TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät

Professur

Praktikumsbericht

MobileMixedReality Report

David Eilers, Oskar Hippe

Chemnitz, den 5. Januar 2022

Prüfer: Tom Uhlmann
Betreuer: Tom Uhlmann

David Eilers, Oskar Hippe,
MobileMixedReality Report
Praktikumsbericht, Fakultät
Technische Universitt Chemnitz, Januar 2022

Zusammenfassung

In diesem Praktikum war es unsere Aufgabe eine augmented reality App zu entwickeln, in der zwei Spieler auf zwei Geräten im lokalen Netzwerk gegeneinander Vier gewinnt spielen können. Dabei sind wir bei der Wahl der AR-Umgebung auf mehrere Sackgassen gestoßen, haben aber mit AR-Core eine ausreichende Lösung gefunden. Die App wurde hauptsächlich in C++ geschrieben. Um dies auf Android zu ermöglichen nutzten wir das Android NDK (native development kit). Damit ist es möglich den für Android üblichen Jave-Code über das Java Native Interface (JNI) mit dem C++ Code kommunizieren zu lassen. Nicht nur der grafische Teil der App sondern auch die Netzwerkfunktionen wurden mit C++ umgesetzt. Lediglich die Activities und Menüelemente wurden in Java implementiert. Beim Testen der App stießen wir auf mehrere Probleme. Zum einen sind Smartphones noch nicht flächendeckend AR-Core-fähig. Ältere Geräte sind dadurch für Tests ausgeschlossen. Zum anderen ist der Umgang mit dem Android-Emulator noch schwierig und teilweise garnicht möglich, wenn der PC keine Virtualisierung unterstützt. Zum Testen der Netzwerkfunktionen kam deshalb ein Python-Skript zum Einsatz. Die Entwicklung der Netzwerkschnittstelle war insgesamt gesehen aber unkompliziert im Vergleich zum grafischen Teil. Die App war am Ende des Praktikums Funktionsfähig und kann auf AR-Core-fähigen Smartphones ausgeführt werden. Für die Zukunft wäre es jedoch Vorteilhafter die Anwendung entweder vollständig in Java oder vollständig in C++ zu entwickeln. Das JNI ist kompliziert und langsam und dadurch hätte man auch auf eine native avtiviy setzen können. Auf diese Weise wäre der eigentliche Code der App vollstädig in C++ gewesen. Die App ist aber auch problemlos in Java umsetzbar und im Context von Android ist Java als Programmiersprache die unkompliziertere Wahl.

Inhaltsverzeichnis

1 Einführung

1.1 Zusammenfassung

Dies ist der Praktikumsbericht zu unserem Teampraktikum. In diesem haben wir eine AR App entwickelt, in der man das Spiel ´Vier gewinnt´ spielen kann. Zwei Spieler können über eine direkte Netzwerkverbindung gegeneinander spielen. Das Spielfeld wird dabei in augmented reality dargestellt. Es kann also mit Hilfe der Kamera des Smartphones von allen Seiten betrachtet und damit interagiert werden.

1.2 Recherche

Am Anfang war nicht klar, ob wir ein VR oder AR Projekt machen. Deswegen haben wir als erstes nach einem Framework gesucht, welches erlaubt VR oder AR Apps zu bauen.
Wir haben mehrere potentielle Lösungen gefunden:

- `C#` Xamarin [Xam]
- WebVR in Webassembly [Webb] [Weba]
- WebXR in Webassembly [Webc] [Weba]
- OpenXR in Webassembly [Ope16]
- ArCore in Android NDK [ARCa]

1.2.1 Xamarin

Wir haben Xamarin[Xam] getestet und hat unter Linux nicht kompiliert. Auch war das AR Framework sehr klein und es wurde längere Zeit nicht upgedatet.

1.2.2 WebVr

WebVR[Webb] hat funktioniert, aber die API gilt als deprecated. Außerdem gab es nur ein nicht funktionierendes Beispiel unter Webassembly.

1.2.3 WebXR

WebXR[Webc] ist der geistige Nachfolger von WebVR und integriert neben VR auch AR support. WebXR war sehr neu und wird bis heute nur in Chromium basierten Browsern supported. In Webassembly gab es keine WebXR API.

