Bilkent University

Department of Computer Engineering

Senior Design Project

Coda

Final Report

Merve Kılıçarslan, Yağız Efe Mertol, Ege Özcan, Çağla Sözen, Murat Tüver

Supervisor: Prof. Dr. Uğur Güdükbay
Jury Members: Prof. Dr. Halil Altay Güvenir, Prof. Dr. Fazlı Can

Innovation Expert: Prof. Dr. Veysi İşler (SimSoft)

Final Report
May 26, 2020

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491.

Contents

Project Final

Coda

1 Introduction

Music as a form of art has been the common interest of people used to express feelings and identity through a composition of rhythm, timbre and melody. In different forms and sounds by making use of the variety of instruments, music is present in almost every context for numerous purposes. Besides the pleasure of listening, it's been proven that playing instruments have positive effects on brain development, especially for spotting statistical patterns enabling the learner to better predict what would happen next in a pattern, so every child has the right to better themselves with the help of instruments insomuch as discovering their musical talents [1].

Over the last 20 years, the number of children learning to play an instrument or playing an instrument has increased significantly [2]. However 26% of children and 49% of adults in the UK stated that they've given up playing instruments although they've learned to play or started to [2]. Most common reasons for this are loss of interest, instrument cost, lesson costs and competing pressures from school [2]. Furthermore, the fact that some instruments are highly immobile by nature makes practicing very challenging for both individuals and for groups of people who practice together. In most cases, instruments become idle and forgotten because of the impracticalities mentioned. As a result, buying instruments may be seen as an unnecessary expense. When the cost of learning instruments and the instruments itself is taken into consideration in addition to immobility, it can be stated that instruments can be made further accessible.

Therefore, there is need for solutions to make playing and learning instruments more sustainable by making them more accessible in several aspects like cost and mobility and we believe that with the increasing number of smartphones, mobile phones can be used to address this problem.

This report explains the details of the final version of Coda including *Requirements Details, Final Architecture, Development/Implementation, Testing Details, Maintenance Plan,* with the help of UML for proper documentation of the system and clarity of the final system design. Furthermore, this report explains the *Consideration of Various Factors, Ethics, Impact, Project Plan and New Knowledge.*

2 Requirements Details

2.1 Functional Requirements

2.1.1 Instrument

The main functionality of *Coda* depends on the rendering of an instrument in Virtual Reality. Thus, the first requirement is modeling the instrument in 3D that enables the user to interact with it visually, and get both auditory and visual feedback. For feasibility reasons, our first instrument of choice is **Drums**.

2.1.2 Free Mode

In the Free Mode, the use is able to play instruments however they like without any constraints or directives. This mode can be used for making new compositions or trying an instrument to learn how the instrument is used and so forth.

2.2 Non-Functional Requirements

2.2.1 Usability

Since one of the main objectives of *Coda* is providing accessibility, the following requirements should be matched

- Gestures to play an instrument should be intuitive. The player should be able to successfully complete the desired action in a single gesture.
- Controls in the system should be straightforward and the interface should be user-friendly. The user should be able to choose a piece from the library and start playing in no longer than 1 minute.
- The application should have clear instructions for initialization. The initialization stage and initial hand recognition should not take longer than 2 minutes when application performance is neglected.

2.2.2 Responsiveness

Responsiveness of *Coda* is critical since the application works in real-time and should provide instant visual and auditory feedback as a real instrument would. The following responsiveness requirements should be met,

- Frames Per Second (FPS) should not drop below 25.

2.2.3 Extendibility

For the main purpose of *Coda,* the application should be extendible by the following,

- Design of the application code should be written in such a way that new instruments that will enable addition of new instruments.

- Storage in the system should be designed in such a way that it can be extended to be shared over a network or by utilising the cloud.
- Design of the network usage should be in a way that it supports a potential implementation of **band mode** which allows people from different devices to collaborate and play different instruments at the same time.
- Practice mode can be implemented to extend Coda, where the user will choose a piece from the library provided with the app. Library will also contain pieces saved using Free Mode. According to the piece chosen, the user will be given visual directives and will be expected to interact correctly with the instrument. This mode can be used to practice a particular piece.
- Recording a session may be implemented for the extendibility of Coda and stop the recording whenever they want. These recordings will be saved in the library of the app and will be offered among the pieces in the library for playing in the Practice Mode.

2.3 Pseudo-Requirements

2.3.1 Implementation Constraints

Since *Coda* only depends on a smartphone and a Google Cardboard headset, it is important that the smartphone used provides the following requirements,

- Phone used has at least one camera.
- Phone used should have Android 7.0 or higher installed.
- *Coda* relies on real-time gesture detection and recognition. Manomotion Lite SDK used for hand gesture recognition runs the models on the cloud, so Coda requires a stable internet connection.
- GitHub was used for Version-Control.
- Trello was used for Issue Tracking.
- Implemented instruments ,if extended to multiple instruments, must not be played while attached to the body. Since we are using a headset, camera is not going to be able to detect the hand gestures while playing instruments like violin. Instead instruments like drums which are played within camera's line of sight were implemented.

2.3.2 Economic Constraints

During the implementation of *Coda*, the following economical constraints was taken into account since the project is not funded by any means,

- Frameworks and libraries used are open-source, so free to use.
- The website is be powered by GitHub thus, there is no domain rental cost.
- GitHub is a free Version Control tool so there is be no expense for Version Control.
- For development and demo, Google Cardboard or a similar headset is used which is expected to cost between 20-60 TL [8].
- One-time-only fee for publishing the app on Google Play is 25 USD. However Coda is not yet be published to the Google Play store.

Total cost of the application will come to approximately 40 USD.

2.3.3 Professional and Ethical Constraints

All the practices during the implementation and during deployment of *Coda* complies with the following in accordance with the Code of Ethics proposed by National Society of Professional Engineers [9],

- *Coda* is an application that depends on music. *Coda* either owns the rights of all the pieces and songs provided in the library or pays for including them in the library to the right owner.
- User data is not be shared with any third parties under any circumstances.
- No user data is not saved without the consent of the user.
- No ads are be displayed for financial means.
- Any external software or library used in the development of the project were properly referenced if it is protected by copyright.

2.3.4 Time Constraints

Development of *Coda* and it's documentation was be in line with the following schedule [10],

- - Project Specifications: **Monday, October 14, 2019**
- - Analysis Report: **Monday, November 11, 2019**
- - High-Level Design Report: **Friday, December 31, 2018**
- - Low-Level Design Report: **Monday, February 17, 2020**
- - Final Report: **Thursday, May 27, 2020**
- - Presentations & Demonstrations: **May 27 - June 14, 2020**

2.3.5 User Experience Constraints

For providing a comfortable user experience the following should be taken into account,

1. The app should not be used for more than 30 minutes at once in order not to lose the notion of spatial awareness which may cause headaches and dizziness [11].
2. Rendered virtual instrument should approximately is the same size as the original instrument for playing intuitively and enhancing the learning process.
3. Visual directives given to the user should be are easily understandable by the user.
4. For better auditory immersion, headphones may be utilised by the user by plugging them in the phone.
5. An internet connection must be established in order to use the application. Otherwise hand tracking is not possible.
6. The app is launched in English since it is more universal. Other language implementations were currently disregarded.

