University of Twente

Design Project

Singing Game SoundLAB

Design Report

Author:
Michael Janssen (s2130785)
Wesley Joosten (s2176610)
Carlijn Meijerink (s2085593)
Wouter Suidgeest (s2175851)

Supervisor:
Dennis Reidsma

Client:
Laura Slakhorst
Benno Spieker

April 21, 2021

UNIVERSITY OF TWENTE.

Contents

Introduction

In Enschede, a new SoundLAB is in development by the Wilmink theatre and ArtEZ Conservatory. This will be a space where children can experiment with all kinds of interactive musical instruments to explore sound and music.

One part of this room will be an interactive singing game that can be used to introduce children to the effect of different acoustics on their voice. Master students of Interaction Technology have already been working on prototypes of this game, but some critical issues were not ready for use, such as a problematic user interface and uncommon software.

We have worked on an improved web-based singing game that meets the client's desired requirements during this Design Module: the game should be usable by children, teachers, and researchers, all with their own purposes and reliability.

For the development process, we decided to work with agile practices, adopting most of the principles of the Scrum framework, like short development sprints, daily stand-ups, and sprint reviews with our clients. We decided on a time frame of one week for the sprints, keeping track of our product and sprint backlogs and progress in a Kanban-like way.

During the development process, we went through different stages. Firstly, we worked on requirements specification to get a clear overview of what the client wants and make sure this is communicated and agreed upon properly. We also performed a risk analysis during this time, which can be found in Appendix F Next, we decided on some global design choices that would influence the rest of the project. Thirdly, we concurrently worked on deciding on more detailed design aspects and implementing these. Since the game and software are pretty compartmentalized, it is possible not to decide everything upfront before starting development ultimately.

In this report, we will further describe our development process structured by the four stages, concluding with a description and evaluation of our way of working and suggestions for further developing the product.

Chapter 1

Requirement specification

To get a clear overview of what our client wanted, we had a few meeting at the beginning of the module to specify the requirements. A general description of the goals and requirements of the project can be found below. A detailed list of all requirements follows this. We have grouped the agreed-upon requirements when they serve the same goal. We also sorted them via the MoSCoW principle to keep the priorities clear.

1.1 Application Goals

The product should be a game that encourages children to explore their voice. Teachers should be able to use it as a tool in their music lessons. Furthermore, game-play data should be collected so academics can research the effectiveness of the game.

1.1.1 The Child: Voice exploration

An essential goal of the game is letting children explore their voice. To make this function well, the pitch detection of the user should work correctly. Furthermore, the visuals of the game should result in changes in the acoustics. The game should also motivate the child to keep playing and exploring. Thus, adding gravity is very useful.

1.1.2 The Teacher: Education

The product should be easy to use for teachers in primary schools. Therefore the set-up should not require much effort and the interface of the game should be clear enough for the teacher to adapt the settings easily. The game should also be intuitive so that not much explanation is needed.

1.1.3 The Academic: Research

The product should be easy to use for teachers in primary schools. Therefore the set-up should not require much effort, and the interface of the game should be clear enough for the teacher to adapt the settings quickly. The game should also be intuitive so that not much explanation is needed.

1.2 Requirements

We went over a few iterations of the requirements list with our client to make sure we were on the same page. Their function groups the requirements, and they are sorted based on the MoSCoW principle.

1.2.1 Musical

Most importantly, children must be encouraged to use their voice and make changes and adaptions based on visuals. This is done by having some visual effect that corresponds with the pitch of the sound they produce. Ideally, they should be singing; thus, they should be discouraged from staying silent. Other options are to have children also explore the length of notes and loudness of their singing, so visual feedback on these criteria should be given. It could also be nice to load in MIDI files so the game can use a specific song. Another option could be for the game to generate some generic, pleasing, but straightforward melodies. It would be fun to have some visual feedback on the style a child is singing in, but this is unnecessary.

Must: As an educator I want a child to be encouraged to explore their voice.

The application must motivate children to interact.

Must: As a user I want the visuals to correspond with the pitch of my voice.

The player must go up and down based on the pitch of the user.

Must: As a user I want the game to motivate me against being quiet.

The game should give some form of "punishment" to not singing when you should (e.g. gravity is added).

Should: As a user I want the visuals to correspond with the loudness of my voice.

There should be some visual feedback or game-play element based on how loud someone is (e.g moving faster when louder).

Should: As a user I want the application to motivate to sustaining a note/notes.

There should be longer notes that need to be sustained, as well as slides from one note to another.

Could: As a user I want to upload a MIDI file for the game input.

It could be possible to create levels by uploading a MIDI file of a song (e.g. make the notes correspond to a children's song).

Could: As a user I want the levels to have melody.

The application could have nice-sounding, logical melodies.

Would: As a user I want the visuals to correspond to my singing style.

Changes in singing style (e.g. twang) would have an impact on the game.

1.2.2 Theming and environments

Since a crucial aspect of the game is children experiencing different acoustics in different environments, the children should hear some form of audible feedback with an appropriate acoustic effect based on the environment the game is showing. Thus, there have to be multiple themes, each with a distinct look and feel, corresponding to a real-world environment with a particular acoustic effect. When playing a level of the game, a child should be exposed to multiple of these environments to experience the changes in acoustics.

Must: As a user I want the acoustics to correspond to the current theme.

The audible feedback must change based on the theme, e.g. reverb in a cave.

Must: As a user I want multiple themes.

There must be multiple themes in the game (e.g water, cave).

Must: As a product owner I want multiple themes in a level.

Within a level, the themes must switch, e.g. moving from a cave to underwater

1.2.3 Ease of use

The game should be easy to use for both children and teachers. It should be intuitive and not require much explanation or knowledge of music theory. Helpful information should be easily accessible and easy to read. Furthermore, children must like playing the game, so it must also look good.

Must: As a teacher I want an user friendly interface for adapting the settings.

Teachers must be able to easily and intuitively change settings.

Must: As a teacher I want to view the children's scores and level.

There should be some way for a teacher to see how well a pupil did (e.g. an overview is shown after the child is finished).

Must: As a child I want the interface to appeal to me.

Children must be engaged by the application.

1.2.4 Research

The game is also intended to be useful for research, so as much data as possible about the game-play of children should be available.

Must: As a researcher, I want proper logging functionality.

Data collected during game-play should be available for research purposes (e.g. the data can be downloaded and analysed in SPSS).

1.2.5 System

The game should be as easy to adopt as possible. Hence, it must not require installing any software, has to always be available and support multiple concurrent instances. It must work with any type of computer and all audio hardware.

Must: As a teacher I want to be able to easily set it up in primary schools.

Installing and starting the tool must be as effortless as possible.

Must: As a user I want to use it without installing software.

It must be usable without installing any uncommon software.

Must: As a product owner I want the application to be reliable.

The application must always work, and work smoothly.

Must: As a product owner I want multiple schools to be able to use it simultaneously.

It must be usable at multiple places at once.

Must: As a teacher I want multiple children to be able to use the application at the same time.

Multiple children within a school must be able to use it at once.

Must: As a user I want the application to work with any audio device.

It must be usable with any speaker and microphone.

Must: As a user I want the application to work on tablets, chrome books and laptops.

The application must work on any device.

Chapter 2

Global Design

In this chapter we describe our global design choices by elaborating on the preliminary choices we made and giving a general overview of our application and its functionalities.

2.1 Previous prototype

Before we started this project, another student(van Soelen, 2020) already designed and tested what kind of game features children would like. Thus, we did not have to start from scratch with the design of the user interface. The previously existing prototype did get the feedback that it was hard to set up and was confusing to use (especially the settings menu). Therefore two important goals of our application would be that the game should be easily set up and that the settings menu would be straightforward and easy to use.

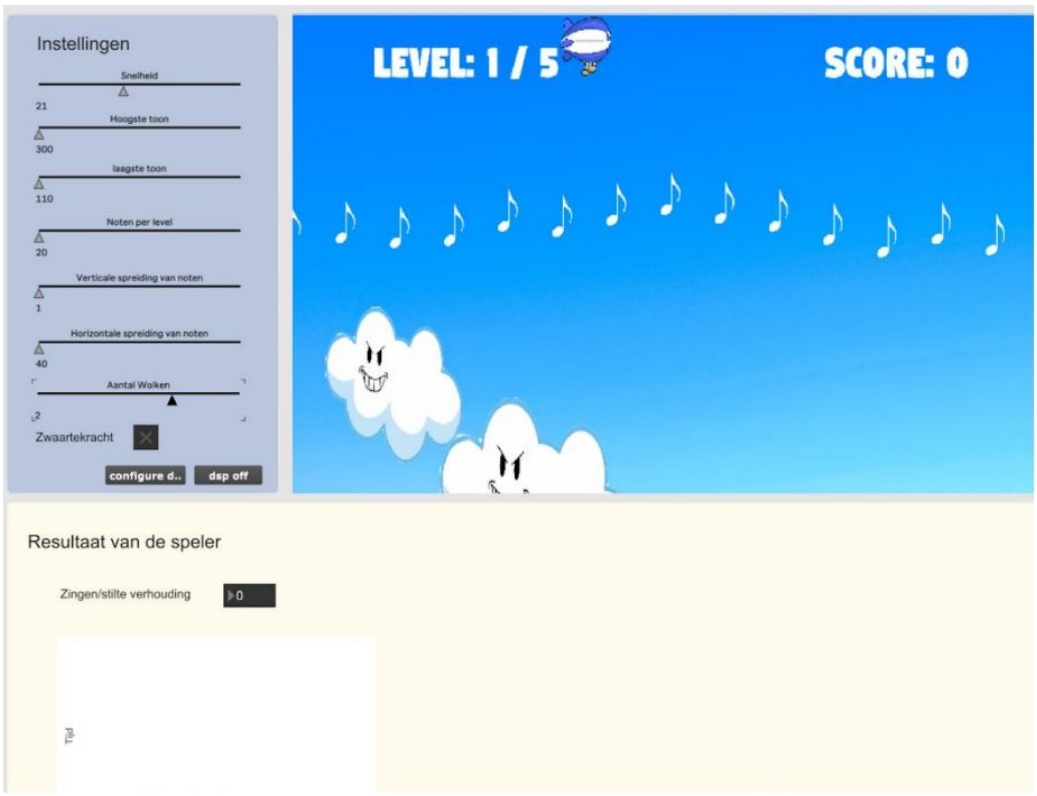

Figure 2.1: The previous prototype

2.2 Preliminary Design Choices

Before we started developing our product, we researched which libraries and languages were suitable for our goal. Furthermore, there already existed a prototype for our game that we could use while making user interface choices.

2.2.1 Programming languages and libraries

We first estimated the complexity and required materials for the application. The programming languages/libraries should handle microphone input and audio output and render images. On top of that, our client expressed a need for the application to be easily accessible, not requiring any installations or extra libraries. These considerations led to the decision to build the application to be used in a web environment, easily accessed through a website.

Language

We decided to use TypeScript, a JavaScript transpiler that adds features such as typing and abstract classes, which helps with a project with much complexity. We use the ParcelJS bundler for bundling and compiling all source files and assets to a minimized packages.

Rendering

As discussed before, the application requires rendering to the screen. There are multiple ways of accomplishing this in modern browsers: One could use the low-level Canvas or WebGL API, use a rendering engine, or a game engine with a rendering engine included. Using the Canvas or WebGL API's would require re-implementing features rendering engine have already created, so we decided to go with a rendering engine. We also felt that because our application requires some abstract features, i.e. using the microphone, we would be hindered by game engines that add abstractions to JavaScript, rendering our microphone request useless. That is why we decided to go for the PixiJS library, a mid-level rendering engine, giving us enough control over the canvas while not limiting our potential.

2.3 System overview

Our game consists of three different components, each with a different purpose and stakeholders. We will elaborate more on this in the paragraphs below. See appendix B for an overview of the use of the game in SoundLAB Enschede.

2.3.1 The game

The game is the part most important to our main users, primary school children. When opening the game, they are introduced to our hero, who needs help finding her golden tuning fork. By selecting a level of their choice, they are shown the flying hero who needs to finish the level while collecting music notes and avoiding obstacles. During this level, the child hears its voice back with different acoustics effects in the different environments. The user interface of the game must be appealing and clear to children. We managed to do user testing and research to reach this goal.

2.3.2 The settings menu

This component aims to provide users and teachers with the option to adapt the game's difficulty to the capability of the current user. When the game is used in a classroom setting, the menu can also be popped out and used on a second screen while playing on an interactive whiteboard. Many different settings like the number of obstacles, frequency of breaks, length of breaks can be adapted to provide much freedom for the user. The note range of the game can also be adapted to the child's age or the preferred voice range of an adult.

