Uncharted Chants

Music Training Game for CI Users using Real Time Simulated Instruments

Master Thesis

Erik Frej Knudsen & Helmer Emo Nuijens

Aalborg University
Department of Architecture, Design and Media Technology

Copyright © Aalborg University 2022

Department of Architecture, Design and Media Technology
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Uncharted Chants - Music Training Game for CI Users using Real Time Simulated Instruments

Theme:
Cochlear Implants, Music Training, Physical Modelling

Project Period:
Spring Semester 2022

Project Group:
-

Participant(s):
Erik Frej Knudsen
Helmer Emo Nuijens

Supervisor(s):
Stefania Serafin

Copies: 1

Page Numbers: 84

Date of Completion:
May 25, 2022

Abstract:

This thesis presents the design and development process of *Uncharted Chants*, a game that is focused on training music perception for cochlear implant (CI) users. The main focus of this project is to make music training engaging and fun by utilizing game-based learning principles. Our approach is to use real-time implemented physical models to synthesize the sounds with the purpose of emulating real instruments. An evaluation is presented, in the form of performance, usability, and user experience tests, for both normal-hearing listeners and CI users. Results show that the implemented game concept can have positive effects on the level of engagement toward music training exercises. Next to this, the models for synthesizing sounds are perceived as realistic and allow for small timbre differences to be perceived based on different interactions.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

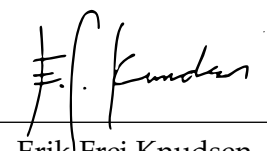
Contents

Preface

This thesis concludes our master's degree in Sound and Music Computing at Aalborg Unversity in Copenhagen. This project started as an internship at Oticon Medical and this thesis is a continuation of that.

We want to express our sincere gratitude to Stefania Serafin and Marianna Vatti for supervising us throughout the project. We also like to thank Silvin Willemsen for helping specifically with the implementation of physical models, as well as everyone participating in our study.

Aalborg University, May 25, 2022



Erik Frej Knudsen
<eknuds20@student.aau.dk>

Helmer Emo Nuijens
<hnuije20@student.aau.dk>

Chapter 1

Introduction

Music plays an important role in our lives and contains cognitive, emotional, and social functions. Music is a source of entertainment and pleasure [50]. It can trigger certain memories, experiences, moods, and emotions [29, 50]. Music is omnipresent, as it plays significant roles in various social, cultural, and spiritual events [50]. Additionally, music can play a substantial role in people's well-being and social lives [34].

Music is a complex form of sound that consists of elements such as rhythm, pitch, timbre, and harmony [36]. These elements requires the perception of layers with varying temporal representation, frequency distribution, and harmonic content [36]. As cochlear implant (CI) systems lack the precision needed to correctly perceive these, music perception of CI users is generally poor [28]. Consequently, CI users commonly describe music listening as unsatisfying [33, 40], making them possibly miss out on the previously mentioned benefits that music can bring.

Still there has been emerging evidence that music perception training may aid CI users as a means to fine-tune their auditory system and could potentially increase music listening enjoyment [18]. Despite this, music training tools are not commonly offered in typical clinical practices [19] nor are they available outside selective research protocols [14]. This is why Oticon Medical [1], a company specialized in hearing solutions such as CIs and bone conducting hearing, proposed us to develop a prototype of an innovative music training application for CI users. One thing they observed was that the training tools that do exist can often be tedious or dull and CI users often fail to complete them. This is why the main challenge of our project was to make music training fun and engaging using digital-based learning.

Digital-based learning (DBL) is a way to facilitate learning using games [12]. Games that are used to promote knowledge acquisition, have gotten increasingly more attention from researchers mainly because of their potential in terms of en-

[1] https://www.oticonmedical.com/

gaging the user in educational practices [12]. Although many attributes contribute to user engagement, engagement can be seen as a form of attention, intrinsic interest, motivation, and curiosity [45]. Because of their potential in increasing engagement, DBL can lead to more effective and enjoyable learning experiences [12, 13]. Games with applied DBL are often referred to as serious games. Serious games integrate a relationship between two main objectives: the game experience, where the goal is to have fun, and a pedagogical component, with an educational objective [8].

1.1 Music-Based Serious Games

To familiarize the concept of serious games, we will examine a selection of related work on music-based serious games.

Rhythm Workers

Rhythm Workers is a game developed to re-train rhythm for people that have disrupted rhythm perception due to neurological or neurodevelopment disorders (such as Parkinson's disease) [4]. Retraining rhythm may help in regaining the lost cognitive functions that are associated with rhythm perception, such as poor performance in language, memory, and attention. For this, construction was implemented as the main game mechanism to train rhythm by both perceptual and motor tasks. In this game, different levels of a building are constructed through listening, of which the aesthetic quality of the building reflects the player's performance. Apart from the positive effect on rhythm skills found for some participants, results show high motivation of the participants when playing the game as well.

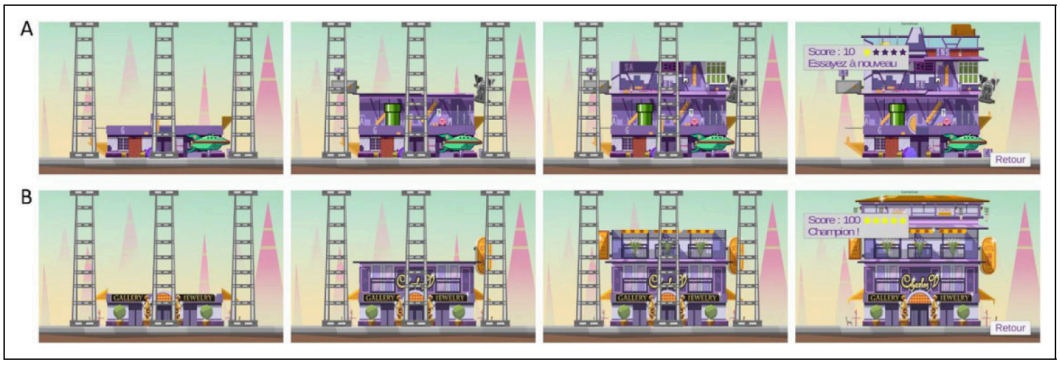


Figure 1.1: One of the tasks of Rhythm Workers where the quality of each level of the constructed building reflects the player's performance, extracted from [4].

iClef

iClef is a phone application that is developed to teach people to correctly identify notes with different clefs [3]. The interface, depicted in Figure 1.2, consists of a clef, the notes that need to be identified, and a piano keyboard. With increasing speed, the player has to identify the note by pressing one of the keys on the keyboard. The developers integrated multiple elements to promote engagement, such as interactivity and competition. iClef has been further developed and is available on the App Store [2].

Figure 1.2: The interface of iClef, extracted from [3].

Speech Perception Training

Another serious game, designed by Larrea-Mancrea et al., focused on understanding speech in the presence of acoustic competition for people with hearing impairments [32]. The game was designed to train several auditory processing skills such as spectral-temporal processing, sound localization, and auditory memory. The main goal of the game is to avoid obstacles by correctly reacting to stimuli. The interface, depicted in Figure 1.3, shows different screenshots for the tasks that the player has to complete, including pitch-discrimination, specialization, and memory tasks. Although no significant improvements are reported, possibly due to the absence of participants with hearing impairments, the study demonstrated the feasibility of this entertaining game to be used in participants' homes and on uncalibrated devices.

[2] https://apps.apple.com/us/app/iclef/id1520882406

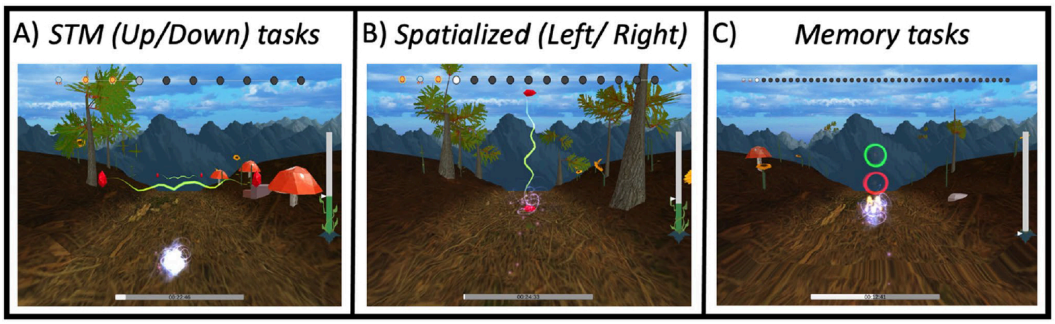


Figure 1.3: The interfaces for the three different tasks, extracted from [32].

1.2 Project Goal

As serious games are only recently starting to be applied to music training for CI users, they are not readily available. This makes it a compelling area for further research and development. In this thesis we do so by designing and developing a music training game for CI users. Our key focus points for this project are engagement and implementing real instrument sounds, both of which are currently explored by Oticon Medical for novel music training games. During our internship project we focused on exploring various game concepts and selecting one to further develop. This concept, titled *Uncharted Chants*, formed the foundation of our thesis.

We formulated the following goal of our thesis:
To design and develop an engaging game experience for training music perception for CI users using real instrument sounds.

To reach this, we first analyze the functionality and perception of CIs as well as investigate potentials for music training in chapter 2. Then in chapter 3, we examine current synthesis techniques that are used to synthesize real instrument sounds. In chapter 4 an overview of the design process that lead to the game concept Uncharted Chants is described. Then in chapter 5, a technical description is given of the game's implementation. Finally, we present the results of both a technical, usability, and user experience evaluation with normal hearing listeners and CI users, in chapter 6, which is further discussed in chapter 7.

Chapter 2

Cochlear Implants

Cochlear Implants, or CIs for short, are devices that utilize electrical stimulation to provide or restore functional hearing [61]. This technology holds the position of being the most successfully neural prosthesis with currently around 736,900 recipients around the world [1]. However widespread it is now, commercialization only began relatively recently. The first phenomenon of auditory sensation by electric stimulation was encountered in the 19^{th} century, almost two centuries ago. This prompted more interest and research into the subject with physicians trying to translate early findings like this into clinical practices. This effort resulted in the first report on successful hearing using electric stimulation in 1957 [61]. After the further development and approval by the U.S. Food and Drug Administration (FDA), CIs became available to consumers [61]. In this chapter, we will briefly go over the CIs' general functionality and design, speech and music perception, and finally music perception training and its relevance to our project.

2.1 Cochlea

Normally, sound is perceived by a collection of mechanisms that together convert physical vibration into an encoded nervous impulse [1]. This conversion begins in the outer ear, where acoustic sound propagates through the auditory canal towards the tympanic membrane (eardrum). The movement of the tympanic membrane then sets a series of small coupled bones into motion, located in the middle ear. These bones, the ossicles (malleus, incus, and stapes respectively), connect the tympanic membrane to the oval window. To do so they serve as an impedance matching system as the medium changes from low-impedance into a high-impedance fluid that is located inside the cochlea [22].

[1] Retrieved from https://www.frontiersin.org/articles/10.3389/fnhum.2021.757254/full, Accessed on: 25/01/22

The cochlea is a snail-shell-like part of the inner ear (hence the name) that houses the organ of hearing. The movement of the liquid inside the chambers of the cochlea creates standing waves in the basilar membrane of which the vibration then in turn is picked up by the hair cells on the membrane. The difference in stiffness of the membrane establishes a tonotopic organization, where hair cells located closest to the oval window are resonating with high frequencies and hair cells located at the apical end of the membrane are resonating with low frequencies. Finally, the movement of the hair cells sends out nerve impulses through the nerve fibers to the brain, where the signals are processed. [1].

2.2 Design and Functionality

CIs are provided for people that have severe sensorineural hearing loss, in which the normal auditory mechanism, discussed above, is bypassed [35]. Instead, the sound is converted into electrical signals and directly stimulates the auditory nerve. Almost all CIs are using the same design, which can be separated into the following components [35, 61]:

1. An external unit, or speech processor, picks up the sound, processes it using a digital sound processor (DSP) unit, and transmits the data using radio frequency (RF) data. This unit is positioned above the ear and held in place by a magnet attracted to the internal unit.

2. An internal unit placed below the skin behind the ear. This system receives and decodes the incoming RF data and converts it into electrical currents that are transmitted to the electrodes.

3. The electrode array that is located inside the cochlea, which stimulates the auditory nerve.

There are multiple strategies when it comes to converting the acoustic audio into electrical pulses, all with different spectral (relating fundamental frequency, harmonics, etc.) and temporal resolution (changes of amplitude and frequency over time) [59]. Commonly used is the CIS strategy [61], of which the block diagram can be seen in Figure 2.1. The acoustic signal is first bandpass-filtered into a series of channels to extract the envelopes of the different frequency ranges. The envelopes are then compressed and used to amplitude modulate a fixed-rate pulse carrier [61]. In an attempt to distribute the frequency channels according to the tonotopic mapping of the cochlea, the frequency channels which represent higher frequencies are sent to basilar electrodes and channels representing lower frequencies are sent to apical electrodes.

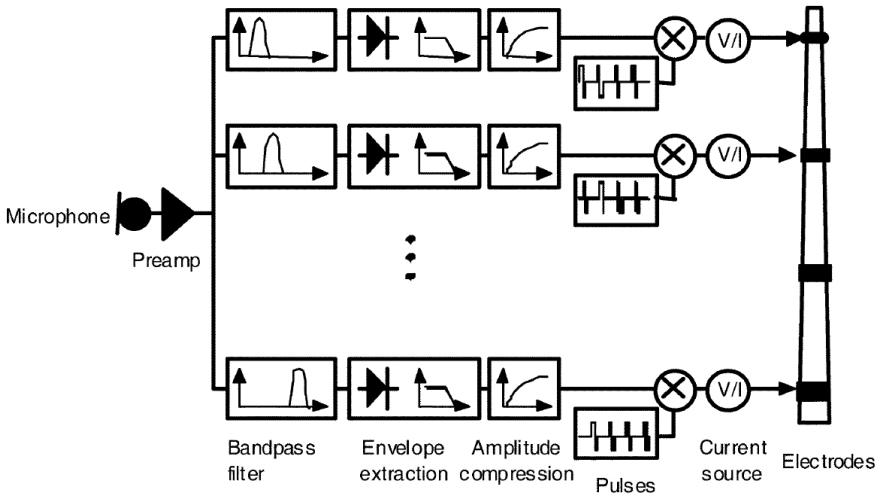


Figure 2.1: Block diagram of audio conversion to electrical pulses using the CIS strategy, extracted from [61].

2.3 Perception

The CI's capability to restore speech perception for deaf people has been very successful [9]. CIs are designed with the focus of enhancing the user's ability to perceive speech and engage in conversation, and they do so very effectively. Present-day cochlear implants convey speech signals in such a way that 80% of the sentence is recognized in quiet environments [28]. However, our environment constantly exposes us to various other sounds that differ significantly from speech signals. Sounds such as birds chirping in a forest or music in a cafe are perceived poorly by CI users. Music perception is usually challenging for CI users due to the complexity of musical signals, often containing harmonies, multiple instruments, overlapping spectral content, etc. Some of the most prominent constraints of CIs for music perception are presented in this section. These constraints, as described by Limb and Roy [37], can be categorized into technological, biological, and acoustic types, that each may limit the perception of a CI user.

Technical Constraints

The technical limitations of CIs become evident when looking at the process of converting an acoustic sound into sequences of electrical impulses that should stimulate the auditory nerve. This works efficiently for speech signals, but difficulties arise when conveying music, because of the spectral, temporal, and timbral complexities and varying dynamic range that music often contains [37].

Usually, the electrode array contains at most 22 electrodes and they only cover

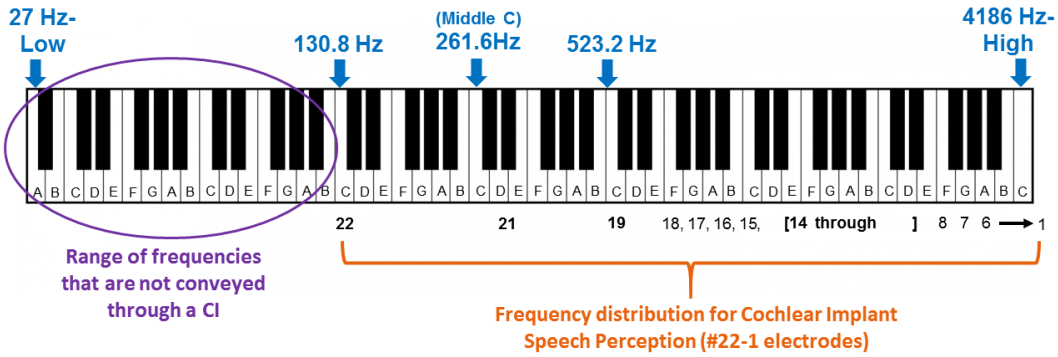


Figure 2.2: An example of a mapping of electrodes to musical notes, extracted from [41]

frequencies from around 200 Hz up to 8500 Hz (strongly varies between individuals) [37]. This means that at most an acoustic signal would be split into 22 frequency channels and that the low and high frequencies of music are not conveyed at all (see example in Figure 2.2). Furthermore, often fine pitch and harmony content would fall under the same frequency channels when bandpass-filtered and encoded to the electrode array. This usually results in poor music perception as the finely graded and frequency-specific information is lost in the conversion [37].

When the electrodes stimulate the auditory nerve at different positions the electric stimulation is imprecise and excites a multitude of nerve fibers when replicating a single frequency. Then there might be surgical and anatomical limitations, such as the placement of the electrode array in relation to nerve fibers, the depth at which the array can be inserted into the cochlea for maximum frequency resolution, or individual cochlear anatomy which are all factors that might decrease the perception-accuracy [37].

Biological Constraints

As a consequence of hearing loss, the auditory system often suffers deficits that affect the general perception of sound for CI users. These deficits may severely impact the ability to ultimately perceive music. When hearing loss is experienced, functional hair cells die, which results in the degeneration and death of nerve fibers. According to Hardie and Shepherd, damage to the nerve fiber network can lead to prolonged latencies, reduced temporal resolution, diminished neuronal efficiency and the abolishment of neuron-to-neuron signal transmission [24]. This means that the CI will have to compensate for these limitations by increasing current densities on electrode surfaces [52].

Several studies have been conducted using positron emission tomography (PET) scanning on both CI users and normal-hearing (NH) people. Here, PET was used to detect which parts of the brain were activated when a person listens to auditory

stimuli [38, 44, 60]. These studies show two tendencies. First, CI users seem to have a greater intensity of activation in the parts of the brain that are traditionally used for auditory processing. Second, due to brain plasticity, CI users can utilize other parts of the brain that are not traditionally used for auditory processing [37]. *Plasticity*, in cognitive neuroscience, is a term used to describe changes in structure and function of the brain that can change behavior and are related to the amount of exposure to stimuli [25]. This plasticity of the brain shows a positive impact on the perception of CI users, because of the found correlation between the intensity of activation in auditory cortices and performance in listening tasks [15, 26].

The potential for brain plasticity in CI users is confirmed by several studies to be age-dependent. The studies show that, in clinical tests, children implanted at a young age, are performing better in speech and music perception than children who obtained their CI later in life [54, 56, 43, 48]. Also, pre-lingually deafened CI users are found to enjoy and interact more with music than post-lingually deafened users. This is partly because pre-lingually deafened CI users have no reference to how music sounds with normal hearing and therefore they rely solely on the capability of the CI to portray how acoustic music sounds. Post-lingually deafened, on the other hand, have a better reference of how music should sound with normal hearing, and therefore they might find the capability of the CI to portray this unsatisfactory and therefore avoid engagement with music [37].

Acoustical Constraints

The perception of musical properties like rhythm, pitch, harmony, and timbre are generally poorly encoded by the CI, and often the conversion of the acoustic signal to electric pulses can drastically alter how the signal is perceived. The perception of rhythm is well preserved for CI users as the extracted envelopes are sufficient for conveying the rhythmic elements of the music [37]. As long as the music is based mainly on percussive rhythmic sounds, the CI users are capable of perceiving them almost as clearly as NH listeners. This was found in a study where CI users were performing a rhythm-listening task at 78% accuracy compared to normal hearing with an 84% accuracy [7].

On the other hand, the perception of pitch, melody, and harmony is severely altered. This is due to how the implant processes sound, the electrode design, as well as previously mentioned reasons. Most NH listeners can detect a pitch direction change of one semitone, however, the threshold of CI users for detecting a pitch direction change may vary from one to eight semitones or even more [30]. Since the perception of relative pitch is crucial to be able to perceive music, this proves to be a highly problematic limitation for CI users to perceive music.

The psycho-acoustic property called timbre, also known as tone color, covers the properties of a sound that makes it possible to distinguish sounds even though they have the same pitch, loudness, and duration [37]. As different instruments

have different timbres, NH listeners use this quality to differentiate them. Various studies have found that the perception of timbre in CI users varies drastically, but generally, CI users experience inferior timbre recognition than NH listeners [37].

Normally, the auditory system decomposes complex signals such as music by applying auditory stream segregation [37]. This is an ability to distinguish the different parts of a musical signal and separate them to be able to comprehend the music as a whole. However, CI users have more trouble analyzing and constructing different streams as the attributes used in this process, like pitch, dynamic range, and timbre, are poorly perceived. This ultimately limits their ability to listen to and enjoy music [37].

2.4 Music Training

Music training has been extensively studied for its relevance to training-based neural plasticity. Music activates multiple bilateral brain regions and involves ongoing encoding and processing of different musical patterns, like pitch, timbre, rhythm, and loudness, and how they correspond to each other (e.g. melodies and harmonies) [20, 25]. Although there seems to be some effect on neural response when being passively exposed to music (seen in musically untrained people) [25], music training does have a larger impact on the perceptual accuracy of speech and music stimuli. A study by Brown et al., where a group of musically trained and untrained people was compared, reported that musically trained people had superior perceptual accuracy when it comes to, among others, pitch discrimination and timbre recognition [20].

