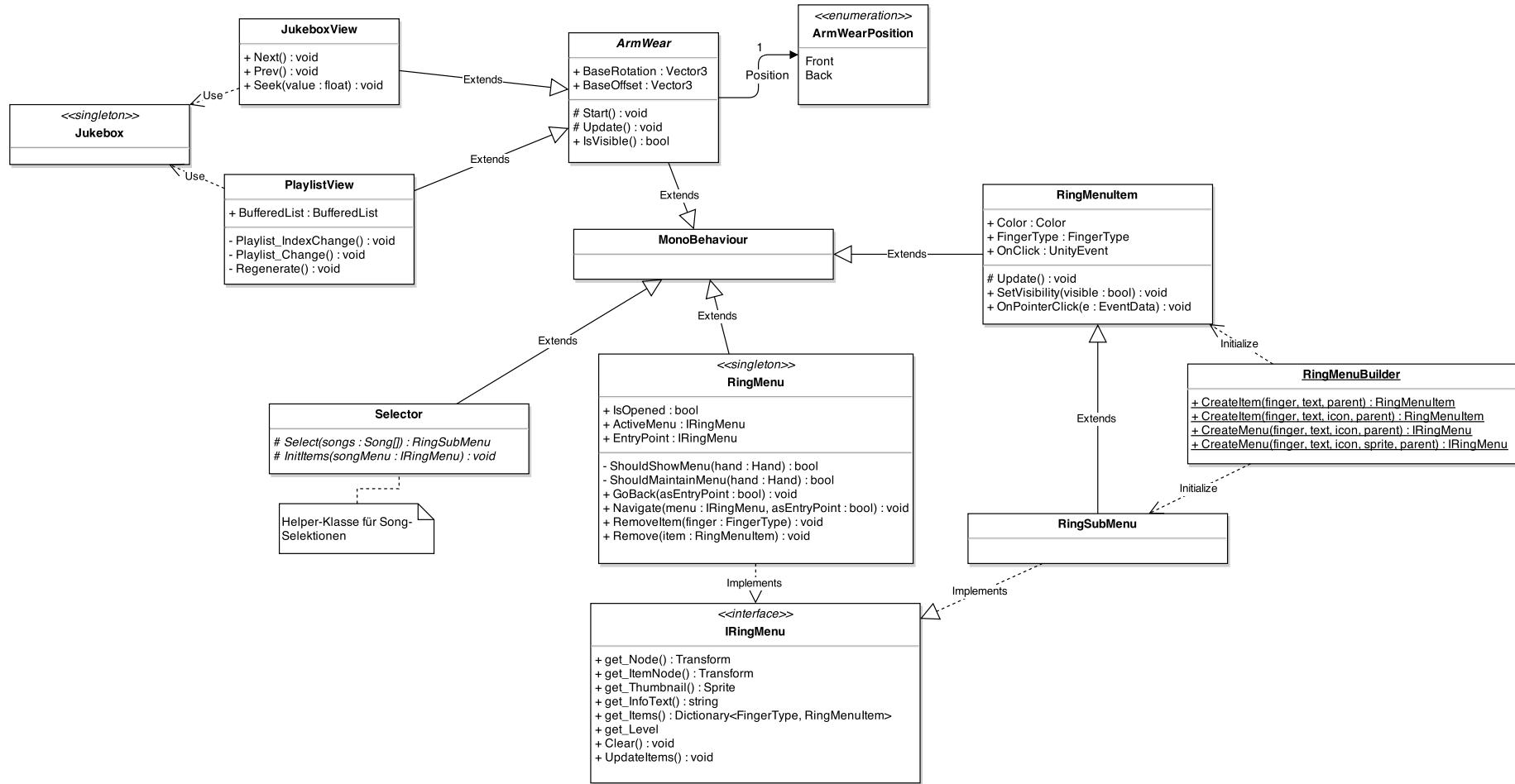


Abbildung 3.6: Klassendiagramm des Music Arms

Das eigentliche Menü, der *Music Arm*, befindet sich im IMVR.Interface.MusicArm Namespace. Er verwaltet die momentan abzuspielende Musik und die selektierten Lieder.

Beim Design der Klassen war ursprünglich ein viel allgemeinerer Ansatz für den Music Arm geplant. Konkret gesagt, wurden die Klassen so gestaltet, dass das Ring-Menü, welches zum Einsatz kommt, im Editor für *irgendeine* Aufgabe konfiguriert werden kann.

In einem gewissen Stadium des Projekts wurde jedoch der Music Arm konzipiert und das Ring-Menü stark umstrukturiert. Neu erhält jede View beim Laden die Möglichkeit, mithilfe der Helper-Klasse RingMenuBuilder ein eigenes Menü zu gestalten, wobei die Belegung des Zeige-, Mittel- und Ringfingers vorgegeben ist.

Klasse	Beschreibung
RingMenu	Die zentrale Klasse des Music Arms und gleichzeitig auch die Root-Ebene des Menüs. Existiert nur einmal, verwaltet, welche Stufe momentan angezeigt wird, und sorgt für das Anzeigen und Verstecken des Menüs.
RingMenuItem	Ein einzelner Menüeintrag im Menü. Platziert sich in den entsprechenden Finger wenn sichtbar und hat einen Event-Handler für Click-Events (in diesem Fall Auswahl des Menüs).
IRingMenu	Interface für Menüs mit Einträgen.
RingSubMenu	Implementiert IRingMenu und funktioniert wie RingMenu, nur ist es selbst auch ein Eintrag in einem anderen IRingMenu.
RingMenuBuilder	Hilft beim dynamischen Erstellen von neuen Menüeinträgen.
ArmWear	Beschreibt eine Komponente, die am Arm "getragen" werden kann.
JukeboxView	Zeigt das momentane Lied in der Jukebox an und wird auf der Frontseite des Arms getragen.
PlaylistView	Zeigt die momentan Playlist an und befindet sich auf der Rückseite des Arms.
Selector	Hilft Komponenten, die im Ring-Menü eine Songselektion machen wollen, dabei.

Tabelle 3.4: Erklärung der wichtigsten Music-Arm-Klassen

Ring Panel

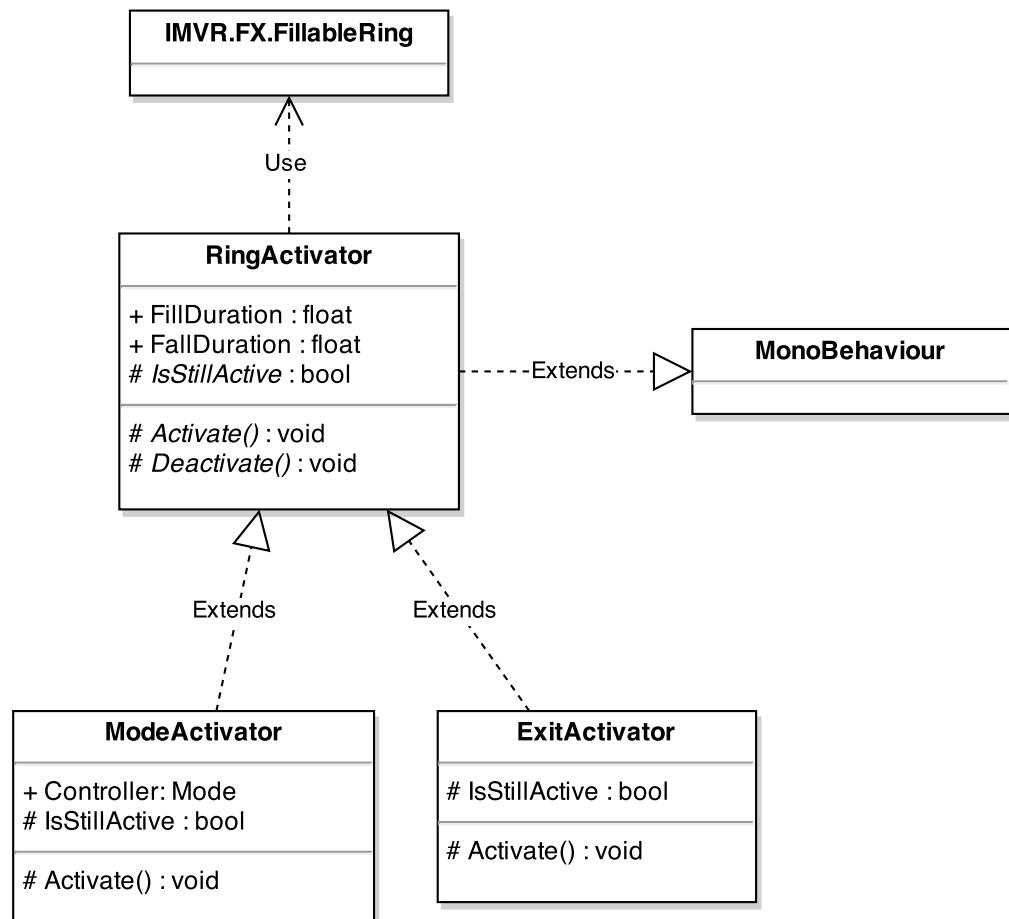


Abbildung 3.7: Klassendiagramm des Ring Panels, womit der Anwender Modi auswählt

Das Ring-Panel befindet sich unter den Füßen des Anwenders und wird mit dem Blick bedient. In diesem Namespace befinden sich nur die Klassen, die für die Logik zuständig sind - der visuelle Teil ist bei den Effekten (IMVR.FX) zu finden.

Klasse	Beschreibung
RingActivator	Abstrakte Klasse, die visuell von einem sich füllenden Ring-Mesh dargestellt wird, und beim anhaltenden Blick aktiviert wird.
ModeActivator	Eine Unterklasse des RingActivators, die nur dazu zuständig ist, einen Modus zu aktivieren (siehe <i>IMVR.Interface</i>).
ExitActivator	Ein weiterer Activator, der beim Aktivieren die Applikation beendet.