2.3.3 Research back end

This component has two goals. Firstly, it should be possible to analyze the sung route of the user by giving an overview of the pitch, notes and obstacles. This can be used in music education to give detailed feedback on the singing style of the user. Secondly, the data of the game should be downloadable for further research. Since the game will be used at SoundLAB and other locations, we have implemented that a redirect will store the data with the provided name. This way, institutions can easily search back their user data.

Chapter 3

Detailed Design

Below, we will discuss all our made design choices in detail and motivate them. We will separate this into three parts; the user interface explaining all the content a user sees, the game logic that fulfils the functionality of the game, and the research back end, which offers the option to collect and extract data.

3.1 User Interface

The user interface encloses multiple aspects of the application. They all represent the part of the application that the user takes in, such as visual design and audible elements. This section will explain the choices made in designing these elements.

3.1.1 Child appealing design

To make sure children of primary school age liked the design, we did research and user testing, which is explained in chapter 6. The following topics are designed based on this.

Game Colors

Since children like bright colours, we made sure to find a colour scheme that suited this. We showed them different colours (all bright) during our user testing, but no clear preference came out. Therefore we chose the current version since it was most appealing to adults as well.

Animations

To keep the attention of the children to the game, we used many animations. The titles appear with a tween, and all the buttons start jiggling when hovered over. Besides, the player and its golden tuning fork are animated on different pages of the game.

Sound effects

Besides the effects and feedback on the voice, we also added victory and failure sounds while playing the game. All the buttons also make a slight noise when clicked.

Story

To make the game's goal interesting for children, we designed, together with an animator, the player's story needing to find a golden tuning fork. This feature will motivate children more to finish a level since then the fork is achieved.

3.1.2 Structure settings menu

An important feedback point from the previously developed prototype was that the settings menu was difficult for (non-musical) teachers. To improve the usability,

we decided on the following:

Explanations

Since we have many different settings, it can be hard to see what they are doing precisely. Therefore when hovering over setting options in the pop-out menu, a short explanation is provided. For a detailed explanation, users can go to the Manual, see appendix A.

Default settings

The default settings of the game are set to be doable for most people. Furthermore, we have predefined frequency ranges for children in different age groups (3-6, 7-9, 10-12) and adults with different voice ranges like bass, alto, and soprano. This feature should make it easy for non-musical teachers to adapt the frequency range to the user.

Easy adaptable

All default settings are easily and real-time adaptable. This way, it is very low-effort for a teacher to customize the settings to a student since they can immediately test if they have the preferred effects.

3.1.3 Components researchers page

For the research page, we did not do much user interface design. We used the basic back-end design provided by Django and, after consultation with our client, decided that this, primarily if explained well in the manual, should be easy to work with for researchers.

3.2 Game logic

The game logic has different aspects. We created a game engine on top of PIXI.js to make the development process more manageable and structured. Using this game engine, we created most of the rest of the game. We use a library written in Rust for pitch detection, and for sound effects, we use a JavaScript library.

3.2.1 Level generation

For generating the game-play of a level, we use a description of the level. These descriptions contain the name of the level, a note generator, an obstacle generator, a list of themes and their timings, and the level's duration. Separately there is a generator for breaks that is the same for every level.

There are currently three different note generators. These all generate notes at a specific height of the screen at a specific time. There is a generator that generates somewhat random notes. It keeps track of a rate of change of the frequency of notes and applies a random acceleration to every note it generates. This makes it, so there is a wave-like pattern. Secondly, there is a generator that generates a significant scale of notes going up and down. Lastly, there is a generator that generates pseudo-random notes in a specific musical key. It picks a random key at the start of the level and then generates notes based on a normally distributed random interval from the last note. This also causes a somewhat wave-like pattern, but changing the variance can change the size of jumps between notes.

For obstacle generators, there is currently only a random generator. It generates obstacles at a random height at random timings with at least a second in between them.

The level takes care of switching themes based on the list contained in the level description. Each theme has its background image, texture for obstacles and acoustic effect.

There is a sort of generator that starts and stops breaks at given intervals and timings determined by the settings for breaks.

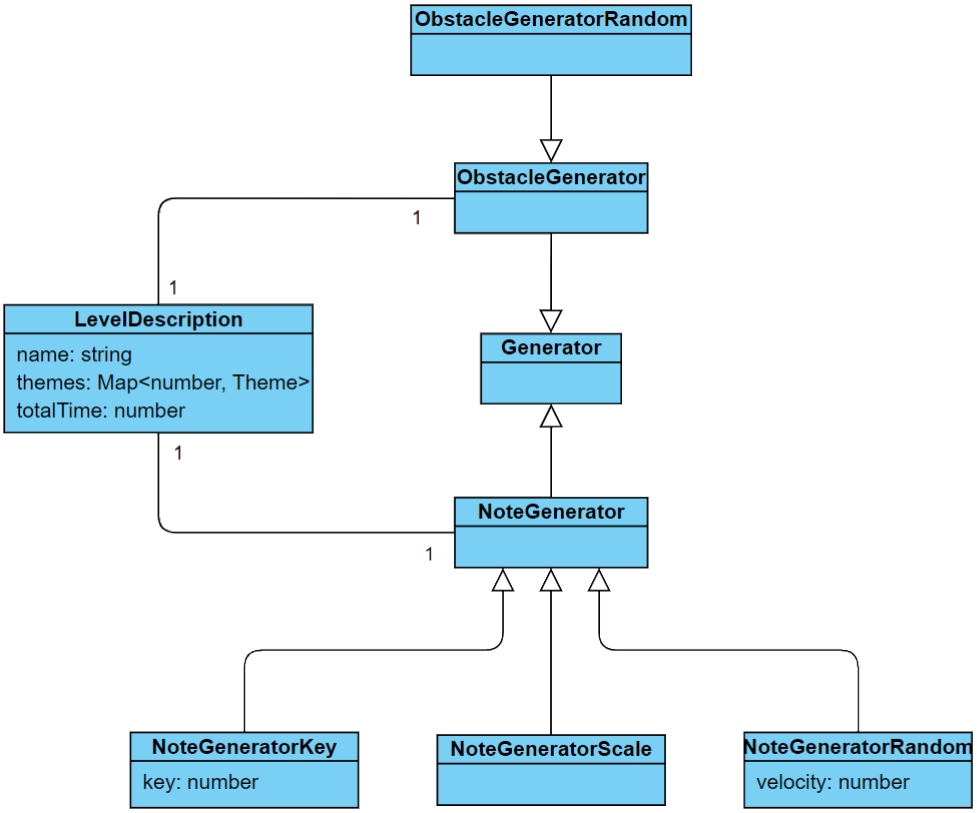

Figure 3.1: The class diagram of the level description and generators.

3.2.2 Game engine

After a week of development, we noticed that the codebase was growing to be too complex. For example, we had many objects that needed to be updated every frame. They all needed to be manually added to the Ticker, causing much overhead. We created an abstraction on the Pixi library, which allowed for some game engine functionality. This abstraction was loosely based on experience with the Godot engine. The application is now represented using a tree, in which the application is represented. Everything in this tree has the following functions: 'enterTree', called when the Node enters the application tree, 'exitTree', called when the Node exits the tree, and 'process' is called every frame. Something which needs to be called every frame now only needs to extend Node and implement the process function. On top of this, there is a Node2D, which handles the Pixi rendering tree automatically. An example of the tree structure of the application can be found in Appendix E

3.2.3 Pitch detection

Since pitch detection was a vital part of our project, it was essential to get it right. It also needed to be quick, as the player needs immediate feedback on their pitch without disturbing the game flow.

The Algorithm

For the pitch detection, we decided to go with the McLeod Pitch Method(McLeod and Wyvill, 2005). This algorithm is a fast, accurate and robust method of finding pitch in our use case and is widely spread and implemented in libraries in many languages.

The Language

After implementing a JavaScript method, we benchmarked the performance and noticed that it had sub-optimal run times, slowing down the application and giving the player a noticeable delay. JavaScript was therefore not an option. Instead, we opted to go for a low-level language that could compile to WebAssembly, a web standard to run low-level assembly code in the browser: Rust. This method was easy, as there was a crate - the Rustacean name for a library - which implemented McLeod pitch detection. We used wasm-pack to compile it to WebAssembly and created a JavaScript class for communication with the pitch detector.

3.2.4 Voice modulation

We needed a way to playback the user's voice with the option of adding extra effects. We decided to use the module PizzicatoJS, which makes this possible with effects such as delay and reverb.

3.3 Research back end

For the research back end, we needed the following requirements: A database to store the data, an admin page behind a login for the researchers, and REST endpoints for posting data. Because of all these requirements, we recognized we needed to go to a high-level web framework. We decided on Django, python-based, because of previous positive experiences and ease of use. It is loosely is composed of two parts: The API and the Admin pages. The API is used for communication with the game, allowing it to post runs and sessions and retrieve that same data. The researchers use the Admin area to maintain, retrieve and combine the data in an effective way.

3.3.1 The Model

To efficiently store the data, the data is stored in a SQL database, with the following structure:

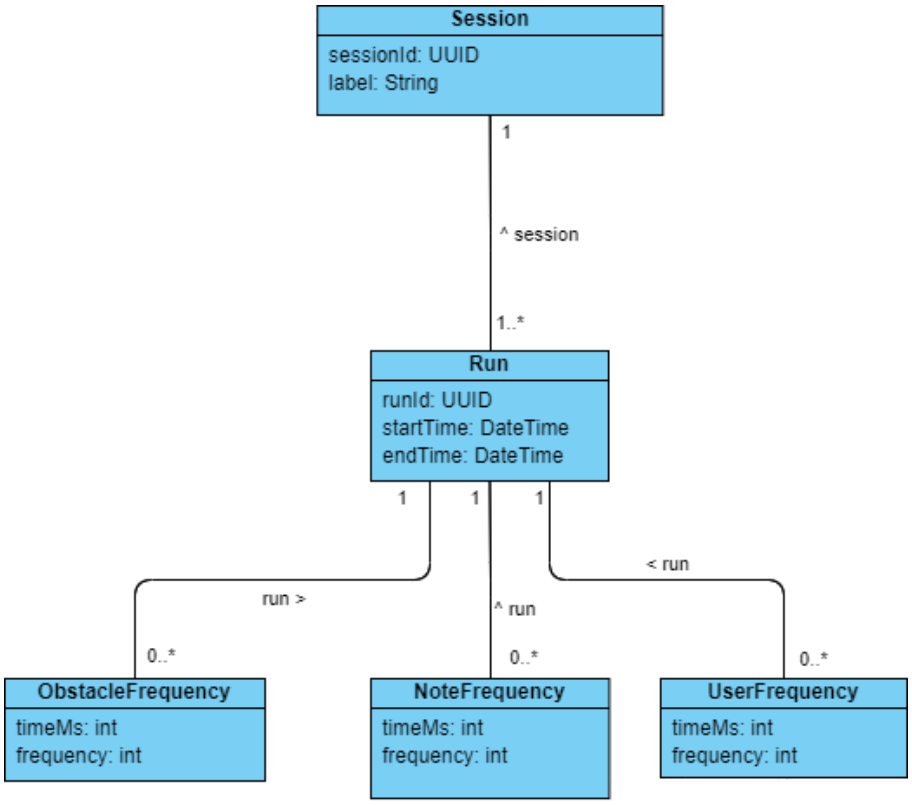

Figure 3.2: The class diagram of the research application

It consists of the following models:

- Session: Represents a session, which can have one or multiple runs. If the user does not
- Run: Represents one successful run of the application. It has a universally unique identifier runId
- ObstacleFrequency: Represents an obstacle that reached the player at time 'timeMs' with frequency 'frequency'
- NoteFrequency: Represents a note that reached the player at time 'timeMs' with frequency 'frequency'
- UserFrequency: Represents the user's perceived pitch 'frequency' at time 'timeMs'

3.3.2 The API

The front end can communicate with the back end through the API. Django-rest-framework is used for serializing the data and REST communication. The requirements dictated that data of all users should be collected, so we decided not to have authentication for the posting of run results. The client only requires to generate two UUID's, a 128-bit identifier which minimizes collisions to a negligible amount. A more specific API specification can be found at Appendix G

3.3.3 The Admin Section

The Admin section is created using the standard Django admin module. We added some extra functionality to it. The user can easily select and download all data from selected sessions and filter based on labels that can be specified in the front end application, i.e. all SoundLAB data can be filtered using the label 'soundlab'.