Playing music, as opposed to music training involving only listening, can lead to even more extensive neural changes. As playing music involves both the sensory-motor system in combination with the auditory system it demands higher-order processes [25]. The brain regions activated when engaged with music are associated with emotional response, arousal and attention, semantic and syntactic processes and motor functions [18]. Playing music often includes positive emotions and rewards (when playing with sufficient accuracy), and requires repetition and focused attention, which is required for experience-based plasticity. Because of this, playing music can increase experience-based plasticity even more [47]. This can also be seen in studies that have been investigating differences between musicians and non-musicians, where musicians show increased thickness, volume, and concentration of auditory cortices, and increased binding of auditory features like pitch and timbre [25]. Also, the interconnection between the auditory and motor areas likely contributes to this enhanced plasticity. In a study by Herholz et al., was suggested that only attentive listening that involves a task was not sufficient for measurable plasticity [25].

These insights have also sparked interest in using music as a training tool for

the rehabilitation of CI users. It is hard to generalize previous findings for CI users, as their perception differs drastically from NH listeners whom the studies were targeted on. However, preliminary research shows that CI users can benefit from auditory training. From the review article of Gfeller et al. [20] came forward that music training on adult CI users shows increased pitch discrimination, timbre recognition, melodic contour recognition, and complex melody recognition. Additionally, music training could improve sound quality ratings (timbre appraisal), general music enjoyment, and participation [39, 21, 20]. This suggests that music training has the potential of improving overall music enjoyment or even the quality of life of CI users [20].

Apart from benefits on music perception and enjoyment, music training can also improve speech perception [31, 20]. This was found to be the case for NH listeners, but also for CI users, where CI users who have more accurate pitch and timbre perception are also more likely to correctly perceive complex elements of speech (e.g. speech in noise and talker identification) [20]. Patel et al. explained this phenomenon using the so-called OPERA hypothesis, stating that there is an overlap between speech and music-processing brain regions and that music listening requires a higher precision when it comes to feature extraction. This could explain (among emotions, repetition, and attention associated with music) how music training could improve speech perception as well [47].

Training Protocol

There are two main approaches when it comes to music training: analytic and synthetic [20]. Analytic music training employs bottom-up processes, where often acoustic features, such as pitch and timbre, are isolated and exposed to the listener with increased difficulty. The aim here is to train the auditory system for increased perceptual precision and efficiency. With synthetic music training, the listener is exposed to more complex, often natural sounds, like real instruments and speech. Here the listener has to extract features themselves and is aimed to train the efficiency of cognitive processing, like increased attention.

Although there is a significant correlation between time listening to music and music listening enjoyment [40], it is hard to determine how much exposure or what duration of music training is adequate. As sufficient training depends on many variables, such as the stimuli used and differences between individuals, it is not yet clear how much training is adequate [20]. However, as experience-based plasticity requires enough exposure to the stimuli, a training program should have sufficient repetition and duration [20]. To do so, the participant has to be persistent as therapy works best when the participant is highly engaged, motivated, and focused [20, 39]. To achieve this, training programs should consider the great differences in perception between individuals, described previously. Therefore, music training should be individually focused in terms of difficulty.

Chapter 3

Sound Synthesis

Digital sound synthesis are numerical algorithms intended to create musically interesting sounds or simulate realistic sounds [55]. Digital sound synthesis began in 1957 by researchers at Bell Telephone Laboratories, where they succeeded in producing a time-varying sound with a certain envelope and waveform [49]. These first examples of synthesized sounds could not be computed in real-time as it was too computationally heavy to calculate these sounds with the former computing power. Sound synthesis has come a long way since this as new techniques started to develop. Of all conventional techniques, also known as abstract methods [5, 55], wavetable synthesis, additive synthesis, FM-synthesis, and granular synthesis are the most well-known [6]. What these early types of sound synthesis techniques have in common is their simplicity and are often extremely computationally efficient [5]. However, as computational power increased over the years, other techniques that focused on simulating natural sounds and instruments, called physical modeling started to emerge.

This chapter will start with an overview of popular abstract methods. Thereafter, various physical modeling techniques will be presented, focusing on Finite-Difference Schemes (FDSs), in more detail as they are taking a significant portion of the project.

3.1 Abstract methods

As abstract methods have been around for some time they have reached a certain level of refinement [5] and a lot of techniques have emerged. The most well-known synthesis techniques are briefly explained next.

- Wavetable Synthesis. A technique that stores one period for one or more waveforms in a table instead of directly calculating values through sine or cosine functions [5].

- Additive Synthesis. A spectral method that adds single sinusoidal components together to create complex tones [55].
- Subtractive Synthesis. Also known as source-filtering, a technique that removes frequencies from the spectrum using filters often applied to a complex tone [55].
- Amplitude Modulation (AM) Synthesis. A synthesis technique where a time-variant modulator is applied to the amplitude of the carrier signal. The resulting spectrum consists of both carrier frequency and two added frequencies (carrier frequency $\pm$ frequency of the modulator) [46].
- Frequency Modulation (FM) Synthesis. Similar to AM, but applying the modulator to the frequency of the carrier. This technique allows the synthesis of more complex spectra, with the use of just two sinusoids [55].

Of these methods, additive synthesis will be described more in detail as it is used later in the implementation.

3.1.1 Additive Synthesis

A popular synthesis technique that has been used since at least the 1960s [5] is called additive synthesis. The underlying concept of additive synthesis, coming from the Fourier theory, is that almost any sound can be constructed using a collection of sinusoids [55]. A pitched sound can be created by selecting a frequency close to that of a multiple of the fundamental frequency, and contrarily, non-pitched sounds, like bells, can be synthesized by avoiding this harmonic coherence [5]. A single sinusoidal signal can be mathematically expressed in continuous time as

$$x(t) = A\cos\left(2\pi f t + \phi\right), \tag{3.1}$$

with A, f, and ϕ the amplitude, frequency and phase of the sinusoid, respectively. These parameters are also called the control functions, although the phase is often left out as for simplicity. After sampling, the same expression can be written down in discrete time as

$$x[n] = A\cos\left(2\pi f n/f_s + \phi\right), \tag{3.2}$$

with sampling frequency f_s, and time step n. Then the output signal $y[n]$ can be constructed by adding the individual sine waves:

$$y[n] = \sum_{l=0}^{M} A_l \cos\left(2\pi f_l n/f_s + \phi_l\right), \tag{3.3}$$

with index l and number of sinusoids M. Even though this method is effective and extremely simple to implement, there are also downsides to using it. Mainly the amount of data stored for tracking each sinusoid can be problematic for the high amount of sinusoids, for example when synthesizing noisy signals. [55].

3.2 Physical modeling

Of the above methods, although they are generally not computationally expensive, it can be difficult to predict the behavior of sound as they include certain parameters that do not possess any association with a physical representation [5]. As computing power increased over the years, physical modeling emerged. with physical modeling sound synthesis, the parameters controlling the sound are linked to physical rather than perceptual attributes. This, in contrast with spectral modeling, is a bottom-up approach, where a target system, often a real instrument, is described using a set of equations [6]. These equations, often a set of partial differential equations (PDEs), can describe the resonator (violin, guitar, trumpet, etc.), but can also model the excitation input (e.g. bowing the violin, plucking the guitar, or the lip pressure when playing the trumpet). The main advantage of this over recording real instrument sounds is flexibility. In this way different interactions can be modeled in real-time, without the need to prerecord all the different possible interactions.

There exists different techniques when it comes to physically modeling a musical instrument, all with different complexities. The most popular ones are briefly described below.

- Digital Waveguide Modeling, an efficient technique that solves the 1D wave equation using delay lines [53].
- Modal Synthesis is a technique that possesses a direct link with additive synthesis where a target system is analyzed in terms of the modes it produces by a particular excitation [55].
- Mass-Spring Networks, a technique in which the system, such as a string or a plate, is described using spring-connected masses [5].
- Finite Difference Time Domain methods (FDTD), a numerical method where the continuous system is approximated by difference operations and represented over a grid in discrete time and space [5].

Even though FDTD require a great amount of computing power to simulate, their advantages, such as their generality, simplicity, and flexibility when it comes to the vast amount of instruments and interactions it can describe [5], make it a very

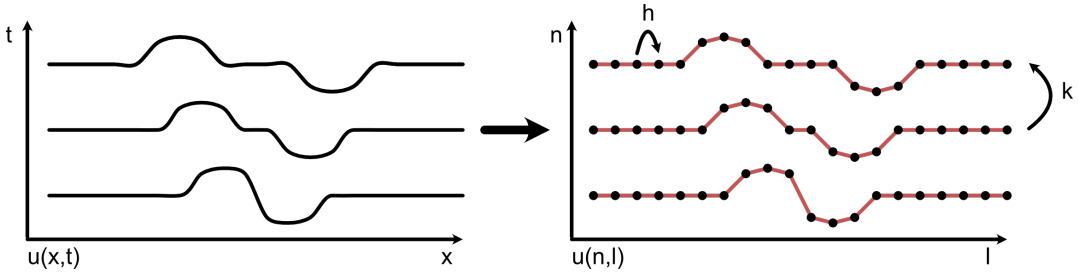


Figure 3.1: To the left the system in continuous time and to the right the system in discrete time, where the system is subdivided in both space and time.

compelling synthesis technique. Because this technique forms the main synthesis technique used in this project, it will be explained more in-depth below. The theory described originates mostly from the book 'Numerical Sound Synthesis' by Stefan Bilbao [5], which can be consulted for further clarifications.

3.2.1 Finite-Difference-Time-Domain Methods

With FDTD methods the state of a continuous system, described by a PDE, is approximated over discrete space and updated over discrete time. This is depicted in Figure 3.1, where the state of a continuous system is discretized and represented by a set of limited points and calculated for each time step. The state of a system, often described by $u = u(x,t)$ can be discretized to a grid function u_l^n, where n and l originate from discretizing the spatial variable $x = lh$ and time variable $t = nk$ respectively. Here h stands for the grid spacing, or distance between two grid points, and k for the time step and is dependent on the sampling frequency f_s ($k = 1/f_s$). The methods of approximating a PDE is also known as Finite-Difference Schemes (FDSs) and is done by using difference operators that can be applied to a grid function [5]. For example, one can approximate the first-order derivative with respect to time (∂_t) applied to u_l^n using the forward, backward, and centered difference operators:

$$\partial_t u \cong \begin{cases} \delta_{t+} u_l^n = \frac{1}{k}(u_l^{n+1} - u_l^n) \\ \delta_{t-} u_l^n = \frac{1}{k}(u_l^n - u_l^{n-1}) \\ \delta_{t\cdot} u_l^n = \frac{1}{2k}(u_l^{n+1} - u_l^{n-1}) \,, \end{cases} \tag{3.4}$$

with δ indicating the difference operator. Similar operations can be used for first order spatial derivatives:

$$\partial_x u \cong \begin{cases} \delta_{x+} u_l^n = \frac{1}{h}(u_{l+1}^n - u_l^n) \\ \delta_{x-} u_l^n = \frac{1}{h}(u_l^n - u_{l-1}^n) \\ \delta_{x\cdot} u_l^n = \frac{1}{2h}(u_{l+1}^n - u_{l-1}^n) \,. \end{cases} \tag{3.5}$$

For second-order derivatives, a combination of the backward and forwards difference operators can be used ($\partial_t^2 \approx \delta_{tt} u_l^n = \delta_{t+t-} u_l^n$, and $\partial_x^2 \approx \delta_{xx} u_l^n = \delta_{x+x-} u_l^n$).

After the PDE has been discretized, the next state of the system can be found by solving the system for the next time step (u_l^{n+1}). A step-by-step example will be given in the following section for discretizing the damped stiff string.

3.2.2 Towards the Damped Stiff String

The 1D Wave Equation

A prominent PDE in the field of acoustics and physics is the 1D wave equation, which describes the movement of a one-dimensional system over time (e.g. a string or air in a tube). This second-order PDE, dependent on space x, and time t, is defined as

$$\partial_t^2 u = c^2 \partial_x^2 u \,, \tag{3.6}$$

where c is the wave speed of the system. This can then be discretized to yield the following FDS:

$$\delta_{tt} u_l^n = c^2 \delta_{xx} u_l^n \,. \tag{3.7}$$

This FDS can then be expanded using the difference operators for space 3.5 and time 3.4:

$$\frac{1}{k^2}(u_l^{n+1} - 2u_l^n + u_l^{n-1}) = \frac{c^2}{h^2}(u_{l+1}^n - 2u_l^n + u_{l-1}^n) \,. \tag{3.8}$$

Finally, by solving for u_l^{n+1} yields following update equation:

$$u_l^{n+1} = 2u_l^n - u_l^{n-1} + \lambda^2(u_{l+1}^n - 2u_l^n + u_{l-1}^n) \,, \tag{3.9}$$

with Courant number $\lambda = ck/h$. It is good to mention that when discretizing, one should pay close attention to the stability of the system [5]. In the case of the 1D wave equation the stability is solely dependent on λ, and when chosen incorrectly an unstable system could 'blow up'. The stability condition for the 1D wave equation is defined as

$$\lambda \leq 1 \implies h \geq ck \,. \tag{3.10}$$

The Damped Stiff String

The 1D wave equation on itself sounds rather synthetic as it is lacking natural elements like energy losses and inharmonicities. In order to model a realistic string sound, one would need to add damping and stiffness to the system. Equation 3.7 can be expanded to yield the damped string equation

$$\partial_t^2 u = c^2 \partial_x^2 u - 2\sigma_0 \partial_t u + 2\sigma_1 \partial_t \partial_x^2 u \,, \tag{3.11}$$

with frequency-independent damping σ_0 and frequency-dependent damping σ_1. This can be further expanded by adding stiffness, yielding

$$\partial_t^2 u = c^2 \partial_x^2 u - 2\sigma_0 \partial_t u + 2\sigma_1 \partial_t \partial_x^2 u - \kappa^2 \partial_x^4 u \,, \tag{3.12}$$

where stiffness coefficient $\kappa = \sqrt{EI/\rho A}$, with Young's Modulus E, moment of inertia $I = \pi^4/4$, material density ρ, and cross-sectional area A. This added coefficient introduces inharmonicities to the system due to a phenomena called dispersion, where higher frequencies travel faster than lower frequencies. Discretizing and expanding Equation 3.12 will yield the following FDS:

$$\begin{aligned}\frac{1}{k^2}(u_l^{n+1} - 2u_l^n + u_l^{n-1}) =& \frac{c^2}{h^2}(u_{l+1}^n - 2u_l^n + u_{l-1}^n) - \frac{2\sigma_0}{2k}(u_l^{n+1} - u_l^{n-1}) \\ &+ \frac{2\sigma_1}{kh^2}(u_{l+1}^n - 2u_l^n + u_{l-1}^n - u_{l+1}^{n-1} + 2u_l^{n-1} - u_{l-1}^{n-1}) \\ &- \frac{\kappa^2}{h^4}(u_{l+2}^n - 4u_{l+1}^n + 6u_l^n - 4u_{l-1}^n + u_{l-2}^n)\,. \end{aligned} \tag{3.13}$$

Again in order to calculate the next states, Equation 3.13 is solved for u_l^{n+1} yielding the following update equation:

$$\begin{aligned}u_l^{n+1} =& \frac{1}{1+\sigma_0 k}\Big[2u_l^n - u_l^{n-1}(1-\sigma_0 k) + \lambda^2(u_{l+1}^n - 2u_l^n + u_{l-1}^n) \\ &+ \frac{2k\sigma_1}{h^2}(u_{l+1}^n - 2u_l^n + u_{l-1}^n - u_{l+1}^{n-1} + 2u_l^{n-1} - u_{l-1}^{n-1}) \\ &- \frac{k^2\kappa^2}{h^4}(u_{l+2}^n - 4u_{l+1}^n + 6u_l^n - 4u_{l-1}^n + u_{l-2}^n)\Big]\,, \end{aligned} \tag{3.14}$$

with stability condition

$$h \geq \sqrt{\frac{c^2k^2 + 4\sigma_1 k + \sqrt{(c^2k^2 + 4\sigma_1 k)^2 + 16\kappa^2 k^2}}{2}}\,. \tag{3.15}$$

Boundary Conditions

To discretize a system in space, boundary conditions, or endpoints must be established. As can be seen from the stencil of the stiff string (Figure 3.2), calculating a new grid point always requires two neighboring grid points due to the fourth-order spatial derivative in Equation 3.12. The most straightforward way to solve this is by fixating the boundary grid points at either side to be zero (clamped). One disadvantage of using this, however, is that the system can quickly become out of tune with higher fundamental frequencies as there are fewer calculated grid points.

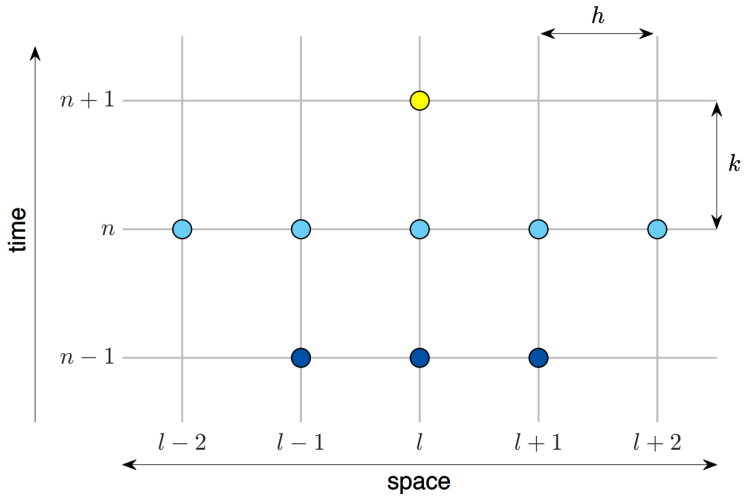


Figure 3.2: The stencil of the damped stiff string, retrieved from [58]. In order to calculate the future state of a grid point, the state of five current grid points should be determined (u_l^n, $u_{l\pm1}^n$, and $u_{l\pm2}^n$) and three points of the previous state (u_l^{n-1} and $u_{l\pm1}^{n-1}$).

A simply supported boundary condition can be used to circumvent this, in which only one endpoint is fixated and the other one is calculated using the second-order spatial derivative. Both conditions are defined below for discrete-time

$$u_l^n = \delta_{x\pm} u_l^n = 0 \qquad \text{(Clamped)}\,, \tag{3.16}$$

$$u_l^n = \delta_{xx} u_l^n = 0 \qquad \text{(Simply Supported)}\,. \tag{3.17}$$

3.2.3 Exciters

The excitation on the resonator determines which modes of vibration (the natural harmonics) will be present [11]. There are two main excitation methods: exciting using initial conditions and time-varying excitations. With initial conditions, one or more grid points are set to non-zero values, and with time-varying excitations, the excitation is also based on a temporal variable. For each, one example will be given that is used in the implementation later in this thesis.

Pluck

A simple way to create a smooth excitation is by using a raised cosine, also known as a Hanning window[1]. Consider a raised cosine excitation with width (e_w), amplitude e_a, and the center location (l_c). Then, the start location of the excitation is given by $l_s = l_c - e_w/2$, and the end location by $l_e = l_c + e_w/2$. Then the excitation

[1] https://ccrma.stanford.edu/ jos/sasp/Hann_Hanning_Raised_Cosine.html

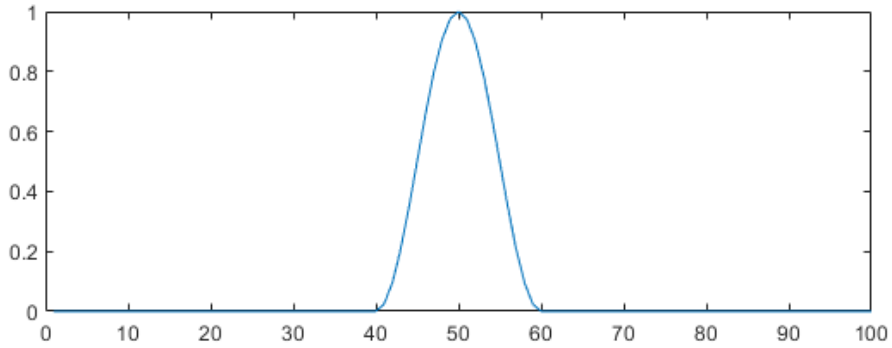


Figure 3.3: A raised cosine with $e_a = 1$, $l_c = 50$, and $e_w = 20$. The vertical axis denotes the amplitude (in m), and the horizontal axis the grid points in space.

