

Hochschule Furtwangen - Fakultät Digitale Medien

## Interaktionsdesign

Wintersemester 2014/15



---

Fabian Gärtner, MIM1

Sarah Häfele, MIM1

Alexander Scheurer, MIM1

Linda Schey, MIM2

Johannes Winter, DIM1

Meike Zöckler, DIM1

Prof. Patricia Stolz

Prof. Dr. Matthias Wölfel

4. Februar 2015

## **Abstract**

# Inhaltsverzeichnis

<b>1 Grundidee</b>	<b>4</b>
1.1 Ideenfindung . . . . .	4
1.2 Zielsetzung . . . . .	4
1.3 Werbebotschaft . . . . .	4
1.4 Strategische Planung . . . . .	4
<b>2 Planung und Skizzierung</b>	<b>5</b>
2.1 Zielgruppenanalyse . . . . .	5
2.2 Tests . . . . .	5
2.3 Technischer Aufbau . . . . .	5
<b>3 Umsetzung des Prototypen</b>	<b>6</b>
3.1 Entwicklungs-Entscheidungen . . . . .	6
3.2 Aufbau . . . . .	6
3.3 DMX-Scheinwerfer . . . . .	8
3.4 Multimonitor Support . . . . .	11
3.5 Grafische Oberfläche . . . . .	13
3.6 Spielfeld und Felder . . . . .	13
3.7 Call to Action . . . . .	15
3.7.1 Der Presentation-Screen . . . . .	15
3.7.2 Spielfeld . . . . .	17
3.8 Personenerkennung mittels Microsoft Kinect . . . . .	18
3.8.1 Aufgetretene Probleme . . . . .	20
3.9 Audiogestaltung und Inspiration . . . . .	22
<b>4 Tag der Medien</b>	<b>25</b>
4.1 kleiner Bericht mit Fotos . . . . .	25
4.2 Praxiserfahrungen . . . . .	25
<b>5 Showreel</b>	<b>26</b>
5.1 Aufgaben-Verteilung . . . . .	26
<b>6 Fazit</b>	<b>27</b>
6.1 Möglichkeiten zur Weiterentwicklung . . . . .	27

# **1 Grundidee**

**1.1 Ideenfindung**

**1.2 Zielsetzung**

**1.3 Werbebotschaft**

**1.4 Strategische Planung**

## **2 Planung und Skizzierung**

### **2.1 Zielgruppenanalyse**

### **2.2 Tests**

### **2.3 Technischer Aufbau**

## 3 Umsetzung des Prototypen

### 3.1 Entwicklungs-Entscheidungen

Zur Entwicklung des Prototypen haben wir uns softwareseitig für die Gameengine Unity<sup>1</sup> entschieden. Gründe hierfür waren, dass der Umgang mit Unity den meisten Projektmitgliedern aus anderen Lehrveranstaltungen geläufig war und die Anbindung externer Softwarebibliotheken (z.B. OpenCV zur Personenerkennung, vgl. Seite 18) sehr einfach ist.

Für die Kommunikation mit den LED-Scheinwerfern wurde *Freestyler*<sup>2</sup>, eine kostenfreien DMX-Lichtsteuerungs-Software verwendet, da es für das verwendete DMX512-Interface in der Produktbeschreibung empfohlen wurde und einwandfrei kompatibel ist. Außerdem bietet Freestyler die Möglichkeit per *SendMessage()* problemlos aus anderen Programmen (hier die »Blinken Tiles«-Anwendung) auf Funktionen zuzugreifen und diese zu steuern.

### 3.2 Aufbau

*Sarah Häfele*

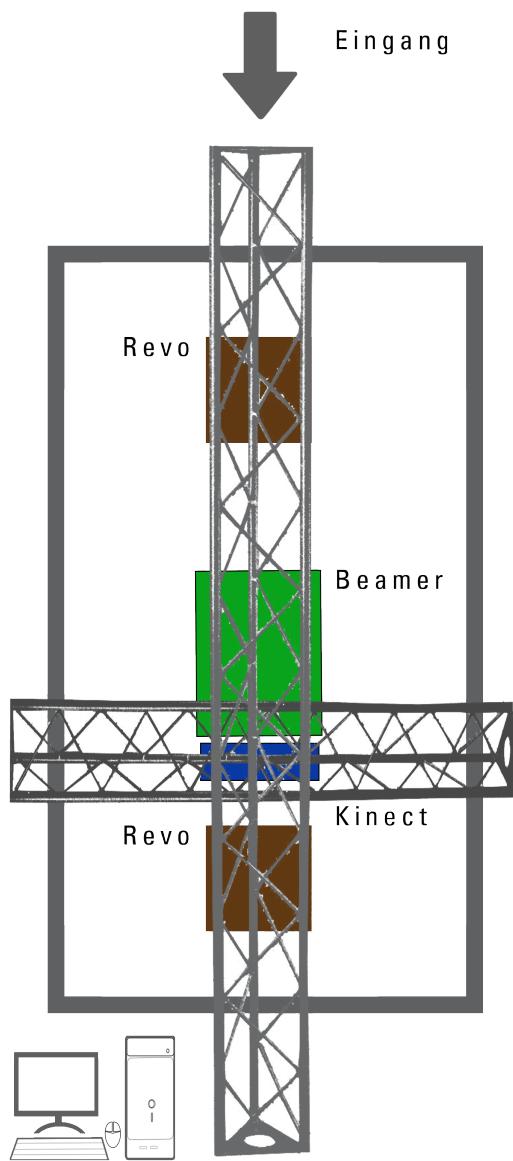
Die Installation besteht aus mehreren Hardware- und Software-Komponenten und soll hier mit Hilfe einer kurzen, modellhaften Übersicht, die den endgültigen Aufbau des Prototypen zeigt, skizziert werden. Die darauf folgenden Kapitel gehen anschließend näher auf die einzelnen Komponenten ein. Der Prototyp der Installation wurde am 20.01.2015 im Rahmen des *Tag der Medien* der Fakultät Digitale Medien aufgebaut und getestet. Auf zwei Ebenen wurden Gerätschaften installiert, damit die Besucher schon beim Eintreten mit akustischen und visuellen Reizen konfrontiert und zum Mitmachen bewegt wurden. Die zwei Grafiken zeigen das Erdgeschoss und den ersten Stock des Gebäudes.

