Bilkent University

Department of Computer Engineering

Senior Design Project

Microgliss: a microtonal editing tool for world music

High-Level Design Report

Team Members: Artun Cura, Sonat Uzun, Orkan Öztrak

Supervisor: Vis. Prof. Dr. Fazlı Can

Jury Members: Assoc. Prof. Dr. Çiğdem Gündüz Demir, Asst. Prof. Dr. Can Alkan

Innovation Expert: Burcu Coşkun Şengül

High-Level Design Report
Dec 27, 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

1.Introduction

Microgliss is the new cutting edge note editor and synthesizer that allows you to write microtonal music and polyphonic glissando using an intuitive node and edge-based workflow. It will allow users to write music in any tuning system or independent of any tuning system. It will allow users to add glissandos between any note and edit these glissandos with further settings. Microgliss is designed keeping in mind the growing interest in microtonal music around the globe and the common limitations of existing midi/note editors and synthesizers.

Using midi editors is a common method for composing computer music. But midi editors have limitations that can't be solved or only can be solved with non-elegant workarounds. Some of these limitations are 128 different values for velocity, 128 different note values (pitch), once a note's value and velocity are set they can not be changed until that note is released, enforcement of a grid system for pitch values (as a result of 128 note limitation), no glissandos for multiple notes, etc.

World music is not constructed on a grid. Every culture has its own unique sounds, pitches, and scales. But today's computer systems make the production of world music(non-western music) harder for the users because all of the systems are constructed around western culture.

1.1 Purpose of the system

Microgliss is a note editor that solves the mentioned problems in the previous section. Our paradigm is a workflow consisting of nodes and edges, which corresponds to notes and glissandos. In microgliss using a grid will be a preference of the user. Every frequency will be available for their use and each note(each node) might be connected with another node. This lets users create a seamless movement between them, and start a new age for computer music.

Briefly, the purpose of the Microgliss is to remove the limitations of the western-based computer music system and let users explore a fresh interface that embraces world music. Also developing this product as a plugin will let users use this within their current Digital Audio Workstation. So no major change will be required for the users' music production environment. This means users will be more willing to combine Microgliss with their existing workflows.

1.2 Design goals

1.2.1 Compatibility

Microgliss should run on all DAW's which support the VST format. Users should be able to import MIDI files generated from different applications or read the data directly from DAW.

Microgliss should be able to emit OSC messages and which can be interpreted by a different synthesizer.

1.2.2 Extendibility

Editor and the synthesizer part should be able to connect via OSC protocol and they should be separable programs. This will enable us to extend the ecosystem of our product in the future (it can work with other synthesizers built by us or others).

1.2.3 Responsiveness

The editor should be able to communicate the changes made in it to the synthesizer, in real-time, and during playback. This aids in the workflows of the users.

1.3 Definitions, acronyms and abbreviations

Microtonal Music: Music that consists of intervals that are smaller than a semitone. This term also includes any tuning system which differs from the western 12-tone tradition.

Synthesizer: An electronic musical instrument that generates audio signals.
Glissando: Sliding from one note to another seamlessly.

DAW: Digital Audio Workstation. Comprehensive musical programs that are designed for recording, editing, and producing music.

Automation: Automation in the context of DAW's is lines usually drawn by hand to change parameters of a track (eg: volume/pan) and/or a plugin (eg: mix) over time.

VST: A plug-in format for a digital audio workstation.

OSC: Open Sound Control. A protocol for networking sound synthesizers, computers, and other multimedia devices for purposes such as musical performance.

JUCE: An open-source C++ Framework that is used for the development of desktop and mobile applications for GUI and Audio tools.

Trello: A basic project management tool. Its main use is for Agile project development but we will also use Trello with the Iterative approach.

Note: A symbol denoting a musical sound.

MIDI: Musical Instrument Digital Interface. A communication protocol for musical instruments and controller devices.

