Bilkent University

Department of Computer Engineering

Senior Design Project

microgliss: a microtonal editing tool for the world music

Analysis Report

Group 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

Analysis Report
November 21, 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

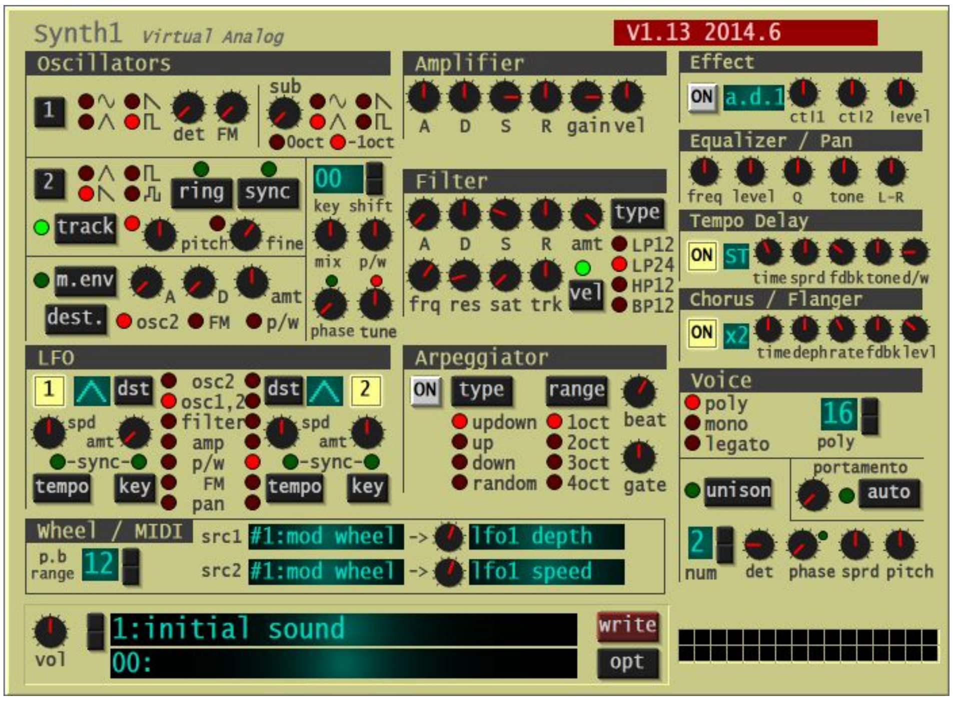
Microgliss is the new cutting edge note editor and synthesizer that allows you to write microtonal music and polyphonic glissando using an intuitive node and edge-based workflow. It will allow users to write music in any tuning system or independent of any tuning system. It will allow users to add glissando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

2. Current System

Using midi editors is a common method for composing computer music. But, midi editors have limitations that are either the product of their roots in traditional sheet music notation or the midi protocol that was established in 1981[1]. These limitations are 128 different values for velocity and 128 different note values (pitch) and no limitation regarding time (rhythm). Also, once a note's value and velocity are set they can not change until that note is released.

This inherent limitation in MIDI protocol can be compensated by the MIDI instruments (such as samplers and synthesizers) that interpret the incoming midi data. 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.

For only monophonic synthesizers (playing only one note at a time), glissando between notes can be achieved.



The user interface of Synth1. ADSR envelopes at the top middle. LFO's at the bottom left. When Voice (at the bottom left) is switched, the mono portamento option next to it allows glissandos [2].

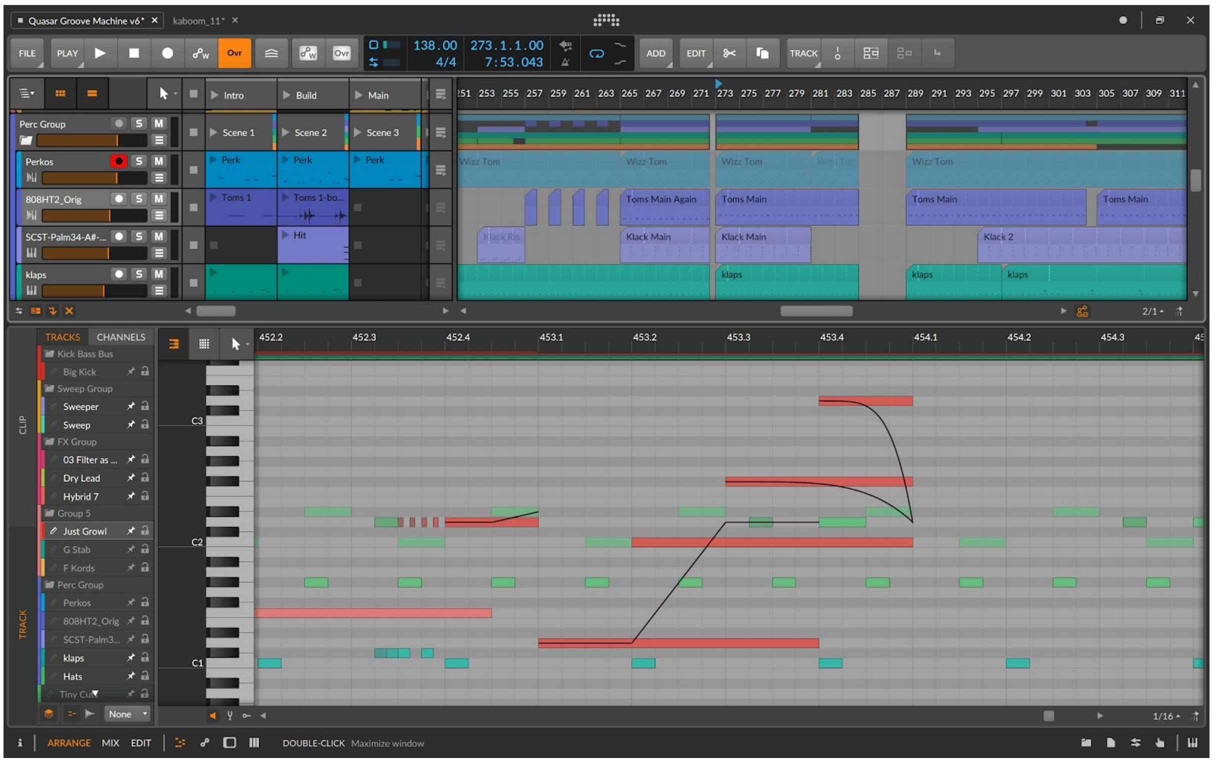
Some synthesizers (and other software instruments) can work with different tuning systems to play microtonal notes that don't fall into 12 tone equal tempered western scale. The number of total notes is still limited to 128 and play notes must be in the specified tuning system [3].

Various file formats to specify microtonal tunings exist, such as *.tun and *.scl [4].

JUCE framework is a popular C++ framework for developing audio applications and plugin [5].

A more flexible protocol called OSC (Open Sound Control) [6] exists but it is usually used for live communication between devices rather than composing music. We can take advantage of this protocol and use it internally in our plugin(s). An experimental music-making program, Max (Max Msp) [7], uses OSC internally. Later versions of Max is also written in JUCE [8].

Many midi editors have tried to break the limitations of traditional midi editors. The one that is most similar to ours is the built-in editor of Bitwig DAW, in which it is possible to make put in off-grid nodes and polyphonic glissandos and play them with instruments that are native to bitwig. Our editor has a different workflow, can be used to achieve additional things and can be used in DAWs other than bitwig.



Bitwig Studio’s midi editor [9]

Melodyne is an editor plugin which is used to alter individual notes in recordings. It has an editor that encompasses the entire timeline and is similar to our project in terms of approach and functionality but it is used to alter already recorded sounds instead of creating new ones like Microgliss [10].

SOUL (Sound Language) is a dedicated audio programming language currently in the early stages of development. It can be used within JUCE [11].

3. Proposed System

3.1 Overview

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. 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.

The innovative and interesting part of the program is the note editor. Our paradigm is a workflow consisting of nodes and edges, which corresponds to notes and glissandos, isn't common in other note editors. We plan to develop the editing capabilities of the editor beyond any existing tool. There are other microtonal editors out there but they are not as flexible as microgliss. Generally, they enforce a grid, a tuning system. In microgliss using a grid is a preference of the user, which also they might prefer not to. They also don't allow individual note glissandos except for Bitwig Studios internal MIDI editor. Our product's different workflow will be more suitable and easy in some cases. Microgliss can be used in DAW's other than Bitwig studio.

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 favourite synthesizers. OSC also has other uses such as light control and microgliss can be used for such side-purposes.

3.2 Functional Requirements:

Editor:

- The user should be able to add nodes to the drawing part of the editor.
- The user should be able to connect these nodes with edges.
- One node should be able to connect to many other nodes.
- One node should not connect to another node with the same time value (x-axis on the grid)
- User should be able to edit and remove nodes
- Connections should reorganize if the node order changes and disable themselves if needed while storing the edges.
- The editor should be able to read MIDI input from the DAW which we can later modify in the editor.

- The editor should be able to send OSC data to other systems.
- The user should be able to zoom in/out in the drawing part of the editor.
- The user should be able to select multiple nodes simultaneously.
- The editor should communicate node inputs to the synthesizer to give auditory feedback to the person composing. This will also make editing the parameters of the synthesizer to achieve desired sounds easier.

Grid:

- User should be able to specify both rhythm (a division of time) and frequency grids (tuning system), which corresponds to x and y dimensions of the editor.
- The editor should be able to read at least one tuning file format (e.g.: tun, scl)
- Users should be able to turn the snap for rhythm and frequency grid on and off easily.
- Users should be able to zoom in/out in the editor and grid should resize respectively.