3 Final Architecture and Design Details

3.1 Overview

Coda aims to provide a minimal architecture in terms of hardware and comply to the architectural principles of Unity. Therefore, the software architecture of the system is restricted to the smartphone components. In this section of the report, the final architecture of the final system will be described in detail by presenting the Subsystem Decomposition and further explaining the layers described in the Subsystem Decomposition individually. Then mapping of the hardware and the software in the system is explained.

Further information and all documentation on *Coda* will be published on if comparison between the final and the planned architecture is inquired,

https://ege0zcan.github.io/coda/

3.2 Subsystem decomposition

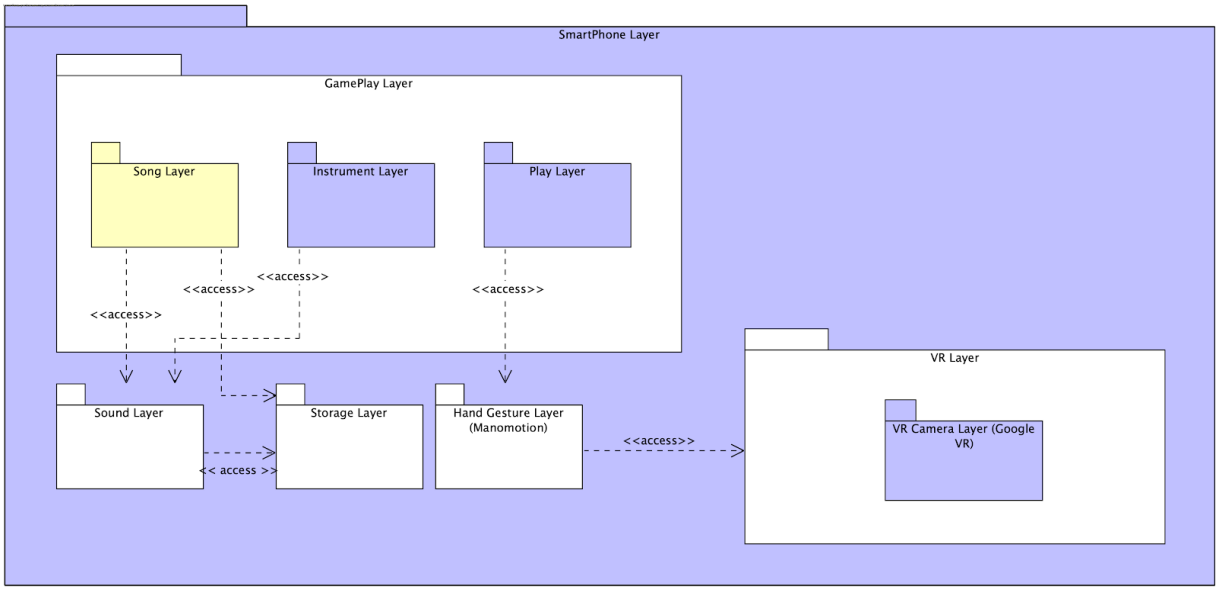

Figure 1: Subsystem Decomposition Diagram of *Coda*

Coda's architecture is limited to the smartphone itself. All sublayers are within the *SmartPhone Layer*, which is the most comprehensive layer of the system. In this architecture, there is no distinction between the client side and the server side since both sides depend on the local system only. The only external subsystem is within the Hand Gesture Layer (Manomotion) since the SDK used for hand tracking runs the Computer Vision models on the cloud requiring internet connection. For the sake of simplicity we did not include the internal subsystem details of the libraries used during development.

Within the *SmartPhone Layer*, there are 5 sub-layers. The *GamePlay Layer* is the layer that provides the gameplay experience to the user, responsible from handling the relations between other layers and the client-side, in a sense. It accesses all other layers for synchronizing the VR, Sound, Storage and Hand Gesture components of the application with the user interactions.

The *VR Layer* is responsible from handling the VR operations within the system and it is accessed by the *GamePlay Layer* for constructing the VR environment to be presented to the user as an augmented replicate of the real environment. For achieving this, the *VR Layer* also handles the information gathered from the camera of the smartphone. Internal implementation of the VR layer consists majorly of Google VR library components and the AR camera provided in the same library for handling VR application requirements like getting data from the camera input and stereoscopic rendering for the VR glasses.

The *Sound Layer* is responsible from providing the auditory experience to the user, also interacting with the device speakers. This layer is also accessed by the *GamePlay Layer* for giving the the auditory response according to the user interaction. This layer consists of the sound assets in unity and the SoundManager class that manages the Audio Listeners of Unity according to the desired functionality. The sound layer accesses the storage layer to fetch the related audio files from the file system once it is called by the Part class in the Instrument layer.

Storage Layer is the layer responsible from handling the persistent data management and storing the game related data like the audio files. This layer is designed to also accessed by the *GamePlay Layer* for storing the library songs as well as the user recordings however this feature is not implemented for the final version of Coda due to time constraints. Storage Layer is mainly used to access the specific Instrument part sounds by the Sound Layer.

Finally, the *Hand Gesture Layer* is the layer for handling and processing the *Hand Gestures* of the user from the camera and recognizing the user interaction, and intention, accordingly. This layer consists of the Manomotion Library which is chosen as the hand tracking library used for the final version of the project. This layer is used with a Manomotion Manager instance to track the position of the hands continuously and to detect hand gestures for interacting with the menu.

3.3 Hardware/software mapping

Coda is an app that is designed to run on Android smartphones implemented on software and also using hardware components such as the camera and the screen. Although Coda is an AR based application, it requires no external hardware mapping other than the smartphone itself. Only the manual and straightforward integration of the smartphone to the Google Cardboard (or alike) headset is required.

Software components of *Coda* is as follows,

- Java will be used as the primary software development language along with Android Studio for platform-independency, object-oriented tools and compatibility with useful frameworks and libraries such as Manomotion, Google VR etc.
- Unity will be the platform for developing the graphics components.

All of the hardware mappings will be done by built-in Android SDK and Unity mappings to the smartphones running on Android. A simple representation of the Hardware/Software Mapping can be found below in the Deployment Diagram.