Preset: Pre-saved synthesizer settings. They can also be produced by different professional sound-designers for end-user use.

Sample: A sample, which is a floating point value, is the smallest component of the digital representation of the sound. A one second sound stored in a computer consists of over 40000 samples (usually 44100 or 48000). These samples are written into buffers in and speakers vibrate according to the data in the buffers, therefor generating sound. Each sample represents subtle changes in air pressure which we perceive as sound.

1.4 Overview

Microgliss is a note editor and synthesizer that allows you to write microtonal music and polyphonic glissandos using a node and edge-based workflow. A workflow consisting of nodes and edges, which corresponds to notes and glissandos, isn't common in other note editors, so the editing capabilities of the editor are beyond any existing tool.

We plan to export our application in VST (Virtual Studio Technology) format thus allowing it to run on any DAW (Digital Audio Workstation) as a plugin. It will have two separate sections and views that correspond to them. It will have a note editor to compose by inputting notes and glissandos and a synthesizer to play them. The editor should encompass the entire timeline of a project.

In addition to the synthesizer section of microgliss, we want our program to work with other synthesizers by using the Open Sound Control protocol, to extend the use and let all users work with their favorite synthesizers. OSC also has other uses such as light control and microgliss can be used for such side-purposes.

The next section of this high-level design report explains the details of the system in terms of Software Architecture including subheaders like Subsystem Decomposition, Hardware and Software Mapping, Data Management, Access Control Policies, Security, Global Control Flow, Boundary Conditions, and Subsystem Services. We expect our design to fulfill the requirements which we discussed in the Analysis report and conclude with a useful program that lets any type of musicians to get in the microtonal world, experiment, and compose easily. In the other section Consideration of Various factors in Engineering design, we will discuss whether if Microgliss contains any risks and in the last section Teamwork Details we will discuss the details of our team under the Contributing and functioning effectively on the team, Helping to create a collaborative and inclusive environment, Taking lead role and sharing leadership on the team headers.

2. Current software architecture

To compensate for the limitations imposed by the MIDI protocol that was established in 1981, MIDI instruments have many features that allow additional control over the sound. For example, ADSR (Attack, Decay, Sustain, Release) envelope and LFO (Low-Frequency Oscillators) can be used to modify the pitch and velocity value of a note through its lifetime. There are also MIDI editors that have tried to break the limitations of traditional MIDI editors, such as the Bitwig DAW, in which it is possible to put in off-grid nodes and polyphonic glissandos and play them with instruments that are native to Bitwig.