Synthesizer:

- Users should be able to modify the parameters of the synthesizer to generate different sounds.
- Users should be able to save currently active synthesizer settings.
- Users should be able to load pre-saved synthesizer settings (load presets).
- All synthesizers settings should be editable by the host(DAW). This will let users automate parameters directly from the DAW. Automation in the context of DAW's lines usually drawn by hand to change parameters of a track (eg: volume/pan) and/or a plugin (eg: mix) over time.

3.3 Nonfunctional Requirements

Compatibility:

Microgliss should run on all DAW's which support 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. Microgliss should be able

Extendibility:

Editor and the synthesizer part should be able to connect via OSC protocol and be separate 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).

Responsiveness:

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

3.4 Pseudo Requirements

Implementation Constraints:

- Most of the programming languages aren't suitable for basic audio functionalities and VST generation support. Only C++ is viable for our purposes.
- JUCE is the most widely used C++ audio/UI generation framework. We'll use JUCE for implementation of microgliss.
- The new audio programming language SOUL can be used within JUCE, for some aspects of the program, if it seems like a good fit. SOUL is a new technology and will have instabilities. We need to approach with caution, if we consider using it.
- Our UI will be implemented using JUCE components.

Development Time:

- The minimum viable product can be done in a short time(an academic semester), but the whole product is infinitely expandable. Microgliss is a non-existing way of microtonal editing, so as long as musicians use it, we will see the new needs and add these features.
- Microgliss lives in another complex host, a Digital Audio Workstation. As DAW's evolve and expand, microgliss might need some updates to keep up with these changes.

3.4.5 Additional Functionality

Above we have defined what our program should do in minimum but there is additional functionality that we can add if we feel that it does not harm the clarity and usability of our program. These functionalities are:

3.4.5.1 Split Glissandos:

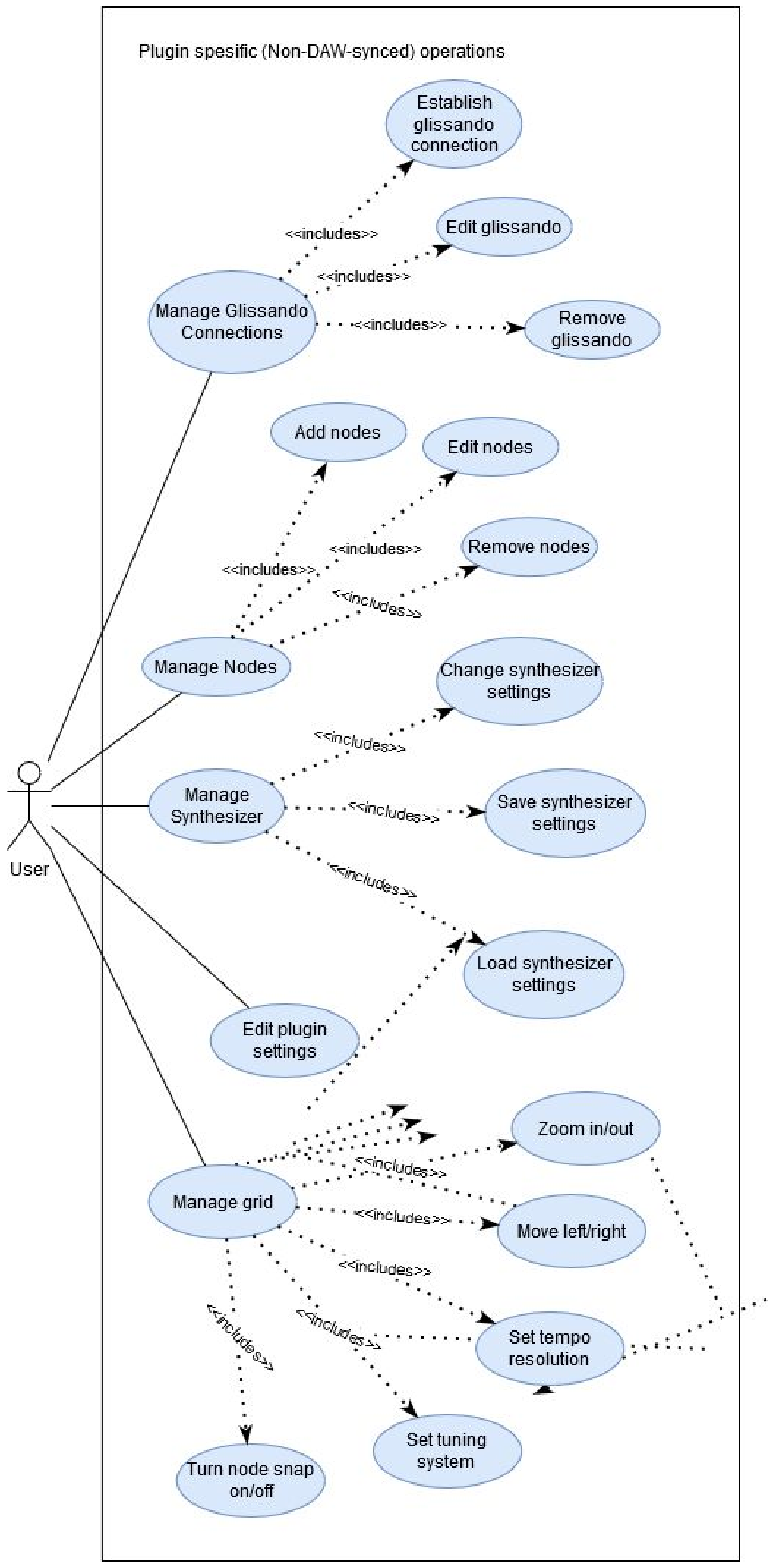
Glissandos that start from one note and end in multiple notes. Or start from multiple notes and end in one note. Or start from multiple notes and end in multiple notes (which would result in a look resembling a neural network in the editor). Support for mass adding editing split glissandos.

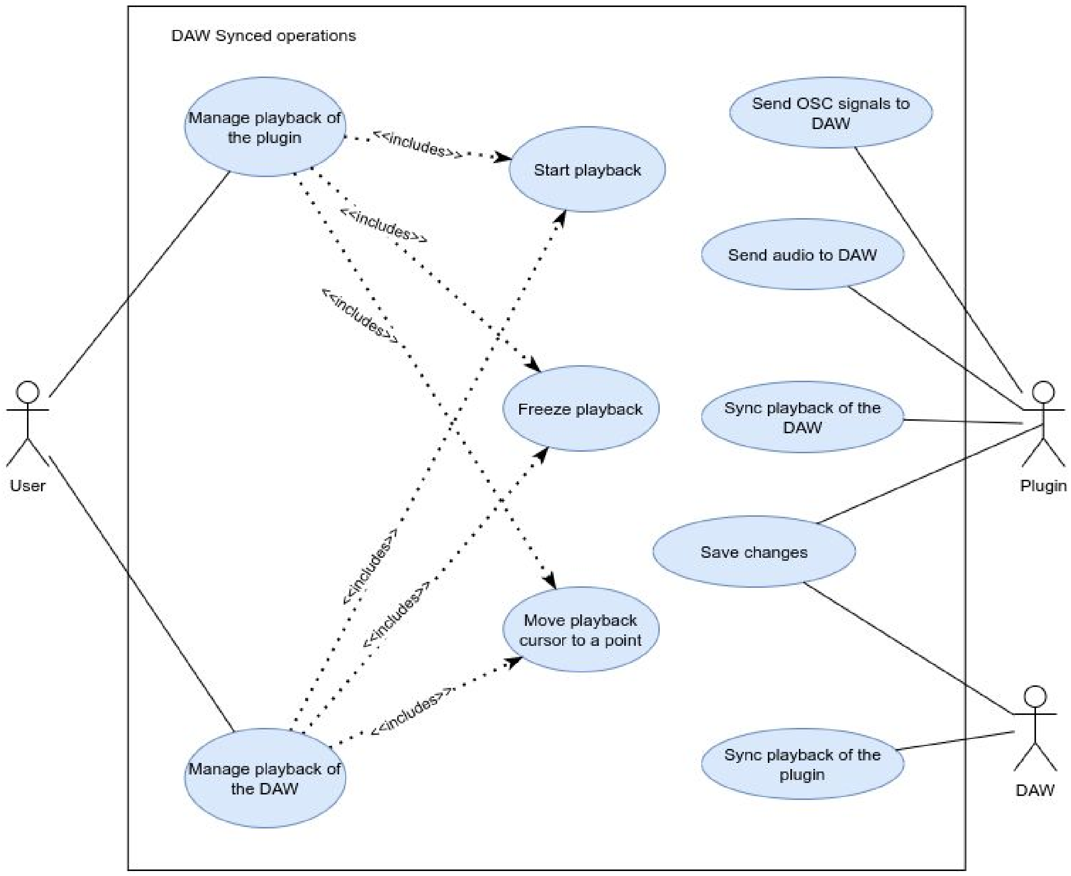
3.4.5.1 Step Glissandos

Support for glissandos between notes that spend more time closer to the notes in the tuning system. This resembles how a glissando will sound in a guitar, which is an instrument with frets, as opposed to a violin which is a fretless instrument.

3.6 System Models

3.5.1 Use Case Model





3.5.2 Scenarios

For the use case model, microgliss has two main operational systems. As it is stated, microgliss is a plug-in, a program which lives in a bigger and more complex program. This creates the need for two types of systems. Synced operations with the host and async operations happen only within the plugin.

3.5.2.1 For Plugin Specific Operations

Name: Establish a glissando connection
Participating actors: User
Description: This use case establishes an editable glissando connection between nodes.
Entry condition: Whenever the user selects multiple consecutive nodes and triggers the glissando operation with command this use case applies.
Exit condition: User stops interacting with the node.
The flow of events:

1. The user selects multiple consecutive nodes.
2. User triggers the glissando operation with a command.
3. System establishes a glissando connection between selected nodes.

