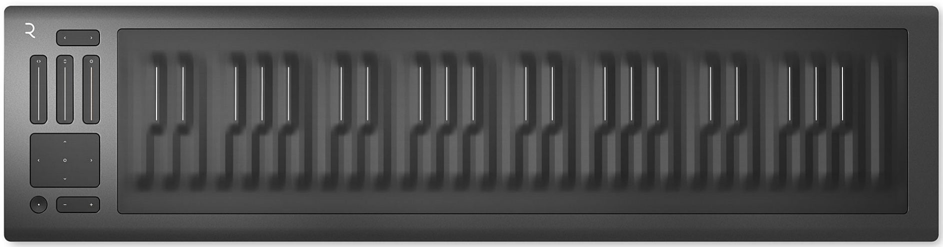
Real-time Animation with a Seaboard



Simon Ringeisen

Bachelor Thesis
May 2017

Supervisors:
Christian Schüller
Oliver Glauser
Prof. Dr. Olga Sorkine-Hornung

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

igl
INTERACTIVE GEOMETRY LAB

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

Real-time Animation with a Seaboard

Verfasst von (in Druckschrift):
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Ringeisen	Simon

Ich bestätige mit meiner Unterschrift:
- Ich habe keine im Merkblatt „Zitier-Knigge“ beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Brig, 04. Mai 2017	Simon Ringeisen

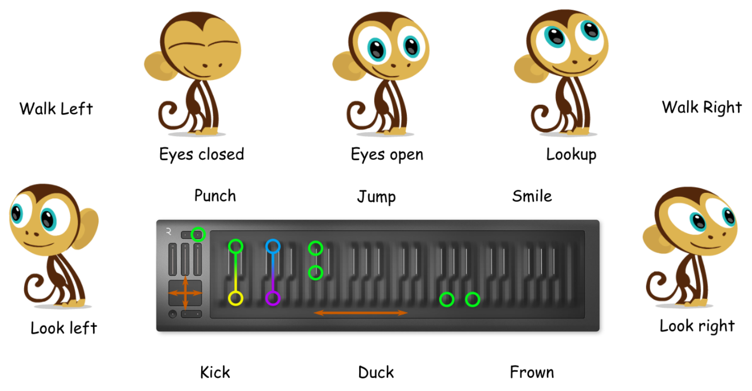
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

igl
INTERACTIVE GEOMETRY LAB

Bachelor Thesis

Real-time Animation with a Seaboard



Introduction

Expressive animations of digital characters and 3D models is a crucial part in animation movies and computer games. This artistic process is very time consuming and requires extensive experience with special tools like 3D editors. In particular, it is difficult to intuitively control the temporal aspect of an animation which is usually done iteratively by key-framing different poses and carefully tuning and playing the transitions between them. Special input devices were developed [1, 2] which allow a less experienced user to quickly manipulate rigged characters. While they are easy to use due to their kinematics structure, it is hard to control independent degrees of freedom (DOF) simultaneously with only two hands. Using dedicated tracking systems for body and face works well, but require an extensive hardware and software setup which is often not affordable for a single users.

Task Description

In this project we would like to explore the possibilities of using a Seaboard as an input device for real-time animation. This innovative evolution of a classic piano exposes 5 DOF for each touch. In theory, this provides 50 DOF if played with 10 fingers. In the initial step, a simple interface should be developed to receive and handle the Seaboard's midi output. Then the challenge is to derive and explore approaches to use this input to puppeteer and animate a character in real-time. What concepts used in the creation of modern electronic music could be adapted to this animation framework?

Detailed Task List

- Implement a simple framework to receive and handle the Seaboard's basic midi output.
- Familiarize yourself with the UNITY framework and create a simple puppet scene.
- Develop and implement a input handling, mapping concept to control the scene with the Seaboard.
- (Optional) Explore the concepts from modern electronic music and integrate them in this animation framework (e.g. Looper-Device, Multi-channel, ...).

Abstract

In this thesis we explore how a digital music instrument can be used for real-time controlling of animations. The device used is a Seaboard, which allows for multidimensional and polyphonic input. It gives the player the possibility, to control many parameters at the same time.

Animating digital characters can be a very unintuitive and difficult process. It often requires extensive knowledge and complex software. To create animations in real-time, which would be interesting for interactive live performances, is even harder. Currently used systems have various deficits. Drawing and sketching devices (like mouse and stylus) can only control a small number of parameters, motion capture systems are limited by the performance of an actor and with tangible devices it can be hard to control multiple body parts at the same time. Further such systems can be very expensive to obtain and time consuming to use.

We propose a setup to use the input of the Seaboard in the game engine Unity, look at different mappings, that could be used to animate a character and present the results of a small user feedback round. We explain, how the Seaboard doesn't offer a revolutionary way to create new animations, but is a suitable device to control animations in real-time.

Zusammenfassung

In dieser Arbeit erörtern wir, wie ein digitales Musikinstrument zur Steuerung von Animationen in Echtzeit genutzt werden kann. Beim benutzten Gerät handelt es sich um ein "Seaboard", welches multidimensionale und polyphone Eingaben erlaubt. Dies erlaubt dem Nutzer, viele Parameter gleichzeitig zu steuern.

Digitale Figuren zu animieren kann ein sehr unintuitiver und komplizierter Prozess sein und erfordert oft Wissen, Erfahrung und komplexe Software. Animationen in Echtzeit zu generieren, was für Aufführungen sehr interessant wäre, ist noch schwieriger. Heute genutzte Systeme haben verschiedene Nachteile: Geräte zum Zeichnen und Skizzieren (wie Maus und Stylus) können nur eine sehr kleine Zahl an Parametern beeinflussen, Motion Capture Systeme sind durch die Fähigkeiten des Schauspielers limitiert und mit Geräten, welche durch Veränderung von Modellen funktionieren, kann es schwer sein, mehrere Körperteile gleichzeitig zu steuern. Ausserdem können solche Systeme teuer in der Beschaffung und aufwändig in der Nutzung sein.

Wir präsentieren ein System, welches es erlaubt, das Seaboard in der Game Engine Unity zu nutzen, betrachten verschiedene Modelle um Eingaben auf Animationen abzubilden und präsentieren die Resultate einer kleiner Nutzerrückfrage. Wir erklären, warum das Seaboard zwar nicht ideal ist, um neue Animationen zu generieren, aber ein sinnvolles Gerät zur Steuerung von Animation in Echtzeit ist.

Contents

1 Introduction **1**

2 Related Work **3**
2.1 Different Devices to control an animation 3

3 Background **5**
3.1 ROLI 5
3.2 Unity 6
3.2.1 Anima2D 8

4 Mapping Methods **11**
4.1 Parameter types 11
4.1.1 Input parameters (Seaboard) 11
4.1.2 Output parameters (character animation) 12
4.2 What is a good Mapping? 12
4.3 Mappings 13
4.3.1 Key-Based Mapping 13
4.3.2 Area-Based Mapping 14
4.3.3 Gesture recognition 15
4.4 Comparison to other Input Devices 15

5 Implementation **17**
5.1 Overview 17
5.2 Technical Implementation 17
5.2.1 Getting the data from the seaboard 17
5.2.2 Connection Unity to the JUCE-library 18

5.2.3 Handling the data withing Unity 18
5.2.4 Manipulating a Character 19
5.2.5 The Character 19
5.2.6 The Mappings 23

6 User Feedback 27
6.1 Test Setup 27
6.2 Test Results 28
6.2.1 Interaction with the Seaboard 28
6.2.2 Ideas for Mappings 28
6.2.3 First Mapping: 1:1 KeyBased 28
6.2.4 Second Mapping: 1:2 KeyBased 29
6.2.5 Third Mapping: AreaBased 29
6.2.6 General Comments 29

7 Conclusion and Outlook 31
7.1 Reflection 31
7.2 Conclusion 32
7.3 Outlook 32

Bibliography 33

1 Introduction

Animated characters are used in many digital products like games or movies. Animating characters is a complex task, which requires a lot of knowledge and, depending on the used method, an extensive setup. It's even more difficult to perform such animations live. An example for such a scenario is envisioned in the Episode "The Waldo Moment" of the TV-Series [Black Mirror, Netflix 2013], where a cartoon character is animated live with a complex combination of face-tracking and other controllers, as shown in Figure 1.1.

Currently used systems have various deficits. Drawing and sketching devices (like mouse and stylus) can only control a small number of parameters, motion capture systems are limited by the performance of an actor and with tangible devices it can be hard to control multiple body parts at the same time. Further, such setups can be very complex and costly, so that amateurs mostly lack the resources to obtain or use them. In chapter 2 we take a look at different systems, and what their limitations are.

In chapter 3 we explore how a digital music instrument can be utilized for real-time animation. The device used is a Seaboard, which allows for multidimensional and polyphonic input: with each touch interaction, the user can control 5 parameters, whereof 3 are continuous. When all fingers are used, this allows for a high dimensional signal, which can then be used to animate a character. We first look at the Seaboard and how we can interact with it. Further we describe the tools that we use for this project.

A big challenge is to find sensible mappings that allow users to get used to the interaction quickly and perform on the surface naturally, while still being powerful enough to control one or multiple characters and their surroundings with many parameters. In chapter 4 we look at factors, which can be used to describe a good mapping and give an overview on possible kinds of interactions.

In chapter 5 we talk about the actual implementation, i.e. how we connect the Seaboard to Unity,

(a)

(b)

Figure 1.1: (a) *The cartoon character Waldo, shown in a live TV interview and* (b) *a performer controlling the character with different input devices, Netflix screenshots*

interpret the data and use it to control our demo-character. We describe, how our character is built up and present three mappings, which follow different approaches to map the input to a character animation.

