Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Whack-AR-Mole Dokumentation

Fynn Kahlert & Jasper Holz
Marilena Jens & Tuong Vy Nguyen

20. Juli 2019

IT Systeme - Medientechnik
Mobile Systeme - Media Systems
Fynn Kahlert (2107552)
Jasper Holz (2210170)
Marilena Jens (2323097)
Tuong Vy Nguyen (2315758)

Inhaltsverzeichnis

1 Einleitung

Für die kollaborative Lehrveranstaltung bestehend aus ”IT-Systeme” der Medientechniker und ”Mobile Systeme” der Media Systems-Studierenden, war die Zielsetzung, ein gemeinsames Projekt mit Einbezug von DMX und Mikrocontrollern (Arduino) zu erarbeiten. Dies sollte außerdem unter der Verwendung und Integration einer mobilen Anwendung auf einem Endgerät realisiert werden. Im Folgenden werden das Konzept und die Umsetzung des Projektes ”Whack-AR-Mole”, welches im Rahmen der oben genannten Veranstaltung entstand, näher erläutert.

2 Konzept und Zielsetzung

2.1 Idee und Konzept

Angelehnt an das Spiel Whack-A-Mole, oder zu deutsch: Hau den Maulwurf, wird eine Spielfläche aufgebaut, auf dem eine AR-Anwendung, die über ein Tablet läuft, Figuren darstellt. Diese werden vor Spielbeginn auf festen Punkten platziert, sie erscheinen willkürlich in zeitlichen Abständen auf der Spielfläche. Der Spieler muss diese mittels Lichtstrahl eines Movingheads ”einfangen”. Für jede getroffene Figur gibt es einen Punkt. Nach Ablauf der Zeit ist das Spiel beendet.

Im Hinblick auf den Rundgang der Finkenau als Projektausstellung entschieden wir und für eine Anwendung, mit der die Besucher und auch andere Interessierte sich spielerisch mit auseinandersetzen und interagieren können. Die AR-Komponente kam hinzu, weil wir diese als besonderen Anreiz empfanden.

Aufgrund des Spielprinzips sowie dem Einbezug von Augmented Reality entstand auch der Name der Anwendung: ”Whack-AR-Mole”.

2.2 Lastenheft

Die HAW Hamburg beauftragt uns mit der Fertigstellung eines Projektes im Rahmen der Lehrveranstaltung ”IT-Systeme/Mobile Systeme”. Ausgestellt werden soll es bei der Jahresausstellung der Finkenau am 11. Juli 2019.

Die Aufgabe lautet wie folgt: ”Konzipieren und erstellen Sie ein IT-System, das ein DMX Signal zur Steuerung oder Auswertung auf mobilen Geräten verfügbar macht.”

Technische Voraussetzungen:

- Eine mobile App, die mit Kotlin, Swift oder Flutter programmiert wurde.
- Nutzung eines Hardware-Interfaces mit Arduino, RaspberryPi oder Ähnlichem.

Zur Erfüllung der oben genannten Vorgaben wird die Anwendung "Whack-AR-Mole" erstellt.

Ziel

- Eine mobile Anwendung, in der die Steuerung eines Movingheads sowie das Spielen einer AR-Applikation, möglich sind.
- Der gewünschte Fertigstellungstermin ist auf den 25.06.2019 datiert.

Produkteinsatz

- Das Produkt soll über verschiedene Android-Endgeräte laufen können, speziell für den unten genannten Einsatz über ein Tablet.
- Es soll beim Rundgang Finkenau, am 11. Juli 2019, ausgestellt und präsentiert werden.

Produktfunktionen

- Ein Movinghead kann über die App gesteuert werden.
- Mithilfe von Augmented Reality sollen Figuren erstellt werden die mit dem Licht des Scheinwerfers interagieren.
- Ist das Licht des Scheinwerers an der gleichen Position wie eine AR-Figur, verschwindet diese.
- Für jede eingefangene Figur bekommt der Spieler einen Punkt.
- Sind die Figuren eingefangen erscheinen neue.
- Nach Ablauf der Zeit ist das Spiel beendet.

Produktanforderungen

- Die Applikation soll leicht bedienbar sein.
- Spielprinzip und Anwendung sollen simpel und gut verständlich sein.

2.3 Pflichtenheft

2.3.1 Zielbestimmung

Ziel ist es, eine mobile Anwendung zu konzipieren, die ein DMX Signal zur Steuerung oder Auswertung auf mobilen Geräten verfügbar macht. Dabei soll Augmented Reality auf eine spielerische Art und Weise mit einbezogen werden.

Muss-Kriterien

Produktfunktionen für den Benutzer:

- Über eine App wird das Videobild der Tablet-internen Kamera per Augmented Reality erweitert.
- Die Bedienung erfolgt über ein User Interface in der App.
- Beim Starten der Anwendung erscheint eine Seite, über die eine Bluetoothverbindung mit der erforderlichen Schnittstelle erstellt werden kann.
- Nach Auswählen eines Gerätes zum Verbinden wird der Nutzer auf die eigentliche Anwendungsseite weitergeleitet.
- Über die App kann der Benutzer den Movinghead bewegen.
- Als Steuerung dient ein “Joystick” auf dem Bildschirm, der die beiden Achsen des Geräts steuert.
- Auf dem Bildschirm sind ein Score-Counter und ein Timer zu sehen. Außerdem noch ein Play-Button, mit dem das Spiel gestartet werden kann.
- Drückt der Nutzer auf den Play-Knopf, startet der Countdown: Der Timer zählt runter.
- Es erscheinen eine zufällige Anzahl von AR-Objekten auf der festgelegten Spielfläche, die über die App auf dem Bildschirm zu sehen sind.
- Trifft der Nutzer mit dem Lichtwurf des Movingheads eines der AR-Figuren, verschwindet dieses vom Spielfeld. Der Spieler erhält einen Punkt.
- Sind alle Objekte verschwunden, so erscheinen wieder eine zufällige Anzahl neuer AR-Figuren.
- Das Spiel kann so lange gespielt werden, bis der Timer abgelaufen ist.

- Nach Ablauf der Zeit verschwinden alle Objekte auf der Spielfläche. Dem Nutzer wird eine Endnachricht angezeigt sowie sein erreichter Score.
- Mit dem Play-Button kann der Nutzer, auch während der Runde, das Spiel beenden.

Technische Produktanforderungen:

- Um den physischen Rahmen des Spiels zu definieren, werden Maße des Spielfeldes und Position des Movingheads bestimmt und die App kalibriert.
- Über die App kann der maximale Raum, den der Movinghead abfahren kann, und der Startpunkt des Movingheads definiert werden.
- Der Arduino wird zwischen das Lichtstellpult und den Movinghead in das DMX eingebunden.
- Die Steuerung des Movingheads geschieht über eine Bluetooth Verbindung mit der Arduino-Schnittstelle.
- Mit dem Pult kann der Movinghead ebenfalls gesteuert und kalibriert werden.
- In den Eingang kommt die Verbindung zwischen Pult und Arduino. In den Ausgang kommt die Verbindung zwischen Arduino und Moving Head. Der Arduino soll in der Lage sein die Steuersignale vom Pult einfach durchleiten zu können.
- Die Steuerung per App soll über einen DMX Kanal vom Pult aus aktivier- und deaktivierbar sein.
- Die Positionen der AR-Objekte, sowie die Position des Lichts vom Movinghead werden auf die Koordinaten der Spielfläche umgerechnet, um so eine Übereinstimmung feststellen zu können.