Name: Edit glissando
Participating actors: User
Description: This use case lets users change glissando's properties such as curve.

Entry condition: Whenever the user selects glissando line/s this user case applies.
Exit condition: User stops interacting with the glissando line.
Flow of events:

1. User selects at least one glissando line.
2. User edits the glissando (different flow for different editing options)

Name: Remove glissando
Participating actors: User
Description: This use case lets users remove the glissando connections between nodes.
Entry condition: Whenever the user selects glissando line/s this user case applies.
Exit condition: User stops interacting with the glissando line.
Flow of events:

1. User selects at least one glissando line.
2. User “removes” the selected glissando lines by pressing delete
3. System deletes the glissando lines the user removed.

Name: Manage Glissando Connections
Participating actors: User
Description: Use case for managing the glissando connections.
Participating actor: User
Entry condition: This use case includes **Establish glissando connection**, **Edit glissando** and **Remove glissando** use cases. Whenever the user selects multiple nodes to add a glissando or select glissando line/s this use case applies.
Exit condition: User stop interacting with the glissando line.

Name: Add Nodes
Participating actors: User
Description: This use case adds an editable node to the click position.
Entry condition: Whenever the user double clicks on a point within the note grid which contains no node, this use case applies.
Exit condition: User stops interacting with the node.
Flow of events:

1. User double clicks a point within the note grid.
2. User selects the option to add a node.
3. System creates a node at the point where the user double clicked.

Name: Edit Nodes
Participating actors: User
Description: This use case lets users change node properties like velocity or frequency.
Entry condition: Whenever the user selects node/s, this use case applies.
Exit condition: User stops interacting with the node/s.
Flow of events:

1. User selects a node.
2. User can edit the node(different flow for different editing options).

Name: Remove Nodes
Participating actors: User
Description: This use case removes the nodes from the grid, and breaks the established glissando connections if any.
Entry condition: Whenever the user selects node/s and triggers the removal with a command, this use case applies.
Exit condition: User stops interacting with the node/s.
Flow of events:

1. User selects a node.
2. User selects the remove option.
3. System deletes the node the user removed.

Name: Manage Nodes
Participating actors: User
Description: Use case for managing nodes.
Participating actor: User
Entry condition: This use case includes **Add nodes**, **Edit nodes** and **Remove nodes** use cases. Whenever the user clicks within the note area to add nodes, or select one or multiple nodes this use case applies.
Exit condition: User stop interacting with the nodes.

Name: Change synthesizer settings
Participating actors: User
Description: Edit the parameters of the synthisizer in order to generate sounds with different characters
Entry condition: Whenever the user changes any of the synthesizer settings such as changing the unison, ADSR parameters, soundwave etc. this use case applies.
Exit condition: User stops interacting with the synthesizer.
Flow of events:

1. User “selects” the synthesizer.
2. User changes the unison.
3. User edits the ADSR parameters.
4. User edits the sound wave.
5. User exits the synthesizer.

Name: Save synthesizer settings
Participating actors: User
Description: Save the settings of the synthesizers so that they can be loaded later
Entry condition: Whenever the user is on the synthesizer module and clicks on the save icon this user case applies. This use case saves the current preset settings.
Exit condition: User stops interacting with the save button.
Flow of events:

1. User selects the synthesizer module.
2. User changes the synthesizer settings.
3. User clicks on the save icon.
4. System applies the setting changes the user made.

Name: Load synthesizer settings
Participating actors: User
Description: Load previously saved settings for the synthesizer
Entry condition: Whenever the user is on the synthesizer module and loads a synthesizer preset from the list this user case applies. This use case also lets the user reset the synthesizer by selecting the Initial preset.
Exit condition: User stops interacting with the loading dropdown menu.
Flow of events:

1. User selects the synthesizer.
2. User selects a synthesizer preset from a list.
3. System applies the setting changes the preset has.

Name: Manage Synthesizer
Participating actors: User
Description: Use case for managing the synthesizer.
Participating actor: User
Entry condition: This use case includes **Change synthesizer settings**, **Save synthesizer settings** and **Load synthesizer settings** use cases. Whenever the user selects the synthesizer module from top right this user case gets applied.
Exit condition: User stop interacting with the synthesizer.

Name: Edit plugin settings
Participating actors: User, Plugin
Description: This user case lets users customize the plugin.
Entry condition: User clicks on the settings icon.
Exit condition: User closes the settings pop-up.
Flow of events:

1. User clicks on the settings icon.
2. User can customize plugin settings.

Use case name: Zoom in/out
Participating actors: User
Description: This user case lets the user zoom in/out within the grid.
Entry condition: Whenever the user triggers the zoom in/out command within the grid this user case gets applied.
Exit condition: User stops interacting with the grid.
Flow of events:

1. User clicks on the grid.
2. User zooms in with the zoom in command.
3. User zooms out with the zoom out command.

Name: Move left/right
Participating actors: User
Description: This user case lets the user to move left or right within the grid.
Entry condition: Whenever the use triggers the move left/right command within the grid this user case gets applied.
Exit condition: User stops interacting with the grid.

Flow of events:
1. User clicks on the grid.
2. User triggers the move right command.
3. User triggers the move left command.

Name: Set tempo resolution
Participating actors: User
Description: This user case lets the user to change tempo resolution on the grid.
Entry condition: Whenever the user changes the tempo this user case gets applied.
Exit condition: User stops interacting with the grid.
Flow of events:
1. User clicks on the grid.
2. User changes tempo resolution on the grid.

Name: Set tuning system
Participating actors: User
Description: This user case lets the user set a tuning system.
Entry condition: Whenever the use selects a tuning system from the tuning system dropdown menu this tuning system gets applied.
Exit condition: User stops interacting with the tuning system dropdown menu.
Flow of events:
1. User selects the tuning system dropdown menu.
2. User selects a tuning system from the menu.
3. System makes the specified changes on the tuning system.

Name: Manage grid
Participating actors: User
Description: Use case for managing grid operations.
Participating actor: User
Entry condition: This use case includes **Set tempo resolution**, **Zoom in/out, Move left/right** and **Set tuning system** use cases. Whenever the user interacts with the grid this user case gets applied.
Exit condition: User stop interacting with the grid.

3.5.2.2 For Daw Synced Operations

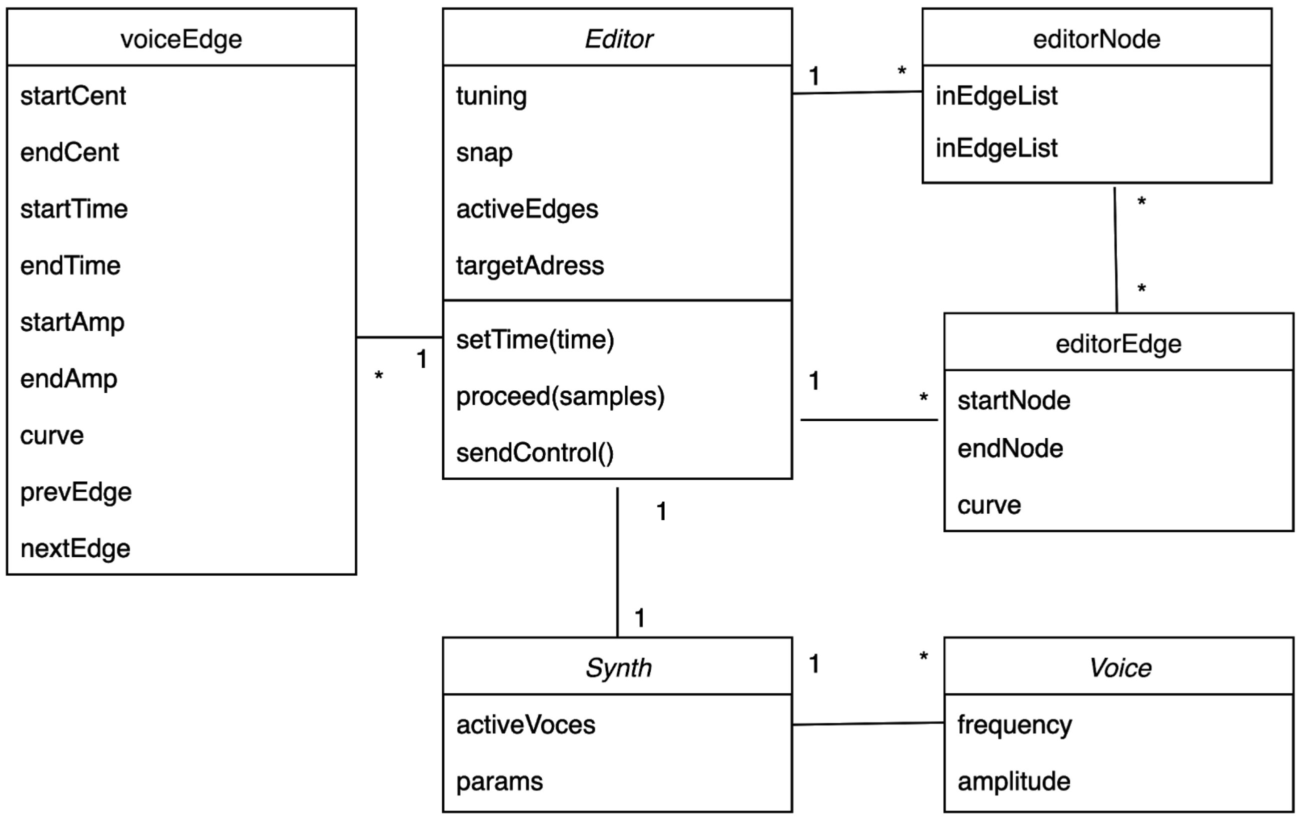
Name: Start playback
Participating actors: User, Plugin
Description: This user case lets the user play the notated notes and triggers the DAW play operation.
Entry condition: Whenever the use triggers the play command or clicks on the start icon.
Exit condition: User stops the playback by triggering the freeze operation.
Flow of events:
1. User clicks on the start icon.
2. System starts playing the notated notes.

