Live Animation theater: Using Facial tracking and Cue Handling to execute live theater performance with interactive, animated characters.

MED 10 project

AALBORG UNIVERSITET

Department of Architecture,
Design & Media Technology
Medialogy, 10th Semester

Title:

Live Animation theater: Using Facial tracking and Cue Handling to execute live theater performance with interactive, animated characters.

Semester Theme:

Master's Project

Project Period:
MED10

Projectgroup no.:

Members:

Martin Bach Nielsen

Supervisor:
Martin Kraus

Circulation: 2

Number of pages: 59

Number of appendices and form:
2 (DVD and attached documents)

Delivered: 31st of May 2012

Abstract:

In this report the design of two systems for real-time animation theater are documented. A system which translates facial capture data from a 2D domain into a 3D domain and maps it to virtual characters. This system is implemented with noise filtering and a countermeasure to lost frames during theatrical performance.
Another system which handles the triggering of events, or cues, during theatrical performance. This system allows a stage manager to create a set of cues for their play, and execute them in accordance with the tempo of the play. Both of these systems are implemented in the Unity game engine, which is part of the initial conditions for the project. Experiments designed to test these systems are described, but no data has been collected at the time of report hand-in.

PREFACE

On the DVD the reader will find a digital copy of this report as well as the full source code in scripts. An AV production is also present, however it shows very old footage. Due to technical difficulties, more recent footage was lost. A proper AV production will be shown at the project exam presentation.

CONTENTS

CHAPTER 1

INTRODUCTION

Cinema Dell' Arte (henceforth CDA) is a project group of students at The Danish Filmschool in Copenhagen, Denmark. The author was approached by student members of CDA and asked to participate in the development of a new kind of theatrical performance. The play is aimed at children in the ages 5-10 years. It contains many fairytale elements like a prince turned into a frog, a princess, trolls and an evil antagonist. This play separates itself from most other plays by being shown on a projector. Everything in the play is computer generated imagery, and the characters are brought to life by actors in a motion capture stage out of sight of the audience. To capture the facial expressions of the actors, a novel camera rig has been produced such that an actor will always have a close-up camera in front of their face. The actors will also be able to see the audience through a screen in the motion capture stage, allowing them to interact with the audience in real-time. The Danish Film School provides a Motion Capture system similar to that of Aalborg University, of the brand OptiTrack. The play premieres June 13th 2012.

CHAPTER 2

INITIAL VISION

The vision of CDA is to use motion capture as a tool for theater performance. This should allow the audience of the performance to interact with the animated characters on screen. The performance is accompanied by an off-screen actor, who serves as a story teller. He tells the audience a magical tale of princesses, frogs and enchanted forests. As his tale begins, the audience will see the story unfold in his magical mirror, the projection screen, which is part of a big forest set inside one of the studios at The Danish Film School. Behind the scenes, actors will use motion capture suits, and novel headmounted camera rigs, to capture their performance. The data provided by these should then be visualized by the Unity game engine.
By taking the performance to a digital space, the director reaches a degree of freedom not possible by regular theater.

CHAPTER 3

PROBLEM ANALYSIS

CDA has previously used motion capture technology in theatrical performances. The sets were built in Autodesk Motionbuilder, but CDA has expressed the desire to replace this with Unity, the reason being that Motionbuilder does not offer as much flexibility and as many special effects as Unity. This project should then aim to map captured data correctly onto virtual characters in the play. As motion capture data for the body of the characters is already implemented by CDA, this project will instead investigate the possibility of a system which maps data from a webcam tracking the face of the actor onto the face of the virtual character. A system like this should meet some criteria to be considered successful. Since the focus of this project will be to facilitate real-time operation, the following critera are stated:

- each tracked point should be correctly mapped to a corresponding point on the character.
- the mapping procedure should take into account general noise and large errors in measurement by the webcam/tracking software.
- the mapping procedure should be robust in the sense that it should manage issues with low framerate and frame skips by the webcam / tracking software.

With these criteria in mind, the next sections will address a some key factors in constructing this system.

3.1 Level of Detail

The play "Prinsessen og Frøen" is meant for children, and generally has a cartoony, non-photorealistic look. A screenshot of the "Frog" character in the play can be seen in figure 3.1. In order to track the actor's face, a custom software application called Facecat has been developed by CDA, **NOT** the author of this project. This application tracks colored markings on the face of the actor. To ease the mapping of these points to the character, the 3D model should be developed in such a way that the

mechanism controlling mesh deformation has corresponding control points. Figure 3.1 shows the control point layout in the case of the mouth. A total of 17 points is used.

Figure 3.1: A screenshot illustrating the layout of control points for the mouth. The red/white squares indicate a control point, while the circles indicate that each point affects a group of neighbouring vertices

3.2 Transforming 2D data to 3D data

The Facecat application supplies data in a 2D image plane. To transform this to 3D data, inspiration is taken from (Wei et al. 2004), in which a group of students develop a novel system to approach an identical issue. This approach involves converting 2D vectors describing the motion of a tracked point into a rotation around the corresponding 3D point. The Unity scripting API offers methods to construct quaternions from Euler angles. Figure 3.2 shows the correlation between the 2D vector and the 3D rotation in the case of a 5 pixel motion in the y-axis.
By using this technique, there is a risk of extreme rotation in the case of noisy data from the webcam which would result in unnatural mesh deformation. This can be overcome by imposing an upper limit before applying the rotation. In the above case of rotating 5 degrees around the X-axis, the formula for constructing the rotation in the form of quaternion Q can be described as $Q = (w,x,y,z) = (cos(\frac{5}{2}), sin(\frac{5}{2}), 0, 0)$ (Bishop and Werth 2008)

This data transformation will be implemented in Unity and tested with different noise filters - determined by the noise signature from Facecat.

3.3 Cue Management

A second aspect of this project will be the development and evaluation of a system to handle theatrical cues. Cues are events triggered during the performance, in tempo

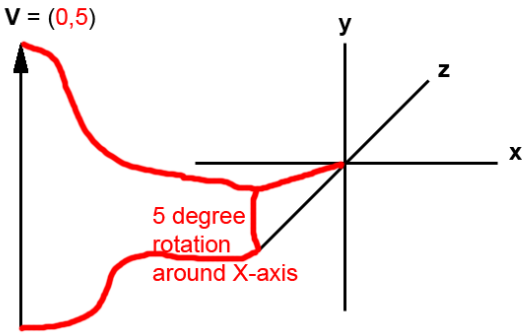

Figure 3.2: A figure illustrating the correlation between pixel motion in the 2D image plane, and control point rotation in 3D space. In this case, we see a 5 pixel motion in the Y-axis being converted to a 5 degree rotation around the x-axis

with the progress of a given scene. A cue could be a piece of music, lighting changes, backdrop switchings etc. These are usually handled by a stage manager. The system (Cue Handler) should allow a stage manager to do the following:

- It should allow a user to build a custom list of cues.
- It should be operated on a remote computer, such that the GUI does not interfere with the computer rendering the theater performance.
- It should allow a user to execute these cues across the network in a planned manner.

This requires scripting with Unity's GUI API. The GUI should integrate with a network connection, such that the user is able to execute cues across a local network. Inspiration for the layout of the Cue Handler is found in the commercial software Qlab by Figure 53 (Figure53 2011). A screenshot of Qlab can be seen in figure 3.3 Figure 3.3 shows some key aspects in the design of the interface, the largest portion of screen space is reserved for a list type view of different cues and information about these cues. The button to execute a cue in the top left corner is big and noticeable, making it hard to misclick. This Cue Handler system is simpler than Qlab, and so the interface design only serves as inspiration. A mockup design of the Cue Handler interface can be seen in figure 3.4. The final design may vary, but will be prototyped from this mockup.

Figure 3.3: The main interface of the commercial software Qlab. Most of the GUI space is used to display a list of cues, and various information about these cues. Image source is http://figure53.com/qlab/

Figure 3.4: A mockup design for the Cue Handler inteface. The screen is divided into 3 parts: A button Control pane, a list type view of cues, and an information pane

CHAPTER 4

PROBLEM STATEMENT

Based on the problem analysis the following problem statement has been phrased:

With the conditions and considerations described in chapter 3, will the described systems succeed in facilitating facial capture data mapping and cue management to a satisfying degree, specified by common evaluation techniques in these fields?

The design and evaluation methods of these systems will be described in the following parts of this report.

CHAPTER 5

SYSTEM DESIGN

In this chapter the following topics will be covered:

- Converting 2D data to 3D data inside Unity
- Filters applied for noise reduction
- Interpolation between frames
- Layout of the Cue Handler system

Implementation details such as code snippets are intentionally left out, however the complete source code for both the facial data processing and the Cue Handler system can be found in the appendix.

5.1 Data conversion

To get an overview of the data flow for facial data, see Figure 5.1.
The Facecat application has a plugin component in the form of a .dll file. This .dll file is compiled with certain attributes which lets the author access it's exposed methods from Unity code. This plugin has methods designed to return data for each relevant tracking point.

Figure 5.2 shows a screenshot of the Facecat application while running. Relevant data points on the face of the actor is being tracked by the color of face paint. The data points are represented as yellow dots with lines inbetween them. These points are saved in the .dll plugin file.

