A GUIDE TO CREATING VR EXPERIENCES IN HIGH FIDELITY

August 9, 2019

Alan Bettis
Danielle Newberry
Lehigh University
CS REU Summer 2019

Contents

0.1 Background

This material is based upon work supported by the National Science Foundation under grant no. CNS-1757787.

High Fidelity is not a well known or well used platform, in fact, the company itself is now re-directing away from being a Virtual Reality Metaverse. Unfortunately, this yields an extremely low number of skilled developers from which to gleam insight into the world of High Fidelity (HiFi). Regardless of this, we chose to use High Fidelity as our platform due to its compatibility with virtual reality and most importantly, the ease it brings to creating NPCs. After a summer of scraping the edges of what the internet has to offer, attending VR QA sessions, and reaching out to High Fidelity both directly and through posting on forums, we have compiled a manuscript of all the useful tidbits we have come across in an attempt to minimize the struggles faced by those who follow us.

Disclaimer: Next to none of the code snippits included in this manuscript are our completely original code. Most of the code shown has been adapted either directly from the High Fidelity API or from High Fidelity's Github.

Chapter 1

Script Types

In order to have an interactive environment, one must be familiar with the various types of scripts High Fidelity uses. Each script type has different functionality. Some programs can be run in various ways and some are stricter in what script type they are supported by. Script type is determined by how you host/run your code. The purpose of your code will dictate what type of script you will have to run it as.

1.1 Client Entity Scripts

Direct applications:

- Used for allowing user input to trigger behaviors

Client Entity Scripts make the environment interactive. They are attached to entities by adding the script's url into the entity's "script" property. This can be done in High Fidelity's create menu or in another script when you spawn an entity.

1.2 Interface Scripts

Direct applications:

- Spawning entities with specific settings (such as interactive scripts)

Interface Scripts are loaded in through the "running scripts" segment and are attached directly to the player. **Note: This means the domain MUST be fully running for the scripts to run. See known issues section for details.**

Running scripts can be accessed through the High Fidelity Interface by selecting Edit > Running Scripts. These can then be uploaded from disk. This is the only script type that cannot be used from the Asset Browser; They must be loaded directly into Running Scripts.

1.3 Assignment Client (Persistent) Scripts

Direct applications:

- Hosting NPC scripts

Assignment Client Scripts are scripts that persistently run in you domain. Every time you re-open or reset your domain, the code in this script will run all over again and anyone who enters your domain will experience the effects of this script.

To add an Assignment Client Script, you must first create your own domain (see Chapter 2). Upload your JavaScript script into the Asset Browser and right click to copy the file's asset browser url. Then, in your domain settings, navigate to settings > scripts, and paste your code's url into the requisite textbox that appears when clicking "+". Once the url is included, click "save and restart" to restart your domain with the script included.

Chapter 2

Private Domains

Private domains can be created by using the High Fidelity Sandbox application that can be downloaded from High Fidelity's website. On Windows, the Sandbox can be run as a program, at which point there will be an icon on the taskbar. Right-clicking this icon will allow the server to be started/restarted, settings to be brought up, and other assorted actions.

Once restarted, all assignment client scripts on the domain will also rerun, although **there is a known issue that the script will begin a seemingly random amount of time after a restart.**

Two important things can be done from the settings menu of your sandbox, which can be accessed by right-clicking the taskbar icon.

- Set Permissions
- Include Assignment Client Scripts

Setting permissions is covered in detail here. It is worth noting that if your interface scripts create entities in the domain, they will not work if you do not have rez permissions in the domain.

Including Assignment Client Scripts is covered in the Assignment Client Scripts section of chapter 1.

Chapter 3

The Asset Browser

After you create your own private domain you will be utilizing the asset browser to add all content, such as scripts, models, and audio files to your environment.

To add an asset, first navigate to Create > Open This Domain's Asset Server. In the asset browser you can either upload files from their url or from the disk. Usually you will be uploading items from the disk of the computer you are working on, in which case you will choose "disk" and then navigate to the place they are stored. After doing this you will be able to add these items to the world or copy their url depending on your needs.