Name: Freeze playback
Participating actors: User, Plugin
Description: This user case lets the user stop the playback.
Entry condition: Whenever the use triggers the freeze command or clicks on the stop icon. This command is always executable but makes more sense if the playback is started.
Exit condition: User stops interaction with the freeze operations.
Flow of events:
1. User clicks the stop icon.
2. System freezes the current playback.

Name: Move playback cursor to a point
Participating actors: User, Plugin
Description: This user case lets the user to move current playback cursor to any point within the timeline.
Exit condition: User stops interacting with playback operations.
Flow of events:
1. User clicks on a position in the timeline

Name: Manage playback of the plugin
Participating actors: User, Plugin
Description: Use case for managing the playback operations of the plugin. This change also affects the DAW. And triggers the **Sync playback of the plugin.**
Participating actor: User
Entry condition: This use case includes **Start playback**, **Freeze playback, Move playback cursor to a point**.
Exit condition: User stop interacting with the grid.

3.5.3 Object and Class Model

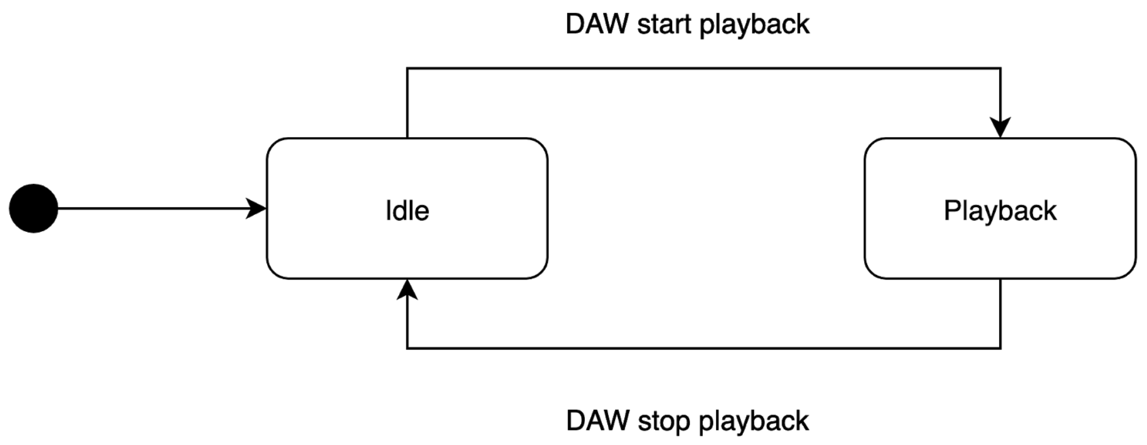


Our class diagram at this is quite simple because much of the responsibility is localized in the Editor and Synth classes. When we begin implementation we will require more classes to construct editor and synthesizer but for our high level class diagram this is sufficient.

- **Editor:** Is the class where we will implement the functionalities of the editor
- **Synth:** Is the class where we will implement the features of the synthisizer
- **EditorNode:** Editor representation of a node
- **EditorEdge:** Editor representation of an edge
- **VoiceEdge:** Is representation of voices (notes and glissandos) that we'll be referenced while it's being sent to the synthesizer. (By differentiating this from the editor representations of nodes and edges we will be able to achieve split glissandos and optimize our code)

3.5.4 Dynamic Models

3.5.4.1 Playback State Diagram



The behaviour of the editor and synthesizer changes slightly depending on if the DAW is in playback mode.

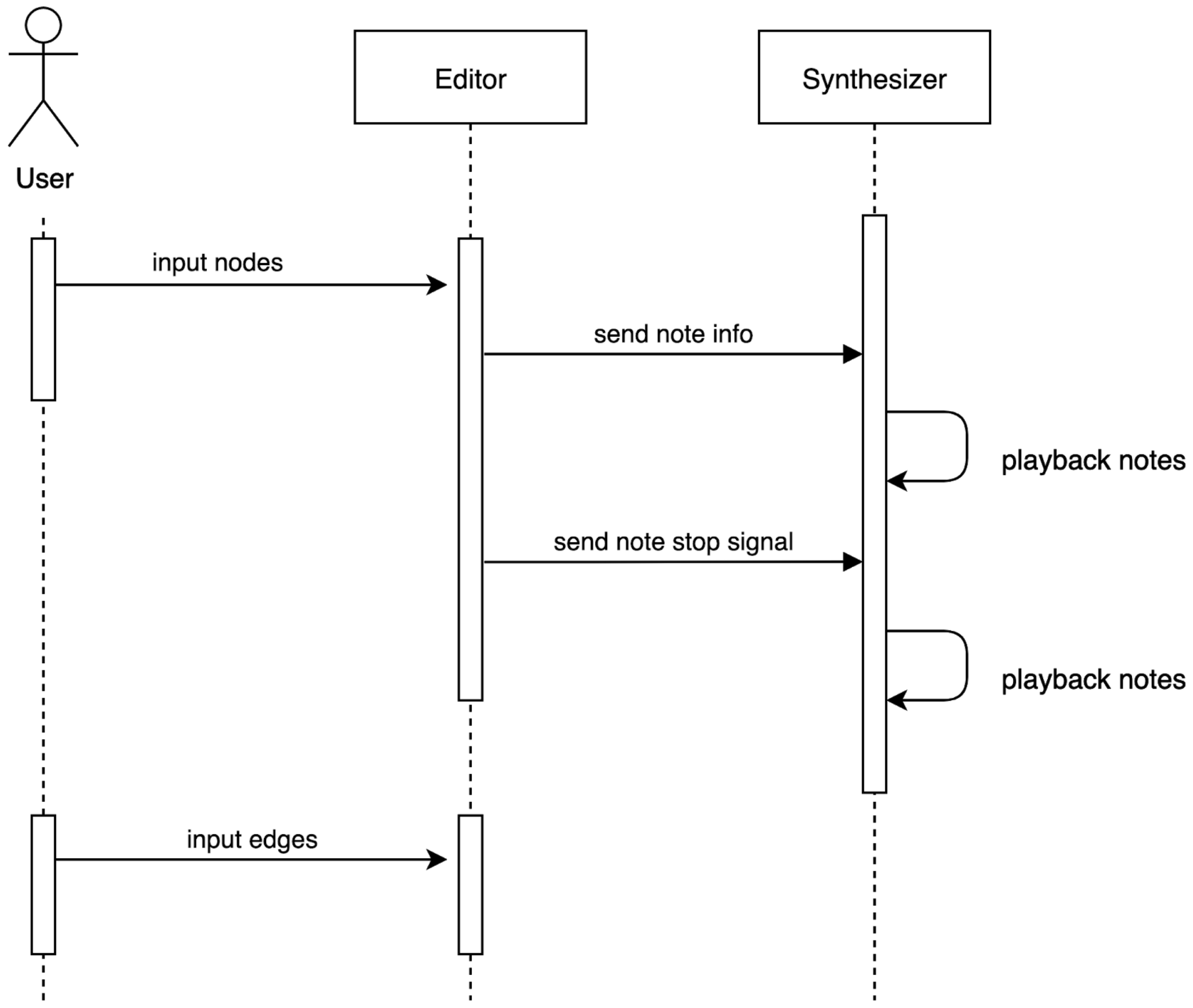
Idle:

- All voices in the synthesizer that are activated by the editor in playback mode are stopped
- However user node input to the editor is still communicated to the synthesizer to provide feedback while composing
- Effects on the synthesizer (such as delay) stays active in this mode

Playback:

- Editor gives synthesizer info on note starts and ends, as well as their current frequency and amplitude based on the current time
- User node input is played for feedback
- Any user changes to the nodes and edges are realculat and sent to the synthesizer