Im ersten Stockwerk hängt über dem Lichthof eine zwei Meter lange Traverse, getragen von zwei Stativen, in deren Mitte ein Epson EB-4950WU Beamer (in der Abbildung Abbildung 1 grün markiert) an einer speziell von der Werkstatt der Hochschule angefertigten Metallhalterung in das Erdgeschoss des Gebäudes strahlt. Der Beamer, eine Leihgabe des Kreismedienzentrums Villingen, kann mit seiner Farbhelligkeit von bis zu 4.500 Lumen und einer Auflösung von 1920

---

<sup>1</sup><http://www.unity3d.com>

<sup>2</sup><http://www.freestylerdmx.be/>



I-Bau, 1. Stock Lichthof  
Aufsicht

Abbildung 1: Aufbau der Installation im 1.Stock des I-Baus

x 1200 Pixeln den dunklen Boden des I-Baus mühelos ausleuchten und ist zudem für längere Betriebszeiten ausgerichtet. Durch die erhöhte Stellung der Traversenstative hängt der Beamer sechs Meter über dem Erdgeschossboden. Auf der Brüstung des Lichthofes liegt im 90° Winkel dazu eine ein Meter lange Traverse. Sie wird nicht ganz mittig zu der längeren Traverse ausgerichtet, damit die mit Spannfix daran befestigte Microsoft Kinect v2 (in der Abbildung in blau markiert) nicht das Bild des Beamers stört und wird durch Holzhalterungen und Doughty-Clamps fixiert. Durch die direkte Anbringung auf dem Geländer des Lichthofes, hängt die Kinect zudem auf 4,5 Meter Höhe und kann so bequem die Bewegungen der im Erdgeschoss befindlichen Personen aufnehmen.

Zudem sind zwei Scheinwerfer (so genannte *Revos*, in Abbildung Abbildung 1 braun markiert ) an der längeren Traverse links und rechts vom Beamer angebracht und versehen den Boden mit zusätzlichen Lichteffekten, indem sie auf den Musiktakt reagieren.

Die Geräte sind per HDMI-Kabel an einen PC angeschlossen, auf welchem das in der Spieleengine Unity geschriebene Kontrollprogramm läuft (hierzu in den folgenden Kapiteln mehr).

Im Erdgeschoss (siehe Abbildung Abbildung 2) wird durch den Beamer ein 5 x 8 flächiges Raster projiziert, welches die einzelnen Töne repräsentiert. Acht LED-Scheinwerfer untermalen die Rhythmuslinie, welche bestimmt, wann ein Ton abgespielt wird. Die Scheinwerfer sind durch ein DMX-Interface untereinander und mit dem Computer verbunden und werden durch ein eigenes Programm gesteuert (siehe Unterabschnitt 3.3).

Ein großer LED-Monitor zeigt den Besuchern Informationen an und dient zusätzlich als Motivation (siehe auch Unterabschnitt 3.7).

Die Musik wird durch einen mobilen, jedoch leistungsstarken Radiorekorder (auch: *Boombox* genannt) abgespielt.

### 3.3 DMX-Scheinwerfer

*Linda Schey*

Für die LED-Scheinwerfer wurden acht *Eurolite LED CLS-18 QCL RGBW* verwendet, die jeweils über einen dreipoligen XLR- Ein- bzw. Ausgang verfügen, so dass sie in Reihe geschaltet über das DMX512 Protokoll mit zwölf Kanälen gesteuert werden können. Ein *Eurolite USB-DMX512-PRO* Interface dient als Schnittstelle zwischen den Lichtern und dem PC. Angesteuert werden die Lich-

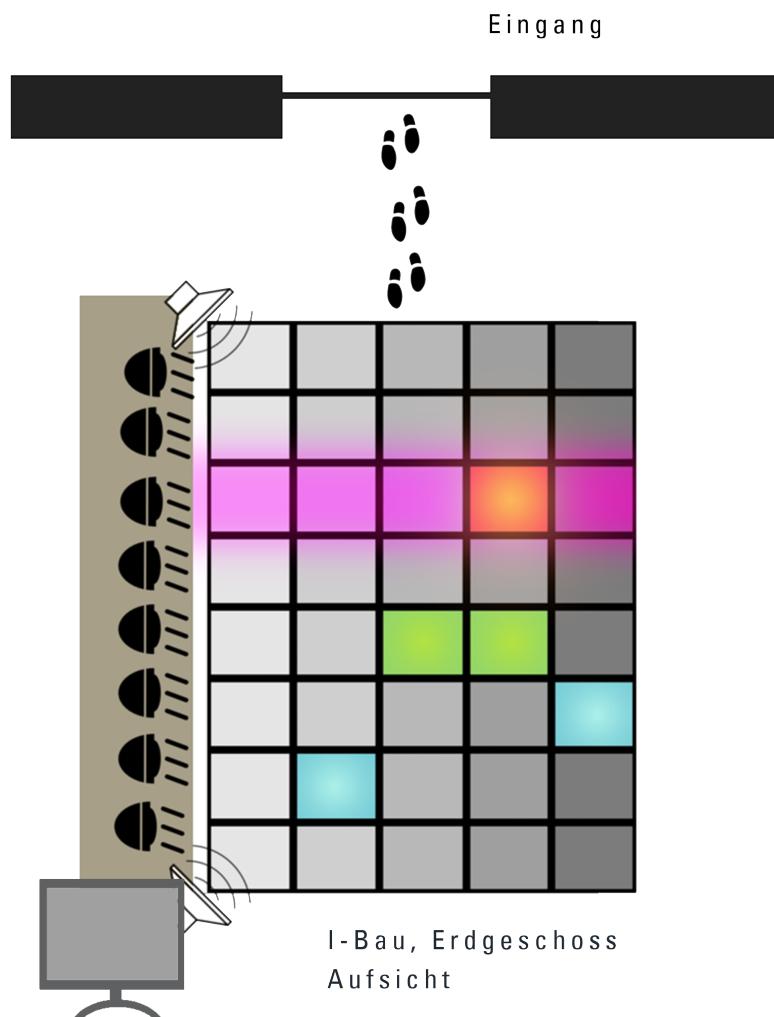


Abbildung 2: Aufbau der Installation im Erdgeschoss des I-Baus

ter dann mittels der in Unterabschnitt 3.1 bereits erwähnten Software Freestyler. Abbildung 3 zeigt grob wie die drei Komponenten zusammenhängen. Es wird hier den Pfeilen der Abbildung entsprechend nur in eine Richtung kommuniziert, da das Verhalten der Scheinwerfer ausschließlich durch die Applikation gesteuert wird und keine rückläufige Kommunikation nötig ist. Jeder Scheinwerfer