1.2.4 OpenXR

Für OpenXR[Ope16] gab es keinen Webassembly support, WebXR als Webalternative für OpenXR gedacht.

1.2.5 ArCore

Das Testprojekt in ARCore[ARCb] wurde erfolgreich kompiliert. Da wir in C++ programmieren sollten, mussten wir auf Android NDK[Andc] zugreifen. Unser Betreuer hatte schlechte Erfahrungen mit Android NDK gemacht und uns dementsprechend davon abgeraten.

1.2.6 OpenCV

Im Verlauf des Projekts entstanden einige Schwierigkeiten und es wurde darüber nachgedacht unter Android NDK[Andc] auf OpenCV[Ope] zurückzugreifen, wir haben uns dabei nur sehr oberflächlich mit Anchor Erkennung mit OpenCV auseinander gesetzt, aber nach grober Recherche, wäre der Aufwand OpenCV zu benutzten, wahrscheinlich größer gewesen als in ArCore.

1.2.7 Entscheidung

Wir haben uns nach der Recherche für ArCore[ARCa] entschieden, da es aktiv von Google entwickelt wird, kompiliert werden konnte und auch die Demo funktionierte.

2 Entwicklung

2.1 Anmerkung

Die App wurde mit NDK entwickelt. NDK steht für Native Development Kit und erlaubt C++ und C Programmierung unter Android. Mit NDK ruft Java C Methoden auf. Auch die API von ARCore ist eine C-API, generell wird in Android NDK aber mit C++ programmiert und die Schnittstelle in

```
extern "C"{

}
```

gekapselt. Deswegen wird im folgenden oft wechselnd über C oder C++ geredet.

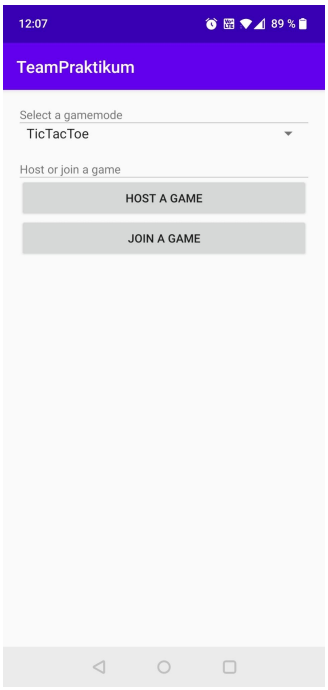
2.2 ArCore

2.2.1 Entwicklung eines Prototypen

Am Anfang musste ich die Abhängigkeiten im Buildsystem auflösen, um ArCore verlinken zu können. Zusätzlich musste ich mich mit dem grundlegenden Lebenszyklus von Apps auseinander setzten und wegen Android NDK damit beschäftigen, wie man Parameter von einer Java-Funktion an C übergibt. Ich habe im nachhinen gemerkt, dass es die Möglichkeit gibt direkt in Android eine ´Native activity´[Andb] zu erstellen die nur C++ nutzt, aber alle OpenGL ES Code-samples von Google[And21], sowie das AR-Core C Beispiel[ARCb] nutzen Java mit eingehängten C++ und auch beim erstellen einer ´Native App´ in Android Studio, wird eine Java Activity mit gelinked C++-Code bereitgestellt.

Daraufhin habe ich eine Dummy- Implementation von ArCore geschrieben, um ArCore zum laufen zu bringen und besser zu verstehen. Die Dokumentation von Google zu AR-Core war ein wenig kurz gefasst und ich musste dementsprechend immer wieder auf das Beispiel ‘`hello_ar_c`‘, um die Funktionsweise von ArCore zu verstehen. Lange Zeit zeigte die Kamera kein Bild an, dabei stellte sich heraus, das ArCore ein samplerExternalOES anstatt eines einfachen sampler2D benötigt.

Danach merkte ich, dass der Android Emulator nur bedingt OpenGL ES 3.0 supportet[Anda], deswegen musste ich den Code auf OpenGL ES 2.0 porten, was gerade wegen Vertex-Array-Objects und eigene Sampler-Objects umschreiben benötigt hat.