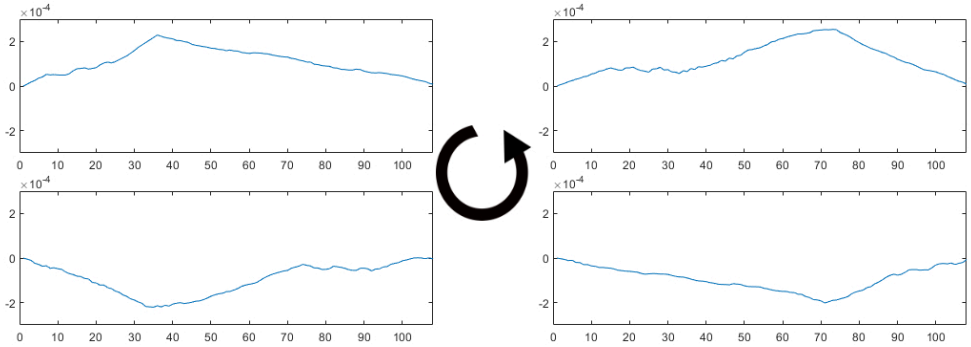


Figure 3.4: The Helmholtz motion visualized. The vertical axis denotes the amplitude (in m), and the horizontal axis the grid points in space.

to the grid points between these points is determined by the raised cosine function in discrete-time given below.

$$e_l = \begin{cases} \frac{e_a}{2}\left(1 - cos\left(\frac{2\pi(l-l_s)}{x_w}\right)\right), & \text{if } l_s \leq l \leq l_e \\ 0, & \text{otherwise}. \end{cases} \tag{3.18}$$

Bow

The timbre of a bowed string originates from the non-linear friction between the bow and the string [6]. When the bow velocity, force, and position are chosen well the recognizable Helmholtz-motion is present. This phenomena, depicted in Figure 3.4, is where a triangular wave is moving around the string. To model a bow interaction for FDSs a friction model is used, which scales how much the bow force affects the resonator. A simple one is the static friction model, where the force is dependent on the relative velocity only. The one used in this project is defined

as

$$\Phi(v_{rel}) = \sqrt{2a} v_{rel} e^{-a v_{rel}^2 + 1/2}, \tag{3.19}$$

with relative velocity between the bow and string $v_{rel} = \partial_t u(x_B, t) - v_B(t)$, and scalar a. A bow force f_b, that is scaled by this friction model, can then be applied to the PDE of the stiff string (Equation 3.12), yielding the following PDE:

$$\partial_t^2 u = c^2 \partial_x^2 u - 2\sigma_0 \partial_t u + 2\sigma_1 \partial_t \partial_x^2 u - \kappa^2 \partial_x^4 u - f_b \Phi(v_{rel}). \tag{3.20}$$

In order to discretize this, spreading operators are used. Spreading operators are used to interact with a FDS (through connections or interactions). In the simplest case the interaction is applied directly to the grid point, which is defined as

$$J_l(x_i) = \frac{1}{h} \begin{cases} 1, & \text{if } l = l_i \\ 0, & \text{otherwise}. \end{cases} \tag{3.21}$$

Using this, discretizing Equation 3.20 gives

$$\delta_{tt} u_l^n = c^2 \delta_{xx} u_l^n - 2\sigma_0 \delta_t u_l^n + 2\sigma_1 \delta_{t-} \delta_{xx} u_l^n - \kappa^2 \delta_{xxxx} u_l^n - J_l(x_B^n) F_B^n \Phi(v_{rel}^n), \tag{3.22}$$

with

$$v_{rel}^n = \delta_{t\cdot} u_l^n - v_B^n. \tag{3.23}$$

In order to solve for u_l^{n+1} at the bowing position, v_{rel}^n has to be determined. Equation 3.23, however, makes the scheme implicit as the centred difference operator making it dependent on future values of the non-linear interaction. An approximation of v_{rel}^n can be found using an iterative root-finding method, such as the Newton-Raphson method [2] defined as

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{3.24}$$

[2] https://www.sciencedirect.com/topics/mathematics/newton-raphson-method

Chapter 4

Design

In this chapter, the design process and final concept will be discussed. This consists of the work we did in our internship at Oticon Medical, our revisited concept design, and the music training minigame concepts. This chapter is concluded with an overview of the design requirements for the implementation.

4.1 Starting point - Internship Work

As formulated by Oticon Medical, our main goal was to make music training fun and engaging to increase music exposure and maximize the therapy. Therefore, we had to design an experience that is motivating for the users to keep them actively training their perception over a long period. To reach this goal, we examined different game elements for enhancing engagement. The elements that we found include play, challenge, premise, story, immersive environments, rewards, and interactivity [2, 17]. Similar methods are used in gamification practices to facilitate changes in behaviors or cognitive processes [23]. We followed such a gamification approach as we aimed to direct users' behaviors towards training music perception regularly.

4.1.1 Initial concept-design process

We started by roughly defining our target group. Initially, we chose teenagers and young adults as our target group. We chose this group as we envisioned that it would be most beneficial to them, both in terms of music training potential and overall engagement, because of their familiarity with the game-medium.

Inspired by Fullerton's [16] iterative concept design approach, we followed the idea of iteratively refining and evaluating ideas into a final concept that would incorporate both our project and user requirements. This design process ensured that our concept had already received the first round of user feedback to antici-

pate immediate challenges before development started. The design process of our internship is illustrated in Figure 4.1). We explored and validated different game-concepts during our design process, which is described in the rest of this section. The full process can be found in our internship report in Appendix A.

We started by brainstorming on various ideas, touching upon user experience, art style, sounds, interactions, characters, difficulty, etc., and we ended up with around 20 different game ideas. Then, we grouped and selected ideas using a priority matrix with two criteria: 1. how engaging the ideas' game experience would be and 2. how effective the music training aspect of the idea would be (as illustrated in figure 4.1).

After rating all our ideas according to the criteria, four ideas, covering different game types, were rated highest and we decided to explore these ideas further. We formalized these four ideas and created storyboards (see figure 4.1) to further explore and describe the interactions and user experience. Each game concept idea is shortly described below (storyboards and descriptions of the ideas can be found in Appendix A).

1. Sound Squash - *Competitive Multiplayer*

   Catch the ball by matching its pitch to be able to direct it towards the opponent

2. Sound Surfer - *Infinite Side Scroller*

   Adjust your surfboard to match each sound wave's pitch.

3. Aquaria - *Puzzle*

   Guide the water flow through a series of objects to mimic a referenced melody.

4. Uncharted Chants - *Dungeon Exploration*

   Explore and find certain musical sounds to compose and match a melody.

Aiming to assess our concepts further, we chose to collect external feedback and created a questionnaire including both descriptions and visuals for each concept. Our goal here was to gather general insights and feedback on the ideas as well as to confirm which game concept(s) would be most engaging and fun. To get as much feedback as possible, we included both NH and CI users in this process and shared the questionnaire to forums for game designers and to contacts provided to us by Oticon.

For each concept, using Likert-scale questions, the respondents had to rate how engaging they found it, how efficient they found the music training element, and their eagerness to play the game when developed further. We also included open questions for suggestions and feedback for each concept.

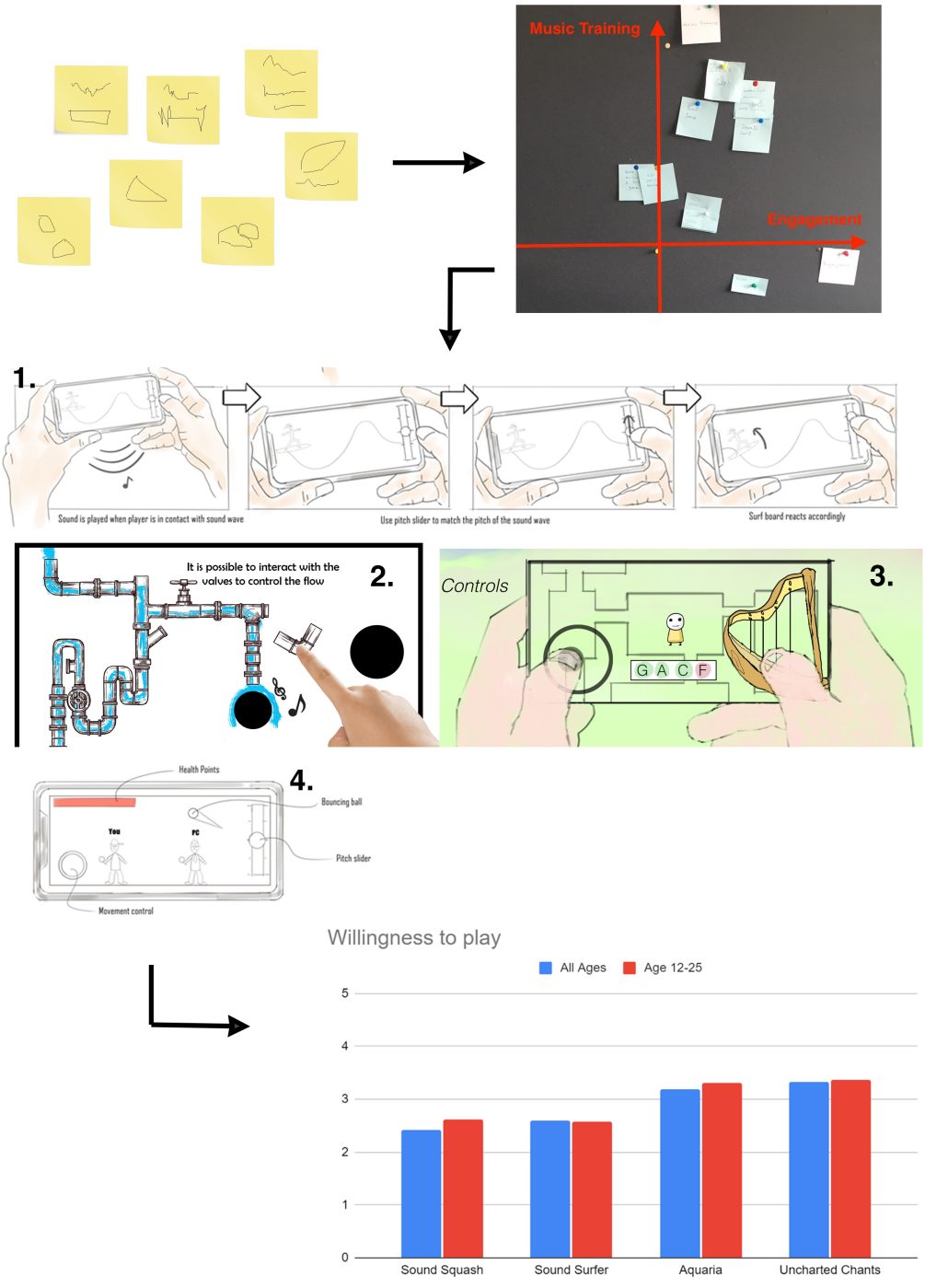


Figure 4.1: Illustration of our design process, going from initial brainstorm through concept-idea assessment, formalization and validation.

We got 113 responses to the questionnaire and respondents were aged from 14 to 56, of which one of the respondents was a CI user (see Appendix A for full questionnaire and results).

In Figure 4.1 the result of the respondents' eagerness to play each game concept is presented. From this can be seen that the puzzle game concept and the dungeon explorer game concept got the highest ratings out of the four concepts.

4.1.2 Concept - Uncharted Chants

The game concept that we decided to develop further was a combination of the two highest-rated concepts in our concept validation. The foundation of our game concept is the dungeon-exploration idea which features several music training minigames that are incorporated into the gameplay (one of the minigames being the puzzle-game concept). The goal of the game is to explore the dungeon to find and defeat different instrument monsters. While playing against these instrument monsters, the user needs to complete a music training exercise, which is related to pitch, rhythm, melody contour, etc. Additionally, at each level, the user needs to find certain items to help them progress through the dungeons. One of these is the sphere fragments that are required to enter the sound puzzle minigame and progress to the next level.

To make exploration interesting and replayable the room layouts of each level are procedurally generated. This introduces a game system that makes each play session unique and potentially engages the user for longer periods. The core concept of having a circular game-play experience (as illustrated in Figure 4.2) is inspired by the rogue-like game genre[1] and aims to have the user repeatedly play through unique levels and restart if they lose. While playing the music training minigames, the user will improve their performance resulting in them progressing further in the game for each play-through. Ideally, the game will include multiple other systems to encourage the player to explore more rooms, such as a shop room where they can buy upgrades, chests, and other rewards. These added systems could prevent the users from feeling stuck when the difficulty is too high, and prevent them from losing motivation as the levels are always different.

4.2 Revisited Concept Focus

We changed the focus of our concept based on experiences and user feedback on the game version from our internship. First, we decided to focus on implementing real instrument sounds and interactions. As we aim to create a game that trains music and expose users to music, we wanted to make the interactions and audio in the game resemble real instruments as accurately as possible. This was also aligned

[1] https://en.wikipedia.org/wiki/Roguelike

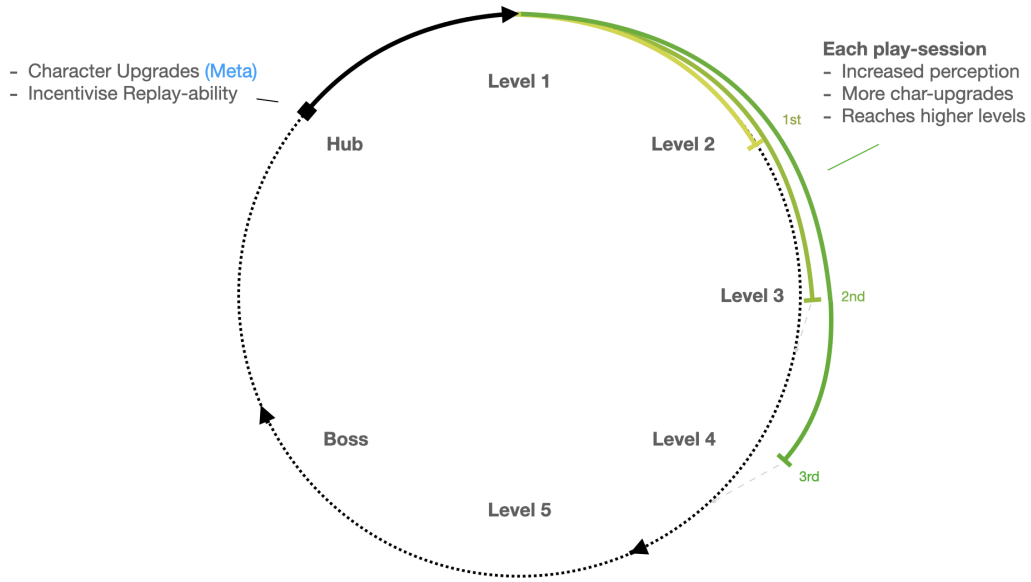


Figure 4.2: The intended gameplay-loop where the user get increasingly further for each playthrough by practicing music perception and upgrading their character.

with Oticon's future work, as they were exploring the usage of more realistic instrument sounds in their music training games. To do so each 'combat' minigame focuses on a different instrument that teaches about its sounds and interaction, and by this indirectly training instrument recognition. To do so as accurately as possible, we investigated the usage of physical modeling for sound synthesis (see chapter 3.2). These models are used to dynamically produce audio that reflects how the user is interacting with an instrument. By generating sound via physical models, the timbre of the virtual instrument changes depending on the interaction. This might contribute to giving the user a more nuanced palette of sounds to associate with a given instrument thus helping them differentiate instruments timbre.

We also decided to switch our target development platform from mobile, which Oticon originally proposed to us, to PC and Mac. As physical models can be computationally heavy, especially implementing more complex ones, mobile platforms would not be sufficient any longer. Next to this, music training games require a certain level of attention and the user must be in a quiet environment. Therefore, it is not ideal to play on the go which eliminates one of the main reasons for developing for mobile platforms. Furthermore, the game concept that we have features many user interface components, such as a health bar, minimap, and currency indicators, which take up screen real-estate. Switching to PC and Mac provides more screen real estate and an opportunity to create a more satisfying character control interaction.

4.3 Music training minigames

Our game concept revolves around five music training minigames, of which four are represented as different instrument monsters. The player has to defeat the instrument monster by completing a music training exercise before returning to exploring the dungeon. After defeating a monster the player is rewarded with musical essence, which is a persistent currency that can be spent to upgrade the player's attributes such as the amount of health or run speed when outside the dungeon (Hub, see Figure 4.2). As the player progresses through the level structure, the instrument monster minigames increase in difficulty.

In Table 4.1 the different minigames and the trained perceptual features are given and are briefly elaborated below.

Minigame	Trained music perception property
Percussion minigame	Rhythm perception via tempo
Harp minigame	Melodic contour recognition
Violin minigame	Timing and loudness contour
Guitar minigame	Pitch perception
Puzzle minigame	Melodic contour recognition

Table 4.1: Overview of minigames and which perceptual properties they aim to train.

Percussion minigame

The percussion minigame is designed to train rhythm perception by evaluating the user's ability to tap into the correct tempo of various rhythmic sample loops. Rhythm perception consists of various components such as rhythmic pattern, meter, tempo, and timing [27]. We chose to focus on evaluating the user's ability to perceive tempo which can be described as the speed or rate of the perceived sound pattern and is related to the cognitive notion of 'beat' [27]. Tempo perception is relatively straightforward for non-musicians and evaluates the user's ability to listen for different rhythmical elements in a musical pattern while specifying the speed of the musical pulse.

Harp minigame

The harp minigame is primarily focused on training melodic contour recognition. A reference melody is played and the user's task is to recreate the reference melody by playing the strings of a harp. To make the minigame more accessible to non-musicians the first note of the reference melody will always be visually indicated.

Next to this, melodies will only consist of two or three notes with large note-intervals in the beginning.

Violin minigame

In the violin minigame, a reference violin sound is played after which the user has to recreate it by moving the violin bow with a sufficient velocity over the string, matching the note duration of the reference. This game is focused mainly on timing as the correct duration should be perceived, but also the dynamics (loudness contour) as the user needs to be able to perceive when the note rises and falls. As this can be tricky, due to the CI user's limited loudness perception, there will be an error margin taken into account that will be scaled depending on the level.

Guitar minigame

The guitar minigame focused on pitch perception training. In contrast to the harp minigame, which trains the user on pitch relationships between notes (intervals), this minigame trains the user on absolute pitch values of single notes. The user's goal is to tune the guitar string to match the pitch of the reference note. We aimed to make this minigame portray a real-life scenario of tuning a guitar by having two simultaneous interactions of plucking the string and turning the guitar peg. Also here, an error margin will be established to make it more accessible for CI users with poorer pitch perception.

Puzzle minigame

This minigame is focused on melodic contour recognition, which is not represented as an instrument monster in the dungeon, but has to solved to progress to the next level. This minigame is inspired by the puzzle game concept (Aquaria) which we decided to incorporate here. The minigame presents the user with several sound-emitting spheres of different sizes corresponding to different tones (larger sphere equals lower frequency). A reference note sequence is played and the user has to draw a path that guides falling music notes through sound-emitting spheres according to the reference melody.

4.4 Difficulty scaling

The game's difficulty should be dynamic and increase as the player progresses. As stated by Mayer & Johnson, digital game-based learning should, amongst other things, accommodate the user's skill level by implementing *"gradual learning outcome-oriented increases in difficulty"*[12]. We aimed to implement a dynamic difficulty system that will tweak the parameters of each music training minigame to adjust the

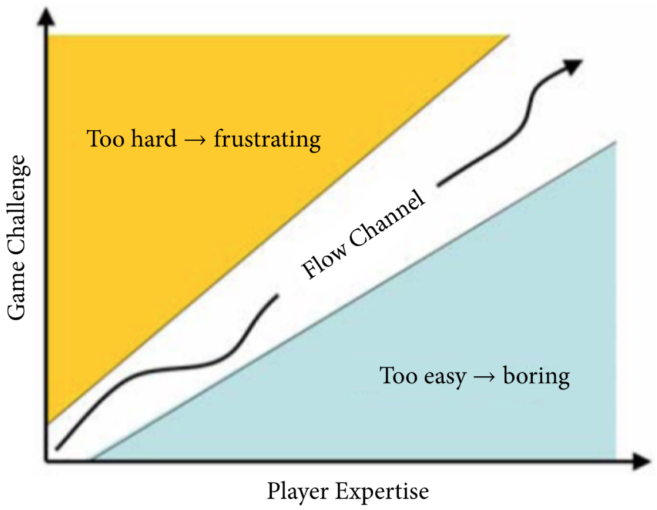


Figure 4.3: The balance of a games difficulty. As the players expertise increases (x-axis) the difficulty of the game (y-axis) should increase as-well to maintain a state of 'flow'. Extracted from: [62]

difficulty according to the level that the user has reached. As described and visualized by Zohaib [62] in Figure 4.3 any game experience needs to have a moderate difficulty level. If it is too easy the user can get bored and if it is too difficult the user can get demotivated.

As proposed by Csikszentmihalyi [42], when the user is not bored nor frustrated she is experiencing a state of 'flow' (see figure 4.3). Optimally, the user should be in a state of flow to maintain their motivation to play. This can be achieved by gradually increasing the difficulty level of the game, at a certain pace, that allows the user to have sufficient time for learning and improving to meet the increase in difficulty[42].

Finding the ideal difficulty level is a challenge because of the huge differences in perception between CI users (elaborated in chapter 2). To tackle this, the game should have a low entry-level, after which the difficulty should scale with higher levels to include better-performing CI users. For this, the minigames' difficulty parameters, including the error margin in pitch, note lengths, tempo, etc., are dynamically scaled. These will be elaborated further in the implementation section.

4.5 Design Requirements

Before starting the implementation phase, we made design requirements to set our priorities (Must Have) and listed the less essential elements (Nice to Have), which is shown in Table 4.2. Apart from the requirements from our internship project, we have real instruments sounds and interaction, language selection, a tutorial,

Must Have	Nice to Have
Minigames training pitch, melody, timbre, and rhythm	Character customization
Dynamic difficulty system	Trophy system
Unique level generation	Aesthetically pleasing artwork
Realistic instrument sounds & interactions	Free-play instruments
Instrument interaction should reflect sound output	Shop system
Character upgrades	Lighting and shadows
Volume control	Post processing visuals
English and Danish language	Hiding unexplored areas
Encourage exploration	
Should run on Windows and MacOS	
Should include a tutorial	

Table 4.2: List of requirements.

and sufficient performance on both PC and Mac as requirements. The Nice to Haves include game elements focused on increasing user engagement but are less essential.