We also conducted a small feedback round, in which we tested our setup by presenting three mappings to some participants, which gave us valuable insights. In chapter 6 we describe the test setup and some interesting results.

In chapter 7 we finally reflect over our work, conclude whether the Seaboard is a suitable device to create live animations with and suggest, what further research could be carried out in this field.

2

Related Work

2.1 Different Devices to control an animation

Sketching / Drawing

Mouse and keyboard are devices, that are regularly chosen for animation systems. They are cheep, highly available and familiar to users. One example of such a system is [Bai et al. 2016], where the user uses a mouse (and keyboard for shortcuts) to modify a character within keyframes, which are then interpolated. Their main work lies in the combination of simulated and keyframed animations. This is a big topic in animation, as keyframing alone gives the user creative freedom, but requires a lot of effort and knowledge, as creating natural movements is not trivial. Other systems like [Choi et al. 2016] rely on existing animations and change them by drawing (either with a mouse or a stylus as shown in 2.1). [Messmer et al. 2016] use modern hand held devices with cameras and touchscreens. Like this, animations can be created intuitively in an iterative process.

Although the creator has big creative freedom in such systems, they do not allow for real-time animation.

Motion Capture

Other systems rely on motion capture. "Character Animator" [Adobe Systems 2017] uses a standard webcam to track the face of a player, which is then mapped to a 2D-character. This enables the user to animate in real-time, but restricts him to facial expressions. Other unparameterized movements can be controlled by keystrokes or special triggering facial expressions. The

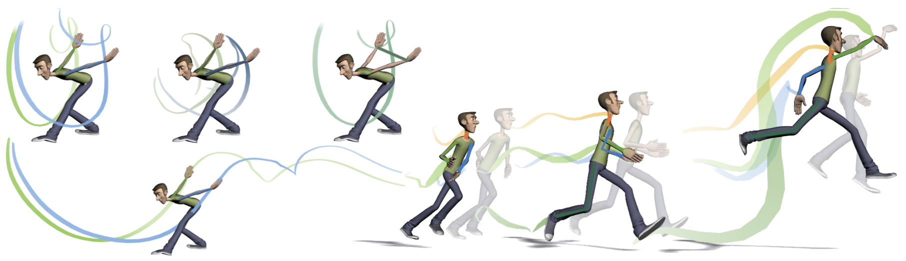


Figure 2.1: *SketchiMo: Sketch-based Motion Editing for Articulated Characters*

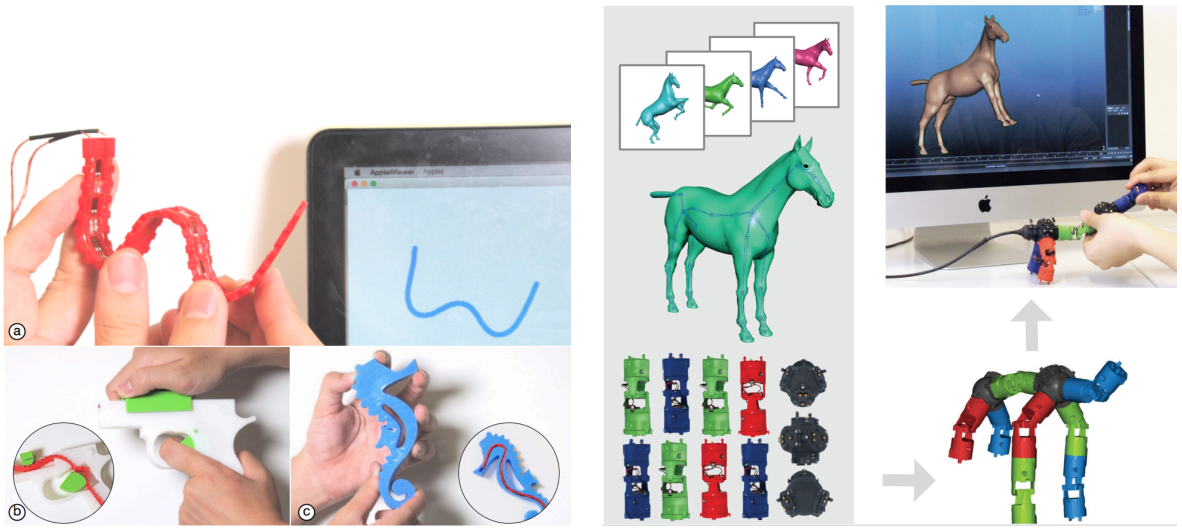


Figure 2.2: (a) *FlexiBend: Enabling Interactivity of Multi-Part, Deformable Fabrications Using Single Shape-Sensing Strip* (b) *Rig Animation with a Tangible and Modular Input Device*

user would e.g. not be able to choose the height in which the character holds its arms. Other systems use body tracking (e.g. [Ishigaki et al. 2009]), which allows for high-quality results, but require expensive setups. Cheep depth-sensing cameras allow a reduction of cost (while also reducing the quality compared to high-quality body tracking), but still restrict the user to humanoid movements. Further the animator is restricted by the physical abilities of the actor. Others (e.g. [Seol et al. 2013]) created systems to map humanoid movements to non-humanoid characters. Such a setup still requires an actor and is limited to a single character.

Tangible Devices

Tangible devices try to facilitate the process of animating by allowing the user to directly manipulate a physical structure. [Chien et al. 2015] rely on a shape sensing strip (2.2 a). This limits the user to a rather small set of character positions. This is overcome by systems like [Jacobson et al. 2014] or [Glauser et al. 2016], which use a modular approach with joints (which detect 3D rotations) and splitters (2.2 b). With such a setup the skeleton of the character can be mimicked. This allows for a very intuitive posing, but changing many bones at the same time can be hard to do in real-time.

3

Background

This thesis is based on the Seaboard, produced by ROLI[1], founded 2009 in London. Further we rely on software provided by the same company and on Unity. Within Unity we use the Plugin Anima2D. In this chapter we provide an overview of these tools.

3.1 ROLI

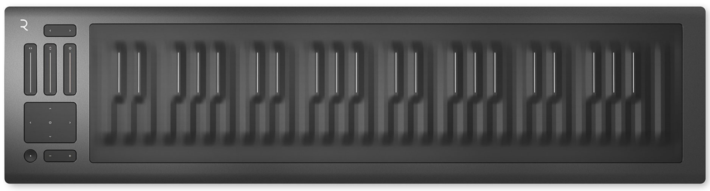
Seaboard

The Seaboard (3.1 a) is a music instrument shaped like a keyboard, but instead of individual keys, its surface consists of a continuous silicone material. The alignment of the so called 'keywaves' corresponds to key on a regular keyboard. These keywaves are surrounded by a ribbon on the top and the bottom. The complete surface is sensitive to 5 types of input, also called "Five Dimensions of Touch" [2]:

Strike	The velocity and force of a finger making contact with a keywave
Glide	Horizontal movements from side to side on a keywave and along the ribbons, also called pitchbend (the pitch of the generated sound)
Slide	Vertical movements up and down a keywave, also called timbre (the characteristic of a sound, e.g. how distorted it is)
Press	The continuous pressure applied to the keywave after the initial strike

[1]For more information see [ROLI Ltd 2017b]

[2]Definitions according to [ROLI Ltd 2017c], for icons see 3.1 b



(a)

(b)

Figure 3.1: (a) *The ROLI Seaboard RISE 49* (b) *The icons used by ROLI to depict the five dimensions of Touch*

Lift	The speed of liftoff from a keywave

Next to the keywave surface the Seaboard also offers 3 (one-dimensional) sliders and 1 (two-dimensional) XY touchpad.

For this project we use the Seaboard RISE, which is nothing but a MIDI-Interface (i.e. only detecting interactions and sending them to an interpreter, but not generating any sound). As regular MIDI [MIDI Manufacturers Association 2017] applies channel-wide messages (like pitch-bend and timbre) to all notes, a modification of MIDI is used: MIDI MPE (Multidimensional Polyphonic Expression) [Ben Supper and others 2015] uses the first MIDI channel for global messages (like the sliders and the XY touchpad) and the remaining 15 channels to transmit one note per channel, including channel-wide messages. We used the 'Seaboard Rise 49', with a note-range of 4 octaves. The sensitive surface covers a space of roughly 70x17 centimeters.