Abbildung 3: Verknüpfung der Komponenten

besteht aus 18 LEDs die in drei Reihen mit jeweils sechs LEDs angeordnet sind. Zwei Zeilen, also sechs LEDs, werden in der Applikation jeweils als ein Segment verwendet und kann mit vier Farbkanälen separat angesteuert werden. Insgesamt werden 12 Farbkanäle pro Scheinwerfer eingesetzt. Innerhalb der „Blinken Tiles“ Anwendung gibt es zwei Klassen, die hauptverantwortlich für die Lichtsteuerung sind. *Spot.cs* und *LightController.cs*. *Spot.cs* repräsentiert in der Anwendung einen LED-Scheinwerfer. Diese Klasse verwaltet die Farbwerte für jeden einzelnen Spot und dessen Segmente. In der Klasse *LightController.cs* werden die Farbwerte für alle acht Scheinwerfer in einem Array gespeichert. Dieses Array wird mit den Farbwerten der einzelnen Scheinwerfer befüllt. Die Farbwerte der Scheinwerfer, werden regelmäßig pro Frame aktualisiert. Dabei wird überprüft welcher der Scheinwerfer gerade aktiv sein muss und anhand einer Zeitreferenz welches Segment des Scheinwerfers aktiviert sein soll. Alle anderen Scheinwerfer und deren Segmente werden aus Schwarz gestellt, also nicht leuchtend. Jeder Scheinwerfer besteht aus 18 LEDs die in drei Reihen mit jeweils sechs LEDs angeordnet sind. Zwei Zeilen, also sechs LEDs, werden in der Applikation jeweils als ein Segment verwendet und kann mit vier Farbkanälen separat angesteuert werden. Insgesamt werden also 12 Farbkanäle pro Scheinwerfer eingesetzt. Innerhalb der „Blinken Tiles“ Anwendung gibt es zwei Klassen, die hauptverantwortlich für die Lichtsteuerung sind. *Spot.cs* und *LightController.cs*. *Spot.cs* repräsentiert in der Anwendung einen LED-Scheinwerfer. Die Klasse *Spot.cs* verwaltet die Farbwerte für jeden einzelnen Spot und dessen Segmente. In der Klasse *LightController.cs* werden die Farbwerte für alle acht Scheinwerfer in einem Array gespeichert. Die Farbwerte der

Scheinwerfer, werden regelmäßig pro Frame aktualisiert. Dabei wird überprüft welcher der Scheinwerfer gerade aktiv sein muss und anhand einer Zeitreferenz welches Segment des Scheinwerfers aktiviert sein soll (*UpdateFaderValues()*). Alle anderen Scheinwerfer und deren Segmente werden auf Schwarz gestellt, nicht leuchtend. Um die Scheinwerfer aus der Anwendung „Blinken Tiles“ heraus zu steuern wird ein Zwischenschritt über die Software Freesyler gemacht, da diese Software die Ansteuerung der einzelnen Segmente eines jeden Scheinwerfers sehr komfortabel gestaltet. Um aus „BlinkenTiles“ mit Freestyler zu kommunizieren wurde eine DLL (*LetThereBeLight.dll*) erstellt, die per *FindWindow()* und *SendMessage()* mit zwischen „Blinken Tiles“ und Freestyler vermittelt.

### 3.4 Multimonitor Support

*Linda Schey*

Zu Beginn des Projektes wurde versucht eine möglichst große Projektionsfläche zu erhalten. In Ermangelung eines Beamers, der die gewünschte Größe projizieren konnte wurde auf drei Beamer zurückgegriffen, die dann nebeneinander ein jeder ein Drittel der Applikation darstellen sollte. Um dies zu realisieren musste aber das komplette Bild das projiziert werden sollte in drei einzelne Bilder geteilt werden. Da die Beamer nicht nebeneinander, sondern über einander aufgebaut werden sollten, damit das komplette Bild hoch und breit genug ist müssen die erhaltenen Bilder jeweils um 90° gedreht werden. Dafür wurde in Unity eine extra Kamera in die Szene eingefügt, die die Szene als RenderTexture aufnimmt. Diese RenderTexture wurde dann in drei einzelne Texturen getrennt. Jede Textur um 90° gedreht und dann nebeneinander als Gui-Elemente dargestellt. Soweit war diese Funktionalität auch schon implementiert, bis auf kleine Justierungen beim Zuschneiden der einzelnen Teil Texturen. Ein anderer Ansatz war drei um 90° gedrehte Kameras in die Szene zu integrieren die jeweils ein Drittel des Spielfeldes filmten. Daraus erhielt man drei RenderTextures, die dann nebeneinander als Gui rendern werden konnten. Dafür wurde zunächst mit einem Beispiel Objekt gearbeitet, das ein Material zugewiesen bekam, welches es bei der Entwicklung vereinfachte zu erkennen, ob die Kameras richtig ausgerichtet waren und dann auch entsprechend richtig auf die Gui gerendert wurden.

Die drei Kameras, mussten jedoch von Hand auf die Szene eingestellt werden, so dass es auch hier wieder zu Ungenauigkeiten hätte kommen können. Ande-

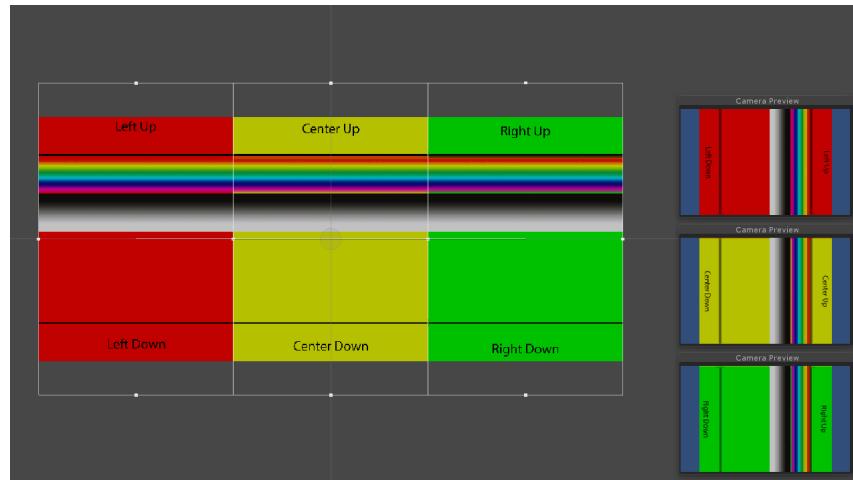


Abbildung 4: Aufbau der Szene in Unity mit den drein RenderTexture Ausschnitten am Beispiel eines Dummy Objektes

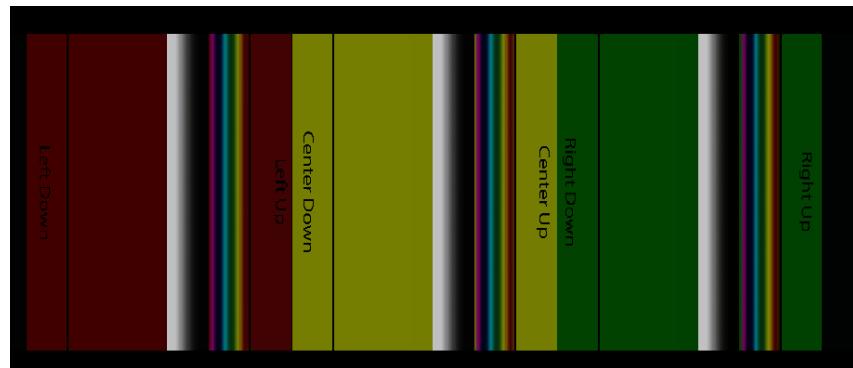


Abbildung 5: RenderTextures nebeneinander auf die Gui gerendert

rerseits wäre man hier flexibler beim Ausrichten der Szene auf die drei Beamer gewesen, in dem man zum Beispiel die Übergänge zwischen den drei Texturen jeweils so hätte Platzieren können, dass sie auf einem Spalt der Felder des Spielfeldes gelegen wären. Auch dieser Ansatz wurde umgesetzt und hat mit funktioniert allerdings auch mit kleinen Ungenauigkeiten.