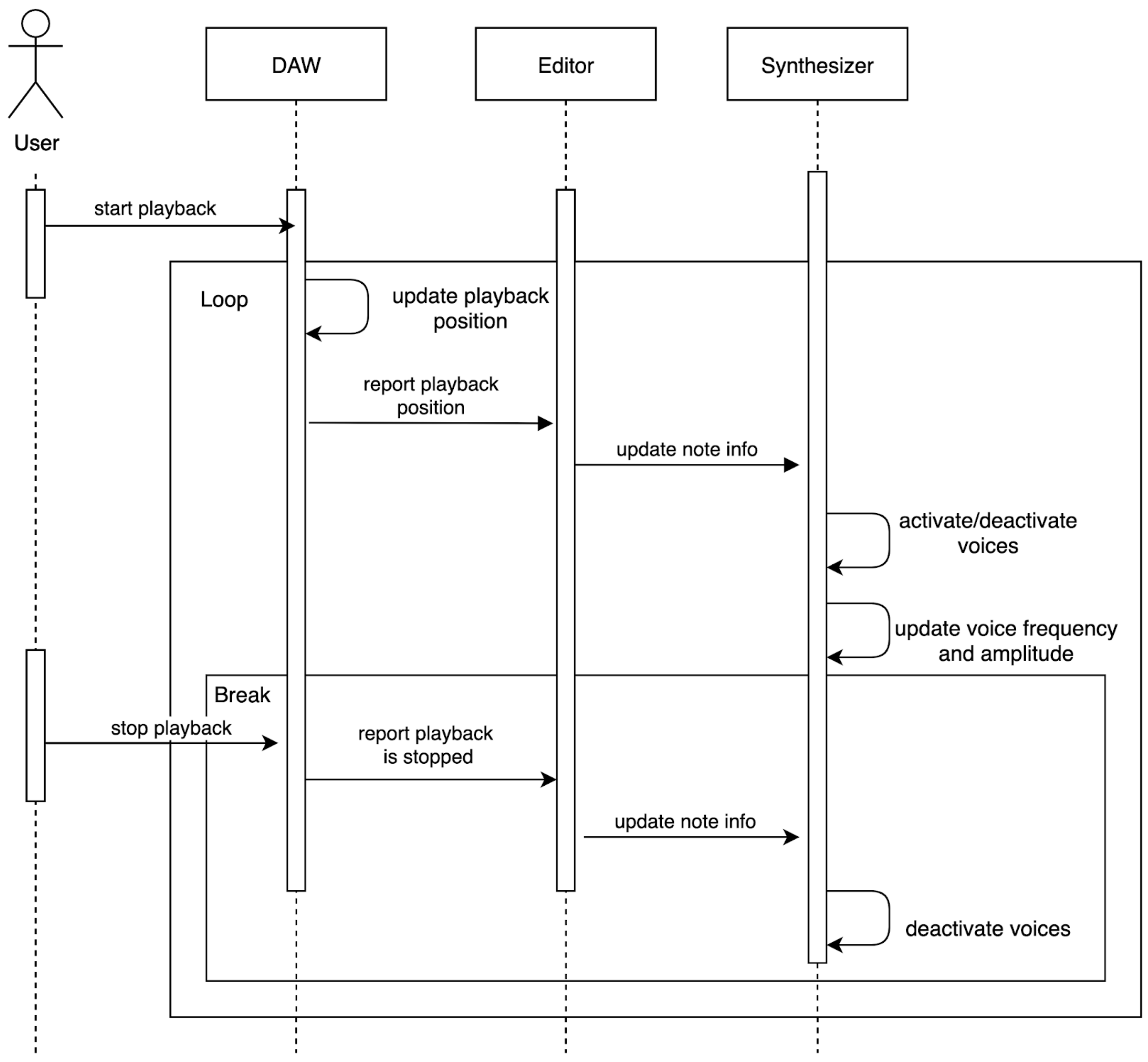
3.5.4.2 Edit Sequence Diagram



Edge input isn't played back because it's duration can be really long. The user can instead check how it sounds by resuming playback from the DAW. This edit sequence can happen both in playback and when playback is stopped.

3.5.4.1 Playback Sequence Diagram

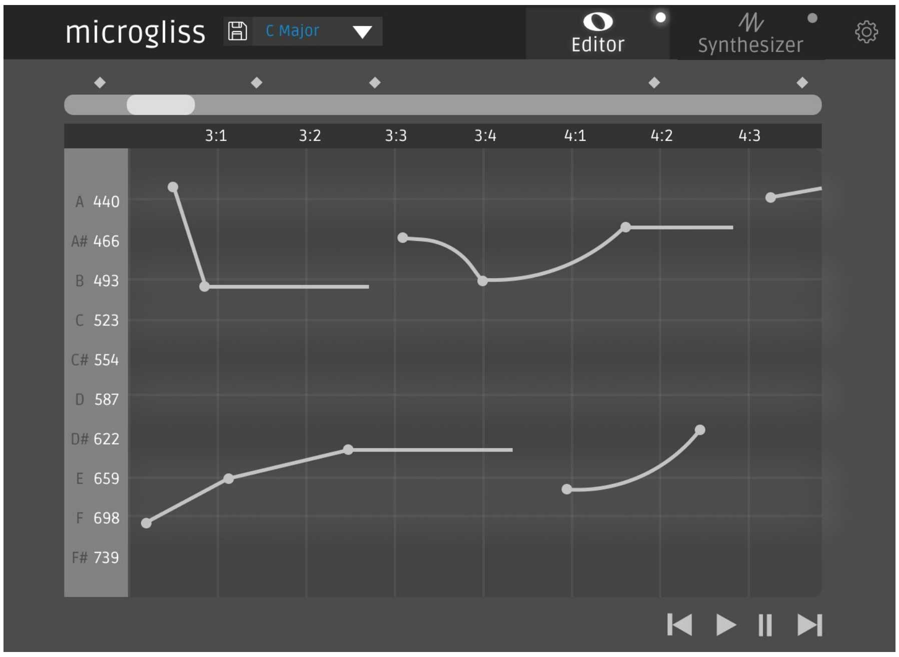
This diagram explains what the user needs to do in order to start and stop playback, and what will be the responsibilities of



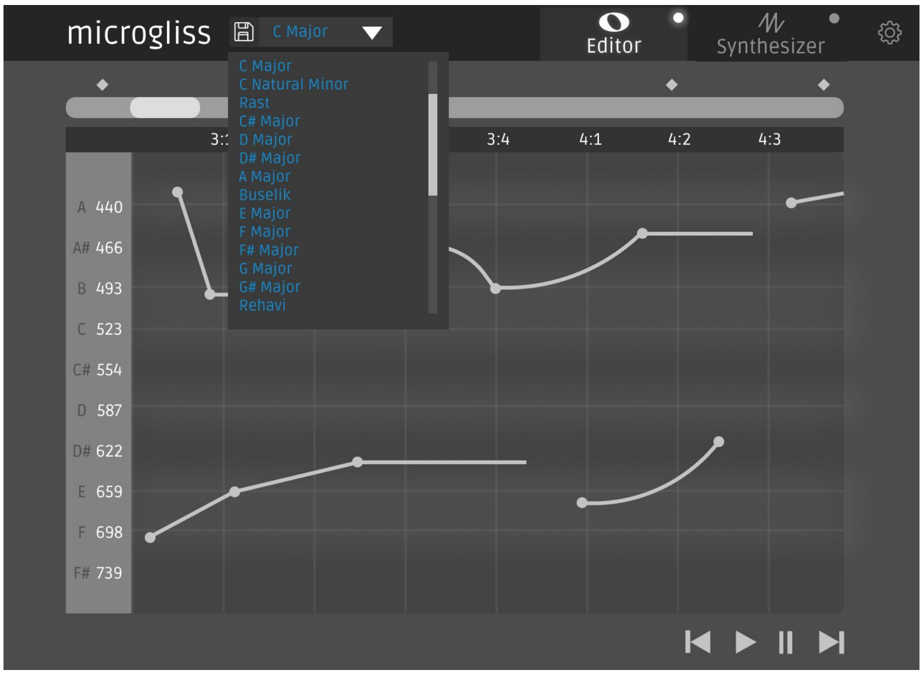
3.5.5 User Interface - Navigational Paths and Screen Mock-ups

As it is discussed in the previous parts, microgliss is composed of two main parts. The microtonal note editor and the synthesizer. In addition to that, Microgliss has a settings window which lets users make quick customizations.

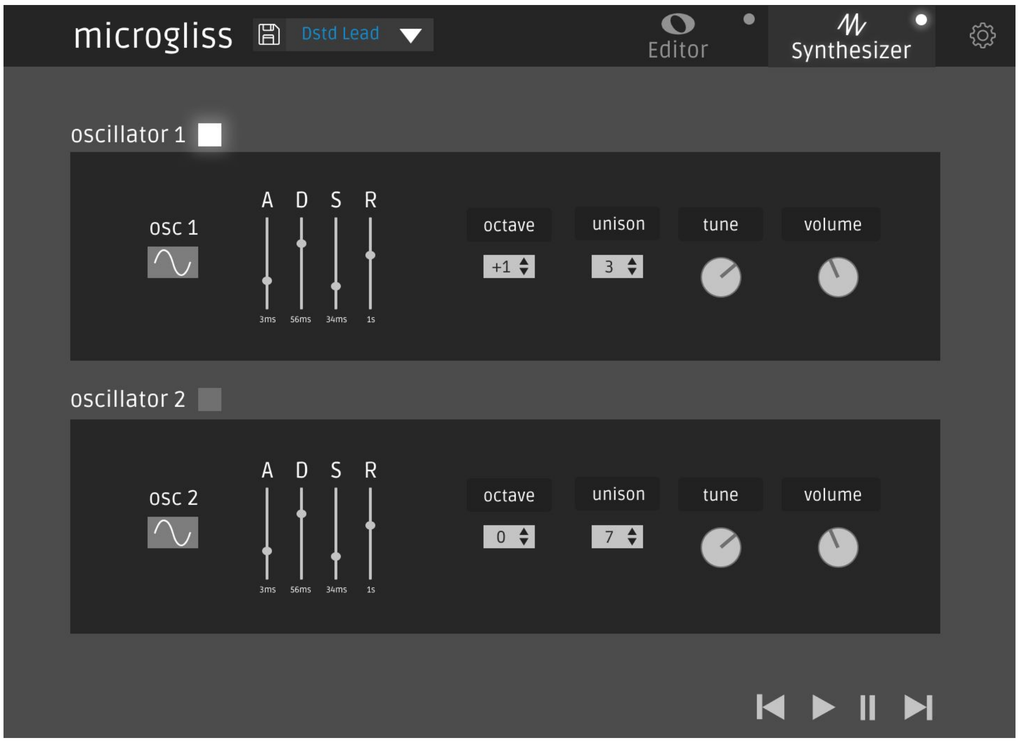
3.5.5.1 Note Editor



This is the standard view for the note editor. On the left top corner there is a logo and a save icon for the current file. There is a scale selector right next to them. Scale selector lets the users to switch between different scales and see the notes belong to that scale on the grid.