Inside Unity, for each data point, it is possible to extract a 2D vector describing the X and Y position of the point relative to an initial calibration point. A calibration consists of the actor keeping a neutral expression. As an example, the point representing the right eyebrow closest to the center of the actors head is called "RBrowsA", and

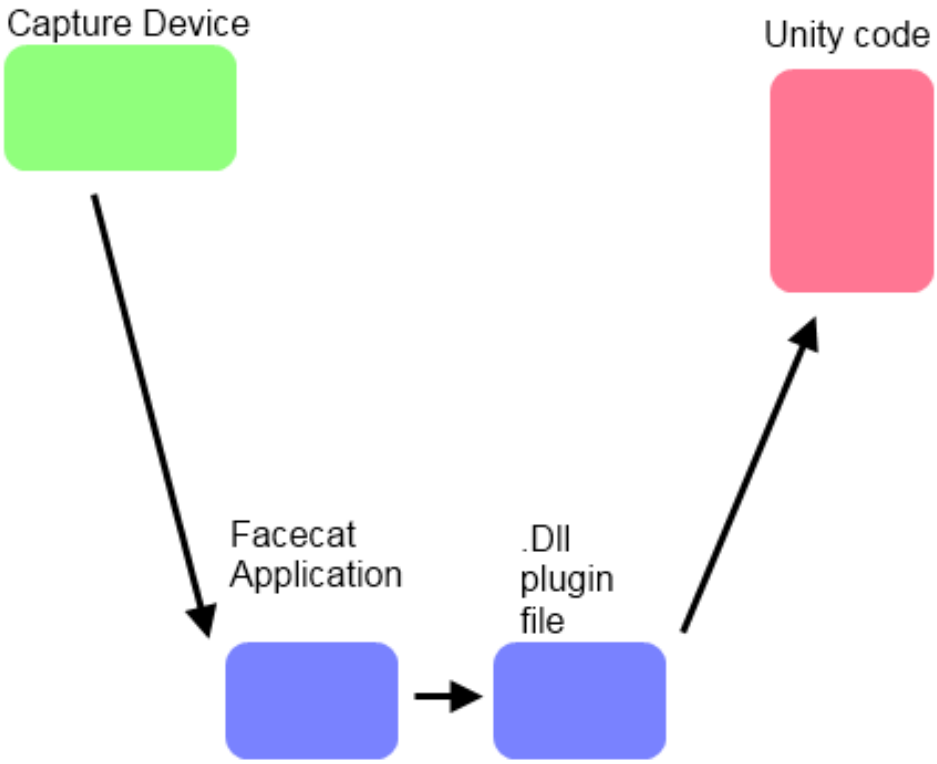

Figure 5.1: The data flow for Facial data. We see the webcam capture device sends a videostream to the Facecat application. Facecat then exposes every relevant tracking point via a custom .dll file. This .dll file is handled by Unity as an external plugin, and exposed methods in this plugin can be called inside Unity.

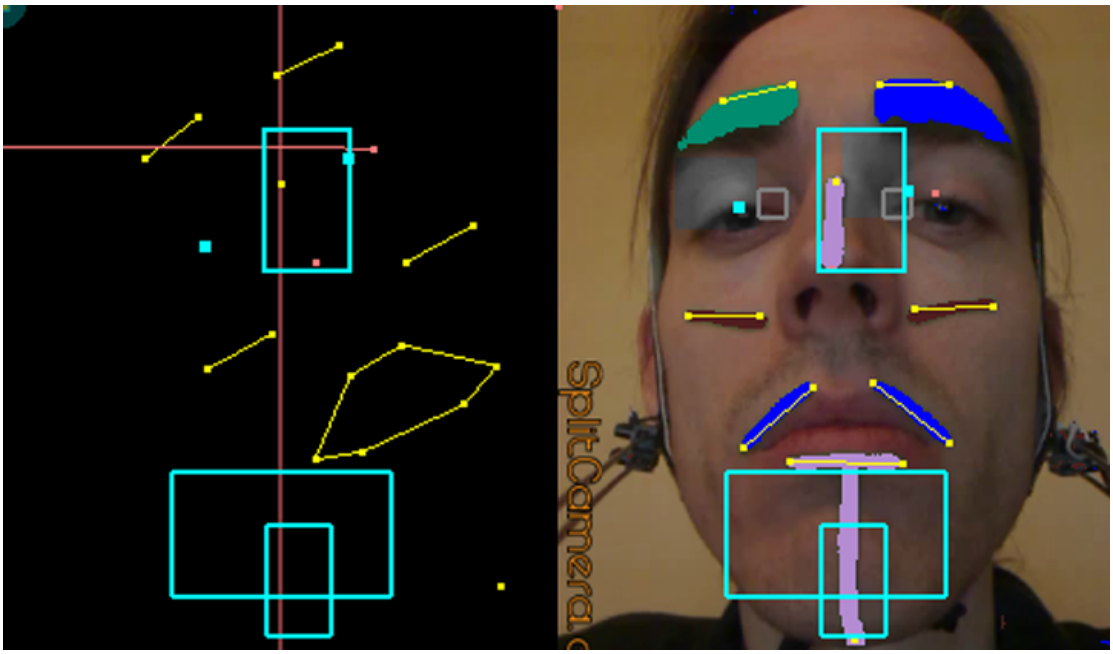

Figure 5.2: A screenshot of Facecat during runtime. The tracked data points are visible by yellow dots connected by lines. Thse dots are saved in the .dll plugin file

represents the deviation in pixels from the initial calibration point for "RBrowsA". The X and Y values can then be stored in a 2D vector, that updates this data each frame. Each 2D vector can then be stored in an array for easy access and easy iteration over each vector in the array.
The main driver of the 2D-3D conversion consists of using the [X,Y] values to construct a quaternion that represents a rotation. The goal is to ensure a deformation of the face mesh which corresponds to the deformation of the actor's face, when expressing themselves. This is ensured by having the 3D model constructed in such a way, that there is a bone in the model for each tracking point on the actor's face. By having the 3D artist rig the model's face in such a manner, many complex issues are avoided already. There is no need for a proprietary mesh deformation system, as most 3D packages already have this system in place with how bones affect mesh. This is a ruleset made by the 3D artist during production of the model. Figure 5.3 shows a character known as "The Frog" in wireframe mode. The red wireframe represents the mesh of The Frog, while the white structure depicts the bones of the character. A similar structure is in every character's face. However, because of the way they are rigged, it is not easy to show in a screenshot. In a consultation between CDA and the author, it was decided to rig every character model's with such a structure. Since each bone controls the deformation of the mesh in areas which correspond to

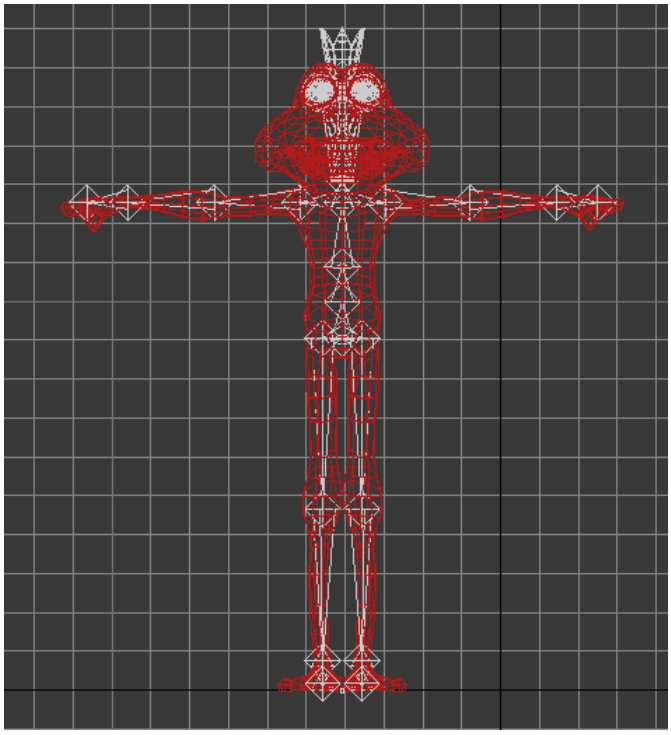

Figure 5.3: A screenshot depicting the bone structure of the Frog character. The white structure shows how bones are rigged for this character. Every 3D model in this project is provided by CDA, not the author

the tracked areas of the actor's face, the displacement of a given tracker point in both X and Y (Horizontal and Vertical) can now be interpreted as a rotation of the bone controlling that area of the mesh. Using this approach, there is an inherent danger of excessive rotation of the bone in the case of mismeasurement by Facecat. This

is solved by limiting both the X and Y values to some range. This range should be variable for each tracking point, as different parts of the face have different ranges of deformation (example - the mouth can move further in the vertical axis than the nose). With this setup it is now possible to construct an algorithm wich can be written in pseudo code seen in listing 5.1.

Listing 5.1: Pseudo code describing the raw mapping algorithm

```
variable : XMax;
Variable: XMin;
Variable YMax;
Variable YMin;
for each tracked point p{
Store each [X,Y] coordinate in Vector2 array.
Limit X and Y to the range given by min and max values.
Quaternion Q = desired rotation constructed by inputting X and Y as
    Euler Angles.
Set rotation of point p to Q
}
```

This represents the data mapping in it's raw form.

5.2 Filters applied for noise reduction

The filters described in this section are meant to reduce noise from the signal to a level deemed acceptable by the CDA Director. The filters alone do not result in completely optimal visual quality, but interpolation between data points described later in section 5.3 will bring the visual standard to an acceptable level. The raw mapping does not take into account the noise in the signal from Facecat. Thus, it is required to filter out noise from the webcam. The noise signature when the actor's face is as much at rest as possible (neutral expression) can bee seen in figure 5.4.