Chapter 4

User testing

To ensure our product is user friendly and appealing, we conducted user tests with both primary school children and our client. We will start by explaining our approach and afterwards discuss the results.

4.1 Preparatory research into UI design children

Children have different preferences in user interface designs than adults, and thus, research into what is pleasant for them was needed. Below we summarized the most important conclusions we have drawn.

4.1.1 Age is important

Since the physical and cognitive development of children goes very rapidly, the difference between a 6- and an 8-year-old can be quite a lot(J. Kientz, 2018). Since our main users are in middle primary school, we tried to focus our design on children around 8-10.

4.1.2 Bright colours, fun characters

Children between 7-10 are in a development stage where they are discovering what they like by trying out lots of activities and looking at many different colours. They are in the "I am not a baby"-stage and therefore designs for them can have quite extensive colours and graphics (Naranjo-Bock, 2011). We will make sure to use a good combination of bright colours and detailed but still clear graphics for our user interface.

4.1.3 Animations and Sound effects

Sound and Animations are usually disliked by adults making a website way too busy and distracting. However, from research, it is concluded that children do like this and even expect sites to have it. In a study, some children even were searching several minutes how to turn on the sounds so they could enjoy playing the game more(K. Sherwin, 2019). Therefore, we will add a slight sound and animation to each button and added multiple animations and sound effects to the game itself.

4.1.4 Clear goal

When playing a game, children want to have a clear goal of how to win and what the result will be (Liu, 2018). Therefore, we had to develop a storyline of why they were travelling through all the different environments and goals.

After considering these main points, we still had some questions about the design preferences, which we wanted to test.

4.2 Test plan

Before we could start with user testing, we created a detailed plan of what we wanted to test. This was especially important for testing with children since it had to be approved by their primary school. In our project proposal, we hoped to get two opportunities, but this was sadly impossible due to the coronavirus. However, we did get the green light to test with group six at a primary school in week eight.

4.2.1 Children

Testing with children was precious for us. The paragraphs below describe the used set-up and topics we wanted to test.

The test set-up

Only one of our group members was allowed to go to the school to minimize the possibility of infection with the coronavirus. The test took place outside the classroom, and two children at a time were asked to participate. In total, we got to test the application with 12 pairs, thus 24 children.

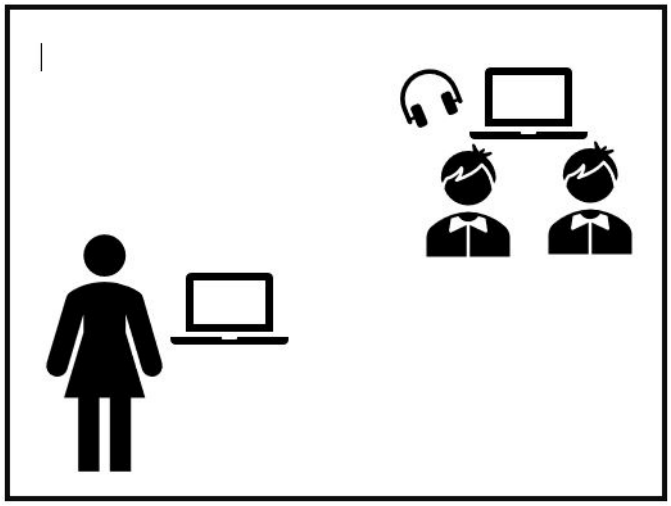

Figure 4.1: The set up used during user testing

The game was opened on a laptop at the start page when the children arrived, and our project member just asked them to start playing the game without any explanation. They used a headphone with a microphone to get the best results and not distract the others around the room. We let them play the first three levels, designed explicitly for this user testing. See Appendix C.1 for an example of these. Based on our previously done research and questions we ran into during the implementation of the game, we wanted to test the following topics.

Topics tested

- Intuitive Design: How logical it is for children to start using their voice?
- Background: Do they prefer photo's with the game items drawn over them or a drawn background withdrawn items?
- Color theme: Since children like bright colours in the design, we designed two colour schemes and wanted to know their preference.
- Collection object: Are coins or notes more fun to collect? Moreover, which do they think fit better in the game?
- Acoustics: Do they notice the difference in acoustics between themes?
- Other: What else do they like/dislike?

We translated these topics into the questions mentioned in appendix C.2 en asked them to the children after they were done playing the three levels.

4.2.2 Client

During our weekly meetings with our client, we always showed our current prototype and got feedback. Nevertheless, to test the game as it would be used in SoundLAB Enschede, we needed a classroom setting. In the ninth week of the project, one of our clients could arrange such a room in the Wilmink theatre. There were several important points we wanted to focus on during testing.

Classroom setting

Since we had only played the game at our homes on our laptops, we were very curious about how it would feel playing the game in a classroom and if there would be problems we had not thought of before.

Settings menu

We had developed a pop-out menu that could be used on a second screen while playing the game on an interactive whiteboard. During this testing, we wanted to discuss whether the setting was intuitive and see what improvements could be made.

Fine tuning

Since we had weekly meetings with our client, it took a week to get feedback again on minor fixes. By meeting physically, we could easily document their feedback and also adapt the setting on the spot. This way, the fine-tuning of, for example, the default settings of the game was done much more straightforward.

4.3 Test results

Both of the conducted tests provided us with much helpful feedback. In the following paragraphs, we will elaborate on the positive feedback and improvement points that resulted from the tests.

4.3.1 Children

Our topics mentioned above were tested with twelve pairs of children. In the table below we provide a clear overview of the results, for a detailed overview see appendix C.3.

Tested Topic	**Results**
Started singing from the beginning	3/12 yes, 9/12 no
Preferred photo or drawing	5/12 Photo, 6/12 Both, 1/12 Unclear
Collect coins or notes	6/12 notes, 2/12 coins, 4/12 both
Color theme preference	2/12 blue, 2/12 purple, 7/12 No preference, 1/12 unclear

We will discuss the conclusions of each topic shortly and mention the approach used if something needed improvement

Intuitive Design

Only three out of two groups immediately noticed that they needed to start using their voice. The other nine groups either used the arrows or mouse pad to try moving the player. Some children also did start using their voice, but the threshold was too high for them to get the player moving. Therefore we adjusted this to be lower. Children should now notice the player going up en down when talking about what they need to do.

When discussing this problem with our client, he said not to worry about it since the used game will be introduced at SoundLAB. Furthermore, the story we added later is related to singing (finding her golden tuning fork, the player having a microphone) to help children in the right direction.

Background

Since five out of twelve groups preferred the photo background and none preferred only the drawing, we decided to go with the photos.

Color theme

We tested both a blue and a purple colour theme, but both did not get much preference. We, therefore, decided to go for the purple theme since this was more appealing to adults and our client.

Collection object

Most children had a preference for collecting notes. The main reason mentioned was that it fitted the game more since it had a direct correlation with singing.

General Feedback

Besides the topics we prepared, we also received and observed some general feedback points:

- The 'Home' button was in English, which they did not understand. We adapted this and later on added the option to switch between Dutch and English for international schools or events.
- Children sometimes did not notice when they died and thought they had won when the game stopped. Since we added the storyline of finding the golden tuning fork and customized the obstacles per theme, this should be clearer.
- Many children seemed to think that singing louder would also raise the player higher. After a while, they noticed singing higher worked better. Since the game will be introduced in SoundLAB, this will also be explained.
- All the children were very enthusiastic when leaving the test session, and some even asked if they could play it at home. This was very nice to see and also motivating to keep working on finishing a well functioning product.

As can be concluded from the points mentioned above, this test session was very valuable.

4.3.2 Client

In a session of about 1,5 hour, we talked our client through the entire product. We also focused on the points mentioned in paragraph 6.2.2. The following points were observed.

Classroom Setting

In this setting, the game could be played on the interactive whiteboard with the use of a directional microphone. We noticed during play that sometimes a feedback loop occurred, which kept getting louder and was very annoying. This was not a problem before since we had used headphones at our homes. After observing, we noticed that the microphone gain was automatically increased when we stopped singing and became sensitive to noise. // The days after the testing, we looked into this and found out that there must be some program that automatically increased the gain control. We managed to turn this off in our program, and now the feedback is decreased to a minimum, and if it occurs, it stops rather quickly.

Settings Menu

We talked to our client through the implemented settings menu and received feedback that the names of the settings should be more understandable and that a small explanation about each of them would be excellent. The interface was also not user friendly enough. We processed all these points and, in our last weekly meeting, got positive feedback on them.

Fine Tuning

While discussing the settings, we decided with our client which default settings would be best for the game.

In general, our client and we were delighted with how the game was looking already during testing.

Chapter 5

Application Testing

To make sure our application was well functioning, both the individual processes and the system as a whole, we came up with a test plan for unit and system testing. Our approach and its results are discussed below.

5.1 Test plan

In the paragraphs below, our focus points for unit and system testing are described.

5.1.1 Unit tests

Unit tests are difficult to create for the project, as most aspects need functionality outside of the scope of unit testing frameworks, such as 2d rendering or audio contexts. However, the game engine abstraction that was created is unit tested, as this did not require any of those contexts.

#	Test name	Test Description	Expected Result	Test Class
0	add/removeChild	Tests if the addChild and removeChild function works appropiately by having a global number x, which is increased by the nodes' process function. Nodes are then added and removed and the ticker is called multiple times to increment the x value	The number x should be equal to the amount of nodes that were active when 'process' was called, in this case '4'	node.test.ts
1	pause/unpause	Tests the pause and unpause functionality of Node, which causes the process function to not be updated anymore. A node is added to the game tree with a process function which increases the variable 'x'. The ticker is then called, which should cause all unpaused nodes to call their process function, this is repeated with the node paused.	The variable x should increase when the node is unpaused and not when it is paused	node.test.ts

2	constructor	Tests the node constructor, which can be supplied nodes that automatically are added as children. A main node is created with node 1 and 2 supplied in the constructor	Node 1 and 2 should be a child of main node	node.test.ts
3	stress test	Creates 100 nodes and adds them to the ticker. The last execution time of the ticker is then measured	The time of the last update of the ticker should roughly equal 1/60 seconds	node.test.ts
4	animate	Tests the animate function of the tween. Animates a property and calls the ticker once	The animated property should increase by an amount	tween.test.ts

5.1.2 System and Integration Tests

For the system as a whole and the integration of separate parts together, we had the following focus points to test:

The game

For the game to be reliable and doable, we wanted to test the following points:

- Do the acoustics switch at the same time as the themes.
- Do the obstacles and notes spawn as they are supposed to: within the screen width, doable distance and not overlapping.
- Does the player die when moving out of the screen.
- Does the player win when successfully flying through the levels.
- Does the score increase when singing.
- Does the score not increase when the player is quiet and hits a note.

The real-time settings menu

Since the settings menu has to work in real-time during the gameplay, we wanted to test if this functioned well and did not break anything; we focused on the following test points:

- Does the game real-time adapt to the change in settings.
- When using the pop-out functionality, do the settings also change in the regular setting menu.
- Do all of the game elements function well during adaption of the settings.

Accurate research data

After the implementation of the research back end, we wanted to test if the data that could be analysed and downloaded there was accurate and appeared on the screen. Therefore we focused our tests on the following:

- Does the graph shown an accurate representation of the sang route.
- Do all the notes and obstacles appear in the graph.
- Does the session always appear on the back end.
- Does the redirecting work well.

5.2 Test results

When testing we came across some problems. Below we will describe which tests were successful and which pointed us to improvement points.

5.2.1 Unit tests

The unit testing of the game engine abstraction was very useful, finding bugs that would be very difficult to find if implemented in the project. These unit tests covered 90% of the classes tested.

5.2.2 System and Integration Tests

Separated by their focus group the results of the results of the system tests are discussed below.

The game

For the game to be reliable and doable we wanted to test the following points:

- Do the acoustics switch at the same time as the themes.
  ✓ We opened the game and while going through the portal we noticed that the acoustics changed. An improvement point was that the sound volume could be increased to make the change more clear.

- Do the obstacles and notes spawn as they are supposed to: within the screen width, doable distance and not overlapping.
  ✗ Sometimes during playing the obstacles and notes overlapped. We implemented a collision function for this that detects if they overlap and if that is the case delete the obstacle. Furthermore we once had the issue that notes spawned outside the screen, this was due to a mistake of the default range being 80 instead of 0,8.

- Do you die when moving out of the screen.
  ✓ When pushed out of the screen by an obstacle the game goes to the game-over screen. At no other scenario the player dies.