Beide Ansätze waren jedoch nicht zu 100 Prozent exakt, was das Aufteilen der Szene in drei Teil Texturen betraf. Außerdem standen keine drei identischen Beamer zur Verfügung, sodass es unter den einzelnen Beamern Unterschiede bei der Lichtintensität gab. Da anstatt der Lösung mit den drei Beamern doch noch ein geeignetes Gerät gefunden wurde, das alleine die gewünschte Projektionsfläche erbrachte, wurde die Idee mit den drei Beamern, sowie die bisherige Implementierung dafür verworfen und ist im finalen Stand des Projektes nicht mehr enthalten.

### 3.5 Grafische Oberfläche

Teil der Grafischen Oberfläche ist das Spielfeld (Abbildung 6), mit dem die Nutzer wie im Konzept beschrieben, interagieren.

### 3.6 Spielfeld und Felder

*Alexander Scheurer*

Das Spielfeld und die Felder werden von zwei Klassen bestimmt, die TileController und TileBehaviour Klasse. Die TileController Klasse ist die Zentrale Klasse, sie beinhaltet die Spiellogik, baut das Spielfeld auf, verarbeitet Eingaben (von Personenerkennung, Maus, Tastatur, Konfiguration und Idle-Mode).

Die Spiellogik umfasst überwiegend das Timing der Abläufe und in Abhängigkeit davon das Zuweisen von Zustände an die einzelnen Felder. Die Zustände die zugewiesen werden beschreiben zum Beispiel ob auf diesem eine Person steht, ob derzeit sich der Zeitindex in der Reihe dieses Feldes befindet oder ob sogar beides vorher genannte zutrifft, was zur Folge hätte dass auch der entsprechende Ton zu dem Feld abgespielt werden muss. Die TileController Klasse weist den Feldern den entsprechenden Zustand sequenziell zu:

1. Alle Felder werden zurückgesetzt (Clear pass)

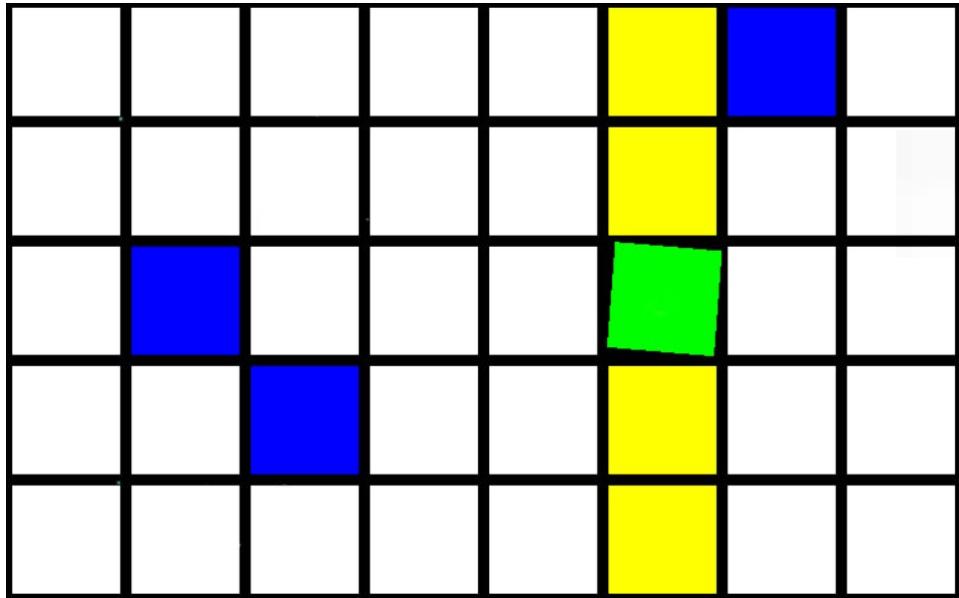


Abbildung 6: Das Spielfeld

2. Der Zustand für Zeitimpuls wird bei den entsprechenden Feldern gesetzt (Timer pass)
3. Der Zustand für Feldern auf denen Personen erkannt wurden, werden gesetzt (People pass)
4. Der vereinigte Zustand von 2. und 3. wird gesetzt (Shake'n'Play pass)

Verarbeitet wird dieser Zustand von jedem Feld selbst mit Hilfe der TileBehaviour Klasse. In dieser passiert die Umsetzung von Zustand zu Aktion, also welcher Shader (Farbe) für das Feld gesetzt wird oder soll es wackeln weil es den entsprechenden Zustand vom Shake'n'Play pass zugewiesen bekommen hat.

Zum Start des Spiels hat der TileController die Aufgabe das Spielfeld aufzubauen, dies macht er anhand von JSON-Konfigurationsdateien einmal für die Grundkonfiguration als auch Konfigurationen für jeden Song. Die Grundkonfiguration definiert unter anderem welche Ausmaße das Spielfeld hat, oder auch welche Farbe einem bestimmten Feldzustand zugewiesen ist. Die Song-Konfiguration beinhaltet zum Beispiel die Angabe über die Beats pro Minute

und die Pfade zu den Audiodateien.

Benutzereingaben nimmt der TileController entgegen und veranlasst die entsprechende Aktion. Zum Beispiel einen Songwechsel oder das verarbeiten eines Feldzustandes von der Personenerkennung oder dem Idle-Node.

## 3.7 Call to Action

*Sarah Häfele*

Die Vermutung, dass bei einer Installation mit Musik und der Notwendigkeit, sich zu bewegen, die Hemmschwelle diese zu nutzen höher liegt, liegt nahe. Neben Lichteffekten durch die Scheinwerfer und dem offiziellen Spielfeld dienen deswegen weitere Komponenten zur *Call to Action*, also zur Motivation der Besucher. Diese sollen nicht nur zum Mitmachen bewegen, sondern auch gleichzeitig anleitend wirken.