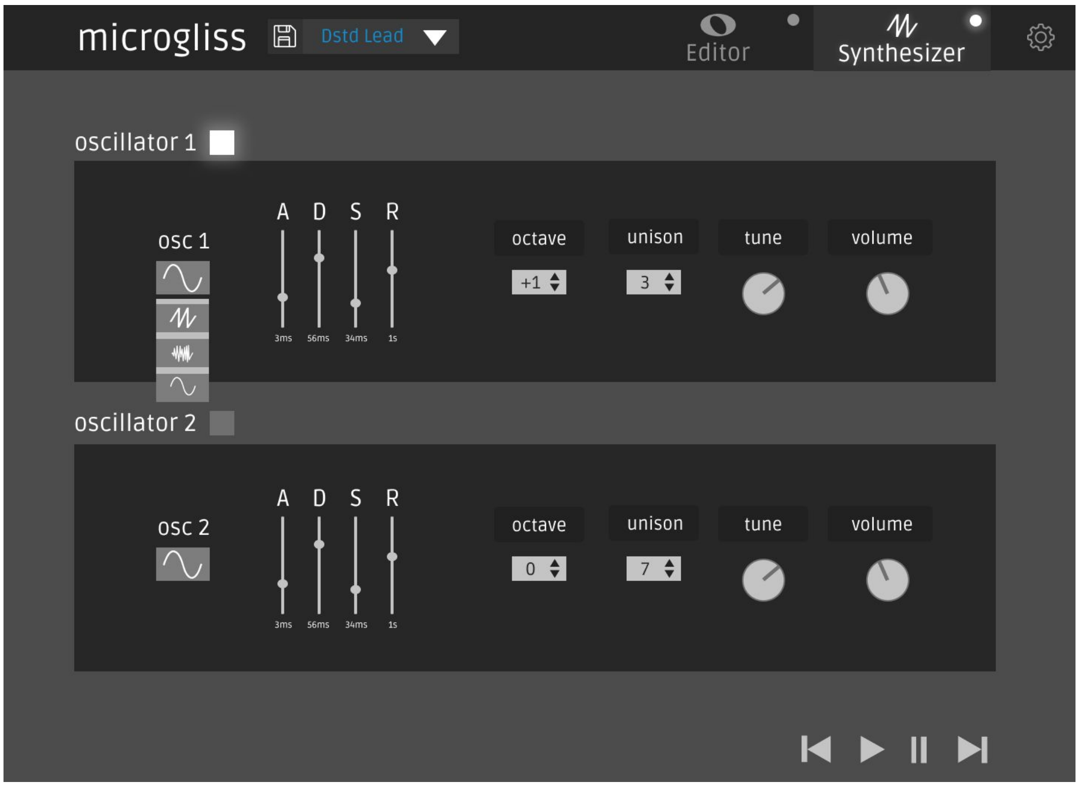


3.5.5.2 The Synthesizer



The synthesizer has 2 different oscillators which can be enabled/disabled by clicking to the box right next to the oscillators name. Box lits if the oscillator is enabled.

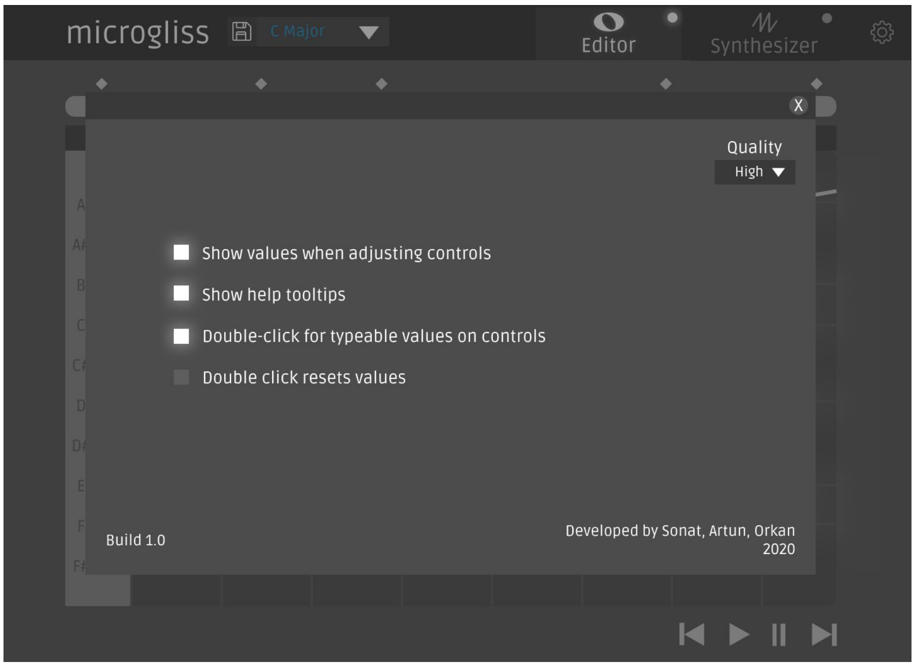
Users can select one of the different soundwaves for the oscillators by clicking on the soundwave image. This opens up a dropdown menu.



Users can select the desired soundwave from the list.

3.5.5.3 Settings

Settings are for the customization of the microgliss.



Quality
High

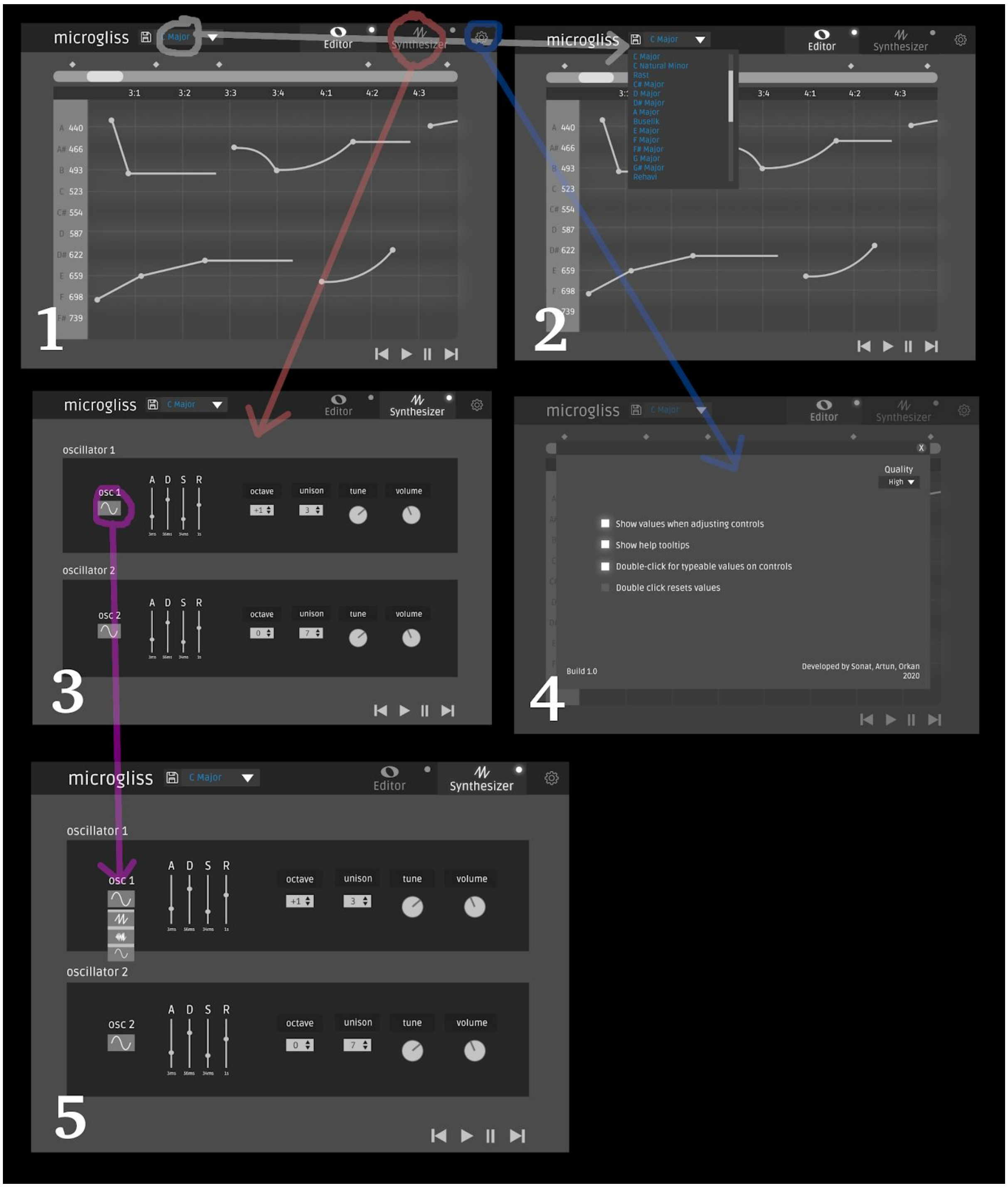
- [✓] Show values when adjusting controls
- [✓] Show help tooltips
- [✓] Double-click for typeable values on controls
- [ ] Double click resets values

Build 1.0

Developed by Sonat, Artun, Orkan
2020

Boxes lit if the setting is enabled.

Navigational path of microgliss:



Show values when adjusting controls
Show help tooltips
Double-click for typeable values on controls
Double click resets values

4. Other Analysis Elements

4.1. Consideration of Various Factors in Engineering Design

The only risk microgliss inherently has is gatekeeping and exclusion, from making music and making music of special kinds. But many of the computer music technologies are already expensive, extremely technical and west oriented. We will design microgliss as accessible and inclusive (culturally, economically etc.) as possible but the bars are very low and we can’t end up doing actual harm.

