UNIVERSITY OF APPLIED SCIENCES
HOCHSCHULE MITTWEIDA

Angewandte Computer- und Biowissenschaften

Professur Medieninformatik

Bachelorarbeit

Konzeption und prototypische Implementierung eines Rhythmik-Videospiels mit automatischer Levelgenerierung auf Grundlage der Analyse von Musikstücken

William Rapprich

Mittweida, den 30. Oktober 2018

Erstprüfer: Prof. Dr. rer. nat. Marc Ritter
Zweitprüfer: M. Sc. Manuel Heinzig
Co-Betreuer: B. Sc. Sebastian Schleier

Rapprich, William
Konzeption und prototypische Implementierung eines Rhythmik-Videospiels mit automatischer Levelgenerierung auf Grundlage der Analyse von Musikstücken
Bachelorarbeit, Angewandte Computer- und Biowissenschaften
Hochschule Mittweida– University of Applied Sciences, Oktober 2018

Referat

In dieser Arbeit wird ein prototypisches Rhythmik-Videospiel für die Virtuelle Realität konzipiert und entwickelt. Das Hauptmerkmal der Anwendung ist die automatische Generierung der Spielrunde auf Grundlage der Analyse von Musikstücken, welche über eine freie Softwarebibliothek realisiert ist, die Audiodateien einlesen und musikalische Merkmale extrahieren kann. Abschließend wird die Eignung der generierten Spielinhalte in Bezug auf die Wahrnehmung der Musikstücke durch den Spieler evaluiert.

Name: Rapprich, William
Studiengang: Medieninformatik und Interaktives Entertainment
Seminargruppe: MI15w1-B

Inhaltsverzeichnis

1 Einleitung und Motivation **1**

1.1 Aufgabenbeschreibung 2
1.2 Kapitelübersicht 3

2 Grundlagen **5**

2.1 Virtuelle Realität 5
2.2 HTC Vive 10
2.3 Unreal Engine 13
2.4 Rhythmik-Videospiele 20
2.5 Technische Akustik und Medienkodierung 21

3 Anforderungen und Spezifikation **25**

3.1 Anforderungsanalyse 25
3.2 Softwarearchitektur 28

4 Implementierung **33**

4.1 Plugins und Bibliotheken 33
4.1.1 Extrahierbare Merkmale von Audiosignalen 33
4.1.2 Überführung in Levelinhalte 34
4.1.3 Wahl der geeigneten Bibliothek 35
4.2 Zentrale Spielkoordination und Menü 37
4.3 Levelgenerierung 39
4.4 Spielrundenlogik 41
4.5 Spielparameter 43
4.6 Blueprints 43

5 Evaluation **45**

5.1 Versuchsaufbau und Testergebnisse 45
5.2 Auswertung und Interpretation 49

6 Zusammenfassung und Ausblick **55**

Anhang **57**

A Datensätze **A1**

A.1 Rock A1
A.2 Hip Hop A3
A.3 Klavier A5
A.4 Jazz A7
A.5 Electro A10

B Mittelwerte **A13**

B.1 Rock A13
B.2 Hip Hop A14
B.3 Klavier A14
B.4 Jazz A14
B.5 Electro A15
B.6 Alle Probanden A15

C Statistische Tests **A17**

D Zusammenfassung der Anmerkungen **A19**

Literaturverzeichnis **III**

1. Einleitung und Motivation

Die Virtuelle Realität (Virtual Reality, VR), deren erste technologische Ansätze bereits in den 1960er Jahren [BC03, DBGJ13, GVT08] zu verzeichnen sind, erfreut sich heutzutage immer größerer Beliebtheit [Mur18]. Zunächst konnte sich ein Markt aufgrund hoher Kosten für den Nutzer nicht ausreichend etablieren [BC03, S. 10]. Mit der Einführung von bezahlbaren *head-mounted displays* (HMD), beispielsweise der *HTC Vive*[1] im Jahr 2016 [Tea16], und zugehörigen Eingabegeräten war jedoch die Verbreitung im professionellen und privaten Bereich möglich. Die Nutzung erstreckt sich von virtuellem Operationstraining in der Medizin über Führungen durch historische Schauplätze bis hin zu Fahrsimulationen [GVT08, S. 165–188], um nur einen Bruchteil der Möglichkeiten zu nennen.

Ein Gebiet der VR-Anwendung im Freizeitbereich stellen Videospiele dar. Das Rhythmik-Videospiel *Beat Saber*[2] zählt zu den erfolgreichsten Virtual Reality Games auf der Vertriebsplattform *Steam*[3] [Gam18]. Kernmechanik ist das bereits aus Spielen wie *Guitar Hero*[4] oder *Rockband*[5] bekannte Prinzip der Aktivierung bestimmter Elemente in Korrelation zu einem Musikstück. Ziel von *Beat Saber* im Speziellen ist das Erreichen möglichst hoher Punktzahlen durch das rhythmische Zerschlagen von Würfeln mittels Lichtschwertern [Ste18]. Der Spieler erhält dafür im virtuellen Raum zwei verschiedenfarbige Lichtschwerter, deren Bewegungen an Hardware-Controller gekoppelt sind, die in den Händen gehalten werden. Zu einem gewählten Musikstück entstehen in der Ferne in einer der Lichtschwertfarben leuchtende Würfel, die sich kontinuierlich auf den Spieler zubewegen, bis sie schließlich nah genug sind, um vom farblich passenden Lichtschwert zerschlagen zu werden (siehe Abb. 1.1). Zusätzlich sind sie mit Pfeilen markiert, die eine Einschränkung für die Schlagrichtung symbolisieren. Es gibt außerdem Hindernisse, deren Berührung durch den Kopf oder

[1] Produktseite: `https://www.vive.com/`
[2] Produktseite: `http://www.beatsaber.com`
[3] Client-Download: `https://store.steampowered.com/about/`
[4] Produktseite: `https://www.guitarhero.com/`
[5] Produktseite: `https://www.rockband4.com/`

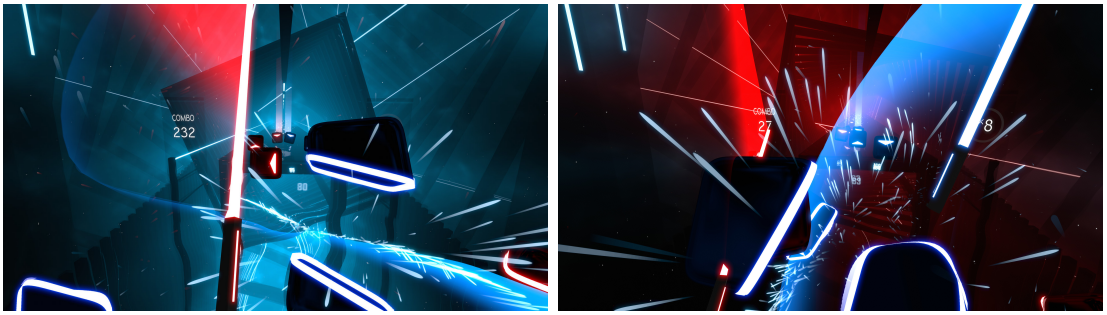

Abbildung 1.1.: Screenshots vom Spielgeschehen in *Beat Saber* (Aus: [Ste18])

die Lichtschwerter vermieden werden muss. Die Platzierung und Markierung aller Elemente in *Beat Saber* wurde dabei von den Entwicklern zu selbst komponierten Musikstücken für jeden Schwierigkeitsgrad manuell vorgenommen. Somit gibt es nur eine kleine Auswahl von spielbaren Levels, die nicht durch den Import von Titeln aus der eigenen Musikbibliothek erweitert werden kann. Die Arbeit beschäftigt sich damit, eine alternative Softwarelösung für ein solches Rhythmik-Spiel zu schaffen, welche die Levelgenerierung auf Grundlage der Analyse von Musikstücken automatisch vornimmt, um zu jedem beliebigen Lied spielen zu können.

1.1. Aufgabenbeschreibung

Im Fokus dieser Arbeit steht die Entwicklung einer prototypischen, zu *Beat Saber* funktionsähnlichen Anwendung, welche die händische Platzierung durch automatisierte Levelgenerierung auf Basis einer Analyse der verwendeten Musikstücke ersetzt. Anhand von Merkmalen, wie beispielsweise dem Tempo, der Amplitude oder der Frequenz an bestimmten Stellen einer Tonspur, sollen die zu zerschlagenden Würfel in einem gewissen Tempo, Größe, Geschwindigkeit usw. zu entsprechend passenden Zeitpunkten generiert werden. Hierfür findet zunächst eine Untersuchung und theoretische Auswahl möglicher aus Audiomaterial extrahierbarer Merkmale unter Berücksichtigung ihrer Eignung im genannten Spielkontext statt. Anhand der erlangten Erkenntnisse werden Kriterien für eine in Frage kommende Implementierung abgeleitet. Diese dienen im Anschluss als Grundlage für einen Vergleich mehrerer frei verfügbarer Softwarebibliotheken zur Audioanalyse. Ziel ist es, die Bibliothek zu finden, welche die Anforderungen am besten erfüllt. Damit wird anschließend die Implementierung der prototypischen Anwendung zur Levelgenerierung als zen-

tralem Bestandteil des Rhythmik-Spiels in der *Unreal Engine*[6] unter Verwendung der Programmiersprache *C++* durchgeführt. Abschließend erfolgt eine Evaluation der Tauglichkeit der gewählten Algorithmen bezüglich ihrer Funktionsfähigkeit im Spielkontext mit einer repräsentativen Auswahl von Audiodateien verschiedener Musikgenres.

1.2. Kapitelübersicht

Kapitel 1 gibt einen Einblick in die Motivation hinter dieser Arbeit und führt den Leser an die Problematik sowie die damit verbundene Zielsetzung heran.

Kapitel 2 baut wissenschaftliches und technisches Grundwissen auf, insbesondere durch die Definition von Fachbegriffen, welches zum Verständnis der folgenden Kapitel beiträgt. Der Fokus liegt hierbei auf dem Umgang mit der *Unreal Engine* im Zusammenhang mit der *HTC Vive* in der Virtuellen Realität. Grundlagen der technischen Akustik werden ebenfalls vermittelt.

Kapitel 3 und **Kapitel 4** befassen sich mit der Planung und Entwicklung des Prototyps. Eine anfängliche Anforderungsanalyse und die anschließende Konzeption der Softwarearchitektur sind nötige Mittel zur erfolgreichen Umsetzung des Programms. Die Wahl der geeigneten Softwarebibliothek für die Audioanalyse wird ebenfalls behandelt. Die Betrachtung der wesentlichen Implementierungsdetails geordnet nach den verschiedenen Funktionsbereichen erfolgt in Kapitel 4.

In **Kapitel 5** wird die Vorgehensweise zur Evaluation der Anwendung dargelegt. Die durch Probandentests erhobenen Ergebnisse werden aufbereitet und interpretiert.

Kapitel 6 gibt einen retrospektiven Blick auf die Umsetzung der prototypischen Anwendung. Außerdem werden die Kernproblematiken angesprochen, die bei der Weiterentwicklung des Programms berücksichtigt werden müssen.

[6] Produktseite: `https://www.unrealengine.com/`

2. Grundlagen

Für die Umsetzung des geplanten Projektes ist die Auseinandersetzung mit zugrunde liegenden Fachgebieten notwendig. Mit der *Unreal Engine* wird ein prototypisches Rhythmik-Videospiel erstellt, welches mittels der Ein- und Ausgabegeräte der *HTC Vive* ein Erlebnis in der Virtuellen Realität bieten kann. Die Analyse von Musikdateien zur automatischen Levelerstellung übernimmt eine Softwarebibliothek, die sich Prinzipien der technischen Akustik und Medienkodierung bedient. Dieses Kapitel betrachtet theoretische Grundlagen aus den eben genannten Bereichen.

2.1. Virtuelle Realität

Das Phänomen der Virtuellen Realität, als eine Schnittstelle zur Kommunikation zwischen Computer und Mensch [BC03, S. 1], hat seit seiner Begriffsbildung unterschiedliche Definitionen aus verschiedenen Quellen erhalten [Bri09, S. 5]. Im Grunde ist der Ausdruck irreführend, da es sich bei den Wörtern „virtuell“ und „real“ um gegensätzliche Adjektive handelt [Bri09, S. 6]. Weitere Begriffe, die in den 1990er Jahren entstanden, sind *virtual environment* (deutsch virtuelle Umgebung) oder *synthetic environment* (deutsch künstliche Umgebung) [GVT08, S. 1]. Sie mögen zwar zutreffender sein, allerdings hat sich weltweit die Bezeichnung Virtuelle Realität und dessen englische Version Virtual Reality mit der Abkürzung VR durchgesetzt, weswegen diese im Folgenden Verwendung finden.

Mit dieser Wortschöpfung ging anfänglich ein großes Erwartungsbild einher, welches jedoch nicht erfüllt werden konnte und somit zerbrach. „Die Idee war, dass diese Technologie erfundene Welten schaffen könnte, die von der realen Welt nicht zu unterscheiden wären“ [GVT08, S.1, Übers. durch Verfasser]. Die Technik war allerdings dafür nicht fortgeschritten genug. [GVT08, S.1]

Auch wenn die erstmalige Vorstellung von der Virtuellen Realität zunächst nicht umsetzbar war und noch immer nicht ist, konnte daraus die Definition für die perfekte

Virtuelle Realität abgeleitet werden: Eine computergestützte Simulation der Welt und die damit zusammenhängende Erzeugung von Reizen, die auf den Menschen wie in der realen Welt einwirken. Dabei müssen alle Interaktionsmöglichkeiten mit der Welt erhalten bleiben und das System muss darauf in Echtzeit reagieren können. Wichtige Komponenten, welche die Gesamtwahrnehmung des Menschen ausmachen, sind das Sehen, das Hören, das Riechen, das Schmecken, das Erfühlen und Tasten, der Gleichgewichtssinn, die Körperempfindung, das Temperaturgefühl und die Schmerzempfindung. Sofern alle diese Empfindungen und Interaktionen realitätsnah simuliert werden, ist dem Menschen nicht mehr bewusst, dass es sich um eine scheinbare Wirklichkeit handelt. Es wird klar, dass ein solches Computersystem aufgrund der hohen Anforderungen enorme Komplexität und Rechenleistung vorweisen muss. Ein perfektes *VR-System* – ein System aus Computer und Ein- und Ausgabegeräten für die VR – gibt es deshalb noch nicht. [DBGJ13, S. 2–7]

Allerdings ist dies für die praktische Anwendung der heutigen VR-Technologie nicht nötig. Vor allem in der Unterhaltungsindustrie wird sich ein Konzept mit dem Namen *willing suspension of disbelief* zunutze gemacht. [DBGJ13, S. 8]

> „Menschen haben [demnach] die Eigenschaft in bestimmten Situationen den augenscheinlichen Widerspruch einer virtuellen oder fiktiven Welt zur Realität auszublenden und dies auch zu wollen“. [DBGJ13, S. 8]

Ein VR-Videospiel muss also keine perfekte Illusion bieten, um vom Spieler als unterhaltsam aufgefasst zu werden oder ihm das Gefühl zu vermitteln, dass die fiktive Welt echt sei. Andere Anwendungen haben nicht einmal den Anspruch an die Realitätsnähe und können trotzdem gewünschte Informationen vermitteln.

Daher könnte eine Definition für die tatsächliche Anwendung von Virtueller Realität in der heutigen Zeit folgendermaßen lauten: Die computergestützte Simulation einer Umgebung, welche von einem Menschen als glaubwürdig wahrgenommen werden kann und die genug Interaktivität aufweist, um gewisse Aufgaben auf effiziente und angenehme Weise zu erfüllen. [GVT08, S. 2]

Der Grad der Trennung von Realität und Simulation durch die Benutzerschnittstelle – wie wenig also von der Wahrnehmung der realen Umgebung verbleibt – wird als *Immersion* bezeichnet. *Presence* (deutsch Präsenz) hingegen beschreibt den psychischen Zustand, dass der Nutzer sich selbst innerhalb der virtuellen Umgebung wahrnimmt. Diese Faktoren spielen eine große Rolle im Erlebnis der Virtuellen

Abbildung 2.1.: v. l. n. r.: Morton Heiligs *Sensorama* (1962) und Ivan Sutherlands HMD (1968) (Aus: [GVT08, S. 5])

Realität. In der Literatur werden der technische und mentale Aspekt oft unter *Immersion* zusammengefasst. Im Weiteren sind diese aber fortwährend getrennt zu betrachten. [GVT08, DBGJ13]

Wichtig für die VR ist, dass sich die gebotenen Wahrnehmungen und Interaktionsmöglichkeiten auf die vom *VR-System* bereitgestellte Oberfläche begrenzen und reale Objekte davon unbeeinflusst bleiben. Abzugrenzen davon ist der Begriff der *augmented reality* (AR), der bedeutet, dass zusätzliche künstliche Reize vom Computersystem bei der Perzeption der realen Umgebung durch den Nutzer simuliert und hinzugefügt werden. „Dabei kommt es zu einer Verschmelzung der Realität mit der Virtualität“ [DBGJ13, S. 242]. Die Kombination beider Technologien nennt sich *mixed reality* (deutsch gemischte Realität). [DBGJ13, S. 241f.]

Im Laufe der Zeit wurden für die Umsetzung der Virtuellen Realität diverse Technologien mit unterschiedlich hoher *Immersion* genutzt. Bereits in den 1960er Jahren gab es Ansätze zur Simulation der realen Welt unter Einbezug verschiedener Sinnesreize. [GVT08, S. 4]

Mit dem *Sensorama* (siehe Abb. 2.1) schuf Morton Heilig 1962 eine Maschine zur Simulation der Fortbewegung mit verschiedenen Fahrzeugen. Dabei wurden durch die Darstellung von 3D-Aufnahmen von Fahrradfahrten, Motorradfahrten oder sogar Helikopterflügen und durch die Stereo-Beschallung und Generation von Wind und Gerüchen erstmals eine hohe *Immersion* und *Präsenz* erreicht. Allerdings bot diese Erfindung durch die Absenz leistungsfähiger Computer keinerlei Interaktionsmöglichkeiten und die „Nutzer war[en] [...][passive] Beobachter“ [GVT08, S. 5, Übers. durch Verfasser]. Der wirtschaftliche Erfolg blieb aus. [GVT08, Bri09]

Kurz darauf brachte Ivan Sutherland weitere Fortschritte in der Technik der Virtuellen Realität. Er schrieb zunächst 1965 ein Paper namens *The Ultimate Display*, in dem er ausführt, wie eines Tages Rechner dafür genutzt werden können, in virtuelle Welten einzutauchen [Sut65]. 1968 baute er dann das erste *head-mounted display* (HMD, deutsch kopfmontierte Anzeige oder Datenhelm, siehe Abb. 2.1). Mit diesem Gerät konnte eine simple 3D-Szene betrachtet werden. Interaktion war durch die Drehung des Kopfes möglich, welche vom HMD erfasst wurde. Damit ließ sich die Darstellung aus einem anderen Blickwinkel berechnen und anzeigen. [DBGJ13, GVT08]

Bald zog das Unterfangen die Aufmerksamkeit des amerikanischen Militärs und der *National Aeronautics and Space Agency* (NASA) auf sich. Das Militär war an günstigeren Methoden für die Flugsimulation interessiert als für jedes Flugzeugmodell einen eigenen Simulator zu bauen, der nur für die Dauer des Einsatzes des Modells relevant war. Die NASA erhoffte sich realitätsnahe Simulatoren für die Ausbildung von Astronauten, da die Herstellung von Weltraumbedingungen beinahe unmöglich war. Mit diesem Ziel wurde das VIEW-Projekt (*Virtual Interface Environment Workstation*) Anfang der 1980er Jahre ins Leben gerufen. [BC03, S. 6f.]