Nice-to-Have

- Nach Beendigung des Spiels kann der Nutzer seinen Namen eingeben, der zusammen mit seinem Punktestand in einer Highscore-Liste/"Hall of Fame" eingetragen wird.
- Es gibt mehrere Level und Schwierigkeitsgrade im Spiel.
- Für bestimmte AR-Objekte müssen bestimmte Einstellungen für das Licht vorgenommen werden, damit diese als getroffen gelten (anderes Muster, Farbe usw.).

2.3.2 Produkteinsatz

Anwendungsbereiche

Das Produkt kann, da es sich hierbei um eine spielerische Anwendung handelt, als Zeitvertreib und Spaßfaktor dienen. Speziell aber soll es bei der Jahresausstellung im Juli zum Einsatz kommen. Auch als Anwendung, die einen kleinen Einblick in die Möglichkeiten mit Augmented Reality kombiniert mit Hardware gibt, ist sie geeignet.

Zielgruppen

Genutzt werden kann das Produkt von jeder Person, die sich für die Anwendung interessiert. Besonders geeignet sind dabei Leute, die bereits eine höhere Affinität zu Technik und mobilen Anwendungen besitzen, da dies die Bedienung intuitiver und einfacher macht.

2.3.3 Technische Produktumgebung

Hardware

- 1x Samsung Tablet S3
- 2x Arduino ATMega 2560
- 2x MAX 485 RS485 Modul
- 1x Bluetooth-Modul HC-06
- 1x Movinghead Robe Robin LEDBeam150
- 1x Lichtstellpult Zero88 FLX s24
- 1x Kopflicht

Software

- Entwicklungsumgebung (IDE): Android Studio 3.4
- Programmiersprache: Kotlin
- ARCore SDK 1.8.0
- Sceneform SDK 1.8.0
- Controlwear Virtualjoystick 1.10.1
- Für Grafiken: Adobe Photoshop

Rigging

- 2x Manfrotto Kurbelstativ
- 1x 2m Pipe
- 3x Molton
- 2x Sandsack

3 Umsetzung

3.1 Aufbau

Abbildung 1: Aufbau Rundgang

Das Spielfeld befindet sich auf dem Boden. Mittig und zentriert darüber angebracht ist der Movinghead, der an einer 2m Pipe, gehalten von zwei Kurbelstativen, befestigt ist. Um einen kleineren Lichtstrahl zu erreichen, ist ein Aufsatz an dem Scheinwerfer angebracht,

der den Lichtausfall verkleinert. Rechts im Bild ist das Lichtstellpult zu sehen, mit dem der Movinghead ebenfalls gesteuert, kalibriert und eingestellt werden kann. Die Installation ist mit schwarzem Molton umhangen, um Lichteinfall von außen und somit Reflexionen zu vermeiden, die das Tracking beeinflussen würden. Um ein gleichmäßiges Licht auf dem Spielfeld für ein optimales Tracking zu erreichen, wird ein kleines LED-Kopflicht verwendet, das an der Pipe neben dem Movinghead befestigt ist. Gespiegelt wird das Tablet auf einen Laptop (im Bild nicht zu sehen), sodass auch Besucher, die das Tablet nicht in der Hand halten, sehen, was passiert.

3.2 Verschaltung

Ein Feature des Gerätes soll das Durchschleifen und Manipulieren des DMX-Stromes sein. Da wir keine Arduino-Library gefunden haben, die uns die Möglichkeit bietet gleichzeitig zu Empfangen und zu Senden, fiel die Entscheidung auf eine Kombination aus zwei Arduinos. Der Arduino "Rx" hat die Aufgabe den DMX-Strom aus der Lichtsteuerkonsole zu lesen und an den Arduino "Main" weiterzugeben. "Main" empfängt die Steuerpakete von App und Arduino "RX" und sendet die korrekten Informationen als neues DMX Signal an den LEDBeam Movinghead. Die Verschaltung sieht im Detail aus wie in Abb. 3.2 auf Seite 11 dargestellt.
Die Wahl der Konsole viel auf die Zero88 FLX s24 von Eaton. Hauptgründe waren die Kompakte Bauweise zwecks Flexibilität, Ausreichende Funktionalität und besonders Verfügbarkeit über den langen Projektzeitraum.
Die Auslesung des des DMX-Signals passiert über ein MAX485 Modul, dass den Standard RS485 interpretieren kann, auf dem das DMX-Protokol basiert. Die Arduino Library "DMXSerial" kann über den einen Seriellen Port des Arduinos (in diesem Fall Serial1) die DMX-Daten interpretieren. Über den seriellen Port 2 sendet "RX" die empfangenen DMX-Werte pro Kanal auf Serial2 von "Main".
Auf Serial3 "Main" steckt ein HM-10 Modul zum Empfangen der Bewegungsinformationen der App per Bluetooth. Aufgrund des Inputs der Lichtkonsole bzw. "Rx", entscheidet "Main", welche Steuerinformationen verwendet werden sollen und schreibt diese auf Serial1 Pin TX1. Das MAX485 Modul "Output" überträgt diese als DMX Signal an den Movinghead und etwaige weitere Geräte in der DMX-Linie.