- Do you win when successfully flying through the levels.
  ✓ When passing the finish line the user is always redirected to the winning screen. There is no other way to win.

- Does the score increase when singing .
  ✓ When collecting a note the score increases.

- Does the score not increase when you are quiet and hit a note.
  ✓ When the user is not singing the player starts flickering and when flying over a note no points are added to the score.

The real-time settings menu

Since the settings menu has to work real-time during the game play we wanted to test if this functioned well and did not break anything, we focused on the following test points:

- Does the game real-time adapt to the change in settings.
  ✓ When adapting the settings on the pop-out menu during the game these changes are seen in the game. All the settings can be adapted.

- When using the pop-out functionality the settings also change in the regular setting menu.
  ✓ When sliding in the pop-out the slides also move in the regular settings menu and the other way around.

- Do all of the game elements function well during adaption of the settings.
  ✗ When adapting the tempo of the game we noticed some problems with the timing of the other game elements like the finish line spawning too late. We had to make sure this was updated every were in the software.

Accurate research data

After the implementation of the research back end we wanted to test if the data that could be analysed and downloaded there was accurate and appeared on the screen. Therefore we focused our tests on the following:

- Does the graph shown an accurate representation of the sang route.
  ✓ The graph shows accurately when the pitch was increased or decreased.

- Do all the notes and obstacles appear in the graph.
  ✓ All the obstacles and notes shown during the generated level are visible in the graph.

- Does the session always appear on the back end.
  ✓ The session always appeared.

- Does the redirecting work well.
  ✓ When using a label for the session this appeared along side the session ID in the back end.

Overall we played our game a lot during implementing which was always useful to notice bugs that occurred during development. When something in the game broke we could usually figure out quite easy from the console in which part something went wrong .

Chapter 6

Evaluation

When starting this project, we had certain expectations of working together, our planning and the requirements. Looking back, we can evaluate whether these were accurate and if we ran into situations we could have prepared for.

6.1 Team work

During this project, we worked a lot together and had to communicate with our client. In the paragraphs below, we will evaluate our chosen approach, how to communicate, and if we met all our requirements.

6.1.1 Approach

At the beginning of the module, we as a group decided to make use of the scrum principle to work on the product. After our weekly meetings with our client, we had a daily stand-up and created a new sprint in which we kept track of our progress in Trello. Because of this, we had a lot of contact moments with each other and our client. We think this was the right approach since problems could easily be discussed, and we kept track of each others' progress. An advantage might have been that we already knew each other and that some of us already had done previous projects together as well. Because of this we already had an idea of what working together would be like. Some sprints turned out to be more productive than others, but since we always kept good contact, this was handled well as a group.

6.1.2 Communication with client

In our weekly meetings with our client, we had the goal to communicate what we had achieved in the previous week and to receive feedback from them. In the beginning, we had some trouble in taking the lead in this, as we also heard after a few meeting from our client. After this, we started to make a small agenda for ourselves of what we wanted to discuss and ask. We noticed this worked really well and got positive feedback from our client after changing our approach. We received another point of feedback in the last week because we sometimes used too technical language to describe a solution or problem. We tried not to do this during the module but will take it into account for further projects.

6.1.3 Responsibilities

At the start of the project, we divided some main tasks between our group members: Scrum master, communication with client and supervisor and quality control. Since we all had different interest during the project, tasks were divided between group members. In total, the main responsibilities were divided as listed below:

- Wouter: Quality control, Interface Designer & Developer, Menu functionalities
- Carlijn: Communication with client and supervisor, Voice modulation, Interface Designer & Developer, Poster Design
- Michael: Scrum Master, Research Back-end Developer and Game Engine Designer
- Wesley: Quality control, Game logic Designer & Developer, Settings implementation

Of course, we tried to ensure that our contributions to the development of the product and this Design report were divided equally.

6.2 Planning

Our original planning can be seen in appendix D. For the most part this was quite accurate. We hoped to do user testing in an earlier stage of development. Since this was not possible mainly our user interface design took longer then expected. We did eventually got help from a professional animator and managed to improve our design a lot in the last two weeks. We planned on implementing the logging functionality in week 6 but finishing other parts took longer than expected. Luckily we had two weeks planned for adding improvements so we had enough time so move our planning a bit.

6.3 Results

During the sprints, we tried to implement as many requirements as possible. Below we will evaluate the results of the essential requirements based on the subsections of requirements.

6.3.1 Musical

Motivating the child to use their voice was significant for our product. We managed to add the game element of collecting points when singing at the correct pitch. As an extra motivating gravity was added, and no points can be collected while being quiet. We did manage to implement all the must requirements, but none of the should and could.

6.3.2 Theming and environments

Hearing a difference in acoustics when switching between environments was an essential part since this would be the learning aspect in SoundLAB Enschede. We managed to implement different themes in levels with corresponding acoustics and therefore accomplished all the requirements.

6.3.3 Ease of use

The game should be easy to use for both children and teachers. We did conduct user testing and listened well to the feedback of our client. Therefore we think we managed to implement all the corresponding requirements.

6.3.4 Research

For researchers, only one thing was paramount: proper logging functionality. We implemented a fully functioning back end where data can be downloaded and analysed to meet this requirement.

6.3.5 System

For the system, many requirements were significant. We managed to implement the ones related to reliability, ease of instalment, and the possibility of having multiple people use the application simultaneously. There was one requirement that stated that the application should work on tables, chrome books and laptops. Since our game only works well in Chrome, we implemented a narrow version of that requirement.

Overall, we succeeded in implementing all must requirements. Of all the topics discussed above, only for the Musical elements we had should, could and would requirements left.

Chapter 7

Suggestions for further development

Since our project is limited to ten weeks, we did not manage to implement everything we thought of during development. Therefore we would like to suggest improvements to future developers of the game.

7.1 Remaining musical requirements

Since we did not have time to implement all our requirements, we suggest these could still be done. That would include the following:

- Let the visuals correspond to the loudness of someone's voice.
- Motivate the user to sustain a note/notes.
- Offer the possibility to upload a MIDI file for the game input.
- Make the levels sound more like music.

7.2 Automatic adaption of settings

It would be interesting to use a form of machine learning on the singing capability of a person. If you notice that they have trouble reaching the highest notes, adapt the highest frequency automatically and vice versa. A less advanced way could probably be done using some logic.

7.3 Different note generation

There is much opportunity for generating notes differently. Since notes are generated dynamically as the game progresses, it could be possible to have levels get gradually harder, for instance, or to have more parameters to customize the ways notes are generated. Currently, also, the scale note generator only generates a predefined hard-coded scale. It would be nice to be able to generate scales in different musical keys.

7.4 Improved obstacle generation

Currently, obstacles and notes can be generated at the same time and height. This makes collecting the note impossible, which can be frustrating. It would be nice to have obstacles only generate in places where there is no note.

7.5 Custom frequencies per person

To let a person quickly set the correct frequencies for them, add options to let them sing their highest and lowest note and adapt the settings to this.

7.6 Tutorial level

For people playing the game for the first time (without introduction at SoundLAB), it can be nice to have a tutorial level that takes them through all the critical parts of the game and maybe even points out the educational purposes of it.

7.7 Level upload

Currently, the levels are loaded from a server together with the rest of the game. It could be possible to have users upload a JSON file with a description of a level to be more customizable.

7.8 iPad support

Since the game uses some browser features currently not supported on the mobile version of Safari, it is impossible to play the game on iPads. It might be possible to adapt the game so it can be playable on iPads. However, mobile Safari will likely support the game anyway, as these browser features are added.

Chapter 8

Closing

This Design Project aimed to deliver an excellent contribution to SoundLAB Enschede by adding an interactive singing game. It should improve the previously designed prototype by being easier to use and offering more functionality. After ten weeks, we provided our client with a working system that meets most of the requirements, and we are proud of the result.

Furthermore, we gained more insight into the development process of an extensive product and learned more about its phases and worked in a team for an extended period. This will all be valuable in future projects.

8.1 Acknowledgements

We want to thank Benno Spieker and Laura Slakhorst for the great collaboration during the entire module. Second, special thanks go to Thijs Steggink for helping us with designing the story and appearance of the application.

Bibliography

J. Kientz, L. Anthony, A. H. (2018). Playful interfaces: Designing interactive experiences for children.

K. Sherwin, J. N. (2019). Children's ux: Usability issues in designing for young people. *Nielsen Normal Group*.

Liu, F. (2018). Designing for kids: Cognitive considerations. *Nielsen Normal Group*.

McLeod, P. and Wyvill, G. (2005). A smarter way to find pitch. In *ICMC*, volume 5, pages 138–141.

Naranjo-Bock, C. (2011). Effective use of color and graphics in applications for children, part ii: Kids 7 to 14 years of age. *User Experience Matters*.

van Soelen, R. (2020). The singing game. *Internship report SoundLAB*.

Appendix A

Manual

This manual will explain how to use or adapt every aspect of the application. A division is made between the user manual and the maintainer manual. The user manual explains how users can correctly play the game, how the settings can be adapted in the game, and how the research data can be collected and extracted from the game. The maintainer manual gives a detailed explanation of how the maintainer, SoundLAB Enschede, can host the complete application or adapt some features of the game in a later phase.

A.1 User manual

The user manual is divided into three subsections; game, settings, and research. These sections describe how the different users can use the application in an intended way. The game section describes how children can play the game, the settings section describes how teachers or SoundLAB employees can adapt the game for individuals, and the research section describes how researchers can extract the preferred data.

A.1.1 Game

The main user group of our application is primary school children. Children can play the game during their visit to SoundLAB Enschede or during a series of lessons about acoustics at their primary school. Because this is such an important user group, and because they generally only want to quickly play the game and have fun, the related design has been kept as simple and straight to the point as possible.

Main menu

When a user starts the application, the first screen they see is the main menu (figure A.1). This screen gives a first impression of the goal of the game. And it contains two buttons to guide you to different menus. When you want to play the game, press button 1. If you want to have the application in full screen, you can do this by pressing F11 on your keyboard. As the application doesn't automatically re-scale, you will need to restart the application by pressing F5 on your keyboard. When the application is restarted, the application will always open at the main menu.

Figure A.1: The main menu of the application

Level selection menu

After pressing button 1 on the main menu screen, the level selection menu (figure A.2) opens. This screen contains five buttons. Buttons 1 to 3 offer an option to select which level you want to play. The only difference between the levels is the total amount of time it takes to complete the level. On default, the medium or, if applicable, the previously played level is selected. After selecting your preferred level, you can begin playing the game by pressing button 4, start. Button 5 can be used to go back to the main menu.

Figure A.2: The level selection menu

Game

After pressing button 4 in the level selection menu, the game (figure A.3) is started. While playing the game, there are multiple important aspects to pay attention to.

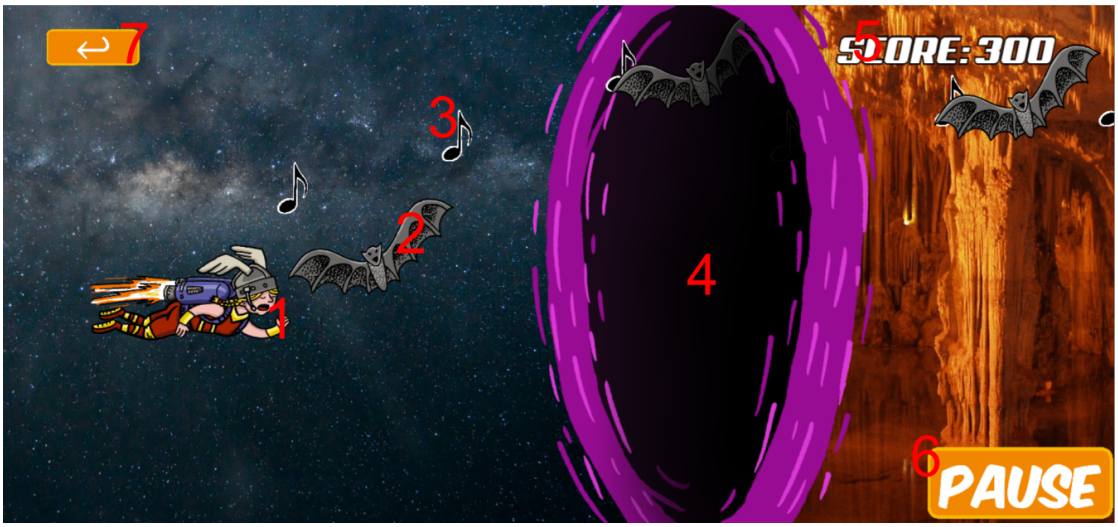

Figure A.3: The application interface while playing a game

1. The soprano; by singing in your microphone, you can control the soprano. While all other game aspects move to the left, the soprano remains on the left side of the screen. If you stop singing into the microphone, the soprano will start to blink and slowly fall to the bottom of the screen.