Chapter 5

Implementation

This chapter contains a technical description of the developed application. We start by giving an overview of the project on a macro level, then we will go into the implementation of the Audio Plugins, and finish with the implementation of the game engine. Although the implementation is a continuation of our internship, for completeness this chapter also contains the foundational work made during our internship project. For brevity, we will focus in this chapter on the core implementation, although the full Unity project and all scripts can be found in Appendix E. Both the game's installation files and demo video can be found in Appendix F.

5.1 Overview

Uncharted Chants can be split into two main parts: the game engine, consisting of the framework of the game and in which all the scenes are established and controlled by the player, and the implemented audio plugins, in which most of the audio inside the game is synthesized. We chose *Unity*, [1] as the game engine for our project during our internship, and continued using it during our thesis, both because of our familiarity with this game engine and because of its extensive functionality. Unity is a native C++ game engine, in which users write their code in the C# programming language. These custom-written components, called scripts, are running on Mono [2], an open-source software platform. In Figure 5.1 a block diagram is depicted of the project. Scripts are loaded inside a scene, a place where all or part of the current game is active, and start running whenever the scene is playing. A scene is constructed through *GameObjects*, objects of which the functionality is determined by the attached Components. This, for example, can be a light source, audio source, or a sprite for a 2D visual. By using the camera component, the created scene is captured and displayed to the user. The specific properties of

[1] https://www.https://unity.com/
[2] https://www.mono-project.com/

the camera and the way it is rendering the visuals to the display are dependent on the used render pipeline, further discussed in 5.3.4. Besides the visual aspects of the application, the audio is set up through the *Audio Source* component. This component can be used to play audio clips, which are then routed and mixed inside the *Audio Mixer* and perceived by an *Audio Listener* component. When it comes to audio, however, Unity can be quite limiting, especially when it comes to synthesizing sounds in real-time. One way to do this in Unity, which we used during the previous iteration, is implementing the algorithms in C# using the *OnAudioFilterRead()* method. This approach, however, would likely not work for more complex models, as it lacks the required efficiency. This is why we decided to set up the audio in a modular way, where we would write audio plugins in C++ for each instrument and import them into our Unity project. This is done using the *Audio Mixer*, which apart from mixing and mastering, can be used to add third-party audio plugins to the project, and can be controlled through scripts using *exposed parameters*. The specific details about the audio setup and each of the written audio plugins will be discussed in the following section.

5.2 Audio

In Figure 5.2 a more detailed overview of the audio architecture is illustrated. The *Audio Mixer* consists of three groups, one for background sounds (Ambient), one for sound effects (SFX), and one group for the sounds inside the minigames (Music Training). This approach allowed us to let the user change the volume of each of the groups according to their preferences from within the settings menu (see Figure 5.3).

The music training group consists of subgroups each with a different audio plugin loaded onto it. We made different audio plugins as the desired instruments required different synthesis techniques. In Table 5.1, each of the instruments, the plugin names, and the synthesis technique are listed. For each of the algorithms, a prototype was made first in *MATLAB* [3]. Then we made the real-time implementations in C++ using *JUCE* [4], of which both implementations can be seen in Appendix B. *JUCE* is a framework for developing audio applications with multi-platform build support, which we chose because it also includes build support for Unity. Communication with Unity works by specifying audio parameters inside the plugins. Audio parameters use the *AudioProcessorParameter* base class and are used to control parameters from an external software like digital audio workstations, or in this case Unity. Setting up an audio parameter is done by defining a default value and limits (minimum and maximum value) in case it is a *float* type.

[3] https://nl.mathworks.com/products/matlab.html
[4] https://juce.com/

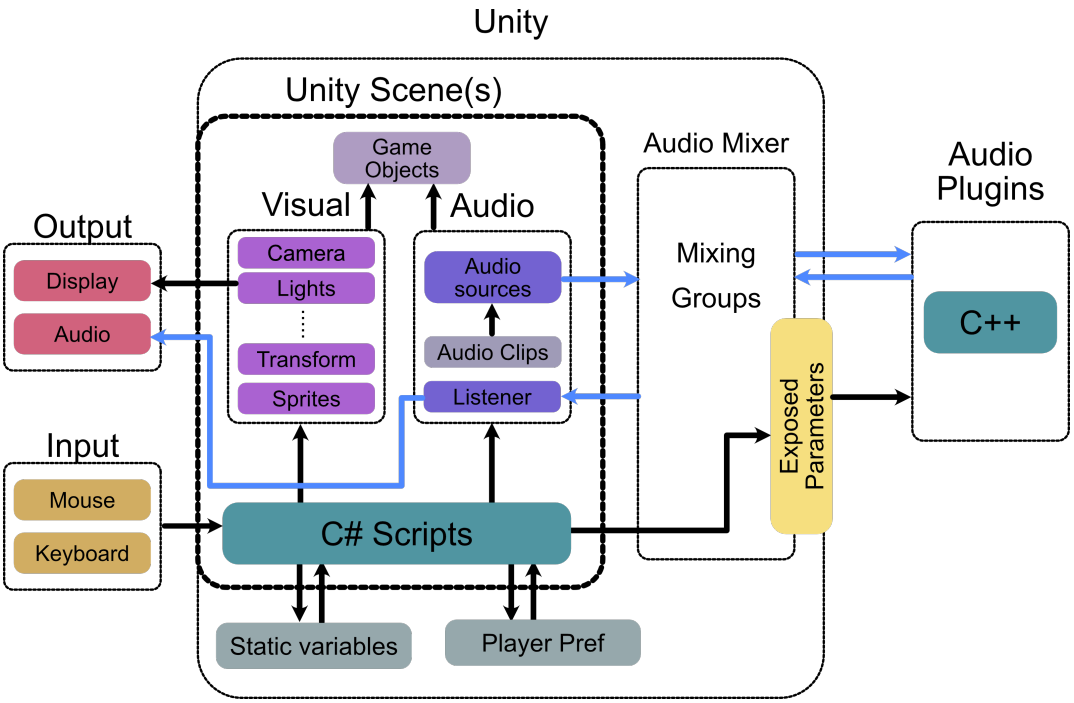


Figure 5.1: Block diagram overview of the project. Whenever a scene is loaded, scripts control certain properties of *GameObjects*, called components. Depending on the program, scripts are controlled by user input or by certain game states. These include static variables or saved player preferences, that are used to save certain parameters whenever scenes are closed or other scenes are loaded. The visuals inside the scene are then rendered using a camera component and displayed. Audio is routed through the *Audio Mixer*, where external audio plugins are loaded and controlled using exposed parameters from scripts. Finally, after the audio is routed through the *Audio Listener* component, the audio is routed to the user's output device.

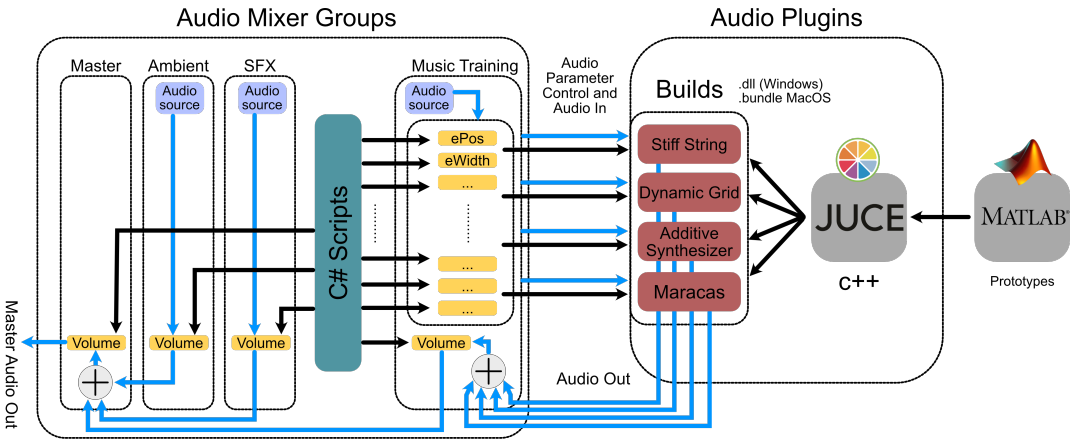


Figure 5.2: Audio routing and control block diagram. The yellow blocks indicate the exposed parameters controlled by C# scripts. The blue lines indicate audio routes.

Figure 5.3: Settings menu with volume control for each group, which can be opened at all times.

An example code snippet is given below for setting up an audio parameter for the fundamental frequency.

```
addParameter(fundFreq = new AudioParameterFloat("fundamentalFreq", // parameter ID
        "Fundamental Frequency", // parameter name
        20.0f,          // minimum value
        10000.0f,       // maximum value
        220.0f));       // default value
```

Exporting the plugin to Unity will create a .bundle file for MacOS or .dll file for Windows, which then can be loaded onto the *Audio Mixer*, after which the audio parameters show up in the inspector. This can be seen in Figure 5.4, where the audio parameters are shown when built for both a standard audio plugin (VST) and a Unity plugin. These parameters in Unity can then be 'exposed', allowing them to be controlled from within C# scripts. For example, one could change the fundamental frequency from a script by referencing the *Audio Mixer* and using the *SetFloat* method:

```
audioMixer.SetFloat("StiffString_f0", (float)Remap(f0, 20.0f, 10000.0f, 0.0f, 1.0f));
```

Here, a mapping function is required as the limits of the audio parameters are converted into values between 0 and 1 (see Figure 5.4).

In Table 5.1, each of the instruments, the plugin names, and the synthesis technique are listed. In the following sections, each of these plugins will be briefly described.

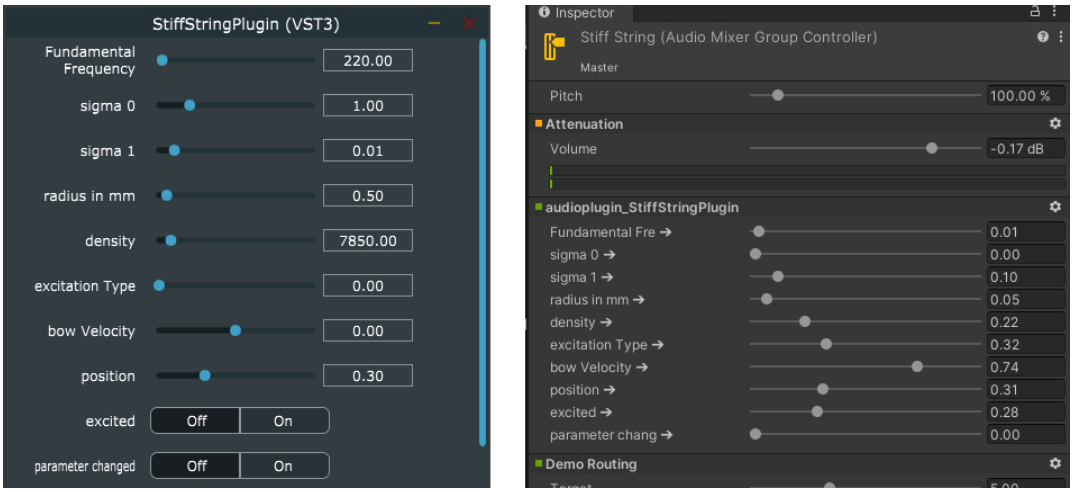


Figure 5.4: Communication between C++ audio plugins and Unity. On the left are the audio parameters in the editor of the plugin (VST), and on the left how they show up when build for Unity. Notice that all parameters are converted to a float between 0 and 1, even if the parameter is a *boolean* type. Exposed parameters are indicated with the arrow next to the parameter name.

Instrument	Audio Plugin	Technique
Percussion	Maracas	PhISM
Harp	StiffString	FDS
Violin	StiffString	FDS, Static friction model
Guitar	DynamicString	FDS, Dynamic Grids
Synthesizer	AdditiveSynth	Additive Synthesis

Table 5.1: Each of the audio plugin and used synthesis technique.

5.2.1 Maraca

The maraca instrument is played inside the rhythm perception training minigame to provide audible feedback to the player's input. We modeled the maraca using a technique called Physically Informed Sonic Modeling (PhISM), a method by Cook [10]. This method uses stochastic events to model percussive instruments. As the sound of maracas is characterized by the random interactions of the beads inside, it can be modeled by adding a grain of sound for each collision. Collisions are then predicted using a certain probability that is dependent on the number of beads inside the maraca. The total envelope of the sound is dependent on the number of collisions and the excitation curve of the maraca, which resets each time there is an excitation. By applying this envelope to a generated white noise signal, the basic shape of the sound can be created. Then, the resonance frequency of the shell can be modeled using a bi-quad filter [5]. The maraca plugin is controlled using one *AudioParameterBool* that excites the maraca whenever it is *true*.

5.2.2 StiffString

The StiffStringPlugin is an implementation of the damped stiff string model using FDS (described in 3.2.2) and is used to model both the harp and the violin. The physical parameters (damping and stiffness coefficients) as well as the fundamental frequency are audio parameters and can be controlled through Unity. The string can be excited by either plucking or by bowing the string, allowing us to synthesize both the harp and the violin, by changing the dedicated audio parameter. The plugin consists of two classes, a *StiffString* class and a *Bow* class. The *StiffString* class calculates the grid size and spacing at startup, and recalculates it whenever an audio parameter has been changed (e.g. the fundamental frequency). For a FDS model to run in real-time, the scheme has to be calculated and updated for each sample. This is done in the *getNextSample()* method. Here, the next state of the system is calculated using the simply-supported boundary condition (see 3.2.2), an output is retrieved, and the states are updated (making $u_l^{n-1} = u_l^n$ and $u_l^n = u_l^{n+1}$).

```
1 double StiffString::getNextSample(float outputPos)
2 {
3     calculateScheme();
4     double out = u[0][static_cast<int> (round(outputPos * N))];
5     out = out * eScalar;  // scale to make excitaiton audible
6     updateStates();
7     return out;
8 }
```

Whenever the string is plucked the public method *exciteSystem(double amp, float pos, int width)* is called, which excites the string using a raised cosine with the specified

[5] https://www.dsprelated.com/freebooks/filters/BiQuad_Section.html

amplitude, position, and width.

The *Bow* class includes the static friction model described in 3.2.3. Whenever the excitation type is set to bowing the public method *setExcitation()* is called each sample.

```
void Bow::setExcitation(std::vector<double*>& u, float bowPosition,
    double bowVelocity)
```

In this method, the relative velocity, dependent on the *bowVelocity*, is calculated using the Newton Raphson method. Then the excitation is calculated and applied to the grid point corresponding to the *bowPosition*.

5.2.3 DynamicString

Using the previous StiffStringPlugin could also work for modeling a guitar, but because the guitar tuner minigame requires the pitch to be smoothly adjusted in real-time, a different approach was necessary. Smooth parameter changes, such as fundamental frequencies, are tricky for FDS methods as the grid has a fixed amount of points. To create a dynamically changing grid, we implemented the method of dynamic grids by Willemsen [57]. The principle of this approach is making the number of grid intervals (N) fractional, which is realized by splitting the system into two connected subsystems. Then smooth parameter changes can be made by dynamically altering the distance between the systems. Whenever this distance, denoted by α, is larger than one, a grid point is added to the system, and when it is smaller than zero, a grid point is removed. The inner boundaries of the two systems are calculated through virtual grid points. These grid points are outside of the boundaries and calculated using interpolation with the other system. We implemented this method for the damped string and created a *DynamicString* class. The main method that is called for each sample is the *getNextSample()* method, which is shown below.

```
double DynamicString::getNextSample(float outputPos)
{
    alpha = N - floor(N); // calc distance between two systems

    // Check if there is a need to add/delete grid points
    if (floor(N) > floor(N1)) addPoint();
    else if (floor(N) < floor(N1)) removePoint();

    getVirtualGridPoints();
    calculateScheme();
    double out = getOutput(outputPos);
    updateStates();
    return out;
}
```

Just like the harp, the string is excited using a raised cosine, which is controlled using an audio parameter.

5.2.4 AdditiveSynth

For the puzzle game, we used a more synthetic synthesis technique as the sound-emitting objects were abstract and had no direct connection to a real musical instrument. We chose additive synthesis (see 3.3) because of its flexibility in the creation of different timbres. Because the minigame required multiple notes to be played at the same time we implemented a *SynthVoice* class. We created six instances of this class allowing six-voice polyphony. Each instance of the *SynthVoice* class has a fundamental frequency, 16 harmonics, and its own ADSR envelope using the JUCE ADSR class [6]. We added the envelope to create smooth amplitude curves, which both get rid of possible audible artifacts (i.e. clicks) and made it sound more natural. The main method in each voice class is *getNextSample()* method, which is shown below.

```
1  double SynthVoice::getNextSample()
2  {
3      double out = 0.f;
4
5      for (int h = 0; h < numHarmonics; h++)
6      {
7          if (f0 * (h + 1) < nyquist) // filter out harmonics above
       nyquist
8          {
9              out = out + adsr.getNextSample() * gainVector[h] * sin(
       currentAngle[h]);
10         }
11         currentAngle[h] += angleChange[h];
12
13         if (currentAngle[h] > 2.f * double_Pi)
14         {
15             currentAngle[h] -= 2.f * double_Pi;
16         }
17     }
18     return out * averagedGain;
19 }
```

Here, the output is created by adding up all the harmonics up to the Nyquist frequency (half the sampling frequency) to avoid aliasing. Then, the angle of each sinusoid is updated according to the angle change (dependent on the frequency of the harmonic). Keeping track of the current angle allowed us to create smooth frequency changes if necessary.

[6] https://docs.juce.com/master/classADSR.html

5.3 Unity

5.3.1 Scene Management

Applications made in Unity are composed of one or more scenes. By organizing different parts of a game in different scenes it is possible to distribute functionality and make the project more modular to increase effectiveness, error-tracking, and bug-fixing effectiveness. This workflow can significantly decrease the time spent managing and organizing the work of multiple developers as well as improve the execution time of the final application.

As illustrated in figure 5.5, the main gameplay loop of Uncharted Chants consists of a sequence of scenes starting from the hub scene to the level-progression-map scene and then to the main dungeon scene. To control the progression through levels and the dynamic increase of difficulty the static variable *int currentLevel* is used. From here, either the player successfully collects the sphere-fragments and finishes the level which results in the *currentLevel* being incremented or the player loses (lost all health) and *currentLevel* is reset to 1. When the player finishes a level the level-progression-map scene is loaded which, on startup, references *currentLevel* to check which level-transition to animate. The same system applies when the main dungeon scene is loaded as it also references *currentLevel* on startup to check which level layout to generate as well as which difficulty the instrument-monsters mini-games should have. If the player loses or completes all levels, the meta-hub scene is loaded again.

The main scenes which hold the majority of the game, as well as their key functionality, are elaborated in the following sections. The game includes some additional scenes (main menu, settings menu, and tutorial) but is not a part of the main gameplay loop.

Meta-hub Scene

The hub-area scene is a 'home' environment where the player is in between play sessions. Here the player can do different activities, like permanently upgrading their character (for the cost of musical essence), watching tutorial videos, viewing obtained achievements, and entering the dungeon. This scene is loaded when the game is started, or whenever the player loses or successfully progresses through the final level.

Level-progression-map Scene

The level-progression-map scene has the purpose of presenting a visual overview of the player's progression through the levels of Uncharted Chants to give the player a sense of progression when playing. The scene is loaded every time the

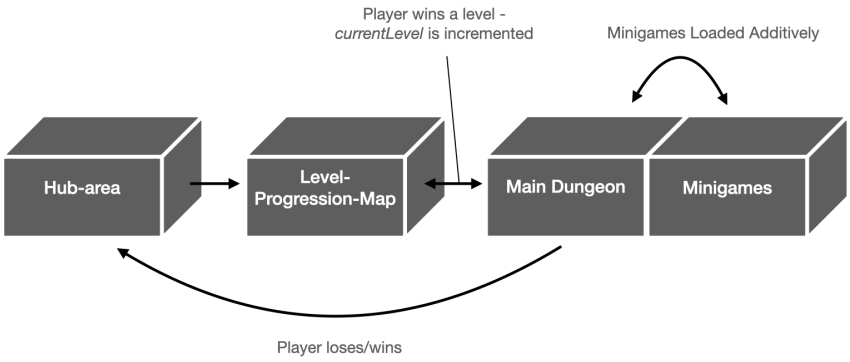


Figure 5.5: The main sequence of scenes to maintain the game-play loop. Starting from the hub area in which the player can enter the dungeon. When entering the dungeon, the level-progression map scene is loaded. In the dungeon, the minigame scenes are loaded/unloaded additively. Whenever a level is completed the level-progression map and the next level are loaded sequentially. Whenever the player loses (or completes all levels) the hub is loaded again.

player starts a new level in the dungeon or, as a transition, when they complete a level and go to the next level. The scene is non-interactable and plays an animation showing the character progressing to the next level.

Main Dungeon Scene

The main dungeon scene holds the core of the game's functionality. When this scene is loaded, a new unique room layout is generated at runtime and all the instrument-monsters, sphere fragments, chests, special rooms as well as the player character are instantiated in specific rooms depending on scripted behavior in the *PostProcessing* script. The player controls the character in the room layout to find the sphere fragments as well as interact with instrument monsters and other items to collect currencies and complete the level.

Minigame Scenes

The minigames-scenes hold the functionality for specific music-training exercises of the percussion, harp, guitar, violin, and sphere puzzle. When the player interacts with a certain type of instrument monster in the main dungeon scene, the respective minigame scene is loaded additively (i.e. on top of the main dungeon).

Whenever the player wins, this scene is unloaded, allowing the player to continue controlling the character in the main dungeon scene.