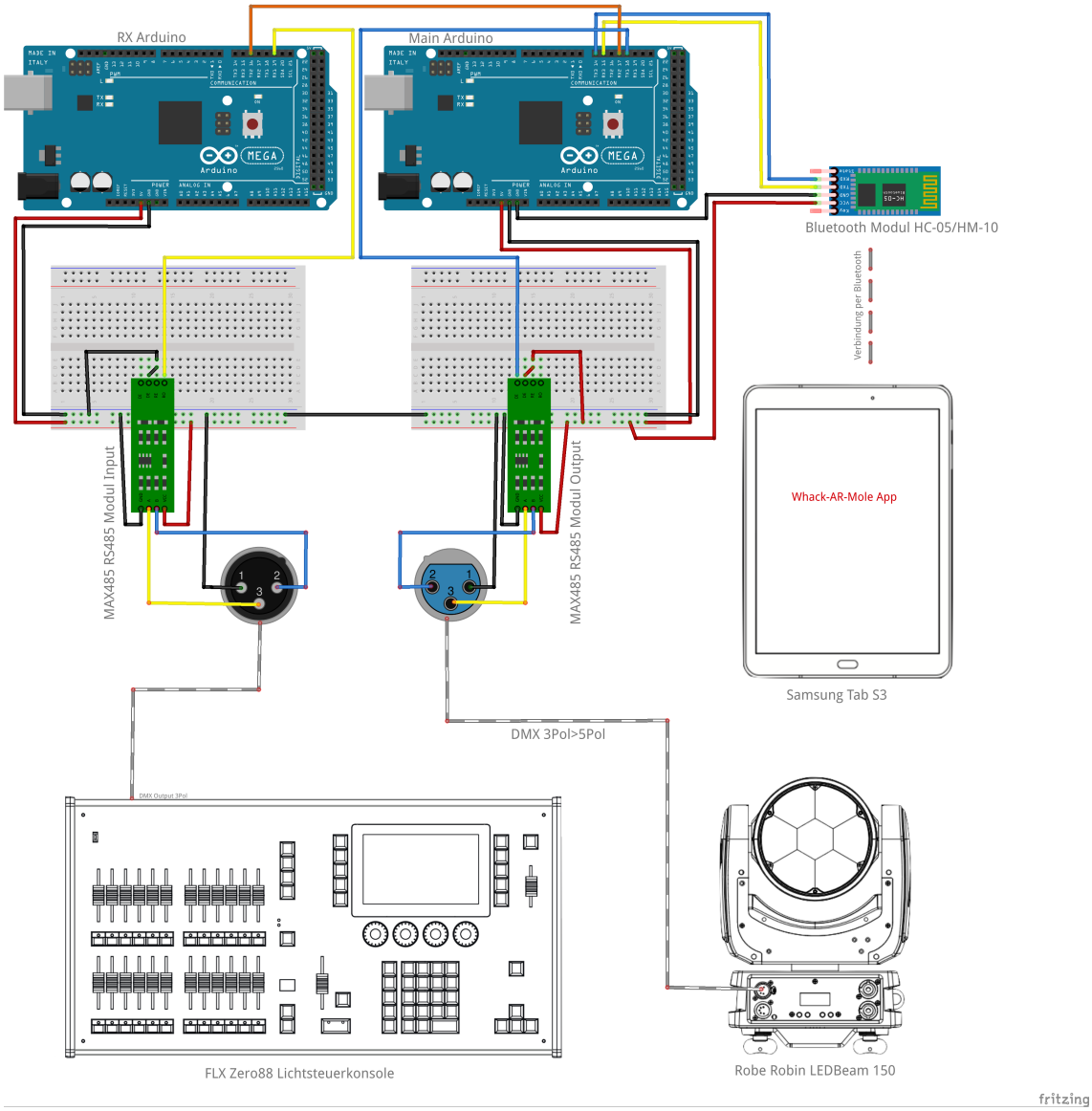

Abbildung 2: Schaltplan, Grafiken:[2][3][1]

3.3 Umsetzung in der App

3.3.1 Appdesign

Da viel Zeit und Energie in die Implementierung der Funktionalität und der Programmierung floss, blieben leider wenig Ressourcen für Verschönerung und App-Design selbst.

Das Design der Anwendung sollte so simpel wie möglich gehalten werden. Oben rechts befindet sich der Timer, oben links die Score-Anzeige. Mittig unten sind der Play-Button sowie der Highscore-Button zu finden. Links daneben befindet sich der Joystick, mit dem der Movinghead gesteuert werden kann. Alles ist außerdem in einheitlichen Farben gehalten.

3.3.2 Funktionalität

Die Steuerung des Movingheads sowie die Berechnung der Positionen und passenden PAN- und TILT-Werten geschieht über die App. (Mehr dazu in den folgenden Abschnitten.) Auch Spiellogik und natürlich die AR-Komponenten sowie die Datenbank für die Highscores sind über die App geregelt. Wie genau alles implementiert wurde wird in Kapitel 4.2 ausführlicher erläutert.

3.3.3 Herausforderungen und Änderungen

Das Tracking des Lichtes war in der App leider nicht so möglich wie wir zu Beginn angedacht hatten. Daher mussten wir uns dort etwas anderes überlegen (Mehr dazu siehe 4.2).

Eine weitere Herausforderung, die damit einher ging, war der AR-Teil. Für die Generierung der AR-Objekte nutzten wir zunächst das Plane Tracking in ARCore. Damit werden horizontale Oberflächen erkannt, die zur Erstellung der Figuren genutzt werden. Dies stellte sich jedoch als Problem heraus, da so keine feste Spielfläche garantiert werden konnte. Für die Berechnung der Positionen der Figuren und auch des Lichtes benötigen wir feste Werte für die Fläche und feste Standpunkte. Nach mehreren Testläufen stellten wir schließlich fest, dass trotz immer selber Rahmenbedingungen und der immer gleichen Einnahme einer bestimmten Position des Nutzers die Spielfläche immer anders war. Mal größer oder kleiner, verschoben, rotiert usw. Daher mussten wir vom Plane Tracking abweichen.

Als Lösung verwendeten wir das Tracken mithlfe eines Bildes. Wir verwenden ein sogenanntes AugmentedImage, das als Marker dient und von uns als Spielfeld genutzt wird. Auf diesem Weg können wir eine immer gleiche Größe der Spielfläche sicherstellen sowie eine feste Position im Raum.

Ein weiteres Hindernis, dass noch auftauchte, war das ungenauere Tracking. Das Spielfeld wurde mithilfe des Bildes zwar erkannt, hatte jedoch eine falsche Rotation oder sogar falsche Größe. Dies behoben wir, indem wir verschiedene Bilder als AugmentedImage ausprobierten und schließlich eins nutzten, dass viel Kontraste und Feature Points besitzt. Die Implementierung der Objekte innerhalb der App passten wir ebenfalls leicht an (Mehr dazu siehe Kapitel 4.2.2).

3.4 Tracking des Lichtkegels

Wie oben bereits erwähnt, war ursprünglich angedacht, das Treffen der AR-Objekte mit dem Spotlight mithilfe der Funktionen der genutzten SDK ARCore zu bestimmen.

ARCore besitzt eine so genannte ”Light Estimation”, die es ermöglicht, die Lichtverhältnisse in einem Raum zu bestimmen. Wenn ein Objekt also getroffen wird, verändert sich dementsprechend das Licht an dieser Stelle und es kann somit erkannt werden, wenn die AR-Figuren abgefahren werden. Leider ist dies in ARCore derzeit jedoch nur für die ganze Szene möglich, also nicht für bestimmte Stellen bzw. bei einzelnen Objekten. Diesen Lösungsansatz mussten wir daher verwerfen.

Da das Vorhaben eines Trackings des Lichtes in der App somit leider nicht möglich war, haben wir ein Workaround erarbeitet. Der Benutzer bewegt ein nicht sichtbares Objekt, um die Spielobjekte zu treffen. Dem Movinghead wird dann die Position in 16Bit DMX-Werten vorgegeben. In Kapitel 3.5 wird erläutert, wie diese Werte berechnet werden.

3.5 Position vs DMX-Werte

Anstatt die Position des Lichtkegels zu erkennen, müssen wir dem Moving-Head vorgeben, auf welche Position gefahren werden muss bzw. welche PAN/TILT-Werte er einnehmen soll. Bevor wir diese berechnen, bestimmen wir zuerst die x,y-Position auf der Spielfläche. Dies tun wir, indem wir den Joystick bzw. den Kreis davon, quasi auf das Feld mappen und die Bewegung des Lichtes dem Joystick imitieren. Von diesem erhalten wir zwei Werte, den Winkel (Wert zwischen 0 und 359) und die Stärke (Wert zwischen 0 und 99). In der unten abgebildeten Grafik erkennt man das Prinzip. Wir führen also eine einfache Berechnung von Polarkoordinaten auf x,y-Koordinaten um.