### 3.7.1 Der Presentation-Screen

Der Presentation-Screen ist als eine Art Präsentationsfläche gedacht, die den Nutzern weitere Informationen zur Installation liefern soll. In der Spieleengine Unity mit C# entwickelt, kann sie als eigenständige Anwendung auf einem großen Bildschirm neben dem Spielfeld angezeigt werden. Die Anwendung besteht bislang aus vier Modi, angefangen mit dem Startmenü, welches das Logo der Installation und weitere Menüpunkte anzeigt. Der *How-To*-Modus zeigt ein Video welches kurz erklärt, wie das Spielprinzip funktioniert, erstellt von Fabian Gärtner in Adobe Premiere mit Grafiken von Sandra Beuck. Im *Behind-the-Scenes*-Modus wird ein aktuelles Bild der Kinect sowie das Beamerbild gezeigt um zu erklären, wie die Technik dahinter funktioniert. Die Anwendung kommuniziert dabei über das Internet mit der BlinkenTiles-Anwendung. Zuletzt zeigt der *Time-Lapse*-Modus ein Video des Aufbaus der Installation, aufgenommen mit einer GoPro Kamera und geschnitten von Fabian Gärtner.

Die Modi können in einem Loop abgespielt werden, wobei nach einiger Zeit oder wenn das Video zu Ende ist das Programm in den nächsten Modus springt. Wenn die Hauptanwendung von BlinkenTiles jedoch in den Challenge-Modus wechselt, verändert sich auch der Presentation-Screen: er springt zum *How-To*-Video und zeigt zudem Information zum abgespielten Lied an. Dies beinhaltet den Musiktitel und einen Countdown (siehe Abbildung Abbildung 8).

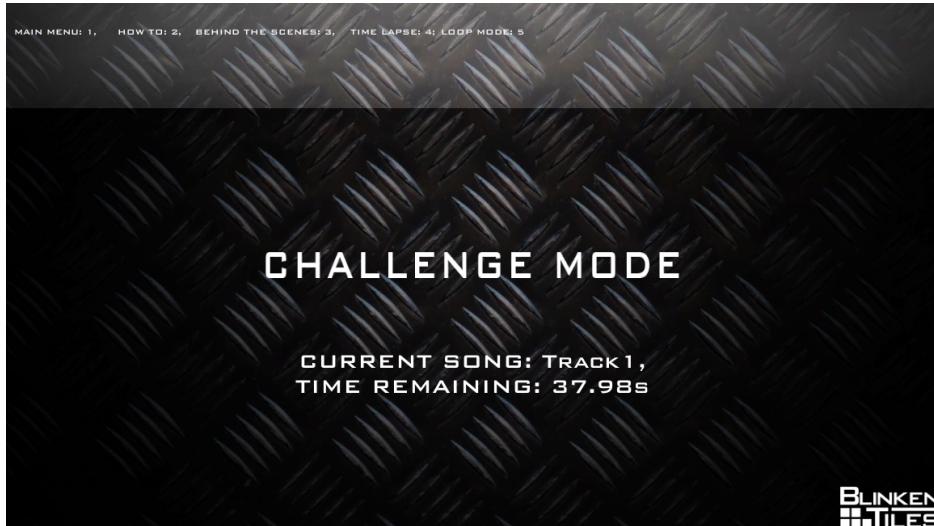


Abbildung 7: Presentation-Screen im Challenge Mode

Die BlinkenTiles-Anwendung kommuniziert mit einem XML-File über das Internet mit dem Presentation-Programm. Die Hauptanwendung erstellt das File und legt es in einem Netzwerkordner ab. Dieser Teil wurde von Linda Schey implementiert. Die Presentation-Anwendung greift auf das File zu, parst die Textinformationen und speichert sie in entsprechenden Variablen. Abbildung Abbildung 8 zeigt ein minimales Beispiel, welche Informationen so an den Presentation-Screen übermittelt werden können.

```
<?xml version="1.0"?>
<!DOCTYPE NetworkSet [
    <!-- Challenge Mode -->
    <!-- Demo Time -->
    <!-- Song -->
    <!-- Length -->
]>
<NetworkSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <ChallengeMode>true</ChallengeMode>
    <DemoTime>16</DemoTime>
    <Song>Track1</Song>
    <Length>42</Length>
</NetworkSet>
```

Abbildung 8: XML-File mit aktuellen Informationen für den Presentation-Screen

Die Variablen werden dann an die einzelnen Codeelemente weitergegeben. Der Countdown wird durch die übermittelte Länge des Liedes ermittelt und dann selbstständig dekrementiert. Endet der Challenge Mode, wird der Loop normal fortgesetzt.

### **3.7.2 Spielfeld**

idle animation Code  
als presentation

### **3.8 Personenerkennung mittels Microsoft Kinect**

*Fabian Gärtner*

Zur Erkennung von Personen auf dem Spielfeld mittels Microsoft Kinect konnten keine Standardfunktionen verwendet werden, da das Kinect-SDK den Einsatz der Kinect aus der Vogelperspektive nicht vorsieht. Stattdessen muss das von der Kinect aufgenommene Tiefenbild mittels Bildverarbeitungstechniken auf Personen bzw. verallgemeinert gesprochen auf Objektkonturen hin untersucht werden. In BlinkenTiles sind dafür die drei Klassen »KinectManager«, »BlobDetection« und »BlobDetectionThread« zuständig. Erstere fragt unter Anwendung des Kinect-SDKs von Microsoft das Tiefenbild (30 Bilder pro Sekunde mit einer Auflösung von 512 x 424 Pixel) und – bei Bedarf – das Farbbild (15 Bilder pro Sekunde mit einer Auflösung von 1920 x 1080 Pixel) der Kinect ab. Das Tiefenbild ist dabei ein zweidimensionales Array, das mit Werten zwischen 0 und 8000 die pro Pixel gemessene Tiefe in Millimeter angibt. Da sich die Farbkamera von der Tiefenkamera in Auflösung und Bildwinkel unterscheidet, müssen die Werte des Farbbildes, um beide Bilder übereinander legen zu können, mittels einer vom Kinect-SDK bereitgestellten Funktion umgerechnet werden. Auch diese Funktionalität und die Möglichkeit, Tiefenbilder in Form von Dateien abzuspeichern und für den Fall, dass die Kinect nicht angeschlossen ist, als Testdaten zu verwenden, stellt die Klasse bereit.