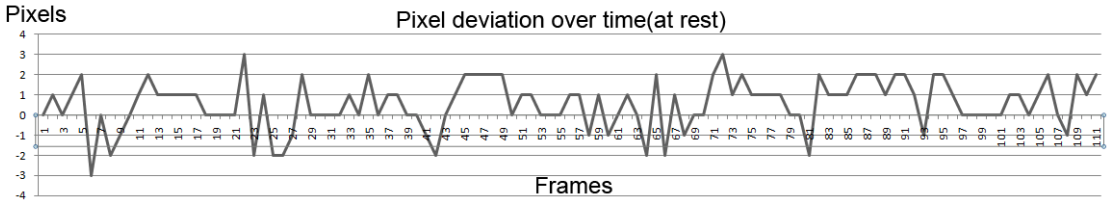

Figure 5.4: A plot of the noise signature when the actor's face holds a neutral expression. As this is considered a baseline, noise filtering is desired

From observing the noise signature it is apparent that a substantial amount of noise can be eliminated by comparing the current pixel data with the pixel data of the previous frame. If this data is beyond a certain threshold, the data is interpreted as a valid facial move and therefore accepted into a new array of filtered data. This threshold should be variable such that a balance between lost data, and a desired visual quality can be found. This filter is described in pseudo code in listing 5.2

Listing 5.2: Pseudo code describing the thresholding algorithm

```
Variable: threshold
for each tracked point P{
store the result of filteredDataP-rawDataP in a tempVector
if(ABS(tempVector.x)> threshold)
      {
            filteredDataP = rawDataP
      }
      Repeat for tempVector.y
}
```

For eye movement, this filtering method shows an undesirable visual quality. By filtering out the low-value deviations, many of the rapid eye movements are lost. Eye movements show different characteristics, they are smaller and quicker than other facial movement. If the thresholding filter is applied here, the eyes will only display movement in the extreme cases, relatively large deviations, and quickly "snap" back to the calibration point, which results in unnatural looking eyes. However, unfiltered eye data results in constantly flickering eye movement. Therefore, a mean filter is applied instead of thresholding. This filter slows down the eye movement slightly, but eliminates the noise which results in flicker eye movement. The mean filter is a simple list, which holds the previous 4 vectors and the current vector. For each frame, the mean value is extracted from this list, and used to set the rotation of the eyes. A pseudo-code description can be seen in listing 5.3.

Listing 5.3: Pseudo code describing the thresholding algorithm

```
Variable: VectorArray;
Variable: EyeFilteredVector;
for each frame{
Update VectorArray with newest tracked vector. Delete oldest vector.
EyeFilteredVector = (Sum of VectorArray)/5
}
```

After all relevant data has been treated by the described filters, it can be applied to the face of the virtual character. As described in Section 5.1, the filtered data is used to construct rotations for the bones in the character rig. All rotations are applied in local space to maintain the proper relationships with the parent bones in the character hierarchy.

5.3 Interpolation

In order to secure robust mapping which should be able to sustain frame loss from Facecat, the rotations are not set directly each frame, which could lead to stuttering in the case of a lost frame. Instead, the script interpolates between the known data points over time. Unity offers SLERP interpolation between quaternions, so this method is used. The `Quaternion.Slerp()` function takes 3 parameters: "From", "To", and "By". "From" is a rotation from which we start interpolating, "To" is the rotation to interpolate towards, and "By" is the amount of interpolation. To make this

interpolation framerate independent,"By" should be `Time.deltaTime` (The time in seconds it took to complete the last frame) multiplied by a speed variable which determines the interpolation rate. As this is variable, the delay introduced by interpolating is also variable. This is entirely up to the CDA director, but in order to establish the upper bound of this delay (maximum interpolation delay before audiences notice) an experiment will be conducted, which will be described in chapter 6

5.4 Cue Handler

As discussed in section 3.3, the Cue Handler should meet certain criteria. These are:

- It should allow a user to custom build a list of cues.
- It should be operated on a remote computer, such that the GUI does not interfere with the computer rendering the theater performance.
- It should allow a user to execute these cues across the network.

This section will focus on describing how the Cue Handler has been designed to meet these criterion.
To enable remote cue execution, the Cue Handler has a server/client system. The machine which holds the built cues acts as a server, and the machine to recieve these cues acts as a connecting client. Unity allows for easy connection with the special component type "NetvorkView". Anything holding this component on both the server and client side, will be able to communicate via Remote Procedure Calls(RPC's). Any method that is prefaced with the `[RPC]` attribute is open to being invoked remotely (UnityTechnologies 2011). By having an RPC method on the client side, which expects to recieve a cue when called, it is possible to send cue data once a connection has been established. The server part of the Cue Handler will instantly initiate a running server. The client side is designed to connect to this server whenever the recieving machine is ready to accept cues(when the performance is ready to start).

The Cue Handler main interface can be seen in figure 5.5. The different areas of the interface are:

- 1: Management buttons. These are used to make new cues, edit existing cues, save and load functionality for different cue templates etc.
- 2: Buttons to execute a cue, delete a cue, or quick-switch to a given scene. This quick-switch functionality is intended for rehearsal runs.
- 3: The main cue list. This shows all the created cues, by the name that was given to them during creation.
- 4: Textbox displaying the text description of the selected cue. This description is entered by the user during creation of the cue.

- 5: A "Log out" button which will terminate the server functionality of the Cue Handler. This also displays the number of connections. During performance, this number should always be 1.

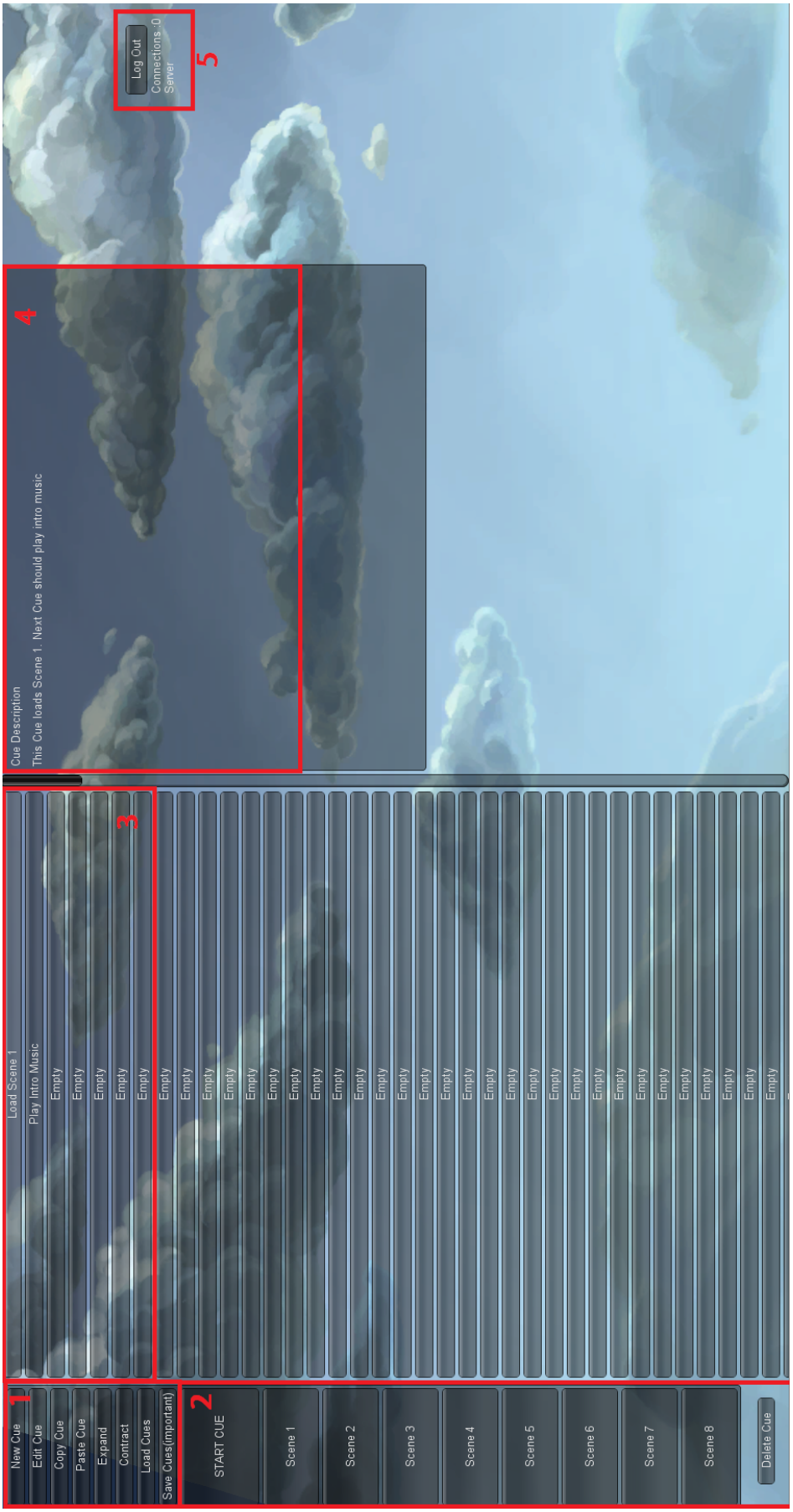

Figure 5.5: A screenshot showing the Cue Handler main interface. This is what the stage manager will use during performance

The first thing the user needs to do, is populate the cue list with the cues for the performance. By clicking the "New Cue" button, a dialog box with creation parameters is rendered. This dialog box can be seen in figure 5.6

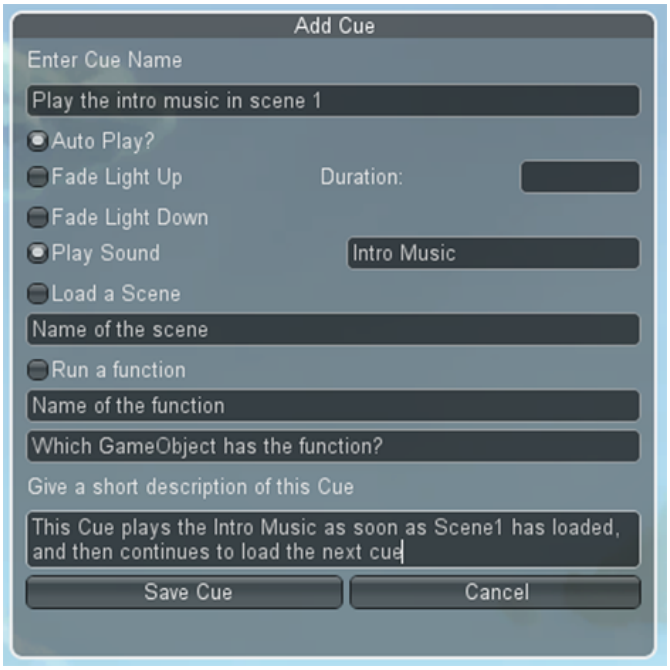

Figure 5.6: A screenshot of the dialog box shown when the stage manager wishes to add a cue. Multiple parameters can be set, and finally saved to the cue list

In this dialog box, the user enters parameters specific for this cue. Most functionality can be executed by simply using the "Run a function" fields. This will call any given function upon cue execution, provided that this function exists, and contains code for the desired functionality. This is the most dynamic parameter of the cue. Other functionality which has been deemed "common use" by CDA is presented in the top half of the dialog box for easy access. Examples are: fading lights up and down, playing a soundfile, or loading a scene. The user may have a need to create a cue which triggers multiple custom events at once using the "Run a function" field. It is possible to let the user run multiple functions within the same cue, however this approach does not scale very well. In theory, the user could find the need to execute n functions at once. So, instead of allowing the user to do so, the "Auto Play" feature has been implemented. With this toggle checked, the cue will automatically continue to the next cue upon execution. So, to run x custom functions, the user will need to create x cues that all have "Auto Play" checked. This will execute these cues in rapid succesion, and the Cue Handler will stop executing cues when it encounters a cue without "Auto Run" checked.