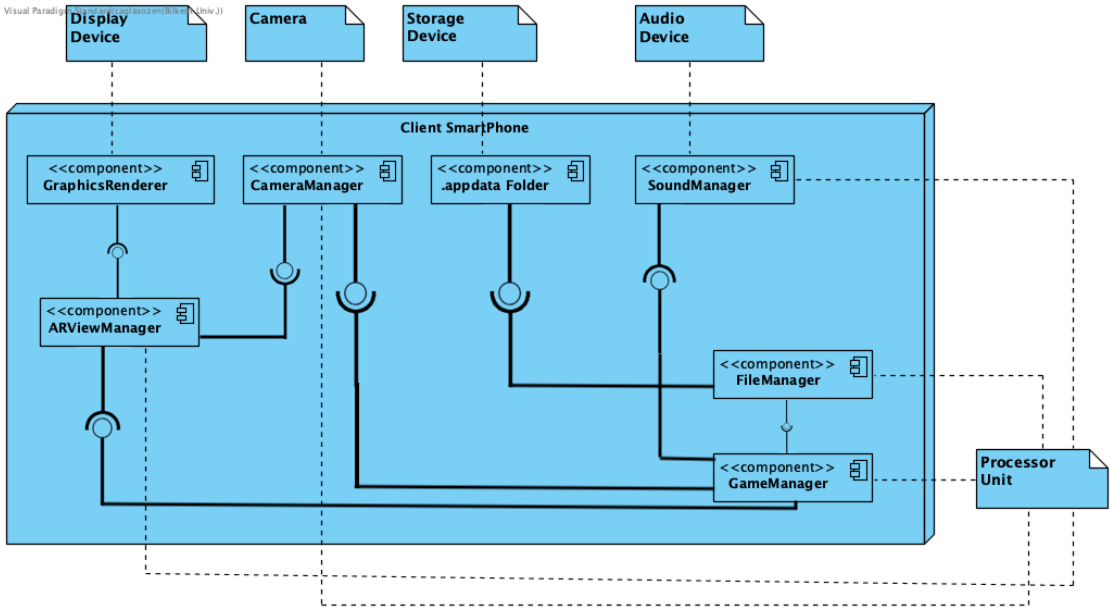

Figure 2: Deployment Diagram of *Coda*

4 Development/Implementation Details

Development details of Coda consisted of 4 main components which can be reduced to the following subsections,

- Working with Google MediaPipe.
- Working with Unity.
- Working with Manomotion.
- Working with GoogleVR.

Some of these components were not included in the final implementation but consumed more time compared to the components that were used in the final version. During the development, we had to make a lot of design decisions according to the time, environment and requirements constraints. Effects of these components will be explored in detail in the proceeding subsections.

4.1 Working With Google MediaPipe

Working with MediaPipe was very challenging in terms of the following aspects,

- Using MediaPipe in Unity.
- Mapping the Hand Model with the real time Hand Data.

We will explore these aspects in the proceeding sub-sections,

4.1.1 Using MediaPipe in Unity

As mentioned before, MediaPipe was the best library to be used for hand tracking and gesture recognition which was highly modifiable according to our needs and which was completely open source. However MediaPipe was a new library which launched in the last year, making it documentation and compatibility lacking. Our initial idea was to use the Vuforia library which also implements MediaPipe inside of it. However with that decision, we realized that because of the Tensorflow models used in MediaPipe, Unity Asset was missing from MediaPipe. So we had to find a way embed Mediapipe as a Unity asset and this led us to the decision to give up on Vuforia to be able to modify the assets of MediaPipe individually such that we would be able to port it to Unity.

We made extensive research on Stack Overflow and GitHub to find people who successfully used MediaPipe in Unity. However we found out that Google explicity announced that regardless of the high number of requests, they were not planning to port MediaPipe to Unity. So we had to come up with our own solutions to use MediaPipe in Unity.

The first approach we used was to port Tensorflow Lite to Unity which would theoretically solve the incompatibility problem of the Tensorflow Models used in MediaPipe. For this reason, we tried to replicate and used a Unity plugin called TensorflowLite4Unity which was used by several other developers for similar projects. This plugin would solve the problem of quantized models that are not currently runnable by the Unity Interference Engine by integrating TFLite inference to the Unity Engine with a good performance. We tried several approaches to be able to use this plugin in unity with MediaPipe, however the code for the plugin was very poorly documented and complex. In the end we could not manage to use the plugin successfully in Unity to support hand tracking with MediaPipe.

The second approach used was to export MediaPipe as an Android Archive (AAR) file to add it as if it was an external android library to call the Java functions in the Unity application. So we needed to write extra functions to the AAR build we created from MediaPipe such that it would behave as we needed in Unity. We wrote several more functions into MediaPipe to provide the functionality we would need in Unity for Hand Tracking and created a build from the modified MediaPipe library. However when we modified the library, dependencies for the core MediaPipe library wouldn’t be injected in the exported AAR package. It would only show the names of the empty functions without the compiled functions. For solving this problem, we tried the FAT AAR Gradle plugin which is a plugin for compacting all the dependencies of an AAR into another library such that it would create another library file to provide independent building from the dependencies.

Finally, despite of all the approaches we tried we would lose the dependencies of MediaPipe while trying to build a custom library which extends mediapipe such that it

would be appropriate for the needs of our application. Since we were not able to solve this problem, we contacted foreign developers, field experts however we were not still able to solve the problem after an extreme amount of effort in almost 3 months. As a result, we finally decided to switch ton Manomotion due to time constraints despite the restricted functionality offered by Manomotion.

4.1.2 Mapping the Hand Model

For mapping a hand model to the data fetched from the Hand Tracking API, we found a 3D hand model that was usable and modifiable in Unity. We first started to build the model for the data received from Google MediaPipe which returned 21 3D points that were used to represent certain points in each hand. Hand model determined to be used consisted of 30 points that were similar to the ones defined in MediaPipe. However the challenge here was to synchronize the movement of the points that were defined in the hand model but not Mediapipe. For this purpose, mathematical and geometrical calculations were needed. So we created three classes to manage the movement of the hands according to the data obtained from MediaPipe: HandData, HandGameObjectData and HandManager.

Hand Data class was responsible from getting the handling the data received form Mediapipe in the form of 21 3D points in the Unity space. This class was never tested as we never managed to get MediaPipe to work in Unity. HandGameObjectData class was responsible from the hand model in unity with 30 points. And finally, HandManager would synchronize the Hand Game Object according to the data received from Hand Data by making the required geometrical calculations to render accurately.

Unfortunately, none of the classes were used in the final implementation since MediaPipe was not used as the Hand Tracking library and Manomotion Lite SDK only provided a bounding box and palm center coordinates for hand tracking which would not be sufficient for mapping a complex hand model in the game with the proper movements.

4.2 Working With Unity

Working with Unity was challenging in terms of the following aspects,

- Understanding and adjusting the camera for VR and Hand Tracking.
- Sound Management and Collision Detection.
- Architecture of Unity.

We will explore these aspects in the proceeding sub-sections,

4.2.1 Cameras in Unity

Camera organization in Unity for a VR application was critical to understand to be able to modify the application according to the needs of Coda. We used two cameras to form a correct view for the VR application with stereoscopic rendering and for also getting the camera input from the physical camera of the mobile phone. We positioned the camera such that it would render a scene close to reality with a real-life like instrument when rendered on the VR glasses. A Tracked Pose Driver was attached to the camera to track the head movements of the user and the headset. The Pose Driver tracked the rotation and position of the head of the user along with the headset and rendered the application view at each update. This driver was controlled by the ARCameraManager Script. Furthermore, a GvrPointerPhysicsRaycaster was attached to the same camera to provide a behavior similar to the standards Physics raycaster, except that it raycast modes are specifically designed for Google VR.