5.3.2 Level Generation

The level generation algorithm runs whenever the main-dungeon scene is loaded. It is responsible for generating a unique room layout and instantiating items and instrument monsters. For this, we have utilized a procedural level generator Unity-asset called Edgar[7], and customized it to our needs. The generation algorithm uses mainly two types of inputs: the room templates, which are the building blocks used by the algorithm, and the level graph, which defines the structure of a level.

Room templates Design

A room template, illustrated in Figure 5.6 defines how the room looks and what the physical boundaries are. The room-templates, which are prefabricated (prefabs), each have a *RoomTemplateSettings* and a *Doors* component. Here, the *RoomTemplate-Settings* script keeps track of the validity of the room template's border wall, and the *Doors* script is used to add door positions to this wall. These door positions are possible sockets where the algorithm might connect a door to another room or to a corridor that leads to another room. The design of the room is established using *tilemaps* for each layer (e.g. walls, floor, windows, tables, etc.). To create enough variety, we designed each room with custom shapes and different content. To engage the player to keep exploring new rooms, we implemented a fog effect that hides the room in the beginning of a level. For this, each room also has a fog layer that is removed when colliding with the player character.

Level-graphs

The level graphs define the structure of the dungeon, including how the rooms are internally connected and where the special rooms (spawn-room, shop-room, puzzle-room) are located. An example of a level-graph structure can be seen in Figure 5.7, which is the structure of the third dungeon level.

On *Start()*, depending on the level (encapsulated by the static variable *currentLevel*) a different level graph is selected. In total, we designed five different level graphs (one for each level), with increasing difficulty. These level graphs have an increasing amount of rooms and a more complex structure for each level. This scales the difficulty progressively as it requires the player to explore more rooms. In the level graph, depicted in Figure 5.7, the rooms are represented by the boxes labeled 'Room', and lines represent corridors between rooms. When generating, the

[7] https://ondrejnepozitek.github.io/Edgar-Unity/

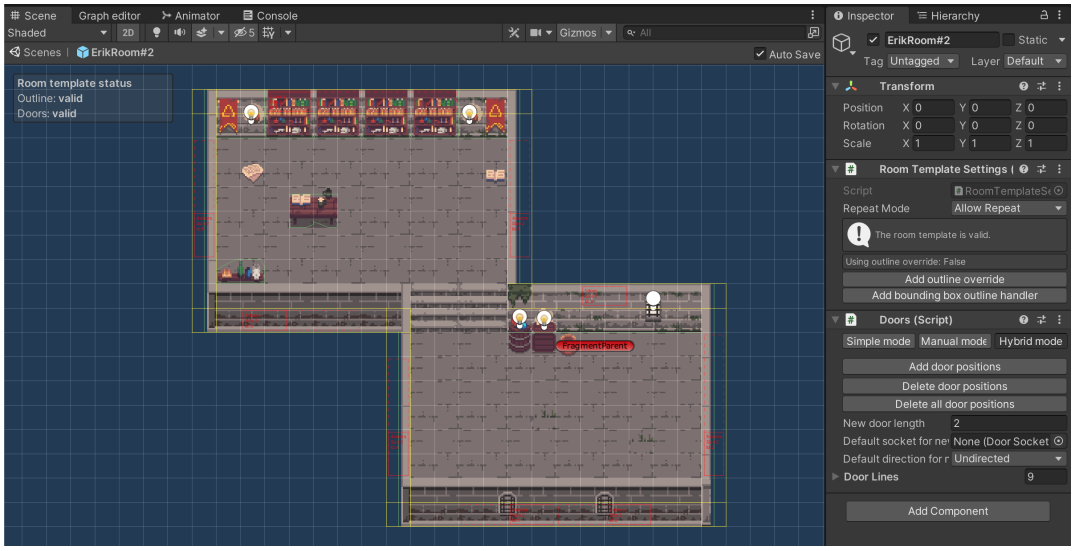


Figure 5.6: How a room-template is designed in the Unity editor. Yellow boxes around the edge represents the border-wall and red squares represent each door-position.

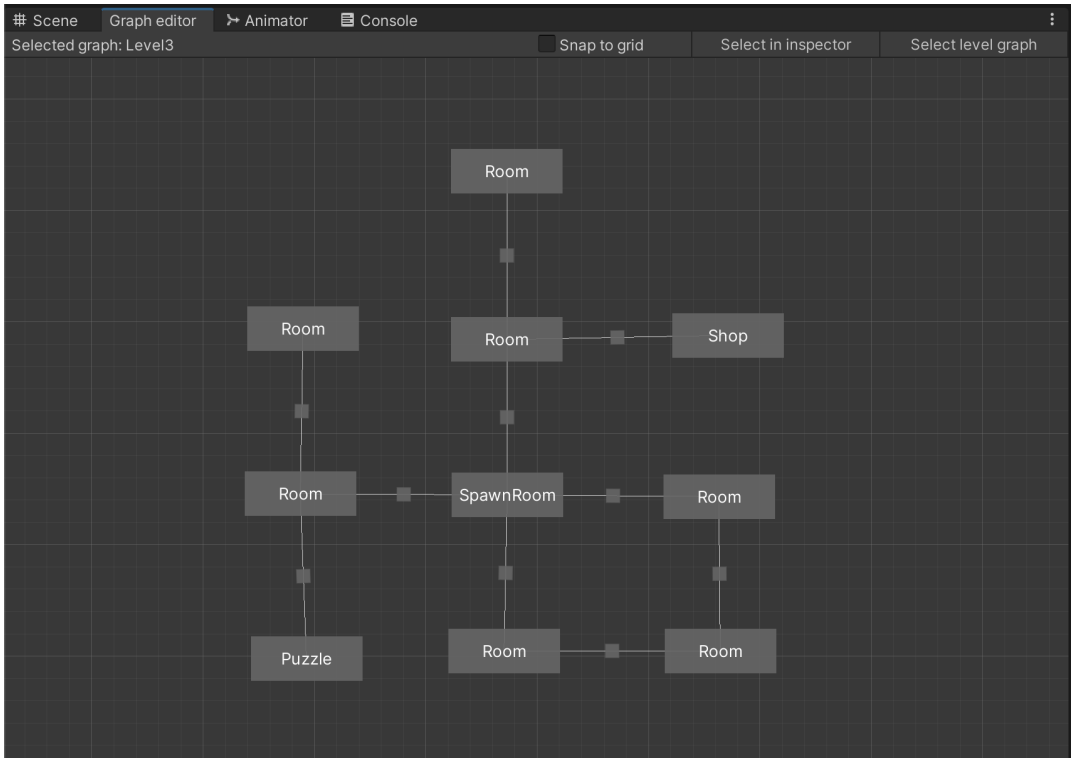


Figure 5.7: Level-graph for level 3. Structure of how rooms should be connected with corridors in a generated room-layout.

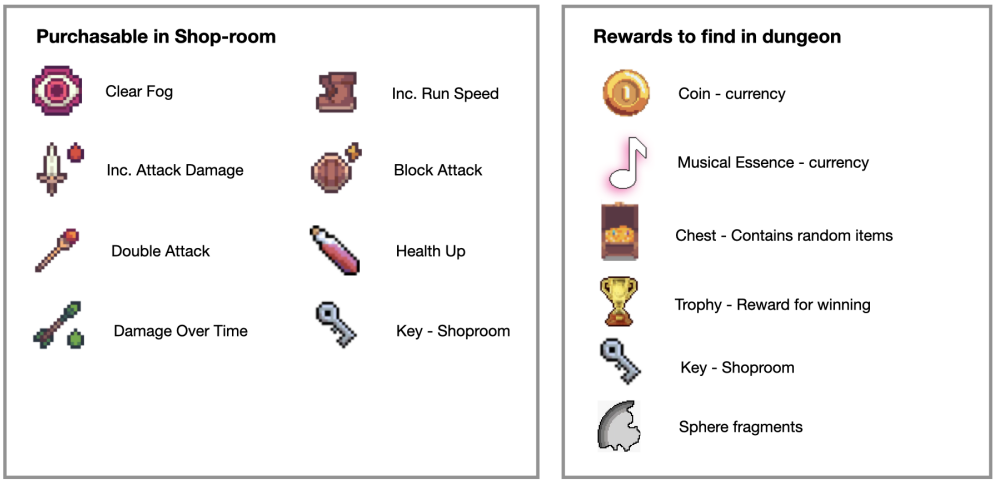


Figure 5.8: Lists of various items, and their functionality, that can be bought in the shop room or found in the dungeon.

algorithm picks one random room from a list of specified rooms (room set). The special rooms (spawn room, puzzle room, or shop room) are indicated as boxes with text referring to their functionality. These special rooms are always the same and are therefore consistently found in the level graphs across all levels. The level-graph setup enables us to generate unique levels, using random room picking, yet keeping a sense of consistency as the structure of the level graph remains the same for a given level. In Figure 5.9 a couple of generated dungeon room-layouts are illustrated, all generated from the third level-graph in Figure 5.7.

Dungeon generation post-processing

We added a customized post-processing script to the Edgar dungeon-generation system which executes a block of code right after the dungeon layout has been generated. This post-processing logic is where we instantiate instrument monsters, sphere fragments, shop items and chests. To make sure each level is a new and unique challenge for the player, random instrument monsters are instantiated in each room. Also, the sphere fragments and chest are instantiated in random rooms. The chest is implemented to further reward the player for exploring the dungeon and will provide the player with coins when opened. The shop room includes random item upgrades which are purchasable for the player. These items can upgrade the player in various ways, like increasing the player's run speed, and are priced according to their effectiveness. The different items that the player can

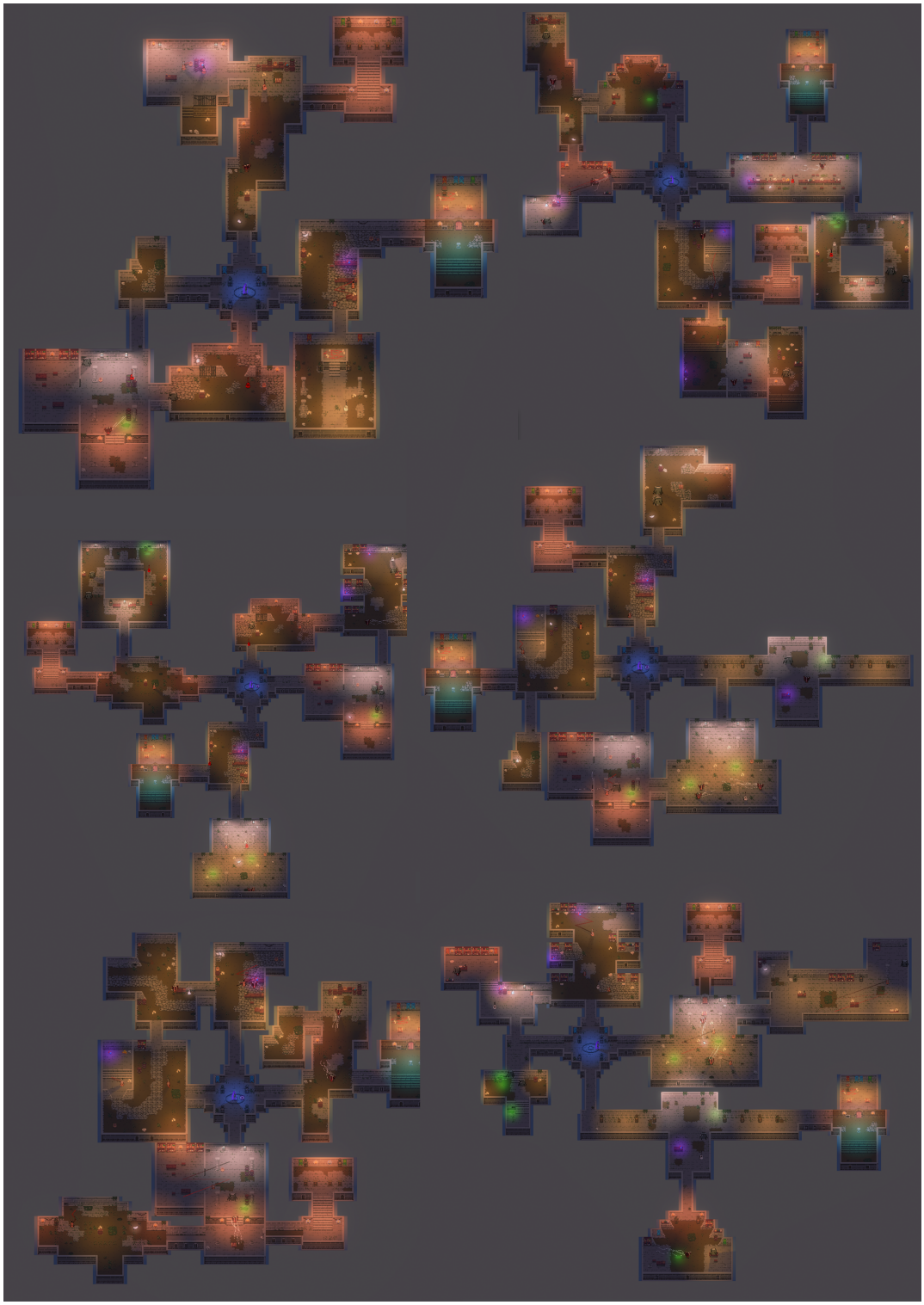


Figure 5.9: Examples of room-layouts that were all generated from the level-graph of level 3 presented in Figure 5.7

Minigame	Enemy Data	Higher difficulty
Percussion	Bpm error margin	Decreases
Harp	Number of notes	Increases
	Interval time	Decreases
	String interval distance	Decreases
Violin	Number of notes	Increases
	Note length error margin	Decreases
Guitar	Interval distance	Decreases
	Pitch error margin	Decreases
	Time to react	Decreases

Table 5.2: Each training minigame and difficulty control

obtain either by exploring the dungeon or visiting the shop room is displayed in Figure 5.8.

5.3.3 Difficulty Scaling

To make sure that we can dynamically change and tweak the difficulty of each minigame we used Unity's *Scriptable Objects* [8], which are data containers often used to provide data to prefabs. We used this principle by creating an *EnemyData* object that defines the difficulty of the instrument monsters using a set of parameters (depicted in 5.10). These parameters are related to the general difficulty, including the health and damage of the enemy, but also the training difficulty. In Table 5.2 an overview is given of all the parameters that are related to the training difficulty, and how they change for higher difficulties. To create small variations between enemies, we created a list of *EnemyData* objects, each with slightly different parameters. Then when the dungeon is created, each instrument monster gets a random *EnemyData* object assigned from a list that corresponds to the current level.

This system allowed us to tweak the difficulty of each minigame instance and each level, and add more *EnemyData* objects to increase variety or add more levels. This modular system prevented us from changing the main functionality scripts of the training minigames, allowing us to test the difficulty of the game and change specific minigames parameters when necessary.

[8] https://docs.unity3d.com/Manual/class-ScriptableObject.html

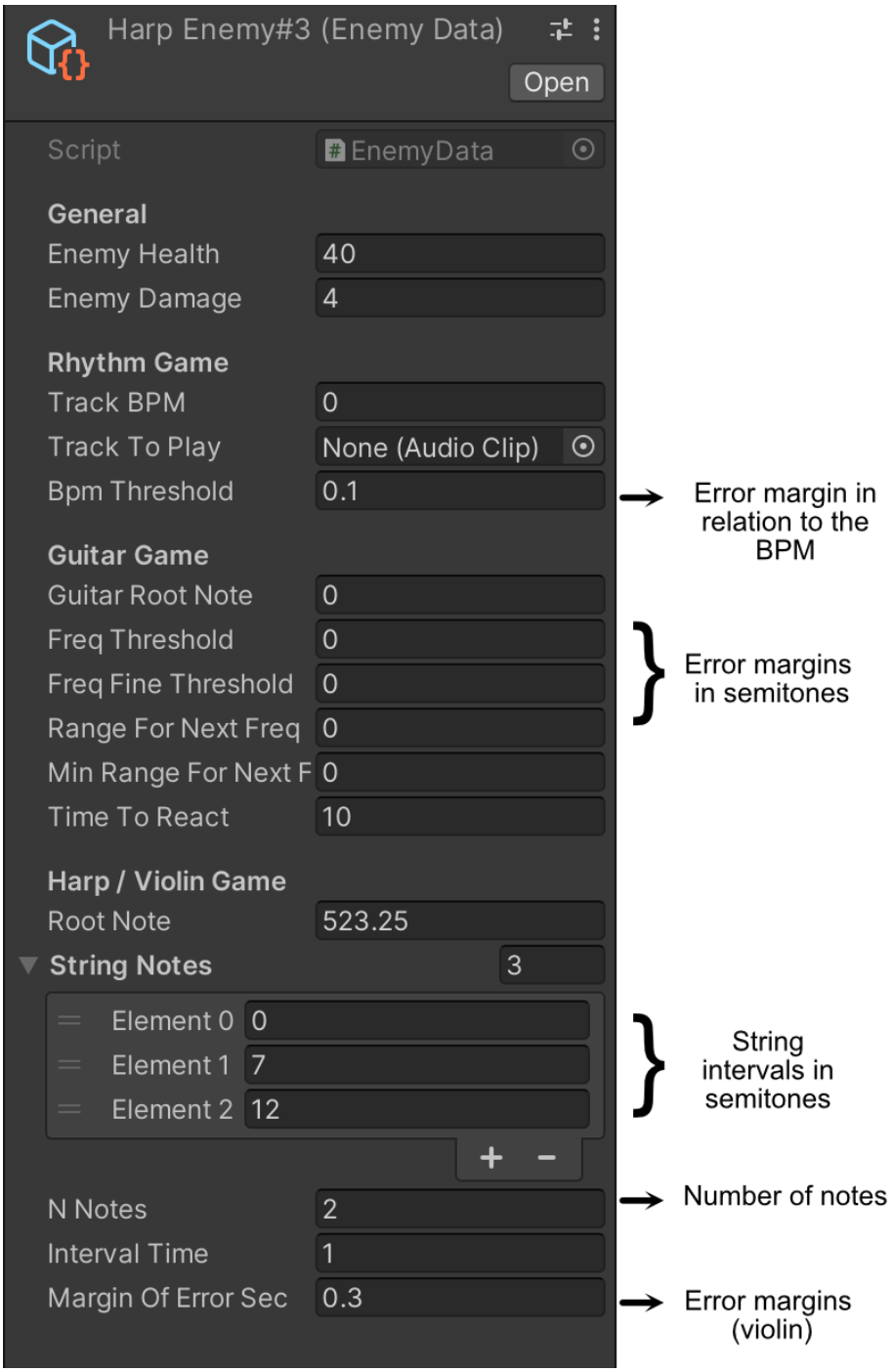


Figure 5.10: *EnemyData* scriptable object for the Harp instrument presented in the Unity editor. The inspector view of the scriptable object shows all the parameters which are used to set the difficulty of all music training instrument-monsters.

Figure 5.11: Sprite sheet of the player's run animation.

5.3.4 Aesthetics

Game art

We implemented the game using two distinct graphic styles to create a more clear distinction between the music training and the non-music training game experience. For everything outside the minigames, we used lower-quality pixelated sprites. Some of these we have extracted from other sources, like the Overgrown Pixel Dungeon Tileset [9], and some we have designed ourselves, including the instrument monsters and the player character. To create the animations we designed multiple frames of a character and put them into sprite sheets. An example of this is the sprite sheet of the player's walking animation can be seen in Figure 5.11. To create a more distinctive art style for the minigames, we created vector-based sprites, allowing us to easily resize them without losing quality. In Figure 5.12 these sprites are shown for the player character. To separately animate each of the character's body parts, they have been split and assembled back in Unity. All designed game art can be found in Appendix C, including the sprite sheets, minigame sprites, and icons and logos that we designed.

Rendering

Apart from sprite design, another way to create a more unique style is by changing the way the sprites are rendered by the camera. In Unity, the way the camera renders the content of a scene is dependent on the used rendering pipeline [10]. For our project, we chose the Universal Render Pipeline (URP) [11], a scriptable render pipeline that allows easy customization and supports multiple platforms. Using this pipeline we implemented a package called *Lights 2D* [12], which provides 2D optimized lighting to be added for sprites. This package allowed us to create distinct

[9] https://foxdevart.itch.io/overgrown-pixel-dungeon-pack
[10] https://docs.unity3d.com/Manual/render-pipelines.html
[11] https://docs.unity3d.com/Manual/universal-render-pipeline.html
[12] https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.1/manual/Lights-2D-intro.html

Figure 5.12: Sprites designed for character inside the minigame. All body parts are exported separately in order to create independent animations for each.

looks for the different objects inside a scene by adding lights with varying colors, intensities, shapes, and sizes, depending on the light source it is representing. We also used it to create different atmospheres between scenes as well, by using different global lights with different intensities to establish a darker environment for the dungeon and a more vibrant atmosphere for the hub area. Additionally, we used this package to make the scene come alive by giving both the player character and the instrument monsters shadows using the *Shadow Caster 2D*. To further create discrimination between scenes, we used URP's post-processing using different volumes. We attached different effects to these volumes, such as *Bloom*, *Vignette*, *Tonemapping*, and *Chromatic Aberration*, depending on the scene. In Figure 5.13 three main atmospheres are depicted, all with different light intensities and post-processing volumes.

5.3.5 Minigames

As has been described in chapter 4, we have designed five minigames, of which four are played whenever the player enters combat with an enemy, and one is the sound puzzle that needs to be solved to proceed to the next level. Although the minigames all work in a slightly different way, the four combat minigames are all set up similarly to a certain extent. From Figure 5.14 the structure of the combat minigames can be seen. To create a seamless transition from the dungeon to the minigames, the minigame scene is loaded on top of the dungeon, and moved

Figure 5.13: The hub area, dungeon, and the final boss level. Each of these scenes have different 2D lights and post processing effects to create the different atmospheres.