2. Obstacles; in all environments, obstacles are coming towards you. You should try to avoid these obstacles with the soprano because if an obstacle completely pushes you out of the screen, you failed the level. When an obstacle only pushes you back a small bit, but you manage to escape by singing in a different pitch, the soprano automatically flies back forward. Obstacles differ per environment; the different obstacles are:

    - Comets in the space environment
    - Bats in the cave environment
    - Thunderclouds in the sky environment
    - Sea mines in the underwater environment
    - Eagles in the mountain environment

3. Notes; notes are constantly coming towards you at different heights. Unlike obstacles, you should aim to collect as many notes as possible. When collecting notes, you earn more points, and you are often safe from obstacles. However, you can not collect coins when your soprano is blinking, indicating that you are not singing.

4. Portals; by going through portals with your soprano, you can enter different environments. When going through a portal, the soprano makes a small sound and animation. Every environment also has different sound effects. If you pay attention to the sounds, you will notice that you can hear your own voice back with the same effect as you would hear it in the same type of environment in real life.

5. Score; in the top right corner of the screen, your current score is displayed. It is increased by 100 points for each collected note.

6. Pause; with the pause button, the entire game can be paused, maybe if you need to catch a breath or just want to take a quick break. The game can also be paused by pressing the space button on your keyboard.

7. Back; with the back button, you can return to the level selection menu and select another new level to play or start the same one again.

Game finish

At the end of the level (figure A.4), two extra game objects will appear, the finish line (1), and the golden tuning fork (2). After crossing the finish line, the level is completed, and the soprano happily flies off with her golden tuning fork.

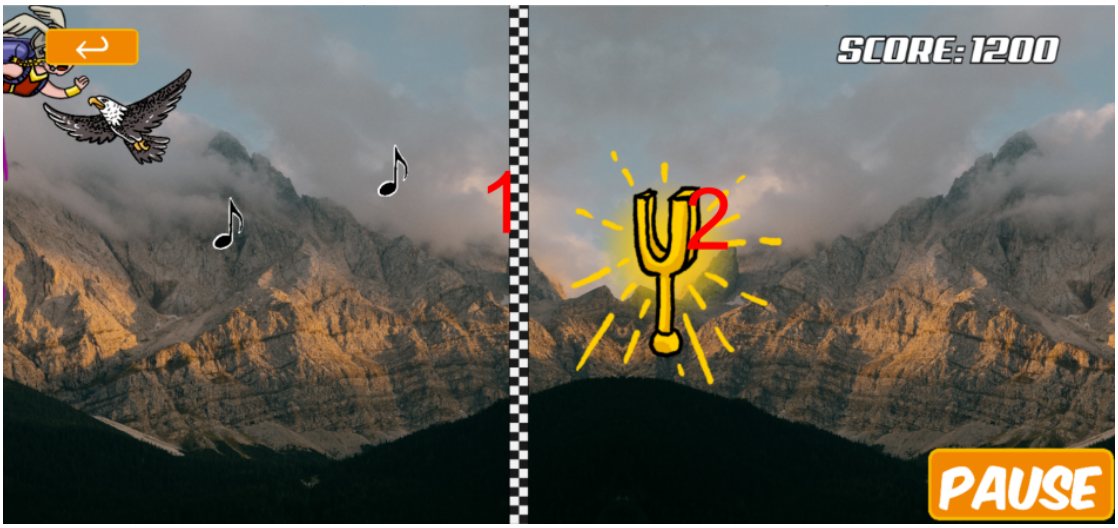

Figure A.4: The application interface at the end of a game

End of level overlay

After crossing the finish line or getting pushed out of the screen by an obstacle, the end of the level overlay (figure A.5) will appear. This screen indicates whether you made it or not. From here, you can go back to the level selection menu with button 1. From here, you can select a new level and start again, or maybe quit playing. With button 2, the same level you just played will start again, in case you don't want to play a different level. Lastly, at place 3, your total amount of points acquired in the last level is shown on this screen.

Figure A.5: The end of level overlay

This concludes the user manual for a user who just wants to play the game as is. Using this detailed manual, you should be able to select start the game, select your preferred level, and play the game without any problems.

A.1.2 Settings

Next to just playing the game directly, it is also possible to adjust the game's settings before or while playing the game. One reason why you might want to do this is that someone can't reach most of the notes with their voice or because the game is not challenging enough with its default settings. This can be done in the settings menus.

Settings menu

To open the settings menu, you need to press button 2 in the main menu (figure A.1). This menu contains a lot of buttons. Most of them change a different game setting. All edited settings are saved immediately, so the next time you open the application, your adjusted settings will still apply. The following list gives a quick overview of the effect and use of each setting.

1. Tempo; the number of seconds that it takes for an obstacle or note to get to the soprano. Setting the tempo lower generally makes it more difficult to avoid obstacles and collect notes.

2. Break length; the amount of seconds a break takes. During a break, no obstacles and notes are deployed, so you have some time to catch your breath.

3. Voice; select a predefined voice range. All notes and obstacles will be spawned in frequencies within the selected range. Clicking on the voice setting switches it to the next voice setting. The different voice settings are:

    - Child: 3-6 years; E4-D5
    - Child: 7-9 years; C4-E5
    - Child: 10-12 years; A3-F4
    - Bass; E2-E4
    - Baritone; A2-A4
    - Tenor; C3-C5
    - Alto; F3-F5
    - Mezzo; A3-A5
    - Soprano; C4-C6
    - Custom; Uses a custom, user-defined voice range. This range can be set in the pop-out menu (figure A.7).

4. Inter-onset interval; the number of notes generated per minute. Setting the inter-onset interval higher makes the notes appear in a more constant stream.

5. Obstacle frequency; The probability of an obstacle spawning every x seconds. In principle, this simply means that with a probability of 1, there's a constant stream of obstacles, and with a probability of 0, there are no obstacles.

6. Range; the range of the voice setting that is used in percentages. For example, if the range is set to 0.5, only the middle half of the voice range is used by obstacles and notes. This setting can be used if the selected voice setting's range is slightly too broad for your voice.

7. Phrase length; the number of seconds that notes are generated before a new break comes. It can also be interpreted as the amount of time in between breaths.

8. Interval size; the average difference between the notes. With a very small interval size, all the note pitches will be relatively close to the previous note's pitch, making it easier to collect notes.

9. Language; Changes all the text in the application to the newly selected language. Currently supports English (EN) and Dutch (NL).

10. Pop-out; opens a new window where all settings can also be adjusted.

11. Back; this button takes you back to the main menu, from where you can start a new game with the newly adjusted settings.

Figure A.6: The settings menu

Pop out menu

When the pop-out button is pressed in the settings menu, a new second window is opened. This window contains the pop-out settings menu (figure A.7). In this menu, all settings can be seen and adjusted just as in the normal settings menu, but the pop-out menu offers some extra functionality. The most important aspect of the pop-out menu is that it's shown on a different screen, which can also be opened while playing the game. This gives the option to adjust the settings directly while playing, making it easier to tweak the settings perfectly. Furthermore, the following list gives a more detailed explanation of the different aspects of the pop-out menu.

1. Just as in the normal settings menu, most settings can be adjusted by simply moving a slider to the preferred values.

2. Next to moving a slider, all numeric values can also be entered directly in an input field. This is useful if you would like to play with very specific settings.

3. If you're wondering what a setting means,.hovering over the name of a setting in the pop-out menu shows a short explanation of the setting. If this explanation is not completely fulfilling, a more detailed explanation of each setting can be found in this manual.

4. The custom voice setting can be defined here. In the first field, you can enter the minimum pitch of a note, and in the second field, the maximum pitch can be set.

5. In the normal settings menu, different voice and language settings are selected by clicking through the list of options. In the pop-out menu, this can be done slightly clearer, using a simple drop-down menu.

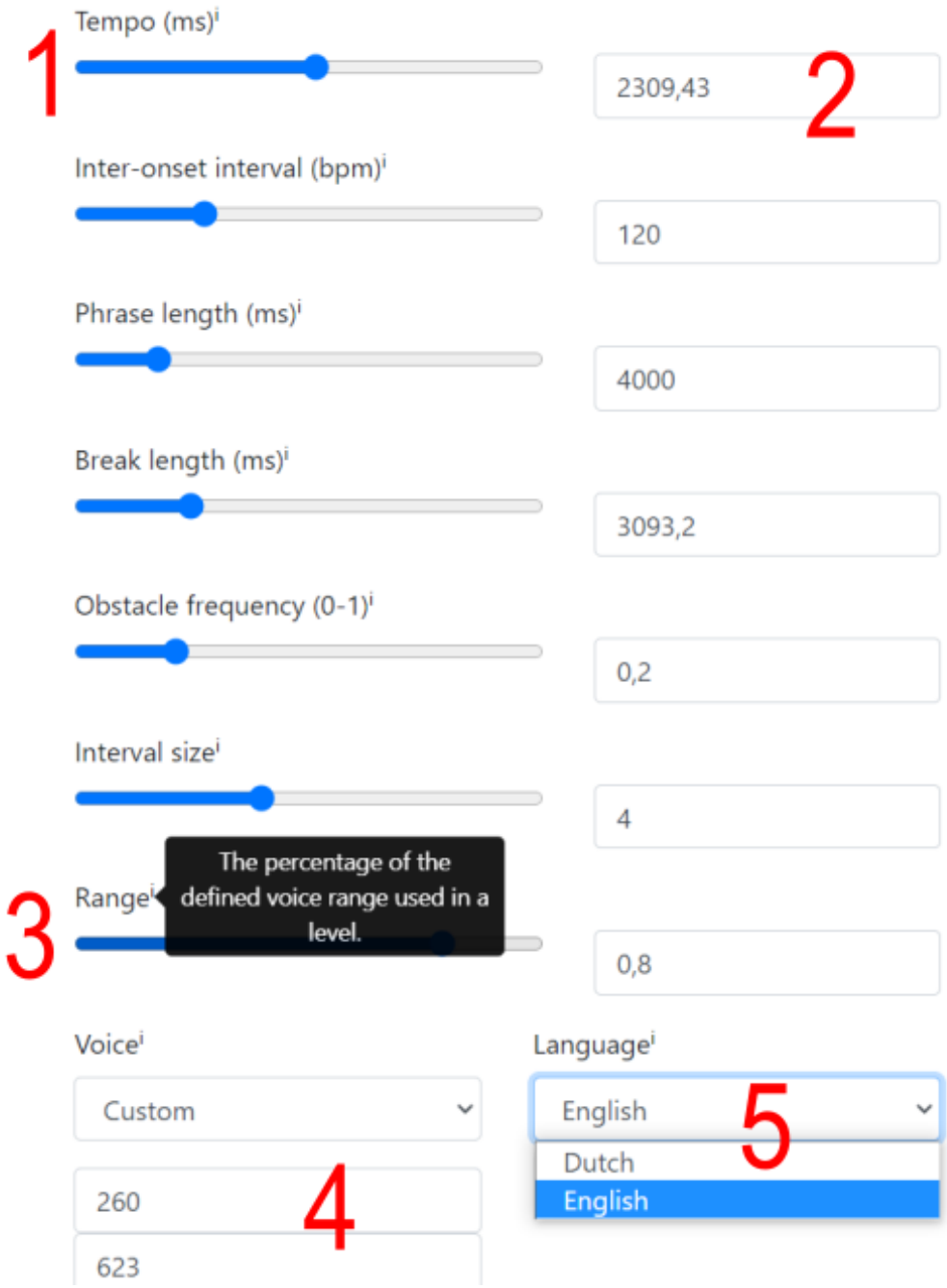

Figure A.7: The pop-out settings menu

A.1.3 Research

The last part of the application is the data collection and extraction application, from here on called the backend. The backend is not open to everyone and is password restricted to SoundLAB and whomever SoundLAB chooses to share their credentials with. When opening the backend, the login screen (figure A.8) is first shown. Here, you can log in with your user name and password.