Zur gleichen Zeit begann das Bestreben, diese neue Technik zu kommerzialisieren. Die von Jaron Lanier und Thomas Zimmermann gegründete Firma *VPL Research* spezialisierte sich auf die Entwicklung von Hardware und Software für VR. Lanier hat den Begriff der Virtuellen Realität erstmals eingeführt und damit geprägt. [GVT08, Bri09, DBGJ13]

Das von VPL entworfene handschuhähnliche Eingabegerät *DataGlove* wurde in das VIEW-Projekt integriert. In dessen Laufzeit entstanden außerdem ein HMD mit *Liquid Crystal Displays* (LCD) und viele weitere Geräte zum Zweck der erhöhten *Immersion*. Das Ergebnis des Projektes war das erste *VR-System*, welches aber

Abbildung 2.2.: Anwendungsfall einer CAVE[1]

aufgrund des Technologie-Standes noch immer nicht zufriedenstellend war. [GVT08, Bri09, BC03, DBGJ13]

Ein weiterer Meilenstein für die Virtuelle Realität war die Erfindung von elektromagnetischer Positionsverfolgung von der Firma *Polhemus* im Jahr 1989. In den Folgejahren ging die Forschung auf alternative Darstellungsmöglichkeiten einer simulierten Umgebung ein. Dabei entstand 1992 die CAVE (*CAVE Automatic Virtual Environment*) an der University of Illinois. Dieses *VR-System* besteht aus vier Leinwänden, die von außen von Projektoren bestrahlt werden. Leistungsfähige Grafikrechner berechnen dafür *stereoskopische* Bilder (Begriffsklärung auf Seite 12). Der Nutzer befindet sich innerhalb des Projektionswürfels und erhält räumliche Sicht über eine Brille (siehe Abb. 2.2). Damit konnte auf das Tragen von schweren HMD verzichtet und zum ersten Mal eine hohe Auflösung garantiert werden. Vor allem durch den Umstand, dass sich der Benutzer darin frei bewegen und seinen eigenen Körper betrachten konnte, wurde mit CAVE eine starke *Präsenz* erzielt. Das gemeinsame Erlebnis durch mehrere Personen war ebenfalls möglich. [Bri09, GVT08, DBGJ13]

Das Tracking-System von *Polhemus* wurde später zuerst von Ultraschall-Tracking und um 2000 von Infrarot-Tracking abgelöst [DBGJ13, S. 21]. Ab diesem Zeitpunkt fand Virtuelle Realität bereits in verschiedenen Bereichen wirtschaftliche und produktive Anwendung. [Bri09, S. 12]

Einen erheblichen Anstieg der Popularität der Technologie im Privatbereich gab es seit dem Jahr 2016. Verantwortlich dafür war die Markteinführung bezahlbarer *VR-*

[1] Quelle: `https://upload.wikimedia.org/wikipedia/commons/6/6d/CAVE_Crayoland.jpg`

Systeme, namentlich *Samsung Gear VR*[2] (November 2015, $99 [VR15]), *Oculus Rift*[3] (März 2016, $599 [VR16]), *HTC Vive* (April 2016, $799 [Tea16]) und *PlayStation VR*[4] (Oktober 2016, $399 [Pla16]). Allerdings ist keines der genannten Systeme ein vollständiges *VR-System*; die Computer zur Simulation und die Simulationen (Anwendungen) selbst sind nicht im jeweiligen Produkt enthalten. Beispielweise ist *PlayStation VR* nur mit einer *PlayStation 4*[5] kompatibel und das *Samsung Gear VR* benötigt ein passendes *Samsung*[6]-Smartphone zur Simulation sowie zur Anzeige der VR-Inhalte. Wegen der ohnehin schon breiten Verfügbarkeit von leistungsfähigen Personal Computers (PC), Smartphones und Spielekonsolen ließ sich der Vertrieb einzelner *VR-Systemkomponenten* realisieren. Hauptsächlich beinhalten die Produkte ein HMD und eventuell Positionsverfolger und/oder Eingabegeräte (Controller) für die Interaktion in der VR. Zusammen mit den fehlenden Komponenten wird eine hoch *immersive* Erfahrung geboten und abhängig von der Anwendung auch eine hohe *Präsenz* erreicht.

Gegenstand der vorliegenden Arbeit war die Entwicklung einer Anwendung für das *VR-System HTC Vive*. Der folgende Abschnitt gibt einen Überblick über dessen Funktionen.

2.2. HTC Vive

Die *HTC Vive* ist ein *VR-System*, das aus der Kooperation zwischen den Firmen *HTC*[7] und *Valve*[8] hervorgegangen ist. Im April 2016 erschien die erste *Consumer Edition* [Tea16], die den Privatkonsum von VR-Anwendungen mittels des von *Valve* veröffentlichten Softwaretools *SteamVR*[9] ermöglicht. Es handelt sich hierbei um ein unvollständiges *VR-System*, da zur Nutzung zusätzlich ein leistungsfähiger PC mit der *SteamVR*-Software und den VR-Inhalten benötigt wird. Die *Valve*-eigene Vertriebsplattform *Steam* bietet dafür hauptsächlich Videospiele zum Kauf an. Damit erlangten sowohl *HTC* als Hardwarefirma als auch *Valve* als Softwarefirma einen

[2]Produktseite: https://www.samsung.com/de/wearables/gear/?gear-vr
[3]Produktseite: https://www.oculus.com/rift/
[4]Produktseite: https://www.playstation.com/de-de/explore/playstation-vr/
[5]Produktseite: https://www.playstation.com/de-de/explore/ps4/
[6]Webauftritt: https://www.samsung.com/
[7]Webauftritt: https://www.htc.com/
[8]Webauftritt: https://www.valvesoftware.com/
[9]Produktseite: https://store.steampowered.com/steamvr

Abbildung 2.3.: Komponenten der *HTC Vive* (Aus: [Tea16])

Einstieg in den VR-Markt. Das Produkt mit einem anfänglichen Kostenpunkt von 799 US-Dollar [Tea16] enthält folgende Hauptkomponenten:

- ein **Headset**, ein *head-mounted display* (siehe Abb. 2.3 Mitte) [HTC18]
- zwei **Controller**, welche den größten Teil der Interaktion mit der VR ermöglichen (siehe Abb. 2.3 rechts und links unten) [HTC18]
- zwei **Basis-Stationen**, die zur Positionsverfolgung der anderen Komponenten notwendig sind (siehe Abb. 2.3 rechts und links oben) [HTC18]

Die *HTC Vive* erlaubt die Bewegung des Nutzers in einem vordefinierten Bereich von maximal 4.5 x 4.5 Metern. Dabei wird jede Positions- und Rotationsänderung des Headsets oder der Controller präzise durch das *SteamVR Tracking* erfasst und an die Anwendung weitergegeben. Das *SteamVR Tracking* ist eine von *Valve* entwickelte Technologie zur Infrarot-Positionsermittlung und ist Bestandteil des *SteamVR*. Die beiden Basis-Stationen werden an gegenüberliegenden Ecken der VR-Zone aufgestellt und tasten diese kontinuierlich mit Infrarot-Signalen ab, welche von Sensoren in den Einbuchtungen des Headsets und der Controller (siehe Abb. 2.3 Mitte und unten) empfangen werden [NLL17, S. 3]. Die Verzögerung zwischen Signalemission und -aufnahme ist ausschlaggebend für die Rekonstruktion der Position und Ausrichtung

der Geräte [NLL17, S. 3]. Um Verletzungen oder die Zerstörung von Gegenständen im Umfeld zu vermeiden, ist das *SteamVR Tracking* mit einem Sicherheitssystem, dem *Chaperone system* (deutsch Aufpasser-System), ausgestattet. Das *Chaperone system* warnt den Anwender in der Virtuellen Realität mit einem Lichtgitter, wenn dieser die Bereichsgrenzen zu übertreten droht. Die Übersetzung von Bewegungen in der realen Welt zu Bewegungen in der virtuellen Welt ist Kern der Interaktivität des *VR-Systems*. [HTC18]

Weitere Interaktionsmöglichkeiten bieten die Controller mit ihren zahlreichen Knöpfen. Dazu zählt der *Trigger* an der Unterseite, der mit dem Zeigefinger bedient wird und bis zu seinem Tiefpunkt beliebig weit gedrückt werden kann. Damit lassen sich beispielsweise verschiedene Stufen der Handöffnung bzw. -schließung in die VR übertragen. Zwei Daumentasten auf der Oberseite, die System- und die Menütaste, sowie die *Grip*-Taste für den Mittel- und/oder Ringfinger an der rechten oder linken Seite (abhängig davon, welche Hand den Controller hält), kennen dahingegen nur zwei Zustände – gedrückt oder unbetätigt. Das runde *Trackpad* zwischen System- und Menütaste fungiert in vier Richtungen als Knopf und registriert zusätzlich die Auflageposition des Daumens auf seiner Kreisoberfläche. Wie die Tasten in der virtuellen Welt genutzt werden, hängt von der jeweiligen VR-Anwendung ab. Die Controller unterstützen außerdem haptische Reizübermittlung und somit die Bereicherung des VR-Erlebnisses um die Wahrnehmung mit einem weiteren Sinn. Die Knöpfe und das *SteamVR Tracking* erlauben freie und intuitive Aktionen im virtuellen Raum. [HTC18]

Das Headset wird am Kopf befestigt und ist für die Visualisierung der Virtuellen Realität verantwortlich. Hauptbestandteil des HMD ist der Bildschirm mit einer Auflösung von 2160 x 1200 Pixeln und einer Bildwiederholrate von 90 Hertz [HTC18]. Diesen trägt der Nutzer brillenähnlich vor dem Gesicht, wobei jedes Auge eine Hälfte der Bildschirmfläche zugeteilt bekommt und dessen Sicht nach außen komplett versiegelt wird. Mit den beiden Teilbildschirmen wird durch die Technik der *Stereoskopie* ein räumlicher Eindruck von der virtuellen Welt vermittelt. Dafür berechnet der Computer pro *Frame* zwei Bilder aus leicht versetzter Perspektive (ungefähr im Abstand zweier Augen), die auf je einer Bildschirmhälfte angezeigt werden. Das Gehirn des Anwenders verarbeitet diese dann wie gewöhnliches dreidimensionales Sehen. Mit dem *SteamVR Tracking* werden alle Kopfbewegungen und -rotationen erkannt und anschließend erfolgt die Berechnung des Bildes mit neuem Ausgangspunkt und Blickwinkel. Für den Nutzer ist also die Bewegung im Raum und das

Umsehen intuitiv möglich und die VR reagiert darauf in Echtzeit. Dadurch erzielt das *VR-System* eine hohe *Präsenz* und *Immersion*. Letztere kann mit der Nutzung von Köpfhörern mit Rauschunterdrückung erhöht werden.

Für Entwickler von VR-Inhalten besteht die Notwendigkeit, Hardwarekomponenten eines *VR-Systems* anzusprechen. Für die *HTC Vive* stellt *SteamVR* die Schnittstelle zwischen einer Anwendung und den Geräten dar. Moderne *Game Engines* wie die *Unreal Engine 4* unterstützen bereits die Entwicklung von Videospielen für die *HTC Vive*. In den Fällen übernimmt die *Engine* die Kommunikation mit *SteamVR* und bietet eigene Methoden für den Zugriff auf die VR-Komponenten. So lassen sich etwa *Input-Events* so abfragen, wie es für die jeweilige *Engine* üblich ist, oder bequem die Bewegung der Kamera oder anderer Objekte in der VR mit denen der *HTC Vive* verknüpfen. Neben dem Auslesen von Eingaben ist auch die Ansprache gewisser Gerätefunktionen, beispielsweise das Auslösen der taktilen Rückmeldung, für manche VR-Anwendungen unabdingbar und deshalb ebenfalls in die *Engines* integriert.

Die *HTC Vive* ist nach den Betrachtungen als *immersives VR-System* zu verstehen, welches den Anforderungen an die Virtuelle Realität in der modernen Unterhaltung gerecht wird. Für die Umsetzung des Vorhabens dieser Arbeit mit der *Unreal Engine* stellt es mit *SteamVR* die nötige Schnittstelle zwischen Hardware und Software zur Verfügung. Nachfolgend erhält der Leser eine grundlegende Einführung in die Entwicklung von Videospielen in der *Unreal Engine* und in die Besonderheiten bei der Entwicklung für VR.

2.3. Unreal Engine

Unreal Engine ist eine von der Firma *Epic Games*[10] veröffentlichte Entwicklungsumgebung für Echtzeitanwendungen oder auch *Game Engine*, da überwiegend Videospiele damit geschaffen werden. Als solche bietet sie dem Entwickler eine Vielzahl von Werkzeugen zur Realisierung eines Projektes. Die aktuelle vierte Iteration sowohl privat als auch kommerziell bis zu einer gewissen Gewinnhöhe kostenfrei und vollständig nutzbar. *Epic Games* selbst setzen ihre *Engine* zur Spieleentwicklung ein. So entstand unter anderem der aktuell erfolgreiche [Kai18] *third person shooter Fortnite*

[10] Webauftritt: `https://www.epicgames.com/`

Battle Royale[11], ein Schießspiel mit Sicht aus dritter Person auf den gesteuerten Charakter.

Zu den Funktionalitäten der *Unreal Engine 4* zählen ...

- ... fotorealistisches *Rendering*, d. h. Berechnung und Darstellung von 3D-Objekten, in Echtzeit ohne Performanzeinbuße. [Epi18]
- ... die Verbindung von der Programmiersprache *C++* und der visuellen Skripterstellung *Blueprint* oder die Auswahl einer Variante, um der Anwendung Funktionalität zu geben. [Epi18]
- ... diverse *Editoren* für die Erstellung von hochqualitativen visuellen Inhalten, beispielsweise *Materialien* oder *Partikelsysteme*. [Epi18]
- ... *Editoren* für Animationen und Videosequenzen. [Epi18]
- ... die Unterstützung von Virtueller und Augmentierter Realität [Epi18], vor allem von *SteamVR*, was eine Voraussetzung für die Umsetzung des prototypischen Rhythmik-Videospiels für die *HTC Vive* ist.
- ... ein System für die Nutzung von AI (*artificial intelligence*, deutsch künstliche Intelligenz). [Epi18]
- ... die Erweiterbarkeit der *Engine* durch Plugins und Eingriff in den Quellcode. [Epi18]

Es ist zu erkennen, dass es sich hierbei um eine sehr umfangreiche Entwicklungsumgebung handelt, die sich laut *Epic Games* weltweit bewährt [Epi18]. Manche der Features sind in vergleichbaren *Game Engines* gar nicht oder nur über Plugins, die nicht von den Entwicklern der *Engine* gepflegt werden, nutzbar.

Die Installation der *Unreal Engine 4* erfolgt über ein Programm namens *Epic Games Launcher*[12] (siehe Abb. 2.4), welches die zentrale Plattform für die Installation von Produkten der Firma ist. Dort ist neben dem Download der *Engine* auch der Zugriff auf Neuigkeiten oder Webseiten für Meinungsaustausch und Hilfestellungen über den Reiter *Community* möglich. Eine Anbindung an alle von *Epic Games* bereitgestellten Lerninhalte gibt es unter *Learn*. Beim *Marketplace* können Entwickler nützliche

[11] Produktseite: `https://www.epicgames.com/fortnite/`

[12] Download: `https://launcher-public-service-prod06.ol.epicgames.com/launcher/api/installer/download/EpicGamesLauncherInstaller.msi`

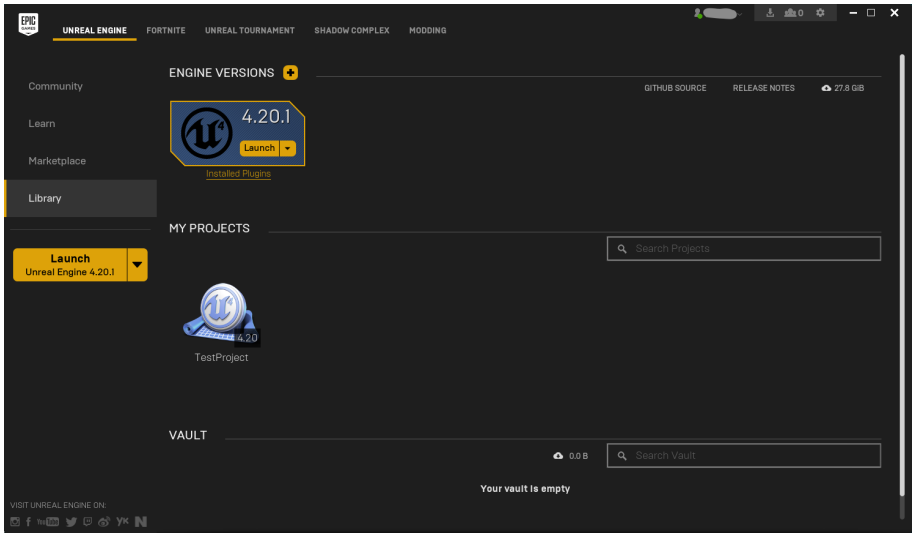

Abbildung 2.4.: Der Reiter *Library* der *Unreal Engine* im *Epic Games Launcher*

Inhalte kaufen, verkaufen oder kostenlos herunterladen. Damit lässt sich bei der Entwicklung Zeit sparen, indem beispielsweise generische Klangeffekte nicht selbst geschaffen werden müssen. Im Reiter *Library* (siehe Abb. 2.4) sind die installierten *Engine*-Versionen sowie alle vorhandenen *Projekte* und gekauften Inhalte einzusehen. Die aktuelle Version der *Unreal Engine* hat die Nummer 4.20.3 und findet in dieser Arbeit Verwendung. Frühere Versionen können über das „+“-Symbol hinzugefügt werden und auf einem Rechner gleichzeitig installiert sein. Beim Umgang mit mehreren *Projekten* ist das essentiell. Ein *Projekt* ist in diesem Zusammenhang eine Einheit, die mitunter alle Mediendateien und die Programmlogik einer Anwendung strukturiert bündelt und als Schnittstelle zwischen *Engine* und Dateisystem dient. Die Bearbeitung eines *Projektes* in einer anderen als der zu seiner Erstellung genutzten Version der *Engine* ist nicht empfehlenswert und bringt anfänglichen Aufwand mit sich.

Die Hauptarbeitsfläche in der *Unreal Engine 4* ist der *Level-Editor*. Dieser wird standardmäßig mit dem *Projekt* geöffnet. Es folgt eine Erklärung aller in Abbildung 2.5 nummerierten Teilbereiche des *Level-Editors*:

1. Die **Tab- und Menüleiste** sind immer am oberen Rand zu finden. Die *Tabs* (deutsch Registerkarten oder Reiter) sind neben dem Logo aufgelistet und repräsentieren in Benutzung befindliche *Editoren*. Ein *Editor* ist in der *Unreal Engine 4* ein Fenster, das Funktionen zur Erstellung oder Anpassung

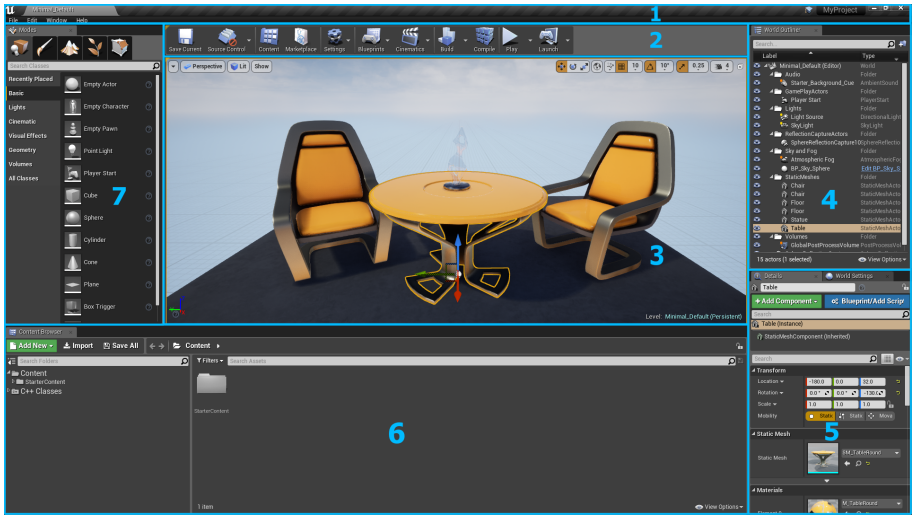

Abbildung 2.5.: *Level-Editor* der *Unreal Engine 4*

bestimmter Anwendungsinhalte zur Verfügung stellt. Der zugehörige *Tab* trägt den Namen des in Bearbeitung befindlichen Inhaltes. So ist der Reiter für den *Level-Editor* nach einem *Level* benannt. *Levels* oder auch *Maps* sind Dateien, die Informationen über die Positionierung von Objekten im Raum sowie deren Eigenschaften und Beziehungen zueinander definieren. Mit einem *Level* lässt sich zum Beispiel die Simulation für eine Virtuelle Realität umsetzen. Außerdem gibt die *Tableiste* den Namen des *Projektes* an und erlaubt den Abruf von Tutorials innerhalb der *Engine*. Die Menüleiste enthält Menüpunkte zugeschnitten auf den jeweils geöffneten *Editor*.