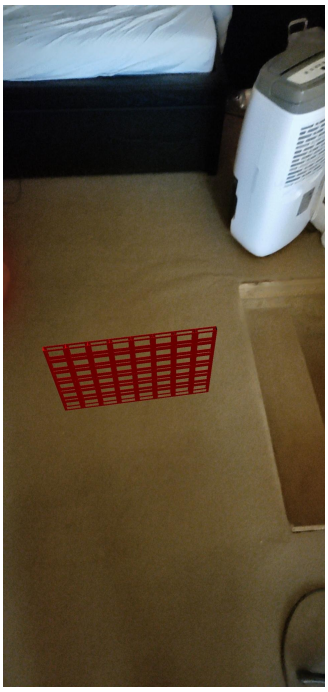


(a) Main Menu

(b) Host Game Menu

(c) Join Game

(d) After Loading



(e) After surfaces where found and clicking on screen

(f) After game has ended

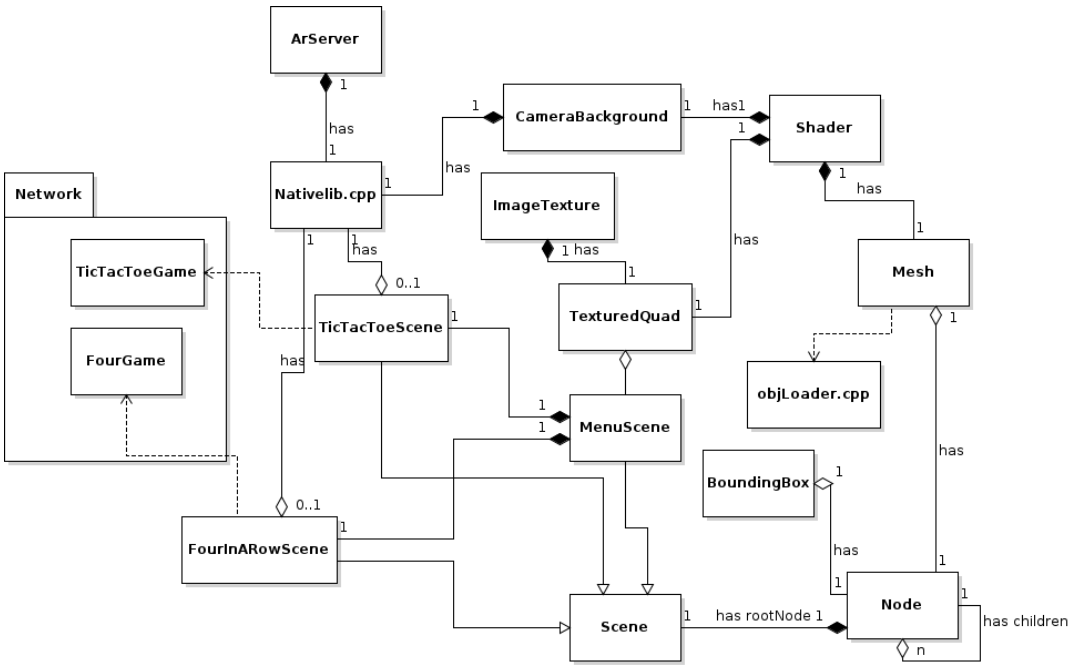


Abbildung 2.1: Overview of the ArCore/GUI Part of the App

Der Android Emulator hatte einen Bug und somit konnte ArCore App nicht auf die virtuelle Kamera zugreifen. Das Problem bestand auch bei der Google Beispiel App "`hello_ar_c`", aber auch in der Java Version.

Da sich auch heraustellte, dass mein Smartphone ArCore nicht supportet, wurde ich darauf aufmerksam, dass es eine Liste von supporten Geräten gibt. [ARCc]

Zu dem Zeitpunkt schlug unser Betreuer, Herr Uhlmann, vor, auf OpenCV[Ope] zu wechseln, um dementsprechend dem Problem zu entgehen. Ich schaute mir OpenCV an, aber da die Entwicklung in OpenCV wohl einen sehr viel größeren Arbeitsaufwand benötigt hätte, entschied ich mich ein ArCore supportetes Gerät zu kaufen.

Auch unser Betreuer, war dementsprechend entgegen kommend und holte sich ein supportes Tablet.