Cameras were also used for managing sounds and audio. Details of their management will be explained in the next section.

4.2.2 Sound Management and Collision Detection.

As explained in the previous subsection, the camera was utilised in the application to manage sounds and audio. An audio listener was attached to the camera to make the sounds. The architecture for managing sounds were controlled by a Sound Manager and each Instrument Part had its unique sound that was attached to its particular sound via it's part name. Collision Detection was handled by the colliders in Unity since we already received 3D Hand position information from the Manomotion SDK and the behaviour in the case of a collision was determined by the *isHit()* function of each instrument part.

Sound Manager in Coda controlled the sounds in the game from the Audio Listener attached to the camera that was also used to control the movement of the phone to adjust the VR rendering. The Audio Listener was commanded by the *Sound Manager* script. Sound Manager class followed the Singleton design pattern and contained an instance of itself as the singleton instance. Lifetime of this instance extended as long as the application was on. According to the instrument part in collision with the hands, the path was determined and the following method was called to make the appropriate sound accordingly,

```
public bool playPartSound(string path)
    {
        print(path);
        GameObject soundGameObject = new GameObject("Sound");
        AudioSource audio =
```

```
soundGameObject.AddComponent<AudioSource>();
        if (audio.PlayOneShot(GetAudioClip(path)))

          return true;
       return false;
    }
```

This method was used in the implementation of the Part class. Parts were used both for detecting collisions and for fetching the appropriate sound in the case of a collision with themselves. Each part in Coda had a part_name that was also used to point the related audio file. When a Part was detected to be collided with the hands of the user, the collider in the model was stimulated and the *isHit* method was used as the Event Trigger in the part to pass the sound manager the related information about the audio file to be played as the following.

```
public bool isHit()
        {
            return SoundManager.Instance.playPartSound(part_name);
        }
```

4.3 Working With Manomotion

Working with Manomotion was challenging in terms of the following aspects,

- Contacting Manomotion for the SDK
- Using Manomotion SDK for Hand Tracking.
- Using Manomotion SDK for Gesture Recognition.

We will explore these aspects in the proceeding sub-sections,

4.3.1 Getting Manomotion SDK

The only available alternative to Google MediaPipe was the Manomotion SDK. We were aware of the fact that Manomotion had two different kinds of SDK, one being free and the other one being paid. Because Manomotion Free SDK did not offer the exact functionalities we needed, our priority was always using Google MediaPipe. But as an alternative, we also requested the Free SDK from Manomotion in January. Unfortunately, we didn't receive it until March and when we received it we realized that it was much more constrained than we initially thought.

For getting the SDK, we first filed a request through the Manomotion website. However we were informed that the requests were addressed only with the discretion of the Manomotion Team arbitrarily. We waited for two months however we did not receive any response from the Manomotion team through the website or via email. Then we contacted some members of the Manomotion team via Linkedin. By this way, we finally received the Manomotion Lite SDK after waiting for almost 3 months. We continued to contact the team to request a 1-month free trial of the Pro SDK as we realized that the Lite SDK was not sufficient for the implementation of the complete version of Coda. However we did not end up with a positive response on the one month trial.

So we postponed working with the Manomotion SDK for hand tracking until we were sure that we wouldn't be able to use Google MediaPipe for hand tracking. Finally when we decided to switch to Manomotion SDK, since we couldn't get the Pro SDK, we implemented according to the constraints proposed by the Manomotion Lite SDK and continued the rest of the development according to these.

4.3.2 Hand Tracking with Manomotion

After we received the Manomotion Lite SDK, we started reading the documentation and watching the tutorials available in the Manomotion Ecosystem where the developers given permission to use the Manomotion SDK discuss issues they went through and their solutions.

Using Manomotion Lite SDK was fairly simple to use because it was designed specifically for Unity. We used two features of the Manomotion SDK: Hand Tracking and Gesture Recognition.

Hand Tracking in Manomotion was used for locating the bounding box of the hand and locating the palm center. Adding the Manomotion Manager to the scene in Unity was sufficient to receive Hand Data in Unity. However there was no rendering of the hands on the virtual environment so we wrote a *HandCollider* script to receive the information from manomotion manager and synchronize the hand/drum stick models according to the data received from Manomotion. The following code segment was used to update the location of the visual hand model according to the hand data from Manomotion with the required calculations,

```
Vector3 fetchedHandInfo =
manomotionManager.Hand_infos[0].hand_info.tracking_info.palm_center;

float handDepth =
manomotionManager.Hand_infos[0].hand_info.tracking_info.depth_estimation;

Vector3 handLocation = new Vector3(
```

```
(fetchedHandInfo[0]*4.1f)-2,(fetchedHandInfo[1]*2.86f)-1.6f,(handDepth*2.5f)+1
);

transform.localPosition = handLocation;
```

4.3.3 Gesture Recognition with Manomotion

For interacting with the menus, we used the Trigger Gestures Class of Manomotion. Gestures in this class are used for events in the application that are used in place of a click. These kinds of gestures can be defined as a sequence of hand states either customly defined or predefined by the Manomotion SDK. We used the predefined gestures defined in Manomotion because the only gesture to be defined in Coda was going to be for menu interactions.

Once again, Gesture Recognition in Manomotion was used for determining the states and hand poses at every update of Unity. We used the ManoClass: Pinch class to compare the current pose of the hands with the state defined in this class. Implementation of this part was fairly easy. The only problem caused by using Manomotion Lite SDK for both Gesture Recognition and Hand Tracking is when we were constrained by the performance of Manomotion hand tracking and hand recognition.

4.4 Working With Google VR

Working with Google VR was challenging in terms of the following aspects,

- Rendering the scene stereoscopically for AR.
- Trying occlusion for showing hands from the camera.
- Stopping Camera from rendering the background for performance while still using the camera for hand detection.

We will explore these aspects in the proceeding sub-sections,

4.4.1 Stereoscopic Rendering

First challenge to setup the application as an AR application was to render the application according to the VR glasses to be used. Coda was planned to be a MR application during the design phases, so focused on using an AR application on the glasses. For this, we initially started out using Google ARCore which is the library designed to create AR applications. However after trying the library with several different example applications, reading the documentation and trying to find similar applications to Coda on StackOverflow, we acknowledged that using an AR application was not feasible with Google ARCore.

Then we used Google VR library to create an application that automatically had stereoscopic rendering because it was already designed to be rendered in VR glasses. After reading the documentation and making some tweaks according to the results of our research, we managed to render the camera input as the background in stereoscopic view instead of a completely virtual background in a MR manner. However after implementing the basic functionality of the application we realised that forcing the application to render the background created a serious overhead for the application and decided to switch into VR instead of MR for performance. Challenges been through after this decision will be discussed in the proceeding subsections.