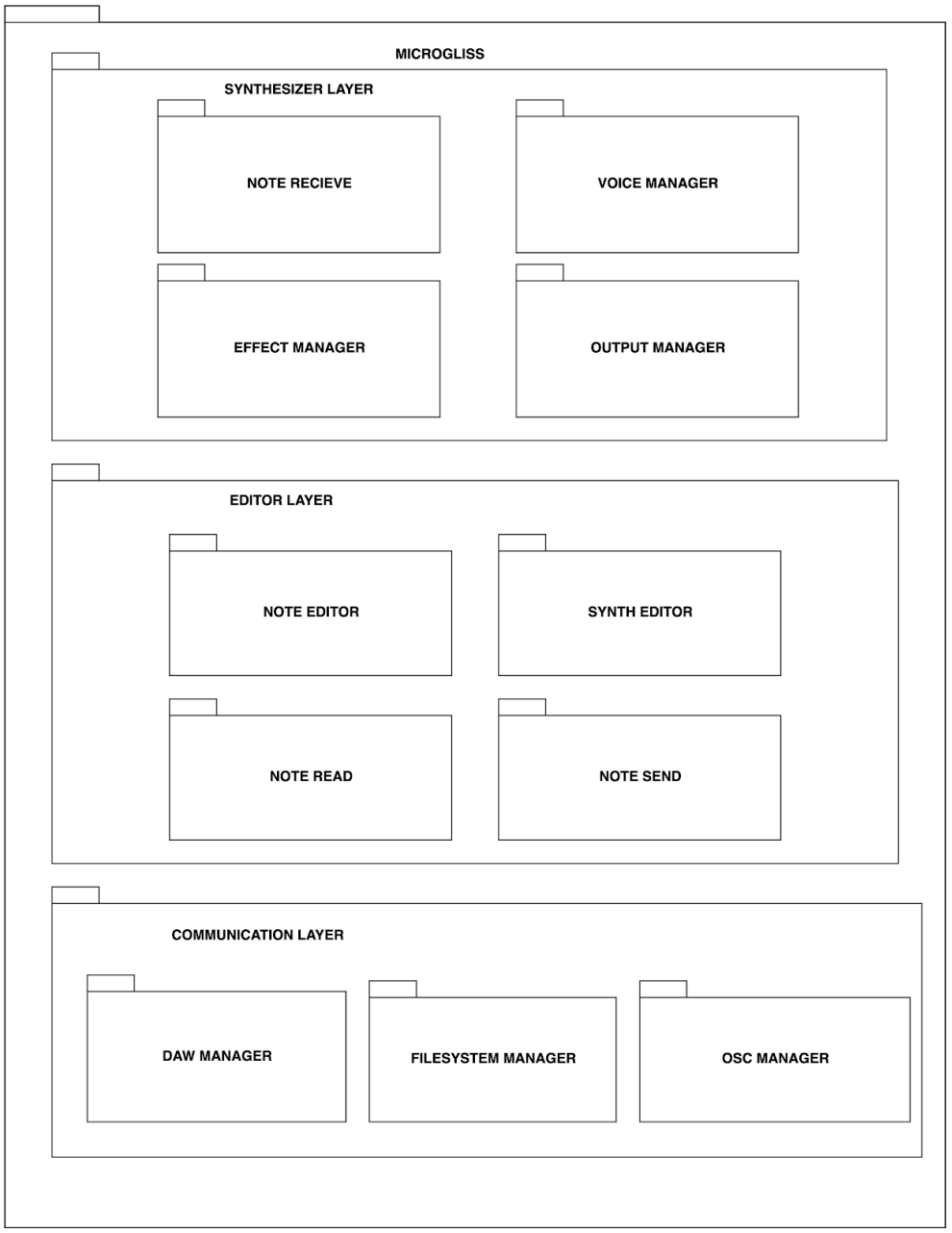
And Melodyne, an editor plugin which is used to alter individual notes in recordings. However, the use cases and flexibilities offered by these programs are different than those of Microgliss. Bitwigs editor only works on Bitwig DAW. And Melodyne alters pre-recorded sound instead of creating sounds from scratch. Also, as mentioned before, nodes and edges in the workflow are uncommon, giving Microgliss much freedom in its editing. Therefore, Microgliss will not rely on existing software architecture but it might occasionally derive ideas from them.

3. Proposed software architecture

3.1 Overview

This section will display the proposed architecture via the subsystem decomposition, then explain every layer individually for better understanding. Management of persistent data, security, control of access, and global software, in addition to boundary conditions, will also be discussed in detail. Further information regarding Microgliss can be found here: https://microgliss.github.io

3.2 Subsystem decomposition



3.3 Hardware/software mapping

Microgliss must run on any DAW that supports the VST format. Users should be able to import MIDI files generated from different applications or read the data directly from DAW. Because it is a plugin, Microgliss will need no hardware mapping other than the mappings of the DAW it runs on. DAW's are usually programs that run on single

For the software components of Microgliss:

- C++ will be used because most programming languages aren't suitable for basic audio functionalities and VST generation support.
- JUCE will be used in the implementation of an audio/UI generation framework based on C++.

- A new audio programming language SOUL might be tried for some portion of the project but we are aware that it is new, and might have some inconsistencies. If we encounter any problems this part will be implemented with C++/JUCE too.