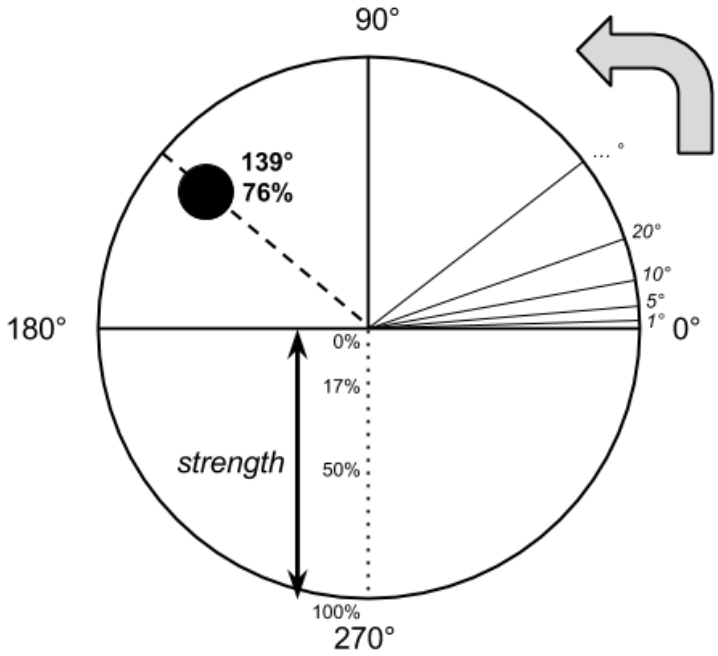

Abbildung 3: Berechnung der Koordinaten

Um die dazugehörigen DMX Werte zu berechnen, betrachten wir das Problem mit der analytischen Geometrie. Wir nehmen an unser Nullobjekt befindet sich an dem Punkt a in einem Koordinatensystem.

$$a = \begin{pmatrix} x & y & z \end{pmatrix}$$

Wenn wir davon ausgehen, dass der der Punkt sich auf der xy-Ebene bewegen kann und z den Abstand des Movingheads zur Spielfläche beschreibt, dann würde ein Vektor $\vec{a}$ einen Winkel α gegenüber der x-Achse aufspannen. Es gilt

$$\sin \alpha = \frac{\vec{a} * \vec{e_x}}{|\vec{a}| \cdot |\vec{e_x}|}$$

mit $\vec{e_x}$ als Einheitsvektor für die x-Achse. Der Winkel $\vec{a}$ ist direkt mit dem Pan-Wert des Movingheads verbunden, der benötigt wird um die gewünschte Position am Punkt a zu treffen. Vereinfacht, eingesetzt und nach α aufgelöst erhält man

$$\alpha = \arcsin\left(\frac{|x|}{\sqrt{x^2 + y^2 + z^2}}\right)$$

und auf die y-Achse angewendet für den Tilt-Winkel β

$$\beta = \arcsin\left(\frac{|y|}{\sqrt{x^2 + y^2 + z^2}}\right)$$

Für die Implementierung haben wir $|x| \rightarrow x$ gesetzt, damit die Formel auch für den negativen Bereich der jeweiligen gilt. Um die DMX-Werte zu bestimmen, haben wir drei Kalibrierungspunkte festgelegt: Die Mitte des Spielfeldes für x_0 und y_0 , der mittlere Punkt an der oberen Spielfeldkante y_{max} und der mittlere Punkt an der rechten Spielfeldkante x_{max}. Diese Punkte repräsentieren die Winkel α_{max} und β_{max} indem sie mit der physikalischen Spielfeldgröße s und Abstand Movinghead z verrechnet werden.

$$\alpha_{max} = \arctan \frac{0.5 \cdot s_{horizontal}}{z}$$

$$\beta_{max} = \arctan \frac{0.5 \cdot s_{vertikal}}{z}$$

Mit einem simplen Dreisatz bestimmen wir die Werte Pan und $Tilt$ zu den berechneten Winkeln α und β:

$$Pan = \left(\frac{x_{max} - x_0}{\alpha_{max}} \cdot \alpha\right) + x_0$$

$$Tilt = \left(\frac{y_{max} - y_0}{\beta_{max}} \cdot \beta\right) + y_0$$

x_0 und y_0 dienen als Offset. Wenn $\alpha = 0; \beta = 0$ fährt der Movinghead auf die 0-Position $y_0; x_0$.
Für die Bestimmung der Übereinstimmung der Position der AR-Objekte mit dem Licht benutzen wir ein weiteres nicht sichtbares AR-Objekt, welches ebenfalls mit dem Joystick gesteuert wird. Das Licht folgt somit quasi der Position dieses Objektes. Stimmen die Koordinaten des Objektes mit denen der anderen überein, gelten diese als getroffen.

4 Code

Im folgenden wird die Umsetzung des Codes im Detail erläutert.

4.1 Arduino

Der Gesamte Arduino-Code ist in sechs Teile aufgeteilt. Davon läuft main_final.ino, LOOP.ino, JSON_BT.ino, JSON_RX.ino, und DMX.ino auf dem Arduino "Main". Auf dem Arduino "RX" wird rx_final.ino verwendet.
Während die Datei für den Arduino "RX" Setup wie Loop enthält, haben wir der übersichtshalber den Setup und Loop für "Main" getrennt.

4.1.1 JSON

Da zwischen diversen Geräten kommuniziert werden muss, ist ein Protokoll notwendig. Die Entscheidung fiel auf das JSON-Protokoll, da dieses ein weltweiter Standard ist und sich in den meisten Programmiersprachen und und IDE's implementieren lässt.
JSON steht für JavaScript Object Notation und ist ein Text-Format für den Datenaustausch. JSON ist besonders gut in C-basierten Programmiersprachen zu verwenden. Grundlegend besteht ein JSON Paket aus Paaren aus Namen und damit verknüpften Werten. Bei einem solchen Paar spricht man von Objekt, das in diesem Fall folgende Struktur hat: {"Name":"Wert"} [10].
Für die Arduino Sprache besteht eine Library, die wir für das Parsen und Generieren verwenden. Wir haben uns dafür an den Beispielen auf der arduinojson.org Website orientiert [11]. In diesem Projekt wird das JSON-Protokoll zwei mal Anwendung finden: Für die Bluetooth Übertragung und die Kommunikation zwischen RX_Arduino und Main_Arduino

4.1.2 Bluetooth

Das Bluetooth Modul muss zunächst auf die korrekte BAUD-Rate, Name und Pin eingestellt werden. Wie nach Vorlesungsskript empfohlen haben wir dazu das Programm von Herrn Prof. Edeler verwendet und das Modul auf die Baudrate 115200 einzustellen [13]. Mithilfe der Library BTSerial können wir einen Serial Port des Arduino als Bluetooth Port definieren. Mit der Funktion lässt sich im weiteren genauso arbeiten, wie mit der Standard Serial-Funktion der Arduino-Sprache [12]. Aus den im Header definierten seriellen Port können damit Zeichenweise die vom HC06 aus gesendeten Strings ausgelesen werden.
In JSON_BT sind dafür folgende Funktionen beschrieben:

- readJsonBT() liest sobald ein Zeichen am dedizierten BT-SerialPort erkannt wird dieses und alle folgenden Zeichen per BTSerial.read() aus und füllt zeichenweise 'json[ ]'. Wenn ein Zeilenende eingelesen wird, wird 'readJson = true' gesetzt. Damit wird für folgende Funktionen der Buffer 'json[ ]' für die weiteren Schritte als freigegen markiert.