4.4.2 Hand Occlusion

After managing to render the application stereoscopically and in AR, the second change decision we made regarding the design of the application arose from the problem of occlusion. Although the Hand Tracking libraries used in the application returned 3D positions of the hands, automatic occlusion of the hands such that they would appear in front of the instrument models were not possible. Because such kind of feature was only provided by mobile phones with a depth-sensitive camera, which was decided not to be a prerequisite for using Coda since the primary mission of the application is to make instruments more accessible. Hence we once again decided that creating a modifiable and moveable hand model such that it's movement would be synchronized with the hand data fetched from the hand tracking libraries in VR instead of occlusion with AR would be a better solution.

4.4.3 No Camera Rendering while Camera is Working

Finally, our problems reduced down to the problems we faced as a result of the decision to use VR instead of AR or MR. The major challenge we faced at this point was to stop the camera from rendering in the background while still fetching the camera input for hand detection with the hand tracking APIs. The redundant rendering of the camera was coverable by the VR room in Unity however it created a huge performance overhead for the application. Furthermore, the room was torn at some arbitrary points and the camera input was showing from those tears.

For being able to get the camera input and render it in VR, our approach is to use two cameras, one being for getting the camera input and the other for rendering the virtual environment and the virtual instrument. Although we only need the AR camera for hand tracking, it is not possible to restrict the functionality of the camera in such a way, instead it also renders the complete input as the background. We worked around this issue by setting the camera input outside of the virtual environment such that it would be left at the back. This way since Unity does not render objects which are left behind another object, we eliminated the overhead of rendering the camera input.

For the tears in the virtual environment, we had to adjust the perspective of the screen to be rendered by the camera according to the virtual room such that it wouldn't intersect with the environment. So we had to adjust the distance between the furthest point of the virtual room and the camera to make it smaller than the distance between the camera and the screen. This way, there were no torn parts of the virtual room.

5 Testing Details

5. 1 Verification

We used Agile development methodology in Coda and used Scrum as a framework, hence we developed the project in small increments called sprints. We performed the verification activities of Coda embedded within the Sprints as soon as related components were ready. So at each Sprint, we made sure that Coda was built correctly which was much easier and much less expensive compared to a huge and comprehensive verification at the end of several sprints.

Following activities will be performed for the verification of Coda,

- **Reviews:** We performed code reviews before completing a code for making sure the code committed was free of first level defects and to increase the quality of code and exploring alternative and better solutions. As a side effect, code reviews promoted sharing knowledge and responsibility between the team members. Since we performed pair programming, reviews were in parallel with programming.
- **Walkthroughs:** We performed walkthroughs whenever a team member was going to take on a task which was previously assigned to somebody else, or when somebody was assigned to make an implementation related to another component for better understanding the code and using it whenever needed.
- **Automatic Static Code Analysis:** We used the automatic static code analysis features of the IDE Visual Studio Code for eliminating compilation time errors.

5.2.2 Validation

Validation activities of Coda were both embedded at the end of the Sprints as a form of testing (Development Testing) and at the end of all sprints to validate the product together with the customer and product owner according to the requirements and needs of the users (Release Testing). All project team members were responsible for validation activities.

We did not use any automated testing tools for the testing of Coda since Coda requires Hand Gestures and a phone camera. We tried to simulate the phone camera with a pre-recorded video in the emulator however realized it was not possible to do so. Instead, we tested on the Android phones, but since not all of the team members owned an Android phone or a VR glass, we assigned one of the pairs during pair programming to perform the testing for the pair. This method of testing was followed throughout the whole project.

Following activities were be performed for the validation of Coda,

5.2.2.1 Development Testing

Development testing is where the system will be tested for finding defects by the development team.

- **Unit Testing:** Individual units (classes) of the system were manually tested to check the functionality of objects or methods.
- **Integration Testing:** Individual units tested during unit testings were integrated this time to create components to check the component interfaces and tested manually.
- **System Testing:** Several or all components are integrated to test the complete system to check the interactions between components and tested manually.

5.2.2.2 Release Testing

Complete version of the system was tested before the demo.

- **Requirements Testing:** We manually tested the system with test cases specifically designed according to the requirements.
- **Scenario Testing:** We manually tested the system with test scenarios specifically designed according to the user stories to check if the main use cases for the user are performed correctly since we developed with Agile methodology.
- **Performance Testing:** Performance is an important feature of Coda since it has to be real time for useability. So once the system is completely integrated, the system will be tested for performance and reliability.

5.2.2.3 User Testing

Users test the system in their own environment, the demo.

- **Acceptance Testing:** This will be done with the supervisor and the juries to check whether the system is ready to be accepted to be deployed and demoed to the customers in the CSFair.

6 Maintenance Plan and Details

Maintenance Plan of Coda is designed according to *14764–2006 — ISO/IEC/IEEE International Standard for Software Engineering — Software Life Cycle Processes — Maintenance* [7]. According to this standard, the maintenance plan of Coda will be analysed in 4 categories.

However since the development team of Coda will no longer reside in the same city, maintenance will be actively pursued only until CSFair 2020 and after CSFair, the team will no longer reside in the same city and will work completely remotely and with different schedules which will be a bottleneck for the maintenance of Coda. So the team will continue the maintenance of Coda passively and slowly whenever there will be an idea or a request to do so.

6.1. Corrective Maintenance

After acceptance testing and first deployment if there will be bugs found by the users and the customers, development team of Coda will make some further implementations to solve the found bugs. This maintenance will continue iteratively as newer versions of Coda will be deployed and tested. Hence this is a continuous maintenance plan since there will be bugs as long as there will be new features and new versions of Coda.

6.2. Preventive Maintenance

After development testings if there will be defects found by the testers or edge scenarios that lead to unexpected behavior of the system, the development team of Coda will make some further implementations to solve the found defects and prevent them to be found by the users after deployment. This maintenance will continue iteratively as newer versions of Coda will be tested before deployment since the focus is to prevent bugs.

6.3. Adaptive Maintenance

Even if all the defects and bugs were found and fixed after the deployments and during the tests, there will still be a need for maintenance as a result of the changing environments and libraries used since change is always a part of softwares. There will surely be updates to the Android OS, to the libraries used like Manomotion, Google VR and Unity which will require updating the impacted components of Coda to make it bugless/defectless one again. So, following the updates to the components and the environments of Coda, the development team will update the impacted parts of the code as required.

6.4. Perfective Maintenance

After deploying the first version of Coda, the team will work on the other promised features of Coda and start optimizing the implementations in Coda. Since the first version of Coda was a strictly time-boxed process, perfective maintenance will be very important for perfecting the product and for delivering what was visioned at the beginning.

7 Other Project Elements

7.1. Consideration of Various Factors

A table explaining the effects of various factors on the design phase can be found below as also shown in the Analysis Report.