Tabelle 3.5: Erklärung der Klassen des Ring-Panels

Daten

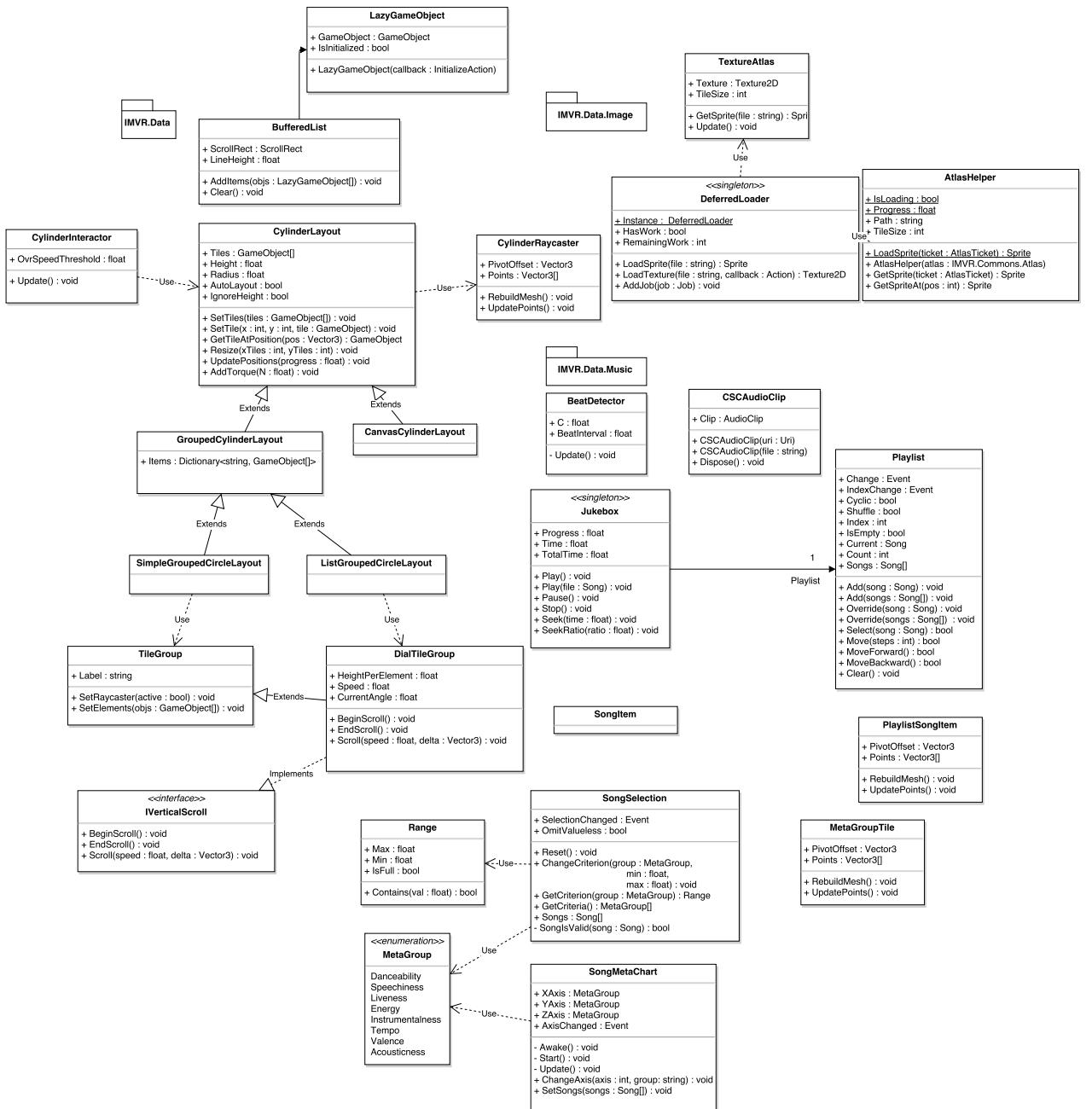


Abbildung 3.8: Klassendiagramm des Teils, der die Daten der Datenbank verwendet

Die Klassen, die sich mit den Daten aus der Datenbank bzw. der Darstellung dieser beschäftigen, befinden sich im Namespace `IMVR.Data`.

Der Namespace ist gegliedert in `Data`, `Data.Music` und `Data.Image`. Ersterer befasst sich mit der generellen Darstellung der Daten, der zweite kümmert sich um das Arbeiten mit Musik, und `Data.Image` schliesslich um Aufgaben mit Bildern.

Ursprünglich war geplant, einen Modus nur für die Bilderbibliothek des Anwenders zu entwickeln, und die dabei verwendeten Klassen in den entsprechenden Namespaces zu platzieren. Eine Grosszahl

dieser Klassen wurde jedoch zusammen mit dem Modus selbst gestrichen. Übrig geblieben sind die Klassen, die noch Verwendung bei der Musik finden (z.B. für das Darstellen von Artwork).

Die allgemeinen Klassen in IMVR.Data enthalten designmässig zum Teil ebenfalls Überbleibsel aus dem Entwicklungsstadium, wo ein separater Bildermodus geplant war. Hauptsächlich zeigt sich das beispielsweise beim CylinderRaycaster. Dessen Aufgabe war ursprünglich, die grosse Anzahl Bilder handhabbar zu machen, weil einzelnes Raycasting zu viel Zeit beanspruchen würde.

Klasse	Beschreibung
CylinderLayout	Komponente für die allgemeine Darstellung von GameObjects in einem Zylinder-Layout. Kann per CylinderInteractor rotiert werden und verfügt über einen optimierten Raycaster.
GroupedCylinderLayout	Ein spezielles CylinderLayout, welches die Elemente gruppiert mit jeweils einem Label darstellt.
SimpleGroupedCylinderLayout	Ein gruppiertes CylinderLayout, welches auf normale GameObjects spezialisiert ist.
ListGroupedCylinderLayout	Ein gruppiertes CylinderLayout, welches auf Listen von Canvas-GameObjects spezialisiert ist.
CylinderInteractor	Kümmert sich um die User-Interaktion mit dem Layout. Hilft beim Rotieren und vertikalen Scrollen.
CylinderRaycaster	Sorgt für ein effizientes Raycasting der Elemente im Layout. Wird wichtig, wenn über 100 interaktive Elemente im Zylinder verteilt sind.
BufferedList	Hilft bei der effizienten Darstellung der Playlist (siehe Kapitel 5.5.4).

Tabelle 3.6: Erklärung der wichtigsten Klassen von IMVR.Data

Der Teil, der für die Bilder zuständig ist, enthält drei Klassen: eine Helper-Klasse, die auf einer hohen Ebene für das einfache Laden von Atlassen aus IMVR.Commons sorgt, eine intelligente Klasse zum Laden von Bildern und eine tiefe Klasse, welche direkt mit den Daten der Atlassse arbeitet.

Klasse	Beschreibung
AtlasHelper	Hilft beim Laden von Atlassen aus der Datenbank, indem er logische Methoden fürs Laden liefert. Gibt zudem Auskunft über die aktiven Ladevorgänge.
DeferredLoader	Lädt und verwaltet Atlassse vom Filesystem, und liefert Texturen und Sprites für den Zugriff darauf.
TextureAtlas	Ein einzelner Texturenatlas auf dem Filesystem mit einer bestimmten Kachelgrösse.

Tabelle 3.7: Erklärung der Klassen von IMVR.Data.Image

Im Namespace für die Musik gibt es grob gesagt zwei Arten von Klassen: die Jukebox und die Klassen zur Darstellung von Musikdaten.

Klasse	Beschreibung
Jukebox	Kontrolliert die momentane Wiedergabe und reagiert auf Events der Playlist.
Playlist	Eine Liste von Songs, die für die Wiedergabe vorgesehen sind. Hat eine momentane Selektion, die geändert werden kann. Sender Events beim Ändern der Liste bzw. der momentanen Selektion.
SongItem	Zeigt ein Lied an.
PlaylistSongItem	Zeigt ein Lied an, optimiert für die Anzeige in PlaylistView.
SongMetaChart	Steuert die FX-Klasse PointChart um damit Songmetriken dreidimensional darzustellen (siehe Kapitel 5.4.2)
SongSelection	Enthält eine Ansicht auf die Musikdaten, die durch Auswahlbereiche in verschiedenen MetaGroups eingeschränkt ist. Jede MetaGroup ist ein Kriterium mit einem Wert im Bereich [0..1].

Tabelle 3.8: Erklärung der wichtigsten Klassen von IMVR.Data.Music

Effekte

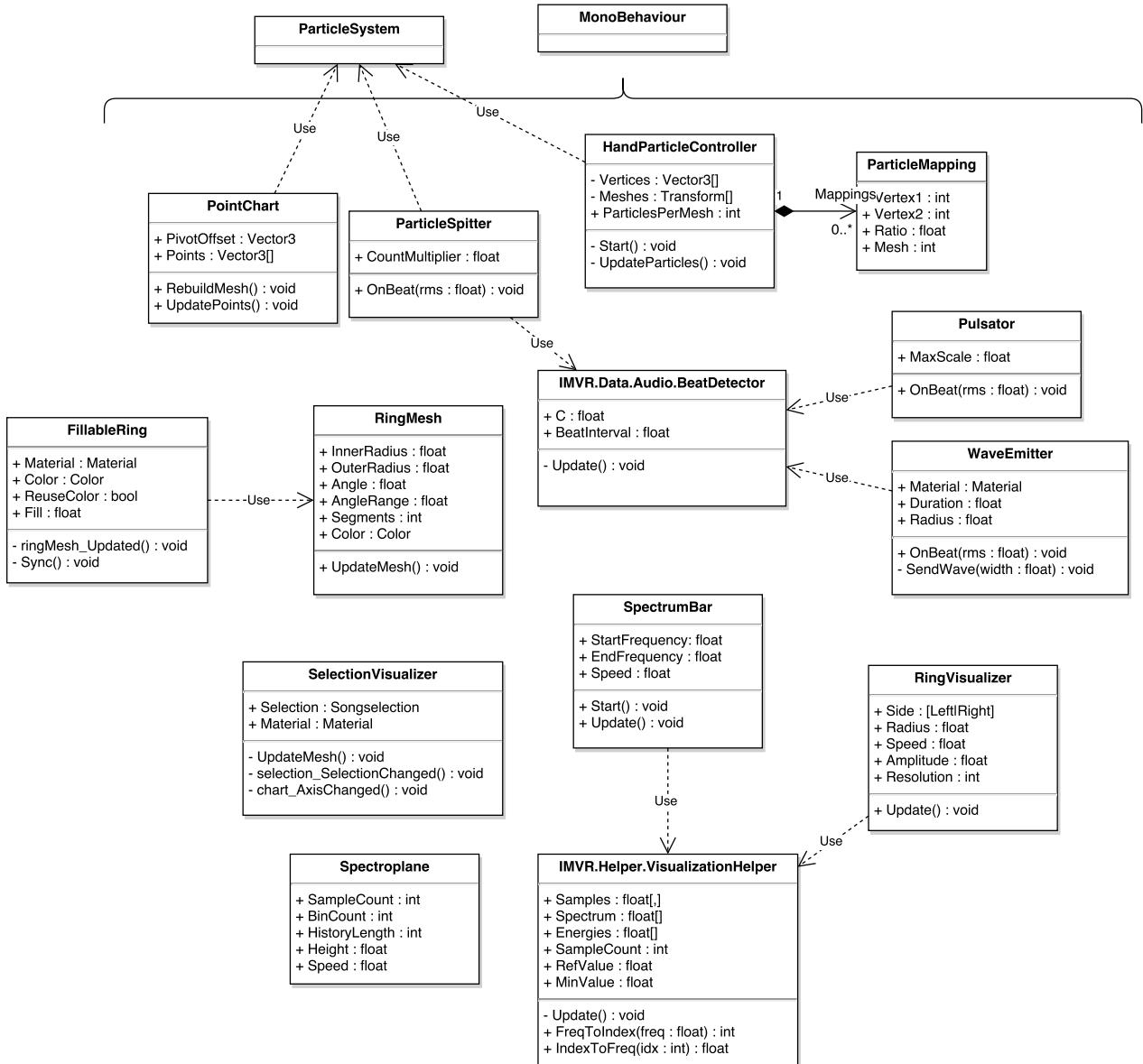


Abbildung 3.9: Klassendiagramm der Effekte und prozeduralen Meshes

Der visuelle Aspekt der Applikation ist im Namespace *IMVR.FX* aufbewahrt. Hier befinden sich die prozeduralen Meshes und die Partikelsysteme. Was man hier nicht finden wird, sind die visuellen Elemente, die direkt für das Interface verwendet werden. Die Klassen in *IMVR.FX* haben eine unterstützende Funktion.