2.2.2 Funktionsweise von ArCore

ArCore[ARCa] ist eine Bibliothek von Google, die unter Android ermöglicht, AR Applikationen zu entwickeln. ArCore analysiert dafür den Kamerastream und kann anhand dessen, die Umgebung erkennen.
ArCore bietet dafür folgendes:

arCamera: gibt einem Metainformationen über das Kamerabild,sowei eine Projektionsmatrix und Kameramatrix.

arPlane sind Oberflächen die von ArCore erkannt worden sind.(Wird automatisch generiert)

arPoint sind Punkte im reellen 3D Raum die erkannt worden sind.(Wird automatisch generiert)

arAnchor wird vom Entwickler erzeugt, arAnchor erwaret eine Bildposition(zumeist Touchposition), an dieser Bildposition versucht ArCore anhand von eigenen Daten (arPlane,arPoint), den Punkt auf dem geklickt wurde zu erzeugen. Wenn das geklappt hat, wird pro Kameraframe versucht,diesen Anchor zu tracken und gibt für diesen Anchor eine Transformationmatrix zurück.

2.2.3 Rendering

Erste Hilfsklassen

Für das Rendering habe ich am Anfang ein paar Hilfsklassen gebaut:

Shader in Shader.cpp für das lesen von Shadern aus Dateien, compilen und linken.

objRenderer in objRenderer.cpp für das Rendern von obj-Files. Später wurde die Funktionalität in die Mesh Klasse überführt.

cameraBackground in cameraBackground.cpp, die das Kamerabild auf einen Screen-Quad sampelt.

Aus dem `hello_ar_c` von Google habe ich die objLoader.cpp entnommen, um obj-Dateien laden zu können.

Weitere Entwicklung

Nachdem nun ArCore grundlegend lief habe ich einen Würfel erfolgreich geladen und rendern können. Nachdem ich dann mit einem Klick auf dem Bildschirm einen Anchor erzeugen lassen konnte und der Anchor richtig getracked wurde. Habe ich dann ein simples Szenen System implementiert:

Mesh in Mesh.cpp, dieses lädt eine obj-Datei mithilfe von objLoader.cpp und ermöglicht das rendern dieses Meshes über draw().

Node in Node.cpp enthält

- Mesh zum rendern
- ModelMatrix in Relation zur Elternnode
- Kinder, die auch Nodes sind

Scene in Scene.cpp, diese enthält die Root-Node. Im Konstruktor der Scene werden die Nodes an die Root-Node angehängt oder an andere Nodes die schon angehängt wurden. Beim draw, wird das draw der Root-Node aufgerufen und jede Node ruft widderum die draw Methode ihrer Childnodes auf und übergibt dabei die eigene Modelmatrix.

Damit wurde das Rendering sehr viel leichter und der Code sehr viel aufgeräumter.

2.2.4 Probleme mit ArCore

Das entwickeln mit ArCore verursachte mehrere Schwierigkeiten, dass größte Problem dabei war die NDK(C) Version von ArCore. Dabei wurde die objektorientierte API in eine C-API umgewandelt, was dazu führt, dass wenn in der Javaversion ein Objekt ein anderes managed, muss in C immer wieder das Objekt in Aufrufen mitgeschliffen werden.
Hier ein Beispiel, um in der C API, die Modelmatrix von einem getrackten Punkt(anchor) zu bekommen:

```
ArPose *pose_;
ArPose_create(arSession, nullptr, &pose_);
ArAnchor_getPose(arSession, anchor, pose_);
ArPose_getMatrix(arSession, pose_, glm::value_ptr(modelMatrix));
ArPose_destroy(pose_);
```

Siehe [ARAa] für mehr Details.
Wie zu sehen, muss erst eine ArPose erstellt werden, die ArCore erzeugt, dann muss man an diese Pose, die Pose des Anchors binden und kann dann die ModelMatrix bekommen und am Ende muss die Pose wieder gelöscht werden. Dabei muss die arSession immer wieder übergeben werden. Der gleiche Code in Java:

```
modelMatrix = anchor.getPose().toMatrix();
```

Siehe [ARAb] für mehr Details.
Somit hat es recht lange gedauert, bis ich einen guten Überblick über die C-API hatte.

Am Anfang hatte ich die API direkt in der Nativelib.cpp Datei, die die Schnittstelle zwischen Java und C++ darstellt und habe diese später in arServer.cpp migriert, was widerrum die Codekomplexität massiv senkte.