	Effect level	Effect
Public health	8/10	Considering the duration limitation of using AR applications, the amount of time a user can continuously spend using the application is limited to approximately 20 minutes to avoid possible occurrences of dizziness, headaches etc. This condition limits the amount of time a user can practice an instrument using the app.
Public safety	6/10	Coda is an AR application that partially limits the hearing and the vision of the user. Visual content will be rendered upon the real world view of the user to avoid isolating the user completely from the real world, but there might be delays in rendering that may result in accidents caused by limited sight or hearing. But this does not have any specific limitations on the design or the usability of the product.
Public information security	7/10	Coda is an application that requires neither a user profile nor getting any information from the user. Hence, there will be no possible security gaps or additional information claims that require getting the consent of the user. We fully respect the privacy of the user in our design.
Public welfare	5/10	Coda is a non-profit application which neither provides nor demands for any kind of fees. It is important for us to offer this application to users from all kinds of economical levels without any economical drawbacks. Hence, Coda will not impose and economical insecurities to it's users.
Global factors	3/10	Coda aims to access people from anywhere hence the application will be designed in such a way that any user will be able to use the application in any way they wish to. Nonetheless, the current design goals of Coda does not aim to connect users but rather aims to create a more individualistic experience. Hence, the design phase is not highly affected by global factors.
Cultural factors	8/10	The essential value of Coda is making music which is a fundamental structure of Culture. The design will be made such that the variety instruments offered in Coda will be extendible after the demo. Cultural factors will be considered in implementing instruments, and the team will place emphasis on ethnic instruments.
Social factors	10/10	Idea of Coda debouched from increasing accessibility of instruments, therefore social factors are critical for all kind of design phases. . For that reason, the idea design was made such that Coda will be in full service of the society and increase the mobility and the accessibility of instruments. Design in terms of

Table 1: Consideration of Various Factors Table

7.2. Ethics and Professional Responsibilities

As engineers, we believe that our first and foremost responsibility is to increase the availability of products and opportunities of every context for the benefit of the people while following proper ethical values.

Ethics: First of all, we acknowledged during our analysis phase that Coda is responsible from respecting the copyright laws for the musical content. We investigated the extent to which we can use different types of musical data and did our planning accordingly. In other words, Coda protects the rights of the music owners and does not use any unauthorized music in the application.

Environmental Responsibilities: In the environmental context, Coda recommends using recyclable VR glasses like Google Cardboard which are mostly made out of paper. No additional waste is created by using Coda and Coda is believed to decrease the amount of waste by eliminating the need for physical instruments and hence the material that the instruments they are made of, which is mostly plastic.

Economical Responsibilities: As explained in the former parts of this report, Coda stemmed from the idea of creating a mobile, accessible and cheap way of playing and learning instruments. For that reason, Coda acknowledges its responsibilities to aid the society to enhance musical education without seeking any profit. Coda requires no more than a mobile phone with a camera in order to be functional. For increasing accessibility in terms of economical constraints, we require no extra hardware or no advanced hardware. With this approach, we also provide accessibility to instruments without the cost of purchasing real instruments and the increase the mobility of instruments by keeping them in virtual level only.

Societal Responsibilities: Coda aims to serve people from all kind of societal and economical levels. Coda is especially suitable for being used in schools for musical education. Schools in the countryside in which students do not have the privilege to purchase real musical instruments, can use our project to provide musical education. Moreover, everyone with the application will have access to music instruments even they do not own any. People will able try and choose between the music instruments to their liking and maybe get further education for it. Hence, playing a music instrument can be more mainstream and music education can be supported in the society. Social impacts of arts argued to improve the quality of life and welfare. We are hoping to shape our community and improve lives through the power of art and engineering.

7.3. Judgements and Impacts to Various Contexts

As engineers, we followed the responsibilities in various contexts we recognized and employed since the Analysis report during the implementation and project management of Coda.

Judgement Description:	Minimizing extra hardware for the development of Coda while providing the best possible experience.	
	Impact Level	**Impact Description**
Impact in Global Context	Low	This version of Coda does not have high impact in Global Context, however if Coda is extended to work with several users to remotely make music, it would have a high impact for remote musicians.
Impact in Economic Context	High	Coda has a high impact in Economic Context because it reduces the price of trying & playing an instrument dramatically without the need for a complex hardware ot mobile phone.
Impact in Environmental Context	High	Coda has a high impact in Environmental Context because it reduces the material waste and required hardware of playing an instrument dramatically.
Impact in Societal Context	Medium	Coda has a medium impact in Societal Context because it eliminates the economic constraints of playing and purchasing an instrument, making it much more accessible for everyone. Furthermore, since Code requires no physical effort to play an instrument whereas some instruments do, people with disabilities or disadvantages may also use Coda.

Table 2: Judgement #1 Table

Judgement Description:	Not requiring a phone with advanced technologies like depth-camera and using an API that makes cloud computing for exhaustive calculations.	
	Impact Level	**Impact Description**
Impact in Global Context	Medium	Since the judgement made for Coda makes Coda more accessible, Coda addresses to a wide range of users worldwide.
Impact in Economic Context	High	Coda has a high impact in Economic Context because it reduces the price the mobile phone needed for using Coda and makes Coda, therefore playing instruments much more accessible.
Impact in Environmental Context	Low	This judgement of Coda does not have an environmental impact.
Impact in Societal Context	High	Coda has a high impact in Societal Context because it eliminates the economic constraints of requiring an advanced mobile phone to run the application on.

Table 3: Judgement #2 Table

7.4. Teamwork and Peer Contribution

Teamwork in Coda was a very successful aspect of the whole project. Following tools were used to manage the development of project Coda,

- For managing the project plan and ensuring teamwork we organized online meetings via **Zoom** at least every week.
- We used **Trello** boards to keep track of Issues and Tasks to be implemented as well as tracking the assignees.
- **GitHub** was used for collaboration and version management on the project.
- **Slack** was used to communicate asynchronously about the project. & effort consuming to do so.

The whole team of Coda worked together throughout the project and there weren't any members who disrupted the project flow. Even when Corona hit and working on a project that required hand gestures, Android phones and VR glasses was hard, the team managed to work together effectively although it was much more challenging and time.

During Corona, it was hard to keep up with the overlapping work and understand what each member was doing, so we increased the frequency of meetings and employed an Agile approach for planning each development increment and made several daily scrum meetings during the day as well as peer programming.

Tasks and their implementers can be found in the *Table 4*,

Report & Documentation Work	● All team members participated actively while all the reports were being written. The team started working in the summer to come up with the project idea, and the team continued to worked together to come up with the ideas, requirements and design needed for this report. Then the work was distributed evenly to the team members to draw the diagrams or to document them in the reports. The workload was unbalanced only when a part of the team would better work on the implementation than on the reports. This method was accepted by all the team members and was often employed.
MediaPipe	● MediaPipe was the first library tried for Hand Tracking. Cagla and Ege tried the library functionalities on its own to check the performance of hand tracking. ● Ege, Yağız and Murat worked on MediaPipe only approximately for 2 whole months because Unity was chosen as the development platform but MediaPipe was not yet ported to Unity. This was the a bottleneck task because there were no other alternatives which performed as well as MediaPipe. Although this task was quite overwhelming even for developers of higher levels than us Ege, Yağız and Murat made an incredible and steel effort to get MediaPipe working in Unity, solved a lot of problems but we figured that we still couldn't foresee how much effort would be needed to successfully use MediaPipe in Unity. After contacting numerous field experts, other developers trying to do the same thing we finally decided to finally switch to a worse performing library for the sake of implementability. ● Çağla and Merve tried to work on attaching the hand model to the data expected from MediaPipe however since we could not manage to port MediaPipe to Unity, this part was never completed.
Manomotion	● Although Manomotion was one of the libraries reviewed at the beginning of the implementation, it was initially cancelled because it required internet connection and it's lite version only detected the palm center unlike MediaPipe.