2. Die **Werkzeugleiste** bietet wichtige Funktionen zur Arbeit mit dem geöffneten *Editor* in Form von übersichtlichen Knöpfen mit Bild und Text. Hier gibt es unter anderem die Möglichkeit, Änderungen am *Level* zu speichern, Qualitätseinstellungen vorzunehmen, Beleuchtung neu zu berechnen, den *C++*-Code zu kompilieren oder das *Level* zu simulieren oder zu testen. Ein kleiner Pfeil rechts neben einem Knopf gewährt Zugriff auf alternative Aktionen oder Zusatzoptionen.

3. Im **Viewport-Panel** erfolgt die Bearbeitung des *Levels*. In einem oder mehreren *Viewports* werden die Weltobjekte auf unterschiedliche Weise dargestellt. Das *Layout* legt fest, wie viele *Viewports* in welcher Anordnung gezeigt werden, sofern nicht eines davon auf die Größe des *Viewport-Panels* maximiert ist. Die Menüs in der linken Hälfte des oberen Randes sind neben der *Layout-*

Einrichtung zum Großteil für die Einstellung der Projektionsart und für das Filtern der Anzeige gedacht. So kann etwa in einem *Viewport* die *Map* perspektivisch und in voller Beleuchtung dargestellt werden und in einem anderen mittels einer Draufsicht auf Drahtgittermodelle. Das *Viewport-Panel* wird insbesondere dazu verwendet, sichtbare und unsichtbare Objekte in das *Level* zu platzieren und sie über Translation, Rotation und Skalierung mittels sogenannter *Handles* (deutsch Anfasser, siehe bunte Pfeile am Tischfuß in Abb. 2.5) zu manipulieren. Einstellungen für diese Manipulationsarten – wie die diskrete anstelle der kontinuierlichen Wertänderung – werden in der rechten Hälfte des oberen *Viewport*-Randes vorgenommen. In der *Unreal Engine* tragen diese Objekte den Namen *Actor* und erben von der gleichnamigen *C++*-Klasse, welche die Grundlogik für die Existenz in einem *Level* mit sich bringt. In der unteren linken Ecke befindet sich ein Referenz-Weltkoordinatensystem für den simulierten Raum und in der unteren rechten Ecke eine Aufführung des Namens des im *Viewport* dargestellten *Levels*. Die Navigation im aktiven *Viewport* ist mit Maus und Tastatur möglich. Das *Viewport-Panel* ist außerdem die standardmäßige Anzeige für das Testen der Anwendung, wobei auch das Öffnen eines gesonderten Fensters für diesen Zweck bevorzugt werden kann.

4. Die Hierarchie der im *Level* befindlichen *Actors* ist in der **World Outline** abgebildet. *Actors* können dort so miteinander verknüpft werden, dass sich ein *Actor* dem anderen unterordnet und sich in gleichem Maße ändert, sofern eine Manipulation auf dem übergeordneten *Actor* stattfindet. Eine solche Hierarchie trägt die Bezeichnung *parent child relationship* (deutsch Eltern-Kind-Beziehung) und ist per Ziehen und Loslassen einzurichten. Es ist zudem möglich, Ordner in der *World Outline* anzulegen, um *Actors* zu gruppieren. Die Suchleiste am oberen Rand erlaubt eine Filterung aller Einträge nach einem eingegebenen Begriff und die darunterliegende Leiste das Sortieren nach den aufgeführten Kategorien. Die Augensymbole auf der linken Seite repräsentieren die Sichtbarkeit der *Actors* im *Level* bzw. im *Viewport*. Bestimmte Unterklassen von *Actor* haben in der *World Outline* vor ihrem Namen spezielle Symbole passend zu deren Funktion. Ein Beispiel dafür ist das Lautsprechersymbol für Audio-*Actors*. Die *World Outline* ist insoweit mit anderen Komponenten des *Level-Editors* verbunden, dass die Auswahl eines Eintrages zur Auswahl des entsprechenden *Actors* in allen anderen Bereichen führt und umgekehrt.

Zum Beispiel wird beim Klick auf den Tisch im *Viewport* dieser auch in der Gliederung angewählt.

5. Unter der World Outline befindet sich das **Details-Panel**. Wie der Name bereits andeutet, ist diese Komponente für die Anzeige von Details, genauer genommen die der *Actors*, zuständig. Sobald in einer anderen Komponente ein *Actor* selektiert wird, werden hier alle seine änderbaren Eigenschaften aufgelistet. Im oberen Teil befinden sich Knöpfe für das Hinzufügen von *Components* und für die Konvertierung des *Actors* in eine *Blueprint-Klasse*. *Components* und *Blueprints* (deutsch Blaupausen) sind wichtige Konzepte beim Umgang mit der *Unreal Engine*. *Components* erweitern die Funktionalität eines *Actors* und können ohne diesen nicht existieren. Standardmäßig besitzen *Actors* eine *SceneComponent* oder eine *Component*, welche von *SceneComponent* erbt. Diese versorgt den *Actor* mit den nötigen Daten zur Koordinatentransformation im *Level*. *Blueprint-Klassen*, kurz *Blueprints*, sind Dateien, die *Actors* und ihre *Components* zu einer Einheit zusammenfassen und deren Wiederverwendung ermöglichen. Außerdem kann einem *Blueprint* Programmlogik mit Hilfe der visuellen Skripterstellung angefügt werden. Die Bearbeitung von *Blueprint-Klassen* erfolgt im gesonderten *Blueprint-Editor*. Handelt es sich beim betrachteten *Actor* um einen *Blueprint*, so wird im *Details-Panel* der blaue Knopf zur Konvertierung durch einen Knopf zur Öffnung des *Blueprint-Editors* ersetzt. Unter den Knöpfen befindet sich eine Suchleiste, welche die darunterliegende Übersicht über die dazugehörigen *Components* nach einer Eingabe filtern lässt. Die *Component*-Übersicht ist wie die *World Outline* hierarchisch aufgebaut und führt neben den *Components* auch deren *Actor* auf. Die erste und oberste *Component* wird als *RootComponent* (deutsch Wurzelkomponente) bezeichnet und definiert die Transformation des gesamten *Actors* in der *Map*. Unter der Gliederung sind alle variablen Eigenschaften der jeweils markierten *Component* oder des *Actors* innerhalb klappbarer Kategorien verzeichnet. So ist nach Auswahl des Tisch-*Actors* eine Verschiebung dessen über *Location* in der Kategorie *Transform* möglich. Eine Suchleiste sowie weitere Filtermöglichkeiten sind hier ebenfalls vorhanden.

6. Der **Inhaltsbrowser** ermöglicht den Zugriff auf das *Projekt* im Dateisystem. Es können neue Dateien hinzugefügt und Ordnerinhalte eingesehen werden. Das Einbringen von außerhalb des *Projektes* existierenden Inhalten ist über den *Import*-Knopf möglich. Mit *Save All* lassen sich Änderungen auf allen

Dateien gleichzeitig speichern. Die Navigation durch die Ordnerstruktur verhält sich wie im *Windows-Explorer*. Beim Doppelklick auf kompatible Dateien öffnet sich ein geeigneter *Editor* der *Engine*, wobei für *C++*-Klassen in den *Projekt*-Einstellungen ein externer *Code-Editor* festgelegt ist. Sowohl *Actors* als auch *Blueprints* sind per Ziehen und Loslassen im *Level* instanzierbar.

7. Das **Modes-Panel** stellt verschiedene Modi zur Bearbeitung der *Map* zur Verfügung. Es besitzt insgesamt fünf Tabs, die unterschiedliche Werkzeuge darstellen. Der erste Reiter wird für die Platzierung von *Actors* und *Blueprints* in das *Level* genutzt. Dies funktioniert wie im Inhaltsbrowser, allerdings sind hier alle platzierbaren Inhalte nach Kategorien vorsortiert. Die weiteren Registerkarten sind für die Texturierung und Kolorierung von Objekten, das Sculpting von Landschaften, die Kreation von Vegation und die Anpassung von Objektgeometrien zuständig.

Abschließend ist anzumerken, dass es sich bei den eben vorgestellten Komponenten des *Level-Editors* nicht um den vollen Umfang handelt. Es können weitere nützliche Werkzeuge, wie beispielsweise der *Sequencer* zum Anlegen und Bearbeiten von Videosequenzen, an beliebige Stellen des *Editors* gelegt werden oder als *Tab* einem bereits genutzten Teilbereich angefügt werden, wobei immer nur ein *Tab* aktiv die Fläche einnimmt. Es ist zudem in vielen Fällen möglich, über einen Rechtsklick kontextsensitive Menüs aufzurufen. So können beispielsweise im *Inhaltsbrowser* mit einem Rechtsklick neue Inhalte wie *C++*-Klassen hinzugefügt werden. Die Nutzung fremder *C++*-Softwarebibliotheken ist durch das Kopieren des Quellcodes in die *Projektordner* ohne Probleme möglich, sofern sie nicht auf andere Bibliotheken verweisen.

Die Entwicklung von VR-Anwendungen für die *HTC Vive* wird von der *Unreal Engine 4* seit einigen Versionen unterstützt. Dafür ist bereits das *SteamVR*-Plugin vorinstalliert. Um eine Vorschau des *Levels* in VR freizuschalten, muss zunächst das *SteamVR* auf dem System installiert und gestartet werden und die Komponenten der *HTC Vive* angeschlossen und mit *SteamVR* eingerichtet werden. Dann erkennt die *Engine* automatisch die Hardware und unter dem *Play*-Knopf in der Werkzeugleiste erscheint die VR-Vorschau-Option, mit der die *Map* getestet werden kann. Für den Zugriff auf die Geräte über *C++*-Code oder *Blueprints* stellt die *Unreal Engine* unter anderem die *Components Motion Controller* und *Steam VRChaperone* zur Verfügung.

Die *Unreal Engine 4* ist somit eine geeignete Entwicklungsumgebung für die Erstellung des Rhythmik-Videospiels unter dem Gesichtspunkt, dass es für die Virtuelle Realität und unter Einbezug einer freien Softwarebibliothek zur Audioanalyse entwickelt wird.

2.4. Rhythmik-Videospiele

Die Videospieleindustrie hat ihre Anfänge im Jahr 1961, in dem das erste interaktive Computerspiel *Spacewar* von Steve Russell entwickelt wurde. In einer Zeit, in der Spielehallen gut besucht waren, wurde damit experimentiert, die spielerischen Unterhaltungsmöglichkeiten auf neuartige Medien auszubreiten. Einen großen Erfolg konnte dabei die Firma *Atari* mit dem Spieleautomaten *Pong* in den 1970er Jahren erzielen. Mit dem Vertrieb von Spielekonsolen und *Personal Computers* wurden Videospiele für den Konsum im Privathaushalt immer beliebter. Gleichzeitig erschienen immer neue computergestützte Spieleautomaten. Der Markt wuchs und es entstanden eine Vielzahl verschiedener Computerspielegenres. Eines davon ist das der Musikvideospiele, auch Audiospiele genannt. [Ken10, PK07]

Musikvideospiele sind Computerspiele, die als zentrales Konzept das Erleben von Musik oder Klängen beinhalten. Es lässt sich weiterhin in zwei Kategorien teilen: Rhythmik-Videospiele und elektronische Musikinstrumente. Letztere zeichnen sich durch ein Prinzip der spielerischen Erschaffung oder Entdeckung von Musik aus, bei welcher das Spiel die Rolle eines digitalen Instrumentes einnimmt, auf welchem der Nutzer auf eine oder mehrere Weisen Klänge erzeugen kann. Ein Beispiel dafür ist *Electroplankton*. Rhythmik-Videospiele hingegen geben dem Spieler die Musik und den Rhythmus vor und die Aufgabe ist, diesen zu verstehen und darauf zu reagieren. So ist beispielsweise Kern des berühmten Spiels *Guitar Hero* das Drücken verschiedener Tasten zum richtigen Zeitpunkt im vorgegebenen Rhythmus. Die Rhythmen sind abhängig vom gewählten Lied und dem Schwierigkeitsgrad variabel in ihrem Tempo und ihrer Komplexität. Dabei ist messbar, wie gut der Spieler sich an den geforderten Rhythmus hält. Daraus ergibt sich ein Bewertungssystem, welches den Vergleich der Leistung mehrerer Spieler erlaubt. Diese Eigenschaften sind typisch für Rhythmik-Videospiele, zu denen auch *Beat Saber* gehört, welches die Motivation für die vorliegende Arbeit liefert (siehe Kapitel 1). [PK07]

2.5. Technische Akustik und Medienkodierung

Damit eine automatische Levelgenerierung mittels der Analyse von Musikstücken entwickelt werden kann, ist eine Kenntnis über den Aufbau von Audiodateien und über wichtige Konzepte der Musik vonnöten. Dieses Kapitel soll das Wissen vermitteln und eine Grundlage für die Wahl einer geeigneten Audioanalyse-Bibliothek und die Übersetzung extrahierter Merkmale in Levelinhalte schaffen.

Es gibt verschiedene wahrnehmbare Elemente, die Musik ausmachen. Obwohl Musik grundlegend nur aus aneinandergereihten und überlagerten Tönen besteht, geben deren Eigenschaften und Beziehungen zueinander erst eine gewisse Tiefe. Ein Musikstück ist sozusagen aus vielen Parametern in unterschiedlicher Ausprägung zusammengesetzt, die ein Komponist sorgfältig wählt. Diese Ausprägungen der Eigenschaften können von Menschen durch das Hören von Musik begriffen oder eindeutig bestimmt werden. Somit könnten sie als Basis für die Generierung eines Levels für ein Rhythmik-Videospiel passend zu einem Musikstück dienen, sofern die Bestimmung der Parameter ebenfalls durch Computer möglich ist. Es folgt die Erklärung einiger Grundelemente der Musik:

Die *Tonhöhe* oder *Tonlage* gibt einem gespielten Ton eine hörbare Qualität und steht in direktem Zusammenhang mit der Frequenz der Schwingung eines Klangkörpers. Eine Abfolge von einzelnen Tönen in unterschiedlichen oder gleichen *Tonhöhen* mit einem gewissen *Rhythmus* bildet die *Melodie* [VD62, S. 68f.]. Dabei ist der *Rhythmus* das zeit- oder *taktschlag*abhängige Platzierungsmuster der Töne. Ein *Taktschlag* ist das strukturelle Grundelement eines Musikstückes. Das Muster mehrerer aufeinanderfolgender starker und schwacher *Taktschläge* stellt ein Gitter für die Ausrichtung der *Rhythmen* dar, das als *Takt* bezeichnet wird [VD62, S. 51]. Der *Takt* ist nur hörbar, wenn Töne auf den *Taktschlägen* gespielt werden. Die Beziehung zwischen dem *Takt* und der tatsächlich genutzten Zeit stellt das *Tempo* her, welches vorgibt, in welchen Zeitabständen Taktschläge auftreten [VD62, S. 60]. Eine gängige Maßeinheit des *Tempos* ist *beats per minute* (kurz BPM), welche die Anzahl der *Taktschläge* pro Minute angibt. *Melodien* können sich wiederholen oder in einem Musikstück können mehrere verschiedene *Melodien* vorkommen. Eine gewisse Betonung wird der Musik durch *Dynamik* gegeben. *Dynamik* beschreibt die Lautstärke eines Abschnittes und kann sich über ein Stück hinweg ändern, um den gewünschten Eindruck zu vermitteln. Wenn mehrere passende Töne gleichzeitig klingen, entsteht ein Klang, der als *Harmonie* bezeichnet wird. *Harmonien* werden genutzt, um *Melodien* zu bereichern.

Ein wichtiges Element der Gesamtkomposition ist die *Textur*. Sie beschreibt, in wie vielen Ebenen die Gesamtkomposition aufgebaut ist und wie diese Ebenen melodisch genutzt werden. Ein Beispiel dafür ist eine *homophone Textur*, die aussagt, dass das Stück aus *Hauptmelodien* untermalt durch *Harmonien* besteht. Ein wesentlicher Teil der Wahrnehmung von Musik wird durch das *Timbre* oder die *Klangfarbe* von Tönen beeinflusst. Es handelt sich hierbei um eine weitere hörbare Qualität, welche die Unterscheidung verschiedener Instrumente und Stimmen ermöglicht. Das *Timbre* entsteht durch die Überlagerung vieler Frequenzen bei der Bildung eines Tons abhängig vom Klangerzeuger. [Est18, Duc12]

Welche dieser Elemente in welcher Güte von Computern bestimmt werden können, ist jedoch noch nicht geklärt. Um zu verstehen, wie Musik digital aufbereitet und technisch erzeugt werden kann, muss zunächst ein Verständnis dafür entwickelt werden, wie der Mensch Klänge wahrnimmt. *Schallwellen*, ausgelöst durch Schwingungen eines Klangerzeugers, werden vom Trommelfell im Ohr über Gehörknöchel in das Innenohr weitergeleitet, wo unterschiedliche Frequenzen unterschiedliche Reize erzeugen, die das Gehirn verarbeitet und interpretiert [Zen06, LSW09]. In erster Linie werden dabei die *Tonlage*, die Lautstärke und die *Klangfarbe* wahrgenommen. Die *Schallwellen* schwingen entlang der Ausbreitungsrichtung und werden deshalb als *Longitudinalwellen* bezeichnet. Der Luftdruck an einem Punkt im Ausbreitungsraum wird dabei periodisch geringfügig höher und niedriger [LSW09, S. 7]. Der zeitliche Zustand der Schwingung einer *Schallwelle* an einem Ort lässt sich in einem Oszillogramm abbilden. Dafür wird der Druck über die Zeit aufgetragen. Im Falle eines *reinen Tons*, auch *harmonische Schwingung* genannt, würde eine einfache Sinuskurve entstehen. Die Amplitude, also der Druckwert an den Kurvenmaxima, ist ausschlaggebend für die wahrgenommene Lautstärke des Tons, und die Frequenz, reziprok zur Dauer einer Schwingung, bestimmt die *Tonhöhe*. Da die meisten musikalischen Klänge allerdings keine *reinen Töne* sind, liegt üblicherweise eine komplexe Schwingung vor, die als Überlagerung vieler *harmonischer Schwingungen* unterschiedlicher Amplituden und Frequenzen verstanden werden kann [Dem17, S. 338]. Die wahrgenommene Tonhöhe wird durch die niedrigste vorkommende Frequenz, den *Grundton*, vorgegeben und zusammen mit den überliegenden höheren Frequenzen, den *Obertönen*, ergibt sich der *musikalische Ton*, der dem Hörenden den Eindruck des *Timbres* vermittelt. [Ack13, S. 7ff.]