3.4 Persistent data management

Two different kinds of data must persist in the system: parameters of the synthesizers and compositions made in the note editor. DAW's provide interfaces to make these data persistent across sessions/saves. If we decide to build a standalone build we might need to provide our own file format to be able to save and load projects. In addition to making the data persistent, making it reusable is also a concern in audio production applications. Suppose that by tweaking the settings of the synthesizers you have come across a sound that you liked a lot and want to be able to use it across multiple projects. Most synthesizers provide an option to save parameters of the sound as a preset and easily load it in other instances of the synthesizer. Microgliss will also have an option to save and load synthesizer parameters as presets. We might also include default/factory presets in order give starting points for users to create their sounds.

3.5 Access control and security

In a single-user system, there is only one human actor. Microgliss is a single-user system. But also there is also a System actor that regulates the operations between the user and the DAW. This is why the actors which have different rights over the data and functionality are the system itself and the current user for Microgliss. These two need to be separated because a user can directly move a previously saved preset to the presets folder (presets are pre-programmed synthesizer settings) and read it but can't write unless they use the microgliss' interface. Also, the system might use DAW's provided interface for saving and reading, but the users are not allowed to reach and manipulate it.

Each user the data in their program (node editor and synthesizer settings) should be secure and available for the user itself. We applied the Group Based Access Control Scheme for the access rights of the project. We only have two actors:

- System = {Note editor, synthesizer}
- Users = {Anyone who uses the program within their DAW}

The table below (Table 3.5.1) is an Access Control Matrix that specifies which actors have access to which files. In this table R denotes read permission, W denotes write permission.

	Synthesizer Parameters	Composition in the note editor
System	RW-	RW-
Users	W-	---

Table 3.5.1 - access control and security matrix

Users trigger the required reading and writing operations through the system by clicking the necessary buttons, images, or letter combinations. The system stands as a broker for the user and the DAW.

All operations done by the user and any data generated on the program will stay in the program. We will not read or keep any user data without explicitly asking for user permission.

3.6 Global software control

The Editor and Synthesizer layer and DAW Manager subsystem of Microgliss is active as long as the program is active. Changes that are made on the editor is played back in the synthesizer to provide feedback to the user. The sound of the synthesizer is also always responsive to the changes made in the Synth Editor subsystem.

As DAW playback starts and pauses this is detected by the DAW Manager subsystem and the interactions of the Synthesizer and Editor layer is modified accordingly.

For many of its operations Microgliss needs to interact with the DAW. For an example when the user makes a change in the program such as creating a node the program itself notifies the host DAW of this change. DAW handles the necessary operations for the change to occur and will later have control over reverting these changes. Overall the program has 2 main modes of operation.

No Playback

Program starts its life-cycle in this state but may exit from or come back to this state depending on the information provided by the DAW manager. In this state changes that are made in the editor are reflected in the synthesizer to provide user feedback. Any note input to the note editor will be played by the synthesizer for a short moment. Any changes to the synthesizer in the synth editor will result in the synthesizer sound changing.

Playback Active

The DAW manager brings the program into this state when playback is started. When the playback is active in addition to the users real time input the composition on the note editor is fed to the synthesizer. Any changes made in the editor is reflected in the synthesizer as well, similar to when there is no playback.

Two other subsystems OSC Manager and File System manager are only active when they are explicitly called from the editor. OSC Manager only gets activated when an OSC connection with another device is set. And File Manager is only active when presets are being loaded or saved.

Overall, the DAW manager has an effect to alter the ways in which the other subsystems interact with each other. Editor layer controls the output of the Synthesizer layer which produces the ultimate result of the system, the sound. The editor layer might also enable and make use of 2 other subsystems file manager and OSC manager, when it’s requested by the user explicitly.