	Regardless of these constraints, we sent a request to Manomotion for their Lite SDK which was free. However we received no answer for 3 months. Then as we decided to give up on MediaPipe, we contacted employees of Manomotion to get the Lite SDK, and we finally received the Lite SDK and developed the final project as well as we could within the limited time with the limited functionality offered by Manomotion. All team members worked with Manomotion. Ege, Yağız and Murat worked on Manomotion to implement the collision detection and attaching the hand model to the data obtained.
Instrument Models & Sounds	• Cagla and Merve worked on the implementation of the instrument models and managing the sounds in a case of collision in the application. This part was simple as soon as the collision was detected by the Manomotion part.
Google VR	• All team members contributed to using Google VR library in Unity as the AR library. Google VR was simple to use for simple tasks however since using VR glasses for AR applications is not a popular idea yet, we spent some time on figuring out how to display the application in Stereoview. Also, we realised that rendering the camera as the background of the application was very memory consuming for the mobile phone, so Yağız spent a considerable amount of time to figure out how we could cancel the camera rendering for the background while still detecting the hands form the camera.
VR App	• All the team members worked on bringing all the components together such that it would build a complete VR application built for Android with Unity.
Vuforia	• For the AR library Yağız and Murat found Vuforia which successfully implemented interactable AR objects. However the library had some constraints like using an A4 paper underneath the AR object to locate the object. However this constraint made Vuforia useless for Coda because we wanted to use real life objects.

Table 4: Tasks Table

Proof for the contribution of the team members can be found in the repository we used for the development of Coda,

https://github.com/ege0zcan/coda

However, it should be noted that we did a lot of peer programming during Coda using various methods like coding together physically before Corona and coding while talking on Discord and while coding together on Visual Studio Live Sharing. So the commit numbers do not represent the exact amount of effort. The team worked on more than 5 other projects together while working on Coda and we're proud to say that we handled all of them without any quarrels and ended up with successful projects.

7.5. Project Plan Observed and Objectives Met

List of initially designed work packages and the assignees as specified in the Analysis Report can be found below.

WP#	Work package title	Leader	Members involved
WP1	App Structure	Merve Kılıçarslan	Yağız Efe Mertol
WP2	Hand Gesture Recognition	Çağla Sözen	Ege Özcan, Murat Tüver
WP3	AR Modeling	Yağız Efe Mertol	Çağla Sözen, Murat Tüver
WP4	Instruments Graphics Modeling	Ege Özcan	Yağız Efe Mertol, Çağla Sözen
WP5	Auditory Feedback	Murat Tüver	Yağız Efe Mertol, Çağla Sözen
WP6	Hand Gesture Recognition - Auditory Feedback Integration	Murat Tüver	Ege Özcan, Çağla Sözen
WP7	Hand Gesture Recognition - Instrument Graphics Integration	Çağla Sözen	Merve Kılıçarslan, Murat Tüver
WP8	*Instrument Graphics - AR Rendering Integration*	Yağız Efe Mertol	Merve Kılıçarslan, Ege Özcan
WP9	Demo for CS Fair	Çağla Sözen	Merve Kılıçarslan, Murat Tüver
WP10	Final Presentation	Yağız Efe Mertol	Ege Özcan, Çağla Sözen
WP11	Final Report	Çağla Sözen	Merve Kılıçarslan, Murat Tüver, Ege Özcan, Yağız Efe Mertol
WP12	High Level Design	Çağla Sözen	Merve Kılıçarslan, Murat Tüver, Ege Özcan, Yağız Efe Mertol
WP13	Low Level Design	Yağız Efe Mertol	Merve Kılıçarslan, Murat Tüver, Ege Özcan, Çağla Sözen

Table 5: Work Packages in the Analysis Report

Observed work plan and work packages can be found below along with the milestones, dates and assignees. As observed when the two tables are compared, the work packages and work plan changed as a result of the failed usage of MediaPipe and the time consumed by working on MediaPipe because it had no alternatives performing as well as it did.

WP#	Work Package Title	Leader	Members Involved	Objective	Dates & Duration
WP#1	VR App	Yağız Efe Mertol	All	Building the structure of a VR app to render on VR glasses correctly.	15.01.2020 - 25.01.2020 10 days
WP#2	Hand Tracking & Collision Detection	Murat Tüver	Ege Özcan, Yağız Efe Mertol	Trying to port MediaPipe to Unity tool almost 2.5 months. Then trying to get the Manomotion SDK took 2 months in parallel. As soon as we obtained the Manomotion SDK, we developed the Hand Tracking in 1 month.	15.01.2020 - 1.05.2020 3.5 months
WP#3	Instrument Model	Merve Kılıçarslan	Çağla Sözen	Obtaining a usable drum model in Unity with collisions.	25.01.2020 - 31.01.2020 6 days
WP#4	Hand Model	Çağla Sözen	Merve Kılıçarslan	Obtaining a workable hand model in Unity with correct mapping to MediaPipe or Manomotion. Work in this part was majorly for MediaPipe, since we ended up not using MediaPipe, it was also cancelled.	25.01.2020 - 01.04.2020 2 months
WP#5	Sound Manager	Çağla Sözen	Merve Kılıçarslan	Obtaining the instrument sounds and building the sound manager in Unity to make the correct sounds when collided.	15.4.2020 - 1.05.2020 15 days
Milestone #1	Basic Functionality Ready				
WP#6	App Structure	Ege Özcan	Yağız Efe Mertol, Murat Tüver	Getting the menus and canvases in unity to work similar to a normal android app with hand gesture controls.	1.05.2020-27.05.2020
WP#7	Other Features	Çağla Sözen	All	Implementing the extra features promised as much as time and Corona permits.	1.05.2020 - Now
Milestone #2	Project Complete				
WP#8	CSFair Demo	Ege Özcan	All	Preparing and planning the Demo for the Online CSFair.	27.05.2020-15.06.2020
WP#9	Final Presentation	Yağız Efe Mertol	All	Preparing and planning for the Online Presentation.	25.05.2020-27.05.2020
WP#10	Final Report	Çağla Sözen	All	Writing the Final Report according to the Final Project version.	25.05.2020-27.05.2020
WP#11	Low Level Design Report	Merve Kılıçarslan	All	Writing the Low Level Design Report	01.02.2020-17.02.2020
WP#12	High Level Design Report	Merve Kılıçarslan	All	Writing the High Level Design Report	21.12.2019-31.12.2019
WP#13	Analysis Report	Çağla Sözen	All	Writing the Analysis Report	01.11.2019-11.11.2019
WP#14	Project Specifications Report	Çağla Sözen	All	Writing the Project Specifications Report	01.10.2019-14.10.2019

Table 6: Final Work Packages

7.6. New Knowledge Acquired and Learning Strategies Used

7.6.1. Unity

Few of the team members were knowledgeable about Unity and the architecture used in Unity. For learning Unity, we shared our knowledge while coding and making the design decisions according to the design and principles of Unity. As the learning strategies used for learning Unity, we watched Unity tutorials on Youtube and read the official documentation of Unity. Also, during pair programming we paired the members with knowledge of Unity with members without the knowledge of Unity to share the knowledge among the team mates and to make good design decisions.