Listing 1: readJsonBT() (JSON_BT.ino)

```
12    void readJsonBT() {
13    if (BTSerial.available()) {
14        // Zeichenweise auslese des BT Streams
15        char c = BTSerial.read();
16        // Json Buffer bei Zeilenende zum verarbeiten
              flaggen
17        readJson = c == '\n';
18
19        json[jsonCounter] = c;
20        jsonCounter++;
21        }
22    }
```

- freeJson() leert den BT-Buffer 'json[ ]' und setzt ihn damit zurück, um von readJsonBT() neu befüllt werden zu können. Außerdem wird 'readJson = false' zurückgesetzt.

Listing 2: freeJson() (JSON_BT.ino)

```
12    // JSONBT Buffer leeren
13    void freeJson() {
14        for (int i = 0; i < jsonCounter; i++) {
```

```
15              json[i] = 0;
16              }
17          jsonCounter = 0;
18          readJson = false;
19      }
```

- parseJson(*doc,buff[ ]) führt primär die Funktion 'deserializeJson()' aus der Library ArduinoJson aus. Die einzelnen Objekte des angegeben Buffers 'buff[ ]' werden erkannt und aus "auseinandergezogen" und in Form von JsonObjekten in das angegebene JsonDocument 'doc' geschrieben.
  Zusätzlich lässt sich per DeserializationError festellen, ob die Aktion erfolgreich war. Sollte 'error=false' aufweisen, wird eine Fehlernachricht mit der erfassten Fehlerbezeichnung per Funktion 'error.c_str()' ausgeworfen.

Listing 3: parseJson() (JSON_BT.ino)

```
12      bool parseJson(JsonDocument *doc, char buff[]) {
13          // deserialize
14          DeserializationError error = deserializeJson(*doc,
                buff);
15
16          // Validierung des Strings, der im JSON_BT Buffer
                haengt
17          if (error) {
18              Serial.print(F(''deserializeJsonBT() failed: ''
                    ));
19              Serial.println(error.c_str());
20              freeJson();
21              return false;
22          }
23          else {
24              return true;
25          }
26      }
```

- handleJsonBT() beschreibt die Vorgänge, die mit den Informationen aus dem angegeben JsonDocument ausgeführt werden sollen. Den Variablen valPanBT16 und valTiltBT16 sollen die Werte aus den JsonObjekten mit den Namen 'Pan' und 'Tilt' zugeordnet werden.

Die Bewegungsauflösung des Movingheads wäre mit 8 Bit für einen DMX-Kanal zu gering für unsere Anwendung, um präzise die Objekte zu treffen. Durch die Verwendung eines zweiten Channels pro Achse 'finePan' bzw. 'fineTilt' können die meisten Movingheads die Bewegungsauflösung auf 16Bit erhöhen. Der 'Coarse'-Channel Pan entspricht in dem 16Bit den most significant Byte, während der 'Fine'-Channel finePan dem least significant Byte zugeordnet wird. Listing 4 zeigt diese Zuordnung In der App kann also mit der höheren Auflösung gearbeitet werden.

Listing 4: 16Bit in 2x 8Bit aufteilen (JSON_BT.ino)

```
46    // 16Bit P/T Werte in coarse und fine 8Bit-Werte
         aufteilen
47    valPanBT = highByte(valPanBT16);
48    valfinePanBT = lowByte(valPanBT16);
49    valTiltBT = highByte(valTilBT16);
50    valfineTiltBT = lowByte(valTilBT16);
```

4.1.3 DMX

- Die Library DMXSerial läuft im Default über den Seriellen Port 0. Da über diesen Port auch der serielle Monitor der Arduino IDE geroutet ist, hat man in der Library-File DMXSerial.cpp die Möglichkeit #define DMX_USE_PORT1 im Setup hinzuzufügen. Dadurch wir von DMXSerial auch Port 1 verwendet.

- Für jeden DMX Kanal des Movingheads ist eine Kanal-Variable und eine Wert-Variable in der Form angelegt, wie in Listing 5 und 6 beispielhaft erkennbar ist.

Listing 5: Value Variablen (rx_final.ino)

```
11 byte valPan;
12 byte valTilt;
```

Listing 6: Channel Variablen (rx_final.ino)

```
24 byte chPan = 1;
25 byte chTilt = 3;
```

Diesen Value-Variablen werden im loop() mit Werten aus dem eintreffenden DMX-Strom gefüllt:

Listing 7: DMX lesen mit DMXSerial.read (rx_final.ino)

```
52 void loop() {
```

```
53
54    valDim = DMXSerial.read(chDim);
55    valPan = DMXSerial.read(chPan);
56    valTilt = DMXSerial.read(chTilt);
```

Um die gelesenen Werte zum Arduino ”Main” zu übertragen werden die Variablen eins zu eins in JSON-Objekte ”verpackt”. Dafür wird ein JSON Document ”doc” angelegt. Die Länge dieses statischen Dokuments ist für diese kleinen Datenmengen und Programmgröße nicht relevant. 50 Zeichen erschien als eine sinnvolle Größe, um die Ziele des Programms zu erreichen.

Listing 8: JSON Objekte erstellen (rx_final.ino)

```
69    //JSON Objekte erstellen
70    StaticJsonDocument<200> doc;
71
72    doc[''valDim''] = valDim;
73    doc[''valPan''] = valPan;
74    doc[''valTilt''] = valTilt;
```

Alle erstellten Objekte werden seriell an einnander ”gehängt” und direkt in den seriellen Port 2 des Arduinos als JSON-String geschrieben.

Listing 9: JSON schreiben (rx_final.ino)

```
88      serializeJson(doc, Serial2);
```

- Da bei der vollen Leserate das Schreiben des JSON-Strings viele Fehler aufweist, sind wir gezwungen gewesen ein kleines Delay in Zeile 94 einzufügen.

- Auf dem Arduino ”main” funktioniert das Parsen des eintreffenden JSON-Strings genau so wie in Kapitel 4.1.2 bereits für die Bluetoothverbindung erläutert. Um Aktionen erfolgreich nach einander im Loop ausführen zu können, sind alle Variablen, Buffer und JSON Dokumente umbenannt. Die Bezeichnungen sind jeweils mit einem ”RX” erweitert worden, während alle Variablen aus JSON_BT.ino ein ”BT” aufweisen. Die Funktionen, die in LOOP.ino aufgerufen werden sind in JSON_RX.ino beschrieben. Der Unterschied liegt in der Funktion handleJSONRX(), die im Gegensatz zu handleJSON(), DMX-Werte schreibt:

Listing 10: DMX-Werte schreiben (JSON_RX.ino)

```
68      DMXSerial.write(chPan, valPanRX);
69      DMXSerial.write(chTilt, valTiltRX);
```

- Da die App nur Pan und Tilt-Werte vorgibt, werden in handleJSONRX() mehr Objekte Variablen zugeordnet und über die Unterfunktion writeDMX() aus DMX.ino direkt als neue DMX-Information gesendet. In writeDMX() werden alle Parameter als DMX geschrieben, die nicht Daten sind, die, wie in Kapitel 4.1.4 beschrieben, für der Bewegung des Movingheads relevant sind.