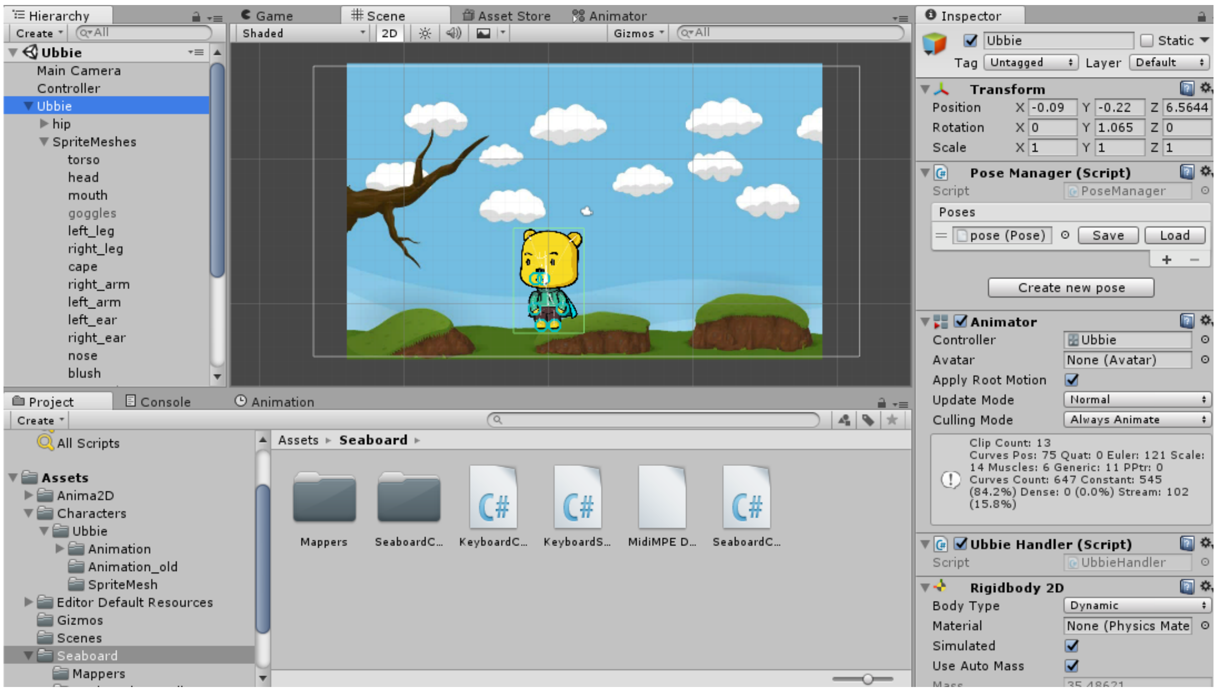
JUCE

JUCE (Jules' Utility Class Extensions) [ROLI Ltd 2017a] is a C++ framework that was first released to public by Jules Storer in 2009. In 2014 JUCE was acquired by ROLI. In this project we use modules from JUCE to easily connect a MIDI instrument and interpret the messages sent via MIDI MPE.

3.2 Unity

Unity [Unity Technologies 2017a] is a cross-platform game engine which was first announced in 2005. Unity is widely used and offers many tools necessary for game development.

Projects, Scenes and Scripts The project window (3.2) allows us to sort resources and updates internal links, when files are moved or renamed. The scene window let's us easily put together the elements of a game (called GameObjects), which then can be further manipulated with the inspector and the hierarchy view. Scripts (written in C# or JavaScript) allow for direct access to GameObjects in code.



Ubbie Static
Tag Untagged Layer Default
Position X -0.09 Y -0.22 Z 6.5644
Rotation X 0 Y 1.065 Z 0
Scale X 1 Y 1 Z 1
Script PoseManager
pose (Pose)
Controller Ubbie
Avatar None (Avatar)
Apply Root Motion ✓
Update Mode Normal
Culling Mode Always Animate
Clip Count: 13
Curves Pos: 75 Quat: 0 Euler: 121 Scale: 14 Muscles: 6 Generic: 11 PPtr: 0
Curves Count: 647 Constant: 545 (84.2%) Dense: 0 (0.0%) Stream: 102 (15.8%)
Script UbbieHandler
Body Type Dynamic
Material None (Physics Mate
Simulated ✓
Use Auto Mass ✓

Figure 3.2: *Overview of Unity: Hierarchy to the left, Scene in the middle, Inspector to the right, Project explorer on the bottom.*

Animations Further, Unity has an extensive toolchain to create and play animations: The Animation window allows us to record animations (either in real-time or step-by-step with keyframes) by tracking changes in the GameObjects parameters (e.g. position or rotation). These animations can be edited on a dopesheet (3.3) (table with time and changed parameters on the axes, allowing to move around dots, expressing when a parameter is changed) or by curves (3.4) (graph with time and values on the axis, allowing to move around curves, changing time or value of a parameter). Animations can then be used in the animator, which helps us to easily create a state machine, switching between animations. We are able to create different layers of a character to create state machines independent of each other (e.g. a head-layer for facial movements and a body-layer for body movements). Changing states can be induced by triggers, parameters (booleans, integers or floats) reaching a threshold or combinations thereof.

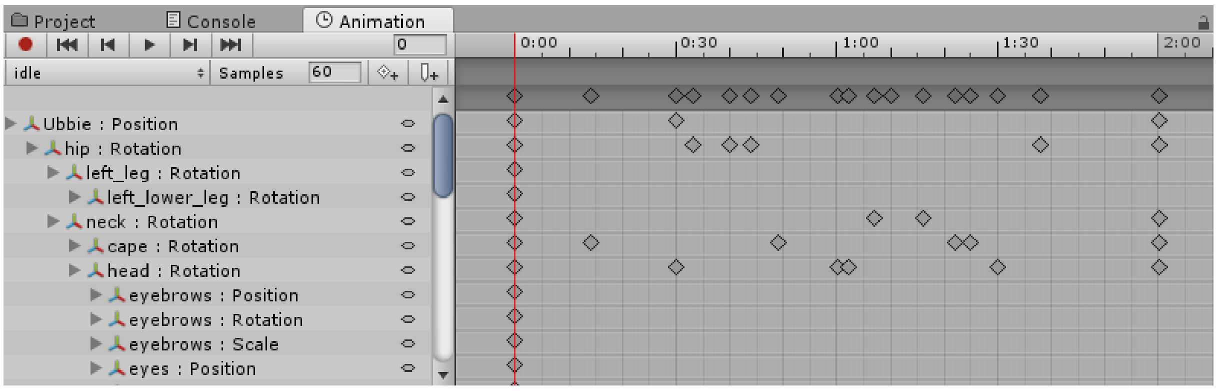


Figure 3.3: *Dopesheet*

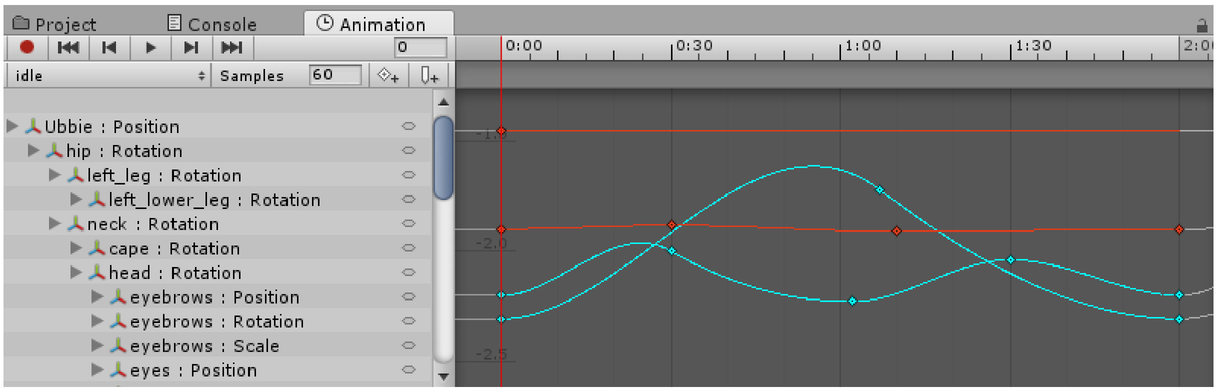
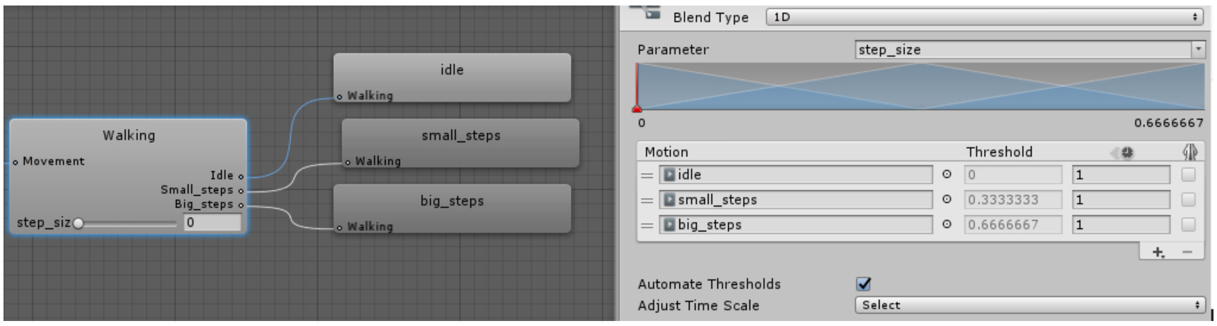


Figure 3.4: *Curves*



Blend Type 1D

Parameter step_size

0 0.6666667

Motion	Threshold	
idle	0	1
small_steps	0.3333333	1
big_steps	0.6666667	1

Automate Thresholds ✓
Adjust Time Scale Select

Figure 3.5: *The actual BlendTree to the left, parameters on the right*

BlendTrees A special kind of state is the so called BlendTree (3.5), which allows for interpolation between animations by a float parameter. In our project we make extensive use of BlendTrees: we use them to interpolate between animations (e.g. a linear interpolation of the character idling, walking in small steps and walking in big steps) and between states (which are technically still animations, but with a single keyframe, e.g. a linear interpolation between the character smiling and frowning).

This setup allows us, to first create a character, then create animations and animators and finally change parameters of the animators in code to play animations. More details on these tools can be found in the Unity user manual [Unity Technologies 2017b].

3.2.1 Anima2D

We use Anima2D to work with bones, skinning and inverse kinematics (IK). It is well integrated in Unity, which allows a seamless workflow.

Bones allow us to create a skeleton for a character, which is very useful when animating. A skeleton holds together the character and is the object we change when doing an animation.

Skinning is necessary, to stick a skin (i.e. a sprite with the texture of the character) to the bones. Anima2D also supports SkinMeshes, dividing a sprite into a mesh, which then gets weights on how much each bone affects a sprite at a certain point. This is useful to e.g. use a single sprite for an arm with two bones: The upper part of such a sprite will have high weights for the upper bone and low values for the lower one and vice-versa. A detailed explanation of this task can be found in [Komatsu 1988].