3.7 Boundary conditions

Set-Up

To set-up, the program an installer or user (manually) will install a VST file and default presets into the defaults folder or the user-specified VST folder. Vst folders are the locations

where the DAW will look for plugins. After the folders are rescanned the plugin can be imported to DAW and start running.

Initialization

In the initialization of the program, necessary connections with the DAW, such as requesting output buffers and setting sound outputs, with the operating system, connecting to the file system, and loading preset libraries will be made.

Playback Start

The note editor will send data about notes starts, ends and glissandos to the synthesizer during playback.

Playback End

All the voices that are activated by the synthesizer must be stopped at the end of the playback. Time-based synthesizer effects (such as delay) may continue.

Termination

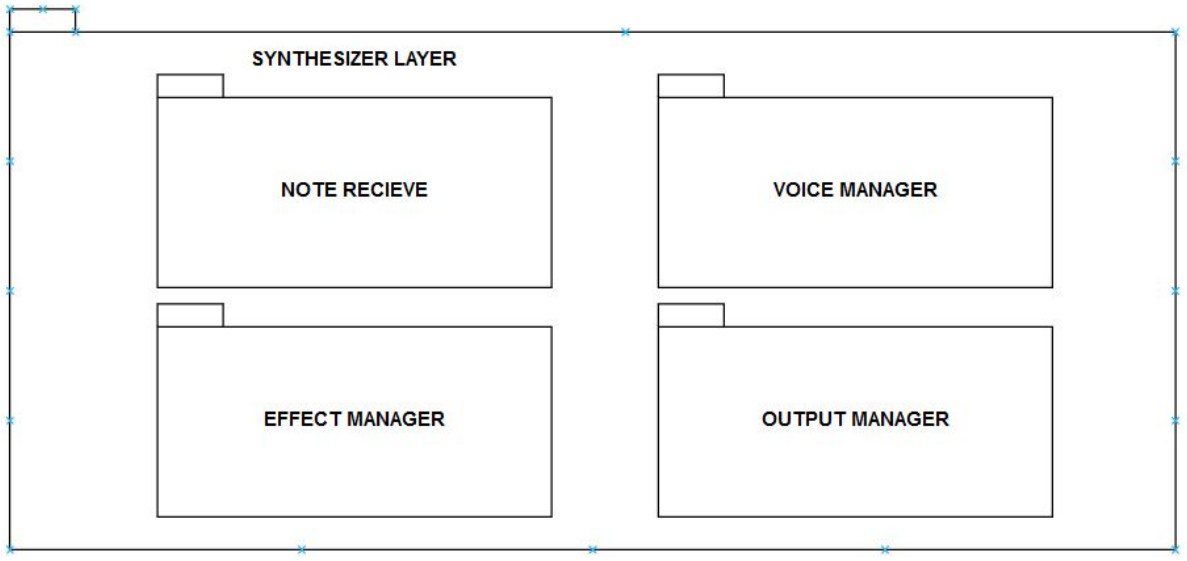
Output buffer contents should be cleared when the plugin is terminated.

File I/O exception

If there is an error while reading files, or if the file is corrupted, any error should be caught. At no point should the program crash because it might mean the loss of important work for the creators.

4. Subsystem services

4.1 Synthesizer Layer



Note Receive

Receives note information from the editor layer and notifies the voice manager about starting, ending, and changing notes.

Voice Manager

Assigns notes to voices (notes to be played and modified by the synthesizer over time) and control the changes in these voices through their lifetime. Managing voice is a specialized process and it involves the task of calculating the value of each sample of the voice. Sample represents speaker movement which corresponds to subtle changes in air pressure that creates sound.