Figure 5.14: The basic structure of each combat minigame with the player character on the left and the enemy sprite on the right, each standing on a floor and with a health bar above them. To increase the focus on the minigame, the background has been slightly darkened.

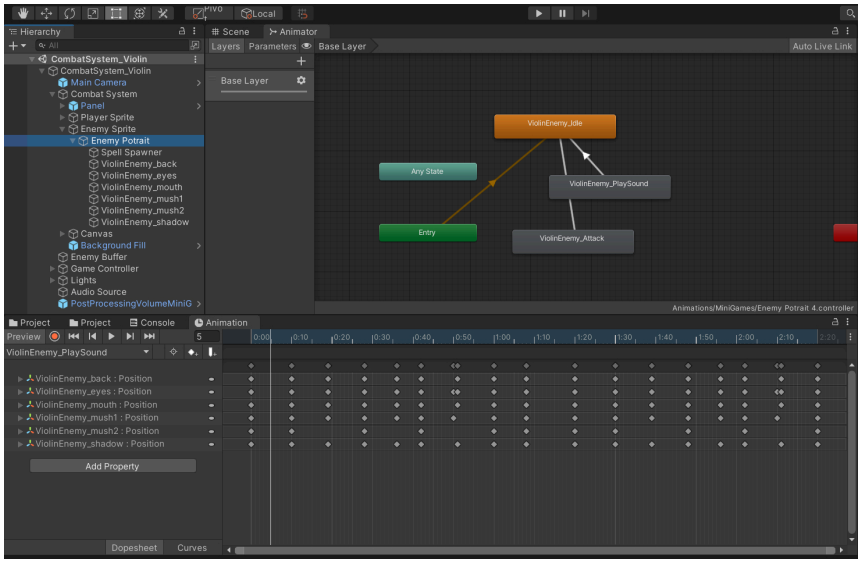


Figure 5.15: Creating animations in Unity. To the left, the scene hierarchy can be seen, with as selected object the enemy portrait of the violin enemy, on which an *Animator* component is attached. On top, the Animator window is shown, in which the different animations are visualized as well as the transitions between them. In this case, there are three animations: one default animation (idle), one for attacking, and one for playing a sound. Then at the bottom, the Animation window is shown, where animations for each sprite can be constructed using so-called keyframes.

to the player's location. However, to shift the focus to the minigame itself, the background is slightly darkened. Each of the combat minigames consists of a combat system GameObject. This parent object contains the basic structure for each minigame, including the floor, player portrait to the left, the enemy portrait to the right, and the Canvas, which includes the health bars of both the player and the enemy. To animate each of the separate images from which the player and enemies are made, each portrait consists of multiple child objects, each with a *SpriteRenderer* component to render one image. This can be seen from the project hierarchy to the left of Figure 5.15. Then on the portrait itself, an *Animator* component is attached, which is used to trigger different animations from script, and handles the transitions between the different animations. Then, the animations themselves are constructed from the Animation window, where each child's sprite is separately animated using keyframes.

The basic game mechanic of shooting a spell that damages the enemy whenever the user has correctly executed the listening task, or reversely, the enemy shooting a spell whenever the player has incorrectly executed the task, is set up in the following way. The portraits for both the player and enemy consists of a *Spell Spawner Handler* script, used to cast a spell on the opponent by calling the *CastSpel()* method. Each of the portraits also includes a *Circle Collider2D* and a *RigidBody2D* component. These components are using Unity's 2D physics system and can be used

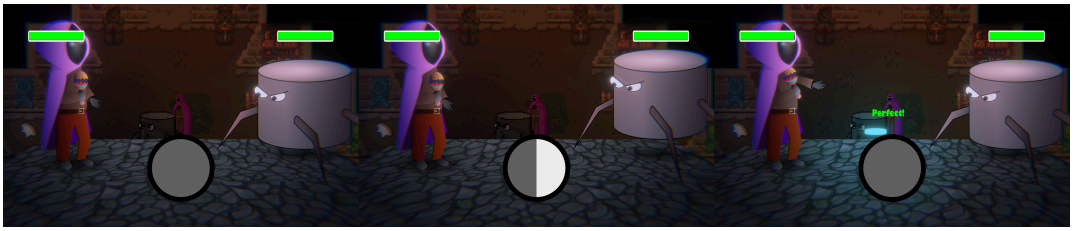


Figure 5.16: The percussion minigame.

to detect collisions, or triggers in this case, whenever it hits another object with a 2D collider. To detect collisions with the spell, each of the portraits also contains a *ReceiveDamageHandler* script, of which the *OnTriggerEnter2D()* method is called each time a spell hits its collider, after which the health bar is updated according to the opponent's damage. The minigame continues until either the player or the enemy has run out of health, in which case the minigame is unloaded. For each minigame, there is a custom game controller script, that specifically controls how the game behaves and when spells are spawned, which will be discussed below.

Percussion

In the percussion game, a drum loop is played and the player needs to tap in the right tempo of the song. The first implementation of this minigame was made during our internship, which seemed to be very inconsistent in evaluating the rhythm correctly. In this implementation, the beat was evaluated with respect to the current beat of the song. We found out however that due to input lag (especially for Windows devices) tapping in the beat would often be evaluated as incorrect because the keypress would be observed after the beat. This would result in the user needing to tap in off-beats, which felt very unnatural. In the new implementation of the percussion game, instead of evaluating if the tap was exactly on, or close to the beat, now the beats-per-minute (bpm) of the tapped tempo is calculated and compared to the bpm of the song. The main script controlling this is the *BPMController*. From within this script, the bpm of the song is extracted from the *EnemyData* and the corresponding song is played at startup. To give the player a reference of the right tempo, a circle starts to pulsate on the beat of the song. For this, the *DetectBeat()* method checks if there is a beat detected for every frame, by starting a timer and comparing this to the beat interval of the song. When detected, the circle is expanded of which the magnitude decreases with each successful evaluation, and increases with each unsuccessful evaluation. This is implemented to make sure that the player does not solely rely on visual cues and has to focus on listening. Each time the player taps the maraca is excited by using the exposed parameter *Maracas_excited* and setting it to *true*. To provide the user with feedback on the current beat in the measure, the circle partially fills

Figure 5.17: The harp minigame. After the harp enemy played the reference melody, the player's harp becomes interactable and the first string that needs to be excited becomes indicated with an arrow. Then, the other string is played, and the played melody is evaluated as successful.

up for each tap as well. After four taps, the bpm is evaluated. This is done by calculating the average time in between the four beats:

```
float averageTime = ((tapTime[1] - tapTime[0]) + (tapTime[2] - tapTime
    [1]) + (tapTime[3] - tapTime[2])) / 3;
evaluatedBpm = (float)System.Math.Round((double) 60 / averageTime, 2);
```

Harp

Most of the harp minigame's functionality is determined by the *HarpGameController* script. When the minigame is loaded, some parameters are set according to the specific enemy that the player is fighting, These parameters, are the root frequency, the intervals of the strings in respect to this root frequency, the number of notes in the melody, and the interval time between the notes of the played reference melody, are extracted from the *EnemyData* and set at *Start()*. Next to that, the physical parameters of the model are set within the inspector using the exposed parameters of the plugin. After startup, three main steps are followed repeatedly in consecutive order. The first step in the loop is to create a reference melody (*void CreateReferenceMelody(int nNotes)*), with takes as input the number of notes in the melody. After this, the next step is to play the reference melody (*IEnumerator PlayReferenceMelody()*), which return-type is an *IEnumerator* making it possible to play the melody with the right interval lengths. Lastly, the harp is made interactable and the program waits for the player to play the melody, which is then evaluated using the method *public void CheckCorrectMelody(double f0)*.

The player's harp consists of three strings, each with a *ExciteString* script, which excites the string whenever the player drags over or clicks on the string. It does so by first calculating the excitation location by subtracting the string's y position from the mouse's y position and adding up the bounds of the *BoxCollider2D*, after which the audio parameter is changed:

```
float excitationPos = mousePos.y -
transform.position.y + 0.5f * collider.bounds.size.y /
collider.bounds.size.y;
audioMixer.SetFloat("StiffString_ePos", excitationPos);
```

Figure 5.18: The violin enemy plays one or more reference notes, which the player needs to imitate, by dragging the bow with the right velocity over the string while clicking the mouse button.

In order for the *HarpGameController* script to play the reference melody, the *ExciteString* script also includes a public method *void ExciteFromScript(float excitationPos)*, of which the excitation location can be freely chosen. For consistency, this excitation location, when playing the reference melody, is always the same, but we could introduce slight variations to make it more difficult at higher levels.

When the location has been set the *ExciteString* script excites the string of the physical model by putting the exposed parameter *"StiffString_excited"* to true. Each string also has an *Animator* component attached which animates the string whenever it is excited.

Violin

The violin minigame works in many ways the same as the harp minigame, with two main differences. With this game, the duration of the played note is evaluated, and the player excites the string by dragging the bow over the string of the violin. The violin consists of four strings that are tuned each with five semitones in between (0, 5, 10, and 15 semitones respectively), but with a variable root note acquired from the *EnemyData* of the violin monster. The main script controlling this scene is the *ViolinGameController*, which sets the frequency of the strings, changes the physical parameters of the *StiffStringPlugin*, and sets the excitation type of the plugin to bowed, by changing the exposed parameter *"StiffString_eType"*. After all the parameters have been set, the game starts by looping through the three main stages. First are the reference note lengths, which consist of one or more notes (depending on the *EnemyData.nNotes*) each with a random duration between 1 and 2.5 seconds. Then these notes are played using the method *IEnumerator PlayReferenceMelody()*, which plays the reference notes from script by changing the bow velocity to 0.2 (slowly increasing to create a smooth transition) for the duration of the reference note, after which the velocity is changed back to zero. When there are more notes in the melody, it waits for a set amount of time (*EnemyData.intervalTime*)

to prevent notes to be connected. Because of the non-linear interaction of the bow and the slow attack and decay of the notes, the reference note lengths might not correspond to the perceived note lengths when the plugin is played from script. To get a more accurate representation of perceived note lengths these note lengths are recorded while the bow is excited from script. This is done by start recording the note length when the intensity is above a certain threshold, and stopping when the intensity goes under the threshold. For this, the Audio is routed back to an *AudioSource* using Native Audio SDK's Audio Routing plugin [13], and then the intensity is calculated by the root mean square of the audio in the *GetVolume()* method:

```
void GetVolume()
{
    gameObject.GetComponent<AudioSource>().GetOutputData(samples, 0);
    // fill array with samples
    int i = 0;
    float sum = 0;
    for (i = 0; i < qSamples; i++)
    {
        sum += samples[i] * samples[i]; // sum squared samples
    }
    rmsValue = Mathf.Sqrt(sum / qSamples); // rms
}
```

After the notes are played from script and the note lengths are recorded the violin strings and bow show up, and the string that needs to be excited becomes intractable. While the player drags the bow over the string both the excitation position and velocity are calculated and used to control the plugin. When the sound intensity is above the threshold, the note length is being recorded in the same way as before, and the string is animated by getting changing the vertical position of the string according to the intensity of the sound:

```
Vector3 vibration = new Vector3(0, (rmsValue / 3.0f) * Mathf.Sin(2 *
    Mathf.PI * 10 * Time.time), 0);
strings[currentString].transform.localPosition = stringPositions[
    currentString] + vibration;
```

When the player releases the mouse button, the played note length is compared to the recorded reference note length. The amount that this can be outside of the reference is dependent on the error margin set in the *enemyData*, which decreases for higher difficulty.

Guitar

In the guitar minigame, the player needs to tune the guitar according to a reference frequency that is played. To do so, the player needs to click and rotate the tuning

[13] https://github.com/Unity-Technologies/NativeAudioPlugins

Figure 5.19: The guitar minigame. The guitar monster plays a reference note, after which the guitar becomes visible and intractable. To make it slightly simpler, an arrow shows if the guitar has to be tuned up or down. The player can tune the guitar by clicking and rotating the peg and excite the string by clicking any key.

peg of the guitar and can excite the string by clicking any key on the keyboard. The main script controlling this scene is the *PegRotationController*. Just like some of the other minigames, this script first changes the fundamental frequency of the guitar string at *Start()*, according to the *EnemyData*, by changing the exposed parameter *"DynamicString_f0"*. To allow smooth frequency changes that are needed for tuning a string, this plugin also has a modulation parameter. The *PegRotationController* has several stages that are looped throughout the minigame. First, it creates a new random reference interval value, which has a minimum and a maximum value determined by the *EnemyData*. Then this reference frequency is played by exciting the string of the plugin by setting the exposed parameter *"DynamicString_excited"* to *true*. In the final stage, the player needs to find the right pitch by rotating the peg while exciting the string. The string is animated in a similar way as in the violin minigame, where the intensity of the sound changes the vertical position of the string. The functionality of the peg is implemented by using the *EventTrigger* component that is attached to the peg *GameObject*. This component has two events: *Drag* and *EndDrag*. The *Drag* event is triggered while the player is dragging the peg. This starts a countdown, in which the player has to find the correct pitch, and updates the modulation parameter according to the rotation of the peg. The pitch is evaluated either when the *EndDrag* event is triggered, signifying that the player has released the peg, or when the countdown has run out. The allowed margin of error that the player may either be above or below the reference pitch, is also dependent on the *EnemyData* and decreases for higher levels.

Sound Puzzle

In the sound puzzle, depicted in Figure 5.20, the player needs to recreate a melody by guiding music notes through the correct sound spheres by drawing lines on the screen. The scene consists of three buttons, one for playing the melody, one for spawning the music notes, and one for deleting the previous line that the player has drawn. As described in section 5.2, the sound for this minigame is synthe-

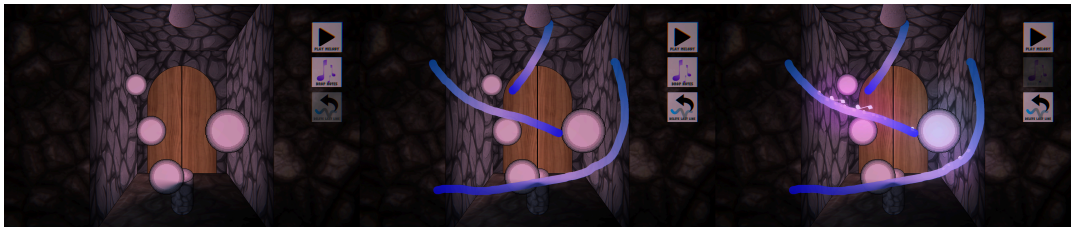


Figure 5.20: The sound puzzle minigame.

sized by an additive synthesizer. This minigame has, other than recreating the c# implementation of the synthesizer to c++, and creating more presets, not received significant changes since the internship. In short, the *PuzzleGameController* script, the main script controlling the scene, initialized the sound spheres according to a preset containing the number of sound spheres, their location, and fundamental frequency. This preset also determines what sound spheres need to be played and in what order. Each sound sphere controls one of the voices of the *AdditiveSynthesizerPlugin* by changing the exposed parameter *"AdditiveSynth_v#"*, with # indicating the voice number. The *PuzzleGameController* script also sets the audio preset of the plugin at *Start()*, by changing the *"AdditiveSynth_preset"* parameter. To provide a rich sound this preset is set to a saw-tooth waveform by default.

Chapter 6

Evaluation

The results of various evaluations are provided in this section. These include performance tests, initial user tests during the CoolHear Workshop, and a user experience study.

6.1 Performance Testing

Relating performance, we had two assumptions that we wanted to evaluate. First, we expected C++ plugins to be preferable to doing all sound synthesis in Unity because of the higher efficiency, but how significant are the benefits of using audio plugins? Secondly, as 2D game environments are generally not difficult to run for modern devices, we envisioned that the application would run on most devices. However, considering we implemented more complicated audio plugins we decided to analyze how well it was running on different devices before doing the actual user tests.

Audio Plugins

To find out if it is worth the additional effort of creating the audio plugins in C++ and controlling them from within Unity, we first compared their performance. For this, we created a blank Unity project and created a new scene with only one *Audio Source* component. Then in the first test, we compared the Audio CPU load between the two implementations of the Stiff String. Figure 6.1 shows Unity's *Profiler* [1], from which can be observed that the CPU load of the C++ implementation was around 1.6%, and of the C# implementation was around 36% while continuously exciting the string with a fundamental frequency of 220 Hz. To find how much this affects performance, we compared the frame times between the two as well. By recording the frames using the *Profiler* we could compare the two using Unity's

[1] https://docs.unity3d.com/Manual/Profiler.html

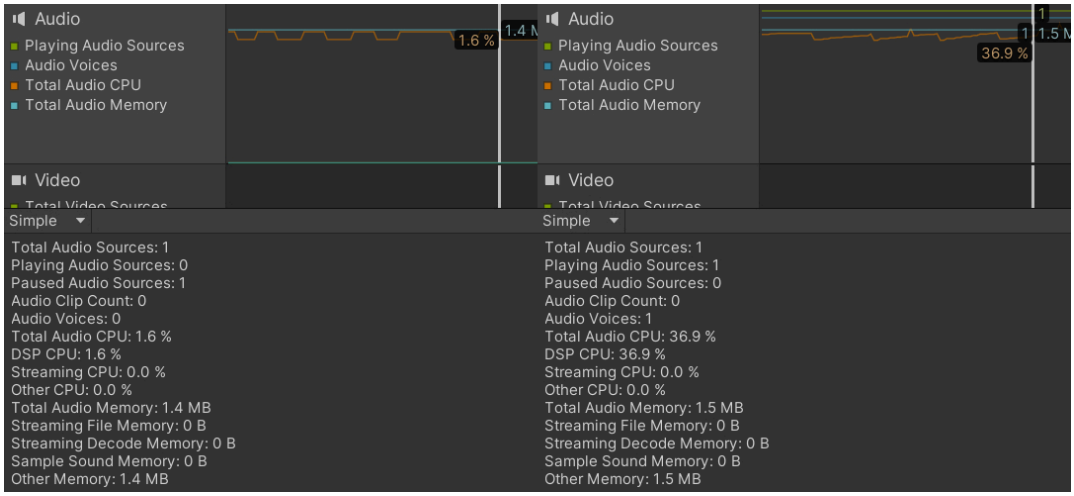


Figure 6.1: Audio comparison between the C++ (left) and the C# (right) implementation of the Stiff String model with a fundamental frequency of 220 Hz.

$f_0 = 220Hz$	C++	C#	Difference	Percentage
Min (ms)	1.32	2.68	9.31	49.25 %
Max (ms)	8.29	15.63	13.03	53.04 %
Median (ms)	1.37	8.12	6.75	16.87 %
Mean (ms)	**1.47**	**8.54**	**7.07**	**17.21 %**
Avg fps (Hz)	**680.27**	**117.10**	**563.17**	**580.95 %**

Table 6.1: Comparison of frame time and frames per second (fps) between the C++ and C# implementation of the Stiff String model. In total the frame time of 500 frames were recorded using the *Profiler* and compared using the *Profiler Analyzer*.

Profiler Analyzer [2], of which the data can be seen from Appendix D. In Table 6.1 the results of the frame times are compared to recordings consisting of 500 frames of each implementation. From this can be seen that the average frame time of the C++ implementation is 1.47 ms, and that of the C# implementation is 8.12 ms. When calculating this to the number of frames per second (fps), the C++ implementation turned out to be 580.95 % faster.

Overall Performance

To get an idea of the overall performance of the application we compared the average fps between four devices, each with different specifications. The frames were captured using the *Profiler* from within the editor while playing two scenes: the

[2]https://docs.unity3d.com/Packages/com.unity.performance.profile-analyzer@0.4/manual/profiler-analyzer-window.html

Device	HP Omen 15	HP zBook	MacBook	Desktop
Year	2021	2015	2015	2018
OS	Windows	Windows	MacOS	Windows
CPU	AMD 5800H	Intel i7	Intel i7	Intel i5
Dedicated GPU	RTX 3060	-	-	GTX 1050
FPS Dungeon	**201.61**	**134.22**	**77.04**	**147.28**
FPS Minigame	**143.27**	**101.01**	**38.83**	**136.61**

Table 6.2: Comparison of the average frames per second (fps) between four different devices when playing the game in two different scenes. In total the frame time of 500 frames were recorded using the *Profiler* and analyzed using the *Profiler Analyzer*.

dungeon, and the percussion minigame. from within the editor on four different devices with different specifications. In Table 6.2 the results are outlined of the calculated average fps. From this can be seen for each instance the framerate is above 30, even for older devices. Next to this, when running the minigames the fps is significantly lower than when running the dungeon scene alone.

6.2 CoolHear Workshop

The CoolHear Workshop [3] was an event organized by the Multisensory Experience Lab in Aalborg University CPH, the Center of Hearing and Balance at Rigshospitalet, and the Royal Danish Academy of Music. The goal of this workshop was to provide a hands-on listening experience for people with hearing impairments, although, many others visited, including Medialogy students of Aalborg University CPH and audiologists. To gain initial feedback, and insights on the usability of the application, we used this opportunity to demo our game throughout the day. The demo was running on two laptops, with both a mouse and wired headphones connected. During the day observations and feedback were collected, of which the full report can be found in Appendix D.