2.2.5 Game

Nachdem ich ArCore und ein gute Szenenabstraktion hatte, habe ich noch eine Hitbox detection geschrieben, diese basiert auf [Hit] die mir Herr Uhlmann gab.

Danach haben wir unsere beiden Entwicklungsbranches(Netzwerk, ArCore) gemerged, damit konnte ich dann auf die GameStates zugreifen und dementsprechend für TicTacToe das Feld rendern. Dabei sind keine größeren Probleme aufgetreten.

Da wir jetzt 'Vier Gewinnt' implementieren sollten, habe ich die Scene zu einer vererbaren Klasse refactored (in TicTacToeScene.cpp) und daraufhin Vier gewinnt implementiert(FourInARowScene.cpp).

Zu guter Schluss habe ich ein Menü eingebaut, dass am Ende des Spiels angibt, ob man verloren oder gewonnen hat und ein Button, um das Spiel neuzustarten. Um den Text anzuzeigen, habe ich einen TGA Loader geschrieben und musste erfahren, dass vom Compiler nicht gewährleistet wird, dass structs tightly packed sind, weshalb die Headerdaten der TGA nicht richtig ausgelesen wurden. Weswegen ich das Problem mehrere Stunden debuggen musste. Mit dem fertigen Menü, war dann auch die App fertig.

2.3 Netzerk

2.3.1 Grundlegende Netzwerkfunktion

Als erstes stellte sich die Frage, welche Art der Datenübertragung für TicTacToe und Vier gewinnt am sinnvollsten ist. Ich musste mich hier zwischen UDP und TCP unterscheiden. Beide haben Vor- und Nachteile, die für Echtzeitspiele zu beachten sind:

Heutige Onlinespiele verwenden häufig UDP, da dieses Protokoll verbindungslos ist. Das bedeutet bei einer Übertragung wird ein Paket ohne Absicherung verschickt. Der Sender kann dabei nicht wissen, ob das Paket beim Empfänger angekommen ist oder, falls mehrere Pakete gesendet wurden, ob diese in der richtigen Reinheinfolge beim Empfänger eingetoffen sind. Bei kurzen Statusmeldungen in Spielen, wie Positionsupdates von Spielern, ist UDP passend, da diese Updates sehr schnell wieder durch neuere Informationen obsolet werden. Der Client sollte sich also nicht damit