Die drei hörbaren Komponenten der Musik lassen sich also vollständig physikalisch beschreiben. Es stellt sich nun die Frage, wie ein Rechner die benötigten Infor-

mationen aus dem vorliegenden kontinuierlichen und komplexen Signal gewinnen kann, um Rückschlüsse auf die Parameter der Komposition ziehen zu können. An erster Stelle muss dafür ein Musikstück in einem geeigneten Format vorliegen. Eine Umwandlung des analogen Signals in ein digitales Signal ist vonnöten. Hierfür wandelt beispielsweise ein Mikrofon die *Schallwellen* in ein analoges elektrisches Signal um, das anschließend von einem sogenannten *analog-to-digital converter* (ADC) abgetastet wird. Das nachfolgend beschriebene Verfahren trägt die Bezeichnung *Pulse Code Modulation* (PCM). Die *Abtastung*, auch *Sampling* genannt, funktioniert üblicherweise so, dass in immer gleichem Zeitabstand, der *Abtastperiode*, der Wert der Auslenkung bzw. der korrespondierenden elektrischen Spannung gelesen wird. Anschließend wird mit dem Verfahren der *Quantisierung* der mögliche Wertebereich in eine endliche Anzahl von Intervallen eingeteilt und der Signalwert einem Intervall zugeordnet. Es liegt ein zeit- und wertdiskretes Signal vor. Der entstehende *Quantisierungsfehler* kann durch eine entsprechend feine Einteilung bis zur Unhörbarkeit vermindert werden. Um das Signal später etwa für die Ausgabe aus Lautsprechern wieder fehlerfrei rekonstruieren zu können, muss die *Abtastrate*, der Kehrwert der *Abtastperiode*, mindestens doppelt so groß wie die maximal auftretende Frequenz sein. In der Praxis kommt ein Faktor von circa 2,2 zum Einsatz. Zuletzt erfolgt die *Kodierung* des digitalen Signals. Hierbei wird jedem *Quantisierungsintervall* ein *Codewort* aus einer bestimmten Anzahl von Bits zugeschrieben. Die entstandene Kette von *Codewörtern* kann dann zusammen mit nötigen *Metadaten* auf einem Computer gespeichert werden. *Metadaten* enthalten unter anderem direkte oder indirekte Informationen über die *Abtastrate* und die Einteilung von *Quantisierungsintervallen*. Letztere ist auf verschiedene Arten möglich. So können etwa alle Intervalle gleich groß sein (lineare PCM) oder der Bereich niedriger Werte feiner unterteilt sein als der hoher Werte (zum Beispiel logarithmisch bei der dynamischen PCM). Damit kann die Anzahl der benötigten Bits zur *Kodierung* bei geringem Qualitätsverlust verringert werden. Weiterhin ist die Kompression der Daten möglich, auf die jedoch nicht eingegangen wird. Ein passender *digital-to-analog converter* (kurz DAC) kann schließlich das Ursprungssignal wiederherstellen. [MS09, Neu05]

Liegt dem Computer ein Musikstück in Form einer Audiodatei vor, kann er also daraus Informationen über den Verlauf der Schwingungsauslenkung lesen. Es könnte vermutlich eine einhüllende Kurve berechnet werden, mit der sich Aussagen über Abschnittswiederholungen oder Lautstärkeänderungen im Stück treffen lassen. Mit Hilfe eines geeigneten Dekodierers ließe sich auch näherungsweise die analoge

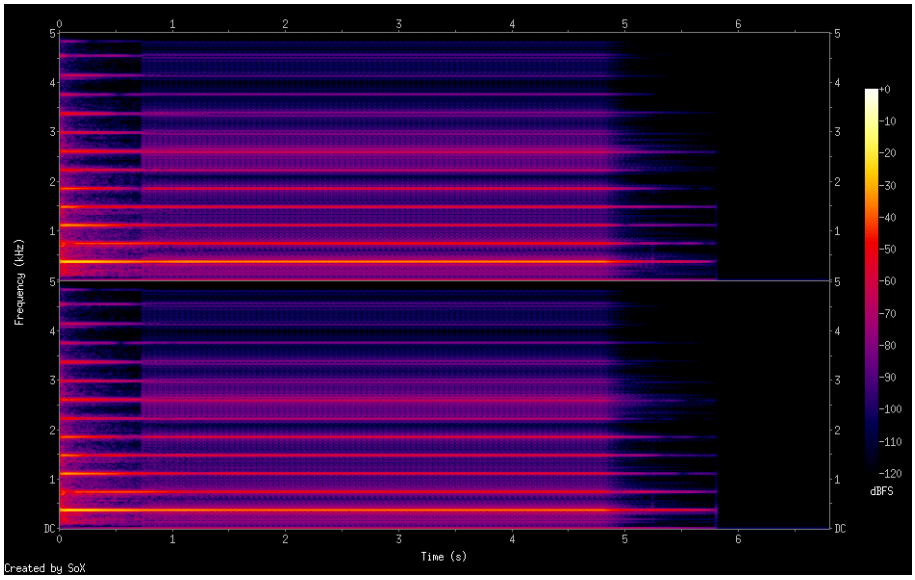

Abbildung 2.6.: Zwei *Frequenzspektren* eines mit einem Flügel gespielten Tones, dargestellt in einem *Spektrogramm* bzw. *Sonagramm*. Die Frequenzanteile sind über der Zeit aufgetragen. Die Färbung eines Punktes gibt die Lautstärke der jeweiligen Frequenz an.[13]

Signalkurve rekonstruieren und analysieren. Die Betrachtung *musikalischer Töne* ist allerdings nicht trivial. Um beispielsweise *Melodien* zu identifizieren, müsste die Aufspaltung des Signals in seine *Grund-* und *Obertöne* erfolgen, denn nur die *Grundtöne* beeinflussen die Wahrnehmung der *Tonlage*. Für analoge Signale käme dafür die nach dem Mathematiker Fourier benannte *kontinuierliche Fourier-Transformation* in Frage. Sie erlaubt das Auffächern des Signals in sein *Frequenzspektrum*. Eine mögliche grafische Darstellung dieses *Frequenzspektrums* ist in Abbildung 2.6 zu sehen. Die Umwandlung der Audiodatei in ein analoges Signal und die Anwendung der *kontinuierlichen Fourier-Transformation* ist jedoch nicht nötig, da es auch eine Methode für digitale Signale gibt: die *diskrete Fourier-Transformation* (DFT). Ein für Computer optimierter Algorithmus der DFT nennt sich *fast Fourier transform* (FFT, deutsch *schnelle Fourier-Transformation*). Herauszufinden, mit welcher Güte existierende Software-Bibliotheken mit diesen oder anderen Verfahren eine Analyse von Musikstücken ermöglichen, ist Bestandteil des praktischen Teils dieser Arbeit, welcher in den kommenden Kapiteln beleuchtet wird. [Ack13, Neu05, Dem17]

[13]Quelle: `http://thoughtmountain.com/orchestra/FS_00.png`

3. Anforderungen und Spezifikation

Dieses Kapitel befasst sich mit den Anforderungen an die prototypische Anwendung sowie die darauf basierende Gestaltung des Programmaufbaus. Der Prototyp wird im Zuge dieser Arbeit von einer Einzelperson entwickelt. Auf eine Kategorisierung der Vorgehensweise anhand der gängigen Softwareentwicklungsmethoden (beispielsweise *Wasserfall* oder *SCRUM*) und die strikte Einhaltung dessen wird verzichtet. Stattdessen kommt eine flexible Abfolge von Konzeption, Implementierung und Tests einzelner Systemkomponenten zum Einsatz, ähnlich zum agilen Modell. Um einen Grobentwurf der Systemarchitektur zu erstellen, ist zunächst eine Analyse sämtlicher sich aus der Aufgabenstellung ergebenden funktionellen und nicht-funktionellen Anforderungen vonnöten. Die Betrachtung des Rhythmik-Videospiels *Beat Saber* spielt dabei eine wesentliche Rolle.

3.1. Anforderungsanalyse

Die Anwendung dient ausschließlich dem Zweck der Unterhaltung und richtet sich im Wesentlichen an eine einzige Zielgruppe: bewegungsfähige Menschen. Der Prototyp muss deshalb nur einen grundlegenden Ablauf aufweisen, der für alle Nutzer gleich ist. Daraus ergibt sich ein Anwendungsfall mit einer bis auf wenige Abzweigungen linearen Abfolge von Aktionen (siehe Abb. 3.1).

Aus diesem Anwendungsfall lässt sich auf folgende **funktionellen Anforderungen** rückschließen:

(F1) Die Anwendung bietet ein Menü in der VR, über welches der Spieler Spielelemente oder mögliche Aktionen wählt.

(F2) Das Menü enthält ein Fenster zur Bestätigung des Spielstarts.

(F3) Das Menü enthält ein Fenster zur Wahl eines Musikstückes für das Spiel.

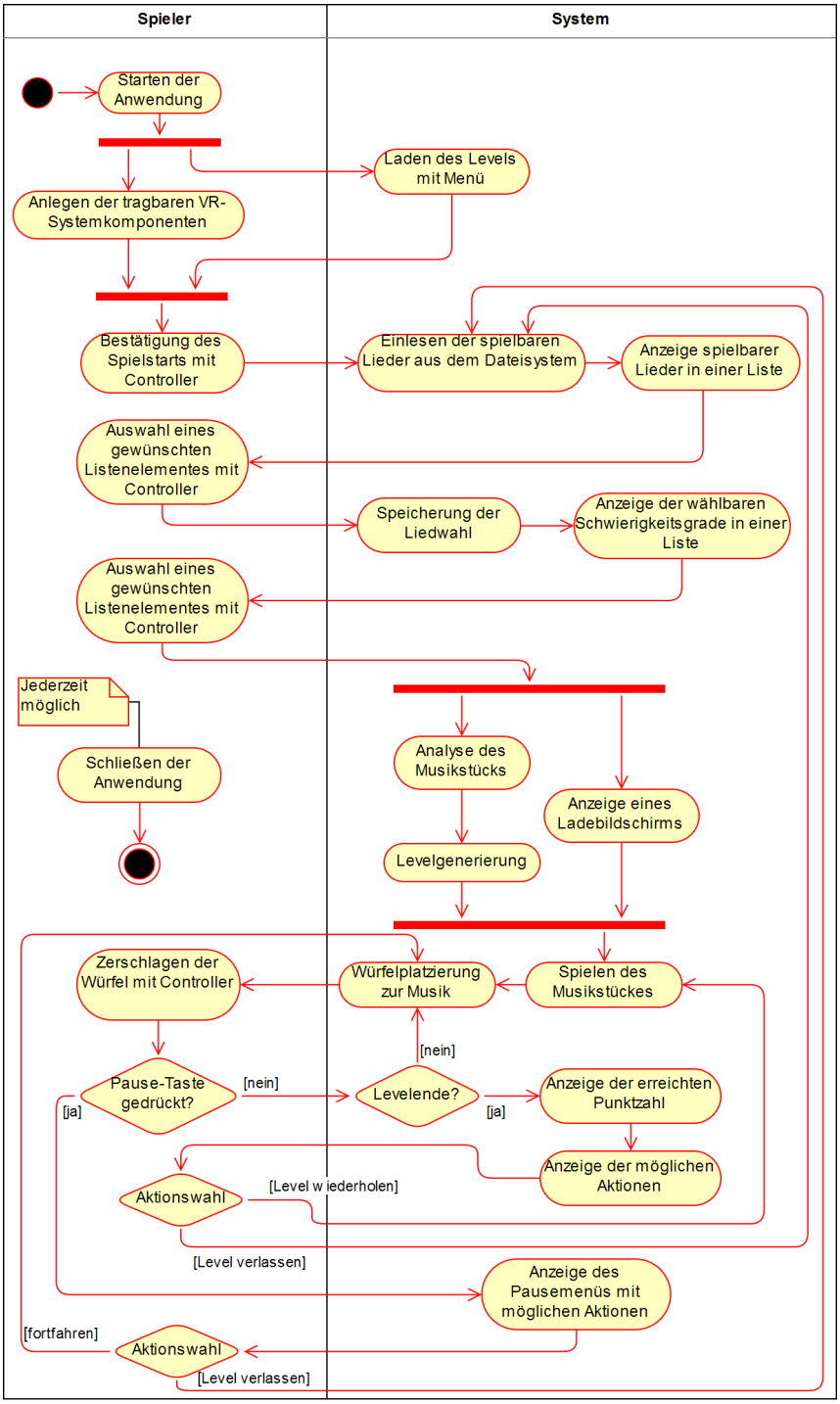

Abbildung 3.1.: Aktivitätsdiagramm des Anwendungsfalls „Spielen des Rhythmik-Videospiels“ (Angefertigt mit: https://www.draw.io/)

(F4) Das Menü enthält ein Fenster zur Wahl des Schwierigkeitsgrades für das Spiel.

(F5) Die Anwendung erlaubt das Pausieren einer laufenden Spielrunde und das Fortfahren sowie den Abbruch dieser.

(F6) Die Behandlung von Tastendruckeingaben des Nutzers erfolgt kontextsensitiv.

(F7) Das Spiel bewertet den Erfolg des Spielers mit Punkten.

(F8) Das Spiel erlaubt das Wiederholen einer Spielrunde mit gleichen Levelgenerierungsparametern.

(F9) Die Anwendung besitzt eine Schnittstelle zum Dateisystem, über die mindestens eine Art von Musikdatei eingelesen werden kann.

(F10) Die Anwendung kann mindestens eine Art von Musikdatei mit Audio-Ausgabegeräten abspielen.

(F11) Die Anwendung unterstützt die Vorerstellung von Schwierigkeitsgraden über die *Unreal Engine* durch Wahl der Ausprägung gewisser Spiel- und Levelgenerierungsparameter.

(F12) Die Anwendung kann Musikstücke analysieren und daraufhin für die bevorstehende Spielrunde entscheiden, zu welchem Zeitpunkt und mit welchen Parametern Würfel zum Zerschlagen erscheinen.

(F13) Die Anwendung verfolgt die Bewegungen des HMD und der Controller und bildet diese passend in der Virtuellen Realität ab.

(F14) Das Spiel platziert im unpausierten Zustand Würfel wie bei der Levelgenerierung festgelegt und ermöglicht das Zerschlagen dieser mit der VR-Repräsentation der Controller auf eine bestimmte Weise.

(F15) Die Bewegung der Würfel und die laufende Musik werden synchronisiert.

(F16) Die Anwendung beinhaltet einen Ladebildschirm in der VR, der dem Spieler signalisiert, dass ein Level asynchron generiert wird.

Da es sich bei der Anwendung um einen Prototyp handelt, wird kein hoher Qualitätsanspruch, der über die Grundfunktionalität hinausgeht, an die einzelnen Spielelemente gestellt. Die Anzahl der **nicht-funktionellen** Anforderungen bemisst sich deshalb sehr knapp:

(NF1) Die Levelgenerierung führt bei der wiederholten Eingabe ein- und desselben Musikstückes stets zum gleichen Ergebnis.

(NF2) Bei gleichzeitiger Platzierung mehrerer Würfel erlaubt die Levelgenerierung keine Parameter, die dazu führen, dass ein Zerschlagen aller Würfel gleichzeitig unmöglich ist.

(NF3) Zerschlagbaren Würfeln ist anzusehen, auf welche Weise sie zu zerstören sind.

(NF4) Der Spieler hat mindestens einen Orientierungspunkt in der Virtuellen Realität.

(NF5) Während des Spielens kommt es nicht zum Stillstand der ausgegebenen Bilder im HMD, damit die Gesundheit des Spielers nicht beeinträchtigt wird.

(NF6) Reaktionszeiten auf Nutzereingaben im Menü liegen in 95 Prozent aller Fälle unter einer halben Sekunde.

(NF7) Die Levelgenerierung dauert nicht länger als 20 Sekunden.

Die Überprüfung der nicht-funktionellen Anforderungen erfolgt nicht automatisiert und unterliegt der Subjektivität des Anwenders.

3.2. Softwarearchitektur

Die Umsetzung der Anforderungen in einer prototypische Anwendung bedarf einer Grobkonzeption der Systemarchitektur. Damit werden sowohl die Implementierung als auch die spätere Wartbarkeit und Erweiterbarkeit vereinfacht. Die Definition von Modulen bzw. Funktionsbereichen der Programmlogik, welche sich klar voneinander unterscheiden, und deren Interaktionen untereinander schafft dem Entwickler einen guten Überblick für das weitere Vorgehen. Abbildung 3.2 ist eine schematische Darstellung der prototypischen Softwarearchitektur, die dem Prototyp zugrunde liegt. Aus ihr ist ersichtlich, dass die Anwendungslogik in fünf Funktionsbereiche

eingeteilt ist. Die Klassennamenpräfixe „A“ und „U“ werden von der *Unreal Engine* für bestimmte Klassen benötigt. Bei allen Klassen, die von *UObject* oder dessen Unterklasse *AActor* erben, wird der entsprechende Anfangsbuchstabe erwartet.

Die **Zentrale Spielkoordination** ist dafür zuständig, die anderen Module untereinander abzustimmen und wichtige Parameter global zur Verfügung zu stellen. Für Ersteres löst der *AGameManager Broadcasts* aus, die andere Module beispielsweise darüber informieren, dass die Levelgenerierung abgeschlossen ist und die Spielrunde gestartet werden muss. Solche Zustandsänderungen werden dem *AGameManager* mitgeteilt. So können alle Objekte auf generelle Ereignisse reagieren, ohne dass sie den Zustand vieler anderer Elemente verfolgen müssen. Außerdem ist durch diese Zentralisierung die Erweiterung oder der Austausch von Funktionsbereichen weniger aufwendig. *AGameManager* soll weiterhin Referenzen auf Generierungs- und Spielparameter (*UDifficultySettings* und *UGeneralSettings*) oder etwa den aktuellen Punktestand allgemein zugänglich führen.

Der Bereich **Dateisystemzugriff** hat die Aufgabe, einen festgelegten Ordner auf dem *Windows*-System nach vorhandenen Musikstücken zu durchsuchen und deren absoluten Dateipfade zu ermitteln. Diese Funktionalität wird von der Klasse *SoundFileHandler* mit Hilfe des Dateimanagers der *Unreal Engine* implementiert, die außerdem eine Methode zur Umwandlung von Dateipfaden in Liednamen enthalten soll. Genutzt wird *SoundFileHandler* vom Modul **Menü**, um beim Spielstart eine Liste aller spielbaren Lieder zur Wahl anzuzeigen. Dabei kommen *Widgets* zum Einsatz – Objekte zur Anzeige von 2D-Elementen in einem *Unreal Engine Level*. *USelectionList* und *USelectableListItem* sind solche *Widgets*, die von *UUserWidget* erben und für alle Auswahllisten des Prototyps wiederverwendet werden. Die Hauptkomponente von **Menü** ist die Klasse *AMenu*, welche abhängig vom aktuellen Spielfortschritt ein passendes *Widget* präsentiert und Nutzereingaben kontextsensitiv behandelt. Dafür soll das Modell der *finite-state machine* (FSM, deutsch *endlicher Automat*) [Nys14, S. 87–94] verwendet werden, um sicherzustellen, dass sich *AMenu* immer nur in einem von vielen möglichen Zuständen befindet. Unabhängig von *AMenu* sind die Klasse *AGenerationLoadingScreen* und ihr Hilfsinterface *ILoadingScreenRequester*, welche für die Anzeige des Ladebildschirms verantwortlich sind.

Das Modul **Levelgenerierung** extrahiert mit *ALevelGenerator* die benötigten Merkmale aus einem gewählten Musikstück und erstellt darauf gestützt eine Abfolge (*SoundLevel*) von Blöcken (*SoundBlock*), die darauf von der **Spielrundenlogik**

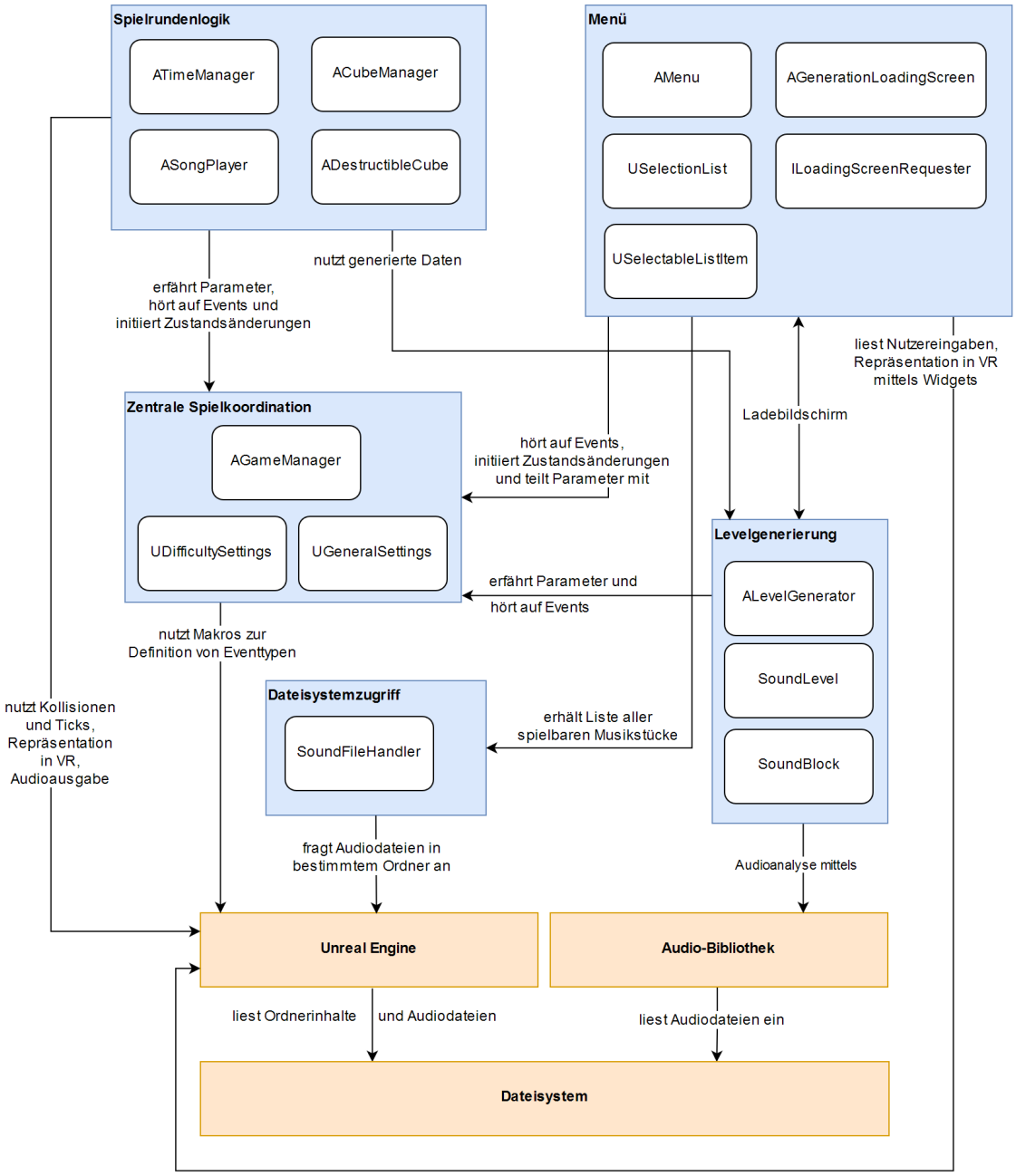

Abbildung 3.2.: Vereinfachte Systemarchitektur des Rhythmik-Videospiels. Blaue Kästen stehen für Module des Prototyps, welche ihre wichtigsten Klassen (weiße, abgerundete Kästen) umschließen. Orange Kästen repräsentieren fremde Systemkomponenten. Pfeile zeigen mögliche Zugriffe von Funktionsbereichen auf andere sowie die jeweiligen Zugriffsmodalitäten. (Angefertigt mit: `https://www.draw.io/`)

zu passenden Zeitpunkten in zerschlagbare Würfel (*ADestructibleCube*) in der VR umgewandelt werden. *ALevelGenerator* greift dabei auf eine geeignete fremde Softwarebibliothek zurück, deren Auswahl im folgenden Kapitel näher betrachtet wird. **Spielrundenlogik** vereint mehrere Objekte, die während der Spielrunde aktiv sind und im Zusammenspiel das Spielen des generierten Levels ermöglichen. *ATimeManager* kontrolliert den Zeitfluss vor, während und nach der Wiedergabe des Musikstückes mittels *ASongPlayer*, an dem sich beispielsweise *ACubeManager* orientiert, damit die Würfel präzise zur Musik platziert und bewegt werden können.