Inverse Kinematics are used to further facilitate the process of animating. Instead of moving every bone of an arm to the correct position, we simply move an IK-point at the hand of the character. The IK-mechanism then updates all bone positions automatically. Anima2D supports two types of IK-point. Limb-IK (as described by [Welman 1993]) is ideal for bone-structures like arms and legs, having a fixed, small number of bones bending in a certain way. CCD (Cyclic Coordinate Descent) IK (as described by [Wang and Chen 1991]) are made for a chain of bones, e.g. a tail.

4 Mapping Methods

The Seaboard provides users with a plethora of input parameters: if all ten fingers are used simultaneously, 30 continuous, 10 initial and 10 ultimate parameters can be modified. We need an intelligent mapping to the parameters of the animation, as it is already hard to control few fingers individually, especially if every finger has another purpose. In this section we look at some possible mappings and discuss the benefits and disadvantages.

4.1 Parameter types

4.1.1 Input parameters (Seaboard)

As stated in section 3.1.1, each finger on the surface of the seaboard provides us with 5 parameters. 'Strike' and 'Lift' only change when the finger touches, respectively lifts from the surface. 'Glide', 'Slide' and 'Press' are updated continuously while the finger is touching the surface. Further, there are global parameters, which can be used. On the left part of the Seaboard, there are 3 Sliders, one X/Y-pad, a function and an octave shift. Depending on the mode in which the Seaboard is, the 3 sliders can be used as global MIDI-parameters or to manipulate the sensitivity of the 3 continuous parameters 'Glide', 'Slide' and 'Press'. The X/Y-pad and the function shift can only be used as global MIDI-parameters, while the Octave-shift does not send any additional MIDI-parameters, but only changes the note-range of the keys.

The Seaboard also has some limitations, most of them being implied by the fact, that it was designed to be a music instrument. Especially two fingers, which are to close together, might be interpreted as a single finger.

4.1.2 Output parameters (character animation)

There are many different parameters that can be used to animate a character. We group these into direct parameters, which directly manipulate a physical entity, like the position of an IK point or the angle of a bone, and indirect parameters, like the speed a character runs at. Although all of these parameters can be represented by real numbers (as every direct parameter of a digital character can be parametrized by position, scale and rotation and indirect parameters are already abstracting some behavior, e.g. the blending between two animations, by using a real number), for some (e.g. the direction in which the character looks), a binary representation makes more sense. In 5.2.5 we give a concrete example of such parameters.

4.2 What is a good Mapping?

To rate Mappings, we first need to know, what the advantages of a good mapping are. [Seffah et al. 2006] look at multiple models to compare usability of software and describes the QUIM model, consisting of ten factors. Not all aspects for software usability are relevant for a mapping (such as safety[1], trustfulness[2], accessibility[3], and universality[4]) and others (i.e. scope) are missing. We will therefore use the following set of factors to compare the mappings.

Learnability

Learnability describes how much effort it takes to be able to understand a mapping. It highly depends on how intuitive a mapping is. This also includes the effort to learn the physical movement (and therefore also the complexity of the necessary movements), which can be very dependent on the skills and previous experience of the user.

Efficiency and Effectiveness

These factors describe, how much effort it takes to reach a goal and how accurate the task can be executed. The factor would be reduced by physical limitations, like very fast movements, which have to be performed for a long time or keys to touch, which lay too far away from each other.

[1]Safety: How safe a certain software is to use, i.e. not harming resources or data, as described in [ISO/IEC 9126-4 2001]

[2]Trustfulness: How much the user trusts the software, can be very important for e.g. e-commerce websites

[3]Accessibility: How well software can be used when being disabled. This would also be an interesting aspect, but is not the focus of our work.

[4]Universality: How well people with different cultural backgrounds can use the software, e.g. if it is translated into different languages. This is not relevant for the rather technical term of a mapping.

Scope

Scope is not included in QUIM, but as one of the biggest advantages in using the Seaboard lays in the high amount of manipulatable parameters, we need a notion to describe how many of these are actually utilized. Scope therefore defines the ratio of available parameters on the input device compared to the parameters which are used in a mapping. If this factor is very low, we waste some potential of the Seaboard. This parameter loses its importance, when we only want to control few parameters which easily can be spaced on the Seaboard, but is very important for controlling a big scenery with many characters and lots of parameters to manipulate the environment.

Usefulness

The Usefulness depends on the previous parameters, but instead of focusing on the overall mapping, we want to know how many parameters we can control at the same time. An example for a low usefulness would be, if we mapped 3 parameters onto three keys, which can't be reached with one hand at the same time. For a higher usefulness these three keys could be placed close enough, that the user can control all of them simultaneously. A high Usefulness is often opposed to the Learnability, as a higher density of parameters can imply a more complex mapping, being more difficult to understand.

Other Parameters and the Perfect Mapping

Satisfaction, the subjective feeling when using a thing, is very important when using a new technology and will be observed in chapter 6, but can't be seriously discussed on an abstract level. Productivity (how much time is spent on the actual task, not on setting up the environment etc.) only makes a difference if there is a task involved, not directly solving the actual task. We will not focus on this, as most presented mappings are static. This would however play a role when comparing different learning mapping systems.

Where the sweet spot between Learnability and Usefulness lies, depends both on the situation where the mapping is used and the abilities of the player. For a novice user, e.g. at an exhibition, an intuitive mapping is more valuable, while a proficient user, e.g. the animator of a TV-show, would prefer a high Usefulness of the mapping.

4.3 Mappings

4.3.1 Key-Based Mapping

In a Key-Based Mapping each key relates directly to one parameter (one-to-one) or to a set of parameters (one-to-many) of the character. An example of this could be a linear relation between the timbre and the walking speed.

In a one-to-one mapping only one type of interaction is allowed per key and the interaction may not span over multiple keys. For this reason, 'glide' is not very interesting for this kind of mapping. A one-to-one mapping can be very intuitive, especially due to it's simple 'cause -> effect' relationship. It is also effective, as a finger is only used for a single interaction-type. Further, a one-to-one mapping has a pretty high Scope, as we can in total map 49 keys to 49 parameters. But this kind of mapping has a pretty low Usefulness: if we assume, that we can only move a couple of fingers at the same time, we end up with only a couple of manipulatable parameters. An example for a one-to-one mapping would be the pressure of a single key being linked to the speed the character walks at.

A one-to-many mapping can tackle this problem. In using more than only one interaction-type on a single key (but still not allowing interaction over multiple keys), we can manipulate many more parameters at the same time. It therefore has a higher Efficiency and a higher Usefulness. The Learnability now depends on the set of parameters which we map to a key: if they are parameters of one logical parameter group, such a mapping can be very intuitive. The Effectiveness also depends on the concrete mapping: a mapping that requires big movements (timbre/pitchbend) of fingers on the same hand can be very confusing. This is a situation where advanced piano players have a big benefit, as they already have some muscle memory to perform these complex movements. An example for a one-to-many mapping would be the pressure and timbre of a single key being linked to the speed the character walks at, respectively how zestfully he walks .

4.3.2 Area-Based Mapping

Instead of looking at each key individually, we can take advantage of the Seaboard's continuous surface and combine keys into areas. In these, we can not only look at the keys as separated elements, but as groups. This allows us to use not only the 5 standard interaction-types of the Seaboard, but also combinations thereof. Especially elements from music theory, like intervals and chords, might be interesting. These are already well known by pianists and allow a low learning-curve. Also, this already allows for some gesture-based movements, further increasing intuitivity: Opening the gap between two fingers can easily be visualized as opening the mouth of a character. The usage of such visualizable movements help to make a mapping intuitive. But this can, especially for non-musicians, decrease the Efficiency, as the finger-positioning does not feel natural. This kind of mapping allows for many parameters within a small area: depending on the different combinations of the fingers we can choose which parameter to manipulate in which way. The Usefulness of such a mapping can be very high, but is also dependent on the abilities of the player.

Special case: Select/Modulate

A special case of this area-based mapping is the Select/Modulate mapping: we use one area to decide which parameters we want to manipulate on the other area. An example would be the left hand deciding which bodyparts the right hand manipulates. This again provides an extensive usage of the available space, but decreases the Usefulness in the case where we use both hands for one and the same action. As we are not differentiating the modulator's input we have to apply the same changes to the selected parameters. Cognitively, this mapping might be very interesting in respect to Learnability: as with many music-pieces we lay the base with our left hand and modulate the melody with the right hand.

4.3.3 Gesture recognition

A higher-level approach would be, to rather look at gestures than at exact inputs. This could allow for very intuitive mappings: we could pinch to increase the size of our character or could mimic a walking-gesture to define the walking speed.

For this system, one would require a well trained gesture recognition system. With such a system, the user could teach the system its own gestures and mapping. In respect to intuitivity this would be very interesting for proficient users, but could be very confusing for novel and short-time users. Depending on the sensitivity of such a recognition system, one could extensively use the Seaboard's surface while playing. In general such a system could leave many choices to the player, which would especially be very interesting for proficient users.

4.4 Comparison to other Input Devices

Sketching/Drawing devices often only offer a very limited range of available parameters. A mouse offers two continuous dimensions (position on XY-axis) and a stylus might additionally provide applied pressure and the angle, at which it is held. The devices only have a small number of parameters, especially when compared to a device like the Seaboard, which limits the amount of parameters that can be changed simultaneously (and therefore reducing the Usefulness). On the other side, sketching and drawing devices are familiar to users, which helps them to get used to the setup faster (increasing the Learnability).

Tangible Devices suffer from a similar problem. Although they may (depending on the actual device and setup) have many more parameters which can be changed at the same time, it can be hard to control several joints with high precision as it is hard to grab and move them individually. The biggest benefit uf such a device is that the user can represent the characters structure, making the interaction very intuitive, which leads to a very high Learnability.