Visitors were advised to play the demo first. Completion of the tutorial took around 5-10 minutes. Most people stopped playing after 5-15 minutes. However, on some occasions participants kept playing the game, sometimes reaching level 3 or 4, playing over 20 minutes. The majority of people seemed to understand all the goals, including exploring, playing minigames, and proceeding to the next levels. On a few occasions, however, it happened that a participant did not immediately know how to play the minigames. Although this seemed only the case either when the participant did not play the tutorial beforehand, or accidentally skipped the tutorial videos. Most people that gave feedback found that it had the right

[3] https://melcph.create.aau.dk/coolhear-workshop/

difficulty, although some said that there was a high learning curve for some of the minigames, such as the guitar and the puzzle game. One participant stated “I found it really intuitive, and I heard myself get better” and another one said, “the interactions felt natural”.

Interestingly, the way participants played the minigames differed sometimes from how we envisioned them to be played. One instance of this is the percussion game, where some players did not tap continuously on the tempo, but in rounds. Some people had the problem of finding the right BPM and played either in double tempo or half tempo. This indicated that it was not clear that the pulse of the circle represented the right BPM.

The harp and violin minigames seemed pretty easy to most players. Some people seemed to be confused about the harp playing the reference strings at the beginning of the minigame and thought it was part of the melody that they needed to recreate. The violin was also well received and was perceived as fun. One person stated that he liked the sound overall, but specifically of this game.

The guitar was the hardest of the minigame for most participants. Apart from the interaction that was slightly more complicated for this minigame, it seemed difficult for most people to remember the reference pitch. One person suggested that it would be nice if the reference pitch could be heard again. One participant with musical experience found a way to perform better by humming the pitch herself to remember it.

Overall the game performed quite well with no major technical issues. One bug that appeared, however, was that the guitar disappeared when the player ran out of time (timer zero).

Apart from insights of NH listeners playing the game discussed above, also one hearing aid (HA) user and one CI user played the game, both around 11 years old. In both cases, communication went in English via the parent. The HA user found the guitar game very difficult and did not understand what to do. Other minigames were a lot easier for him to carry out. The CI user was able to play all minigames without any difficulty, and specifically said during playing the violin that he found it too easy.

6.3 User Experience

We evaluated the user experience both quantitatively and qualitatively. In order to get a general idea of the user experience, and identifying possible flaws, we evaluated the game on NH listeners using an online survey. Secondly, in order to get more in-depth insights on the thoughts and feelings of some CI users we conducted interviews.

6.3.1 Online Survey

To get insights into the user experience of our application, we created a questionnaire for NH people that needed to be filled out after they had played the game. The questionnaire consisted of installation and play instructions, and three main sections with questions, including background questions, the user experience questionnaire (UEQ), and additional questions to make it easier for us to interpret the data. The UEQ [4] is designed to get a good understanding of the user experience and usability by measuring both classical usability aspects (efficiency, perspicuity, dependability) and user experience aspects (originality, stimulation). This is done using 26 questions that each have a seven-point scale with two opposing words on either side (semantic differentiation). The final sections of the questionnaire consisted of both 5-point Likert-scale and open questions to get more in-depth insights into the users' thoughts on sound quality, interaction, and music training.

Results

We received 23 responses, of which two persons were not able to install the game and one person was filtered out because of too many inconsistencies in the UEQ. Participants' ages ranged from 15 to 32 years old ($M = 24,75$, $SD = 3,40$) and played the game on average for around 30 minutes, using mostly wired headphones (few exceptions with people wearing wireless earphones). Most respondents were experienced in playing video games, with most of them playing either daily or multiple times a week.

The results of the UEQ were analyzed using the UEQ Data-Analysis Tool [5]. For almost all questions, the mean was rated above 1, except the *usual-leading edge* ($M = 0,3$) and *fast-slow* ($M = -0,2$) questions. In the UEQ the data is grouped into six scales, of which the results are given in Figure 6.2. Here, Attractiveness scored the highest ($M = 1,92$, $SD = 0,46$), followed by Perspicuity ($M = 1,56$, $SD = 0,55$), Stimulation ($M = 1,49$, $SD = 0,58$), and Dependability ($M = 1,48$, $SD = 0,31$). The lowest scores were obtained for Efficiency ($M = 1,00$, $SD = 0,61$) and Novelty ($M = 1,20$, $SD = 0,41$). These results can be grouped further into three categories: Attractiveness ($M = 1,92$), Pragmatic quality ($M = 1,35$), and Hedonic quality ($M = 1,34$). Here, the pragmatic quality consists of the Perspicuity, Efficiency and Dependability, and the Hedonic quality of Stimulation and Novelty. In order to test if the user experience is 'sufficient', we used UEQ's benchmark comparison tool [51], of which the results are shown in Figure 6.3. Here, the results of each of the scales are compared to existing values from a benchmark data set, consisting data from 21175 persons from 468 studies. Here, Attractiveness scored *Excellent* (in the range of the 10% best results), Stimulation and Novelty scored *Good* (10% better,

[4] https://www.ueq-online.org/
[5] https://www.ueq-online.org/

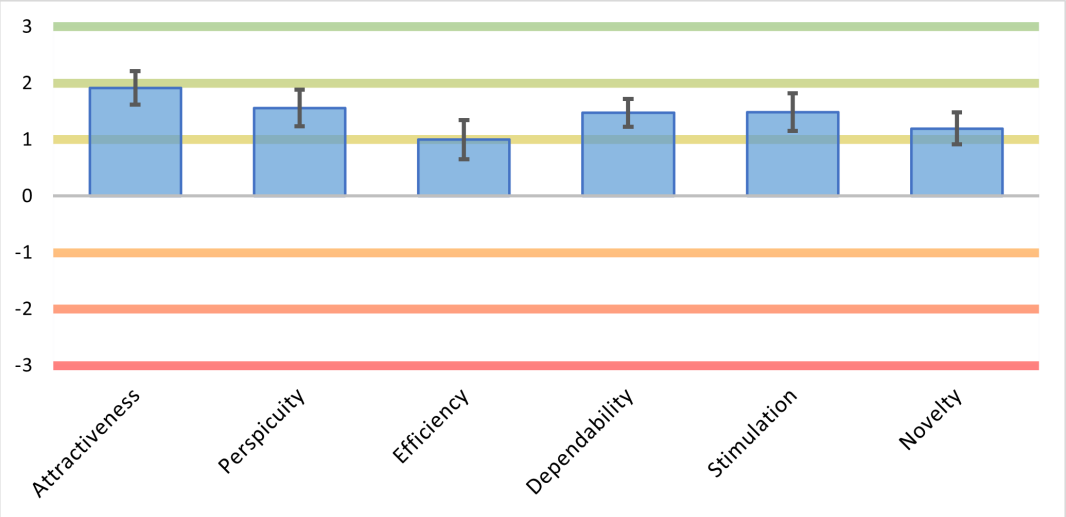


Figure 6.2: Results of the UEQ. Mean and standard deviation for each scale.

75% worse), Perspicuity and Dependability scored *Above Average* (25% better, 50% worse), and Efficiency scored *Below Average* (50% better, 25% worse).

In addition to the UEQ, we added one open question for people to give suggestions to improve the overall game experience. Here, some people indicated that they had missed the tutorial videos (accidentally skipped it by clicking next), or that a voice-over explaining the interaction could make it more clear what to do. One participant suggested adding more instrument monsters, for example, wind instruments. Another found that, although it was fun at first, it lacked variety, the game phase was very slow, and it took much time to explore the full dungeon. Three people indicated that they had issues playing the harp, either because it was too difficult, or because they felt it was evaluated randomly. One participant suggested improving the latency in the drum game, and one participant had to quit the game because of a frozen screen after level 4.

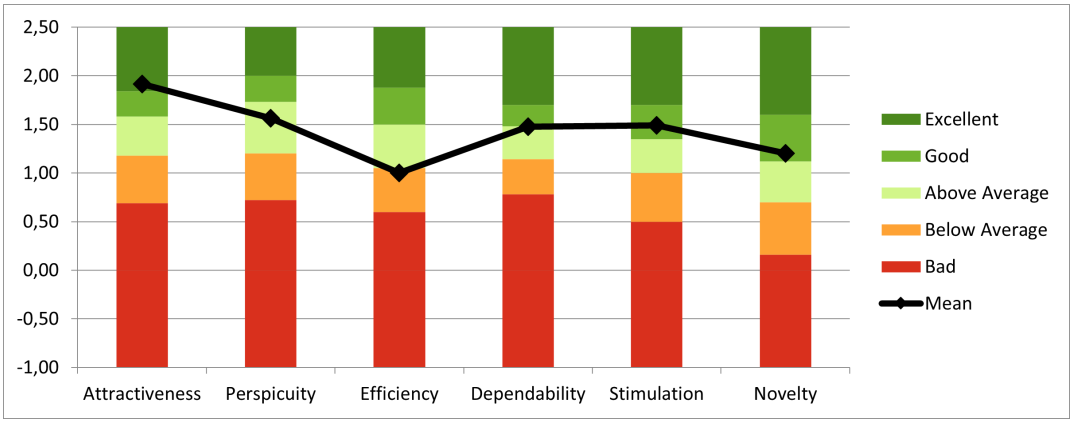


Figure 6.3: Results of the UEQ benchmark.

The final section of the questionnaire consisted of three five-point Likert-scale questions in combination with open questions for more explanation and feedback. The mean and standard deviations are given in Figure 6.4. Of these, the question "How realistic do you find the sounds of the instruments in each minigame?", got rated the highest ($M = 4,25$, $SD = 0,79$). Most people seem to find it accurately represents real instruments. One participant stated *"They sound like they do in real life"*, and another one said, *"I'm used to playing instruments and they sound pretty alive"*. However, one participant mentioned that the guitar sounded artificial in the higher registers, and some others stated that the harp and violin were slightly synthetic or missing something. Additionally, one participant perceived cut-outs in the sound.

The question "To which degree do you think the minigames increase your understanding and perception of musical aspects like pitch, timbre, and rhythm?" was rated slightly lower ($M = 3,95$, $SD = 1,00$). One participant stated: *"They are very simple and effective in representing the musical aspects"*. Two participants said that it would be for people that are not musically inclined, although another participant with more musical experience said: *"The pitch game is a nice training though, and the interval game can be useful training if expanded"*. Also, another participant said that it could increase the perception of these musical aspects even for people that are not hearing impaired. One participant said that the game gives a good foundation of the musical aspects but lacks proper explanation.

The question "To which degree do you think interacting with the simulated instruments inside the minigames will improve your perception and understanding of real instruments?", was rated the lowest ($M = 3,3$, $SD = 1,22$). Here, respondents who gave a lower score indicated that it was because they are experienced in playing musical instruments. Others mentioned that even though it can help, playing physical instruments is better and that the 'feeling' of playing an instrument is critical. One participant stated: *"I definitely think it will improve perception.*

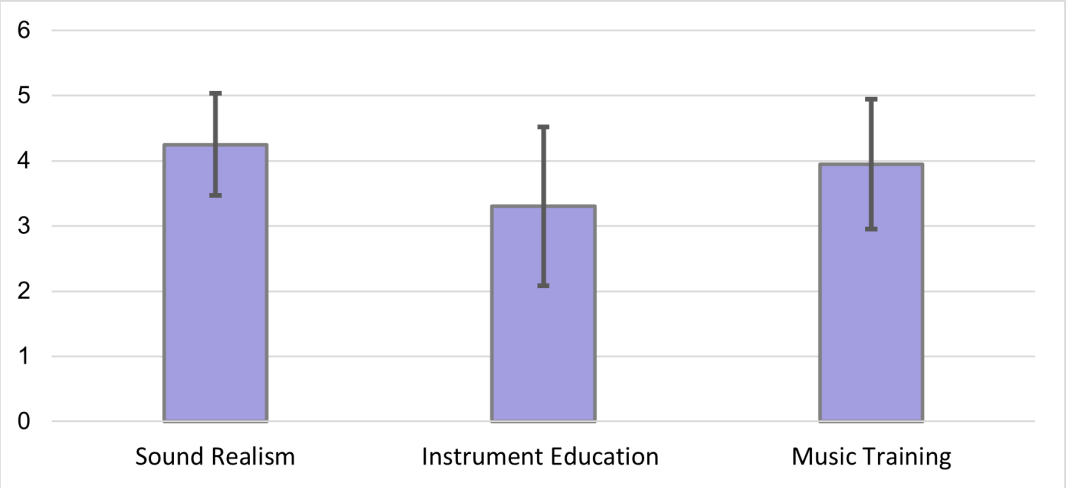


Figure 6.4: Results of the sound and training related questions

Especially for bowed instruments whose interaction gives a very distinct/sustained sound. By playing with this one might attune to the behaviour and will maybe be able to recognize it in a musical context". However, another participant mentioned that it could have helped in understanding the instruments if free-play was possible, as opposed to the single-click interactions.

Two final open questions were about the difficulty and optional general feedback. Most people said that all minigames got easier after getting used to the game mechanics and interactions. Two indicated that all got easier except for the violin, and another said there was a slight learning curve in dealing with the bow of the violin. One participant said: *"It also gets easier to remember e.g. the pitch in the guitar game, which makes it easier to recreate"*. One participant mentioned that the minigames got harder the further you proceeded into the game. As final feedback, some participants emphasized that they liked the artwork and animations. One participant said that the puzzle game would be less relevant as it does not represent a real instrument. Some people also mentioned possible improvements to make it less cumbersome to walk around, like being able to zoom in on the minimap or teleport.

6.3.2 Interviews

To complement the user experience questionnaire presented above, we also conducted semi-structured interviews with CI users from our target group to get more in-depth insights into their experience, feelings and thoughts. For this we focused on acquiring knowledge about their opinions on the instrument interactions and sounds. We conducted two interviews via Teams with a 10 and 11 year old CI user. The interviews were around 50 minutes and 30 minutes respectively. The

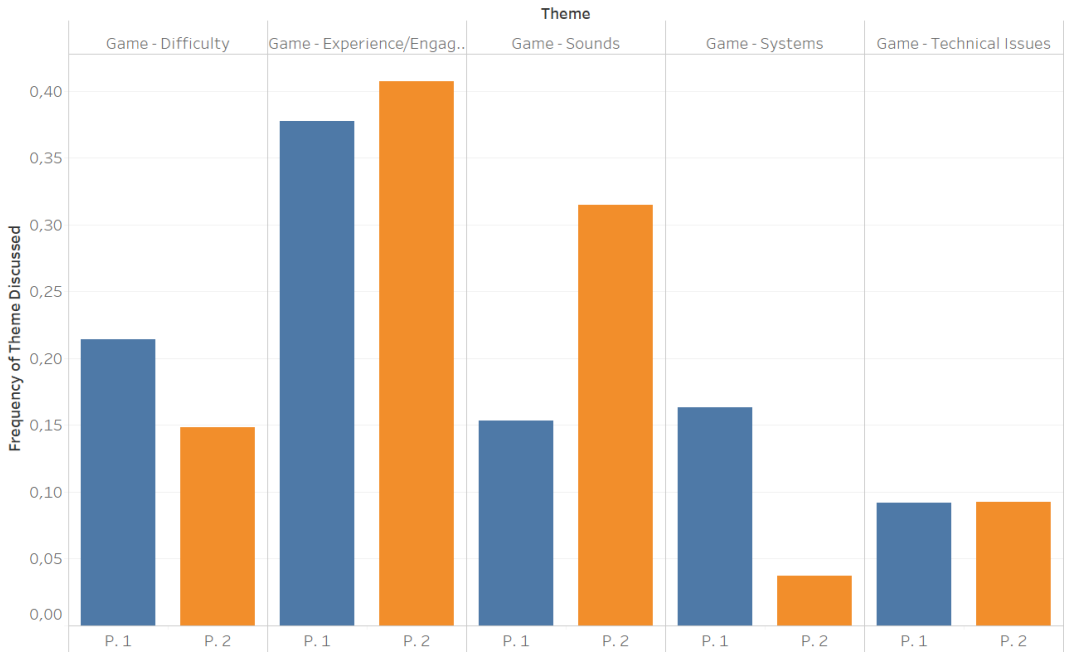


Figure 6.5: Interview categorisation of common themes of feedback and their frequency for each participant.

second participant had one of his mother help him answer some questions as she was also present whenever he played the game. Interviews were conducted based on a semi-structured interview outline which can be found in Appendix D.

The interviews were transcribed (Appendix D) and then analyzed and categorized to be able to summarise different themes of discussion (as color codes). Figure 6.5 gives a brief overview of the themes and parts of the game that were discussed with each participant. The feedback from the interviews was categorized into themes, including the game's difficulty, game experience or engagement, game sounds, game systems, or technical issues. This system provides an overview of the frequency of each theme being addressed by the participants. From this it can be observed that both participants frequently addressed their engagement with the game, but participant one addressed many of the game systems during his interview, and participant two addressed the sounds more often. Each participant's feedback is elaborated more in-depth in the sections below.

Participant 1

Participant one was a 10 year old boy with bilateral CIs. His obtained his right CI one year ago and just received his left one, of which he was still recovering from. Due to this he was yet unable to hear with his left CI and was relying solely on his right CI to listen while playing the game. He was generally not actively

engaging with music, other than when he heard it in television commercials or as background music for visual content. Music was generally not bothering him but he found specific melodies and especially dynamically quiet music to sound annoying. He specifically mentioned that violin music was unpleasant for him to listen to. Participant one liked to play video games on computer and PlayStation such as fighting, exploration, and social -games.

Participant one played Uncharted Chants over a period of two weeks and he said that he played around two hours per day with only a couple of days where he was not playing. To him, Uncharted Chants was fun and engaging. He said that he enjoyed playing the game in general and that he was motivated to play again each day. When he was asked if he looked forward to playing the game or if it felt more like a chore to him, he replied: *"Nahh it was actually for pure enjoyment"*. Furthermore, he addressed many systems in the game, such as the item upgrades, currencies, and shop-rooms, and seemed to have understood the systems and shared his tactics on how to optimize his performance in the game using various items. The interactions of the minigames made sense and were easy for him to grasp. He could beat every minigame in all levels up until the final boss level. However, he tried to avoid the violin instrument monster because he found the sound of it unpleasant and he reported that he found the harp more challenging than the guitar and the percussion instruments. He found the puzzle minigame a bit difficult to understand at first but he then found out how it worked and was able to consistently succeed in it. When he played, he reported to have reached the final boss level several times but he, for the time being, failed to beat the boss. He said that the harp-instrument monster in the boss level was currently too difficult for him mostly due to the speed at which the notes of the reference melody were played. He mentioned that for each playthrough he prioritized playing all instrument minigames in each level with the purpose of training his perception so that he could beat the boss. He elaborated on his experience stating: *"The first many times I did speed-runs, I avoided all monsters [...]"* then *"But then I found out that you have to practice quite a lot to reach the boss so now I [...] go around and beat all the monsters in each level"*.

Participant one found the overall game really fun and after the two weeks he had the game, he was still motivated to play more. He found the harp strings to have a pleasant sound and he liked the percussion beats, stating that: *"[...] the harp gives nice sounds which are not annoying, it is such a relaxing melody one could say. It is really only the violin that is a bit annoying, otherwise, everything else is fine"*. He even recorded some of the percussion-instruments beats because he liked to just listen to them. For him, the guitar sounds were pleasant as well and were working well for him to indicate the tuning of the guitar in the guitar training exercise. However, he found the synthetic saw-waves of the puzzle minigame to be more clear and easy to perceive than the harps string sounds. Participant one reported

some technical issues that he noticed while he was playing. First, the game did not save his progress when he exited during a playthrough. Then, he found that the harp monster had a weird spinning behavior sometimes which made it difficult to dodge. Finally, he reported that he noticed some parts of the game would be in English even though he selected danish. He added that he found himself improving his performance in the training exercises towards the end of each of his playthroughs and that this was why he was confident that he could beat the boss. Participant one previously used another perception training app that focused on training speech and everyday sounds. He said the app had helpful exercises but that he would rather play Uncharted Chants because he found it more fun. He also mentioned that he would recommend Uncharted Chants to other CI users that had recently received their implants as he thought the game was good for training perception.

Participant 2

Participant two was a 11 years old boy who also had bilateral CIs. He acquired his CIs when he was 9 months old. He was not interested in listening to music on his own but when he encountered music as part of other media, such as games, television etc., he generally likes it. He liked playing video games, specifically sports games like soccer or basketball, on either his PlayStation or phone, and with his friends.

Participant two played Uncharted Chants two times for 30 minutes each session and was generally not motivated to play it on his own. He found that he understood how the minigames worked from the tutorials but found the guitar minigame difficult to succeed in. He found the violin difficult as well but did like the sound of it as he answered: *"It sounded good, it sounded like a real violin."* when asked about the sound of the violin. He found that with the violin it felt like he had to let go of the bow a little bit early for it to evaluate as successfully, which made it more difficult. Generally, participant two liked the sounds in the game and reported that he liked especially the violin and harp string sounds. He said that he found the sounds realistic and that he liked the string sounds more than the puzzle's saw-wave tones because they sounded like actual strings. None of the sounds were unpleasant or annoying to participant two. Participant two also liked the percussion beats and reported that he enjoyed listening to those. Participant two found that in the beginning he was confused with the harp strings having different timbres when interacting differently. He listened for the exact same sound as the reference and was confused when he played a string that did not have the same sound (even though it played the same note).

In the beginning, he did not understand the concept of the game and why he had to play from the beginning again after he died. He felt that it was a bit demotivating to lose his coins. But after he understood that he kept the musical

essence and could upgrade the character he understood why he had to play from the first level again. He also found that if he closed the game in the middle of a level he would lose the progress that he had and would have to start over, this was also demotivating for him.

Participant two had tried another music training app in VR and he thought that he would rather play a music training game on the computer. He reported also that he thinks the game works well on computer and that he thinks it works better than it would on PlayStation or a phone as he stated: *"I think computer works best. It would not be good on a PlayStation because... I can not explain it. On the phone, it would probably lag too much"*.