Mit diesem übersichtlichen Plan für die Softwarearchitektur der prototypischen Anwendung kann nun die Umsetzung erfolgen. Im folgenden Kapitel werden wesentliche Implementationsdetails näher beschrieben.

4. Implementierung

Zu Beginn der softwaretechnischen Umsetzung des Prototyps ist die Erstellung eines *Unreal-Engine-Projektes* vonnöten. Dabei wird festgelegt, dass es sich um ein *C++-Projekt* handelt. Die *Engine* legt anschließend alle nötigen Dateien in einem gewählten Verzeichnis im Dateisystem an. Es öffnet sich ein *Level-Editor* mit einem leeren *Level* (ähnlich wie in Abb. 2.5 in Kapitel 2.3). Bevor nun in diesem die eigentliche Arbeit am Rhythmik-Videospiel beginnen kann, müssen alle benötigten Plugins und Bibliotheken installiert werden.

4.1. Plugins und Bibliotheken

Da es sich um eine Anwendung der Virtuellen Realität handelt, wird zunächst die Funktionsfähigkeit der *HTC Vive* im Zusammenspiel mit der *Unreal Engine* sichergestellt. Das dafür zuständige *SteamVR-Plugin* ist bereits vorinstalliert und aktiviert, und bedarf keiner weiteren Einstellungen. Dies ist über den Menüpunkt *Edit* → *Plugins* → *Virtual Reality* einzusehen.

Aus der Aufgabenstellung ist zu entnehmen, dass sich die automatische Levelgenerierung auf eine freie Softwarebibliothek stützen soll. Eine solche Bibliothek wurde nicht vorgegeben und muss deshalb noch ausfindig gemacht werden. Dafür müssen zuerst die Anforderungen an diese aufgestellt werden. Es wird nach geeigneten extrahierbaren Merkmalen eines Audiosignals gesucht, die sich sinnvoll in geplante Levelinhalte überführen lassen.

4.1.1. Extrahierbare Merkmale von Audiosignalen

In Kapitel 2.5 wurde bereits eine Einführung in die verschiedenen von Menschen wahrnehmbaren Elemente der Musik gegeben. Da das Rhythmik-Videospiel vom

Spieler passend zu den wahrgenommenen Elementen bestimmte Aktionen fordern soll, beschränkt sich die Suche nach möglichen extrahierbaren Audiomerkmalen auf die Menge ebendieser Elemente. Resultierend aus Überlegungen wird festgelegt, dass die Grundelemente *Tonhöhe*, *Rhythmus*, *Melodie*, Lautstärke, *Taktschläge*, *Takt* und *Tempo* für den Zweck der Levelgenerierung weiter betrachtet werden. Zusätzlich könnten Metainformationen wie der Name oder die Gesamtlänge des Musikstückes interessant werden. Als nächstes müssen Entscheidungen über die Gestaltung eines Spiellevels getroffen werden, um die Konzeption der Umwandlung von Audiomerkmalen in passende Spielelemente zu ermöglichen.

4.1.2. Überführung in Levelinhalte

Das prototypische Spiel soll grundsätzlich wie *Beat Saber* aufgebaut sein. Bei der Gestaltung der Levelgenerierung hält sich der Entwickler deshalb an die Spielmechaniken des Vorbilds. Aus Beobachtungen geht hervor, dass sich zerschlagbare Würfel an einem unsichtbaren Gittern ausrichten und sich ab dem Zeitpunkt ihrer Platzierung im *Level* auf den Spieler zubewegen. Befinden sie sich in unmittelbarer Nähe zum Spieler, ist anhand der Musik der ideale Schlagzeitpunkt für maximale Punktzahl zu erkennen. Die Levelgenerierung muss demnach für die Würfel Koordinaten auf einem Gitter sowie dessen erwarteten Zerstörungszeitpunkt festlegen. Auch eine Bewegungsgeschwindigkeit muss definiert werden, aus der sich zusammen mit der *Spawnentfernung* der *Spawnzeitpunkt* berechnen lässt. Der Begriff *Spawnen* ist synonym für das Entstehenlassen eines Objektes im *Level*. Eine weitere Spielmechanik von *Beat Saber* ist die Farbdualität. Sowohl die Lichtschwerter als auch die Würfel treten in zwei verschiedenen Farben auf. Ein Würfel lässt sich nur mit dem Lichtschwert der gleichen Farbe zerschlagen. Außerdem kann es geschehen, dass Würfel vermehrt auf einer Seite des Spielbereiches auftreten. Die Levelgenerierung soll also Entscheidungen darüber treffen, mit welcher Hand ein Würfel zu zerschlagen ist und auf welcher Spielbereichseite er *spawnt*. Die Einschränkung, mit welcher Schlagrichtung sich ein Würfel zerstören lässt, wird ebenfalls bei der Generierung festgelegt. Letztlich soll es vorkommen, dass zwei Würfel zeitgleich *spawnen*. Diese Eventualität muss vom Generator berücksichtigt werden.

Bei der Gegenüberstellung der extrahierbaren Audiomerkmale und der geforderten Levelinhalte ist erkennbar, dass sich nicht alle Spielmechaniken sinnvoll durch die Merkmale abbilden lassen. Deshalb wird ein weiterer Faktor hinzugezogen: Zufall.

(NF1) verlangt jedoch einen deterministischen Algorithmus. Durch das Erzeugen von scheinbar zufälligen Zahlenfolgen aufbauend auf einer Anfangszahl, dem *Seed*, entsteht eine Pseudo-Zufälligkeit, die bei wiederholter Wahl desselben *Seeds* die Ausgabe immergleicher Zahlenfolgen beobachten lässt. Fällt die Wahl des *Seeds* etwa auf eine Ganzzahlenrepräsentation des Namens des Musikstückes, so kann **(NF1)** eingehalten werden. Nachfolgend wird die Umwandlung einzelner Elemente in geplante Levelinhalte festgelegt:

- *Rhythmus* → erwarteter Zerstörungszeitpunkt des Würfels
- *Tonhöhe* → Höhe der *Würfelspawnposition* (y-Wert für die Gitterkoordinaten)
- Lautstärke eines Tons → seitliche Ausrichtung der *Würfelspawnposition* (x-Wert für die Gitterkoordinaten)
- *Tempo* → Bewegungsgeschwindigkeit des Würfels
- Name des Musikstücks → *Seed* für Zufall
- Zufall → *Spawnseite*
- Zufall → geforderte Schlaghand
- Zufall → geforderte Schlagrichtung
- *Taktschlag* einhergehend mit erwartetem Zerstörungszeitpunkt → *Spawn* eines zweiten Würfels
- Zufall → Eigenschaften eines zeitgleichen Würfels

4.1.3. Wahl der geeigneten Bibliothek

Aus dem Wissen, welche Merkmale von einer Audioanalyse-Bibliothek extrahiert werden müssen, ist die Ableitung von Vergleichskriterien trivial. Wie Kapitel 2.3 andeutet, ist weiterhin darauf zu achten, dass die Softwarebibliothek nicht von anderen Bibliotheken abhängig ist. Zusatzfunktionen sind für die Levelgenerierung implementierbar, falls bei der Audioanalyse die Ähnlichkeit mehrerer Musikabschnitte identifiziert werden kann oder falls eine mehrstimmige *Melodieerkennung* möglich ist.

Bibliotheken→ ↓Kriterien	aubio	beatdetektor	Essentia	Gist	CLAM	openSMILE	Yaafe	MARSYAS	LibXtract	Maaate	CAMEL
Unabhängigkeit von anderen Bibliotheken	-	-	-	-	-	-	-	-	-	-	-
Lesen von Musikdateien	+	-	+	-	+	+	+	+	?	+	+
Einsehen von Metadaten	-	-	?	-	?	?	-	?	-	+	-
Tempo berechnen	+	+	+	-	?	?	-	+	-	-	-
Takt analysieren	+	-	+	-	?	?	-	+	-	-	-
Tonhöhenbestimmung für eine Stelle	+	-	+	+	?	+	-	+	+	-	+
Lautstärkebestimmung für eine Stelle	-	-	+	-	?	+	+	?	+	-	+
Rhythmus/Melodien erkennen	-	-	+	-	?	?	-	-	-	-	-
Bonus: Erkennung von Wiederholungen	-	-	+	-	?	?	-	?	-	-	+
Bonus: mehrstimmige Melodieerkennung	-	-	?	-	?	?	-	-	-	-	-
Summe positiv	4	1	7	1	1	3	2	4	2	2	4

Abbildung 4.1.: Vergleichsmatrix für die Wahl einer geeigneten Softwarebibliothek anhand von Anforderungskriterien. Grüne Felder bedeuten, dass ein Kriterium erfüllt wird, rote Felder das Gegenteil und gelbe Felder signalisieren Ungewissheit. Die Fußzeile gibt die Summe aller grünen Felder für die jeweilige Bibliothek an. (Angefertigt mit: `https://docs.google.com`)

Es wird eine Auswahl von freien Softwarebibliotheken ermittelt, die mit der Extraktion von Audiomerkmalen werben und eine Kompatibilität zu *C++* aufweisen. Anschließend werden die gefundenen Bibliotheken zum Vergleich mittels der aufgestellten Kriterien hinzugezogen. Dabei wird oberflächlich anhand der Feature-Auflistung oder der API-Dokumentation der Bibliothek – API steht für *Application Programming Interface*, also eine Schnittstelle für die Anwendungsentwicklung – entschieden, ob ein Kriterium erfüllt wird. Kann keine eindeutige Aussage über das Vorhandensein einer Funktion getroffen werden, wird dies ebenfalls vermerkt. Die Vergleichsmatrix in Abbildung 4.1 ist das Ergebnis dieser Untersuchungen.

Auffällig ist, dass keine einzige Bibliothek dem Anspruch an Unabhängigkeit genügt. Dies würde beispielsweise kritisch, falls Funktionen auf die *C++*-Standardbibliothek gestützt sind, weil diese für jede Plattform unterschiedlich implementiert ist. Da jedoch die *Unreal Engine* eine Lösung für die Entwicklung von Anwendungen für mehrere Plattformen darstellt, kommt es zu Konflikten. Auch die Abhängigkeit einer Bibliothek von vielen anderen Bibliotheken führt zu Problemen beim Einbezug in die *Unreal Engine*.

Die frei verfügbare Softwarebibliothek *Essentia* ist nach Auswertung der Vergleichsmatrix (siehe Abb. 4.1) offensichtlich die beste Wahl für die Implementierung des Prototyps. Allerdings konnte diese nicht erfolgreich in die *Unreal Engine* integriert

werden. Die Gründe dafür sind vielfältig. *Essentia* hängt von zu vielen fremden Bibliotheken ab, sodass ein Kopieren des Quellcodes in das *Projekt* nicht für dessen Funktionsfähigkeit genügt. Beim Kompilieren des *Projektes* können Verweise auf unbekannte Bereiche nicht aufgelöst werden. Also bleibt als alternativer Lösungsweg der Einbezug dieser Bibliotheken beim Linkprozess. Jedoch sind einige davon nicht für das Betriebssystem *Windows* gemacht, was zu weiteren Fehlermeldungen führt. *Essentia* stellt zudem keine Mittel zur eigenständigen Zusammensetzung der Bibliothek auf *Windows* zur Verfügung. Es wird versucht, stattdessen eine der Bibliotheken *MARSYAS* oder *aubio* zu integrieren. Bei beiden gibt es ähnliche Komplikationen wie bei *Essentia*, mit der Ausnahme, dass *aubio* kaum Fremdabhängigkeiten aufweist und ausreichende Mittel zur plattformübergreifenden Kompilierung bereitstellt. Die daraus erhaltene „.lib"-Datei lässt sich in den Linkprozess der *Unreal Engine* integrieren. *Aubio* ist in der mit *C++* verwandten Programmiersprache *C* geschrieben. Zur Nutzung derer Funktionen im *C++*-Quellcode ist allein die Inklusion der *C-Headerdateien* vonnöten. Diese müssen schließlich dem *Projekt* zusammen mit der kompilierten Bibliothek in einer Datei namens „<Projektname>.build.cs" bekannt gemacht werden. Bei der Implementierung der Levelgenerierung können nun von *aubio* bereitgestellte Funktionen genutzt werden.

4.2. Zentrale Spielkoordination und Menü

Beim Spielstart lädt die *Unreal Engine* das *Level* mit allen platzierten *Actors*. Die Darstellung erfolgt auf dem HMD, welches vor dem Spielstart zusammen mit den Controllern und den *Basis-Stationen* mit der *SteamVR-Software* verbunden werden muss. Der Spieler kann sich frei auf begrenztem Raum in der Virtuellen Realität bewegen und die Controller werden durch prototypische Lichtschwerter repräsentiert **(F13)**. Die Hauptkomponenten der verschiedenen Funktionsbereiche liegen ebenfalls als *Actors* im *Level* vor, allerdings größtenteils ohne sichtbare Elemente. Jedes davon hält eine Referenz auf das von *AGameManager* abgeleitete *Actor-Objekt*, woran sie sich für für die jeweils benötigten Ereignisse anmelden. Das Objekt *AMenu* fügt beispielsweise dem *Delegat GameOver* per Funktionsaufruf eine Methode hinzu, die später beim Ende einer Spielrunde im Zuge eines *Broadcasts* des *AGameManager* aufgerufen wird. Ursprünglich war die Umsetzung des *AGameManager* mittels des Programmiermusters *Singleton* [Nys14, S. 73–86] geplant, welches einen simplen globalen Zugriff auf eine einzige Instanz von *AGameManager*

ermöglichen und die Erstellung weiterer solcher Objekte unterbinden würde. Jedoch ist das Muster nicht mit der *Unreal Engine* kompatibel, welche innerhalb des *Editors* von *Actors* jeweils mehrere Instanzen für verschiedene Zwecke erstellt und verwaltet. Es wird deshalb auf die manuelle Zuweisung von Referenzen über den *Editor* zurückgegriffen. Neben der Initialisierung der Ereignisverarbeitung meldet sich *AMenu* im ersten Schritt als Empfänger von Controller-Eingaben und vollführt den Übergang in seinen Ausgangszustand. Jeder Menüzustand korrespondiert mit der Anzeige eines bestimmten *Widget-Blueprints* mittels seiner von *UWidgetComponent* abgeleiteten *Component*. Das FSM (siehe Kapitel 3.2) ist dafür verantwortlich, dass sich *AMenu* immer nur in einem einzigen Zustand befindet und dass die Überleitung nur zu festgelegten Zuständen stattfinden kann [Nys14, S. 87–94]. Hierbei wird das aktive *Widget-Blueprint* durch ein neues ersetzt und die Inputbehandlung neu konfiguriert **(F6)**. Das Menü bietet vor dem Spielrundenbeginn eine Folge von *USelectionList*, welche als scrollbare Listen mit durch Nutzereingaben wählbaren Elementen (*USelectableListItem*) dargestellt werden. Vor einer Zuständsänderung erfolgt die Übergabe der selektierten Elemente an den *AGameManager* als Levelgenerierungsparameter. Darunter zählen die *UDifficultySettings* und der Dateipfad des Musikstückes **(F1–4)**. *UDifficultySettings* erbt von *UDataAsset* und kann deshalb als *Asset-Objekt* mit festgelegten Eigenschaften im *Unreal Editor* persistiert und von anderen Objekten referenziert werden. Somit lassen sich mehrere Schwierigkeitsgrade vorerstellen, die für einen dynamischen Austausch der Generierungsparameter zur Laufzeit genutzt werden **(F11)**. Ähnlich verhält sich auch *UGeneralSettings*, welches für alle Objekte zugängliche, von der Schwierigkeit unabhängige Parameter enthält. Nach der Wahl aller benötigten Einstellungen durch den Spieler lädt *AMenu* das *Widget* für die Anzeige des Punktestandes und informiert *AGameManager* über den Spielrundenbeginn. Dieser löst einen zugehörigen *Broadcast* aus, um die automatische Levelgenerierung anzustoßen. Eine laufende Spielrunde lässt sich pausieren und abbrechen **(F5)**. Wird sie erfolgreich abgeschlossen, so erfolgt eine Zustandsänderung zur Rundenabschlussanzeige, die den erreichten Punktestand angibt **(F7)** und aus der eine Wiederholung des gespielten Levels **(F8)** oder die erneute Wahl der Generierungsparameter möglich sind.

4.3. Levelgenerierung

Beim Empfang des *Broadcasts* zur Spielrundenvorbereitung registriert sich *ALevelGenerator* bei *AGenerationLoadingScreen* für die asynchrone Ausführung der Hauptmethode zur Levelgenerierung und die gleichzeitige Anzeige eines Ladebildschirm-*Widgets* **(F16)**. Es liegen zwei verschiedene Algorithmen zur Generierung eines *SoundLevel* vor, zwischen denen im *Unreal Editor* gewählt werden kann. Algorithmus 1 skizziert denjenigen, der zu besseren Ergebnissen führt und deshalb standardmäßig gewählt ist.

SoundBlock ist dabei eine Struktur zum Speichern aller Eigenschaften eines Würfels, der während der Spielrunde im *Level* platziert wird. Die Klasse *SoundLevel* definiert ein Array, das alle geplanten *SoundBlocks* für eine Spielrunde beinhaltet. Dieses *SoundLevel* ist öffentlich abrufbar von allen Objekten, die eine Referenz auf *ALevelGenerator* halten.

Bei der Implementierung mit der Audioanalyse-Bibliothek *aubio* ist die unterschiedliche Handhabung von Objekten gegenüber der in *C++* zu beachten. Speicherallokation und -befreiung sind nicht mit Hilfe von Schlüsselwörtern sondern über den Aufruf von dedizierten Methoden zu realisieren. So würde beispielsweise ein Zeiger auf ein fiktives Objekt vom Typ *aubio_object_t* von der Methode *new_aubio_object()* zurückgegeben und der Speicher über *del_aubio_object()* wieder freigegeben. Ein Methodenaufruf auf Objekten von *aubio* ist ebenfalls nicht möglich. Stattdessen wird eine globale Methode aufgerufen und ein Zeiger auf das spezifische Objekt als Parameter übergeben. Das Auslesen und Analysieren von Audiodateien erfolgt inkrementell **(F9)**. Dafür müssen die Objekte *aubio_source_t*, *aubio_tempo_t* und *aubio_notes_t* mit der gleichen *hopSize* initialisiert werden. Diese gibt an, wie viele Stichproben (oder Samples) beim einem einzelnen Aufruf der *aubio*-objektspezifischen Hauptarbeitsmethode mit dem Suffix *_do* gelesen bzw. verarbeitet werden. Solche Funktionen werden anschließend so oft aufgerufen, bis alle Samples der Audiodatei durchlaufen wurden.