Die Weiterverarbeitung des Tiefenbildes und damit die eigentliche Bildverarbeitung findet in der Klasse »BlobDetectionThread« statt und wird zur Laufzeit in einem eigenen Thread ausgeführt. Dies war zu Beginn der Entwicklung noch notwendig, da die anfangs prozessorlastige Objekterkennung zu einer niedrigen Bildrate in der Hauptanwendung führte und damit auch Auswirkungen auf den Spielablauf und das Abspielen der Musik hatte. Dieser Prozess konnte im Verlauf der Entwicklung aber stetig optimiert werden, sodass die Bildverarbeitung auf eine Dauer von lediglich 12 Millisekunden pro Tiefenbild reduziert werden konnte. Mit etwas mehr als 80 Bildern pro Sekunde liegt die Objekterkennung damit bereits weit über der Bildrate der Kinect mit rund 30 Bildern pro Sekunde und kann ohne Verzögerung und in Echtzeit stattfinden. Die Auslagerung in einen eigenen Thread wurde aber beibehalten, sodass die Arbeitslast auch weiterhin gleichmäßig auf mehrere CPU-Kerne verteilt wird und mögliche Verzögerungen bei der Abfrage der Tiefenbilder keine Auswirkungen auf die Hauptanwendung und den Spielablauf haben.

Hauptsächlich für die Bildverarbeitung verwendet wird die kostenfreie Open-Source-Bibliothek OpenCV, die über den C-Sharp-Wrapper EmguCV angesteuert wird. OpenCV erlaubt die Anwendung von Algorithmen und grundlegenden mathematischen Operationen auf Matrizen und Bilder. Konkret wird bei BlinkenTiles zunächst das als Array vorliegende Bild gefiltert und normalisiert. Die Filterung arbeitet anhand vom Nutzer eingestellter Werte für minimale und maximale Tiefe. Alle Werte über und alle Werte unter diesen Vorgaben werden auf 0 gesetzt, sodass diese die Erkennung nicht weiter stören. In der Praxis kann damit ein stark reflektierender Boden herausgefiltert werden. Die anschließende Normalisierung hilft, den Kontrast des Tiefenbildes zu erhöhen und so die dann folgende Binarisierung des Bildes zu optimieren. Auf das binarisierte Bild wird letztlich die Konturerkennung von OpenCV angewendet, die Objekte ab einer bestimmten Größe erkennt und ein entsprechendes Konturrechteck zurückliefert. Dieses Rechteck kann anschließend mit den Feldern der Matrix abgeglichen werden und bei Überschneidungen zwischen Konturrechteck und einem Feld das Feld aktiviert werden. Jeder dieser Schritte ist vom Nutzer über eine im Folgenden näher erläuterte GUI separat darstellbar und konfigurierbar, um eine optimale Objekterkennung zu ermöglichen. Des Weiteren werden regelmäßig die einzelnen Schritte als Bilder herausgespeichert, sodass sie den Spielern über den Präsentationsbildschirm angezeigt werden können.

Die Klasse »BlobDetection« schließlich ist hauptsächlich für die Verwaltung der grafischen Benutzeroberfläche (GUI) der Bildverarbeitung und die Verwaltung der eben beschriebenen Klassen zuständig. Dieser Teil der GUI, wie er in Abbildung X auf Seite X dargestellt ist, nimmt die linke Hälfte der gesamten GUI von BlinkenTiles ein und erstreckt sich damit über einen Monitor. Die GUI stellt das Tiefenbild in seiner ursprünglichen Form oder nach einem der zuvor beschriebenen Verarbeitungsschritte, bspw. in binarierter Form oder mit farbigen Konturrechtecken um erkannte Objekte, dar. Des Weiteren erlaubt die GUI über eine Reihe von Slidern die Einstellung von Filterwerten, Spielfeld- und Feldergröße sowie Spielfeldposition. Ebenso kann ein Raster über dem Tiefenbild dargestellt werden, bei dem alle aktivierten Felder, also alle Felder auf denen Personen erkannt wurden, farbig markiert sind. Um dieses Raster dem vom Beamer erzeugten Spielfeld anzupassen, also um Beamer und Kinect zu synchronisieren, besteht die Möglichkeit eine Überlagerung von Farb- und Tiefenbild anzuzeigen, die dann die Anpassung der Größe und Ausrichtung des Rasters auf dem Tiefenbild mittels der Slider erlaubt. Alle Einstellungen können auch zuvor über die externe Konfigurationsdatei geladen werden.

### **3.8.1 Aufgetretene Probleme**

Am Tag der Medien zeigte sich, dass die Tische, auf denen u. a. die LED-Scheinwerfer standen, aufgrund ihrer hellen Oberfläche nicht aus dem Tiefenbild herausgefiltert werden konnten und daher ebenfalls als Objekte erkannt wurden. Dies führte dazu, dass Personen, die auf einem der Felder in der Nähe der Tische standen, zusammen mit den Tischen als ein großes Objekt erkannt und dadurch gelegentlich zu viele Felder gleichzeitig aktiviert wurden. Beheben ließe sich das Problem, indem das Tiefenbild vor der weiteren Verarbeitung auf den relevanten Bereich, also auf Spielfeldgröße zugeschnitten wird. Auch hierfür könnten entsprechende Regler in die GUI integriert werden, sodass der Zuschnitt manuell und je nach Bedarf eingestellt werden kann.

Ein weiteres Problem ist die Perspektive, vor allem dann, wenn die Kinect nicht mittig über dem Spielfeld hängt. So funktioniert die Erkennung von Personen, die sich direkt unter der Kinect befinden zwar tadellos, je weiter eine Person aber von der Kinect entfernt ist, desto mehr Felder deckt sie aufgrund der schrägen Aufsicht gleichzeitig ab. Eine kurzfristige Lösung am Tag der Medien war es, die Felder in der Anwendung größer als in der Realität einzustellen. Des Weiteren wird bereits nicht das ganze Feld auf Überschneidungen mit Konturen überprüft, sondern lediglich ein kleinerer (ebenfalls einstellbarer) Bereich in der Mitte der Felder. Um das Problem aber besser zu lösen, wäre entweder eine zweite Kinect oder eine perspektivisch korrekte Berechnung und ein unsymmetrisches Raster notwendig. So könnten bspw. die Felder des Rasters in den Randbereichen vergrößert oder die Tiefenfilterung zu den Rändern hin verstärkt werden, sodass in diesen Bereichen nur noch die Beine von Personen für die Überschneidung relevant sind, nicht aber der Kopf, der möglicherweise ein anderes Felder überdeckt. Welche Methode sich hier am besten eignet, müsste durch weitere Tests herausgefunden werden.

Störend war auch, dass die Einstellungen zwar wie beschrieben aus einer Konfigurationsdatei geladen werden können, die Werte aber momentan noch per Hand eingetragen werden müssen. Interessant neben einem Button zum Speichern der Einstellungen könnte für eine solche Installation daher auch eine automatische Kalibrierung sein, um bei einem Neustart die manuelle Neukonfiguration zu vermeiden. Technisch ließe sich das über ein vom Beamer dargestelltes Testmuster lösen, das von der Bildverarbeitung erkannt wird und das Raster entsprechend dimensionieren und positionieren lässt. Auch die optimalen Werte für die Filterung könnten so automatisiert ermittelt werden.