5.5 The Cue Class

When a cue is saved, a custom class named Cue is instantiated. This class serves as a data holder for a cue event, and has public properties to store information about what events this cue will trigger. After instantiation, it's properties are set according to the parameters the user have entered, and this instance is saved in an array at the selected index of the cue list. A class diagram of the Cue class can be seen in figure 5.7

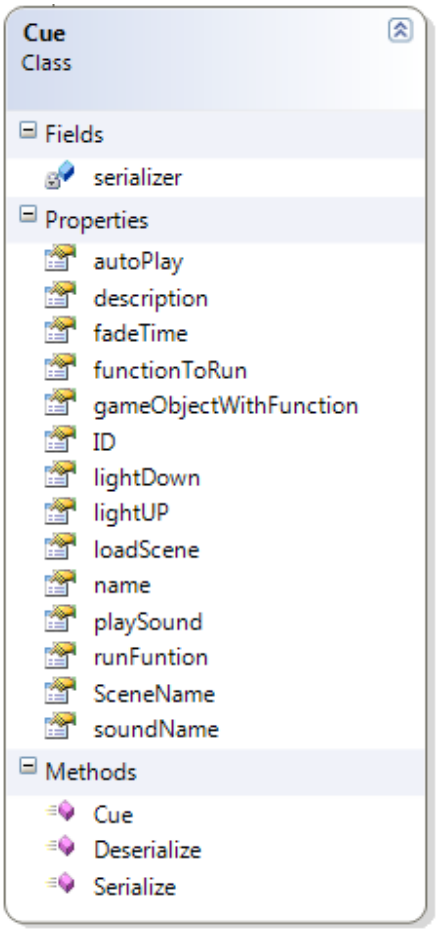

Figure 5.7: A class diagram of the Cue class. Generated in Microsoft Visual Studio 2010

As seen here, the Cue class has two methods other than its constructor, `Serialize()` and `Deserialize()`. These methods serve the purpose of serializing and deserializing the Cue data into a string format and returns this string. This is used when sending and recieving the cue data across the network. When the user clicks the "Start Cue" button, an RPC method on the recieving machine is called. This method deserializes the string, which represents the Cue, and appropriate action is taken based on the Cue parameters (playing a sound, instantiating a particle system, etc.). The "Edit Cue" button will bring up the same dialog box, but in this case, it will observe the properties of the selected cue, and load these into the text fields of the dialog. The "Expand" button will insert a new empty cue at the selected index, and

push every following cue one slot. The "Contract" button removes the selected cue, and contracts the list to cover the empty slot. As seen in figure 5.5, the Cue Handler also supports Copy/Paste functionality.
When done with a cue list, the user needs to click "save cues", in order for the list to appear next time the program is loaded. The save button will write the cue list to an XML file saved locally. This file can be copied remotely to share cue templates between multiple computers if desired. The save feature should execute as a standard feature, when closing the program by clicking the "X" in the window titlebar, however Unity displays strange behaviour here, and will not complete writing to the XML file in time, resulting in XML corruption. This concludes the design chapter, and the following chapter will discuss the methods for evaluating these two systems.

CHAPTER 6

EXPERIMENT OVERVIEW

This chapter describes the experiments intended to evaluate the two parts of this project in regards to the problem statement in chapter 4. The author of this project has not been able to carry out the experiments at the time of writing. The software written to handle facial data mapping and cue execution is complete, however other parts of the CDA project are not yet at a stage where the authors software can be tested. The Cue Handler system has not yet been used by CDA to an extent where test feedback is to be considered valuable. The experiments described in this chapter will be carried out, and results will be presented and discussed at the project exam.

To evaluate the biggest issue with the facial data mapping described in Chapter 5, is the inherent lag introduced by interpolating between data points over time. As previously mentioned, this was introduced to secure robustness in the cases where Facecat skips a frame. This could lead to the image trailing behind the audio, which is supplied by a microphone on the actor. Research in this field by Conrey & Pisoni suggests that a maximum likelihood of an observer giving an "in sync" response is found when the video is leading the audio by approximately 40ms (Conrey and Pisoni 2003). This suggests the neccesity to delay audio signals in order to keep audio and video synched during performance. CDA is prepared to implement this, since other systems such as motion capture data processing, and the general rendering of the visually complex scenes are already causing a significant amount of lag during performance. However, to test this system as a separate entity, it will be tested without external lag sources like render-heavy scenes. Two versions of the face mapping process will be constructed. One version with interpolation, and one version without interpolation. These two versions will be used in an ABX test, to determine if the viewer can distinguish the lag introduced by interpolating. The ABX test works by displaying a known sample A and a known sample B, followed by an unknown sample X, to a test subject and asking the subject to identify sample X as either A or B. In order to ensure that the subject's statement has some degree of confidence, several trials must be run. A confidence level of 95% is considered statistically significant by the author(this leaves a 5% chance that they are guessing correctly), and is obtained

if the subject answers correctly in 9 out of 10 trials. If the subjects cannot correctly identify the X sample in 9 of 10 cases, the null-hypothesis is rejected, and it cannot be proven that there is a discernable difference between the two versions
The test will be based on showing video on a dual screen setup. The two screens are back to back, and the test facilitator will control which video is shown as sample X.

To evaluate the Cue Handler interface, it is decided to perform a usability test which relies on self-reported data by the user. The best way to capture self-reported data is to ask the user to rate certain tasks on a scale (Tullis and Albert 2008). To avoid social desirability bias (Nancarrow and I. 2000) in the self-reported data, the survey is done remotely via a questionnaire. To further avoid any bias, the author has asked not to know the identity of the user from CDA beforehand (Tullis and Albert 2008). This questionnaire will ask the user to rate a set of tasks on a Likert scale. It is common to analyze the Likert results by assigning a numeric value to each point on the scale and averaging these values (Tullis and Albert 2008). In the case of "Strongly disagree" to "Strongly agree", they will range from 1-5. By asking the user to rate the tasks in terms of ease, problem areas that need improvement should become clear. The questionnaire can be found in the appendix of this report.

CHAPTER 7

DISCUSSION

In this chapter the work conducted in relation to this report will be discussed. Two different systems were designed and implemented as described in Chapter 5.
Methods of evaluating these systems has been presented, however due to timing, these evaluations have not been performed at the time of writing. Preliminary observation of the systems running show that the facial data mapping method yields a good result in terms of delivering a correct mapping of data points. It is still unknown if the lag introduced by interpolation is noticeable. The author of the report has successfully created and executed a complete template of cues with the Cue Handler. This means that the core functionality of the system is working correctly, however the usability of the interface is still to be determined. At the project exam, the results will be presented and discussed in relation to the problem statement.

BIBLIOGRAPHY

Bishop, L. and J. V. Werth (2008). *Essential mathematics for games and interactive applications*. Morgan Kaufmann.

Conrey, B. L. and D. B. Pisoni (2003, September). Audiovisual asynchrony detection for speech and nonspeech signals. *AVSP 2003 - Internation Conference on Audio-Visual Speech Processing*, 5–5.

Figure53 (2011). Figure53.com. `http://figure53.com/qlab/`. Accessed May 30, 2012.

Nancarrow, C. and B. I. (2000). Saying the "right thing": Coping with social desirability bias in marketing research. *Bristol Business School Teaching and Research Review Issue 3, Summer 2000, ISSN 1468-4578*. Accessed May 30, 2012.

Tullis, T. and W. Albert (2008). *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Morgan Kaufmann.

UnityTechnologies (2011). Unity3d.com. `http://unity3d.com/support/documentation/Components/net-RPCDetails/`. Accessed May 30, 2012.

Wei, X., Z. Zhu, L. Yin, and Q. Ji (2004). A real time face tracking and animation system. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 3–4.