Motion Capture has the big benefit, that it can track human movements in the most natural way possible, which makes it an ideal system for humanoid animations. This benefit decreases, when tracking animations for non-humanoid character. Even though some methods can solve this problem with intelligent mappings, the animator is still restricted to one animated character and the ability of the actor. This limitation to create unrealistic (cartoonist) movements can be interpreted as a limited Usefulness.

5 Implementation

In this chapter we present the developed system to animate characters with the Seaboard, using JUCE and Unity as main tools.

5.1 Overview

The Seaboard is a MIDI-Interface, which sends it's data, in the form of MIDI-MPE to a PC via USB-cable. This data is caught by a DLL, which uses the C++ framework JUCE to connect to the Seaboard and decode the MIDI-MPE binary data, and forwards it to Unity via callback-functions. Within Unity this data is organized in a Controller, which forwards the data to a mapper. Using a character handler, the mapper manipulates the parameters of the animated character. A symbolic overview is provided in 5.1.

5.2 Technical Implementation

5.2.1 Getting the data from the seaboard

To connect to the Seaboard and decode the binary data we rely on JUCE (3.1). We allow connection to an instrument with the InstrumentHandler class. This class exposes callback functions, which will report new and removed notes, as well as changes in pressure, pitchbend and timbre. Further the InstrumentHandler class automatically connects to the first MIDI-input device that can be found. It enables input for this device, creates zones on the instrument, for which we want to get the data, and initializes the AudioDeviceManager, which will trigger Callbacks, as

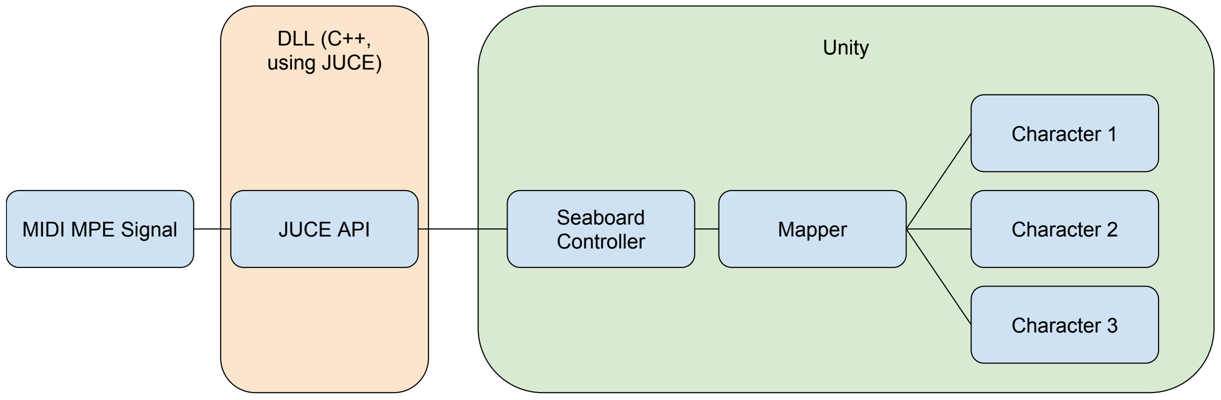


Figure 5.1: *Overview of our system setup to get the data from the Seaboard into Unity to animate a character*

soon as we get new inputs. Further we add a Listener to the MPEInstrument, which will in our InputHandler class, call functions for adding and releasing notes, as well as changing pressure, pitchbend and timbre. In these functions we change the values to be conveniently usable in our system and call the appropriate function to return them to the consumer.

5.2.2 Connection Unity to the JUCE-library

As JUCE is written in C++ and we use C# in Unity, we need to connect these two languages with each other. We used C# Marshaling for this, offering external C-style functions in a C++ DLL to link to the instrument. In Unity we import these C-style functions from the DLL and setup the callback functions using delegates.

5.2.3 Handling the data withing Unity

The main component to connect the Seaboard to the mapper is the SeaboardController. Latter uses the Linker to establish a connection with the DLL file and fills a local data structure with the currently played notes on the Seaboard using the aforementioned delegates. The currently active notes (interactions) are stored in a Dictionary, the ID generated by the Seaboard being the key of the interaction. This allows to track the interaction, even if the user moves over multiple keys. Further the Controller offers the possibility to listen to changes, again using callback functions. The SeaboardController listens to all these changes and adds them to a List. As Unity only allows certain changes on Characters within the game loop, we have to await the next time the 'Update' function is called. As soon as one occurs, we go through this list and send all the events to the responsible mapper. In our demonstration, we divide the Seaboard into 4 blocks, each with the size of an octave. In the first block, we can select which mapping we want to use. The other 3 blocks are then evaluated using this mapping. Each mapping offers a 'map' function, which receives a short version of which note has encountered which change (NoteOn, NoteOff, PitchbendChanged, TimbreChanged and PressureChanged) and a detailed object, in which state the current note is. This allows the mapper to implement a state based, event based or combined mapping.

5.2.4 Manipulating a Character

The mapper (5.2.6) then depicts what to manipulate on the character. The manipulation is done with the help of a CharacterHandler, which abstracts the interaction between Unity's GameObject and the mapper. A CharacterHandler defines a list of Elements, which are visible to the mapper. These Elements all consist of 'getParams', 'offsetParams' and 'setParams' functions, which allow the mapper to analyse the character and change it. This dynamic approach is ideal to add dynamic mapping, which might also depend on some learning system. Also, this allows for a rather simple exchange of the Character.

5.2.5 The Character

In our demonstration, we use a bone-based 2D-character. The character consists of sprites, which are put together, bound to bones (as described by [Komatsu 1988]) and appended with IK points.

We then created animations using Unity's Animation tool, which allows for keypoint based animations. Especially, all facial expressions are animations with a single frame. Using Unity's Animator and Layers created with Anima2D, we use blendtrees and statemachines to blend and switch between animations. This allows for a simple, parameter based interface and spares us doing any interpolations.

Sprites and Bones

As a starting point for our sprites we use "Ubbie", a Character of [UYoung Culture & Media Co., Ltd. 2017], which is provided as a demo-project in [Dragonbones 2017]. We exported the character to a 1024x1024 pixel image using 'DragonBones' and later edited it to fit our needs. Modifications to the sprites were necessary, as we wanted to separate facial elements (mouth, eyes,...) from the head itself. Further we combined the upper and lower arm to get a seamless transition when bending the arm with bones and added some own elements, like eyebrows and red cheeks. This sprite image (5.3) is then separated into individual sprites, one per body part and facial element. After placing these sprites in the scene, we add bones using Anima2D. Starting with the root bone, the hip, we create a skeleton covering arms, legs, head, cape and ears. As we also want to animate the facial expressions, we add bones to control mouth, eyes and eyebrows. We add a further bone to be able to animate the goggles. Finally, we need to bind the sprites to the bones, which is accomplished using Anima2D's 'SpriteMesh Editor'. Now the sprites will follow the movement of the bones. An image of the setup is shown in 5.3 (a).

IK-points

'Anima2D' allows to handle IK points in two different ways: with 'Cyclic Coordinate Descent' (CCD) (as described in [Wang and Chen 1991]), which can handle a chain of bones, and 'Limb' (as described in [Welman 1993]), which is specialized to handle two-bone structures like arms