Auch in diesem Namespace gibt es Klassen in leicht unterschiedlichen Anwendungsszenarios. In einem Szenario geht es darum, Musik zu visualisieren wie es z.B. ein Musik-Player tut.

Klasse	Beschreibung
VisualizationHelper	Steht im Mittelpunkt der Musik-Visualisierung. Verwaltet die Sample- und Spektrumdaten, die in Echtzeit aus der Musik gewonnen werden.
SpectrumBar	Ein Mesh, welches je nach Stärke des Signals in einem bestimmten Frequenzbereich, höher steigt bzw. tiefer sinkt.
RingVisualizer	Eine kreisförmige, kontinuierliche Linie, welche die Lautstärke eines Kanals (links oder rechts) darstellt.
BeatDetector	Klasse zur Erkennung von Beats in der wiedergegebenen Musik.
Pulsator	Komponente, welche ein Objekt im Takt der Musik pulsieren lässt.
WaveEmitter	Sendet im Takt der Musik Wellen in entsprechender Stärke aus (siehe Kapitel 5.4.4)
ParticleSpitter	Wirft im Takt der Musik Partikel in die Szene.

Tabelle 3.9: Erklärung der FX-Klassen, die sich mit der Musik-Visualisierung befassen

Neben diesen Visualisierungsklassen gibt es auch Komponenten, die sich mit Effekten befassen, die nicht von der momentanen Musik abhängen.

Klasse	Beschreibung
---------------	---------------------

Tabelle 3.10: Erklärung der FX-Klassen, die sich mit der Musik-Visualisierung befassen

Gesten

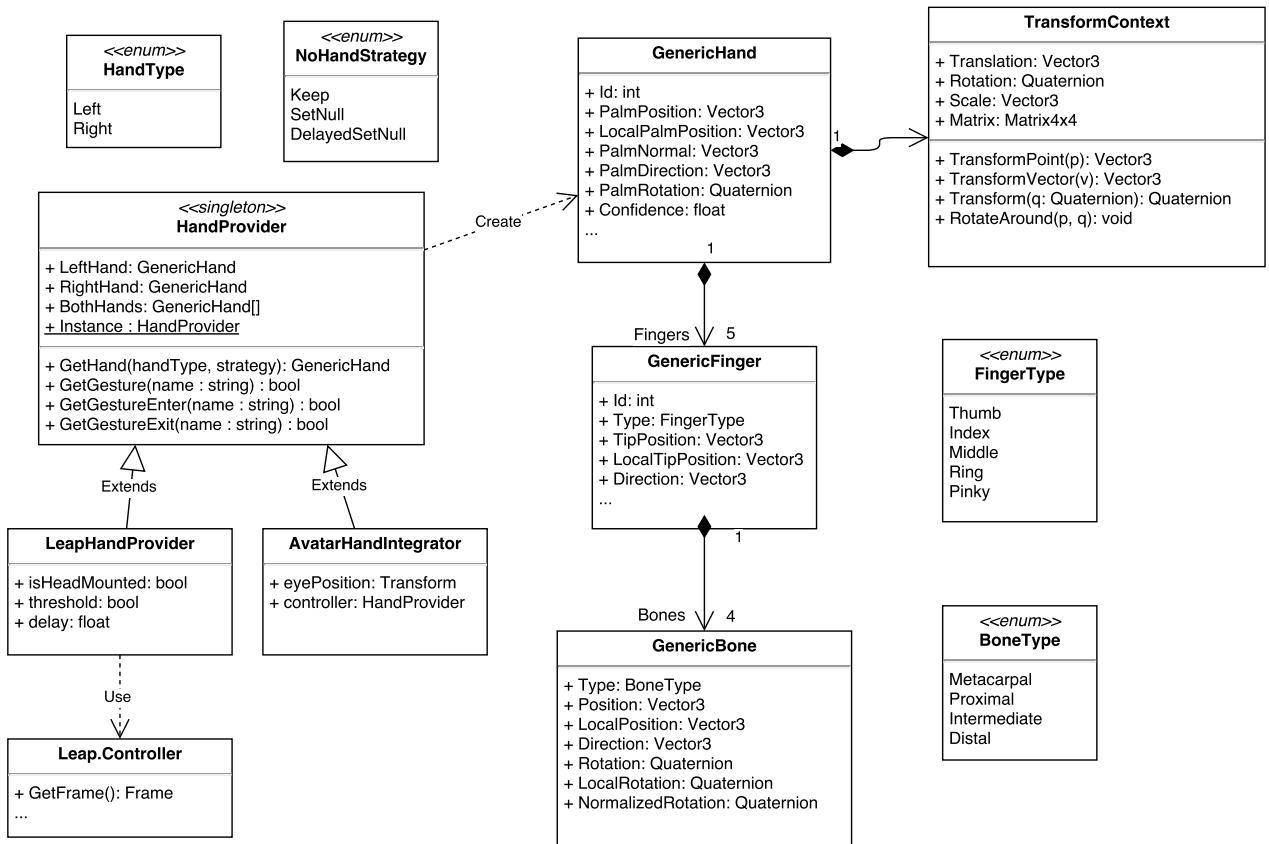


Abbildung 3.10: Klassendiagramm des Abstraktionslayers für die Leap Motion

4 Implementation des Indexers

Ein Teil der Arbeit war es, einen Indexer zu implementieren, der die Dateien des Benutzers durchläuft und - wie es der Name vermuten lässt - interessante Dateien indexiert. In diesem Kapitel soll auf den Aufbau und die Implementationsdetails eingegangen werden.

4.1 Aufbau

Der Indexer sowie die Datenstruktur und alle anderen Tools, die eine ergänzende Funktion zur Hauptapplikation haben, wurden in einer separaten Visual Studio Solution zusammengefasst. In dieser befinden sich vier Projekte:

Tabelle 4.1: Die Projekte in Auxiliary Tools

Name	Beschreibung
IMVR.Commons	DLL-Projekt, welches die Klassen enthält, die zwischen Applikation und Indexer geteilt werden.
IMVR.Commons.Tests	Testprojekt mit Unit-Tests um die Integrität der Daten sicherzustellen.
IMVR.Indexer	Konsolenprojekt, welches die Musikdateien auf dem Host-System indexiert.
IMVR.SpeechServer	Konsolenprojekt, welches in der Vorarbeit verwendet wurde und in dieser Arbeit keine Verwendung fand.

Der Indexer (IMVR.Indexer) ist als parallelisiertes, knotenbasiertes System konzipiert worden. Die Parallelität wurde deshalb gewählt, weil es beim Einholen der verschiedenen Datenquellen teils zu Wartezeiten kommt, die gut anders genutzt werden können. Dieser Faktor kam besonders ins Spiel als noch zusätzlich zu den Musikdaten auch Bilderdaten gesammelt worden sind.

Ein weiterer Grund für die Parallelisierung ist, dass als mögliches Feature eine real-time Indexierung vorgesehen war, die es letztendlich allerdings nicht in das fertige Programm schaffte. Weitere Informationen zu diesem Thema werden in Abschnitt 4.4 gegeben.

4.2 Datenquellen

Da IMVR zum Ziel hat, die Musik des Benutzers in verschiedenen Formen und Farben darzustellen, werden Daten von diversen Quellen benötigt. Wir leben in einer wundervollen Zeit in Sachen

Datenvielfalt: Es gibt viele Online-Services, welche zu einem Grossteil gratis sind, von denen man diverse Daten erhalten kann.

Es folgt eine kurze Zusammenstellung von untersuchten Datenquellen.

Tabelle 4.2: Eine Übersicht von verfügbaren Online-Datenquellen.

Name	Daten	API-Limite
Last.fm	Bilder, Meta-Daten	5 Request / Sekunde
The Echo Nest	Bilder, Meta-Daten, Analyse-Daten	120 Requests / Minute
Spotify	Bilder, Meta-Daten, Playlisten, Streams	?
Amazon	Bilder, Beschreibungen	1 Request / Sekunde
Gracenote	Bilder, Fingerprinting, Meta-Daten	~1000 Requests / Tag
MusicBrainz	Bilder, Meta-Daten	~1 Request / Sekunde

Im Falle von IMVR kommen die Daten grundsätzlich von drei verschiedenen Quellen:

- ID3 Tags der Musik-Dateien
- The Echo Nest
- Last.fm

Diese wurden gewählt aufgrund der durchsichtigen API-Limite und den verfügbaren Daten. The Echo Nest aggregiert zudem die Daten von anderen Seiten wie Spotify und MusicBrainz, und enthält wertvolle Analyse-Daten, die in eine zentrale Position in IMVR haben.

In einem ersten Schritt werden grundlegende Daten wie der Titel des Liedes, der Name des Artisten, usw. aus der Datei selbst entnommen. Wenn möglich wird auch gleich geprüft, ob ein Album-Cover hinterlegt ist.

In einem zweiten Schritt wird über eine .NET Bibliothek [2], welche im Rahmen des Projektes geforkt und erweitert wurde, zur API von The Echo Nest verbunden und diverse Meta-Daten zur Musik heruntergeladen.

Was bei der Originalimplementation leider fehlt, sind neuere Features wie Genres und Ortsdaten, sowie ein intelligenter Bremsalgorithmus, der dafür sorgt, dass die Datenlimite eingehalten wird. Deshalb wurde ein Fork erstellt, der diese Daten und Funktionalitäten ergänzt¹.

4.3 Datenstruktur

Für die Abspeicherung der Daten wurde zuerst ein Ansatz gewählt, der auf einer SQLite Datenbank basierte. Diese wurde in einem Prototyp auch erfolgreich implementiert. Es stellte sich jedoch heraus, dass diese Abhängigkeit das Programm unnötig verkomplizieren würde und eine simple In-Memory Datenstruktur völlig ausreicht.