4.1.4 Quelle der Steuerung

- Für die Möglichkeit per DMX die Quelle der Steuerung zu bestimmen, ist sind die Variablen "chCtrl" und "varCtrl" vorhanden. Sie beschreiben die DMX-Adresse des Gerätes, denn mit ihr kann die aktive Funktion über das Pult gesteuert werden. Im Pult ist dafür ein einfacher generischer Kanal angelegt, der von Arduino "RX" normal ausgelesen wird und weitergereicht wird:

Listing 11: Kontrollvariablen (rx_final.ino)

```
6     //Kontrollvariable
7     byte chCrtl = 40;
8     byte valCrtl;
9     [...]
10    valCrtl = DMXSerial.read(chCrtl);
11    [...]
12    doc[''valCtrl''] = valCrtl;
```

- Auf Arduino "Main" wird in der Funktion handleJsonRX() über eine einfache if-Abfrage und den Wert der Variable valCtrl entschieden, ob die P/T-Werte RX (valCtrl>0) oder BT (valCtrl=0) als DMX geschrieben werden sollen.

4.1.5 Loop

In der Datei 'LOOP.ino' wird die Hauptprogrammschleife ausgeführt. fktreadJsonRX() liest am seriellen Port 2 eintreffende Strings in jsonRX[ ] ein. Sobald readJsonRX=true wird ein JsonDocument docRX erstellt und die Funktion parseJsonRX() auf jsonRX angewendet. Wenn diese als erfolgreich markiert wurde (bool successRX), wird 'docRX' durch handleJsonRX() geschoben. Im letzten Schritt wird der RX-Buffer geleert (freeJsonRX()). Genau das gleich wird im folgenden auch für die Bluetooth-Operation mit den entsprechenden Funktionen und Variablen ausgeführt.

4.2 App

Die App ist in unterschiedliche Komponenten und Klassen aufgeteilt, die dann größtenteils alle in der MainActivity zusammenkommen. Im Folgenden sind diese aufgeführt und deren Funktionen näher erläutert:

4.2.1 Bluetoothverbindung

Für die Steuerung des Movingheads von der App aus verwenden wir die Schnittstelle zum Arduino via Blutoothmodul. Damit das Gerät über die Anwendung eine solche Verbindung herstellen kann, müssen in der AndroidManigest.xml entsprechende Permissions gesetzt sein:

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

Zur Implementierung haben wir uns dabei an diverse Musterbeispiele orientiert, besonders haben wir uns dabei auf den zur Verfügung gestellten Quellcode von AppsInTheSky[5] gestützt.

- Startet man die Anwendung landet man zunächst auf der ”BluetoothActivity”, wo eins der gekoppelten Geräte aus einer Listenansicht ausgewählt werden kann, mit dem sich verbunden werden soll.
- Die Klasse ”BluetoothConnection” dient dazu, die Verbindung zwischen Socket und Device dann auch tatsächlich herzustellen und aufrechtzuerhalten.

4.2.2 Augmented Reality

ARCore ist eine Augmented Reality-Programmierschnittstelle für Android-Geräte. Sceneform ermöglicht es, unkompliziert 3D-Szenen in AR zu rendern, ohne, sich mit OpenGL auseinandergesetzt haben zu müssen. Ähnlich wie bei der Bluetoothverbindung muss für die Benutzung der Kamera eine Permission dafür in der Manifest-Datei gesetzt sein. Das Gerät muss außerdem eine geeignete ARCore-Version installiert haben, um die Anwendung nutzen zu können. Ein Requirements sind dafür ebenfalls in der AndroidManifest.xml gesetzt:

```
<uses-feature android:name="android.hardware.camera.ar"/>
// ...
<meta-data
    android:name="com.google.ar.core"
    android:value="required"/>
```

Bevor auf die Implementierung eingegangen wird, eine kurze Übersicht über einige AR-Begriffe[14]:

- Trackable: Ein Trackable ist ein Objekt, dass ARCore tracken kann und auf dem Anchor angebracht werden können. Ein Trackable kann eine plane sein (horizontale Flächen, die unter anderem anhand von Feature Points von der Kamera entdeckt werden), AugmentedImages (Bilder, die als Marker für AR dienen), usw.
  Ursprünglich nutzten wir für das Projekt das plane tracking, dass wir jedoch aufgrund der Pronleme, die in 3.3.3 erläutert wurden, verwerfen mussten. Wir nutzen für das Tracking nun ein Bild, welches unten zu sehen ist. Im Projekt wird es als AugmentedImage und als Spieleld genutzt. Erstellt wurde es mithilfe von Adobe Photoshop, es setzt sich aus Stock-Images und anderen Opensource-Bildern zusammen.

Abbildung 4: Das Spielfeld bzw. AugmentedImage

  Für Bilder gibt es in ARCore außerdem ein sogenanntes arcoreimgtool, mit dem getestet werden kann, wie gut sich ein Bild als AugmentedImage eignet. Es wird ein Score von 0 bis 100 angegeben, eine Datei sollte dabei mindestens einen Score von 75 erzielen.

- Anchor: Damit virtuelle Objekte im Raum plaziert werden können wird ein Anchor benötigt. Dieser beschreibt einen fixen Ort und fixe Orientierung in der echten Welt.

- Pose: Eine Pose beschreibt die Transformation eines Objektes von lokalen Koordinaten zu Weltkoordinaten (Translation und Quaternion).

- Node: Eine Node repräsentiert eine Transformation innerhalb der graphischen Szene. Sie kann ein renderable enthalten, dass von der rendering engine gerendert werden kann. Eine Subklasse von Node, die im Projekt verwendet wird, ist AnchorNode - Eine Node, die anhand des Anchors automatisch im world space positioniert wird.

Der Augmented-Reality-Teil stellt programmiertechnisch den größten und aufwendigsten Teil des Projektes dar. Um hier Ordnung und Struktur zu behalten, haben wir die verschiedenen Komponenten in eigene Klassen und Objekte eingeteilt:

- CustomArFragment Die einfachste Art und Weise mit Sceneform eine Szene zu erstellen, ist, ein ArFragment zu benutzen. Das "CustomArFragment" erweitert die Klasse ArFragment. Hier wird die getSessionConfiguration()-Methode überschrieben, um beispielsweise zusätzlich die Konfigurierung der Image-Datenbank für das AugmentedImage zu setzen (Mehr dazu siehe unten).

- Auf dem Spielfeld werden Eckpunkte in Form von vier AR-Objekten angezeigt, die sich in Aussehen und Größe von den restlichen unterscheiden. Sie dienen lediglich zur optischen Abgrenzung des Spielfeldes. Die Klasse "ARCornerNode" erweitert AnchorNode. Der ARCornerNode werden vier Nodes zugeordnet, die vier Eckpunkte des Spielfeldes, die jeweils ein Renderable enthalten.

- "ARObjectsNode" erweitert ebenfalls die Klasse AnchorNode, hier kommt der "RandomNumberGenerator" zum Einsatz. Er liefert eine zufällige Zahl zwischen eins und fünf, dem ARObjectsNode werden dann diese Anzahl von Nodes zugeordnet, die zufällig über die Spielfläche verteilt sind.

- In "ARValues" sind die Felder/Variablen enthalten, auf die von überall aus zugegriffen werden kann. Beispielsweise das getrackte AugmentedImage, die Werte der CenterPose, die AnchorNodes usw.

- Im "ARGameComponent" kommen die verschiedenen Teile der AR-Komponente zusammen. Hier werden die Anchor gesetzt, erstellt oder zerstört und das Objekt, dass zur Imitation des Lichtes des Movingheads dient, bewegt, abhängig von den Werten, die vom Joystick kommen.