Da die Softwarebibliothek *aubio* keine Ermittlung von Lautstärke zu einem beliebigen Zeitpunkt erlaubt, wird auf deren Fähigkeit zur Findung von MIDI-Noten (*Musical Instrument Digital Interface*, deutsch digitale Schnittstelle für Musikinstrumente) zurückgegriffen. MIDI-Noten enthalten Informationen über die Höhe eines gespielten Tons und dessen Anschlagstärke, welche als Lautstärke interpretiert

Algorithmus 1 : Generierung eines *SoundLevel*

```
1  fetch player-chosen parameters from AGameManager;
2  set up aubio objects for sound analysis;
3  set up random number generator;
4  set up SoundBlock array;
5  initialize analysis variables, vectors and arrays;
6  while there are samples to be read from audio file do
7      have aubio_source_t object read a few samples from source file;
8      have aubio_tempo_t object search samples for beat;
9      if beat was found with enough confidence then
10         add beat information to array of found beats;
11     end
12     have aubio_notes_t object search samples for midi note;
13     if midi note was found then
14         add midi note information to array of found notes;
15     end
16 end
17 while there are still elements in arrays of found beats and notes do
18     determine time frame between the first two elements in the beats array;
19     while still within time frame and maximum amount of SoundBlocks for this
        frame haven't yet been created do
20         remove all midi notes from array which have a lower time signature than
            currently observed time or than the last created SoundBlock;
21         determine whether midi note is on beat;
22         create SoundBlock from midi note;
23         add created block to SoundBlock array;
24         if SoundBlock is on beat then
25             create second SoundBlock from midi note;
26             add created block to SoundBlock array;
27         end
28     end
29     remove first element from array of found beats;
30 end
31 make SoundLevel from SoundBlock array;
32 clear memory;
```

werden kann. Beide Werte befinden sich in einem Bereich zwischen 0 und 127 und werden in die Koordinaten eines *SoundBlocks* auf dem *Spawngitter* umgerechnet. Die *Spawnseite*, Schlagseite und Schlagrichtung werden wahrscheinlichkeitsbasiert zugewiesen. Das *aubio_tempo_t*-Objekt bestimmt das aktuell vorherrschende musikalische Tempo, welches in die Geschwindigkeit eines *SoundBlocks* umgewandelt wird **(F12)**. Ein eventueller zweiter Block wird auf einen horizontal, vertikal oder diagonal angrenzenden Gitterplatz gesetzt und seine weiteren Eigenschaften werden so abhängig vom Hauptblock gewählt, dass keine unabsolvierbaren Kombinationen entstehen **(NF2)**.

Der aktuell verwendete Algorithmus hat die Herangehensweise, zuerst alle Taktschläge und MIDI-Noten aus der Audiodatei zu extrahieren und daraus später geeignete Blöcke zu wählen. Der erste Implementierungsversuch basiert dahingegen darauf, immer zuerst zwei Taktschläge ausfindig zu machen und zwischen diesen eine geeignete Anzahl von MIDI-Noten zu finden, welche in *SoundBlocks* umgewandelt werden. Dafür erfolgt eine Aufteilung der *Samples* in gleich große Intervalle, welche nacheinander durchsucht werden. Bei einem Treffer werden die restlichen *Samples* des Intervalls übersprungen. Dies bringt zwar einen geringeren Rechenaufwand mit sich, jedoch ist das generierte *SoundLevel* in den Augen des Entwicklers weniger zufriedenstellend. Dies könnte daran liegen, dass durch manuelles Setzen des *aubio_source_t*-Objektes auf bestimmte Positionen der Audiodatei starke Brüche bei der Analyse von aufeinanderfolgenden *Sample*-Vektoren entstehen, was Fehlinterpretationen zur Folge haben kann.

4.4. Spielrundenlogik

Ein weiterer Empfänger des *Broadcasts* zur Spielrundenvorbereitung ist der *Actor ASongPlayer*. Wie *ALevelGenerator* implementiert dieser ebenfalls *ILoadingScreenRequester* und beinhaltet damit eine Routine zur asynchronen Ausführung während der Anzeige des Ladebildschirms. Darin wird mit Hilfe der *Unreal Engine* eine Audiodatei im WAVE-Format (*Waveform Audio File Format*) ausgelesen und in ein von der mit dem *Actor* verknüpften *UAudioComponent* nutzbares Objekt geschrieben **(F9–10)**. Abschließend muss das Objekt für die Audiowiedergabe gepuffert werden, um zu verhindern, dass das erste Abspielen den Hauptthread blockiert. Allerdings ist die Vollendung des Puffervorganges nur auf ebendiesem möglich und

eine Blockierung ist unumgänglich. *SteamVR* beugt dem Stillstand der dargestellten Bilder im HMD der *HTC Vive* vor und versetzt den Spieler kurzzeitig in einen alternativen Raum, in dem Bewegung weiterhin möglich ist **(NF5)**.

AGenerationLoadingScreen wartet auf den Abschluss aller asynchronen Vorbereitungen, bevor er über den *AGameManager* den Start der Spielrunde bekanntgibt. Darauf reagieren *ATimeManager*, *ASongPlayer* und *ACubeManager*. Ersterer ist für die Verfolgung der vergangenen Zeit in einer Spielrunde zuständig. Dabei werden eingestellte Wartezeiten vor und nach der Wiedergabe des Musikstückes berücksichtigt. Sobald das anfängliche Zeitfenster mittels *tickbasierter* Zeitzählung abgewartet wurde, weist *ATimeManager* den *ASongPlayer* an, das Musikstück abzuspielen. Ein *Tick* ist der Zeitpunkt, zu dem die *Unreal Engine* vor dem Aufbau eines jeden Bildes den Aufruf der gleichnamigen Methode auf allen *Actors* durchführt. Dieses Programmiermuster ist auch unter dem Namen *Update Method* bekannt [Nys14, S. 139–152]. Ab diesem Zeitpunkt erfragt *ATimeManager* in jedem *Tick* den Wiedergabefortschritt als Grundlage für die Zeitrechnung. Damit ist die Synchronisierung aller Prozesse, die von der aktuellen Spielzeit abhängen, mit dem Musikstück sichergestellt **(F15)**. Nach vollständiger Wiedergabe wird erneut ein festgelegter Zeitraum abgewartet und schließlich am *AGameManager* ein *Broadcast* für den Rundenabschluss initiiert.

ACubeManager ist für das Platzieren, Bewegen und Zerstören der zerschlagbaren Würfel (*ADestructibleCube*) verantwortlich. Dafür holt er nach der Generierung das entstandene *SoundLevel* ein und legt ein leeres Array für alle *gespawnten* Würfel an. In jedem *Tick* wird unter Einbezug des *ATimeManager* ermittelt, ob ein Würfel im *Level* platziert werden muss, damit dieser sich rechtzeitig zu seiner geplanten Schlagzeit an der richtigen Position befindet. Ist dies der Fall, *spawnt ACubeManager* ein passendes *Blueprint*-Objekt, übergibt ihm wichtige Parameter und fügt es dem zuvor angelegten Array hinzu. Jeder Würfel hat die gleiche Farbe wie eines der beiden prototypischen Lichtschwerter und eine farblich differenzierte Markierung, die die erwartete Schlagrichtung symbolisiert **(NF3)**. Weiterhin werden bei jedem *Tick* die Bewegungsfortschritte der *gespawnten* Würfel anhand der Musik-Wiedergabezeit interpoliert und die aktuellen Positionen daraus abgeleitet und festgelegt **(F15)**. Abschließend erfolgt eine Prüfung der Würfel auf einen Zerstörungswunsch, der das Entfernen aus dem *Level* und aus dem Array zur Folge hat. Ein solcher Zerstörungswunsch wird vom *ADestructibleCube* durch einen Bool-Wert geäußert, entweder bei erfolgreichem Schlag durch den Spieler oder nach Überschreiten der maximal er-

laubten Bewegungsentfernung. Die Auswertung der Berührung eines Würfels durch ein Lichtschwert ist über die *Unreal Engine Kollisionserkennung* realisiert. Sobald der Spieler einen Würfel auf die richtige Weise zerschlägt, berechnet dieser eine Punktzahl abhängig von der erwarteten und tatsächlichen Schlagzeit und übergibt diese an den *AGameManager*. Dieser informiert per *Broadcast* über den neuen Punktestand, der auf dem *UScoreWidget* aktualisiert wird, welches von *AMenu* während der Spielrunde angezeigt wird.

4.5. Spielparameter

Die für alle Objekte zugänglichen *UGeneralSettings* und *UDifficultySettings* sind bereits bekannt. Jedoch besitzen die *Actors* der Anwendung weitere klassenspezifische Eigenschaften, die abhängig von ihrer Ausprägung zu unterschiedlichem Spielverhalten führen. Diese Spielparameter können im *Unreal Editor* im *Details-Panel* angepasst werden, sofern den jeweiligen Klassenvariablen das Makro *UPROPERTY()* vorangestellt wird. Somit lassen sich Feineinstellungen vornehmen, ohne den Code für jede Änderung erneut kompilieren zu müssen. Es können auch beispielsweise Zeiger auf *Actor*-Klassen durch *Editor*-Referenzen auf *Actors* im *Level* realisiert werden, ohne dass die Suche nach diesem über *C++*-Code erfolgen muss.

4.6. Blueprints

Im *Projekt* kommen drei verschiedene Arten von *Blueprints* zum Einsatz. *Widget-Blueprints* basieren auf der Klasse *UUserWidget* oder einer Unterklasse und bieten im *Widget-Editor* eine Auswahl von Werkzeugen zum Entwurf von 2D-Elementen. Sie werden im *Level* durch *UWidgetComponents* auf einem *Actor* gezeichnet. Sie finden Anwendung beim Modul *Menü*, wo beispielsweise jeder Selektionsliste ein eigenständiges *Widget-Blueprint* zugrunde liegt. *Actor-Blueprints* haben *AActor* als Grundklasse und im *Blueprint-Editor* einen *Viewport* zur relativen Positionierung von zugehörigen *Components*. *Actor-Blueprints* können analog zu *Actors* im *Level* instanziert werden. Von den beiden verschiedenen Würfelarten *ADirectionalCube* und *ANonDirectionalCube*, welche von *ADestructibleCube* erben und sich in der Implementierung der Schlagerkennung unterscheiden, existieren jeweils eine *Blueprint-*

Klasse, die von *ACubeManager* zur Laufzeit *gespawnt* werden. Als letztes ist das *Pawn-Blueprint* namens *VR_Pawn* zu betrachten, welches die Repräsentation eines Spielers in der Virtuellen Realität ermöglicht. Im Gegensatz zu *Actors* haben *Pawns* zusätzliche Funktionalitäten typisch für durch Nutzereingaben gesteuerte Charaktere. Der *Editor* ist jedoch gleich aufgebaut. *VR_Pawn* beinhaltet die in Kapitel 2.3 beschriebenen *SteamVR-Components*, die unter anderem die Verbindung der Lichtschwerter in der VR mit den Controllern in der Realität ermöglichen. Alle drei Arten von *Blueprints* erlauben das Hinzufügen von Programmlogik zum Objekt mit Hilfe des visuellen *Blueprint Scripting*. Dieses Feature wird jedoch nur für die dynamische Anpassung der Kamerahöhe des *VR_Pawns* genutzt.

5. Evaluation

Die vorliegende prototypische, zu *Beat Saber* funktionsähnliche Anwendung wird nun einer Evaluation unterzogen. Deren Ziel ist es, herauszufinden, ob die vom Programm generierten Levels zum jeweiligen Musikstück passen. Dabei müssen vor allem die mittels Audioanalyse gewählten Parameter eines Würfels im Zusammenhang mit der Wahrnehmung des Spielers betrachtet werden. Es wird dafür eine Reihe von Probandentests durchgeführt. So können Informationen darüber gewonnen werden, wie gut menschlich wahrgenommene Musikelemente auch von der Softwarebibliothek *aubio* registriert werden. Außerdem sollen mit der Evaluation schwerwiegende Fehler in der Programmlogik der Levelgenerierung aufgedeckt werden, wie beispielsweise das Auftreten unabsolvierbarer Schlagkombinationen bzw. -abfolgen. Schließlich können Rückschlüsse darüber getroffen werden, wie geeignet die gewählte Bibliothek für den gegebenen Zweck ist und in welcher Hinsicht die Anwendung Optimierungsbedarf aufweist.

Aufgrund der hohen Komplexität einer Analyse polyphoner Musikstücke liegt die Vermutung nahe, dass unterschiedliche Kompositionen auch zu einer Differenz in der Güte des generierten Levels führen. Musikgenres sind Gruppen, die aus dem Gesamtspektrum aller Musikstücke diejenigen zusammenfassen, deren Aufbau sich ähnelt. Somit können bei der Evaluation durch die Wahl repräsentativer Musikstücke aus sich unterscheidenden Genres Abweichungen in der Levelqualität herausgestellt werden. Damit ist die Eignung der verwendeten Softwarebibliothek von einem weiteren Parameter abhängig.

5.1. Versuchsaufbau und Testergebnisse

Die Menge der Probanden mit einer Mächtigkeit von 35 wird in fünf gleich große Versuchsgruppen eingeteilt. Die Versuchsgruppe ist ausschlaggebend dafür, welches Musikstück bzw. Musikgenre beim Test zum Einsatz kommt. Abgesehen davon sind

die Versuchsbedingungen und Spielparameter für alle Probanden gleich. Die Wahl der Genres zielt auf eine möglichst hohe Diversität ab. Außerdem sind die Genres *Rock*, *Hip Hop*, *Klavier*, *Jazz* und *Electro* so bekannt, dass sich die Probanden mit hoher Wahrscheinlichkeit vorher bewusst sind, was sie in der VR erwartet. Ein *Gaming-Labor* stellt eine ideale Räumlichkeit für die Durchführung der Probandentests dar, da dort ein *VR-System* und ausreichend Platz zur Nutzung der Anwendung vorhanden sind. Wegen des hohen Grades der *Immersion* und dem zusätzlichen Tragen von Kopfhörern können sich die Probanden trotz des Aufenthaltes anderer Personen im Labor auf das Spiel konzentrieren. Ein Probandentest läuft generell wie folgt ab:

1. Empfang des Probanden
2. Briefing
3. Wahl eines Musikgenres und damit Zuteilung zu Versuchsgruppe
4. Aufnahme von Vorerfahrungen
5. Test des Rhythmik-Videospiels
6. Auswertung der Eindrücke

Beim Briefing wird ein Proband über das grundlegende Spielprinzip aufgeklärt und woran sich in der VR-Umgebung orientiert werden sollte (siehe Abb. 5.1)**(NF4)**. Weiterhin wird der Proband gebeten, beim Spielen gezielt auf einige Dinge zu achten, die bei der nachfolgenden Auswertung hilfreich sind: die Ankunftszeit der Würfel in Bezug zum Takt der Musik, das Vorkommen von Würfelabfolgen passend zu *Rhythmus* und *Melodie*, die horizontale und vertikale Positionierung der Würfel abhängig von der wahrgenommenen Lautstärke und *Tonhöhe*, und das Vorkommen von unabsolvierbaren Schlagabfolgen. Bei Letzterem wurde nachdrücklich darauf hingewiesen, dass eine Schlagabfolge nur unabsolvierbar ist, wenn sie trotz ausgiebiger Übung und ausreichender Fitness tatsächlich unmöglich zu schaffen ist, und nicht, wenn der Proband sie beim ersten Test als zu schwierig empfindet.

Jeder Proband sucht sich ein Musikgenre aus. Allerdings stehen nur jene Genres zur Wahl, deren Versuchsgruppe höchstens einen Teilnehmer mehr enthalten als die aktuell kleinste Versuchsgruppe. Außerdem darf die maximale Probandenzahl von sieben nicht überschritten werden. Eine Verfälschung des Ergebnisses durch Präferierung eines Musikgenres mit einhergehender höheren Aufmerksamkeit oder

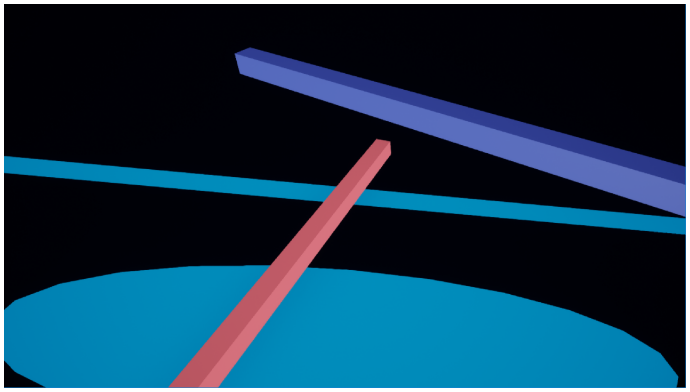

Abbildung 5.1.: Spielumgebung in der VR mit Orientierungspunkten: blaue Plattform als Standpunkt für den Spieler und blaue Linie als Schlagposition für die maximal erreichbare Punktzahl. Die prototypischen Lichtschwerter sind in rot und blau abgebildet.

Reaktionsfähigkeit ist somit nicht ausgeschlossen. Anschließend wird aufgenommen, ob der Proband schon Vorerfahrungen mit VR oder sogar *Beat Saber* hat und ob er schon einmal ein Instrument gespielt hat. Diese Informationen können später aufschlussreich sein bei der Identifizierung von Einflussfaktoren auf die Wahrnehmung der Levelgenerierung.

Sobald alle Geräte des VR-Systems am Teilnehmer befestigt worden sind und dieser sich auf seine Ausgangsplattform in der VR begeben hat, wählt die Versuchsleitung das Musikstück und den Schwierigkeitsgrad *medium* aus. Dies ist ebenfalls durch folgende Steuerung mit dem rechten Controller möglich: *Trackpad* oben und unten zur Auswahl und *Trigger* zum Bestätigen eines Listenelementes, Menütaste zum Rückgang oder zum Pausieren während einer Spielrunde. Der eigentliche Test der Anwendung findet nun statt. Auszüge der Spielrunde aus der Sicht des Probanden sind in Abbildung 5.2 zu sehen.

Danach werden dem Probanden folgende Aussagen vorgelesen:

(A1) Die Würfelsetzung passte zur Musik. (Gesamteindruck)

(A2) Die Würfelsetzung erfolgte taktgenau.

(A3) Die Würfelhöhe korrespondierte stets mit der wahrgenommenen Tonhöhe.

(A4) Die Entfernung eines Würfels zur Spielfeldmitte korrespondierte stets mit der wahrgenommenen Lautstärke.

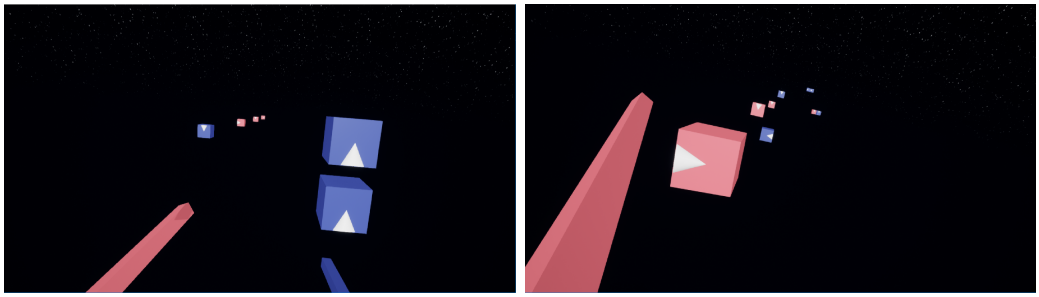

Abbildung 5.2.: Spielrunde des Prototyps aus der Sicht des Spielers. Generierte Würfel mit symbolisierter erwarteter Schlagrichtung und prototypische Lichtschwerter kurz vor dem Zerschlagen eines oder mehrerer Würfel. Außerdem existiert ein Sternenhimmel zur Höhenorientierung.

(A5) Melodien und deren Rhythmen wurden passend durch Würfel dargestellt.

(A6) Die Musik hat dabei geholfen, den richtigen Zeitpunkt für das Zerschlagen eines Würfels zu finden.

(A7) Alle Schlagabfolgen waren absolvierbar.