APPENDIX A

APPENDIX

Appendix B

Questionnaire

				Please check one answer for each statement in the left side		
		Strongly disagree	Disagree	Neither disagree nor agree	Agree	Strongly agree
Creating a new cue was easy						
Editing an existing cue was easy						
Deleting a cue was easy						
Moving a cue was easy						
Starting a cue was easy						
The "Run a Function" field was helpful						
Loading a scene was easy						
I understand how to use "Auto Play"						

Figure B.1

APPENDIX C

FACIAL DATA MAPPING SOURCE CODE

Listing C.1: The Facecat script, which should be on every characters face GameObject for data mapping to occur

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System;
using System.Runtime.InteropServices;


public class FaceCat : MonoBehaviour {

      public int port = 6832;
      public int status = -1234;
      public string capturename = "frog";
      public string seperator = ":";
      public int debuge = 0;
      public int debugx = 0;
      public int debugy = 0;
      public static Vector2[] rawData;
      public static Vector2[] filteredData;
   public bool debugMode;

      public Transform RBrowsA;
      public Transform RBrowsB;
      public Transform LBrowsA;
      public Transform LBrowsB;
      public Transform RCheeks;
      public Transform LCheeks;
      public Transform LNasal;
      public Transform RNasal;
      public Transform LMouthU;
      public Transform RMouthU;
      public Transform LMouthC;
```

```
32      public Transform RMouthC;
33      public Transform LMouthB;
34      public Transform RMouthB;
35      public Transform Jaw;
36      public Transform LEyeBall;
37      public Transform REyeBall;
38      public Transform LLidT;
39      public Transform RLidT;
40      public Transform LLidB;
41      public Transform RLidB;
42      public float mouthSmoothness = 18.0f;
43      public float cheekSmoothness = 5.0f;
44      public float browSmoothness = 18.0f;
45      public float nasalSmoothness = 5.0f;
46  public float eyeSmoothness = 18.0f;
47      bool setRotations = false;
48      int filterThreshold;
49      Vector2 tempVector;
50  public float eyeXBias;
51  public float eyeYBias;
52  Vector2[] rawEye;
53
54  //Variables for adjusting min-max values for different face parts
        to solve clipping. Extremely silly way of doing this. Find
        clever solution
55
56  public float LMouthUMin;
57  public float LMouthUMax;
58  public float RMouthUMin;
59  public float RMouthUMax;
60  public float LMouthCMin;
61  public float LMouthCMax;
62  public float RMouthCMin;
63  public float RMouthCMax;
64
65      //DLL import calls for ServerClient.dll external methods
66      [DllImport ("ServerClient")]
67            private static extern int ServerNetwork(int port);
68      [DllImport ("ServerClient")]
69            private static extern int CloseNetwork();
70      [DllImport ("ServerClient")]
71            public static extern int getX(string name, string tag);
72      [DllImport ("ServerClient")]
73            public static extern int getY(string name, string tag);
74      [DllImport ("ServerClient")]
75            private static extern int test(string test);
76
77
```

```
78
79
80      // Use this for initialization
81      void Start () {
82            filterThreshold = 3;
83
84            rawData = new Vector2[17];
85            filteredData = new Vector2[17];
86      rawEye = new Vector2[10];
87            status = ServerNetwork(port);
88
89      //Assign our transform variables
90            RBrowsA = transform.Find(capturename+seperator+"RBrowsA");
91            RBrowsB = transform.Find(capturename+seperator+"RBrowsB");
92            LBrowsA = transform.Find(capturename+seperator+"LBrowsA");
93            LBrowsB = transform.Find(capturename+seperator+"LBrowsB");
94            RCheeks = transform.Find(capturename+seperator+"RCheeks");
95            LCheeks = transform.Find(capturename+seperator+"LCheeks");
96            LNasal = transform.Find(capturename+seperator+"LNasal");
97            RNasal = transform.Find(capturename+seperator+"RNasal");
98            LMouthU = transform.Find(capturename+seperator+"LMouthU");
99            RMouthU = transform.Find(capturename+seperator+"RMouthU");
100           LMouthC = transform.Find(capturename+seperator+"LMouthC");
101           RMouthC = transform.Find(capturename+seperator+"RMouthC");
102           LMouthB = transform.Find(capturename+seperator+"LMouthB");
103           RMouthB = transform.Find(capturename+seperator+"RMouthB");
104           Jaw   = transform.Find(capturename+seperator+"Jaw");
105           LEyeBall =
                  GameObject.Find(capturename+seperator+"Leye").transform;
106           REyeBall =
                  GameObject.Find(capturename+seperator+"Reye").transform;
107           LLidT = transform.Find(capturename+seperator+"LLidT");
108           RLidT = transform.Find(capturename+seperator+"RLidT");
109           LLidB = transform.Find(capturename+seperator+"LLidB");
110           RLidB = transform.Find(capturename+seperator+"RLidB");
111
112     }
113
114     // Update is called once per frame
115     void Update () {
116
117           //populate an array with raw data from FaceCat. If
                  different regions of the face should have different
                  noise thresholds, this array should be split up into
                  regional arrays instead.
118
119           rawData[0] = (new
                  Vector2(getX(capturename,"RBrowsA"),getY(capturename,"RBrowsA")));
```

```
120             rawData[1] = (new
                   Vector2(getX(capturename,"RBrowsB"),getY(capturename,"RBrowsB")));
121             rawData[2] = (new
                   Vector2(getX(capturename,"LBrowsA"),getY(capturename,"LBrowsA")));
122             rawData[3] = (new
                   Vector2(getX(capturename,"LBrowsB"),getY(capturename,"LBrowsB")));
123             rawData[4] = (new
                   Vector2(getX(capturename,"RCheeks"),getY(capturename,"RCheeks")));
124             rawData[5] = (new
                   Vector2(getX(capturename,"LCheeks"),getY(capturename,"LCheeks")));
125             rawData[6] = (new
                   Vector2(getX(capturename,"LNasal"),getY(capturename,"LNasal")));
126             rawData[7] = (new
                   Vector2(getX(capturename,"RNasal"),getY(capturename,"RNasal")));
127             rawData[8] = (new
                   Vector2(getX(capturename,"LMouthU"),getY(capturename,"LMouthU")));
128             rawData[9] = (new
                   Vector2(getX(capturename,"RMouthU"),getY(capturename,"RMouthU")));
129             rawData[10] = (new
                   Vector2(getX(capturename,"RMouthC"),getY(capturename,"RMouthC")));
130             rawData[11] = (new
                   Vector2(getX(capturename,"LMouthC"),getY(capturename,"LMouthC")));
131             rawData[12] = (new
                   Vector2(getX(capturename,"LMouthB"),getY(capturename,"LMouthB")));
132             rawData[13] = (new
                   Vector2(getX(capturename,"RMouthB"),getY(capturename,"RMouthB")));
133             rawData[14] = (new
                   Vector2(getX(capturename,"Jaw"),getY(capturename,"Jaw")));
134         rawData[15] = (new Vector2(getX(capturename, "REye"),
               getY(capturename, "REye")));
135         rawData[16] = (new Vector2(getX(capturename, "LEye"),
               getY(capturename, "LEye")));
136
137
138         //Populate an array with raw data for eyes
139         rawEye[0] = (new Vector2(getX(capturename, "REye"),
               getY(capturename, "REye")));
140         rawEye[1] = (new Vector2(getX(capturename, "REye"),
               getY(capturename, "REye")));
141         rawEye[2] = (new Vector2(getX(capturename, "REye"),
               getY(capturename, "REye")));
142         rawEye[3] = (new Vector2(getX(capturename, "REye"),
               getY(capturename, "REye")));
143         rawEye[4] = (new Vector2(getX(capturename, "REye"),
               getY(capturename, "REye")));
144         rawEye[5] = (new Vector2(getX(capturename, "LEye"),
               getY(capturename, "LEye")));
```

```
145         rawEye[6] = (new Vector2(getX(capturename, "LEye"),
                getY(capturename, "LEye")));
146         rawEye[7] = (new Vector2(getX(capturename, "LEye"),
                getY(capturename, "LEye")));
147         rawEye[8] = (new Vector2(getX(capturename, "LEye"),
                getY(capturename, "LEye")));
148         rawEye[9] = (new Vector2(getX(capturename, "LEye"),
                getY(capturename, "LEye")));
149
150         //Create new Vectors that hold mean-filtered data for the eyes.
                Thresholding isnt viable here as it makes the eye movement
                look very unnatural
151         //Vector2 rightEyeFiltered = (rawEye[0] + rawEye[1] + rawEye[2]
                + rawEye[3] + rawEye[4]) / 5;
152         Vector2 leftEyeFiltered = (rawEye[5] + rawEye[6] + rawEye[7] +
                rawEye[8] + rawEye[9]) / 5;
153
154
155
156
157             //We loop through the array of raw data, and if the
                    difference from past frame is greater than Threshold,
                    put it in filteredData[]
158             // which we use to assign final rotations to the face.
                    Basic noisefilter by thresholding
159             for (int i = 0; i < rawData.Length-1; i++) {
160                   int tempXInt = 0;
161                   int tempYInt = 0;
162
163                   Vector2 tempResult = filteredData[i]-rawData[i];
164
165                   if (Mathf.Abs(tempResult.x) > filterThreshold) {
166                         tempXInt = (int)tempResult.x;
167                   }
168                   if (Mathf.Abs(tempResult.y) > filterThreshold) {
169                         tempYInt = (int)tempResult.y;
170
171                   }
172                   filteredData[i] = new Vector2(tempXInt,tempYInt);
173
174             }
175
176         // Check if Facecat numbers are being funky before mapping
                rotations to the characters face. They sometimes are. Dont
                know why
177         if (getY(capturename, "RBrowsA") > -50 && (getY(capturename,
                "RBrowsA")) < 50)
178         {
```

```
179                   setRotations = true;
180
181               }
182
183                   if (setRotations) {
184
185
186
187         //Set rotation and rotational smoothing for all the bones
               found. Smoothing is variable for different parts of the face
188
189               RBrowsA.localRotation =
                     Quaternion.Slerp(RBrowsA.localRotation,
                     Quaternion.Euler(-filteredData[0].y,0,filteredData[0].x),
190               Time.deltaTime*browSmoothness);
191
192               RBrowsB.localRotation =
                     Quaternion.Slerp(RBrowsB.localRotation,
                     Quaternion.Euler(-filteredData[1].y,0,filteredData[1].x),
193               Time.deltaTime*browSmoothness);
194
195               LBrowsA.localRotation =
                     Quaternion.Slerp(LBrowsA.localRotation,
                     Quaternion.Euler(-filteredData[2].y,0,filteredData[2].x),
196               Time.deltaTime*browSmoothness);
197
198               LBrowsB.localRotation =
                     Quaternion.Slerp(LBrowsB.localRotation,
                     Quaternion.Euler(-filteredData[3].y,0,filteredData[3].x),
199               Time.deltaTime*browSmoothness);
200
201               RCheeks.localRotation =
                     Quaternion.Slerp(RCheeks.localRotation,
                     Quaternion.Euler(-filteredData[4].y,0,filteredData[4].x),
202               Time.deltaTime*cheekSmoothness);
203
204               LCheeks.localRotation =
                     Quaternion.Slerp(LCheeks.localRotation,
                     Quaternion.Euler(-filteredData[5].y,0,filteredData[5].x),
205               Time.deltaTime*cheekSmoothness);
206
207               LNasal.localRotation =
                     Quaternion.Slerp(LNasal.localRotation,
                     Quaternion.Euler(-filteredData[6].y*0.2f,0,
208               filteredData[6].x*0.2f),Time.deltaTime*nasalSmoothness);
209
210               RNasal.localRotation =
                     Quaternion.Slerp(RNasal.localRotation,
```