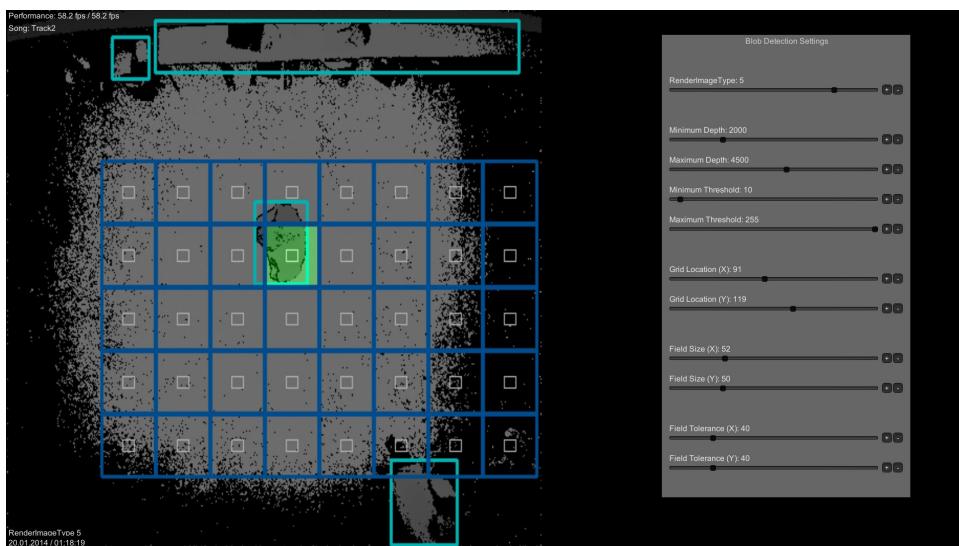


Abbildung 9: Schneiden der Spuren in Ableton Live

3

---

<sup>3</sup>caption

### **3.9 Audiogestaltung und Inspiration**

*Johannes Winter*

Das Konzept der Installation beruht auf einem klassischen Step-Sequenzer. Solche Sequenzer steuern die Klangerzeugung eines Synthesizers dahingehend, dass sowohl Rhythmus als auch Tonhöhe programmiert werden können. Der Name bezieht sich dabei auf die einzelnen »Steps« die mit Tönen belegt werden können. Klassische analoge Step-Sequenzer wie der Roland TB 303 bieten 16 Schritte an, womit also pro Durchlauf 16 Töne gespielt werden können. Somit entspricht jeder Schritt einer 16tel Note eines Taktes. Es ist hervorzuheben, dass bei einem Step-Sequenzer die Noteneingabe nicht unmittelbar zu einer Klangerzeugung führt. Stattdessen tastet ein Impuls nacheinander alle „Steps“ ab und übermittelt die Daten an den Klangerzeuger. Nachdem der Impuls einmal durchgelaufen ist, beginnt der er wieder bei dem ersten Step. Dadurch entstehen repetitive Tonfolgen, sogenannte Loops. Step-Sequenzer haben gerade aufgrund dieser Beschränkung zahlreiche Genre wie Acid House, Electronic Body Music und Drum and Bass entscheidend geprägt.

Das Konzept der Installation weicht in mehreren Punkten vom klassischen Step-Sequenzer ab. Zum Einen gibt es statt 16 Schritten nur 8. Außerdem ist die Tonauswahl auf 5 Tonhöhen pro Schritt begrenzt. Des Weiteren wird ein Ton nur so lange aktiviert, wie auch eine Person auf dem entsprechenden Feld steht. Das musikalische Konzept berücksichtigt diese Einschränkungen. Da nur 5 Töne gespielt werden können und auch nur 8 Steps zur Verfügung stehen, wird ein Backing Track benötigt, der eine harmonische Grundlage für das Spiel des Instrumentes bietet.

Anspruch der Installation war es aber weniger ein komplexes Instrument zu bieten, viel mehr eine musikalische Spielwiese. Aus diesem Grund liegt die Entscheidung nahe, sich die technischen Beschränkungen zu Nutze zu machen. Es wurde also auf eine Skala zurückgegriffen, die einerseits nur 5 Töne hat und andererseits keine Halbtorschritte aufweist: die Anhemitonische Pentatonik. Der Vorteil liegt darin, dass keine kleinen Sekunden und Tritoni gespielt werden können, die für unsere Hörgewohnheiten »unrein« klingen. Durch diese Maßnahme wurde also sichergestellt, dass unabhängig von Menge und Position der Benutzer ein recht harmonisches Gesamtbild entsteht, auch wenn dadurch auf die Leittonwirkung einer Diatonik oder Hemitonischen Pentatonik verzichtet werden muss. Außerdem war der Tonumfang natürlich auf eine Oktave be-

schränkt, mehr als 5 Tonhöhen hätten das musikalische Ergebnis interessanter gestaltet aber auch die Nutzerfreundlichkeit eingeschränkt. Da die Installation in erster Linie intuitiv bedienbar sein sollte, wurde der Tonumfang nicht weiter ausgebaut.

Für den experimentellen Modus der Installation wurden drei Backing Tracks mit jeweils 5 zugehörigen Tonhöhen vorproduziert. Der durchlaufende Impuls wurde auf die Geschwindigkeit des entsprechenden Backing Tracks angepasst und die zugehörigen Töne auf die Felder gemappt. Da der Klangerzeuger kein Synthesizer sondern ein Sampler war, mussten die Einzeltöne im Voraus synthetisiert und gerendert werden. Grundsätzlich wurden eher sphärische Klänge mit langem Nachhall (entweder Reverb oder Hüllkurvengenerator) eingesetzt, da die Benutzer der Installation keinen Einfluss auf die Länge des Tons hatten. In Hüllkurvenparametern gesprochen musste also der Attack deutlich hörbar sein um ein auditives Feedback zu geben, die Sustainlautstärke musste relativ schnell erreicht werden und wesentlich leiser als der Attack sein, damit sich bei mehreren Personen die Sounds nicht zu stark überdeckten. Dadurch wurden relativ perkussive Klänge erzeugt, die durch den Nachhall eine gewisse Stetigkeit erreichten. Die Backing Tracks bildeten das rhythmische und harmonische Grundgerüst und wurden ebenfalls im Voraus produziert und gerendert. Dabei wurde Wert darauf gelegt, zwar eine harmonische Orientierung zu bieten, der Backing Track sollte jedoch keine komplexe harmonische Struktur aufweisen. In den Backing Tracks wurde also ebenfalls weitgehend auf die oben erwähnte Pentatonik zurückgegriffen, um aber ein wenig musikalische Spannung zu kreieren, wurden an ausgewählten Stellen auch Töne aus diatonischen Skalen verwendet.