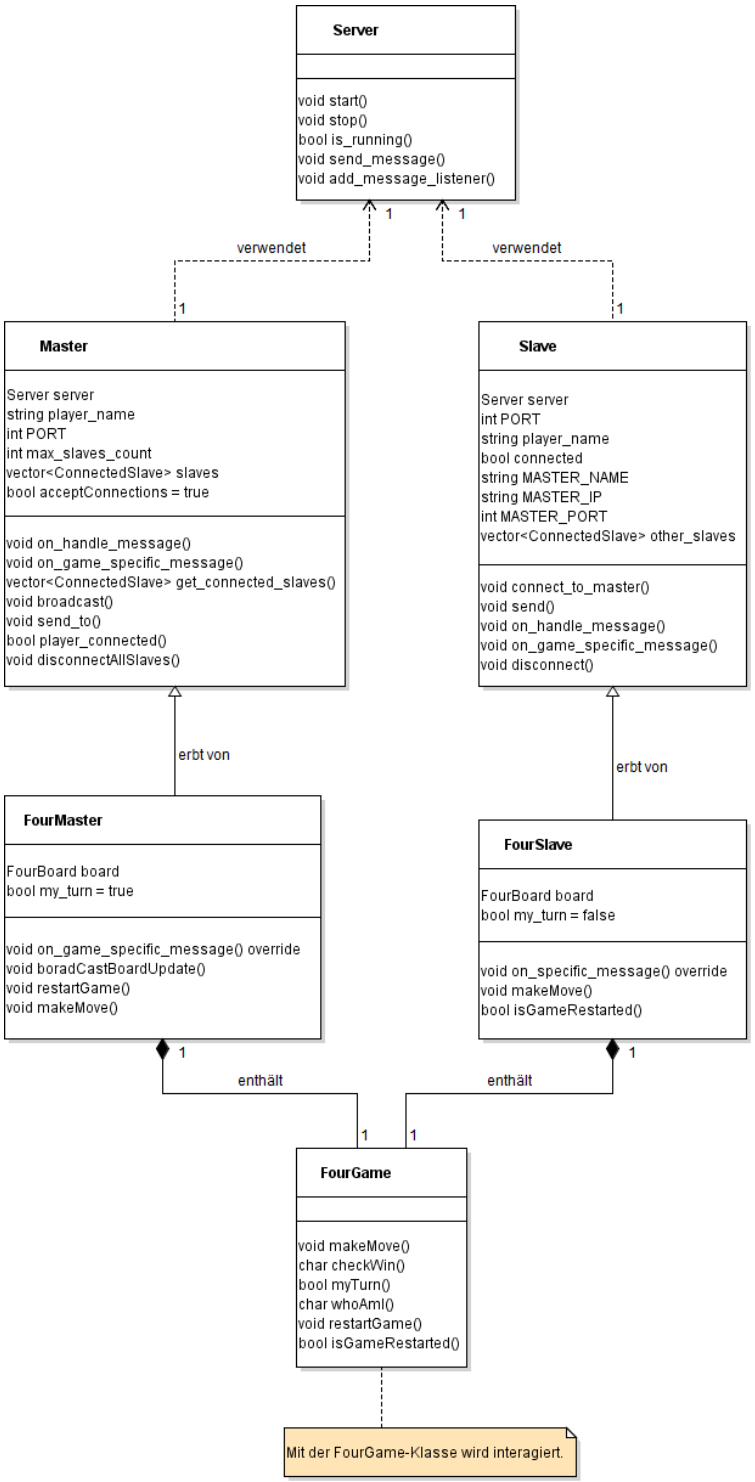


Abbildung 2.2: Overview of the Network Part of the App

beschäftigen alte Pakete zu ‘retten‘, sondern möglichst immer bereit für das neuste Paket sein.

Es gibt aber verschiedene Gründe TCP in Spielen zu verwenden. Gerade wenn es sich bei den zu übertragenden Daten um wichtige Statusmeldungen handelt, die nur einmalig kommuniziert werden, ist es wichtig, dass diese beim Empfänger ankommen. Wenn beispielsweise ein Spieler in einem Spiel beitritt, wäre es fatal, wenn diese Information bei einem der Mitspieler nicht ankommt. Die Spielumgebung wäre nicht mehr gleich für alle Teilnehmer und das Spiel unter Umständen garnicht spielbar. TCP eignet sich sehr gut für solche Statusmeldungen, da es erst eine Verbindung aufbaut. Solange diese Verbindung existiert, ist garantiert, dass die gesendeten Pakete eintreffen. Auch die richtige Reihenfolge ist garantiert.

Wenn man sich Spiele wie TicTacToe und Vier gewinnt anschaut, bemerkt man schnell das diese Spiele keine großen Mengen von Updates der Spielumgebung benötigen. Es sind nur wenige, dafür aber kritische Meldungen erforderlich, um die Spiele für beide Teilnehmer synchron zu halten.

Benötigt werden beispielsweise für TicTacToe:

- Statusmeldung wo ein Spieler sein X oder O setzt.
- Statusmeldung ob jemand gewonnen hat und wer.
- Statusmeldung über den Abschluss des Spiels wenn das Spielbrett voll ist und es keinen Sieger gibt.
- Statusmeldung für den Neustart des Spiels.

Alle diese Meldungen sind per TCP besser umzusetzen, da sie in jedem Fall beim Empfänger ankommen müssen. Sonst wäre das Spiel nicht mehr synchron und ggf. in einem undefinierten Zustand.

2.3.2 Implementierung der Grundfunktionen

Da wir uns für die Entwicklung für Android NDK entschieden haben, setzte ich die Implementierung in C++ um. Dafür war es notwendig zu verstehen, wie C++ mit der normalerweise unter Android verwendeten Programmiersprache Java zusammen arbeitet. Das Java Native Interface (JNI) wird benutzt, um Objecte zwischen Java und C++ hin und her zu ‘senden‘. Diese Übertragungen sind allerdings zeitaufwendig. Da der visuelle Teil mit ARCore von David auch in C++ implementiert wurde, wollte ich das Netzwerk auch in C++ umsetzen.