¹https://github.com/EusthEnoptEron/echonest-sharp/tree/additional_apis

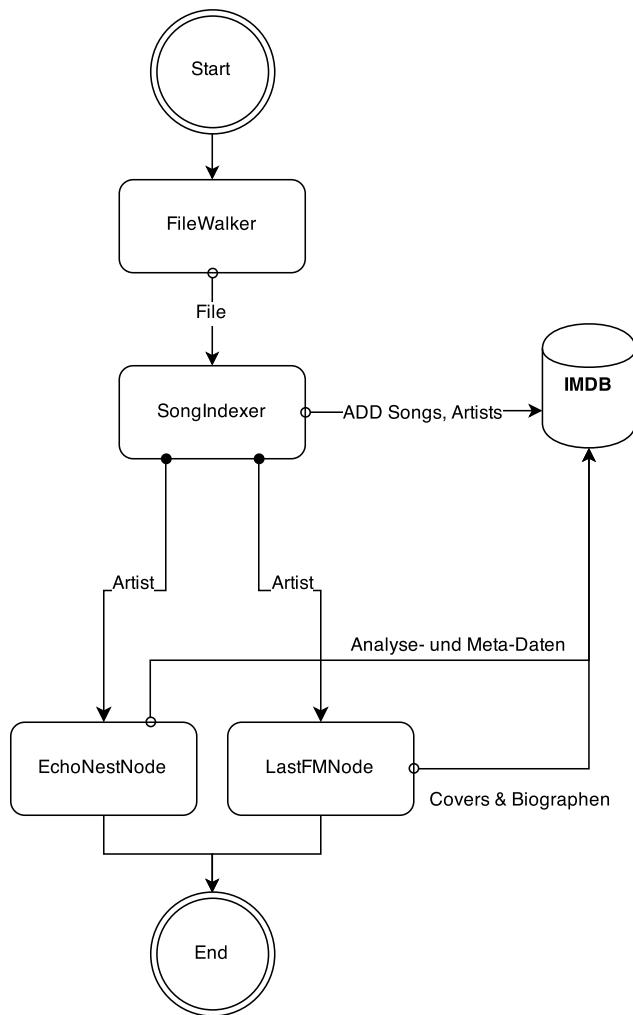


Abbildung 4.1: Der Datenfluss, den die Files beim Indexieren nehmen.

Das finale Produkt verwendet also eine simple, objektorientiert Datenstruktur, die serialisiert und so zwischen den zwei Projekten (Indexer und IMVR) geteilt werden kann. Das Schema ist in Abbildung 4.2 zu sehen. Damit beide Projekte Zugriff auf die genutzten Klassen haben, wurde das Schema in eine separate DLL ausgelagert, die ebenfalls als Abhängigkeit in beiden Projekten referenziert wird.

Anzumerken ist, dass mehrere Einstiegspunkte auf die Daten existieren und so eine gewisse Redundanz geschaffen wird. Diese Redundanz ist hilfreich, weil in IMVR mehrere Modi eben diese Einstiegspunkte benötigen. Aufgrund der gewählten Serialisierungsmethode entsteht jedoch kein grosser Speicher-Overhead.

Die Serialisierung wird durch sogenannte Protocol Buffers [8] realisiert, wofür eine Open-Source Bibliothek namens *protobuf-net*² existiert. Protocol Buffers ist ein binäres Datenformat, welches von Google Inc. entwickelt wurde und bekannt ist für seine Kompaktheit und Simplizität. Ge wählt wurde es, weil die herkömmliche Serialisierung mit .NETs BinaryFormatter zum Teil zu Problemen mit Unity führen kann.

²<https://code.google.com/p/protobuf-net/>

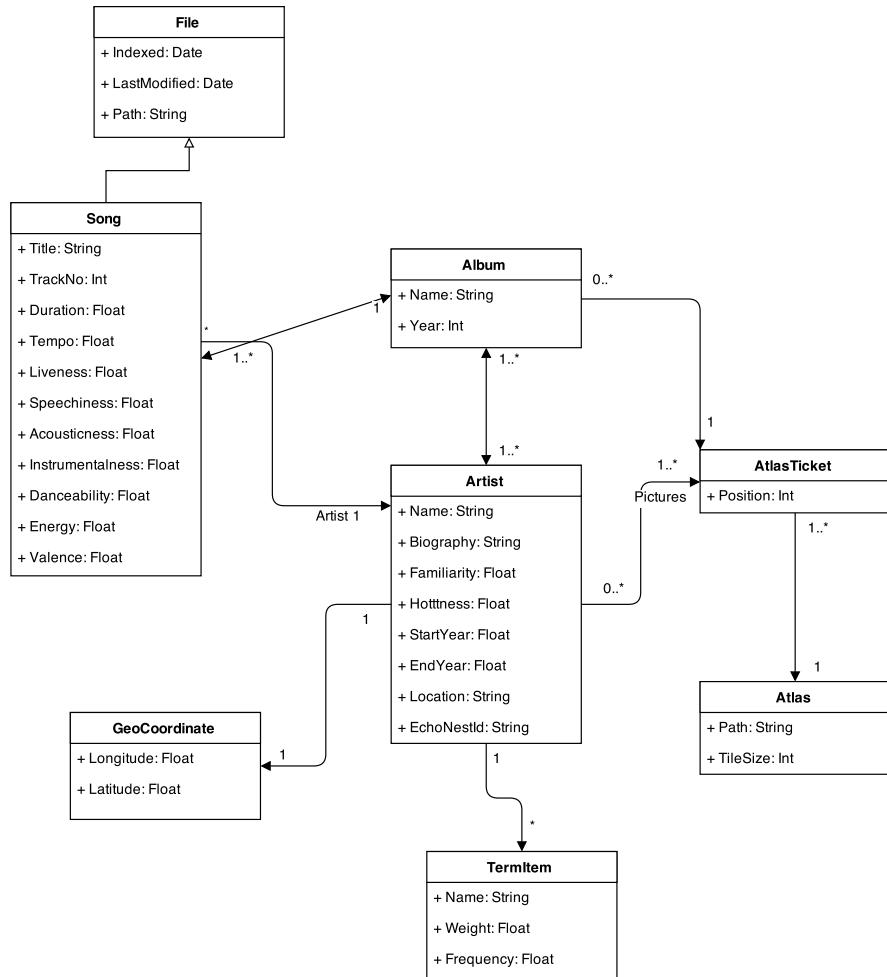


Abbildung 4.2: Klassendiagramm für die Klassen im IMVR.Commons Package.

4.4 Herausforderungen

Während der Entwicklung des Indexers kristallisierten sich diverse Schwierigkeiten, welche grundsätzlich in drei Kategorien einordnen lassen: organisatorische und datentechnische.

Die Organisation, oder Planung, war deshalb problematisch, weil einer der wichtigsten Faktoren für den Aufbau des Indexers die Vielfalt der Datentypen war. Da anfangs geplant war, Musik *und* Bilder zu indexieren und darzustellen, machte eine parallele Verarbeitung der Daten Sinn und war auch notwendig.

Es wurden parallel mehrere Bilder auf mehreren CPU-Kernen analysiert, und gleichzeitig wurde auch die Musikdatenbank erweitert. Mit dem Wegfall des Bilder-Parts wird die parallele Verarbeitung also um einiges unwichtiger. Dies gilt besonders, weil bei den Datenquellen der Musikindexierung jeweils nur ein Thread Sinn macht, da die APIs mit Limiten ausgestattet sind.

Datentechnisch stellte es sich als problematisch heraus, geeignete Datenquellen zu wählen. Von Anfang an war klar, dass der Service von The Echo Nest verwendet würde, doch dieser alleine ist nicht ausreichend.

Bevor schliesslich Last.fm als Quelle für Album-Covers gewählt wurde, wurden besonders zwei APIs untersucht: Amazon und Gracenote.

Im Falle von Amazon führten zwei Probleme zum Ausschluss: das Fehlen einer guten C#-Bibliothek und die schwerfällige Registrierung. Momentan sieht die Situation so aus, dass recht viel manuell gemacht werden muss, bzw. eine umfangreiche Service-Description importiert werden muss. [7] In Sachen Registrierung wird erwartet, dass man eine Webseite registriert, Kontaktdaten angibt und dann geprüft wird.

Bei Gracenote beläuft sich das Problem hauptsächlich auf die Daten-Limite. Diese ist nirgends öffentlich ersichtlich, und sobald diese überschritten wird, sind keine Requests mehr möglich für den ganzen Tag. In einer Applikation wie IMVR, wo in einem Schritt alles indexiert werden soll, ist die ein Killer-Kriterium.

5 Implementation von IMVR

Der Indexer ist implementiert und dokumentiert, also fehlt nur noch die eigentliche Applikation. Vieles hat sich im Laufe des Projekts verändert und entwickelt, und viele Probleme traten zum Vorschein, die an dieser Stelle genauer erläutert werden sollen.

5.1 Unity 5

Die Implementation von IMVR lässt sich leider nicht ohne einen gewissen Hintergrund in Unity 5 erklären.

5.1.1 Grundkonzepte

Unity ist eine Entwicklungsumgebung und eine Spiel-Engine, die momentan aufgrund ihrer Bedienungsfreundlichkeit und einer frei erhältlichen Version sehr beliebt in der Szene der Indie-Developer ist. Gleichzeitig dient sie auch als gutes Prototyping-Tool, um schnell Ideen umzusetzen.

Um Unity in groben Zügen zu erklären, sollen zwei Ansichtspunkte beschrieben werden: der Szenenaufbau und die Ressourcenverwaltung.

Ein Projekt in Unity ist in sogenannte *Scenes* (Szenen) gegliedert, welche aus Objekten (*GameObject*) bestehen – das Grundprinzip der Scene-Graphs wird also angewandt. Speziell ist, dass die Interaktion und Spiellogik grundsätzlich nur innerhalb von *Components* geschieht. Jedes Script und jede *Eigenschaft* wird als Component einem GameObject zugeordnet. So hat zum Beispiel ein Licht ein *Light*-Component oder die Kamera ein *Camera*-Component. Selbst die Position jedes GameObjects ist nur ein Wert im *Transform*-Component.

Ein anderer Aspekt von Unity ist die Ressourcenverwaltung. Im Grunde genommen ist es dem Programmierer überlassen, wie er seine Ressourcen verwaltet. Das einzige, was beachtet werden muss, ist, dass alle Ressourcen im *Assets*-Folder abgelegt werden müssen. Üblicherweise wird dann ein Ordner für jede Art von Asset erstellt, z.B. für Materialien, Texturen, Meshes, Scripts, etc.

Ein weiterer wichtiger Begriff sind die *Prefabs*. Damit sind Vorlagen gemeint, die man erstellt, indem man fertige Objekte aus der aktuellen Szene in das Asset-Folder zieht, und danach wiederverwerten kann.

5.1.2 Unity im Kontext von IMVR

Was sofort auffällt bei der Betrachtung der Ressourcenverwaltung, ist, dass diese starke Kopplung von Assets zu Projekten sich mit der grundlegenden Aufgabe dieses Projektes beisst. Unity sieht vor, dass der Programmierer während der Entwicklung alle seine Assets in den dafür vorgesehenen Ordner platziert, das Projekt am Schluss kompiliert, und dann höchstens im Nachhinein neue Assets als *Asset Bundles* an seine Anwender verteilt. In diesem Projekt ist es jedoch zwingend nötig, dynamisch Bilder und Musik anhand der Dateien auf dem Anwender-PC zu laden. Auf die Folgen und Lösungen zu diesem Problem wird in den Kapiteln 5.5.1 und 5.5.2 näher eingegangen.

5.2 Aufbau

5.3 Interaktionskonzept

Unüberraschenderweise findet die Interaktion des Users mit IMVR fast ausschliesslich mit seinen Händen statt. In einer frühen Phase des Projektes war noch geplant, eventuell die Spracheingabe modal zu den Händen zu gebrauchen, jedoch reichte dafür die Zeit nicht mehr. Ein Artefakt dieses Vorhabens ist das *SpeechServer*-Projekt, welches sich immer noch unter den *Auxiliary Tools* befindet.