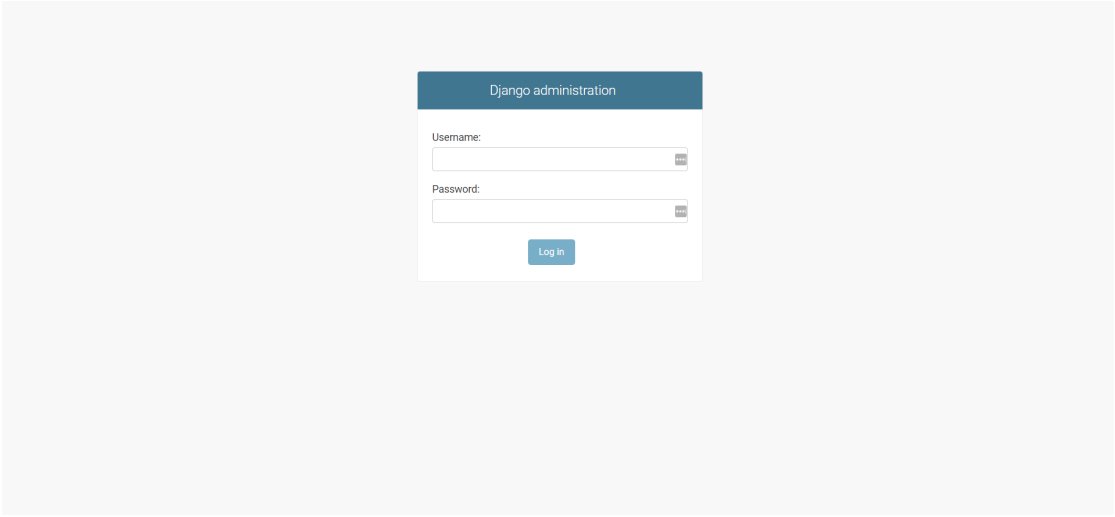

Figure A.8: The login page of the backend

Backend main screen

After logging in, the main page (figure A.9) is opened. This page offers quite some functionality, out of which some is redundant. When normally using the application, only buttons 2 and 3 need to be used. The buttons at locations 1, 4, 5, and 6 are unnecessary and should not be used. With button 7, the password for the research account can be changed. And with button 8, you can log out of the backend when you are done. Buttons 2 and 3 give an overview of the individual game runs and the application sessions, respectively.

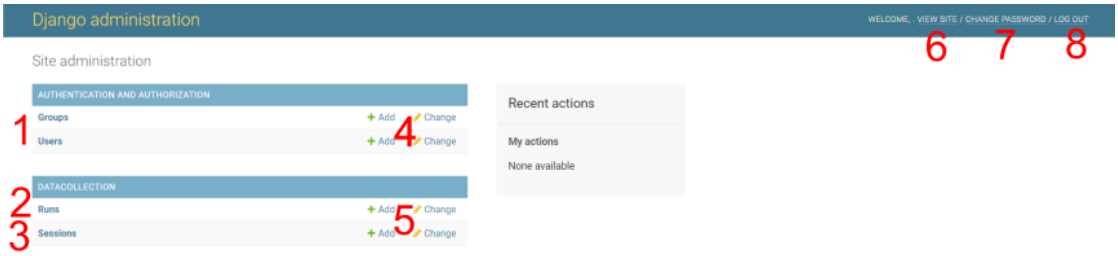

Figure A.9: The main menu of the backend

Backend runs and sessions pages

One can open the list of sessions by clicking on button 3 of figure A.9. This shows the screen in figure A.1.3.

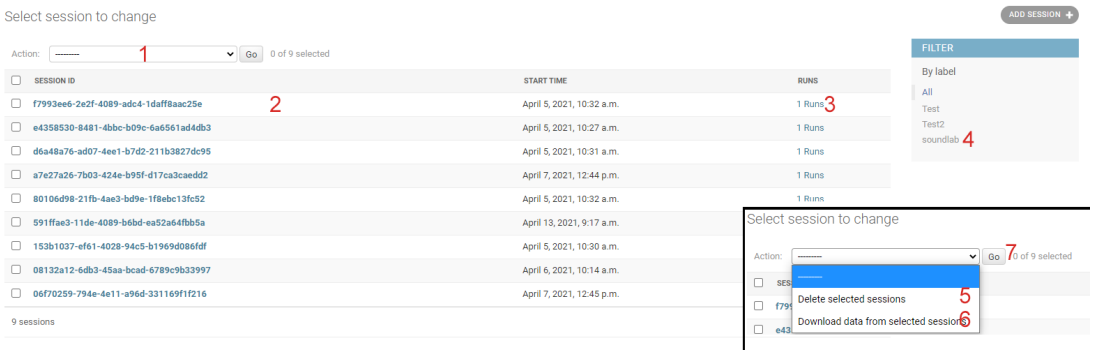

Figure A.10: Example page of session list

Filtering by data label The data can be filtered by labels given by the front end. This can be done by clicking on one of the names in the list underneath 'filter'. The SoundLAB data, for example, can be selected by pressing button 4, labelled 'soundlab'. This shows all sessions with the label 'soundlab'

Exporting session data Data from sessions can be exported by selecting them using the checkboxes on the left-hand side and clicking the drop-down labelled 'action' (button 1). One can then click "Download data from selected sessions" (button 6) and click "go" (button 7). This exports all runs from the selected sessions to a .json file and downloads them.

Showing runs from a session Clicking on the button "# Runs" (button 3) will redirect the user to the Runs list page, filtered by the runs from that session. On that screen, one can inspect all individual runs from that session.

Backend run list

One can open the list of runs by clicking on button 2 of figure A.9. This shows the screen in figure A.1.3.

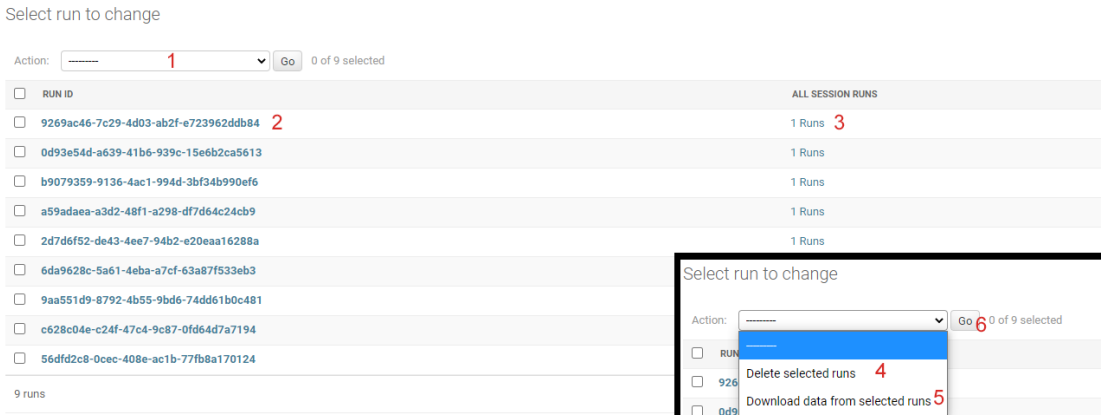

Figure A.11: Example page of run list

Exporting run data Data from runs can be exported by selecting them using the checkboxes on the left-hand side and clicking the drop-down labelled 'action' (button 1). One can then click "Download data from selected runs" (button 5) and click "go" (button 6). This exports all selected runs to a .json file and downloads them.

Selecting all runs from a session If one clicks on "# Runs" (button 3), it will go to a new screen, which shows all runs from the same session as the selected run.

Selecting a specific run Clicking on the UUID (button 2) opens the screen described in A.1.3

Backend run overview

When opening a single run, a page for that single run (figure A.12) is opened. This page gives detailed information about this specific run and implements functionality to download or adapt the data. The following list gives a short description of every aspect of this page.

Figure A.12: The overview page of a single run

1. This small menu contains the same information as a part of the main backend page. You can return to the main page by clicking on home or return to the runs or sessions overview pages.

2. The top right corner offers the exact same three buttons with the same functionality as on every page.

3. Using this button, the run data of this specific run can be downloaded in a file of a .json format. This file contains all information about this run, including lists with every, sang frequency, every deployed note, and every deployed obstacle. Using a simple statistical data processing application, this can easily be processed.

4. This table gives a visual overview of the run. It can be used to verify that you opened the correct run or give a first insight on how successful a run was.

5. An overview of some of the other details of the run, namely the session it belongs to, and the start and end date and time. This can also be edited here if necessary.

6. Using this button, the entire run can be deleted. For example, if you wish to analyze all runs in a certain session, except for this one. However, it is important to note that once a run has been deleted, it is not possible to retrieve the run again later.

7. If you have edited the values at 5, the made changes can be saved with any of the three buttons here. However, the most relevant one is the most right save button, after which this run is closed, and you return to the runs overview.

8. If you're looking at a run and wonder whether any of this data has been altered after being collected, you can show the run history here. By default, this page shows, "This object doesn't have a change history. It probably wasn't added via this admin site.", as the run objects are normally added automatically and not edited afterwards.

A.2 Maintainer manual

The maintainer manual is divided into two sub-sections. First, a detailed explanation will be given on how the application can be run locally and how the application can be hosted in a cloud environment. Next, the file structure of the application will be explained, including instructions on how certain aspects can be changed.

A.2.1 Running the application

The application consists of two parts, the frontend, being the singing game itself, and the backend, which handles the functionality for the research data extraction. These applications can be run separately, but in order to collect data, both need to be running correctly. The following two paragraphs will give a short explanation of how both parts of the application can be run for development on your machine. A more detailed explanation of how to run the application in production for both the frontend and the backend sides can be found in the README.md of the projects' root folders.

Fronted

To run the frontend in development, you need to have the following programmes installed: Node version $\geq$ 8.0 (recommended 10.6.0), NPM $\geq$ 5 (recommended 6.1.0) and Yarn. Afterwards, with the following commands, you can start the front end.

```
# go to the repo
cd singing-game

# install the dependencies via npm
yarn install

# start the server in dev mode with HMR
yarn start
```

After these commands are done, you should be able to open the application on `http://localhost:1234`

Backend

To run the backend in development, you need to install Docker and open the application in Docker. Here, you can start the backend using `docker-compose up --build -d`.

A.2.2 Adapting the application

The application has been structured such that any aspects that one might want to change, later on can easily be changed. Examples of this would be the text shown in the game, the images and drawings that are used, or the type and length of the available levels. Some other parts can be less easily modified, such as the actual game logic. However, in the case that SoundLAB ever wishes to alter this, thorough documentation was provided for all code.
When adapting any aspect of the application, there are two things that require attention in order to make the update completely successful. First, the version needs to be updated. The current version can be found in line 3 of the file `src/package.json`. By setting this version to a higher number, the application will know that changes have been made, so it has to reset everything on the first startup. Afterward, the server will have to be restarted in order to actually apply those changes to the online application.

Adding or editing textual contents

All textual contents of the game can be altered in one single translation file. Think of the titles of the pages, the contents of the text boxes and the text inside buttons. The only exception is the main title, "Planet Opera" as this is a custom drawing. This can still be changed, but how this can be done will be explained later.

For textual contents, all text can be found in the file: `src/app/menu/translations.json`. The translations file is structured in a very logical format, it contains different languages, and each language contains the same keys for each different text. The value after a key, which is the text displayed in the game, can be changed to any desired text using a simple text editor.

New languages can also be added very easily using this file, after Dutch and English, just by coping with another instance of "English" at the end and changing the name and all of the values. The language option will then automatically be added to the settings menu.

Adding or editing levels

Currently, the game contains three different levels; short, medium, and long. A level defines which themes occur in a level, in which order they occur, and how long they occur. The level definitions can be found in the file: `srs/app/level/levels.json`. For each level, a number of features are defined:

- Name: the key of the name is, this is the key for the translations in the translations file
- The note, obstacle, and break generators: Currently, obstacles can only be generated randomly, and breaks are always constant. The notes can be generated using KEY, RANDOM, or SCALE; out of these KEY is currently used, as it gives the most pleasant levels to play.
- Themes: This is a list that contains all themes in the level. For each theme, the start time and the theme should be specified. The options for the themes can be found in the `getThemeFromString()` function in `src/app/themes/Themes.ts`. The start time is the moment, in the number of seconds after starting the game, at which the theme should start. The theme ends when at the first theme with a higher start time.
- Time: the moment, in the number of seconds after starting the game, at which the level ends, and the user completes the game.

By simply changing any of the features explained above, the current levels can be altered quite a lot. To add a level, a new level can simply be added in the same JSON file. However, it would also need to be manually added to the level select menu.