Zuerst habe ich Sockets verwendet, um Rohdaten empfangen und senden zu können. In unseren Meetings mit Herrn Uhlmann wurde mir dann vorgeschlagen statt Rohdaten Structs zu verschicken. Nachdem ich das umgesetzt hatte, wurde diese Idee

in einem weiteren Meeting erweitert zu Message-Klassen, welche Methoden mitbringen, mit denen die Objekte sich selbst in einen Bytestrom übersetzen und sich aus einem solchen Bytestrom wieder rekonstruieren können.

2.3.3 Asnychrones Empfangen

Als nächstes befasste ich mit dem Asynchronen empfangen der Daten. Da man dauerhaft mit einer Funktion auf eine TCP-Verbindung warten muss, um dann die Nachricht zu verarbeiten, macht es Sinn, zumindest das Warten auf neue Verbindungen in einem seperaten Thread zu realisieren. Dieser Thread ruft dann eine Callbackfunktion auf, die dann mit der Nachricht weiterarbeiten kann.

2.3.4 Asynchrones Senden

Das asynchrone Senden hatte ich mir erst im späteren Verlauf des Projektes vorgenommen. Ich dachte, das das Senden nicht sonderlich zeitaufwendig sein würde und den Spielfluss kaum beeinträchtigen könnte. Damit lag ich falsch. Wenn nämlich ein Fehler beim Senden auftritt oder das Senden länger braucht als sonst, kann das die ganze App anhalten. Das ist natürlich nicht erstrebenswert. Also realisierte ich eine Warteschlange in die Nachrichten eingefügt werden. Diese versendet dann der Reihe nach ein anderer Thread. Bei einer langsamen Verbindung wird dann nicht die gesamte App verlangsamt.

2.3.5 Abstaktion

In Spielen gibt es im Netzwerk oft eine Hierarchie, welche regelt, wer der Host eines Spiels ist und wer nur als Client beitritt. In unserem Projekt haben wir die Begriffe Master und Slave gewählt. Der Master hält den einzig wahren Spielzustand. Die Slaves übernehmen diesen und senden dem Master ihre Spielzüge wenn sie an der Reihe sind. Wenn der Master den Spielzug als valide einstuft, werden entsprechende Änderung am Spielfeld vorgenommen und diese allen Slaves mitgeteilt. So ändert ein Slave also nie sein eigenes Spielfeld. Nur der Master kann Änderungen für sich und alle Slaves durchführen. Dadurch wird es viel einfacher die Spielfelder zwischen allen Spielern synchron zu halten.

2.3.6 Konkrete Umsetzung für Spiele

Um für David den Netzwerkcode einfach zugreifbar zu machen, habe ich Subklassen von Master und Slave für das jeweilige Spiel abgeleitet. Diese können dann genutzt werden, um Spielzüge zu machen, ohne dass man sich um die Netzwerkschnittstelle kümmern muss. Master und Slave sind dann nochmal in einem Game-Objekt zusamengefasst, damit David den Code für Master und Slave einheitlich aufrufen kann.

2.3.7 Testen der Netzwerkfunktionen

Ursprünglich wollte ich den Emulator nutzen, um die Kommunikation zwischen zwei Instanzen der App zu testen. Der Emulator hat allerdings eine eigene Firewall, welche für mich nicht so offensichtlich zu konfigurieren war. Zusätzlich hatte ich das Problem, das mein eigentlicher Arbeitscomputer durch unglückliche Umstände keine Virtualisierung unterstützt. Dadurch musste ich für die Emulation immer einen seperaten PC verwenden. Das war mir auf Dauer dann zu aufwendig und so entschied ich mich ein Python-Skript zum schreiben. Damit sollte man das TicTacToe-Spiel in der Kommandozeile gegen einen Spieler mit der App spielen können. Das Skript kann die Rolle das Masters und des Slaves einnehmen. Damit gelang es dann auch die Netzwerkfunktionen zu testen.

2.3.8 Aufbauen der Verbindung in der App

Damit Spieler nicht IP-Adressen auslesen und eintippen müssen, erfolgt der Austausch über einen QR-Code. Diese Idee kam David, als er beim Testen der App ständig seine IP-Adresse eingeben musste. Der Spieler, der das Spiel als master eröffnet, kann nun einfach einen QR-Code generieren und anzeigen lassen, welcher die lokale IP-Adresse des Geräts enthält. Der beitretende Spieler kann diesen in seiner App einscannen und die Verbindung einfach und bequem aufbauen.