5.3.1 Ringmenü

Bei der Entwicklung von VR-Applikationen stösst man zwingenderweise auf Situationen, in denen herkömmliche Konzepte nicht mehr verwendet werden können. Die Platzierung und der Aufbau des Menüs ist so ein Punkt.

Es ist nicht leicht ein Menü korrekt zu platzieren. Eine statische Platzierung als Overlay hält das Interface zwar im sichtbaren Bereich, kann sich jedoch als "lästig" herausstellen. Lässt man es verzögert mitschweben, gerät das Menü sofort ausser Kontrolle, und stellt man es irgendwo in die Szene und belässt es dabei, verliert man es sofort aus dem Blick.

Im Falle von IMVR bieten sich jedoch die Hände als gut verwendbarer Ankerpunkt für das Menü an. Es gibt ein freies Projekt auf GitHub [3], welches die Finger der Hand für die Platzierung der Buttons in einer ringartigen Struktur verwendet. Leider befand sich das Projekt in einem zu instabilen Stadium für diese Arbeit, aber es lieferte die Idee für eine eigene, ähnliche Implementierung.

Für IMVR wurde ebenfalls ein Ringmenü entwickelt, doch dieses verfügt über keine Schaltflächen im herkömmlichen Sinn. Die Finger werden auch mit Funktionen versehen, aber zum Betätigen benutzt der Anwender nicht seine andere Hand, sondern hebt einen Finger. Wenn ein Finger lange genug gehoben wird, wird die zugewiesene Aktion ausgeführt.

5.3.2 Fussplatten

Beim Erstellen einer visuellen Applikation stellt sich die Frage, wie man am besten die Struktur verdeutlichen kann. Eine Technik, die dafür gewählt wurde, ist der Einsatz von "Fussplatten".

Hierbei befinden sich unter den Füßen des Anwenders ringförmige Platten, welche die zwei Modi der Applikation repräsentieren. Eine dritte, zentriert abgehobene Platte dient zur Beendigung der Applikation.

Bei diesen Platten handelt es sich um das einzige Interaktionsmittel, welches bewusst keine Eingabe durch die Hände erfordert. In diesem Fall wird die Oculus Rift selbst als Eingabegerät verwendet, und zwar durch den Blickwinkel.

Die Idee ist, dass der Benutzer feststellen will, "wo er steht", herunterschaut, und durch gehaltenen Blickkontakt mit den Fussplatten diese aktivieren kann. Entsprechend den *Best Practices* wird dabei ein Indikator gefüllt, der anzeigt, wie lange der Blick noch gehalten werden muss.

Diese Art von Eingabe lässt sich oft bei bereits erschienen Demos für die Oculus Rift beobachten. Das Prinzip ist sehr leicht zu implementieren und daher auch verlockend, jedoch muss mit Vorsicht vorgegangen werden: Für den Benutzer ist es auf Dauer unangenehm, wenn von ihm ständige Kopfbewegungen gefordert werden. In IMVR wurde jedoch bewusst Gebrauch von dieser Methode gemacht, weil der Anwender nur selten nach unten schauen wird, und es relativ intuitiv ist.

5.3.3 Music Arm

Da es in dieser Arbeit voll und ganz um das Eingreifen ins Geschehen mit den eigenen Händen geht, wäre es bedauerlich, wenn es nicht möglich wäre, die Musik mit den Händen zu steuern. Ein weiteres Konzept, welches genau dieses Problem löst, ist der sogenannte "Music Arm".

Beim Music Arm handelt es sich um das Interface, welches an Stelle des menschlichen Arms angezeigt wird. Dieser digitale Arm bietet die Möglichkeit, die Playlist zu verwalten, sowie die momentan abgespielte Musik anzuschauen bzw. vor- und zurückzuspulen.

Hierbei wird eine Technik verwendet, welche einen Unterschied zwischen der Vorder- und der Rückseite macht. Schaut der Benutzer auf die "Uhr", also hält seinen Handrücken vor sich, dann erscheint ein Interface, in welchem er alle wichtigen Informationen über das momentane Lied erhält (Name, Artist, Cover, Länge und Fortschritt).

Dreht der Anwender jedoch seinen Arm und schaut auf die Rückseite, wird er eine Liste erhalten, die er scrollen und selektieren kann. Bei der Selektion wird das Element auf das Ringmenü "gesendet", wo der Anwender dann die Möglichkeit hat, das Lied abzuspielen, oder die Selektion aufzuheben. Würde man sofort bei der Selektion ein Lied abspielen, ergäbe sich das Problem, dass durch die mangelhafte Genauigkeit der Leap Motion und der Bedienung allgemein viele Fehlselectionen passieren würden und somit diverse Lieder zufällig abgespielt würden. Abbildung 5.1 zeigt, wie dieses Interface aussieht.

Der Entscheid, welche Seite angezeigt wird, geschieht über ein simples Skalarprodukt. Der Forward-Vektor der Kamera wird verglichen mit der Normale des Arms, welche in die gleiche Richtung zeigt, wie der Handrücken. Wenn der Wert tiefer als -0.5 ist, die Vektoren also entgegengerichtet sind, wird die Unterseite erkannt, und wenn der Wert höher als 0.5 ist, die Oberseite. Der Entscheid

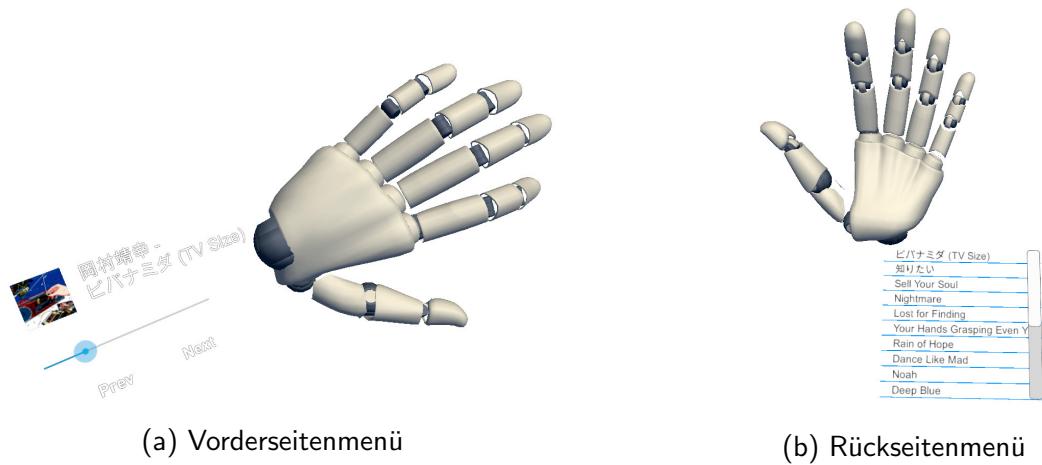


Abbildung 5.1: Darstellung des Music Arms

könnte auch bei 0 gefällt werden, aber dann würden auch Grenzfälle angezeigt werden, in denen besser gar kein Menü angezeigt wird.

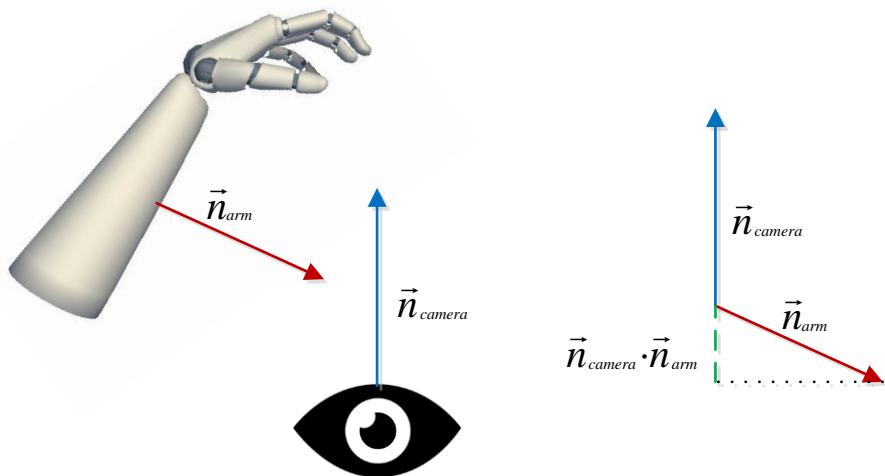


Abbildung 5.2: Bestimmung der momentanen Armseite

5.3.4 Slider Balken

Im Hörteil der Applikation, dem blauen Teil, sollte es dem Anwender ermöglicht werden, die Features der gewünschten Musik anzugeben, und diese dann anzuhören. Wie bereits erwähnt, bilden diese Werte einen Bereich zwischen 0 und 1 ab (mit Ausnahme des Tempos). Will der Anwender

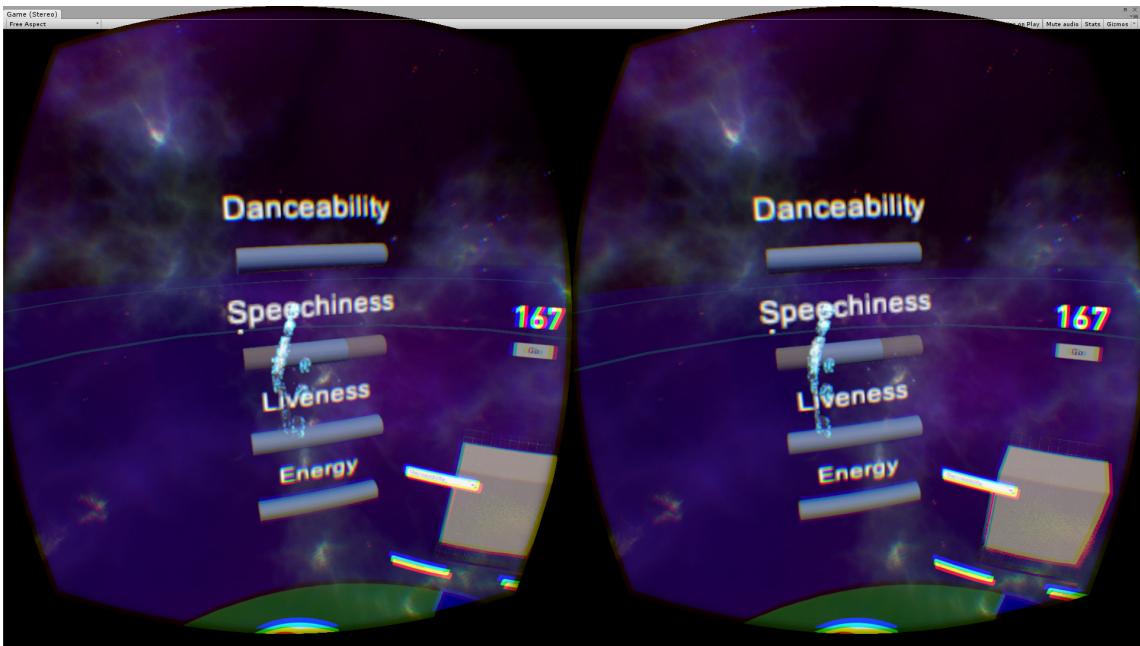


Abbildung 5.3: In-Game Screenshot zur Bedienung der Slider-Balken

eine energievolle Musik hören, muss es ihm deshalb möglich sein, neben dem Maximalwert auch einen Minimalwert anzugeben. Es liegt also ein typischer Use-Case für einen Range-Slider vor.