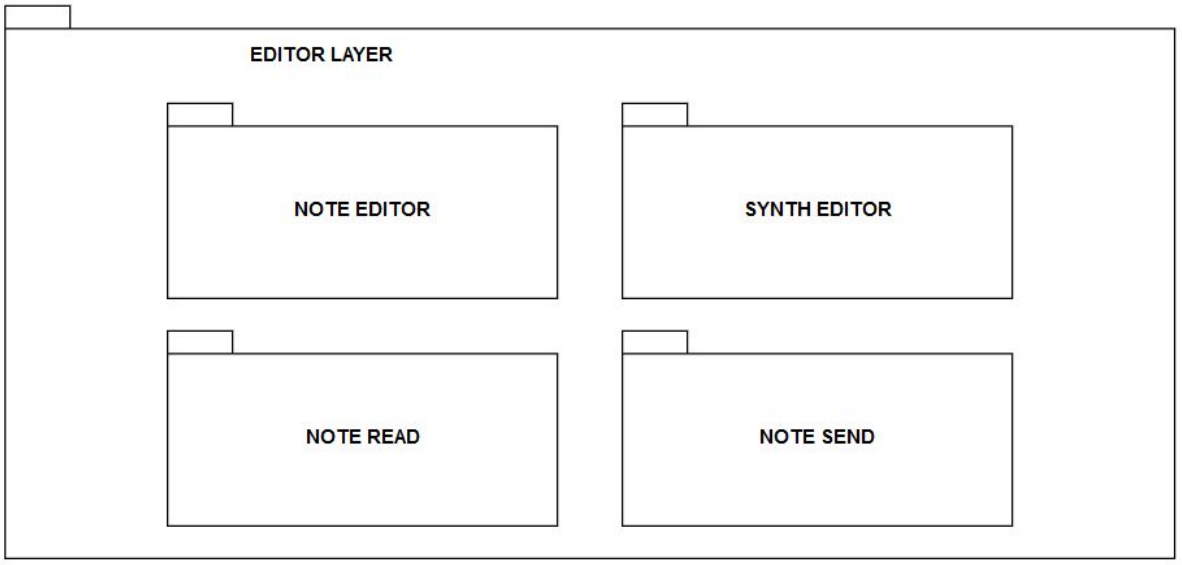
Effect Manager

Effects such as delay and EQ will modify the sound generated by the voice manager for further control over the character of the sound.

Output Manager

Normalizes and finalizes the sound output and sends it to DAW.

4.2 Editor Layer



Note Editor

Let the users compose music by adding, modifying, and removing nodes and edges.

Synth Editor

Edits the parameters of the synth which determines the characteristics of the sound to be produced.

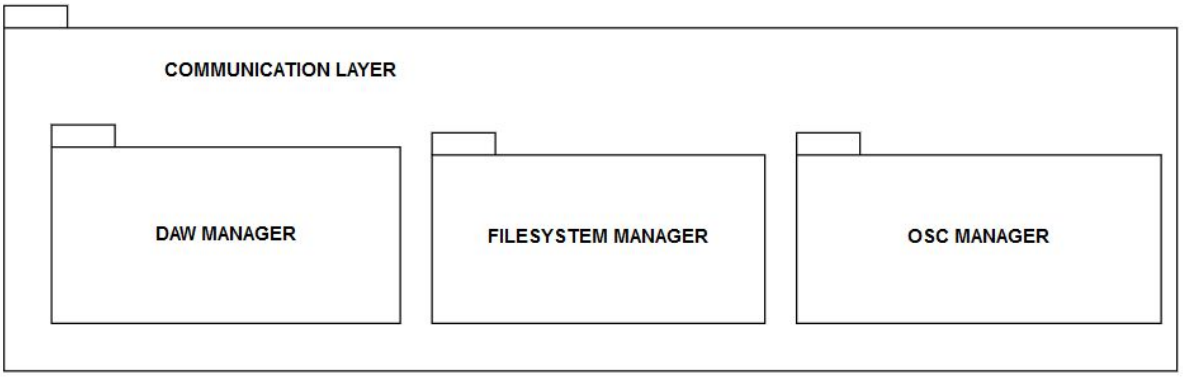
Note Read

Reads midi messages from the DAW and writes these messages into the note editor. Written notes can later be modified to use the full capabilities of Microgliss.