4.2. Risks and Alternatives

As it is discussed in the previous sections, microgliss consists of two separate modules which will be using JUCE framework and OSC protocol. Both of these modules live in a more complex host, the digital audio workstation. Usage of different libraries, protocols makes the implementation process easier, but also brings multiple risks. This shows that microgliss might have multiple risks during the implementation time, and we need to come up with solutions.

1. JUCE, OSC or VST format gets a major update and stop supporting previous versions:

   JUCE Framework(C++) and Open Sound Control protocol are the main tools that we will use during the implementation of the project and VST format is the format we need to export in, to make our program live in a DAW. This might cause maintenance problems. If any of these tools gets a major change and stops supporting the previous versions we will stick with the version we'll be working on and won't update the tool version until we release our first build. We will be aware of the possible problems of the old versions with their next release, and patch the problems that our users might have with our own code.

2. JUCE Framework might become paid for any use:

   For the time being JUCE Framework is free for students if you let their splash art get shown at the beginning. And additionally it is free for personal use if your income is less than 50k dollars, which is suitable for us. If the JUCE framework becomes paid during the implementation, we will write the built-in audio synthesis and generation part by ourselves, and use another GUI library such as Qt. This will drastically increase the implementation duration, but also guarantee we will have the product in the end.

3. OSC output might be unusable for other plugins:

   If our OSC output is unusable for any other plugins, users might have problems with controlling their own tools with microgliss. We are developing the synthesizer part to solve these problems. And if we come across this kind of problem, we will be updating our synthesizer part for a better use. For the time being our synthesizer only provides the basic features, but this might change.

4. It might be difficult to find other synthesizers to connect to our editor:

   It might be difficult to find open source synthesizers that we can easily modify to use with our editor and developers of closed-source (and open-source) synthesizers might be hesitant to cooperate. In that case we will put more focus into the synthesizer part of the program as well.

5. One of our teammates leaves during the implementation:

Our team consists of three people, and every one of us is needed during the whole development process, but if any of us need to leave during the development process, his tasks will be shared among others. This will increase the workload for the remaining, but also let the development process continue.

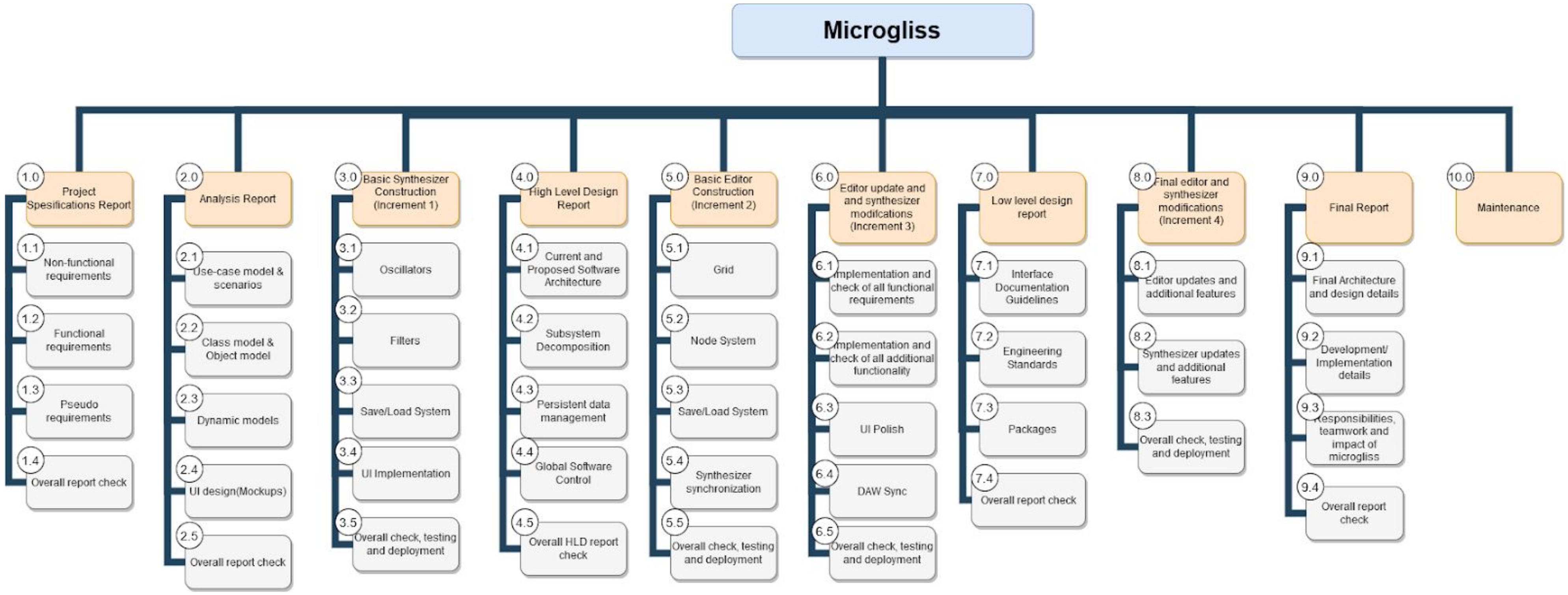
Risk description	Likelihood	Effect on the project	Action
JUCE, OSC or VST format gets a major update and stops supporting previous versions.	Low	Maintenance problems	Stick with the version in development
JUCE Framework might become paid for any use	Low	Product release is not possible due to legal conditions	Code the parts that we use from the JUCE library for the audio and use other GUI tool for implementation of UI
OSC output might be unusable for other plugins	Low	Might not be expandable for users who wants to use microgliss with different programs	Built-in synthesizer will be updated
Might be difficult to find other synthesizers to connect to our editor	Medium	Might not be expandable for users who wants to use microgliss with different programs	Built-in synthesizer will be updated
One of our teammates leave during the implementation	Low	Work power decreases	Workload increases for the remaining team members to balance

4.3. Project Plan

There exist multiple software engineering project management approaches like waterfall, agile etc. For our project we will be using Incremental Approach with free project management tool Trello. The main use of the Trello is agile development, but we can use it with an incremental approach with little changes easily. It is very suitable for a small team like us.

In the incremental build model, software is designed, developed and tested incrementally. Incremental means a little more added in each iteration(build), and all these added parts (components) will work and will be tested by themselves. Each component will be added in another build and this will continue until the product is finished.

Our Work Breakdown Structure for microgliss is:


Microgliss
1.0 Project Spesifications Report
1.1 Non-functional requirements
1.2 Functional requirements
1.3 Pseudo requirements
1.4 Overall report check
2.0 Analysis Report
2.1 Use-case model & scenarios
2.2 Class model & Object model
2.3 Dynamic models
2.4 UI design(Mockups)
2.5 Overall report check
3.0 Basic Synthesizer Construction (Increment 1)
3.1 Oscillators
3.2 Filters
3.3 Save/Load System
3.4 UI Implementation
3.5 Overall check, testing and deployment
4.0 High Level Design Report
4.1 Current and Proposed Software Architecture
4.2 Subsystem Decomposition
4.3 Persistent data management
4.4 Global Software Control
4.5 Overall HLD report check
5.0 Basic Editor Construction (Increment 2)
5.1 Grid
5.2 Node System
5.3 Save/Load System
5.4 Synthesizer synchronization
5.5 Overall check, testing and deployment
6.0 Editor update and and synthesizer modifications (Increment 3)
6.1 Implementation and check of all functional requirements
6.2 Implementation and check of all additional functionality
6.3 UI Polish
6.4 DAW Sync
6.5 Overall check, testing and deployment
7.0 Low level design report
7.1 Interface Documentation Guidelines
7.2 Engineering Standards
7.3 Packages
7.4 Overall report check
8.0 Final editor and synthesizer modifications (Increment 4)
8.1 Editor updates and additional features
8.2 Synthesizer updates and additional features
8.3 Overall check, testing and deployment
9.0 Final Report
9.1 Final Architecture and design details
9.2 Development/ Implementation details
9.3 Responsibilities, teamwork and impact of microgliss
9.4 Overall report check
10.0 Maintenance

Work Package Title	Leader	Members Involved
Project Specification Report	Sonat Uzun	Artun Cura, Sonat Uzun, Orkan Öztrak
Analysis Report	Artun Cura	Artun Cura, Sonat Uzun, Orkan Öztrak
Basic Synthesizer Construction (Increment 1)	Sonat Uzun	Artun Cura, Sonat Uzun, Orkan Öztrak
High Level Design Report	Orkan Öztrak	Artun Cura, Sonat Uzun, Orkan Öztrak
Basic Editor Construction (Increment 2)	Artun Cura	Artun Cura, Sonat Uzun, Orkan Öztrak
Editor update and synthesizer modifications (Increment 3)	Orkan Öztrak	Artun Cura, Sonat Uzun, Orkan Öztrak
Low level design report	Sonat Uzun	Artun Cura, Sonat Uzun, Orkan Öztrak
Final editor and synthesizer modifications (Increment 4)	Sonat Uzun	Artun Cura, Sonat Uzun, Orkan Öztrak
Final Report	Artun Cura	Artun Cura, Sonat Uzun, Orkan Öztrak
Maintenance	Orkan Öztrak	Artun Cura, Sonat Uzun, Orkan Öztrak

Work packages for this WBS are:

Package Set:Project Specification Report	
Start date:September 12, 2020	End Date: Oct 12, 2020
Leader: Sonat Uzun	
Members Involved:Artun Cura, Sonat Uzun, Orkan Öztrak	
Objectives: 1. Construction of the project specification report to give a brief description of the proposed project including project members responsibilities, project constraints.	