Chapter 7

Discussion

During the evaluation we have gained various insights regarding the current performance and usability of Uncharted Chants, and its user experience regarding both NH listeners as CI users. These are discussed below.

Firstly, even though Uncharted Chants has not been optimized, it runs sufficiently on all tested devices. For all devices in the performance test, the game was running effortlessly with sufficient frame rates. Probably contributing to this performance is the way we implemented the physical models as the results show that C++ has a significant advantage over implementing them using Unity's C# scripts. Some respondents of the survey were not able to install the game (2 out of 23), which was likely due to Windows' antivirus software labeling it potentially harmful when downloading. Besides that, other than a few bugs reported, no major problems were found when playing the game. This indicates the game's potential for using it for a wide audience and various devices.

Results from the UEQ indicate that Uncharted Chants has a high user experience score, with positive ratings for both the *Attractiveness*, *Pragmatic quality* and *Hedonic quality*. From the benchmark, we can interpret the data better, with the *Attractiveness* scoring 'excellent' and all other scales scoring 'higher than average', except for the *Efficiency* scale scoring 'below average'. This scale scoring worse is due to the game being perceived as slightly *slow*. Although it is difficult to say why this is from the UEQ alone, the attached open questions could give a better indication. Here, elements like the time it takes to explore the dungeon and slow walking speed were mentioned, which indicate aspects to further improve the rating for this scale. Next to this, another factor possibly influencing the user experience, is that the interactions were not always as clear. Especially the guitar and violin minigames seemed to be significantly more difficult than the percussion and harp minigames. This can be both due to the interactions being more complex as well as the listening exercise being more difficult. This could be due to the tutorial videos being unclear or that users accidentally skipped them as some

participants reported.

The results from the interviews indicate that the game can be perceived as highly engaging for CI users as well. This became evident as one out of the two CI users was motivated to play the game daily. However, the level of engagement seems to be strongly dependent on the user's personal preferences towards game genres. For example, when the user is more interested in other types of games, like social or sports, the game can be less engaging. Both CI users reported to have used other tools for perceptual training, including both speech and music, and mentioned that they would prefer training their perception using Uncharted Chants. Although we do not know the specifics of these tools, this could be due to the game-based learning approach that we took.

The implemented difficulty system seems to be approachable to most participants. The CI users reported having reached level two or three the first time they played the game. This was also observed during the CoolHear workshop where the CI user did not have any issues with the starting difficulty. From this, it seems like the low entry-level that we implemented works for a wide audience. However, the way we designed the difficulty scaling, where the margin of error decreases for higher levels, could potentially mean that CI users misinterpret their performance. With the guitar for example, even if CI users tuned the guitar to the wrong pitch, it still could have been evaluated as successfull. We tried to accommodate for this by giving the user different feedback (e.g. 'alright', instead of 'perfect'), but some participants still reported that they felt like their input was evaluated randomly. This issue could be addressed by implementing more detailed visual feedback for all minigames, but most importantly the guitar minigame.

Regarding the sound of the implemented physical models, most respondents of the survey considered it highly realistic. A few improvement points do exist, however, with some people indicating it was lacking 'something'. This could be due to the fact that we only modeled part of the resonator (e.g. the string), and not full instruments (adding the bridge, body, etc.). The CI users enjoyed most of the used sounds, although there was a slight difference found between them. The prelingual CI user indicated that all instrument sounds were realistic and sounded good, preferring them over the synthetic sound of the puzzle game. The post-lingual CI user, however, found the violin highly unpleasant and found the sound of the puzzle clearer. This might be because he only had his CI for one year and therefore was still training and adapting to electrical hearing, as opposed to the participant that used CIs almost his whole life who found the violin sound satisfying and realistic. Some participants reported being confused about the different sounds they were able to produce while playing the harp strings, including one CI user. This indicated, that even though it was not clear that the sound was dependent on the interaction, some CI users may be sensitive to small perceptual changes in timbre.

Although we did not specifically evaluate the game in terms of training out-

come, we can discuss its potential. From the survey, most people see the potential of playing instruments for music training and found that the minigames conveyed the foundations of the different musical aspects, like pitch, timbre, and rhythm. Potentially, it could even be used as a music training tool for NH listeners when adjusting the difficulty accordingly. Additionally, one CI user indicated that he would recommend the game to others that recently obtained their CIs, as he found it helpful for rehabilitation. Some participants reported that one way to complete a level very fast was to avoid walking into instrument monsters in the dungeon. Exploiting this shortcut might result in worse performance in higher levels, where the training exercises are more difficult and sufficient training exposure is required. However, for higher performing CI users this ultimately results in limited exposure to perception training and should be addressed in future iterations of the game.

Limitations

Certain limitations affect the level at which generalization of the results is possible. The biggest of which is the sample size of the CI user group. As a result of difficulties regarding the recruitment of CI users, we were only able to interview two CI users. Although this is sufficient for getting preliminary insights, it makes it impossible to generalize the findings. For the UEQ the sample size was sufficient in the way that it gave stable results (small standard deviation). However, it could lack variety as most participants seemed to be highly engaged in games. Secondly, the data is susceptible to bias. Although the survey was shared on forums, some participants were acquaintances of ours. Additionally, even though we did not know the interviewees, they were possibly encouraged by their parents to play the game, skewing the results.

Chapter 8

Conclusion

During this project, we have designed and developed a music training game for CI users, with the primary goal to make perception training fun and engaging. While doing so, we tried to investigate the potential of using simulated instrument sounds and interactions in training applications.

This game, named Uncharted Chants, shows that utilizing gamification principles in music training practices can result in high engagement for some users. However, the user's preferences towards the game genre play a major role in the exact level of engagement. The current state of the game is fully functional and has shown to score high on user experience. Nevertheless, several shortcomings exist regarding some game elements, like game speed, interactions, and bugs. Additionally, the game is built using systems that are easy to expand upon. This could be done by adding, for example, more instruments, more levels, and different environments.

Using physical modeling for creating instrument sounds yields promising results in terms of perceived sound realism. Next to this, the different timbres, created by different interactions can be perceived by CI users. This shows that some users can notice small timbre differences, although it can also cause confusion relating to the training exercise they need to complete. To familiarize the user with the different interactions and timbre it would be ideal to implement a way for users to play the simulated instruments freely, outside the training exercises. Furthermore, it is yet to be determined if using physical modeling could potentially educate CI users on the relationship between interaction and sound output, and would be an interesting topic for further research and exploration.

Other possible future works include personalized difficulty scaling. Our current system seems to be accessible for CI users, but mainly because of the low-entry level, which can be too easy for higher-performing CI uses. A simple approach to improve the current difficulty system could be by incorporating checkpoints that let the player start every new playthrough from a previously reached checkpoint.

However, the ideal way to improve the difficulty system is to develop a more versatile system that could dynamically tweak the difficulty depending on how well the player performs in the training minigames. Using this approach, the game could also be targeted for NH listeners.

Finally, future work could show whether playing Uncharted Chants ultimately increases music perception and music enjoyment, and to what degree. For this, a long-term study could be conducted to see if music exposure in this form benefits CI users.

Bibliography

[1] Peter W Alberti. "The anatomy and physiology of the ear and hearing". In: *Occupational exposure to noise: Evaluation, prevention, and control* (2001), pp. 53–62.

[2] Albena Antonova and Krassen Stefanov. "Applied cognitive task analysis in the context of serious games development". In: *Third International Conference on Software, Services and Semantic Technologies S3T 2011*. Springer. 2011, pp. 175–182.

[3] Adriano Baratè, Mattia G Bergomi, and Luca A Ludovico. "Development of serious games for music education". In: *Journal of e-Learning and Knowledge Society* 9.2 (2013).

[4] Valentin Bégel, Antoine Seilles, and Simone Dalla Bella. "Rhythm Workers: a music-based serious game for training rhythm skills". In: *Music & Science* 1 (2018), p. 2059204318794369.

[5] Stefan Bilbao. *Numerical sound synthesis: finite difference schemes and simulation in musical acoustics*. John Wiley & Sons, 2009.

[6] Stefan Bilbao et al. "The NESS Project: Physical Modeling, Algorithms and Sound Synthesis". In: (2019).

[7] S. J. Brockmeier et al. "The music perception test: A novel battery for testing music perception of cochlear implant users". In: *Cochlear Implants International* (2011). ISSN: 14670100. DOI: 10.1179/146701010X12677899497236.

[8] Sandra Cano et al. "Model for Design of Serious Game for Rehabilitation in Children with Cochlear Implant". In: *International Workshop on ICTs for Improving Patients Rehabilitation Research Techniques*. Springer. 2015, pp. 94–105.

[9] Graeme M. Clark. *Personal reflections on the multichannel cochlear implant and a view of the future*. 2008. DOI: 10.1682/JRRD.2007.05.0064.

[10] Perry R Cook. "Physically informed sonic modeling (phism): Synthesis of percussive sounds". In: *Computer Music Journal* 21.3 (1997), pp. 38–49.

[11] Perry R Cook. *Real sound synthesis for interactive applications*. CRC Press, 2002.

[12] S. Erhel and E. Jamet. "Digital game-based learning: Impact of instructions and feedback on motivation and learning effectiveness". In: *Computers and Education* (2013). ISSN: 03601315. DOI: 10.1016/j.compedu.2013.02.019.

[13] Deniz Eseryel et al. "An investigation of the interrelationships between motivation, engagement, and complex problem solving in game-based learning". In: *Journal of Educational Technology & Society* 17.1 (2014), pp. 42–53.

[14] Qian-Jie Fu and John J Galvin III. "Perceptual learning and auditory training in cochlear implant recipients". In: *Trends in Amplification* 11.3 (2007), pp. 193–205.

[15] Nobuya Fujiki et al. "Correlation between rCBF and speech perception in cochlear implant users". In: *Auris Nasus Larynx* (1999). ISSN: 03858146. DOI: 10.1016/S0385-8146(99)00009-7.

[16] T Fullerton, C Swain, and S Hoffman. *Game design workshop: A playcentric approach to creating innovative games*. Amsterdam: Elsevier Morgan Kaufmann, 2008.

[17] Tracy Fullerton. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. 2008. ISBN: 9780240809748. arXiv: 9809069v1 [arXiv:gr-qc].

[18] Kate Gfeller. "Music-based training for pediatric CI recipients: A systematic analysis of published studies". In: *European Annals of Otorhinolaryngology, Head and Neck Diseases* 133 (2016), S50–S56.

[19] Kate Gfeller, Virginia Driscoll, and Adam Schwalje. "Adult cochlear implant recipients' perspectives on experiences with music in everyday life: A multifaceted and dynamic phenomenon". In: *Frontiers in neuroscience* (2019), p. 1229.

[20] Kate Gfeller et al. "A preliminary report of music-based training for adult cochlear implant users: rationales and development". In: *Cochlear implants international* 16.sup3 (2015), S22–S31.

[21] Kate Gfeller et al. "Effects of training on timbre recognition and appraisal by postlingually deafened cochlear implant recipients". In: *Journal of the American Academy of Audiology* 13.03 (2002), pp. 132–145.

[22] Richard L Goode et al. "New knowledge about the function of the human middle ear: development of an improved analog model." In: *The American journal of otology* 15.2 (1994), pp. 145–154.

[23] Juho Hamari. "Gamification". In: *The Blackwell Encyclopedia of Sociology*. John Wiley & Sons, Ltd, 2019, pp. 1–3. ISBN: 9781405165518. DOI: https://doi.org/10.1002/9781405165518.wbeos1321. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781405165518.wbeos1321. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781405165518.wbeos1321.

[24] Natalie A Hardie and Robert K Shepherd. "Sensorineural hearing loss during development: Morphological and physiological response of the cochlea and auditory brainstem". In: *Hearing Research* 128.1-2 (1999), 147–165. DOI: 10.1016/s0378-5955(98)00209-3.

[25] Sibylle C Herholz and Robert J Zatorre. "Musical training as a framework for brain plasticity: behavior, function, and structure". In: *Neuron* 76.3 (2012), pp. 486–502.

[26] Hans Herzog et al. "Cortical activation in profoundly deaf patients during cochlear implant stimulation demonstrated by h215o pet". In: *Journal of Computer Assisted Tomography* (1991). ISSN: 15323145. DOI: 10.1097/00004728-199105000-00005.

[27] Henkjan Honing. "Structure and Interpretation of Rhythm in Music". In: *The Psychology of Music*. 2013. ISBN: 9780123814609. DOI: 10.1016/B978-0-12-381460-9.00009-2.

[28] Nicole T Jiam, Meredith T Caldwell, and Charles J Limb. "What does music sound like for a cochlear implant user?" In: *Otology & neurotology* 38.8 (2017), e240–e247.

[29] Patrik N Juslin and Daniel Västfjäll. "Emotional responses to music: The need to consider underlying mechanisms". In: *Behavioral and brain sciences* 31.5 (2008), pp. 559–575.

[30] Robert Kang et al. "Development and validation of the University of Washington clinical assessment of music perception test". In: *Ear and Hearing* (2009). ISSN: 01960202. DOI: 10.1097/AUD.0b013e3181a61bc0.

[31] Nina Kraus et al. "Experience-induced Malleability in Neural Encoding of Pitch, Timbre, and Timing: Implications for Language and Music". In: *Annals of the New York Academy of Sciences* 1169.1 (2009), pp. 543–557.

[32] E de Larrea-Mancera et al. "Training with an auditory perceptual learning game transfers to speech in competition". In: *Journal of Cognitive Enhancement* 6.1 (2022), pp. 47–66.

[33] Luis Lassaletta et al. "Changes in listening habits and quality of musical sound after cochlear implantation". In: *Otolaryngology—Head and Neck Surgery* 138.3 (2008), pp. 363–367.

[34] Petri Laukka. "Uses of music and psychological well-being among the elderly". In: *Journal of happiness studies* 8.2 (2007), pp. 215–241.

[35] Huawei Li and Renjie Chai. *Hearing loss: mechanisms, prevention and cure*. Vol. 1130. Springer, 2019.

[36] Charles J Limb. "Cochlear implant-mediated perception of music". In: *Current Opinion in Otolaryngology & Head and Neck Surgery* 14.5 (2006), pp. 337–340.

[37] Charles J. Limb and Alexis T. Roy. *Technological, biological, and acoustical constraints to music perception in cochlear implant users*. 2013. URL: https://www.sciencedirect.com/science/article/abs/pii/S0378595513001020?via%3Dihub.

[38] Charles J. Limb et al. "Auditory cortical activity during cochlear implant-mediated perception of spoken language, melody, and rhythm". In: *JARO - Journal of the Association for Research in Otolaryngology* (2010). ISSN: 15253961. DOI: 10.1007/s10162-009-0184-9.

[39] Valerie Looi, Kate Gfeller, and Virginia D Driscoll. "Music appreciation and training for cochlear implant recipients: a review". In: *Seminars in hearing*. Vol. 33. 04. Thieme Medical Publishers. 2012, pp. 307–334.

[40] Valerie Looi and Jennifer She. "Music perception of cochlear implant users: a questionnaire, and its implications for a music training program". In: *International journal of audiology* 49.2 (2010), pp. 116–128.

[41] *Medicine CI CI and Music*. https://medicine.uiowa.edu/iowaprotocols/music-and-hearing-loss/cochlear-implant-ci-and-music/cochlear-implant-ci-pages-audiologists/cochlear. Accessed: 2022-05-18.

[42] Philip H. Mirvis and Michael Csikszentmihalyi. "Flow: The Psychology of Optimal Experience". In: *The Academy of Management Review* (1991). ISSN: 03637425. DOI: 10.2307/258925.

[43] Chisato Mitani et al. "Music recognition, music listening, and word recognition by deaf children with cochlear implants". In: *Ear and Hearing* (2007). ISSN: 01960202. DOI: 10.1097/AUD.0b013e318031547a.

[44] Yasushi Naito et al. "Increased cortical activation during hearing of speech in cochlear implant users". In: *Hearing Research* (2000). ISSN: 03785955. DOI: 10.1016/S0378-5955(00)00035-6.

[45] Heather L O'Brien and Elaine G Toms. "What is user engagement? A conceptual framework for defining user engagement with technology". In: *Journal of the American society for Information Science and Technology* 59.6 (2008), pp. 938–955.

[46] Tae Hong Park. *Introduction to digital signal processing: Computer musically speaking*. World Scientific, 2009.

[47] Aniruddh D Patel. "Why would musical training benefit the neural encoding of speech? The OPERA hypothesis". In: *Frontiers in psychology* 2 (2011), p. 142.

[48] Shu Chen Peng et al. “Perception and production of Mandarin tones in prelingually deaf children with cochlear Implants”. In: *Ear and Hearing* (2004). ISSN: 01960202. DOI: 10.1097/01.AUD.0000130797.73809.40.

[49] Curtis Roads. *The computer music tutorial*. MIT press, 1996.

[50] Thomas Schäfer et al. “The psychological functions of music listening”. In: *Frontiers in psychology* 4 (2013), p. 511.

[51] Martin Schrepp. “User experience questionnaire handbook”. In: *All you need to know to apply the UEQ successfully in your project* (2015).

[52] Robert K. Shepherd and Natalie A. Hardie. “Deafness-induced changes in the auditory pathway: Implications for cochlear implants”. In: *Audiology and Neuro-Otology* 6.6 (2001), 305–318. DOI: 10.1159/000046843.

[53] Julius O Smith. “Physical Audio Signal Processing for virtual musical instruments and digital audio effects”. In: *online book at http://ccrma. stanford. edu/jos/pasp/, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University* (2010).

[54] Wooi Teoh Su, David B. Pisoni, and Richard T. Miyamoto. “Cochlear implantation in adults with prelingual deafness. Part I. Clinical results”. In: *Laryngoscope* (2004). ISSN: 0023852X. DOI: 10.1097/00005537-200409000-00006.

[55] Tero Tolonen, Vesa Välimäki, and Matti Karjalainen. “Evaluation of modern sound synthesis methods”. In: (1998).

[56] Susan B. Waltzman, J. Thomas Roland, and Noel L. Cohen. “Delayed implantation in congenitally deaf children and adults”. In: *Otology and Neurotology* (2002). ISSN: 15317129. DOI: 10.1097/00129492-200205000-00018.

[57] Silvin Willemsen et al. “Dynamic grids for finite-difference schemes in musical instrument simulations”. In: *24th International Conference on Digital Audio Effects*. 2021, pp. 144–151.

[58] Silvin Willemsen et al. “Real-time control of large-scale modular physical models using the sensel morph”. In: *16th Sound and music computing conference*. Sound and Music Computing Network. 2019, pp. 151–158.

[59] Blake S Wilson and Michael F Dorman. “Cochlear implants: current designs and future possibilities”. In: *J Rehabil Res Dev* 45.5 (2008), pp. 695–730.

[60] Donald Wong et al. “PET imaging of cochlear-implant and normal-hearing subjects listening to speech and nonspeech”. In: *Hearing Research* (1999). ISSN: 03785955. DOI: 10.1016/S0378-5955(99)00028-3.

[61] Fan-Gang Zeng et al. “Cochlear implants: system design, integration, and evaluation”. In: *IEEE reviews in biomedical engineering* 1 (2008), pp. 115–142.

[62] Mohammad Zohaib. *Dynamic difficulty adjustment (DDA) in computer games: A review*. 2018. DOI: 10.1155/2018/5681652.

Chapter 9

Appendix

Appendices for this thesis are external and can be found in "External Appendix.zip" which is included in the hand-in. It can also be accessed from Google Drive via this link: https://drive.google.com/drive/folders/1GuDqbewn7EDOXorJVMWxsyUtsBaxQPy6?usp=sharing.

A Internship

- Internship Report: **A: Uncharted Chants: A game experience to engage cochlear implant users in music-perception training exercises.pdf**
- Concept idea descriptions and sketches: **A: Initial concept-ideas.pdf**
- Data of the idea evaluation questionnaire: **A: Data Idea Evaluation.xlsx**
- Results and graphs of idea evaluation: **A: Idea Evaluation Results.docx**

B Audio Plugins

- Additive synth JUCE plugin: **B: /AdditiveSynthPlugin**
- StiffString JUCE plugin: **B: /StiffStringPlugin**
- Dynamic string JUCE plugin: **B: /DynamicStringPlugin**
- Matlab implementations of physical models: **B: /MATLAB**

C Game Art

- Instrument monsters sprites: **C: /Instrument Monsters**
- Player character sprites: **C: /Player Character**

- Sound puzzle art: **C: /SoundPuzzle**
- Uncharted Chants logo designs: **C: logo2.png and logo3.png**

D Evaluation

- User experience questionnaire results: **D: User experience survey/UEQ-Results.xlsx and Results.xlsx**
- Transcribed CI user interviews: **D: CI User Interviews/Participant 1 - Transcribed Interview.docx and Participant 2 - Transcribed Interview.docx**
- Interview guide for semi-structured interviews: **D: CI User Interviews/Semi Structured Interview Guide.docx**
- Performance test: C# and C++ comparison: **D: Performance Tests/C# C++ comparison/**
- System performance comparison: **D: Performance Tests/System Performance Comparison/**

We collected consent forms from interview participants which can be shown on request.

E Unity Project

- Game Project (To be opened with Unity Hub): **E: Uncharted-Chants-master.zip**

F Game Demo

- Game installation files for Windows: **F: Windows - Uncharted Chants.zip**
- Game installation files for Mac: **F: macOS - Uncharted Chants.zip**
- Uncharted Chants Demo Video: **F: Uncharted Chants Demo.mp4**
- Training Exercise Tutorial Videos: **F: TutorialVideos/**
- Uncharted Chants Installation Instructions: **F: Game Installation Instructions.docx/**