Der Teilnehmer hat die Auswahl zwischen 6 Stufen der Zustimmung. Er kann „überhaupt nicht zustimmen“ (0), „nicht zustimmen“ (1), „eher nicht zustimmen“ (2), „eher zustimmen“ (3), „zustimmen“ (4) oder „völlig zustimmen“ (5). „Keine Angabe“ (-) steht zur Verfügung, falls der Proband kein Gefühl für die Bewertung einer Aussage hat. Auf ein neutrales Element wird verzichtet, damit eine Entscheidung in eine Richtung gut überdacht wird. Anhand der gebotenen Funktionen der Bibliothek *aubio* und der Implementierung der Levelgenerierung erwartet die Versuchsleitung eine durchschnittliche Bewertung jeder Versuchsgruppe mit 4 Punkten für alle Aussagen. Weiterhin ist die Aufnahme und Betrachtung der erreichten Punktzahl nicht vorgesehen, da die Anwendung selbst eine Fehlerquelle ist und auch der Erfolg des Spielers von vielen Faktoren (beispielsweise Lerngeschwindigkeit oder körperlicher Zustand) abhängt, die nicht der Evaluation der Levelgenerierung dienen. Die vergebene Punktzahl beim Zerschlagen eines Würfels wird abhängig vom Abstand zur optimalen Schlagposition berechnet. So könnte es passieren, dass bei einem Spieler jeder Würfel einmal exakt auf dieser Position gezeichnet wird und somit auch dort zerschlagen werden kann, jedoch bei anderen Spielern die Maximalpunktzahl nie erreichbar ist. Zum Schluss werden alle Anmerkungen des Probanden zur Anwendung vermerkt.

Alle Randdaten und Beobachtungen eines einzelnen Probanden werden in einem Datensatz festgehalten, der anschließend für die Auswertung genutzt wird. Eine Auflistung der erhobenen Datensätze getrennt nach Versuchsgruppen ist in Anhang A einzusehen. Jedem Datensatz wird außerdem ein eindeutiges Identifikationskürzel zugewiesen.

5.2. Auswertung und Interpretation

Die erhobenen Daten müssen in eine geeignete Form zur Bewertung der Funktionalität der Levelgenerierung überführt werden. Die Nullhypothese [Rüg02, S. 6]

> H_0: Die Wahl des Musikgenres wirkt sich nicht auf die Güte der Levelgenerierung aus.

wird aufgestellt und mit der Absicht, diese zu widerlegen, sind alle Versuchsgruppen zunächst getrennt voneinander zu betrachten [Sie56, S. 6–14]. Das Bilden des arithmetischen Mittels (siehe Anhang B) ist anfänglich sinnvoll für einen groben Überblick. Es wird bewusst das arithmetische Mittel gegenüber dem Median gewählt, da bei einer begrenzten Anzahl möglicher Ausprägungen des Zustimmungsgrades keine Ausreißer existieren können, die das arithmetische Mittel verfälschen. In Abbildung 5.3 sind die Mittelwerte grafisch veranschaulicht, woraus bereits erste Informationen gewonnen werden können.

Auf den ersten Blick ist zu erkennen, dass bis auf ein paar wenige Ausnahmen die Erwartungen aus Kapitel 5.1 nicht erfüllt werden. **(A1)** und **(A7)** sind als besondere Aussagen anzusehen. **(A1)** stellt den Gesamteindruck des Spiellevels und dessen Generierung dar und die Anordnung der Versuchsgruppen um den Wert 3 lässt eine nur teilweise Zufriedenheit mit der Anwendung vermuten. Um mögliche Ursachen dafür zu ergründen, müssen die Aussagen **(A2)** bis **(A6)** betrachtet werden, die spezifische Punkte ansprechen. **(A7)** ist insofern ein Sonderfall, als dass sie für die Bewertung der Subjektivität oder der Glaubwürdigkeit eines Probanden hinzugezogen werden kann. Hier wird eine volle Zustimmung erwartet, da unter den gewählten Versuchsbedingungen keine unabsolvierbaren Schlagkombinationen oder -abfolgen entstehen. Dies wurde von der Versuchsleitung vorher sichergestellt.

Es sind außerdem deutliche Unterschiede zwischen den Ergebnissen der verschiedenen Versuchsgruppen zu erkennen, besonders bei **(A3)**, **(A5)** und **(A6)**. Um H_0 für

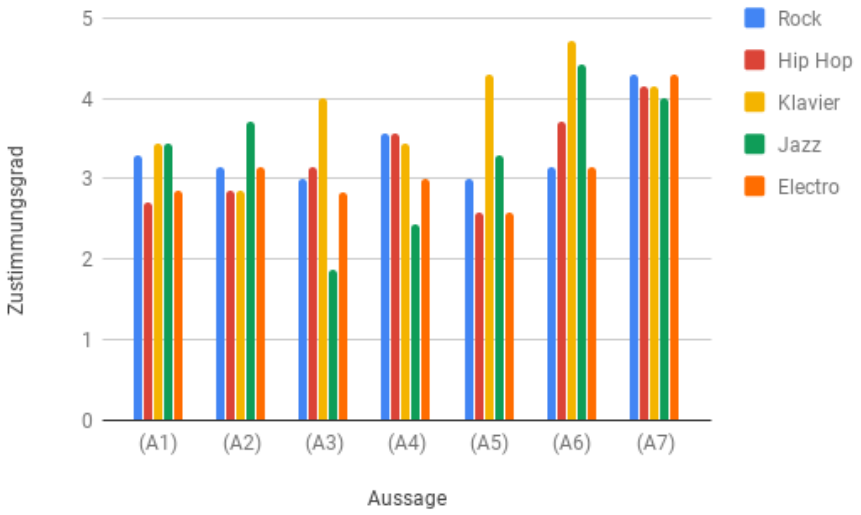

Abbildung 5.3.: Durchschnittliche Zustimmung aller Probanden getrennt nach Genres. Zu jeder Aussage ist das arithmetische Mittel des Zustimmungsgrades jeder Versuchsgruppe aufgetragen. (Angefertigt mit: `https://docs.google.com`)

jede Aussage gesondert widerlegen zu können, muss eine Prüfung der statistischen Signifikanz der Unterschiede der verschiedenen Genres stattfinden. Für unverbundene und nicht normalverteilte Stichproben mit einer Anzahl größer als zwei (fünf Genres) findet der *Kruskal-Wallis-Test* [Sie56, S. 184–192] Anwendung [dPRHB10, Sie56]. Es soll damit H_0 abgelehnt und die Alternative [Rüg02, S. 6]

> H_1: Das Genre des gewählten Musikstücks hat einen Einfluss auf die Güte des generierten Levels.

bewiesen werden. Es wird dafür vorher festgelegt, dass die Irrtumswahrscheinlichkeit p, dass H_0 abgelehnt wird, obwohl sie gilt, maximal bei 5% liegen darf. Daher erhält das Signifikanzniveau α den Wert 0.05. Die Ergebnisse des *Kruskal-Wallis-Test* (siehe Anhang C) zeigen, dass die Unterschiede unter den gewählten Bedingungen nur für **(A3)** und **(A5)** als signifikant einzuordnen sind und für alle übrigen Aussagen sehr wahrscheinlich der Zufall für die Schwankungen der Mittelwerte verantwortlich ist. **(A6)** ist ein Grenzfall, da p nur minimal über α liegt und eher der Größenordnung der beiden $p < \alpha$ entspricht als dem Rest. Es ist also anzunehmen, dass für diese Aussage ebenfalls das Genre eine Signifikanz hält.

Allerdings ist noch nicht bewiesen, dass die ermittelte Signifikanz wirklich ihren Ursprung in der Wahl der Genres hat. Andere Faktoren, die einen Einfluss auf die Wahrnehmung bzw. Bewertung der Levelgenerierung haben können, sind nicht gleichmäßig über alle Versuchsgruppen verteilt. So könnte beispielsweise die Taktgenauigkeit der Würfelsetzung von Musikern präziser eingeschätzt werden, welche sich wiederum zufällig auf einem Genre anhäufen und auf einem anderen Genre abwesend sind. Zwar hat die Wahrnehmung der Teilnehmer keinen Einfluss auf die Levelgenerierung, aber auf die Ergebnisse des Versuches. Um solche Fälle auszuschließen, wären weitere Untersuchungen mittels *Kruskal-Wallis-Test* bzw. *Mann-Whitney U-Test* (bei nur 2 Gruppen [dPRHB10, Sie56]) auf den verschiedenen Untergruppen einer Versuchsgruppe sinnvoll. Dafür sind die Stichproben aber zu klein. Stattdessen können Vermutungen durch Vergleich der Mittelwerte (siehe Anhang B) aufgestellt werden, deren Exaktheit ob der geringen Stichprobengröße jedoch anzuzweifeln ist. Deswegen wird auf weiteres Verfahren verzichtet und H_1 angenommen.

Auf der Suche nach Abhängigkeiten der Ergebnisse der übrigen Aussagen von anderen Faktoren werden nun die Datensätze nicht mehr den Musikgenre-Versuchsgruppen untergeordnet. Der *Kruskal-Wallis-Test* wird für drei neue Gruppen getrennt nach der Vorerfahrung mit VR bzw. *Beat Saber* durchgeführt. Für die beiden Gruppen mit bzw. ohne Instrumentalerfahrung wird der *Mann-Whitney U-Test* [Sie56, S. 116–126] angewandt. Beide Tests liefern die Wahrscheinlichkeit für das irrtümliche Widerlegen von H_0. Die Werte sind in Anhang C einzusehen. Es ist erkennbar, dass die Vorerfahrung mit VR oder *Beat Saber* keine Auswirkung auf die Ausprägungen der Probandenergebnisse hat. Das Vorhandensein von Instrumentalerfahrung hat jedoch einen signifikanten Einfluss auf die Bewertung der Taktgenauigkeit der Würfelsetzung.

Für **(A1)**, **(A4)** und **(A7)** liegen nach der Auswertung keine bekannten Einflussfaktoren vor, also wird der Durchschnitt der Gesamtdatenmenge interpretiert. Bei den restlichen Aussagen werden die Mittelwerte der signifikant verschiedenen Gruppen betrachtet. Abbildung 5.4 veranschaulicht dies grafisch.

(A7) ermöglicht, wie zuvor erwähnt, die Einschätzung der Objektivität der Teilnehmer. Die Lage des arithmetischen Mittels weicht nur schwach vom erwarteten Wert ab. **(A2)** wird von Probanden mit Instrumentalerfahrung besser bewertet als von den übrigen Teilnehmern. Dies kann daran liegen, dass sie ein tieferes Verständnis vom Takt haben und diesen besser analysieren können. Eine präzisere

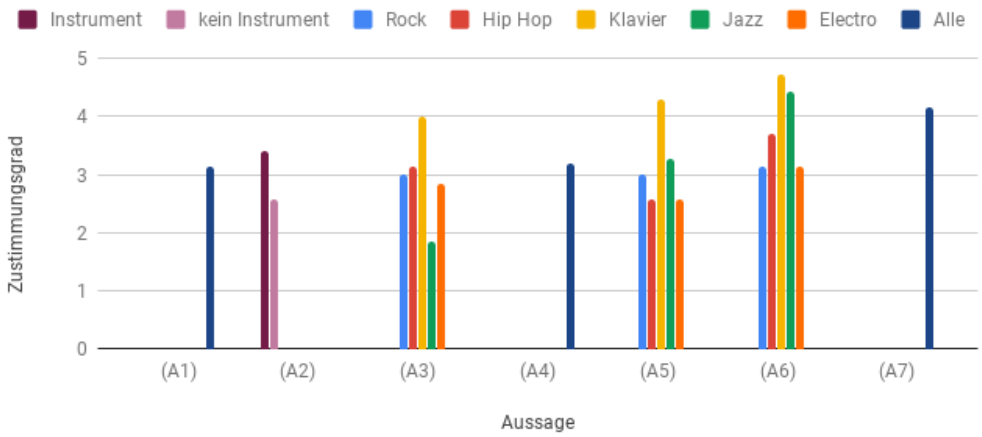

Abbildung 5.4.: Durchschnittliche Zustimmung aller bedeutenden Gruppen (Angefertigt mit: https://docs.google.com)

Einschätzung der Musikalität der Probanden könnte bei weiteren Tests erfolgen, was allerdings für diesen ersten Überblick nicht notwendig ist. Unabhängig davon scheint die Taktgenauigkeit der Anwendung verbesserungswürdig. Da sowohl die Bewegung der Würfel als auch deren Ankunftszeit am Schlagort mit der Musik synchronisiert sind, liegt die Ursache wahrscheinlich bei einer fehlerhaften Takterkennung durch die Audioanalyse-Bibliothek oder der Wahl eines ungeeigneten Tons zur Würfelgenerierung. Ein Einschätzungsfehler vonseiten der Probanden, beispielsweise wegen Überforderung beim Spielen, ist ebenfalls nicht auszuschließen.

Wenig Übereinstimmung mit der zuvor getroffenen Vermutung zeigen die Ergebnisse der Aussagen **(A3)** und **(A4)**. Für die horizontale oder vertikale Veränderung der Würfelplatzierung ist der Algorithmus zur Levelgenerierung verantwortlich. Hier können Anpassungen vorgenommen werden. Was den Unterschied zwischen den beiden Aussagen in Bezug auf den Einfluss von Genres betrifft, lässt sich vermuten, dass die Probanden sensibler auf Tonhöhenabweichungen reagieren als auf Lautstärkeunterschiede. Die Erkennung nicht vorhandener Töne bei der Audioanalyse könnte ebenfalls die Levelgenerierung und damit die Bewertung beeinträchtigen. Die Differenz der Bewertung zwischen den Genres *Jazz* und *Klavier* bei der Tonhöhenkorrelation ist besonders auffällig. Das Tempo beider Musikstücke ist sehr ähnlich. Als mögliche Ursache kommt hier deshalb der markante Unterschied in der Komposition in Frage. Das *Klavier*-Musikstück beinhaltet nur ein Instrument, wohingegen *Jazz* eine Mehrzahl von Instrumenten und zusätzlich Gesang bietet. Die

Vermutung, dass die Audioanalyse für manche *Timbres* und Überlagerungen von Tönen besser funktioniert als für andere, liegt nahe.

(A5) zeigt, dass Melodien und Rhythmen am besten in den beiden eben genannten Genres dargestellt werden. Die zugehörigen Musikstücke haben das langsamste Tempo aus allen gewählten Musikstücken. Das Problem bei den übrigen Versuchsgruppen könnte die Begrenzung des minimalen Zeitabstandes zwischen zwei *Würfelspawns* in den Schwierigkeitseinstellungen sein. Somit erlaubt die Anwendung nicht das Platzieren von Würfeln, falls die Melodie eine zu schnelle Tonabfolge enthält. Dies ist vor allem bei schnellen Musikstücken problematisch und verfälscht das Ergebnis. Ob Melodien nun also tatsächlich nicht richtig vom Programm erkannt werden, oder aber einfach eine Auslassung von Würfeln vorliegt, muss weiter untersucht werden.

(A5) liefert Aufschluss über die wahrgenommene Synchronität der Würfel zur Musik, sofern diese tatsächlich existierende Töne repräsentieren. Abgesehen von **(A7)** hat diese Aussage die höchste Bewertung. Eine mögliche Ursache für die Abweichungen zwischen den Genres ist die Komplexität des jeweiligen Musikstückes. Je langsamer und berechenbarer ein Musikstück, desto besser erahnen Probanden den richtigen Schlagzeitpunkt und müssen sich nicht an der blauen Linie in der VR (siehe Abb. 5.1) orientieren. Allerdings sind technische Fehlerquellen, wie das Erstellen unpassender Würfel, nicht auszuschließen.

Insgesamt kann gesagt werden, dass das Spiel in allen untersuchten Bereichen Optimierungsbedarf hat. Bemerkenswert ist die überdurchschnittlich hohe Bewertung beim Genre *Klavier*. Mit hoher Wahrscheinlichkeit ist dies auf die simple Zusammensetzung des Musikstückes, welches nur ein Instrument enthält, zurückzuführen. Inwiefern weitere Faktoren, wie beispielsweise die Spielparameter und die Affinität der Probanden zu Musik oder Videospielen, Einfluss auf die Bewertung der Levelgenerierung haben, ist ungewiss. Allerdings sind die durchgeführten Untersuchungen ein aufschlussreicher erster Ansatz zur Identifikation von spezifischen Problemen des prototypischen Rhythmik-Videospiels.

6. Zusammenfassung und Ausblick

In dieser Arbeit wird die Konzeption, Implementierung und Evaluation eines prototypischen Rhythmik-Videospiels in der Virtuellen Realität mit automatischer Levelgenerierung auf Grundlage der Analyse von Musikstücken beschrieben. Dafür wird unter anderem die geplante Softwarearchitektur dargestellt. Für die Analyse von Musikstücken werden einige Softwarebibliotheken in Betracht gezogen und vor der Implementierung methodisch verglichen. Die Aufstellung der Vergleichskriterien und die Darlegung wesentlicher Implementierungsdetails sind ebenfalls Gegenstand dieser Arbeit. Die abschließende Evaluation zielt auf die Einschätzung der Funktionsfähigkeit der automatischen Levelgenerierung ab und wird einschließlich der Vorgehensweise und aller Ergebnisse beschrieben.

Der entstandene Prototyp wird als erster Meilenstein in der Umsetzung des gewünschten Rhythmik-Videospiels angesehen. Weitere Iterationen sind nötig, um ein vollwertiges Spielerlebnis zu schaffen. Einige fehlende Features gehen aus der Auswertung der Anmerkungen der Probanden hervor (siehe Anhang D). Oberste Priorität hat dabei die Implementierung von haptischer und visueller Rückmeldung beim Zerschlagen eines Würfels. Es ist weiterhin zu erkennen, dass den Schlagabfolgen bisher eine gewisse Systematik fehlt, die zukünftig in die Levelgenerierung einfließen sollte. Der Wunsch der Abbildung von Musikwiederholungen im generierten Level sowie eine geforderte dynamische Schwierigkeit während der Spielrunde sind nicht ohne Weiteres umsetzbar, da dies eine komplexere Audioanalyse voraussetzt. Das kann nur durch Austausch der verwendeten Softwarebibliothek realisiert werden. Aufgrund der Evaluationsergebnisse ist dieser Schritt jedoch ohnehin sinnvoll. Es empfiehlt sich zukünftig die Implementierung neuer Algorithmen zur Levelgenerierung, die auf unterschiedlichen Bibliotheken basieren, und deren Ergebnisse untereinander zu vergleichen. Die beobachtete schlechte Korrelation der Würfelpositionen mit der Tonhöhe und -lautstärke lässt sich möglicherweise über eine nichtlineare Abbildung lösen, welche die Häufigkeit des Vorkommens bestimmter Wertausprägungen berücksichtigt.

Um ein besseres Spielerlebnis bieten zu können, muss außerdem eine visuelle Überarbeitung erfolgen. Bisher sind alle sichtbaren Objekte im *Level* Prototypen, die ihren Zweck erfüllen, jedoch nicht visuell ansprechend sind. Eine Erweiterung der Anwendung um Punktelisten ermöglicht das Messen von Spielern untereinander und bringt Spielspaß für wettbewerbsorientierte Nutzer. Eine Feinabstimmung aller Spielparameter und Schwierigkeitsgrade hat noch nicht stattgefunden und sollte für die weitere Entwicklung vorgemerkt werden. Neue Schwierigkeitsgrade mit mehr Parametern würden dem Spiel eine weitere Tiefe verleihen.

Aus der Implementierung und den Probandentests geht hervor, dass die Anforderungen aus Kapitel 3.1 erfüllt wurden. Ein höherer Qualitätsanspruch und damit die Aufstellung weiterer nicht-funktioneller Anforderungen an das Endprodukt sowie deren exakte Überprüfung beispielsweise durch System- oder Unittests wird für die Zukunft ebenfalls angedacht. Die Optimierung der Programmlogik und -struktur sollte ebenfalls in Betracht gezogen werden.

Schließlich sollten weitere Iterationen der Anwendung auch weiterhin mittels Probandentests evaluiert werden, da ein Rhythmik-Videospiel von der Wahrnehmung des Spielers abhängt und nicht durch rein objektive und technische Maßnahmen für maximalen Spielspaß abgestimmt werden kann. Die Zielgruppe in die Entwicklung des Spiels mit einzubeziehen, bringt somit einen Mehrwert. Allerdings empfiehlt sich eine größere Stichprobenzahl und eine exaktere Untersuchung aller möglichen Einflussfaktoren für besser verwertbare Ergebnisse.

Anhang

A. Datensätze

Bei den Probandentests fanden fünf verschiedene Versuchsbedingungen Anwendung, die sich allein in der Wahl des Musikstückes bzw. des Genres unterschieden. Neben dem Zustimmungsgrad eines Probanden zu jeder der gegebenen Aussagen wurden Erfahrungen mit VR oder *Beat Saber* (BS), das aktuelle oder ehemalige Spielen eines Instrumentes und Anmerkungen zur Anwendung erfasst. Die erhobenen Datensätze der fünf Versuchsgruppen sind nachfolgend tabellarisch aufgelistet.