- Die 3D-Modelle, die wir für das Projekt nutzen, sind unterschiedlichen Opensource-Seiten entnommen:
  - Die einzusammelnden AR-Objekte[9] (die Digdas),
  - das nicht sichtbare Objekt, das als Licht des Movingheads dient[4] (was zu Testzwecken sichtbar war),

  - und die Eckpunkte des Spielfeldes, dessen 3D-Model aus den sample Projekten der Sceneform SDK entnommen ist[8].

- Die "ModelArInitializerKlasse erstellt die verschiedenen ModelRenderables aus den gegebenen 3D-Modellen, die dann den Nodes entsprechend hinzugefügt werden, um gerendert zu werden.

- Für das Erstellen und Einbinden der Imagedatenbank haben wir uns an den Sample-Code vom AugmentedImage-Projekt der Sceneform SDK orientiert[7]. Das Setup und Laden der Imagedatenbank geschieht in der MainActivity. Das CustomArFragment besitzt einen onUpdateListener, der bei jedem Frame trackt und das Image detected. Wenn der TrackingState == TrackingState.TRACKING ist, wird das Image in "ARValues" gesetzt und das Spiel kann gestartet werden.

.

4.2.3 Spiellogik

Für die Spiellogik selbst kommen vier verschiedene Klassen zum Einsatz.

- Im "GameController"-Objekt sind die boolean-Werte gespeichert, die den Spielverlauf bestimmen, beispielsweise ob das Spiel bereits begonnen hat, schon beendet ist usw. Auf diese Werte greifen die anderen Klassen zu, um zu bestimmen, ob etwas geschehen soll.

- Bei "GameTimer" handelt es sich um einen CountdownTimer, der auf 30 Sekunden gesetzt ist. Damit wird die zeitliche Begrenzung des Spiels festgelegt. Die Timer-Anzeige auf dem Bildschirm wird dabei immer aktualisiert.

- Die Klasse "GameLogic" ist eine Komponente für den Play-Button. Darin befinden sich die Methoden startGame() und endGame(), die bei Betätigung des Buttons ausgeführt werden. Bei ersterer werden die Werte aus "GameController" gesetzt, die Methoden der ArGameComponent werden aufgerufen und der Timer wird gestartet. Bei endGame() wird alles zurück in die Ausgangslage versetzt.

- Der "GameCoordinatesCalculator" ist dazu da, die x,y-Koordinaten des Lichtes auf der Spielfläche zu berechnen und in entsprechende DMX-Werte umzuwandeln. Dieser Schritt ist in Kapitel 3.5 genauer erläutert. Außerdem werden mit der Klasse die Positionen der AR-Objekte mit den Koordinaten des nicht sichtbaren Movinghead-AR-Objektes verglichen. Weichen die jeweiligen x- und y-Werte nicht mehr als 0.06f voneinander ab, gilt die Figur als getroffen. Damit der Nutzer Feedback erhält, wenn

dies der Fall ist, vibriert das Tablet kurz. Dafür muss in der AndroidManifest.xml ebenfalls eine Permission für gestzt sein.

4.2.4 Steuerung des Moving Heads

- Für die Implmentierung des Joysticks verwenden wir die virtual-joystick-android Library[6].
- Wie oben bereits erwähnt, werden in der "GameCoordinatesCalculator"-Klasse die Koordinaten des Lichtes auf der Spielfläche berechnet. Diese werden mit dem "JsonObjectCreator" in ein Json-Objekt umgewandelt, was dann in der MainActivity über den Bluetoothsocket an den Arduino übergeben wird.
- Im "GameFieldValues"-Objekt werden die Werte bei der Kalibrierung dem Spielfeld angepasst. Die PAN- und TILT-Werte der Ausgansposition, der maximalen Position und die Maße des Spielfeldes sind hier zu finden.

4.2.5 Highscore Datenbank

Eines der Nice-to-Have-Kriterien, die ebenfalls implementiert ist, ist die Highscore Datenbank.

- In der HighscoreActivity, die man über einen Highscore-Button auf der Hauptseite erreicht, sind die Top 5 Spielergebnisse zu sehen.
- In der Klasse "GameScoreDatabaseHelper" sind die nötigen Methoden für die Erstellung einer Datenbank zu finden, is extended die Klasse "SQLiteOpenHelper". Die Highscore Datenbank besitzt drei Spalten: Den Key, einen Namen und den Score. Sortiert ist er nach abnehmendem Score-Wert. Ausgelesen werden in der HighscoreActivity dann die ersten fünf Einträge.
- Mithilfe der "DatabaseComponent" wird überprüft, ob der Spieler nach einem Spiel einen dieser fünf Punktestände erreicht bzw. geschlagen hat.
- Hat es der Benutzer geschafft, einen dieser Score-Werte zu schlagen, öffnet sich eine Dialogbox, implementiert über einen "DialogBoxHelper", in dem der erreichte Spielstand angezeigt ist. Hier kann der Nutzer seinen Namen eintragen.
- In der Klasse "Player" befinden sich die beiden Felder "name" und "score". Über dieses Model werden die Werte dann ausgelesen und in die Datenbank eingetragen.

4.2.6 Finale Struktur

Im Folgenden eine Übersicht über die finale Struktur der mobilen Anwendung:

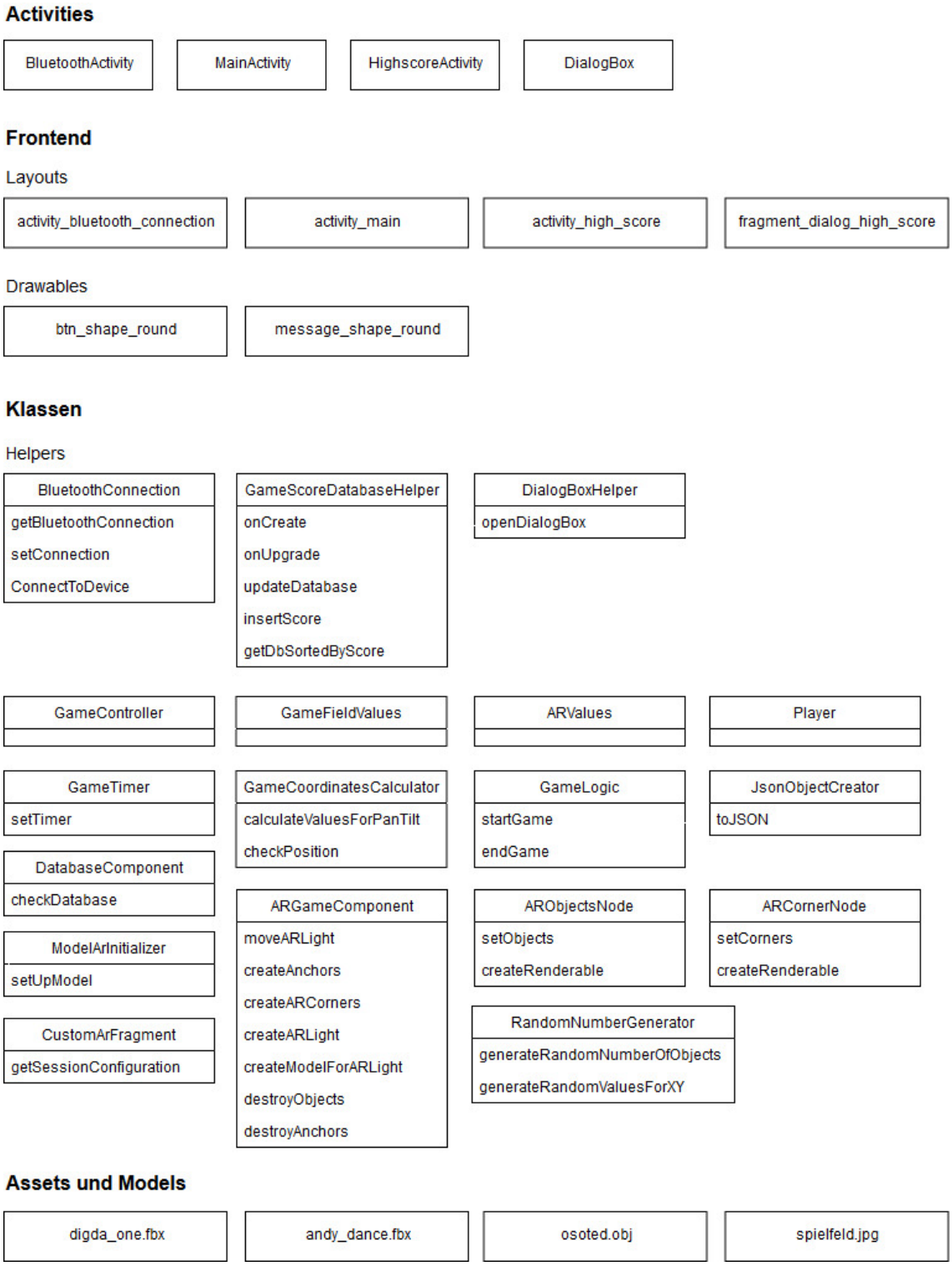

Abbildung 5: Finale Struktur der App

5 Zielsetzung und Optimierungen

5.1 Ausblick Arduino

Der nächste Schritt für eine mögliche Optimierung im Bereich der Schnittstelle DMX durch Arduinos könnte es sein, den Hardware Aufwand auf einen statt zwei Arduinos zu reduzieren. Dafür wäre ein Ansatz die Library DMXSerial noch einmal im Detail zu untersuchen. Gegen Ende des Projektes gab es Hinweise, dass es eventuell doch möglich sein könnte, die Library für das Empfangen und Senden gleichzeitig zu benutzen. Wenn das nicht der Fall ist, könnte man versuchen die Library zu manipulieren und um zuschreiben. Ein weiterer interessanter Schritt könnte es sein, die Funktionalität nicht nur auf die wenigen DMX-Kanäle des Movingheads zu beschränken. Eine Erweiterung auf ein volles DMX-Universum wäre eine spannende Aufgabe. Das Gerät könnte im Theaterbereich als eine Followspot verwendbar sein, um in einem Licht-Cue eine externe Steuerung eines Movingheads zu ermöglichen um z.B. einen Schauspieler auf der Bühne zu verfolgen.
Im Code ist sehr viel Hardcode vorhanden. Eine Verbesserung könnte darauf abzielen Schleifen oder Arrays zu verwenden, um den Code übersichtlicher und flexibler zu gestalten.

5.2 Ausblick App

Was zum einen an der App optimiert werden könnte, was aufgrund mangelnder Zeit leider nicht mehr möglich war, ist das Tracking mit dem AugmentedImage. Bis zum Schluss gab es damit einige Schwierigkeiten: Das Tracking läuft instabil und die Größe des Spielfeldes ändert sich manchmal noch immer. Nach einiger Recherche kamen wir zu dem Ergebnis das dies ein Bug ist, der bereits bei mehreren Personen aufgetreten ist. Die Kamera des Gerätes selbst besitzt einen Tracking State, der sich zu oft aktualisiert und somit das instabile Tracking hervorruft. App-seitig kann dies leider nicht behoben werden. Mit mehr Zeit und Recherche könnte man eventuell einen anderen Löungsansatz finden, um den AR-Teil zu implementieren.

Ein weiterer Punkt, der mit mehr Zeit noch hätte verbessert werden können, ist das Aussehen und Design der Anwendung. Dies kam während der Projektphase leider zu kurz und blieb bis zuletzt in der Prototypphase.

Auch die Platzierung des Joysticks war ein wenig unglücklich gewählt, was wir beim Rundgang bemerkten: Der Joystick befindet sich links unten auf dem Bildschirm. Wenn ein Benutzer das Tablet in die Hand nimmt und diese dabei intuitiv so plaziert, dass er gut an den Joystick herankommt, verdeckt er dabei die Kamera, was die Anwendung funktionsuntüchtig macht.

6 Auswertung

Das Endergebnis ist eine funktionierende mobile Anwendung, über die ein Movinghead mittels Bluetooth-Verbindung gesteuert werden kann. Auf einem Spielfeld können AR-Objekte erzeugt werden, die scheinbar mit dem geworfenen Licht des Movingheads interagieren und im Rahmen eines Spiels eingefangen werden.
Während der Entwicklungsphase mussten viele Dinge umgeändert und neu überdacht werden, jedoch konnten wir für alle Probleme eine Lösung finden, um Augmented Reality als Kern der Anwendung beibehalten zu können, ohne das Spielprinzip und den kompletten Aufbau umändern zu müssen.
Es konnten am Ende sowohl alle Muss-Kriterien als auch ein Nice-to-Have umgesetzt werden: Die App besitzt neben den geforderten Funktionalitäten noch eine Highscore-Datenbank. Außerdem erhält der Spieler Feedback in Form einer kurzen Vibration, wenn in AR-Objekt getroffen ist. Der Movinghead kann über die Arduino-Schnittstelle und dem Bluetooth-Modul angesprochen werden. Außerdem ist die Steuerung durch die Lichtsteuerkonsole möglich und kann durch einen Kontrollkanal per DMX aktiviert werden. Über diese Funktion kann ebenfalls die Kalibrierung der App vorgenommen werden.

```
91   //JSON verpacken und Serialisieren
92     %serializeJson(doc, Serial2);
```

Literatur

[1] http://downloadcenter.samsung.com/content/UM/201805/20180515153346560/SM-T825_UM_Open_Oreo_Ger_Rev.1.0_180509.pdf

[2] https://de.wikipedia.org/wiki/XLR

[3] https://zero88.com/de/datasheets/FLX_Datenblatt_DE_Rev5_0418.pdf

[4] https://free3d.com/de/3d-model/oso-bear-756009.html

[5] https://github.com/appsinthesky/Kotlin-Bluetooth

[6] https://github.com/controlwear/virtual-joystick-android

[7] https://github.com/google-ar/sceneform-android-sdk/tree/master/samples/augmentedimage

[8] https://github.com/google-ar/sceneform-android-sdk/tree/master/samples/hellosceneform/app/sampledata/models

[9] https://sketchfab.com/

[10] https://json.org/

[11] https://arduinojson.org/v6/example/

[12] https://github.com/arduino/BtSerial

[13] https://github.com/tedeler/SS19_ITS/tree/master/BT_finder

[14] https://blog.novoda.com/getting-started-with-google-arcore-on-a

Abbildungsverzeichnis

Tabellenverzeichnis

Listings