```
211             Quaternion.Euler(-filteredData[7].y*0.2f,0
            ,filteredData[7].x*0.2f),Time.deltaTime*nasalSmoothness);
212
213         LMouthU.localRotation =
                Quaternion.Slerp(LMouthU.localRotation,
                Quaternion.Euler(-filteredData[8].y*0.5f,0,
214         filteredData[8].x),Time.deltaTime*mouthSmoothness);
215
216         RMouthU.localRotation =
                Quaternion.Slerp(RMouthU.localRotation,
                Quaternion.Euler(-filteredData[9].y*0.5f,0,
217         filteredData[9].x),Time.deltaTime*mouthSmoothness);
218
219         LMouthC.localRotation =
                Quaternion.Slerp(LMouthC.localRotation,
                Quaternion.Euler(-filteredData[10].y,0,
220         filteredData[10].x),Time.deltaTime*mouthSmoothness);
221
222         RMouthC.localRotation =
                Quaternion.Slerp(RMouthC.localRotation,
                Quaternion.Euler(-filteredData[11].y,0,
223         filteredData[11].x),Time.deltaTime*mouthSmoothness);
224
225         LMouthB.localRotation =
                Quaternion.Slerp(LMouthB.localRotation,
                Quaternion.Euler(-filteredData[12].y,0,
226         filteredData[12].x),Time.deltaTime*mouthSmoothness);
227
228         RMouthB.localRotation =
                Quaternion.Slerp(RMouthB.localRotation,
                Quaternion.Euler(-filteredData[13].y,0
229         ,filteredData[13].x),Time.deltaTime*mouthSmoothness);
230
231         Jaw.localRotation = Quaternion.Slerp(Jaw.localRotation,
                Quaternion.Euler(rawData[14].y*0.7f,0,rawData[14].x)
232         ,Time.deltaTime*mouthSmoothness);
233
234      //Applying eyeball rotations from filtered eyeball data. The
            eyeBias variables are here to make up for eyeball
            transforms having differing pivot orientations in the model
            file.
235
236      REyeBall.localRotation =
            Quaternion.Slerp(REyeBall.localRotation,
            Quaternion.Euler(leftEyeFiltered.y
237      * 1.4f - eyeXBias, 0, (-leftEyeFiltered.x*1.4f) - eyeYBias),
            Time.deltaTime * eyeSmoothness);
238
```

```
239         LEyeBall.localRotation =
                Quaternion.Slerp(LEyeBall.localRotation,
                Quaternion.Euler(leftEyeFiltered.y*
240         1.4f - eyeXBias, 0, (-leftEyeFiltered.x*1.4f) - eyeYBias),
                Time.deltaTime * eyeSmoothness);
241
242
243                     //Clear filtered data. Very important.....apparently
244
245                     for (int i = 0; i < filteredData.Length-1; i++) {
246
247                           filteredData[i] = Vector2.zero;
248
249                     }
250
251
252                 }
253         }
254
255     void OnGUI()
256     {
257
258         if (debugMode)
259         {
260
261
262             GUI.Label(new Rect(Screen.width / 2, Screen.height / 10,
                    200, 80), "Sliders determine responsiveness\nEyes
                    require a good Facecat calibration, but can be
                    offset/biased in script");
263             GUI.Label(new Rect(10, 120, 200, 20), "Mouth");
264             GUI.Label(new Rect(65, 120, 200, 20), "Cheek");
265             GUI.Label(new Rect(115, 120, 200, 20), "Brow");
266             GUI.Label(new Rect(160, 120, 200, 20), "Nose");
267             GUI.Label(new Rect(195, 120, 200, 20), "Eyes");
268             mouthSmoothness = GUI.VerticalSlider(new Rect(25, 150, 20,
                    300), mouthSmoothness, 20.0f, 0.1f);
269             cheekSmoothness = GUI.VerticalSlider(new Rect(75, 150, 20,
                    300), cheekSmoothness, 20.0f, 0.1f);
270             browSmoothness = GUI.VerticalSlider(new Rect(125, 150, 20,
                    300), browSmoothness, 20.0f, 0.1f);
271             nasalSmoothness = GUI.VerticalSlider(new Rect(175, 150, 20,
                    300), nasalSmoothness, 20.0f, 0.1f);
272             eyeSmoothness = GUI.VerticalSlider(new Rect(200, 150, 20,
                    300), eyeSmoothness, 20.0f, 0.1f);
273             GUI.Label(new Rect(10, 480, 50, 55),
                    Convert.ToString(Mathf.Round(mouthSmoothness * 100) /
                    100));
```