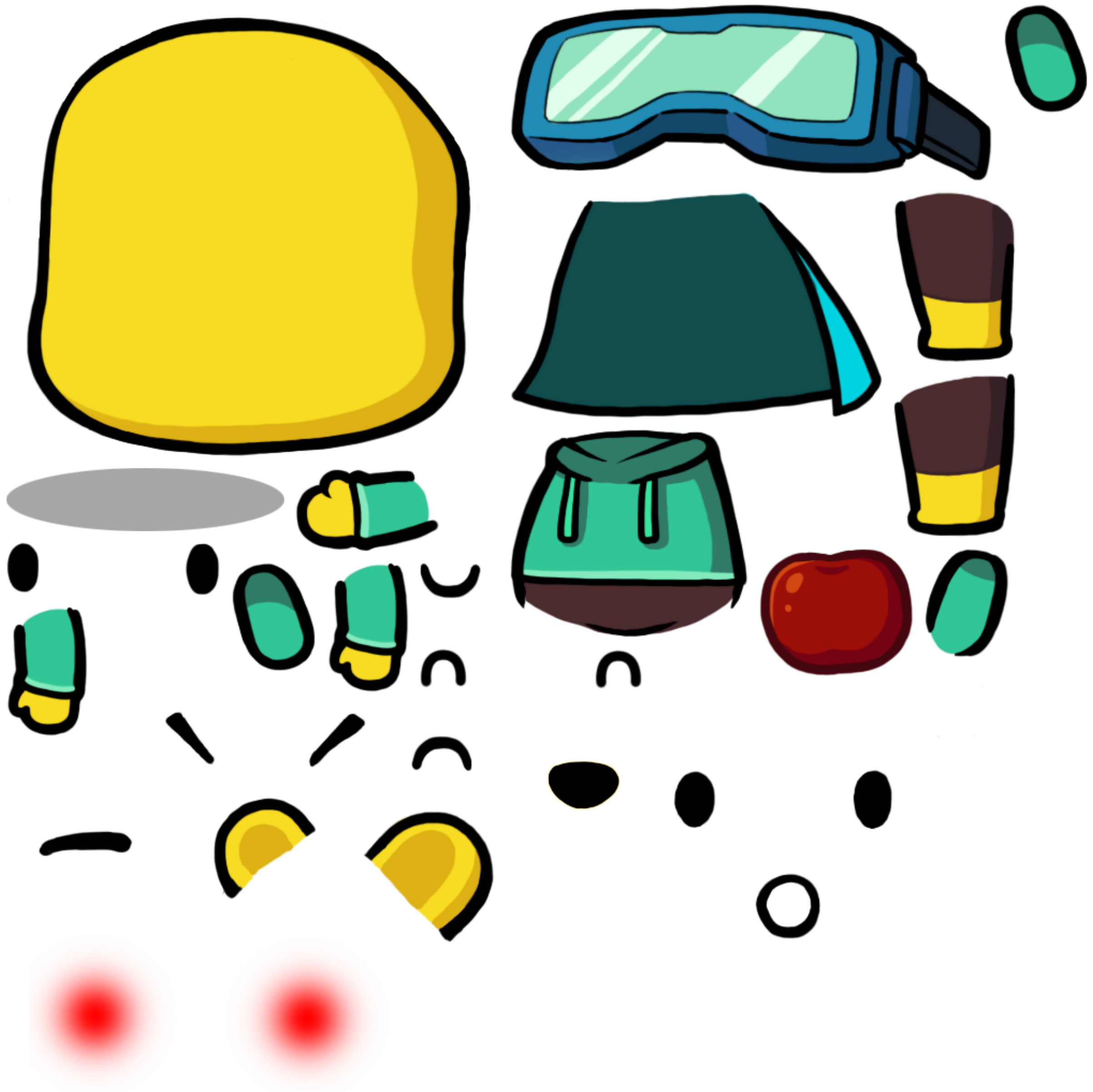


Figure 5.2: *The texture of our character "Ubbie"*

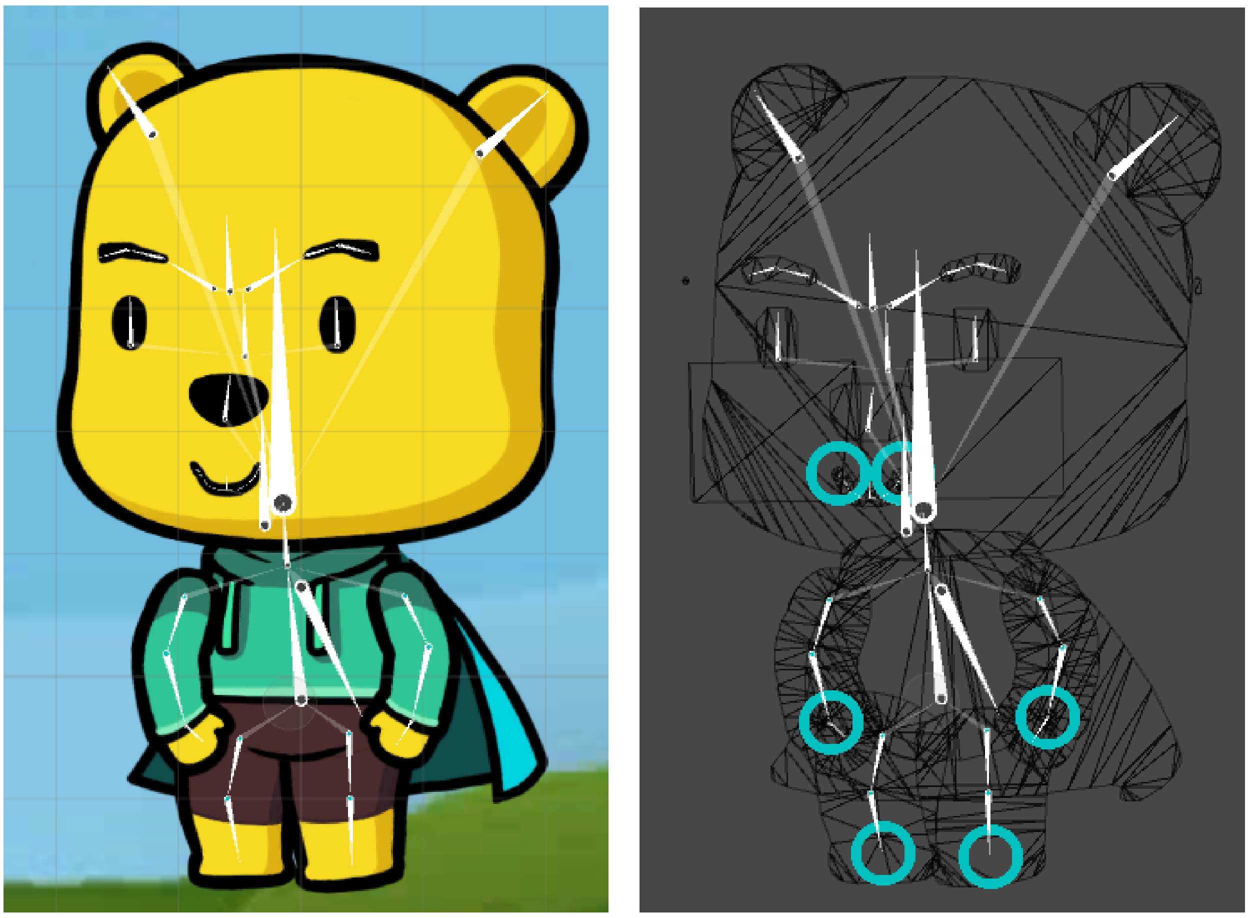


Figure 5.3: (a) *Sprites and Bones of our character.* (b) *IK points of our wireframed (showing the actual mesh applied to the sprites) character.*

and legs. Especially, the 'Limb'-IK allows to define in which direction the arm/leg is facing, giving the user the possibility to choose in which direction the middle joint will bend, thus bending arms and legs in unnatural ways can be prevented. For our character, we use Limb-IK for arms and legs and CCD-IK in the mouth corners. Using these IK-Points it is much easier to pose the character to a desired position, which facilitates the process of animating. An image of the setup is shown in 5.3 (b).

Animations and BlendTrees

To animate our character, we use Unity's Animation tool, which allows to create keypoint based animations. The record tool enables the user to pick a frame and then manipulate the character. The Animation tool will detect the changed positions, rotations and scales, only adding these to the so called Dopesheet, which is basically a table with the frames on the X-, and the animated elements on the Y-axis. Further Unity's Animation tool ensures smooth animations by manipulating the actual curves interpolating between the different positions of the character.

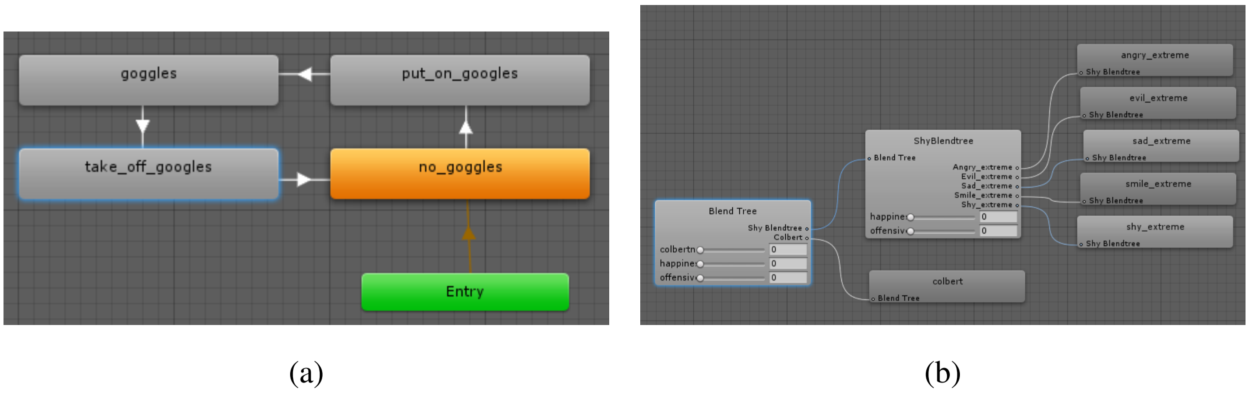


Figure 5.4: (a) *The state machine controlling the goggles.* (b) *The BlendTree controlling facial expressions.*

For our character we created a set of animations: 'idle' (the character is slightly moving to prevent an unnatural still standing position), 'small steps' and 'big steps' (where the character is walking in different step sizes, but with the same pace) and 'put on goggles' and 'take off goggles'. Further we create animations with a single frame for different facial expressions: smile, sad, evil, angry, shy and colbert[1]. We use animations for facial expressions as we want to be able to plug them into blendtrees where Unity can interpolate between them.

Unity also provides us with Animator, a tool to create state machines and blendtrees to switch and interpolate between animations. As we want to be able to control the face independent of the body, we separate the character into multiple layers, one for the complete body, one with the right arm and the goggles for the "put on goggles" and "take off goggles" animations and one with all facial elements for facial expressions. As we want the latter ones to be more visible, we set the weight of their layers to 100%. This will overwrite movements in the uppermost layer of the complete body.

With the layer for the complete body we control the walking movement. Two parameters for speed and stepsize control a blendtree: speed changes the actual speed at which the animation is played, stepsize controls a blendtree that interpolates between the three animations 'idle', 'small steps' and 'big steps'.

To control the goggles we use a state machine (5.4 a), which goes from an animation, where the character is wearing no goggles, over the animation of putting on the goggles, to a animation, where the character wears the goggles (and the other way around). Such a state change is brought forth by a trigger. We further use another parameter to change the animation's speed

To control the facial expressions we use a blendtree (5.4 b) with multiple steps. In a first step, we handle the more extreme expressions: colbert and shy. We use a parameter to interpolate between shy, other expressions and colbert. In a second step, we use two parameters (happiness and aggressiveness) to blend between the four animations angry, evil, sad and smile, where we say that smile and evil have a high happiness score and angry and evil have a high aggressiveness score.