7.6.2. Hand Tracking

For Hand Tracking, we did not have to learn about the inner implementations of the Computer Vision models used in the Hand Tracking Libraries. However since Hand Tracking is a technology that is very novel, learning to use the libraries required to use them in the environments used for the project was challenging enough with the sparse documentation. Two libraries we had to learn were MediaPipe of Google and Manomotion Lite SDK.

To learn MediaPipe, we read the the documentation, read the code, the comments in the code and used the examples written for demonstration purposes by Google. We also read a lot of issues in the GitHub Issues of Mediapipe. We contacted the MediaPipe developers, other developers working on MediaPipe with Unity and used StackOverflow to solve our sub-problems. To learn Manomotion, we used the online documentation and the discussion forum which were the only source of information on Manomotion since it is available in a limited environment.

7.6.3. VR

For VR we did not have to learn about the inner implementations of the Virtual Reality libraries. However since Augmented Reality is a technology that is very novel, it's not used with VR glasses which was what we designed initially. Although we used Stack Overflow to modify the library used for AR, which was Google VR library and the issues in GitHub issues in the repository of Google VR, we realized that the performance decreased significantly when we rendered the camera as the background.

To learn Google VR, we read the the documentation, read the code, the comments in the code and used the examples written for demonstration purposes by Google. We also read a lot of issues in the GitHub Issues of Google VR.

8 Conclusion and Future Work

This version of Coda will be deployed as the final version of Coda in the context of CS491/492. The current system is limited to providing the basic functionalities of the system which are essential for the mission of Coda. Since this project was a non-commercial project we made the design decisions accordingly and also made decisions according to the time constraints. As a result of these constraints, we made a lot of on-the-fly design and requirement changes to be able to deliver a demoable product in time.

The current system supports only one of the modes presented in the Analysis document of Coda, which is the Free Mode with a single instrument being the Drums. However considering Free Mode was estimated to consume the highest amount of time, we believe that this version of Coda can be easily extended with additional time

and additional resources which a commercial project would have. An example of the constraints other than the course schedule would being limited to the non-commercial (free) versions of the SDK's used during development. Furthermore, the application is restricted to using a single hand since once again, the Pro SDK of Manomotion is strictly used by commercial applications and the Lite SDK does not support hand tracking with two hands. However we believe that these constraints could be easily overcome with a larger and more experienced team with commercial level resources and schedule.

As a final remark, we realized that using multiple very novel technologies in an application increases the risk factors highly and requires the development process to be very flexible in terms of requirements and schedule. We acknowledged that a project that requires so many novel technologies would need to wait a little more such that these technologies would reach a certain maturity level to be able to integrate them with each other and modify according to our needs by reading the mature and complete documentation in addition to the existence of sufficient examples.

If the team of Coda would wish to extend the current version of Coda, it should be known that the development team of Coda will no longer reside in the same city, so future work for Coda will be actively pursued only until CSFair 2020 and after CSFair, the team will work completely remotely and with different schedules which will be a bottleneck for the future work on Coda. So the team will continue the future work on Coda passively and slowly whenever there will be an idea or a request to do so.

As final remarks, we believe that this has been a valuable experience for us to try Agile-Waterfall development in a tight schedule with extensive need for documentation in addition to the course work. Although the final system has some constraints as explained above, we are content with the version of Coda that we are delivering which we believe that it delivers the basic idea of the project. It's been a journey for us to learn new technologies, overcome difficulties, experience a real life like project with different stakeholders and most significantly experience effective teamwork. We are proud to say that the team of Coda worked effectively throughout the two semesters, completed more than 5 different projects together and succeeded in all of them with good teamwork without having any problems in the team.

9 Glossary

AR : Augmented Reality. A computer enhanced version of the real-world where virtual objects can be put on real world objects in an interactive environment.

VR : Virtual Reality. A computer generated world that is completely virtual in terms of environment and the objects in it which provides a virtual interactive environment to the user.

MR : Mixed Reality. A computer generated world that is a mix of the real world with virtual components on top of the virtual world.

CV : Computer Vision. Combining/Using advanced Image Processing, Machine Learning and Deep Learning techniques to enable computers to see as a human does.

AAR : Android Archive Library. These may contain Android resources and a manifest file, which allows you to bundle in shared resources like layouts and drawables in addition to Java classes and methods [8].

10 References

[1] Smith. B, “*New study demonstrates link between music and statistical learning*,” The Sydney Morning Herald, 2019. [Online]. Available: https://www.smh.com.au/technology/new-study-demonstrates-link-between-music-and-statistical-learning-20170514-gw4eec.html. [Accessed: Oct. 10, 2019].

[2] ABRSM, *ABRSM:*. [Online]. Available: https://es.abrsm.org/en/making-music/4-the-statistics/. [Accessed: Oct. 10, 2019].

[3] Hepsiburada.com. “*Vr Google Cardboard Vr 3D Sanal Gerçeklik Gözlüğü Fiyatı*,” 2019. [Online]. Available: https://www.hepsiburada.com. [Accessed: Oct. 10, 2019].

[4] Nspe, "*Code of Ethics | National Society of Professional Engineers*", Nspe.org, 2018.
[Online]. Available: https://www.nspe.org/resources/ethics/code-ethics.[Accessed: Oct. 11, 2019].

[5] Bilkent CS. Department, “CS491/2 - Senior Design Project I/II,” *Bilkent CS. Department,* 2019. [Online]. Available: http://www.cs.bilkent.edu.tr/CS491-2/current/. [Accessed: Oct. 10, 2019].

[6] Business Insider, “*Here's what happens to your body when you've been in virtual reality for too long*,” *Business Insider,* 2017. [Online]. Available: https://www.businessinsider.com/virtual-reality-vr-side-effects-2018-3. [Accessed: Oct. 10, 2019].

[7] IEEE Std 14764-2006 - ISO/IEC/IEEE International Standard for Software Engineering - Software Life Cycle Processes - Maintenance, 2006.

[8] "Create an Android library | Android Developers", *Android Developers*, 2020. [Online]. Available: https://developer.android.com/studio/projects/android-library. [Accessed: 26- May- 2020].