```
274         GUI.Label(new Rect(65, 480, 50, 55),
               Convert.ToString(Mathf.Round(cheekSmoothness * 100) /
               100));
275         GUI.Label(new Rect(115, 480, 50, 55),
               Convert.ToString(Mathf.Round(browSmoothness * 100) /
               100));
276         GUI.Label(new Rect(175, 480, 50, 55),
               Convert.ToString(Mathf.Round(nasalSmoothness * 100) /
               100));
277
278         if (GUI.Button(new Rect(80, 70, 80, 20), "+ Threshold"))
279         {
280            filterThreshold++;
281
282         }
283         if (GUI.Button(new Rect(80, 90, 80, 20), "- Threshold"))
284         {
285            filterThreshold--;
286
287         }
288
289
290
291         if (GUI.Button(new Rect(10, 70, 50, 30),
               Convert.ToString(filterThreshold)))
292         {
293            Debug.Log("Call exit");
294            CloseNetwork();
295            Application.Quit();
296
297         }
298      }
299   }
300 }
```

APPENDIX D

CUE HANDLER SOURCE CODE

Listing D.1: The Cue Handler GUI code

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System;
using System.IO;
using System.Xml.Serialization;
using System.Xml;

public class CueGUI : MonoBehaviour {
   Vector2 scrollPosition;
   public string cueName;
   //List<Cue> cueList = new List<Cue>();

   int selGridInt = 0;
   public static List<string> stringList = new List<string>();
   string[] selStrings = new string[400];
   int counter = 0;
   bool showNewCueWindow = false;
   bool showEditCueWindow = false;
   bool fadeLightUp;
   bool fadeLightDown;
   string fadeTime = "";
   bool playSound;
   bool loadScene;
   string sceneName = "Name of the scene";
   bool runFunction;
   string GO = "Which GameObject has the function?";
   string functionName = "Name of the function";
   string soundName = "name of soundclip";
   string description = "";
   public Rect newCueRect = new Rect(1000,800,400,400);
```

```
32    public Rect editCueRect = new Rect(1000,800,400,400);
33    Cue[] cueArray = new Cue[400];
34       Cue cueCopy;
35    bool firstCueDone = false; //never used?
36    bool auto;
37    GUIText gt;
38    float result;
39    public String fileName = ".\\savedCues.xml";
40    FileStream ofs;
41    string activeCueDesc = "";
42    int checkInt = 0;
43    Cue[] tempArray = new Cue[400];
44    bool cueArrayAvailable = true;
45    Cue cueToEdit = new Cue();
46    bool playMode;
47       bool advance;
48
49       Cue defaultCue;
50
51    void Start()
52    {
53
54             //if (Application.isEditor) {
55       Debugger.activateLogCallback();
56             //}
57
58       defaultCue = new Cue();
59       defaultCue.name = "Empty";
60             defaultCue.SceneName = "";
61             defaultCue.soundName = "";
62
63       for (int i = 0; i < cueArray.Length - 1; i++)
64       {
65          cueArray[i] = defaultCue;
66       }
67       //gt = GameObject.Find("debugText").guiText;
68
69             if (firstCueDone) print (firstCueDone); //De.B.S. warnings
70
71    }
72
73
74    void Update()
75    {
76             if (Input.GetKeyDown(KeyCode.Backspace)) {
77                   selGridInt--;
78                   if (selGridInt<0) selGridInt = 0;
79             }
```

```
80
81              if (Input.GetKeyDown(KeyCode.Space)) {
82                    playMode=true;
83                    if (!Input.GetKey(KeyCode.LeftControl))
84                          advance=true;
85              }
86
87
88        if (playMode)
89        {
90                    if (Network.isServer) {
91              networkView.RPC("sendCue", RPCMode.Others,
                    cueArray[selGridInt].Serialize());
92                    } else {
93                          Debug.Log ("Trying to send, but no connection
                              was started!");
94
95                    }
96
97                    if (advance || cueArray[selGridInt].autoPlay) {
98
99                          selGridInt++;
100                         advance = false;
101                   }
102                   else
103                   {
104                         playMode = false;
105                   }
106
107       }
108       //Code that handles updating the description area if the user
              is clicking around in the cue list
109       if (selGridInt != checkInt)
110       {
111                   selGridInt =
                        Mathf.Clamp(selGridInt,0,cueArray.Length-1);
112
113          //print("clicked: " + selGridInt);
114          if (cueArray[selGridInt] != null)
115          {
116             activeCueDesc = cueArray[selGridInt].description;
117          }
118
119          for (int i = 0; i < cueArray.Length-1; i++)
120          {
121             try
122             {
123                selStrings[i] = cueArray[i].name;
```

```
124             }
125             catch (Exception e)
126             {
127                Debug.Log(e.Message);
128 
129             }
130 
131 
132          }
133                   //selGridInt++; //select text cue
134          checkInt = selGridInt;
135 
136       }
137 
138    }
139 
140 
141 
142    void loadCues()
143    {
144       //First, we make new default cues and to load into the array
145       Cue defaultCue = new Cue();
146       defaultCue.name = "Empty";
147       cueArrayAvailable = false;
148       for (int i = 0; i < cueArray.Length-1; i++)
149       {
150          cueArray[i] = defaultCue;
151       }
152 
153       // Make a new serialiser, that deserializes the xml file into
             the cueArray, and sets counters and bools as if the user
             entered the cues manually
154       ofs = new FileStream(fileName, FileMode.OpenOrCreate,
             FileAccess.ReadWrite);
155       XmlSerializer xs = new XmlSerializer(typeof(Cue[]));
156       tempArray = (Cue[])xs.Deserialize(ofs);
157       ofs.Close();
158       //Check if the current index is occupied by a default Cue. If
             it isnt, then load that into the CueArray and update the
             counter to reflect where the next cue should be inserted
159       for (int i = 0; i < tempArray.Length-1; i++)
160       {
161          if (tempArray[i].name != "Empty")
162          {
163             cueArray[i] = tempArray[i];
164             counter++;
165 
166          }
```

```
167         }
168         firstCueDone = true;
169         cueArrayAvailable = true;
170 
171         for (int i = 0; i < cueArray.Length-1; i++)
172         {
173            try
174            {
175               selStrings[i] = cueArray[i].name;
176 
177            }
178            catch (Exception e)
179            {
180 
181               Debug.Log(e.Message);
182            }
183 
184         }
185 
186      }
187 
188         void updateselStrings () {
189 
190         for (int i = 0; i < cueArray.Length-1; i++)
191         {
192 
193            selStrings[i] = cueArray[i].name;
194 
195         }
196 
197         }
198 
199 
200      void newCueWindow(int windowID)
201      {
202         //Start new Vectical group
203         GUILayout.BeginVertical();
204         //Ask for Cue name
205         GUILayout.Label("Enter Cue Name");
206         //Hold Cue Name
207         cueName = GUILayout.TextField(cueName);
208         auto = GUILayout.Toggle(auto,"Auto Play?");
209         //Bools for holding information on what this cue is supposed to
               do
210         GUILayout.BeginHorizontal();
211         fadeLightUp = GUILayout.Toggle(fadeLightUp, "Fade Light Up");
212         GUILayout.Label(" Duration:");
213         fadeTime = GUILayout.TextArea(fadeTime);
```

```
214         GUILayout.EndHorizontal();
215
216         fadeLightDown = GUILayout.Toggle(fadeLightDown,"Fade Light
               Down");
217         GUILayout.BeginHorizontal();
218         playSound = GUILayout.Toggle(playSound, "Play Sound");
219         soundName = GUILayout.TextField(soundName);
220         GUILayout.EndHorizontal();
221         loadScene = GUILayout.Toggle(loadScene,"Load a Scene");
222         sceneName = GUILayout.TextField(sceneName);
223         runFunction = GUILayout.Toggle(runFunction, "Run a function");
224         functionName = GUILayout.TextField(functionName);
225         GO = GUILayout.TextField(GO);
226         GUILayout.Label("Give a short description of this Cue");
227         description = GUILayout.TextArea(description);
228
229         GUILayout.BeginHorizontal();
230         //When clicking save cue, we add it to the cuelist, and save
               the information somewhere so we can access it later
231         if (GUILayout.Button("Save Cue"))
232         {
233            if (cueName != "")
234            {
235              // int myInt = Convert.ToInt32(cueName);
236               Cue cue = new Cue();
237               cue.name = cueName;
238               //cue.ID = myInt;
239               cue.autoPlay = auto;
240               cue.lightUP = fadeLightUp;
241               cue.lightDown = fadeLightDown;
242               cue.playSound = playSound;
243               cue.loadScene = loadScene;
244               cue.description = description;
245               cue.soundName = soundName;
246               cue.runFuntion = runFunction; //runFuntion
247               cue.functionToRun = functionName;
248               cue.gameObjectWithFunction = GO;
249               cue.SceneName = sceneName;
250               //Convert the string fadetime to a float value
251               if (float.TryParse(fadeTime,out result))
252               {
253                  cue.fadeTime = result;
254               }
255
256               cueArray[selGridInt] = cue;
257
258               //fill "strings to display"-array with the name of each
                     Cue in cueArray
```

```
259             for (int i = 0; i < cueArray.Length-1; i++)
260             {
261 
262                selStrings[i] = cueArray[i].name;
263 
264             }
265             firstCueDone = true;
266 
267             cueName = "";
268             description = "";
269             counter++;
270          }
271          showNewCueWindow = false;
272       }
273 
274       if (GUILayout.Button("Cancel"))
275       {
276          showNewCueWindow = false;
277       }
278       GUILayout.EndHorizontal();
279 
280 
281       GUILayout.EndVertical();
282       GUI.DragWindow(new Rect(0, 0, 10000, 10000));
283 
284    }
285 
286    void editCueWindow(int windowID)
287    {
288       //GUI.color = new Color( 1, 0, 1, 1 );
289             //GUI.backgroundColor = new Color( 1, 0, 1, 1 );
290       //Start new Vectical group
291       GUILayout.BeginVertical();
292       //Ask for Cue name
293       GUILayout.Label("Enter Cue Name");
294       //Hold Cue Name
295             //GUI.SetNextControlName ("cueName");
296       cueName = GUILayout.TextField(cueName);
297       auto = GUILayout.Toggle(auto,"Auto Play?");
298       //Bools for holding information on what this cue is supposed to
          do
299       GUILayout.BeginHorizontal();
300       fadeLightUp = cueToEdit.lightUP;
301       fadeLightUp = GUILayout.Toggle(fadeLightUp, "Fade Light Up");
302       GUILayout.Label(" Duration:");
303       fadeTime = GUILayout.TextArea(fadeTime);
304       GUILayout.EndHorizontal();
305 
```

```
306         fadeLightDown = GUILayout.Toggle(fadeLightDown, "Fade Light
                Down");
307         GUILayout.BeginHorizontal();
308         playSound = GUILayout.Toggle(playSound, "Play Sound");
309         soundName = GUILayout.TextField(soundName);
                //this problem with edit??
310         GUILayout.EndHorizontal();
311         loadScene = GUILayout.Toggle(loadScene, "Load a Scene");
312         sceneName = GUILayout.TextField(sceneName);    //this problem
                with edit??
313         runFunction = GUILayout.Toggle(runFunction, "Run a function");
314         functionName = GUILayout.TextField(functionName);
315         GO = GUILayout.TextField(GO);
316         GUILayout.Label("Give a short description of this Cue");
317         description = GUILayout.TextArea(description);
318
319               //GUI.FocusControl ("cueName");
320
321         GUILayout.BeginHorizontal();
322         //When clicking save cue, we add it to the cuelist, and save
                the information somewhere so we can access it later
323         if (GUILayout.Button("Save Cue"))
324         {
325            if (cueName != "")
326            {
327              // int myInt = Convert.ToInt32(cueName);
328               Cue cue = new Cue();
329               cue.name = cueName;
330               //cue.ID = myInt;
331               cue.lightUP = fadeLightUp;
332               cue.lightDown = fadeLightDown;
333               cue.playSound = playSound;
334                           cue.soundName = soundName;
335               cue.loadScene = loadScene;
336               cue.description = description;
337               cue.soundName = soundName;
338               cue.runFuntion = runFunction; //runFuntion
339               cue.functionToRun = functionName;
340               cue.gameObjectWithFunction = GO;
341               cue.SceneName = sceneName;
342               cue.autoPlay = auto;
343               //Convert the string fadetime to a float value
344               if (float.TryParse(fadeTime, out result))
345               {
346                  cue.fadeTime = result;
347               }
348
349               cueArray[selGridInt] = cue;
```

```
350
351             //fill "strings to display"-array with the name of each
                    Cue in cueArray
352             for (int i = 0; i < cueArray.Length - 1; i++)
353             {
354
355                selStrings[i] = cueArray[i].name;
356
357             }
358             firstCueDone = true;
359
360             cueName = "";
361             description = "";
362             counter++;
363          }
364          showEditCueWindow = false;
365       }
366
367       if (GUILayout.Button("Cancel"))
368       {
369          showEditCueWindow = false;
370       }
371       GUILayout.EndHorizontal();
372
373
374       GUILayout.EndVertical();
375       GUI.DragWindow(new Rect(0, 0, 10000, 10000));
376
377
378    }
379
380       void CueCopy() {
381
382             cueArray[selGridInt] = cueCopy;
383             updateselStrings();
384
385       }
386
387       void CuePaste() {
388
389             cueCopy = cueArray[selGridInt];
390             updateselStrings();
391
392       }
393
394
395
396    void OnGUI()
```

```
397     {
398         //If the user pressed NEW Cue, we show the new Cue dialog window
399         if (showNewCueWindow)
400         {
401             newCueRect = GUI.Window(0, newCueRect, newCueWindow, "Add
                    Cue");
402 
403         }
404         if (showEditCueWindow)
405         {
406             try
407             {
408                 editCueRect = GUI.Window(1, editCueRect, editCueWindow,
                        "Edit Cue");
409             }
410             catch (Exception e)
411             {
412 
413                 Debug.Log(e.Message);
414             }
415 
416 
417 
418         }
419 
420         //Gui groups
421         GUILayout.BeginHorizontal();
422         GUILayout.BeginHorizontal();
423         GUILayout.BeginVertical("box");
424             GUILayout.BeginVertical();
425 
426         //Button to create a cue
427             if (GUILayout.Button("New Cue"))
428             {
429                 cueName = "New Cue ("+selGridInt+")";
430                 fadeLightUp = false;
431                 fadeLightDown = false;
432                 playSound = false;
433                             soundName = "sound";
434                 loadScene = false;
435                 sceneName = "Name of the scene";
436                 runFunction = false;
437                 functionName = "Name of the function";
438                 GO = "Which GameObject has the function?";
439                 soundName = "name of soundclip";
440                 description = "";
441                             auto = false;
442 
```

```
443                     showNewCueWindow = true;
444             }
445
446                 //Button to edit a cue
447             if (GUILayout.Button("Edit Cue"))
448             {
449                     //print (cueToEdit.name);
450                 cueToEdit = cueArray[selGridInt];
451                 //Cache values to display
452                 cueName = cueToEdit.name;
453                 fadeLightUp = cueToEdit.lightUP;
454                 fadeLightDown = cueToEdit.lightDown;
455                 playSound = cueToEdit.playSound;
456                     soundName = cueToEdit.soundName != null ?
                            cueToEdit.soundName : ""; //?
457                 loadScene = cueToEdit.loadScene;
458                 sceneName = cueToEdit.SceneName != null ?
                        cueToEdit.SceneName : ""; //?
459                 runFunction = cueToEdit.runFuntion; //runFuntion
460                 functionName = cueToEdit.functionToRun;
461                 GO = cueToEdit.gameObjectWithFunction;
462                 description = cueToEdit.description;
463                 auto = cueToEdit.autoPlay;
464
465                 //int ("attempt to edit :"+ selGridInt);
466                     showEditCueWindow = true;
467
468
469
470             }
471
472                     if (GUILayout.Button("Copy Cue"))
473                     {
474                             CuePaste();
475                     }
476
477                     if (GUILayout.Button("Paste Cue"))
478                     {
479                             CueCopy();
480                     }
481
482                     if (GUILayout.Button("Expand"))
483                     {
484                             //print(selGridInt);
485                             for (int i=0;i<cueArray.Length-1;i++) {
486                                     tempArray[i] = cueArray[i];
487                                     cueArray[i] = tempArray[i];
```

```
488                                 if (i > selGridInt) cueArray[i] =
                                        tempArray[i-1];
489                                 else if (i == selGridInt) {
490                             Cue cleanCue = new Cue();
491                             cleanCue.name = "Empty ("+i+")";
492                             cueArray[selGridInt] = cleanCue;
493                                         cueArray[i] = cleanCue;
494                                         //cueArray[i].name =
                                            "("+i+")";
495                                 }
496
497                         }
498
499                         selGridInt++;
500                         updateselStrings();
501
502                 }
503
504                 if (GUILayout.Button("Contract"))
505                 {
506
507                         for (int i=0;i<cueArray.Length-1;i++) {
508                                 tempArray[i] = cueArray[i];
509                                 cueArray[i] = tempArray[i];
510                                 if (i > selGridInt) cueArray[i-1]
                                        = tempArray[i];
511
512                         }
513
514                         //selGridInt++;
515                         updateselStrings();
516
517                 }
518
519
520         if (GUI.Button(new
                Rect(Screen.width*0.015f,Screen.height*0.96f,100,20),"Delete
                Cue"))
521         {
522            Cue cleanCue = new Cue();
523             cleanCue.name = "Empty";
524            cueArray[selGridInt] = cleanCue;
525         }
526
527         //Button to load a cue
528         if (GUILayout.Button("Load Cues"))
529         {
530            loadCues();
```

```
531
532         }
533
534      //Button to save the cues to an xml file on the drive
535         if (GUILayout.Button("Save Cues(important)"))
536         {
537            saveCues();
538
539         }
540
541         GUILayout.EndVertical();
542      //Button to execute the cue across the network. If nothing
            happens, ensure that Main Camera has a NetworkView
            Component, and that the correct IP-adress is entered
543         if ( GUILayout.Button("START
            CUE",GUILayout.MinHeight(Screen.height/10)))
544         {
545            playMode = true;
546
547         }
548         if (GUILayout.Button("Scene 1",
            GUILayout.MinHeight(Screen.height / 14)))
549         {
550
551               networkView.RPC("LoadScene", RPCMode.Others, 1);
552
553         }
554         if (GUILayout.Button("Scene 2",
            GUILayout.MinHeight(Screen.height / 14)))
555         {
556
557               networkView.RPC("LoadScene", RPCMode.Others, 2);
558
559         }
560         if (GUILayout.Button("Scene 3",
            GUILayout.MinHeight(Screen.height / 14)))
561         {
562
563               networkView.RPC("LoadScene", RPCMode.Others, 3);
564
565         }
566         if (GUILayout.Button("Scene 4",
            GUILayout.MinHeight(Screen.height / 14)))
567         {
568
569               networkView.RPC("LoadScene", RPCMode.Others, 4);
570
571         }
```

```
572         if (GUILayout.Button("Scene 5",
                GUILayout.MinHeight(Screen.height / 14)))
573         {
574
575               networkView.RPC("LoadScene", RPCMode.Others, 5);
576
577         }
578         if (GUILayout.Button("Scene 6",
                GUILayout.MinHeight(Screen.height / 14)))
579         {
580
581               networkView.RPC("LoadScene", RPCMode.Others, 6);
582
583         }
584         if (GUILayout.Button("Scene 7",
                GUILayout.MinHeight(Screen.height / 14)))
585         {
586
587               networkView.RPC("LoadScene", RPCMode.Others, 7);
588
589         }
590         if (GUILayout.Button("Scene 8",
                GUILayout.MinHeight(Screen.height / 14)))
591         {
592
593               networkView.RPC("LoadScene", RPCMode.Others, 8);
594
595         }
596
597       GUILayout.EndVertical();
598       GUILayout.BeginHorizontal();
599       //Gui Code to display the Cue List
600       scrollPosition =
              GUILayout.BeginScrollView(scrollPosition,GUILayout.MinWidth(700));
601       selGridInt = GUILayout.SelectionGrid(selGridInt, selStrings, 1);
602
603       GUILayout.EndScrollView();
604       GUILayout.EndHorizontal();
605       GUILayout.EndHorizontal();
606       GUILayout.BeginVertical("box");
607       GUILayout.Label("Cue Description");
608       //To the left of the scrolling CueList
609
610         if (cueArrayAvailable)
611         {
612            GUILayout.Label(activeCueDesc,GUILayout.MinHeight(Screen.height*0.5f),GUILayo
613         }
614
```

```
615
616
617
618         GUILayout.EndVertical();
619         GUILayout.EndHorizontal();
620
621
622     }
623
624     void OnApplicationQuit()
625     {
626         Network.Disconnect(250);
627         //saveCues();
628
629     }
630
631     void saveCues()
632     {
633         XmlSerializer serializer =
634                 new XmlSerializer(typeof(Cue[]));
635
636         Stream fs = new FileStream(fileName,
              FileMode.OpenOrCreate,FileAccess.ReadWrite);
637       // XmlWriterSettings settings = new XmlWriterSettings();
            //never used!?
638         XmlWriter writer = new XmlTextWriter(fs,
              System.Text.Encoding.Unicode);
639
640         // Serialize the object, and close the Stream
641         serializer.Serialize(writer, cueArray);
642
643         writer.Close();
644
645
646     }
647
648
649
650 }
```