Da sich kein solcher im Werkzeugkasten von Unity befindet, musste ein eigener erstellt werden. Um einen anderen Approach zu testen, als mit der starken Verwendung von Unitys neuem UI-System in den anderen Teilen der Applikation gewählt wurde, wurde in diesem Fall bewusst ein Slider mithilfe eines Zylinder-Meshes erstellt.

Beim implementierten Slider hat der Anwender die Möglichkeit, direkt mit seinen zwei Händen einen Auswahlbereich einzustellen, indem er diese parallel in den Zylinder hält und entsprechend bewegt. Anders als die relativ abstrakte Implementation der anderen Interaktionselemente, basiert dieser Slider auf das Kollisionsmodell von Unity und die physikalische Hände, die mit dem Leap Motion Plugin mitgeliefert werden.

Das System funktioniert so, dass zu Beginn eine Achse \hat{v}_{dir} festgelegt wird, in die sich der Zylinder füllt (z.B. von links nach rechts). Bei einer Kollision der Hand mit dem Zylinder, wird nun zuerst geprüft, ob die Hand gültig ist. Sofern das der Fall ist, wird die Position der Handfläche \vec{P}_{hand} auf die Achse \hat{v}_{dir} projiziert und auf die Länge des Zylinders skaliert.

Der Wert, der dadurch entsteht, wird dann je nach geprüfter Handseite als neues Minimum bzw. Maximum verwendet. Allerdings geschieht davor noch eine lineare Interpolation vom vorigen Wert für einen weichen Übergang und ein Snapping zum Vereinfachen der Auswahl des Maximums bzw. Minimums.

Damit andere Teile der Applikation auf diese Wertänderungen reagieren können, wird beim Ändern der Extrema jeweils die Events `onMinValueChanged` bzw. `onMaxValueChanged` sowie `onValueChanged` ausgelöst. Durch Auffangen dieser Events wird z.B. der Zähler im `Selector` bei Änderungen der Selektion aktualisiert.

5.4 Visual Design

Viele Elemente in IMVR haben mit dem Visuellen zu tun. Sie sollen die Musik unterstützend begleiten, Verhältnisse darstellen und natürlich gefallen.

5.4.1 Darstellung der Hände

Das erste Thema, das beim Visuellen einfällt, sind die Hände. Bei der Gestaltung der Hände waren grundsätzlich folgende Voraussetzungen zu erfüllen:

1. handförmig
2. abstrakt
3. momentaner Modus ist erkennbar

Der erste Punkt versteht sich von selbst. Der zweite Punkt, die Abstraktheit, kommt daher, weil es am besten zur Applikation passt. Eine echt-aussehende, materielle Hand würde fehl am Platz wirken, wenn der Rest der Szene fast ausschliesslich aus Geometrie besteht und keinen Zusammenhang mit der Realität hat.

Neben dem erwähnten Argument, hat eine abstrakte Hand auch den Vorteil, dass die Darstellung der momentan Musik angepasst werden kann, z.B. durch Farbe oder Bewegung.

Beim dritten Punkt ist mit *Modus* der aktuelle Applikationsmodus gemeint, also der Browse- bzw. Listen-Modus. Diese werden unter Anderem durch die Farbe unterschieden, und diese Farbe zeigt sich auch bei der Visualisierung der Hände.

Für die erste Art der Visualisierung (Abbildung 5.4) wurde eine leicht abgeänderte Form einer Standardhand im Leap Motion Package gewählt. Dabei handelte es sich um eine auf Voxel basierende Darstellung. Für jeden Knochen in der Hand wurde ein Voxel-Sheet erstellt, welches dann den Bewegungen der Hand gefolgt ist. In der Implementation in IMVR wurden lediglich die Voxels mit Kugeln ersetzt, um ein bisschen Individualität zu schaffen.

Diese Darstellungsform wurde jedoch im Laufe des Projekts zugunsten einer partikelbasierten Implementation verworfen. In der ersten partikelbasierten Form, wurde versucht eine gerigchte Hand mit Partikeln zu umrahmen statt das Mesh selbst zu rendern. Das war jedoch mit ein paar grossen Nachteilen verbunden:

- (a) Die Bone-Weights werden auf der GPU appliziert, deshalb muss das Mesh in jedem Frame auf der CPU nachgebildet werden.
- (b) Durch die hohe Zahl der Vertices werden über 10 000 Partikel pro Hand erstellt, was sich sehr schlecht auf die Performance auswirkt.
- (c) Da das Finden von Punkten innerhalb eines komplizierten Meshes nicht trivial ist, müssen alle Punkte auf der Hülle platziert werden.

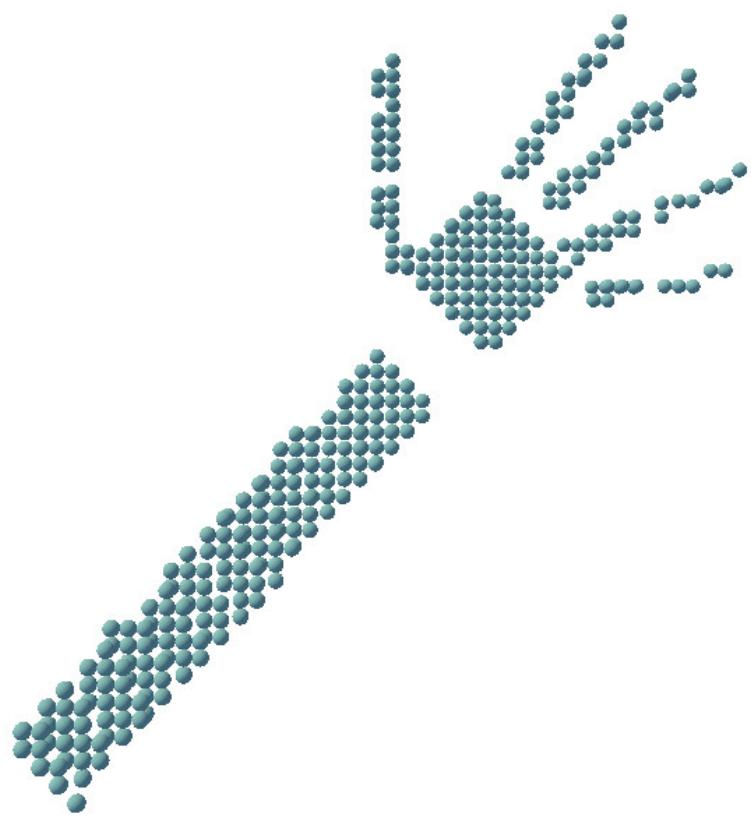


Abbildung 5.4: Die ursprüngliche Visualisierung der Hand

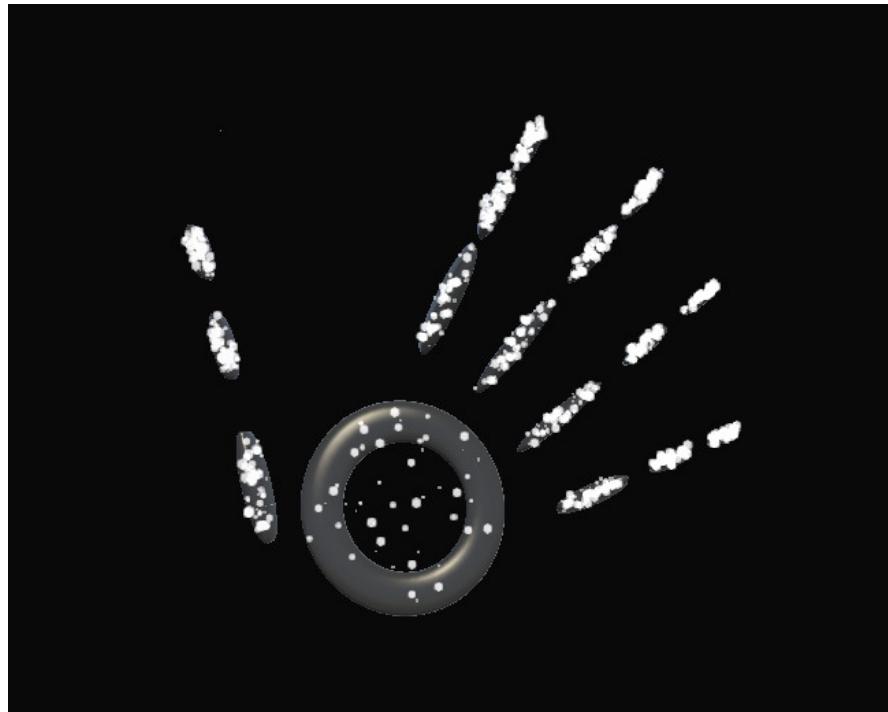


Abbildung 5.5: Die finale Visualisierung der Hand mit Partikeln

Aufgrund dieser Nachteile wurde die Implementation schliesslich noch einen Schritt weiter abgeändert. In der finalen Visualisierung (Abbildung 5.5) wird eine abstrakte Hand, welche aus mehreren Meshes besteht, mit Partikeln *gefüllt*. Damit erhält der Anwender ein intuitives Feedback, wenn die Hand erkannt wird, anstatt dass diese aus dem Nichts erscheint. Damit er trotzdem nicht auf den Effekt warten muss, erhält die Hand einen halb-durchsichtigen Rahmen.

Ein Grossteil des Codes konnte von der vorhergehenden Implementation mit dem geriggen Modell übernommen werden. Der Vorteil hier ist, dass die Hand auf *Transforms* aufbaut: Die Submeshes, ein Mesh pro Knochen, werden durch ihre *Transforms* korrekt platziert, damit eine Hand gebildet wird. Dadurch entsteht zwar nicht ein fliessender Übergang wie bei einer geriggen Hand, aber man kann mit primitiven Elementen arbeiten.

In diesem Fall wurde ein Handmodell gewählt, welches grösstenteils aus einfachen, abgerundeten Quadern besteht. Bei der Erstellung der Hand, werden alle Meshes abgelaufen und für jedes Mesh eine gewisse Anzahl Partikel erstellt. Damit diese *innerhalb* der Hand erscheinen, und nicht auf der Hülle wie bei der vorherigen Implementation, wird jede Partikelposition aus zwei zufälligen, verschiedenen Vertices gebildet mit einer ebenfalls zufälligen Gewichtung. Weil die Formen alle konvex sind, ist garantiert, dass die gebildeten Punkten alle innerhalb der Meshes liegen.

Um schliesslich einen schönen Effekt zu erzielen, wurden noch ein paar zusätzliche Details eingebaut:

- Die Partikel sind animiert
- Die Partikel erhalten die Farbe des aktuellen Modus'
- Die Partikel werden beim Verlieren der Handerkennung zerstreut

Alles in allem wurde damit eine ansprechende und intuitive Darstellung der Hände erreicht.