A.1. Rock

ID:	**Erfahrung:**	**Instrument:**
R1	keine	nein
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	2	
A3	4	
A4	3	Tempo am Anfang zu hoch
A5	-	
A6	3	
A7	3	
ID:	**Erfahrung:**	**Instrument:**
R2	BS	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	3	
A3	4	Feedback erwünscht, langsame
A4	3	Passagen fragwürdige Würfelsetzung,
A5	4	schnelle Passagen besser

A6	4	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
R3	VR	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	keine
A2	4	
A3	3	
A4	4	
A5	3	
A6	5	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
R4	keine	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	
A2	3	
A3	3	Gleichzeitiges Erscheinen von gleichfarbigen Würfeln mit unterschiedlicher Schlagrichtung wurde beobachtet
A4	4	
A5	5	
A6	4	
A7	2	
ID:	**Erfahrung:**	**Instrument:**
R5	keine	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	
A2	4	
A3	2	
A4	5	keine
A5	2	
A6	4	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
R6	keine	ja
Aussage	**Zustimmung**	**Anmerkungen**

A1	2	
A2	4	
A3	2	Würfeltaktung passte nicht gut zum Genre
A4	2	
A5	2	
A6	0	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
R7	VR	nein
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	2	
A3	3	Musikorientierung war schwierig
A4	4	ohne gespielten Takt (z.B. durch
A5	2	Schlagzeug)
A6	2	
A7	5	

A.2. Hip Hop

ID:	**Erfahrung:**	**Instrument:**
H1	BS	nein
Aussage	**Zustimmung**	**Anmerkungen**
A1	1	
A2	2	
A3	3	Würfelsetzung langsame Passagen
A4	2	besser als schnelle, Feedback
A5	2	erwünscht, Cross-Hand schwierig
A6	1	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
H2	keine	nein
Aussage	**Zustimmung**	**Anmerkungen**

A1	3	
A2	3	
A3	4	Visuell nicht ansprechend, Feedback erwünscht
A4	5	
A5	2	
A6	3	
A7	3	
ID:	**Erfahrung:**	**Instrument:**
H3	keine	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	
A2	2	
A3	3	
A4	3	keine
A5	3	
A6	4	
A7	4	
ID:	**Erfahrung:**	**Instrument:**
H4	VR	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	1	
A3	1	Würfelsetzung nicht einheitlich an
A4	3	wahrgenommenen Elementen
A5	4	(Gesang, Bass), Würfel zu klein
A6	5	
A7	3	
ID:	**Erfahrung:**	**Instrument:**
H5	BS	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	2	
A2	5	
A3	2	
A4	4	Feedback erwünscht

A5	1	
A6	5	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
H6	BS	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	4	
A3	5	
A4	4	Feedback erwünscht, Schlagrichtungen unvorhersehbar
A5	4	
A6	4	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
H7	BS	nein
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	3	
A3	4	
A4	4	Feedback erwünscht
A5	2	
A6	4	
A7	4	

A.3. Klavier

ID:	**Erfahrung:**	**Instrument:**
K1	keine	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	
A2	4	
A3	5	
A4	4	keine
A5	4	
A6	4	

A7	4	
ID:	**Erfahrung:**	**Instrument:**
K2	VR	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	
A2	3	
A3	4	Würfelsetzung manchmal nicht auf
A4	5	dem Taktschlag, Abstand zwischen
A5	4	Blöcken sollte dynamisch sein
A6	5	
A7	3	
ID:	**Erfahrung:**	**Instrument:**
K3	VR	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	3	
A3	4	
A4	4	keine
A5	5	
A6	5	
A7	4	
ID:	**Erfahrung:**	**Instrument:**
K4	VR	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	
A2	2	
A3	4	Wüfelsetzung orientierte sich gut an
A4	4	Leitstimme, beim Fehlen dieser
A5	5	jedoch weniger passend
A6	5	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
K5	VR	nein
Aussage	**Zustimmung**	**Anmerkungen**
A1	2	keine

A2	2	
A3	3	
A4	2	
A5	3	
A6	4	
A7	4	
ID:	**Erfahrung:**	**Instrument:**
K6	keine	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	4	
A3	5	
A4	2	keine
A5	4	
A6	5	
A7	4	
ID:	**Erfahrung:**	**Instrument:**
K7	BS	nein
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	
A2	2	
A3	3	
A4	3	Feedback erwünscht, Farbkontraste auf Würfeln zu gering
A5	5	
A6	5	
A7	5	

A.4. Jazz

ID:	**Erfahrung:**	**Instrument:**
J1	VR	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	keine
A2	2	

A3	0	
A4	0	
A5	4	
A6	4	
A7	3	
ID:	**Erfahrung:**	**Instrument:**
J2	keine	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	2	
A2	3	
A3	2	
A4	4	keine
A5	3	
A6	3	
A7	4	
ID:	**Erfahrung:**	**Instrument:**
J3	VR	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	4	
A3	2	
A4	2	Systematik der Schlagrichtungen bei Würfelabfolgen erwünscht
A5	4	
A6	5	
A7	4	
ID:	**Erfahrung:**	**Instrument:**
J4	BS	nein
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	
A2	3	
A3	2	
A4	2	keine
A5	4	
A6	5	
A7	4	

ID:	**Erfahrung:**	**Instrument:**
J5	VR	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	Abbildung von Wiederholungen erwünscht, Schwierigkeit dynamisch im Level und auch abhängig vom Lied erwünscht, Pausen nicht eindeutig zum Lied erkennbar, Feedback erwünscht
A2	5	
A3	1	
A4	1	
A5	4	
A6	5	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
J6	VR	nein
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	Feedback erwünscht
A2	4	
A3	3	
A4	4	
A5	3	
A6	4	
A7	3	
ID:	**Erfahrung:**	**Instrument:**
J7	BS	js
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	Melodie ungenügend berücksichtigt bei der Würfelsetzung, zu geringe Schwankung der Würfelpositionen, ein- und zweihändige Passagen waren abwechslungsreich, lange Lücken ohne Blöcke nicht erwünscht, Feedback erwünscht
A2	5	
A3	3	
A4	4	
A5	1	
A6	5	
A7	5	

A.5. Electro

ID:	**Erfahrung:**	**Instrument:**
E1	VR	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	2	
A2	3	
A3	-	
A4	4	keine
A5	3	
A6	1	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
E2	BS	nein
Aussage	**Zustimmung**	**Anmerkungen**
A1	4	
A2	3	Für das erste mal schwierig, wenn
A3	2	Blöcke stark Seite wechseln, Pfeile
A4	2	sind nicht groß genug, Würfel gut
A5	4	voraussehbar, sobald man die
A6	4	Melodie erahnen kann
A7	3	
ID:	**Erfahrung:**	**Instrument:**
E3	VR	nein
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	2	
A3	4	
A4	4	keine
A5	2	
A6	4	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
E4	BS	ja
Aussage	**Zustimmung**	**Anmerkungen**

A1	2	
A2	4	
A3	3	Feedback erwünscht, Würfelabfolgen
A4	3	teilweise unangenehm und zu zufällig
A5	1	
A6	4	
A7	4	
ID:	**Erfahrung:**	**Instrument:**
E5	VR	nein
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	3	
A3	2	
A4	4	keine
A5	3	
A6	4	
A7	4	
ID:	**Erfahrung:**	**Instrument:**
E6	BS	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	4	
A3	2	mehr Melodiestimmigkeit erwünscht,
A4	2	Wiederholungen von Abfolgen bei
A5	2	ähnlichen Passagen erwünscht
A6	4	
A7	5	
ID:	**Erfahrung:**	**Instrument:**
E7	VR	ja
Aussage	**Zustimmung**	**Anmerkungen**
A1	3	
A2	3	
A3	4	Feedback erwünscht, Lautstärke
A4	2	nicht besonders wahrnehmbar

A5	3	
A6	1	
A7	4	

B. Mittelwerte

Aus den gegebenen Datensätzen lässt sich das arithmetische Mittel für die Zustimmungsgrade der verschiedenen Untergruppen der Probanden einer Versuchsgruppe bilden und miteinander vergleichen. Die Werte sind auf zwei Nachkommastellen gerundet. Die Untergruppen werden wie folgt bezeichnet:

U1 Alle Probanden der Versuchsgruppe

U2 Alle Probanden ohne Erfahrung mit VR

U3 Alle Probanden mit VR-Erfahrung

U4 Alle Probanden mit Erfahrung in Beat Saber

U5 Alle Probanden, die jemals ein Instrument gespielt haben

U6 Alle Probanden, die noch nie ein Instrument gespielt haben

B.1. Rock

Aussage	U1	U2	U3	U4	U5	U6
A1	3.29	3.25	3.5	3	3.4	3
A2	3.14	3.25	3	3	3.6	2
A3	3	2.75	3	4	2.8	3.5
A4	3.57	3.5	4	3	3.6	3.5
A5	3	3	2.5	4	3.2	2
A6	3.14	2.75	3.5	4	3.4	2.5
A7	4.29	3.75	5	5	4.4	4

B.2. Hip Hop

Aussage	U1	U2	U3	U4	U5	U6
A1	2.71	3.5	3	2.25	3	2.33
A2	2.86	2.5	1	3.5	3	2.67
A3	3.14	3.5	1	3.5	2.75	3.67
A4	3.57	4	3	3.5	3.5	3.67
A5	2.57	2.5	4	2.25	3	2
A6	3.71	3.5	5	3.5	4.5	2.67
A7	4.14	3.5	3	4.75	4.25	4

B.3. Klavier

Aussage	U1	U2	U3	U4	U5	U6
A1	3.43	3.5	3.251	4	3.6	3
A2	2.86	4	2.5	2	3.2	2
A3	4	5	3.75	3	4.4	3
A4	3.43	3	3.75	3	3.8	2.5
A5	4.29	4	4.25	5	4.4	4
A6	4.71	4.5	4.75	5	4.8	4.5
A7	4.14	4	4	5	4	4.5

B.4. Jazz

Aussage	U1	U2	U3	U4	U5	U6
A1	3.43	2	3.5	4	3.2	4
A2	3.71	3	3.75	4	3.8	3.5
A3	1.86	2	1.5	2.5	1.6	2.5
A4	2.43	4	1.75	3	2.2	3
A5	3.29	3	3.75	2.5	3.2	3.5
A6	4.43	3	4.5	5	4.4	4.5
A7	4	4	3.75	4.5	4.2	3.5

B.5. Electro

Aussage	U1	U2	U3	U4	U5	U6
A1	2.86	–	2.75	3	2.5	3.33
A2	3.14	–	2.75	3.67	3.5	2.67
A3	2.83	–	3.33	2.33	3	2.67
A4	3	–	3.5	2.33	2.75	3.33
A5	2.57	–	2.75	2.33	2.75	3
A6	3.14	–	2.5	4	2.5	4
A7	4.29	–	4.5	4	4.5	4

B.6. Alle Probanden

Folgende Mittelwerte wurden aus den zuvor errechneten Mittelwerten der Versuchsgruppen gebildet und besitzen deshalb einen geringen Fehler durch Rundung im Zwischenschritt.

Aussage	U1	U2	U3	U4	U5	U6
A1	3.14	3.06	3.2	3.31	3.14	3.13
A2		3.19	2.6	3.13	3.42	2.57
A4	3.2	3.63	3.2	3.13	3.17	3.2
A7	4.17	3.81	4.05	4.81	4.27	4

C. Statistische Tests

Für die Bestimmung der Signifikanz der Unterschiede zwischen den Ergebnissen der einzelnen Versuchsgruppen wird der *Kruskal-Wallis H-Test* für unabhängige, nicht normalverteilte Stichproben angewandt. Das Signifikanzniveau α beträgt hierbei 0.05. Die erhaltenen Kenngrößen H und p sowie die daraus abgeleitete Bestätigung oder Ablehnung der Signifikanz sind im Folgenden für jede Aussage aufgelistet.

Aussage	**H**	**p**	**signifikant?**
A1	4.2	0.37961	nein
A2	2.7776	0.59571	nein
A3	10.5818	0.03169	ja
A4	3.0619	0.54752	nein
A5	9.6779	0.04622	ja
A6	9.2034	0.05621	wahrscheinlich
A7	1.051	0.90196	nein

Für die Aussagen, bei denen das Musikgenre nicht als Ursache für Unterschiede in der Bewertung in Frage kommt, konnten alle Datensätze der verschiedenen Versuchsgruppen zusammengeführt und eine erneute Untersuchung von Untergruppen gebildet aus den VR- und Instrumenterfahrungen durchgeführt werden. Da bei der Trennung nach Erfahrung mit Musikinstrumenten nur 2 Untergruppen entstehen (ja und nein), wird der *Mann-Whitney U-Test* anstelle des *Kruskal-Wallis H-Test* angewandt. Das Signifikanzniveau bleibt für beide Rechnungen gleich. Folgend sind die Ergebnisse aufgeführt:

Aussage	p_{VR} **signifikant?**	$p_{Instrument}$ **signifikant?**
A1	0.93488 nein	0.88866 nein
A2	0.50687 nein	0.01552 ja
A4	0.54294 nein	0.98404 nein
A7	0.1847 nein	0.37346 nein

D. Zusammenfassung der Anmerkungen

Alle versuchsgruppenunabhängigen Anmerkungen der Probanden wurden zusammengefasst und anhand der Häufigkeit ihres Vorkommens sortiert. Das Ergebnis ist tabellarisch aufgeführt und wird zur weiteren Auswertung herangezogen.

ID	**Anmerkung**	**Anzahl Vorkommen**
F1	Wunsch: haptisches und visuelles Feedback beim Zerschlagen eines Würfels	12
F2	Wunsch: Systematik bei Schlagrichtungen von Würfelabfolgen	2
F3	Wunsch: Abbildung von Musikwiederholungen im Level	2
F4	Richtungssymbole nicht auffällig genug	2
F5	Wunsch: Dynamische Schwierigkeit innerhalb des Levels	2
F6	Hände überkreuzen ist schwierig	1
F7	Spiel ist visuell nicht ansprechend	1
F8	Würfel sind zu klein	1
F9	Schlagrichtungen sind unvorhersehbar	1
F10	Wunsch: Musikstück beeinfluss Schwierigkeit des Levels	1
F11	Wunsch: höhere Schwankung der Würfelpositionen	1

Literaturverzeichnis

[Ack13] Philipp Ackermann: *Computer und Musik: Eine Einführung in die digitale Klang- und Musikverarbeitung*, Springer-Verlag, 1. Aufl., 2013.

[BC03] Grigore C. Burdea und Philippe Coiffet: *Virtual reality technology*, John Wiley & Sons, 2. Aufl., 2003.

[Bri09] Manfred Brill: *Virtuelle Realität*, Springer, 1. Aufl., 2009.

[DBGJ13] Ralf Dörner, Wolfgang Broll, Paul Grimm und Bernhard Jung: *Virtual und augmented reality (VR/AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität*, Springer-Verlag, 1. Aufl., 2013.

[Dem17] Wolfgang Demtröder: *Experimentalphysik 1: Mechanik und Wärme*, Springer, 8. Aufl., 2017.

[dPRHB10] Jean-Baptist du Prel, Bernd Röhrig, Gerhard Hommel und Maria Blettner: *Auswahl statistischer Testverfahren*, *Deutsches Ärzteblatt*, Bd. 107(19):S. 343–348, 2010.

[Duc12] William Duckworth: *A creative approach to music fundamentals*, Kap. 1, S. 3–10, Cengage Learning, 11. Aufl., 2012.

[Epi18] Epic Games Inc.: *Unreal Engine Features*, 2018, URL: https://www.unrealengine.com/en-US/features, besucht am 14.08.2018.

[Est18] Espie Estrella: *An Introduction to the Elements of Music*, März 2018, URL: https://www.thoughtco.com/the-elements-of-music-2455913, besucht am 21.08.2018.

[Gam18] PC Gamer: *VR rhythm game Beat Saber, now the highest rated game on Steam, releasing level editor next week*, Mai 2018, URL: https://www.pcgamer.com/vr-rhythm-game-beat-saber-now-the-

highest-rated-game-on-steam-releasing-level-editor-next-week/, besucht am 07.08.2018.

[GVT08] Mario Gutierrez, Frédéric Vexo und Daniel Thalmann: *Stepping into virtual reality*, Springer Science & Business Media, 1. Aufl., 2008.

[HTC18] HTC: *Vive VR System*, 2018, URL: https://www.vive.com/us/product/vive-virtual-reality-system/, besucht am 12.08.2018.

[Kai18] Erik Kain: *'Fortnite: Battle Royale' Has Made Over $1 Billion As It Completely Dominates Video Game Streaming*, Juli 2018, URL: https://www.forbes.com/sites/erikkain/2018/07/18/fortnite-battle-royale-has-made-over-1-billion-as-it-completely-dominates-video-game-streaming/, besucht am 27.10.2018.

[Ken10] Steven L. Kent: *The Ultimate History of Video Games: From Pong to Pokémon and beyond – the story behind the craze that touched our lives and changed the world*, Three Rivers Press, 2. Aufl., 2010.

[LSW09] Reinhard Lerch, Gerhard Sessler und Dietrich Wolf: *Technische Akustik*, Springer Berlin Heidelberg, 1. Aufl., 2009.

[MS09] Christoph Meinel und Harald Sack: *Multimediale Daten und ihre Kodierung*, in *Digitale Kommunikation*, S. 161–305, Springer, 1. Aufl., 2009.

[Mur18] Sarah Murray: *Worldwide Spending on Augmented and Virtual Reality to Achieve a Five-Year CAGR of 71.6% by 2022, According to IDC*, Mai 2018, URL: https://www.idc.com/getdoc.jsp?containerId=prUS43860118, besucht am 07.08.2018.

[Neu05] Martin Neukom: *Signale, Systeme und Klangsynthese: Grundlagen der Computermusik*, Bd. 2, Peter Lang, 2. Aufl., 2005.

[NLL17] Diederick C. Niehorster, Li Li und Markus Lappe: *The accuracy and precision of position and orientation tracking in the HTC vive virtual reality system for scientific research*, *i-Perception*, Bd. 8(3), 2017.

[Nys14] Robert Nystrom: *Game Programming Patterns*, genever benning, 1. Aufl., 2014.

[PK07] Martin Pichlmair und Fares Kayali: *Levels of Sound: On the Principles of Interactivity in Music Video Games*, in *Situated Play: Proceedings of DiGRA 2007 Conference*, DiGRA, 2007.

[Pla16] PlayStation: *PlayStation VR: Launching October for $399*, März 2016, URL: https://blog.us.playstation.com/2016/03/15/playstation-vr-launching-october-for-399/comment-page-3/, besucht am 07.08.2018.

[Rüg02] Bernhard Rüger: *Test-und Schätztheorie: Statistische Tests*, Bd. 2, R.Oldenbourg Verlag München Wien, 1. Aufl., 2002.

[Sie56] Sidney Siegal: *Nonparametric statistics for the behavioral sciences*, McGraw-Hill, 1. Aufl., 1956.

[Ste18] Steam: *Beat Saber | Steam Store page*, 2018, URL: https://store.steampowered.com/app/620980/Beat_Saber/, besucht am 08.08.2018.

[Sut65] Ivan E. Sutherland: *The Ultimate Display*, in *Proceedings of the IFIP Congress*, S. 506–508, 1965.

[Tea16] Vive Team: *Unveiling the Vive Consumer Edition and Pre-order Information*, Febr. 2016, URL: https://blog.vive.com/us/2016/02/21/unveiling-the-vive-consumer-edition-and-pre-order-information/, besucht am 12.08.2018.

[VD62] Arrey Von Dommer: *Elemente der Musik*, T. O. Weigel Leipzig, 1. Aufl., 1862.

[VR15] Oculus VR: *Samsung Gear VR now available for pre-orders at $99*, Nov. 2015, URL: https://www.oculus.com/blog/samsung-gear-vr-now-available-for-pre-orders-at-99/, besucht am 12.08.2018.

[VR16] Oculus VR: *Oculus Rift Pre-Orders Now Open, First Shipments March 28*, Jan. 2016, URL: https://www.oculus.com/blog/oculus-rift-pre-orders-now-open-first-shipments-march-28/, besucht am 12.08.2018.

[Zen06] Hans-Peter Zenner: *Hören*, in *Neuro- und Sinnesphysiologie*, S. 287–311, Springer, 5. Aufl., 2006.

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Mittweida, den 30. Oktober 2018

William Rapprich