Für den Challenge Modus wurden ebenfalls drei Backing Tracks und passende Töne vorproduziert, da jedoch die technischen und konzeptionellen Bedingungen andere waren, soll darauf noch näher eingegangen werden. Das Ziel des Challenge Modus ist es, mit der Installation bekannte Lieder nachzuspielen, ähnlich wie bei Guitar Hero. Problematisch ist jedoch, dass die Reaktionsgeschwindigkeit der Benutzer nicht so schnell ist wie es eigentlich nötig wäre. Das größte Hindernis ist jedoch, dass die Nutzer die Installation verlassen müssen, wenn sie keinen Ton spielen wollen. Sobald sie auf irgendeinem Feld stehen und der Impuls dieses erreicht wird der Ton getriggert, auch wenn er falsch ist. Anders gesagt: Die Zustände der Nutzer auf dem Spielfeld sind analog, benötigt wären aber digitale Zustände, an oder aus. Aus diesem Grund mussten Songs gefunden werden, die einerseits recht bekannt sind, ein minimales Tonspektrum

aufweisen und langsam genug sind um mit der Installation spielbar zu sein. Die Entscheidung fiel auf »Smoke on the water« von Deep Purple, »Paint it black« von den Rolling Stones und »One« von Swedish House Mafia.

Die Ausschnitte der Lieder die benutzt werden sollten, wurden zunächst nachgespielt und aufgenommen. Dabei war es wichtig, hervorstechende Klangigenschaften der Originale zu berücksichtigen um den Wiedererkennungswert nicht zu verlieren. Bei »Smoke on the water« wurde beispielsweise ein bekannter britischer Röhrenverstärker mit vorgesetztem Overdrive verwendet. Nachdem alle Einzelspuren aufgenommen waren musste bestimmt werden, welche Spuren den Backing Track bildeten und welche Spur von den Nutzern gespielt werden sollte. Der Backing Track wurde dann ohne die entsprechende Spur gerendert. Die Spur die der Nutzer spielen sollte, musste nun in kleine Samples geschnitten werden um auf die verschiedenen Felder aufgeteilt werden zu können.

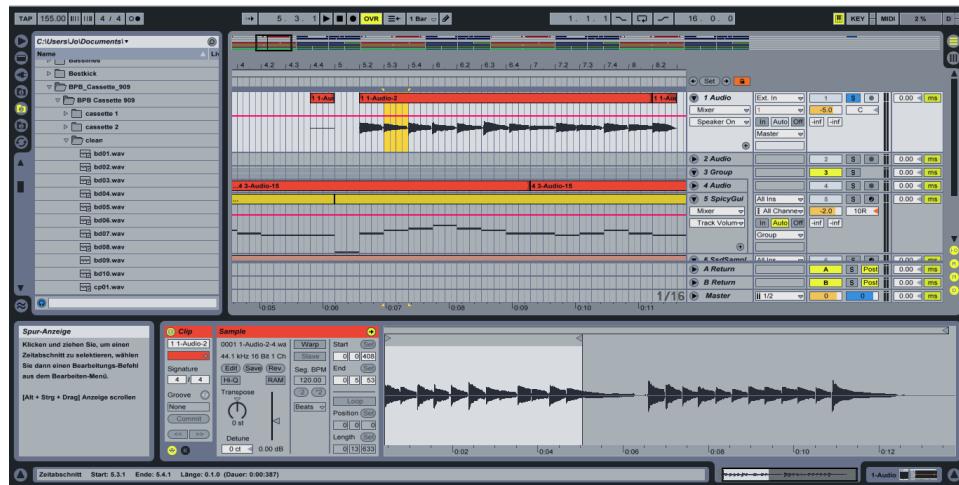


Abbildung 10: Schneiden der Spuren in Ableton Live

Dabei ergaben sich in einigen Fällen Probleme bezüglich der Tonlänge. Da die Schritte des Step-Sequenzers immer auf einen bestimmten Rhythmuswert quantisiert waren, zum Beispiel 8tel oder 16tel-Noten, mussten hinsichtlich Synkopen oft Kompromisse eingegangen werden. Bei »One« führte dies dazu, dass der Impuls so schnell war, dass das Lied damit praktisch unspielbar wurde. Für den Challenge Modus erwies sich damit das Konzept der Soundmatrix insgesamt als relativ ungeeignet, während im experimentellen Modus musikalisch interessante Ergebnisse erreicht wurden.

## **4 Tag der Medien**

**4.1 kleiner Bericht mit Fotos**

**4.2 Praxiserfahrungen**

## 5 Showreel

### 5.1 Aufgaben-Verteilung

#### *Das Team*

Im Folgenden eine kurze Auflistung, welche Aufgabenbereiche von welchem Gruppenmitglied bzw. welchen Gruppenmitgliedern hauptverantwortlich bearbeitet wurden. Einige der genannten Aufgaben wurden von mehreren Personen erledigt oder es wurden kleinere Teile, die hier nicht explizit aufgelistet sind, von anderen Personen beigesteuert. Genaueres dazu kann dem Repository auf GitHub oder den einzelnen Kapiteln dieser Doku entnommen werden.

Aufgabe	Hauptverantwortlich
Idee und Konzeption	Fabian Gärtner, Sarah Häfele, Alexander Scheurer, Linda Schey, Johannes Winter, Meike Zöckler
Tests und Aufbau	Fabian Gärtner, Sarah Häfele, Alexander Scheurer, Linda Schey
Techn. Zeichnungen	Fabian Gärtner, Sarah Häfele
Beamer-Halterung	HFU Werkstatt
Logo und Grafiken	Sandra Beuck
DXM-Scheinwerfer	Linda Schey
Bau »Photonenbündler«	Meike Zöckler, Linda Schey, Fabian Gärtner
Multimonitor-Support	Linda Schey
Personenerkennung	Fabian Gärtner
Spielfeld und Tiles	Alexander Scheurer
Idle-Animation	Sarah Häfele
Spiellogik	Alexander Scheurer
Präsentationsbildschirm	Sarah Häfele
Netzwerk/XML	Sarah Häfele, Linda Schey
Musik	Johannes Winter
Orga Technik	Alexander Scheurer, Fabian Gärtner
Schnitt Teaser	Meike Zöckler, Johannes Winter, Fabian Gärtner

## 6 Fazit

### 6.1 Möglichkeiten zur Weiterentwicklung

*Sarah Häfele*

Rhythmische Geschicklichkeitsspiele sind heutzutage bei PC- wie auch bei Konsolen-Nutzern beliebt. Als Beispiel sollen hier Spiele wie *Guitar Hero*, *Audiosurf* oder das kürzlich erschienene *Crypt of the Necrodancer* genannt werden.