Note Send

Sends note data to Synthesizer, for the note to be played.

4.3 Communication Layer



Daw manager

Manages communication of important information such as playback location, midi-input, and sound output with the Digital Audio Workstation.

Filesystem manager

Communicates with the file system to save and load presets (pre-configured synthesizer parameters).

Osc manager

Establishes connections with other OSC (Open Sound Control) devices such as instruments and synthesizers. Creates and sends OSC messages to control them.

5. Consideration of Various Factors in Engineering Design

For any project, consideration of various factors such as public health, public safety, public welfare, global factors, cultural factors and social factors is really important. Due to the nature of the Microgliss(a music composition program basically) doesn't contain any harm for most of these subjects, but the music that microgliss will be used for mainly will have cultural basis and might have global effect for contemporary music. There are some points we should consider during the process and these points are stated in the table below.

	Effect Level	Effect
Public health	0	No effect
Public safety	0	No effect
Public welfare	0	No effect
Global factors	6	Increase in the accessibility of the microtonal music might lead to the birth of new genres which embrace the microtonality more. Also popular music might start to contain more microtonal elements.
Cultural factors	4	Increase in the accessibility of the microtonal music might cause more embracement of the cultural sounds by the new generation.
Social factors	0	No effect

Table 5.1 - various factors in engineering design table

6. Teamwork Details

Microgliss's development team consists of three Bilkent University Computer Engineering students: Sonat Uzun, Artun Cura, and Orkan Öztrak. We all believe that teamwork is really important for any project because every member is there for a reason and without their contribution, the development process will slow down. This is why our main focus is to keep the project moving. We constructed our plan and responsibility-sharing circularly. Everyone must have an idea about the other parts of the software, even if each of us becomes more involved with different systems. This ensures that if one of us can't work due to some reason others can take care of that part.

6.1 Contributing and functioning effectively on the team

We all believe that contributing to the team and playing our role as a team member is really important. We are a small team so ensuring that each of us is working and active most of the time, we selected some parameters which can show us how we are working through the team. These measures can be listed as:

- # of Github commits
- # of Github Issues Resolved
- # of Trello tasks finished
- Accuracy in regards to meeting Trello deadlines
- Success to implement deliverables and completion of assigned tasks

- Activeness in Whatsapp senior project conversations
- Activeness in Discord talks about the project
- Participation in online meetings
- Participation in real-life meetings (We might not have any due to pandemic conditions)

All of us should have similar numbers for each of these parameters if there is no special condition such as sickness or no specific task assigned. Being a part of the team is important. In an exceptional case, we will try to motivate the lower scored team member because letting someone not doing any work shouldn't be an option.

6.2 Helping to create a collaborative and inclusive environment

To create a collaborative and inclusive environment we divide the task into subtasks and share them at the start of the process. Everyone knows that their part is important for the accomplishment of the project, so everyone tries to improve the project by doing their task well. We always try to keep in touch, check each other, and discuss our problems using mobile communication apps such as Whatsapp and Discord. After we complete our parts we merge them and everybody checks the whole product from start to end and ensures that these smaller parts fit into each other as a whole smoothly and creates one final product.

6.3 Taking lead role and sharing leadership on the team

We believe that having a leader on the team is important because leaders are important for the regulation of the project. Planning, talking with other team members, motivating or warning them if necessary, and ensuring that the project is moving forward each day is their job and this keeps the team alive.

We all know that this leader and team member relationship is not something like a boss and employee relationship. Instead of a hierarchical imposition, our aim for having a leader is to regulate the process. Everyone is equal and has a right to talk. Contributing is essential for this project to improve. For our project we have chosen Artun as the leader, but depending on our schedule leadership can be transferred and shared among each member throughout the project.