Tasks: <1.1>Non-functional requirements <1.2>Functional requirements <1.3>Pseudo requirements <1.4>Overall report check
Deliverables: 1. Project specification report

Package Set:Analysis Report	
Start date:Oct 21, 2020	End Date:Nov 21, 2020
Leader:Artun Cura	
Members Involved:Artun Cura, Sonat Uzun, Orkan Öztrak	
Objectives: 1. Construction of the analysis report which contains the detailed analysis of the project. This report should address all relevant issues about our project.	
Tasks: <2.1>Use-case model & scenarios <2.2>Class model & Object model <2.3>Dynamic models <2.4>UI design(Mockups) <2.5>Overall report check	
Deliverables: 1. Analysis Report	

Package Set:Basic Synthesizer Construction (Increment 1)	
Start date: Nov 22, 2020	End Date:Dec 14, 2020
Leader:Sonat Uzun	
Members Involved:Artun Cura, Sonat Uzun, Orkan Öztrak	
Objectives: 1. Construction the basic synthesizer which can generate sounds and have editable	

parameters. 2. Learning details about C++, JUCE framework and OSC. 3. Understanding the details of the program coding-wise. This will help our team to apply changes and represent the project more correctly in the high level design report
Tasks: <3.1>Oscillators <3.2>Filters <3.3>Save/Load System <3.4>UI Implementation <3.5>Overall check, testing and deployment
Deliverables: 1. First build of the application which only contains the basic synthesizer.

Package Set:High Level Design Report	
Start date:Dec 14, 2020	End Date: Dec 21, 2020
Leader:Orkan Öztrak	
Members Involved:Artun Cura, Sonat Uzun, Orkan Öztrak	
Objectives: 1. Construction of the high level design report.	
Tasks: <4.1>Current and Proposed Software Architecture <4.2>Subsystem Decomposition <4.3>Persistent data management <4.4>Global Software Control <4.5>Overall HLD report check	
Deliverables: 1. High level design report	

Package Set:Basic Editor Construction (Increment 2)	
Start date:Dec 22, 2020	End Date:15 Jan, 2021
Leader:Artun Cura	

Members Involved:Artun Cura, Sonat Uzun, Orkan Öztrak
Objectives: 1. Construction of the basic editor 2. Connect modules(synthesizer and the editor) together
Tasks: <5.1>Grid <5.2>Node System <5.3>Save/Load System <5.4>Synthesizer synchronization <5.5>Overall check, testing and deployment
Deliverables: 1. Second build of the application which contains a basic synthesizer and a basic editor which runs together.

Package Set:Editor update and synthesizer modifications (Increment 3)	
Start date:16 Jan, 2021	End Date:24 Jan, 2021
Leader:Orkan Öztrak	
Members Involved:Artun Cura, Sonat Uzun, Orkan Öztrak	
Objectives: 1. A fully functioning editor and synthesizer module 2. An esthetic User Interface 3. A VST export which we can load to other DAW's 4. OSC output which lets our program to connect other systems 5. A program which is ready for musician use. This version will be send to various musicians to gather feedback and expand the use in next iteration.	
Tasks: <6.1>Implementation and check of all functional requirements <6.2>Implementation and check of all additional functionality <6.3>UI Polish <6.4>DAW Sync <6.5>Overall check, testing and deployment	
Deliverables: 1. Third build of the application, which is fully functional and installable.	

Package Set:Low level design report

Start date: 25 Jan, 2021	End Date:Feb 8, 2021

Leader:Sonat Uzun

Members Involved:Artun Cura, Sonat Uzun, Orkan Öztrak

Objectives:

1. Construct the low level design report which checks validity of the design principles and explains the current phase of the project.

Tasks:
<7.1>Interface Documentation Guidelines
<7.2>Engineering Standards
<7.3>Packages
<7.4>Overall report check

Deliverables:

1. Low level design report

Package Set:Final editor and synthesizer modifications (Increment 4)

Start date:Feb 9, 2021	End Date:April 15, 2021

Leader:Sonat Uzun

Members Involved:Artun Cura, Sonat Uzun, Orkan Öztrak

Objectives:

1. Apply the changes according to the feedback.

Tasks:
<8.1>Editor updates and additional features
<8.2>Synthesizer updates and additional features
<8.3>Overall check, testing and deployment

Deliverables:

1. Final version of microgliss which is ready for the market.

Package Set: Final Report	
Start date: April 15, 2021	End Date: April 30, 2021
Leader: Artun Cura	
Members Involved:Artun Cura, Sonat Uzun, Orkan Öztrak	
Objectives: 1. Construct the final report which discusses the final architecture of the project.	
Tasks: <9.1>Editor updates and additional features <9.2>Synthesizer updates and additional features <9.3>Overall check, testing and deployment	
Deliverables: 1. Final report	

Package Set: Maintenance	
Start date: April 30, 2021	End Date: undefined
Leader:Orkan Öztrak	
Members Involved:Artun Cura, Sonat Uzun, Orkan Öztrak	
Objectives: 1. Keep project updated with necessary environmental changes 2. Add new functionalities according to the user demands	
Tasks: Not defined	
Deliverables: 1. Updated version of the microgliss	

We will follow this structure for the project. As long as we meet the deadlines, the project will finish on time.

4.4. Ensuring Proper Teamwork

As a team we believe that teamwork and responsibility sharing is really important. We are a 3-student group and we believe that even if one of us is busy for some-time the project must always continue. So our main focus is to keep the project moving. This is why we construct our plan and responsibility sharing in a circular manner. Everyone must have an idea about the other parts of the software, even if each of us become 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.

To ensure that each of us is working and active all 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 to online meetings
- Participation to 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 for every one of us and we all know that our contribution is essential for this project to move one. 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.

4.5. Ethics and Professional Responsibilities

One of microgliss' main purposes is opening up the cultural doors to computer music and starting a new era in which every cultural musical element is representable in computer music.This is why the first topic is related with the Global Impact of our project. We aim to let any kind of user to reflect their cultural influences in their music easily. Due to this purpose, microgliss must be culturally appropriate and open and understandable to every musician.
We can summarize our responsibilities as:

- It should be culturally appropriate.
- It should be easy to use for any musician who has used a musical plugin.
- It should have room for creativity.

Due to its creation, microgliss should support creativity and ensure that rules are there to be broken. Ensuring these are our responsibility as the developers of the microgliss.

When we look at today's music market we see that microgliss has no competitor. All other tools are different due to their use and absence of this kind of tool shows that microgliss might have a big Economical Impact.

4.6. Planning for New Knowledge and Learning Strategies

Even though we have a detailed conceptual knowledge about music technologies, audio and music, coding a tool for it is an entirely new challenge. EEE391 Basics of Signals and Systems is an obligatory course for us Bilkent University Computer Engineering students, this course doesn't have enough information for developing a project like this. And also we also were introduced with C++ in CS201 and CS202, we need to remember what we've learned and become better at it. We need to learn many things for the development process. Main topics are:

- Expertise in C++
- Digital Signal Generation and Processing
- Expertise in JUCE framework
- Knowledge in OSC format
- Basic knowledge in VST and DAW connection basics
- Basic knowledge about microtonal music
- Communicating with musicians to collect constructive feedback

Our main learning source for the coding and implementation aspect of our project will be the internet. For the JUCE framework and C++ there is a youtube channel called "Audio Programmer" (https://www.youtube.com/channel/UCpKb02FsH4WH4X_2xhIoJ1A) which has multiple series and many videos about VST development with JUCE.JUCE forum will also be an important resource for us (https://forum.juce.com/)
JUCE, OSC and VST have easy to understand documentations and code examples. Also for the whole development process we will be talking with each other to teach other. Because as a team all of us have different expertises, we need to share them to improve each other. Also during the process we will talk with the experts to get constructive feedback and criticism. Until today we've already talked with our instructors Dr. Ufuk Önen and Dr. Tolga Yayalar. We will keep working with them through the project and we can be directed to other instructors and experienced people with their reference. We think that learning from experts is really important, and we thank them for their support.
Microgliss will be a special musical tool which embraces cultures and will be open for experimentation by the end user. And of course we will be experimenting during the development process too. We are open to a learn-by-doing approach.

5. Glossary

Microtonal Music: Music that consists of intervals which 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 which 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.

6. References

[1] Craig Anderton's Brief History of Midi.

https://www.midi.org/midi-articles/a-brief-history-of-midi. [Accessed: 21 Nov 2020].

[2] Synth1 by Daichi Laboratory (Daichi Toda).

https://www.kvraudio.com/product/synth1-by-daichi-laboratory-ichiro-toda [Accessed: 21 Nov 2020].

[3] List of Microtonal Software Plugins.

https://en.xen.wiki/w/List_of_Microtonal_Software_Plugins. [Accessed: 21 Nov 2020].

[4] Resources for Microtonal Music. https://sevish.com/music-resources/#tuning-files. [Accessed: 21 Nov 2020].

[5] JUCE. https://juce.com/. [Accessed: 21 Nov 2020].

[6] An Enabling Control For Media Applications. http://opensoundcontrol.org/. [Accessed: 21 Nov 2020].

[7] What is Max? https://cycling74.com/products/max. [Accessed: 21 Nov 2020].

[8] Building Max with JUCE. https://juce.com/discover/stories/building-max-with-juce. [Accessed: 21 Nov 2020].

[9] Bitwig Studio. https://www.bitwig.com/overview/. [Accessed: 21 Nov 2020].

[10] New in Melodine 5. https://www.celemony.com/en/melodyne/new-in-melodyne-5. [Accessed: 21 Nov 2020].

[11] SOUL, The Future of Audio Coding. https://soul.dev/. [Accessed: 21 Nov 2020].