Adding or editing themes

Themes are the combinations of backgrounds images with voice effects and obstacle images on a level. The different themes are defined in `src/app/themes/themes.ts`. The themes exist of:

- effect: the voice effect, defined with a name as is specified in the voice modulator.
- texture: the background image of the theme. Images are defined by exporting them in `src/assets/loader.js`. When adding a new texture, the image also has to be included in the `@registerAssets()` list in `src/app/themes/backgroundspritespool.ts`. Texture images are placed after one another in the background, so for an optimal background look, the images should be seamlessly tiling.
- obstacle: the image of the obstacles in this theme. When adding a new obstacle, the image also has to be included in the `@registerAssets()` list in `src/app/gameObjects/Obstacle.ts`.

By defining a new theme here and after setting its key in the `getThemeFromString()` function, the added or edited themes can directly be used in levels.

Adding or editing voice effects

Voice effects during themes are handled by the file: `src/app/audio/VoiceModulator.ts`. To add or edit a voice effect, e.g. when adding or editing a theme, its name can be defined in `switchEffect()`. After which, a function call should be added in the switch case statement. In return, this function should call `changeEffect()` with a PizzicatoJS effect. Detailed explanations of the different Pizzicato effects and their usages can be found in `https://github.com/alemangui/pizzicato#effects`.

Editing images

As quickly mentioned before, all images can be edited quite easily edited. When adding a completely new image, the image file should be included in multiple locations in the application. However, when replacing an already existing image, all of this is unnecessary. Currently, all images can be found in `src/assets/images/<folder>`. To replace any of these images, simply replacing it with another image file with the exact same name and file type will update it correctly in the application. Images are scaled to fit their locations later on, so there's no need to take the size of images into account. However, dimensions are not altered in the application, so images of different dimensions could appear different than expected.

Editing menu layouts

Menu layouts are easily adaptable as well. Each menu page has their own class in `src/app/menu/menus`. Menu's pages are formatted by creating a grid and adding all contents to the grid in the page's `enterTree()` function. The grid is a 12x12 division of the screen in which objects can be placed. The grid scales the objects based on the given width and places them in the specified row and column. If you want to remove one aspect of a menu page, such as the text box on the end of the level overlay, simply removing line 44 completely from `src/app/menu/menus/EndOfLevelOverlay.ts` removes it without breaking anything else.

Editing menu style

The three menu pages of the application are all in the same style; their background colour is the same, the buttons are the same shape and colour, and the titles are in the same font. If you want to change any of this, it can all be done by just adding it to one location.

- Background colour: the default background colour (3D348B) can be added in `src/app/menu/Background.ts` in line 23. Colours are encoded as "0x" followed by their hexadecimal colour code.
- Button colours: the colour of all buttons can be changed in `src/app/menu/buttons/Button.ts`. Line 52 defines the colour of the buttons, and line 53 defines the colour of the borders.
- Slider colours: the colour of the sliders of slider buttons can be changed in `src/app/menu/buttons/SliderButton.ts`. Lines 89 and 90 define the colour of the line it moves over, lines 121 and 122 define the colours of the slider itself.
- Button shape: the default shape of all buttons can be changed in `src/app/menu/buttons/Button.ts` in line 54. Currently, a rounded rectangle is used. This can be changed to any geometrical shape using PixiJS Graphics. A more detailed can be found here: `https://pixijs.download/dev/docs/PIXI.Graphics.html`.
- Title style: all titles, except for the main menu title, are white text with black borders. These colours can be altered in `src/app/menu/Title.ts` in line 31. In this same line, the font can be changed. Note that only some fonts are supported by default. If you want to use a custom font, the .tff files need to be added to `src/assets/fonts` and loaded in `src/main.css` and `src/index.html`.

Editing default settings

All settings in the settings menu have a default value. These are currently fine-tuned to an expectation of what is realistic and pleasant for the main user groups. However, if this expectation doesn't give the expected result, the default settings can be changed very simply. All of the default values of the settings can be found in the settings variable in `src/app/globals/SettingsGlobal.ts`. Changing any value here will set it as the new default value for that variable.

Editing the ranges of the predefined voice settings

Users have the option to select a predefined voice range by choosing it from a multiple-choice selection. If these ranges turn out to be unrealistic for what they represent, e.g. kids of 3-6 years old on average can't reach most of the notes in their voice range, these ranges can be edited easily. All of the predefined voice ranges can be found in the voices variable in `src/app/globals/SettingsGlobal.ts`. The pitch heights in this variable are measured in frequencies.

Appendix B

Activity diagram use in SoundLAB

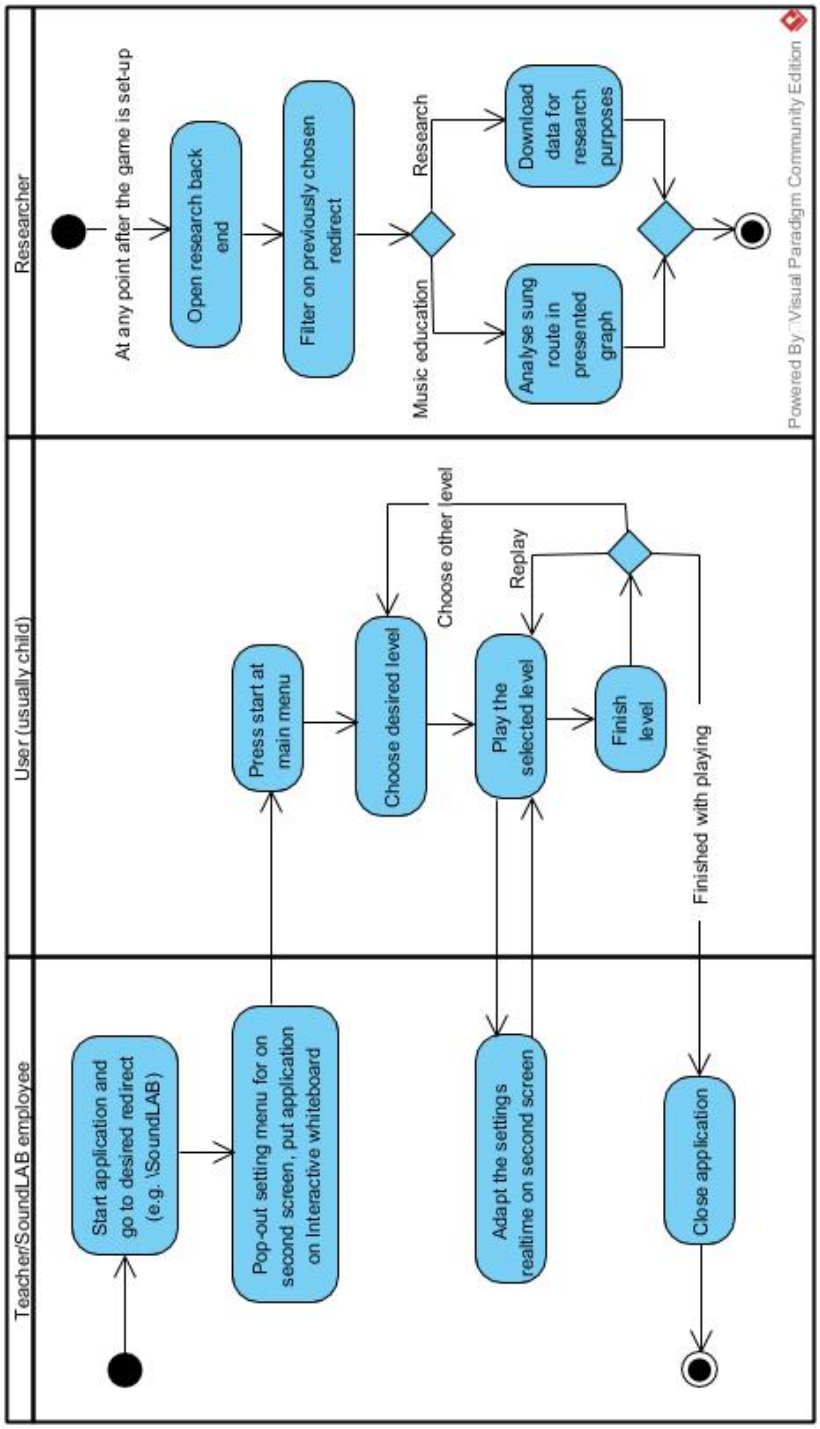

Figure B.1: Activity diagram use in SoundLAB

Appendix C

Overview Test Children

C.1 Designs used for testing

C.1.1 Background

We tested two different background options, both realistic and drawn. Below is an example of what the game looked like in the theme "sky" for both scenarios (figures C.1 and C.2).

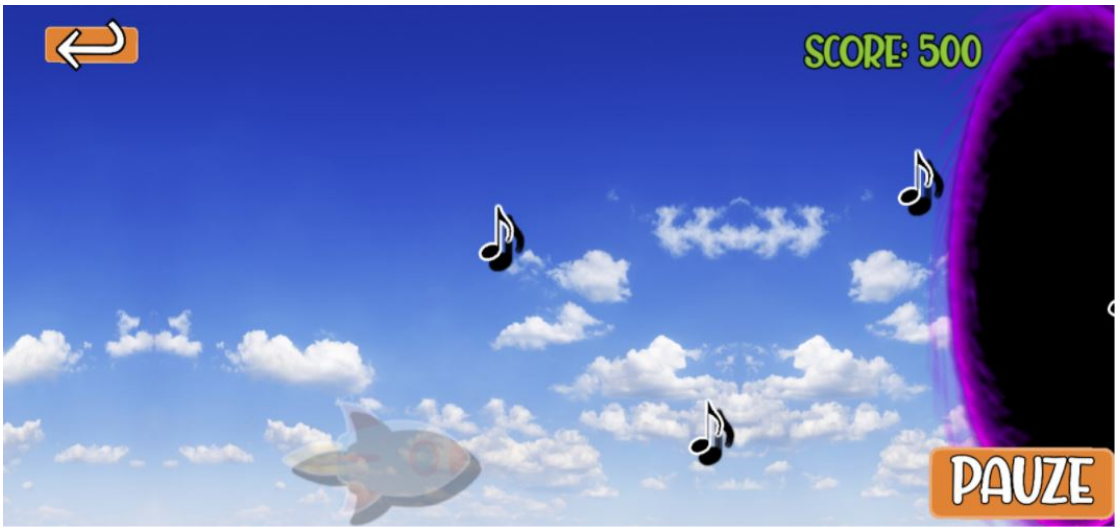

Figure C.1: Test with realistic background

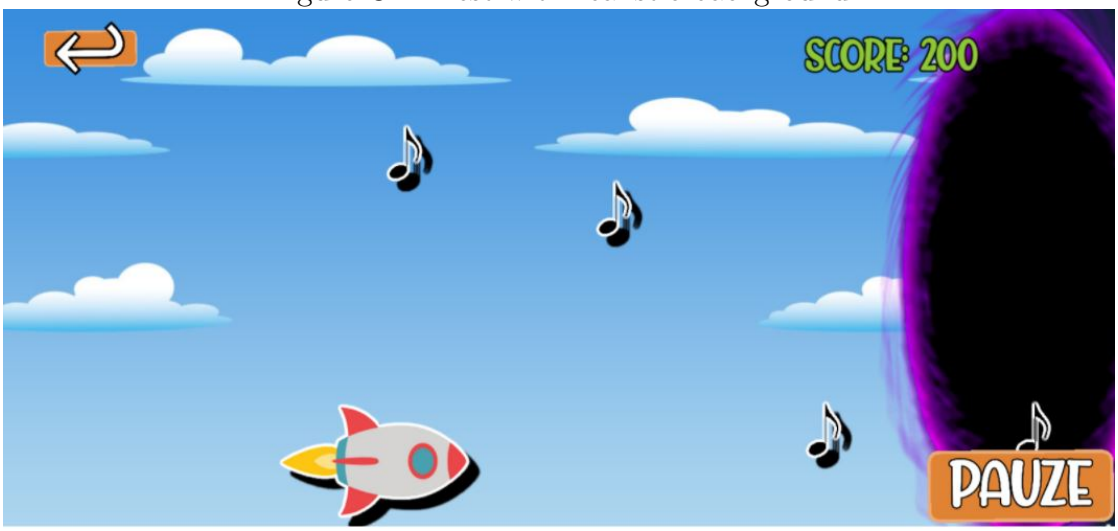

Figure C.2: Test with cartoon background

C.1.2 Note representations

We tested whether children preferred to collect notes or coins (figure C.3).