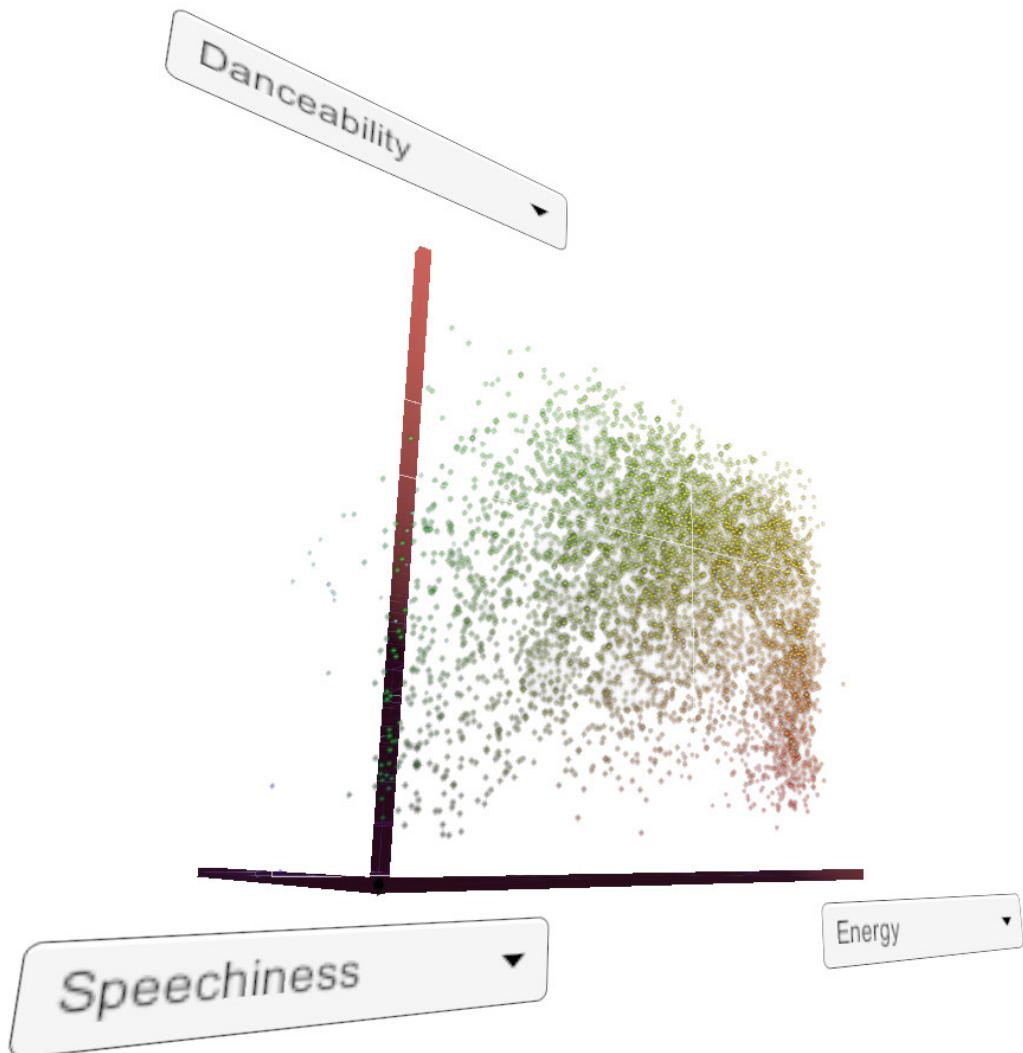


Abbildung 5.6: Visualisierung der Musiksammlung innerhalb eines Point Charts

5.4.2 Point Chart

Wenn man von Datenvisualisierung redet, ist es ganz klar, dass auch Diagramme dargestellt werden wollen. Umso besser, wenn dies stereoskopisch geschehen kann: Dadurch lässt sich die dritte Dimension erheblich besser wahrnehmen.

Im Rahmen dieses Projekts wurde ein Point-Chart entwickelt, der dazu dient, die Musik des Anwenders in ein dreidimensionales System einzuordnen und auf einen Blick darzustellen. Die Dimensionen sind 1:1 die Features, die während der Indexierung von The Echo Nest heruntergeladen wurden, also Merkmale wie *Danceability*, *Energy* und *Tempo*.

Wie auf dem UML ersichtlich, wurde die Implementation des Point Charts in drei verschiedenen Klassen durchgeführt.

PointChart ist eine relativ generisch gehaltene Implementation eines Point Charts, welche sich nur um die Darstellung kümmert.

SongMetaChart ist eine Komponente, die auf der ersten basiert und diese steuert, und die Metadatenlogik ins Spiel bringt.

SelectionVisualizer ist eine weitere Komponente, die ihrerseits auf SongMetaChart aufbaut und die momentane Selektion (nur im Listen-Modus) visualisiert.

Die zentrale Klasse der Funktionalität, PointChart, besteht visuell ebenfalls aus drei Teilen. Es werden drei Elemente bei Änderungen prozedural erstellt:

- Ein Koordinatensystem, das aus drei Balken besteht, welche mit *Triangles* generiert werden
- Ein dreidimensionales Gitternetz, welches mit Linien erstellt wird
- Einer Punktewolke aus Partikeln

Die zwei Meshes werden grundsätzlich nur einmal generiert, können aber mithilfe der Methode RebuildMesh() jederzeit neu erzeugt werden. Die Linien des Gitters bestehen aus einem durchsichtigen Material-Shader, welcher ihnen eine passend subtile Sichtbarkeit gibt.

Das Partikelsystem, welches für die Punkte des Diagramms zuständig ist, sollte immer bei Änderungen der Punktewerte mithilfe der Methode UpdatePoints() erneuert werden.

Ursprünglich war geplant, diese Punkte durch ein *Point*-Rendering zu implementieren, doch diese Art des Renderings ist im Falle von Unity relativ limitiert. Will man lediglich Punkte als einzelne Pixel darstellen, stellt das kein Problem dar. Problematisch wird es, sobald die Grösse der Punkte geändert werden soll, denn dann ist man auf einen OpenGL-Build angewiesen [4].

Die Implementation per Partikelsystem funktioniert gut. Auf was geachtet werden muss, ist die Lebenszeit der einzelnen Partikel. Da sie in diesem Fall nicht kurzlebig sind sondern persistent, wird ihre Lebenszeit bei der Emission der Partikel auf einen Wert gesetzt, der mit sehr hoher Wahrscheinlichkeit nie erreicht werden wird.

```
1 public void UpdatePoints()
2 {
3     float scale = transform.lossyScale.x;
4
5     var particles = new ParticleSystem.Particle[points.Length];
6     particleSystem.Emit(points.Length);
7     particleSystem.GetParticles(particles);
8     for (int i = 0; i < points.Length; i++)
9     {
10         particles[i].size = Mathf.Lerp(0.3f, 0.05f, points.Length / 5000) *
11             scale;
12         particles[i].lifetime = particles[i].startLifetime = 100000f;
13         particles[i].velocity = Vector3.zero;
14         particles[i].position = (points[i] + pivotOffset) * scale;
15         particles[i].color = new Color(points[i].x, points[i].y, points[i].z);
16         particles[i].angularVelocity = Random.RandomRange(-45f, 45f);
17     }
18     particleSystem.SetParticles(particles, particles.Length);
19 }
```

Listing 5.1: Erstellung der Punkte in einem PointChart

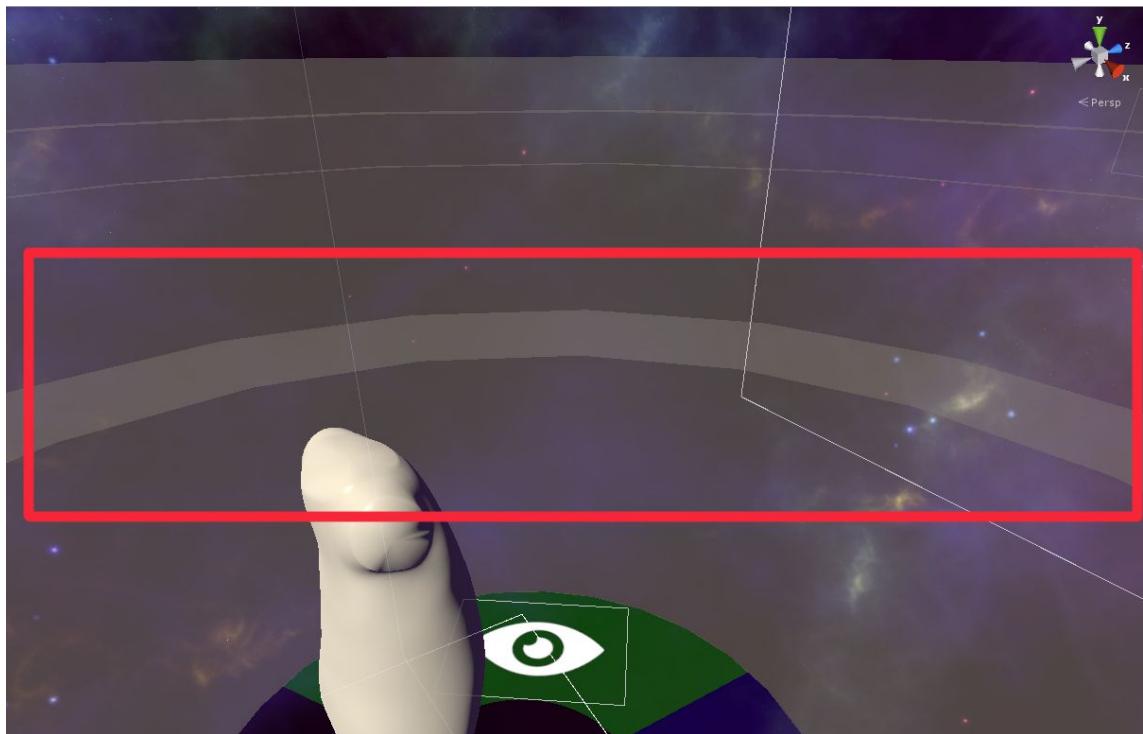


Abbildung 5.7: Beats werden mithilfe von Wellen dargestellt (rot umrahmt)

5.4.3 Farben und Symbolik

5.4.4 Beat Detector und Wellen

Damit die Szene, in der sich der Anwender wiederfindet, nicht zu steril und still ist, wurde ein sehr rudimentärer Beat Detector implementiert, der bei jedem Beat eine Welle mit entsprechender Stärke aussendet.

Für die Programmierung der Beat-Detection wurde ein Artikel von GameDev.com zurate gezogen [6]. Ganz grob gesagt, wird einfach nach Peaks im Energieverlauf der Musik gesucht. Die benötigten Werte werden alle durch die selbsterstellte Helper-Klasse `VisualizationHelper` bereitgestellt. Als kleine Veränderung zum ursprünglichen Algorithmus wurde die Suche nach Beats auf das Frequenzspektrum zwischen 50Hz und 200Hz beschränkt.

Anstatt einer dynamischen Beat-Detection hätte auch ein Pre-Processing durchgeführt bzw. die Werte von The Echo Nest verwendet werden können. Allerdings ist eine so genaue Beat Detection in diesem Fall nicht notwendig, und für ein exaktes Timing wäre eine stetige Synchronisation zwischen Daten und Musik notwendig.