APPENDIX E

CUE SERVER SOURCE CODE

Listing E.1: The Cue Server code

```
using UnityEngine;
using System.Collections;

public class CueServer : MonoBehaviour
{
   public int port = 25001;


   void Start()
   {
      Network.InitializeServer(10, port, true);

   }

   void OnGUI()
   {

      if (Network.peerType == NetworkPeerType.Server)
      {
         GUI.Label(new Rect(Screen.width * 0.85f, 185, 80, 20),
             "Server");
         GUI.Label(new Rect(Screen.width * 0.85f, 170, 200, 20),
             "Connections :" + Network.connections.Length);
         if (GUI.Button(new Rect(Screen.width * 0.85f, 145, 80, 25),
             "Log Out"))
         {
            Network.Disconnect(250);

         }

      }
```

```
29          else if (Network.peerType == NetworkPeerType.Disconnected)
30          {
31              GUI.Label(new Rect(Screen.width * 0.85f, 185, 120, 20),
                    "Server Offline");
32
33              if (GUI.Button(new Rect(Screen.width * 0.85f, 145, 120, 25),
                    "Start Server"))
34              {
35                  Network.InitializeServer(10, port, true);
36
37              }
38
39
40          }
41      }
42
43      void OnApplicationQuit()
44      {
45          Network.Disconnect(250);
46
47      }
48  }
```

APPENDIX F

CUE CLASS SOURCE CODE

Listing F.1: The Cue Class which holds instructions about the cue

```
using UnityEngine;
using System.Collections;
using System.IO;
using System.Xml.Serialization;
using System.Runtime.Serialization;
using System.Text;


[System.Serializable]
public class Cue : ISerializable{
//  SerializationInfo info; //never used!?
//  StreamingContext context; //never used!?

  public int ID {get; set;}
  public string name { get; set;}
  public bool lightUP { get; set; }
  public bool lightDown { get; set; }
  public bool playSound { get; set; }
  public string soundName { get; set; }
  public bool loadScene { get; set; }
  public string SceneName { get; set; }
  public string description { get; set; }
  public float fadeTime { get; set; }
  public bool runFuntion { get; set; }
  public string gameObjectWithFunction { get; set; }
  public string functionToRun { get; set; }
  public bool autoPlay { get; set; }


```

```
32
33
34      public Cue()
35      {
36          functionToRun = "";
37          description = "";
38          gameObjectWithFunction = "";
39
40      }
41
42      static XmlSerializer serializer = new XmlSerializer(typeof(Cue));
43
44      public string Serialize()
45      {
46
47          StringBuilder builder = new StringBuilder();
48
49
50
51          serializer.Serialize(
52
53           System.Xml.XmlWriter.Create(builder),
54
55           this);
56
57          return builder.ToString();
58
59      }
60
61
62
63      public static Cue Deserialize(string serializedData)
64      {
65
66          return serializer.Deserialize(new StringReader(serializedData))
                as Cue;
67
68      }
69
70
71
72
73  }
```