[1]A reference to the Late Show host Stephen Colbert, who proposed the "Face With Raised Eyebrow" Emoji, added to Unicode 10.0

name	*type*	*description*
walking_direction	boolean	The direction, in which the character is facing
toggle_goggles	trigger	Triggers the character to put on / take off the goggles
toggle_goggles_speed	float	Defines the speed, with which the character puts on / takes off the goggles
step_size	float	Defines the size of the steps
walking_speed	float	Defines the walking speed
happiness	float	Defines, how happy the character is (from sad to happy)
offensiveness	float	Defines, how offensive the character is (from shy to aggressive)
colbertness	float	Defines, how strong the colbert expression (Face With Raised Eyebrow Emoji) is done

Table 5.1: *The parameters that can be changed in our character*

Animation Parameters

We have already mentioned the use of parameters to manipulate our character. 5.1 provides a complete list of all parameters, and what they change.

5.2.6 The Mappings

To test different approaches, we implemented three mappings. Each of them focuses on a type of mapping, but can also contain elements of other types. None of these mappings is supposed to be a perfect one, but rather help us, to rate the different approaches.

1:1 KeyBased

The first mapping (5.5) focuses on a one-to-one mapping between keys and parameters. From left to right, we map to these parameters:

The **goggles** are controlled by hitting the key. The harder, the faster the goggles are put on / taken off. The continuous pressure can be used to further change the speed.

The **direction** in which the character is facing, is changed by touching the key.

The **number of steps** are controlled by the timber of the key, low meaning few, high meaning many steps.

The **step size** is controlled by the timber of the key, low meaning small, high meaning big steps.

The **happiness** is controlled by the timber, low meaning sad, high meaning happy.

The **offensiveness** is controlled by the timber, low meaning shy, high meaning aggressive.

The **colbertness** is controlled by the timber, low meaning no colbert, high meaning colbert.

As the colbert expression is incompatible with the other facial expressions, it has precedence over them.

1:2 KeyBased

The second mapping (5.6) focuses on a one-to-two mapping between one key and two parameters. From left to right, we map to these parameters:

The **goggles** are controlled by hitting the key. The harder, the faster the goggles are put on / taken off. The continuous pressure can be used to further change the speed.

The **direction** in which the character is facing, is changed by touching the key.

The **number of steps** and the **step size** are controlled by the timber and the applied pressure on the key, low timbre meaning few, high meaning many steps, low pressure meaning small, high meaning big steps.

The **happiness** and the **offensiveness** are controlled by the timber and the applied pressure on the key, low timbre meaning shy, high meaning aggressive, low pressure meaning sad, high meaning happy.

The **colbertness** is controlled by the timber, low meaning no colbert, high meaning colbert.

As the colbert expression is incompatible with the other facial expressions, it has precedence over them.

AreaBased

The third mapping (5.7) focuses on a mapping between areas and parameters. From left to right, we map to these parameters:

The **goggles** are controlled by hitting the key. We separate the key in two parts: touching the upper part puts the goggles on, the lower part takes them off. The further away from the key's center the user touches, the faster the action is executed.

The **number of steps** and the **step size** are controlled by the timber and the applied pressure on the key, low timbre meaning few, high meaning many steps, low pressure meaning small, high meaning big steps.

The **direction** in which the character is facing, is changed by touching the area with a second finger, while touching the key to the left at the same time. This corresponds to an interval on the piano. The direction is preserved when the finger is removed.

The **happiness** and the **offensiveness** are controlled by the timber and the applied pressure on the key, low timbre meaning shy, high meaning aggressive, low pressure meaning sad, high meaning happy.

The **colbertness** is controlled by touching the area with a second finger, while touching the key to the left at the same time. This corresponds to an interval on the piano. The value is set to zero when the finger is removed.

As the colbert expression is incompatible with the other facial expressions, it has precedence over them.

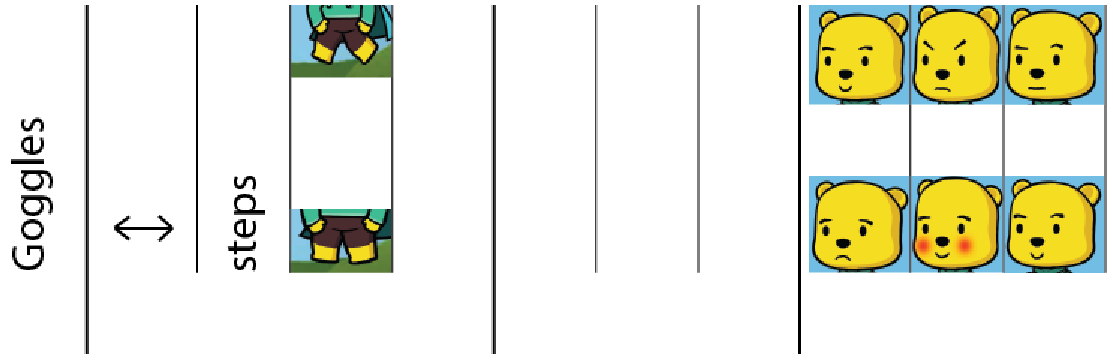


Figure 5.5: *1:1 KeyBased Mapping*

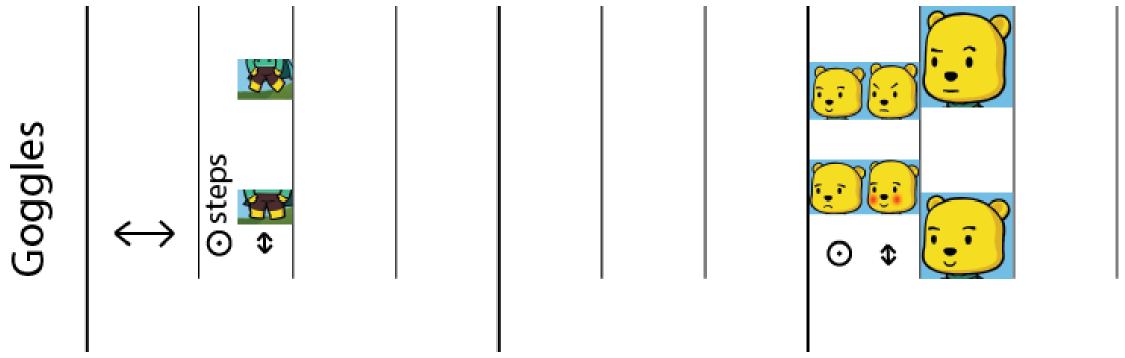


Figure 5.6: *1:2 KeyBased Mapping*

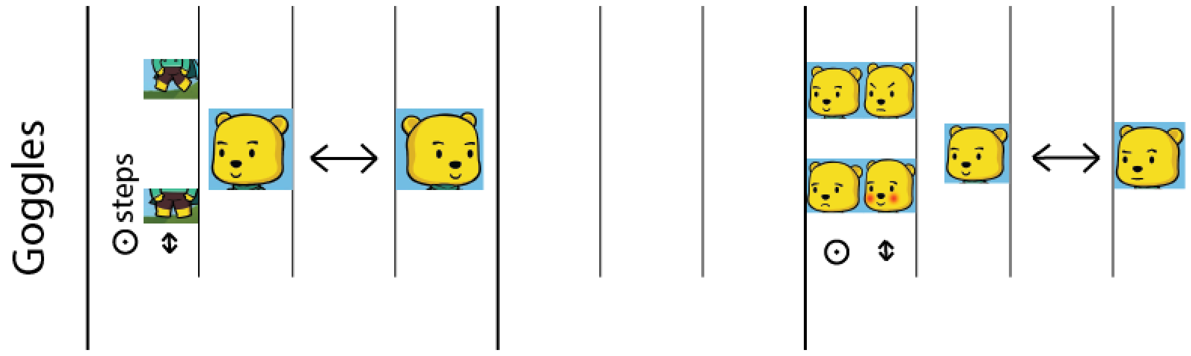


Figure 5.7: *AreaBased Mapping*

6

User Feedback

6.1 Test Setup

We set up a test that takes about half an hour to conduct. It consists of a general part, where we learn about the tested person (e.g. abilities to play instruments, knowledge in animating/puppeteering, whether they had knowledge of or experience with the Seaboard before). After this, we presented the tested person with our character and told them, what it can do. The tested person was then asked to propose a possible mapping[1].

In the next part, we presented the persons with three different mappings (as described in 5.2.6), explained them and let the persons play with them for a while. Later, they were given a list of tasks (like "make the character smile"), which they had to execute. Finally, they were asked, what they liked and disliked about every mapping and what they would improve.

The participants were equally split into females and males and half of the participants were good piano players, but none of the participants played the Seaboard regularly. Most participants were aged between 18 and 30 years old. Six tests were conducted by filming the participants, their inputs on the Seaboard and the moving character, while the other four were conducted like an interview, where we regularly invited the participants to share their opinion. Further, the mappings were improved after a first series of tests. Thus, these results do not represent a scientific study, but rather a feedback which helped us to rate our work and mappings and to conclude, whether the Seaboard is an interesting device to use for our purpose.

[1]Only the second group was asked this question

6.2 Test Results

In this section, we present our findings after testing ten people. The results were attained by questioners, directly interviewing the participants and by analyzing the video-material.

6.2.1 Interaction with the Seaboard

People, who had not interacted with a Seaboard before, approached it very differently. Some were very careful when touching the surface, although it is very durable and most participants were surprised by the sensitivity. Pianists were generally confused that the keywaves are, in contrary to a keyboard, not a flat surface. Further, some participants had difficulties to distinguish black and white keys and/or were confused by the missing tactile feedback.

We noticed two points, where it made a difference, if the participant had experience playing a piano. First, they were able to find the correct key to hit much easier and most pianists also tried to place (and rest) their fingers on the surface according to the mappings. Second, pianists more often had problems, to 'Glide' over the surface, as this is a movement that a normal Pianist would never use and might therefore be contra-intuitive.