Die Ringe, die ausgesendet werden, werden alle prozedural generiert und passen sich der *Theme*-Farbe an. Das System basiert auf drei Klassen:

- Der `BeatDetector` versendet bei Beats eine `OnBeat(strength)` Nachricht an Komponenten im gleichen `GameObject`.
- Der `WaveEmitter` fängt die Nachricht ab und erstellt ein neues `GameObject` mit einer `RingMesh`-Komponente.

- RingMesh sorgt dann schliesslich dafür, dass der Kreisring bzw. das Kreissegment korrekt dargestellt wird.

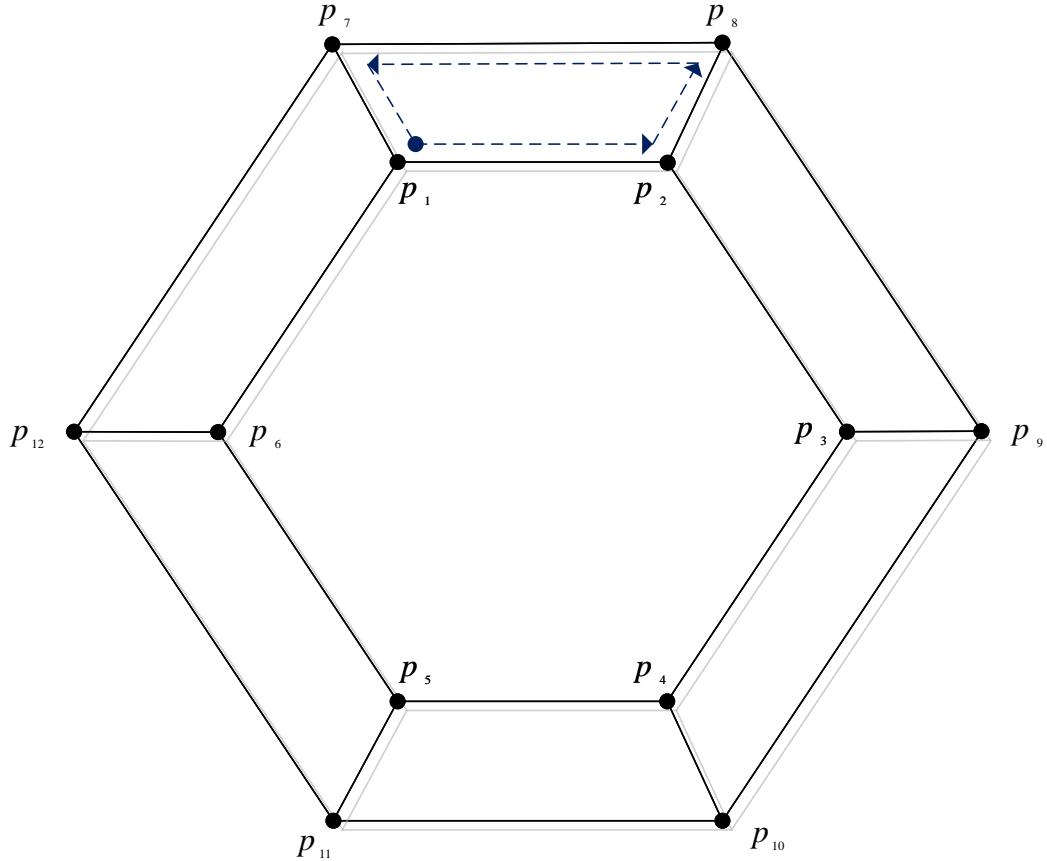


Abbildung 5.8: Die Ring-Meshes werden mithilfe von *Quads* generiert

Die Erstellung des Kreisrings ist sehr simpel gehalten. Die Komponente erlaubt das Konfigurieren von Innenradius, Aussenradius, Startwinkel, Winkelbereich und Anzahl Segmenten. Diese Eigenschaften werden überwacht, und bei jeder Änderung wird beim nächsten Update der Engine eine Neugenerierung des Meshes ausgelöst.

Bei der Generierung wird ein Array der Länge $n_{\text{Segmente}} * 2$ erstellt, welches die Vertices enthält. Die Punkte des Aussen- und Innenkreis werden dann gleichzeitig in dieses Array gefüllt, wobei der Aussenkreis einen Offset von n_{Segmente} erhält (siehe Abbildung 5.8). Schliesslich werden dann die Quads mit jeweils vier Indexen gebildet.

5.4.5 Visualisierung der Musik

5.4.6 Avatar

Auch der Anwender selbst muss irgendwie dargestellt werden. Ein Avatar gilt generell als ein Hilfsmittel, um die Immersivität zu steigern, kann jedoch auch verschlechternd wirken [5].



Abbildung 5.9: Der verwendete Avatar mit Kopf und Armen

Quelle: <http://www.turbosquid.com/FullPreview/Index.cfm/ID/833108>

Im Falle von IMVR wurde jedoch die Entscheidung getroffen, einen simplen Avatar einzubauen. Aufgrund der Art der Applikation, fiel hierbei die Wahl auf eine abstrakte Figur, die frei verfügbar und vollständig geriggt ist. Dadurch bleibt jederzeit die Möglichkeit offen, per IK die Figur zu steuern.

Da die Hände und der Kopf nicht benötigt werden, wurden diese kurzerhand mithilfe von 3DS Max 2015 demontiert. Die vorherig erwähnte Kontrolle über IK wurde nicht implementiert, könnte aber in einem nächsten Schritt eingebaut werden.

5.5 Herausforderungen

5.5.1 Ressourcen-Management

5.5.2 Abspielen externer Musik

5.5.3 Canvas-Elemente

5.5.4 Performance

5.5.5 Auswerten der Musik

6 Testergebnisse

7 Projektmanagement

7.1 Soll-Ist

7.2 Zeitplan

8 Schlussbetrachtung

Zum Schluss soll kurz ein Blick auf die Gegenwart, die Vergangenheit und die Zukunft geworfen werden. War das Projekt erfolgreich? Wo lagen die *Pitfalls*? Wie geht es weiter? Diese Fragen sollen an dieser Stelle beantwortet werden.

8.1 Ergebnis

8.2 Reflexion

Grundsätzlich ist das Ergebnis zufriedenstellend, doch gibt es trotzdem zahlreiche Punkte, welche hätten besser gelöst werden können.

Teilweise aufgrund der experimentellen Natur des Projektes war die Planung nicht so ausgearbeitet und zuverlässig, wie sie es in einem herkömmlichen Projekt wäre. Durch das mehrfache Verändern von Funktionalität, dem wiederholten Refactoring, und der geänderten Zielsetzung entstand viel Komplexität und "Code-Smell" [9].

8.3 Ausblick

Selbständigkeitserklärung

Ich/wir bestätige/n, dass ich/wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt habe/n. Sämtliche Textstellen, die nicht von mir/uns stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Ort, Datum: Biel, 08.06.2015

Name Vorname: Meer Simon

Unterschrift:

Literaturverzeichnis

- [1] "Leap motion - 3D motion and gesture control for Windows & Mac." [Online]. Available: <https://www.leapmotion.com/product/vr>
- [2] T. Auensen, "echonest-sharp." [Online]. Available: <https://github.com/torshy/echonest-sharp>
- [3] Z. Kinstner, "Hover VR interface kit." [Online]. Available: <https://github.com/aestheticinteractive/Hover-VR-Interface-Kit>
- [4] Lovelock, "How do I use PSIZE in a Unity 4.5.4 shader?" [Online]. Available: <http://answers.unity3d.com/questions/807548/how-do-i-use-psize-in-a-unity-454-shader.html>
- [5] L. Oculus VR, "Best practices guide," 2015. [Online]. Available: http://static.oculus.com/sdk-downloads/documents/Oculus_Best_Practices_Guide.pdf
- [6] F. Patin, "Beat detection algorithms," 2003. [Online]. Available: http://www.gamedev.net/page/resources/_/technical/math-and-physics/beat-detection-algorithms-r1952
- [7] O. Trutner, "Signing amazon product advertising API requests ? the missing C# WCF sample," 2009. [Online]. Available: <https://flyingpies.wordpress.com/2009/08/01/17/>
- [8] Wikipedia, "Protocol buffers," 2015, [Online; accessed 24-May-2015]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Protocol_Buffers&oldid=662187371
- [9] ——, "Smell (Programmierung)," 2015, [Online; Stand 1. Juni 2015]. [Online]. Available: [http://de.wikipedia.org/w/index.php?title=Smell_\(Programmierung\)&oldid=137769083](http://de.wikipedia.org/w/index.php?title=Smell_(Programmierung)&oldid=137769083)

Abbildungsverzeichnis

3.1	Eine Übersicht der Technologien und wie sie verbunden sind.	5
3.2	Grober Überblick des Programmablaufs	6
3.3	Klassendiagramm für die Klassen im IMVR.Commons Package	8
3.4	Klassendiagramm des Indexers	10
3.5	Klassendiagramm des Interfaces der Applikation (View-Behavior)	12
3.6	Klassendiagramm des Music Arms	14
3.7	Klassendiagramm des Ring Panels, womit der Anwender Modi auswählt	16
3.8	Klassendiagramm des Teils, der die Daten der Datenbank verwendet	17
3.9	Klassendiagramm der Effekte und prozeduralen Meshes	20
3.10	Klassendiagramm des Abstraktionslayers für die Leap Motion	22
4.1	Der Datenfluss, den die Files beim Indexieren nehmen.	25
4.2	Klassendiagramm für die Klassen im IMVR.Commons Package.	26
5.1	Darstellung des Music Arms	31
5.2	Bestimmung der momentanen Armseite	31
5.3	In-Game Screenshot zur Bedienung der Slider-Balken	32
5.4	Die ursprüngliche Visualisierung der Hand	34
5.5	Die finale Visualisierung der Hand mit Partikeln	35
5.6	Visualisierung der Musiksammlung innerhalb eines Point Charts	36
5.7	Beats werden mithilfe von Wellen dargestellt (rot umrahmt)	38
5.8	Die Ring-Meshes werden mithilfe von <i>Quads</i> generiert	39
5.9	Der verwendete Avatar mit Kopf und Armen	40

Tabellenverzeichnis

2.1	Übersicht der eingesetzten Software.	3
2.2	Übersicht der eingesetzten Hardware.	3
3.1	Erklärung der wichtigsten allgemeinen Klassen	9
3.2	Erklärung der wichtigsten Klassen des Indexers	11
3.3	Erklärung der wichtigsten Interface-Klassen	13
3.4	Erklärung der wichtigsten Music-Arm-Klassen	15
3.5	Erklärung der Klassen des Ring-Panels	16
3.6	Erklärung der wichtigsten Klassen von IMVR.Data	18
3.7	Erklärung der Klassen von IMVR.Data.Image	18
3.8	Erklärung der wichtigsten Klassen von IMVR.Data.Music	19
3.9	Erklärung der FX-Klassen, die sich mit der Musik-Visualisierung befassen	21
3.10	Erklärung der FX-Klassen, die sich mit der Musik-Visualisierung befassen	21
4.1	Die Projekte in Auxiliary Tools	23
4.2	Eine Übersicht von verfügbaren Online-Datenquellen.	24