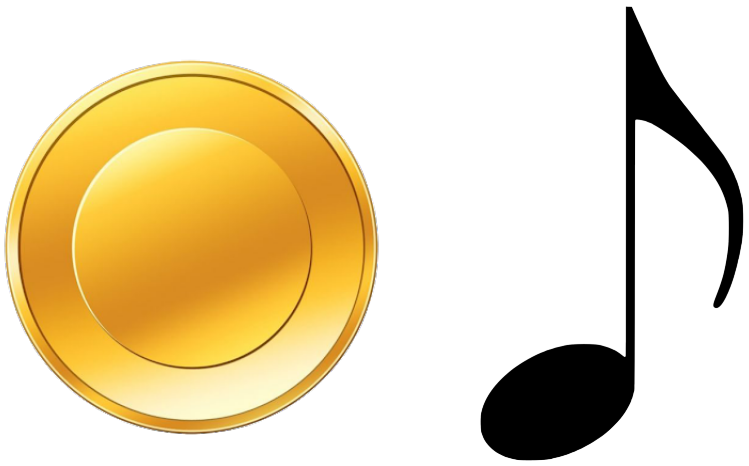

Figure C.3: Tested note and coin

C.2 Overview questions

After children were done playing the first three level we asked them the following questions:

1. Do you prefer the realistic or drawn version of the background?
2. Do you prefer to collect coins of notes?
3. Did you hear a difference in acoustic effects when switching between themes?
4. Do you have a preference for the different colors?
5. Do you have any other general remarks?

We did observe the following ourselves:

1. Did they notice quickly that they needed to use their voice?
2. Are they any other noticeable things?
3. Did they enjoy playing the game?

C.3 Test results

The following table provides an overview of the test results

Group	Singing from the start	Photo/ drawing	Notes/ Coins	Did notice acoustic	Color preference
1	Yes	Photo	Both, Coins	No	Purple
2	No	-	Notes	No	-
3	No	Photo	Both	-	No
4	No	Photo	Both	No	No
5	No	Both, Drawing	Coins	No	No
6	No	Both	Coins	No	No
7	No	Both	Notes	A bit	Purple
8	No	Photo	Notes	No	No
9	No	Combined	Notes	A bit	Blue
10	No	Photo	Both	No	No
11	Yes	Both	Notes	Yes	Blue
12	Yes	Both	Notes	No	No

Appendix D

Original planning

Project week	Goals	Deadlines
1 (01-02 to 05-02)	- First meeting with the client - Forming requirements - Making a general planning - Thinking about the software structure	
2 (08-02 to 12-02)	- Discussing the requirements with client - Improving the requirements - Implementing a basic design - Writing the project proposal	
3 (15-02 to 19-02)	- Finish first iteration (basic design) - Agree on project proposal and requirements - Write user test plan	Peer review on our project proposal and planning
(22-02 to 26-02)	*Spring break*	
4 (01-03 to 05-03)	- First meeting with our supervisor - Finish second iteration (well playable) - Finish general test plan	
5 (08-03 to 12-03)	- Finish third iteration (settings menu implemented) - Finish design diagrams	Peer review on our requirements and test plan
6 (15-03 to 19-03)	- Finish fourth iteration (logging functionality)	

7 (22-03 to 26-03)	- Finish fifth iteration (improvements)	Peer review on our fifth iteration
8 (29-03 to 01-04)	- Finish sixth iteration (improvements) - Thorough system testing - [If possible] user tests with children	
9 (06-04 to 09-04)	- Finish final product	Final product presentation
10 (12-04 to 16-04)	- Finish documentation	Present poster, design report, and deliver instruction manual

Appendix E

Game engine abstraction visual representation

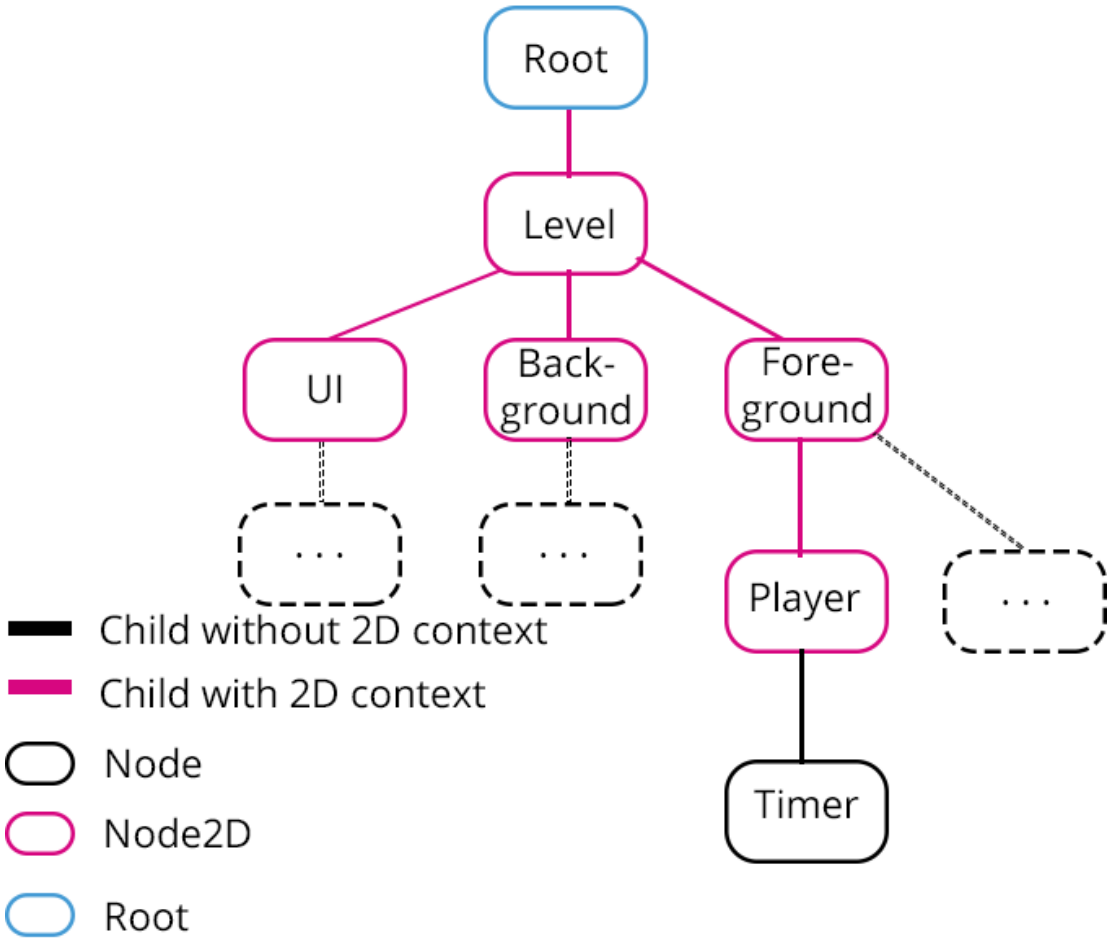

Figure E.1: Example of the tree structure of the game engine

Appendix F

Risk analysis

To assure high quality of the product we have thought of possible risks before we started developing and a way to minimize them. This way we had a clear focus on what parts of the product were essential and how much of a problem it would be if they couldn't be implemented as desired. We divided them between general risks and application specific risks.

F.1 General

F.1.1 Incorrect time estimation

It could be that we estimated the workload of some requirements wrongly which would result in delay in our planning. To prevent not having enough time in the end to finish a well-functioning product due to an incorrect estimation of the workload we will focus on the requirements with the most priority first.

F.1.2 Dysfunctional/unnecessary code

It could be that in the end some code of the product is not used or does not work as expected. To make sure we minimize unnecessary or dysfunctional code we will test all the code and document it well. This way we can check each other's contributions easily and strive to high quality code.

F.1.3 Low user engagement in the product

It would be regrettable if we in the end deliver a product which does not appeal to the intended users. To make sure users are engaged with the product we will test with the intended users to see if our design choices are appealing to them.

F.1.4 Changing requirements

If a fundamental requirement is changed later on during the design process this could result in a lot of extra work and time. By clearly working out the requirements with the client and ordering them via the MoSCoW principle we hope to minimize the risk of this extra work.

F.1.5 Lack of communication

Due to lack of communication we could be doing double work or adding unnecessary parts which are not desired by the client. To keep having a good overview of each other's work we are using Trello and Scrum. Besides we have an iteration to show the client each week to see if we are still on the same page.

F.2 Application specific

F.2.1 No pitch detection

High impact, low risk
Since we already got a reasonable pitch detection working in the first iteration we will definitely have one, since it is high priority we will focus on improving it a lot.

F.2.2 No audible feedback

High impact, low risk
Since we already got a form of audible feedback working in the first iteration we will definitely have this, since it is high priority we will focus on improving it a lot.

F.2.3 Audible feedback doesn't give expected experience

High impact, low risk
If the audible feedback does not sound like a natural result of the theme, it's entire purpose is lost. However, since we can test this quite easily ourselves, we should be able to get a proper result out of this.

F.2.4 No user tests with children possible

Medium impact, medium risk
If we can't conduct tests with children, it will be difficult to predict whether the user interface appeals to them, and whether the application works as intuitive as it is supposed to be. As a fallback, we would research how applications for children should be designed, so that user tests are less necessary. Next to that, the application should be built in a way which enables quick adaptations to the interface, so this could be improved later on if problems appear.

F.2.5 No logging functionality

Medium impact, low risk
Without the logging functionality, the application would still be the same for most of the users. Furthermore, since the desired logging functionality should be quite simple to implement, this will probably not be a problem. As a fallback, we will build the game in such a way that a logging functionality could easily be implemented afterwards.

F.2.6 Application doesn't work with schools' hardware

Medium impact, low risk
Because primary schools might not have the best audio devices, it could happen that our application does not work optimal in those environments. As a fallback, the application could be restricted to only being used in the SoundLAB Enschede, or we could advise schools to arrange audio devices that do work with the application.

Appendix G

API specification

Planet Opera back end

This is the API specification for the research back end of the 'Planet Opera' application.
Version: 1.0.0
BasePath:/api

Access

Methods

[Jump to Models]

Table of Contents

Run

`POST /sessions/{sessionId}/runs`

Add a run to a session (**addRun**)

Adds a run to the session with sessionId. If the session does not exist, it is created

Path parameters

sessionId (required)
Path Parameter —

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body Run (required)
Body Parameter — Run object that needs to be added to the session

Query parameters

label (optional)
Query Parameter — Label to add to the data

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
201
Run successfully created
400
Malformed object

GET /sessions/{sessionId}/runs

Get all runs from the session (**getRuns**)

Path parameters

sessionId (required)
Path Parameter —

Return type
Run

Example data
Content-Type: application/json


```
{
  "noteFrequencies" : [ {
    "timeMs" : 0,
    "frequency" : 6
  }, {
    "timeMs" : 0,
    "frequency" : 6
  } ],
  "userFrequencies" : [ {
    "timeMs" : 0,
    "frequency" : 6
  }, {
    "timeMs" : 0,
    "frequency" : 6
  } ],
  "sessionId" : "046b6c7f-0b8a-43b9-b35d-6489e6daee91",
  "runId" : "046b6c7f-0b8a-43b9-b35d-6489e6daee91",
  "obstacleFrequencies" : [ {
    "timeMs" : 0,
    "frequency" : 6
  }, {
    "timeMs" : 0,
    "frequency" : 6
  } ]
}
```


Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Ok Run
404
Session not found

Stats

GET /sessions/stats

Get usage stats from a specified date range (**getStats**)

Query parameters

startDate (optional)
Query Parameter — default: Current date - 7 days format: DD-MM-YYYY

endDate (optional)
Query Parameter — default: Current date + 1 day format: DD-MM-YYYY

Return type
array[Stats]

Example data
Content-Type: application/json

```
[ {
  "count" : 0,
  "groupBy" : "2000-01-23T04:56:07.000+00:00"
}, {
  "count" : 0,
  "groupBy" : "2000-01-23T04:56:07.000+00:00"
} ]
```

Responses
200
Ok

Models

[Jump to Methods]

Table of Contents

Run -

sessionId (optional)
UUID format: uuid

runId (optional)
UUID format: uuid

obstacleFrequencies (optional)
array[TimedFrequency]

noteFrequencies (optional)
array[TimedFrequency]

userFrequencies (optional)
array[TimedFrequency]

Stats -

groupBy (optional)
Date The date of the stat format: date-time

count (optional)
Integer Amount of sessions on that date

TimedFrequency -

timeMs (optional)
Integer

frequency (optional)
Integer