6.2.2 Ideas for Mappings

We were given many different ideas when asking the participants for their own idea of a mapping. They ranged from an AreaBased-Mapping, where the surface is used like a touch screen to change parameters, over an approach using the addition inputs (XY-pad, sliders) to select a bodypart, which would then be modified on the surface, to a technique, interpreting what the user plays in a musical way (e.g. which harmony is played at which pace). It was further interesting, that most pianists associated basic movements of the character's body in regions with lower keys and more detailed elements to higher pitched keys.

6.2.3 First Mapping: 1:1 KeyBased

The 1:1 KeyBased mapping was well understood. Especially putting on / taking of the goggles and changing the walking direction was reported to be very easy. Some participants had difficulties with the far range of the 'Slide', reaching over nearly the complete key. Also, for some it was very hard to move multiple fingers in different directions, e.g. when reducing the step-size while increasing the step speed at the same time. Further, most participants were confused by the separation of step size and step speed and had difficulties to remember, which does what. In contrary most participants liked the facial control, where happiness and offensiveness are controlled on different keys.

In this mapping, as well as in the 1:2 KeyBased mapping, the precedence of the colbert, respectively the need to disable colbert to be able to use another facial animation, confused many participants.

6.2.4 Second Mapping: 1:2 KeyBased

For most participants, the 1:2 KeyBased mapping was slightly harder to understand and less intuitive than the 1:1 KeyBased mapping. Controlling the facial expressions being more difficult was mentioned most. Some participants had trouble developing a good feeling for how much pressure to apply to get the desired effect. Generally, the walking animation was easier to change, as the combination of step-size and speed onto a single key made more sense for the participants.

6.2.5 Third Mapping: AreaBased

The AreaBased mapping was generally rated more difficult to understand, but easier to use. The new way to control the goggles was not received well. Some participants repeatedly tried to trigger a change by sliding up/down the key instead of pressing at a certain position. The way to change the direction, in which the character looks, further confused some participants. Most of then got it right after trying a couple of times and rated it more intuitive then before. One participant proposed to use this new, finer parametrization to change the direction of the character not only binary. Another one suggested to use two distinct keys for each direction.

Two piano-playing participants further suggested, to change the direction in which the interval is made on the right hand. Like this, the movement would be mirrored from the left hand, as it is often when playing the piano. Generally we noticed, that participants tended to keep their left hand on the surface more often. Many of them even unconsciously controlled the walking animation during the whole test, especially changing the direction when the character was near the boarder.

6.2.6 General Comments

It was very interesting to see, how people with similar abilities preferred different kinds of mappings. Most of the participants further liked the idea of using the Seaboard as a tool for animating and many had a lot of ideas, how the character could be extended to do different things.

The participants needed some time, to get used to the Seaboard, especially to get a feeling for how sensitive the Seaboard is. Further we realized, that users were confused very fast, when the setup did not produce the animation, which they were expecting. This partially lies in the missing ability of the Seaboard to give any direct (e.g. haptic or visual) feedback.

7

Conclusion and Outlook

In this chapter we reflect over our work, conclude whether the Seaboard is an suitable device to create live animations with and describe, what further research can be carried out in this field.

7.1 Reflection

Thanks to the well written, fairly well documented and useful examples of the JUCE library, we were able to quickly create our C++ interface to get the data from the Seaboard. We then had some problems using Unity, as it has a modified way of interpreting and handling C# code. After solving these problems, we familiarized ourselves with Unity and conducted many experiments, creating projects which displayed the inputs of the Seaboard and animated two 3D characters in many different ways. In retro perspective, it would have made more sense to first decide on a scenario, which we want to use for our demo. Like this, we spent a lot of time learning a 3D setup, which we eventually did not use. We decided to use a 2D-scenario, as it was much easier (for a user not familiar to creating 3D animations and characters) to create own 2D animations and modify characters. Due to lacking time, we only were able to implement and test three small mapping schemata.

In general, the weekly meetings helped a lot to keep in pace with this rather open topic. Although we, as already mentioned, lost some time on test-projects, we were able to achieve nearly all goals set in a first timetable. The only thing missing is an augmented view within Unity, to allow the user to see more, than just the animated character. Further, we would have liked to additionally look into some methods of modern music creation, like loopers and multi-channel setups.

7.2 Conclusion

With this project, we created a working toolchain to test the applicability of a Seaboard in the field of animating. We present methods for basic mappings and enable simple interaction with a character by providing a CharacterHandler, that can control parameters of animations, bones and sprites. Other users will be able to easily create new mappings and extend the ability of the character within Unity. Especially, with this toolchain no noticeable delay between the touch interaction and the reaction of the character can be observed.

The Seaboard is an instrument with a very high degree of input variability. While this makes the Seaboard a superior input device for changing many parameters at the same time, it suffers from other problems. Interaction with the surface can be hard, as the user needs to coordinate his fingers in an unfamiliar way on top of a surface, that doesn't give a direct feedback. As with any instrument the user needs times to familiarize himself with it. This is a process that, depending on the abilities of the user, takes a lot of time. In this project, we further looked at manipulating characters indirectly. It can be expected that using direct manipulation would be even more difficult, as the number of parameters to continuously change would increase. In this respect one can say, that the Seaboard is a suitable device to control animations in real-time, but doesn't offer a revolutionary interface to create them.

7.3 Outlook

This project is a useful first step for many research projects in this field. We have seen, that the Seaboard has big potential in the area of real-time animation controlling and is a very versatile input device.

For further projects, creating new kinds of (learning) mappings could be a rewarding task: this could simplify the interaction and enable the user to control more parameters in a more intuitive way. As the Seaboard is lacking a direct way of feedback, an augmented view for the player could also be very helpful. Extensive user studies would be needed, to be able to make scientific relevant statements about different mappings and to find out, how the Seaboard compares to other methods. Working with experienced animators could further deliver useful insights.

Another area, in which further research would be interesting, is facilitating the process of creating animations with a Seaboard. As it is hard to even picture a detailed animation cognitively, tools like a looper to iteratively create animations or a system, learning from similar animations, would be worth looking into.

Bibliography

ADOBE SYSTEMS, 2017. Character animator. [Online; accessed 25-April-2017].

BAI, Y., KAUFMAN, M. D., LIU, C. K., AND POPOVI?, J. 2016. Artistic-dynamics for 2d animation. In *ACM Transactions on Graphics (SIGGRAPH 2016)*.

BEN SUPPER AND OTHERS, 2015. Midi mpe specifications draft. [Online; accessed 04-May-2017].

BLACK MIRROR, NETFLIX, 2013. Black mirror. [Online; accessed 29-April-2017].

CHIEN, C.-Y., LIANG, R.-H., LIN, L.-F., CHAN, L., AND CHEN, B.-Y. 2015. Flexibend: Enabling interactivity of multi-part, deformable fabrications using single shape-sensing strip. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software &#38; Technology*, ACM, New York, NY, USA, UIST '15, 659–663.

CHOI, B., I RIBERA, R. B., LEWIS, J. P., SEOL, Y., HONG, S., EOM, H., JUNG, S., AND NOH, J. 2016. Sketchimo: Sketch-based motion editing for articulated characters. *ACM Trans. Graph. 35*, 4 (July), 146:1–146:12.

DRAGONBONES, 2017. Dragonbones. [Online; accessed 27-April-2017].

GLAUSER, O., MA, W.-C., PANOZZO, D., JACOBSON, A., HILLIGES, O., AND SORKINE-HORNUNG, O. 2016. Rig animation with a tangible and modular input device. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH) 35*, 4.

ISHIGAKI, S., WHITE, T., ZORDAN, V. B., AND LIU, C. K. 2009. Performance-based control interface for character animation. *ACM Transactions on Graphics (SIGGRAPH) 28*, 3.

JACOBSON, A., PANOZZO, D., GLAUSER, O., PRADALIER, C., HILLIGES, O., AND SORKINE-HORNUNG, O. 2014. Tangible and modular input device for character articulation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH) 33*, 4, 82:1–82:12.

KOMATSU, K. 1988. Human skin model capable of natural shape variation. *The Visual Computer 3*, 5, 265–271.

MESSMER, S., FLEISCHMANN, S., AND SORKINE-HORNUNG, O. 2016. Animato: 2D shape deformation and animation on mobile devices. In *Proceedings of ACM SIGGRAPH ASIA Symposium on Mobile Graphics and Interactive Applications*.

MIDI MANUFACTURERS ASSOCIATION, 2017. Midi specifications. [Online; accessed 04-May-2017].

ROLI LTD, 2017. Juce. [Online; accessed 27-April-2017].

ROLI LTD, 2017. Roli ltd. [Online; accessed 27-April-2017].

ROLI LTD, 2017. Seaboard, 5d touch. [Online; accessed 27-April-2017].

SEFFAH, A., DONYAEE, M., KLINE, R. B., AND PADDA, H. K. 2006. Usability measurement and metrics: A consolidated model. *Software Quality Journal 14*, 2, 159–178.

SEOL, Y., O'SULLIVAN, C., AND LEE, J. 2013. Creature features: Online motion puppetry for non-human characters. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '13, 213–221.

UNITY TECHNOLOGIES, 2017. Unity. [Online; accessed 25-April-2017].

UNITY TECHNOLOGIES, 2017. Unity manual. [Online; accessed 25-April-2017].

UYOUNG CULTURE & MEDIA CO., LTD., 2017. Uyoung culture & media co., ltd. [Online; accessed 27-April-2017].

WANG, L. C. T., AND CHEN, C. C. 1991. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *IEEE Transactions on Robotics and Automation 7*, 4 (Aug), 489–499.

WELMAN, C. 1993. Inverse kinematics and geometric constraints for articulated figure manipulation title of thesis: Inverse kinematics and geometric constraints for articulated fig- ure manipulation.