3 Resümee

3.1 NDK

Für NDK wäre es zu empfehlen, direkt eine Native Activity zu nutzen. Dabei ist zu bedenken, dass NDK keine UI Elemente hat, somit muss alles selbst gerendert werden. Es wäre also zu empfehlen, mindestens auf eine UI Library zu setzten wie beispielsweise Nuklear oder ImGUI oder auf ein komplettes Rendering-Framework, wie beispielsweise Raylib.

3.2 ARCore

Für ArCore sollte man auf die Java API(oder Unity-/Unreal Plugin) setzen oder, wenn man es in C++ benutzten möchte, wäre es zu empfehlen, einen Wrapper zu bauen, der die C-API in eine C++ API wandelt, die ähnlich zu der Java-API ist.

3.3 Netzwerk

Durch die Socket-Struktur, welche in eigentlich allen Programmiersprachen zum Einsatz kommt, ist die Umsetzung einer Netzwerkschnittstelle im Kern unkompliziert. Die Hauptaufgabe ist es darauf dann eine oder mehrere abstrakte Schichten aufzubauen, die sich nach den Anforderungen des Projekts richten. Auf diese Weise gelang es mir die Interaktion mit dem Netzwerk stark zu vereinfachen.

3.4 Projektstruktur

Die Projektstruktur unterscheidet zwischen network und arCore. Generell wäre es empfehlenswert, das Projekt stärker wie eine Bibliothek oder Framework zu strukturieren, damit wäre von ‘TicTacToe‘ zu ‘Four in a row‘ weniger aufwand notwendig gewesen.

Literaturverzeichnis

[Anda] *GLES3 content gles3jni from ndk examples fails with 'java.lang.RuntimeException: createContext failed: EGL_BAD_CONFIG' [68496715] - Visible to Public - Issue Tracker*.
URL `https://issuetracker.google.com/issues/68496715`

[Andb] *NativeActivity*.
URL `https://developer.android.com/reference/android/app/NativeActivity?hl=de`

[Andc] *Android NDK*.
URL `https://developer.android.com/ndk?hl=de`

[And21] *NDK Samples*, Nov. 2021, original-date: 2015-05-16T10:02:15Z.
URL `https://github.com/android/ndk-samples`

[ARAa] *ArAnchor | ARCore | Google Developers (NDK/C-API)*.
URL `https://developers.google.com/ar/reference/c/group/ar-anchor`

[ARAb] *Anchor | ARCore | Google Developers (Java API)*.
URL `https://developers.google.com/ar/reference/java/com/google/ar/core/Anchor`

[ARCa] *Build new augmented reality experiences that seamlessly blend the digital and physical worlds*.
URL `https://developers.google.com/ar`

[ARCb] *Releases · google-ar/arcore-android-sdk*.
URL `https://github.com/google-ar/arcore-android-sdk/releases`

[ARCc] *ARCore supported devices*.
URL `https://developers.google.com/ar/devices`

[Hit] *A Minimal Ray-Tracer: Rendering Simple Shapes (Sphere, Cube, Disk, Plane, etc.) (Ray-Box Intersection)*.
URL `https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-box-intersection`

[Ope] *Home (OpenCV)*.
URL https://opencv.org/

[Ope16] *OpenXR - High-performance access to AR and VR —collectively known as XR— platforms and devices*, Dez. 2016, section: API.
URL https://www.khronos.org/openxr/

[Weba] *WebAssembly*.
URL https://webassembly.org/

[Webb] *WebVR - Bringing Virtual Reality to the Web*.
URL https://webvr.info/

[Webc] *WebXR Device API*.
URL https://www.w3.org/TR/webxr/

[Xam] *Xamarin | Open-source mobile app platform for .NET*.
URL https://dotnet.microsoft.com/apps/xamarin

Selbstständigkeitserklärung

Hiermit erkläre ich, da 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 ausdrcklich als solche gekennzeichnet.

Chemnitz, den 5. Januar 2022

David Eilers, Oskar Hippe