Chapter 4

Creating NPCs

High Fidelity allows a user to record a set of actions in VR or out of VR and have these actions playback on an agent avatar. This feature was for us the main draw of High Fidelity despite all its numerous other bugs and issues. To create an NPC from a recorded avatar animation, first you must make sure that the EZrecord interface script is in your current running scripts. This can be found by going to the top of the screen and selecting Edit > Running Scripts. Once in the running scripts window, there should be a folder titled "developer," and the EZrecord script should be found there. Select it, then click the add button to add it to the list of running scripts. Once you have done this, an icon should appear next to the create button on the bottom bar, or on your tablet in VR (which can be opened with the top button on the Vive controller). Click on this button and record the actions you want your agent to perform. Audio will be recorded as well if your microphone is unmuted. Click again to stop the recording. After creating your recording file, navigate to your account under Users then navigate through AppData > Local > High Fidelity > Interface > Avatar Recordings and rename your hfr file. Upload this file in the asset browser so you can copy its url. Paste the recording url into the code below. This code can be modified to loop the recording or be triggered by different actions, but the basic animation code will remain the same.

Save this code in a Javascript script and upload it onto your asset browser. Then paste the script's url into your domain's scripts as described in Chapter 2.

The playFromCurrentLocation variable determines if the agent begins its animation from where it was when it was recorded or from a given current location (if your hooking up animations to play one after another this avoids having to be meticulous when making animations.) It is important that toggling the loop variable alone is not enough to stop the animation from looping. You must implement the disconnect function.

```
var play = function (){
        var PLAYBACK_CHANNEL = "playbackChannel";
        Recording.loadRecording(/*Insert animation URL here*/);

        Recording.setPlayFromCurrentLocation(false);
        Recording.setPlayerUseDisplayName(true);
        Recording.setPlayerUseAttachments(true);
        Recording.setPlayerUseHeadModel(false);
        Recording.setPlayerLoop(false);
        Recording.setPlayerUseSkeletonModel(true);

        Agent.isAvatar = true;
        if (Recording.isPlaying())
        {
                Script.update.disconnect(play);
        }
        Recording.setPlayerTime(0.0);
        Recording.startPlaying();
}

Script.update.connect(play);
```

Chapter 5

Audio Recordings

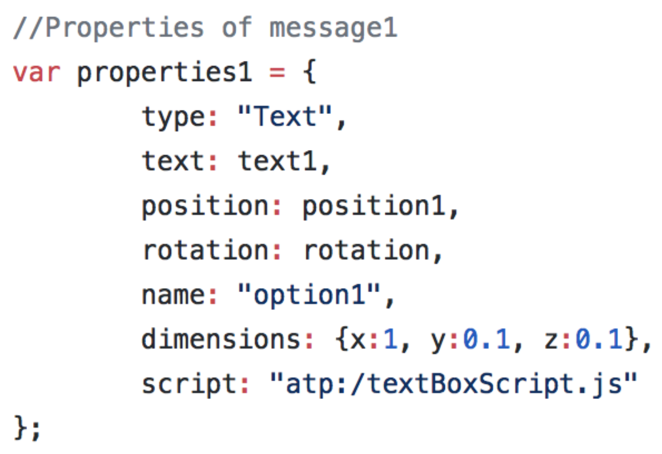
Audio Recordings are compatible with all script types. The easiest way to create a MP3 audio file is by finding a website that you can record and download directly on/from. Once you have your audio file uploaded in the asset browser, simply paste the url into the code below:

```
var sound = SoundCache.getSound("atp:/AudioEx.mp3");

function playSound() {
    var injectorOptions = {
        position: MyAvatar.position
    };
    var injector = Audio.playSound(sound, injectorOptions);
}

function onSoundReady() {
    sound.ready.disconnect(onSoundReady);
    playSound();
}

if (sound.downloaded) {
    playSound();
} else {
    sound.ready.connect(onSoundReady);
}
```

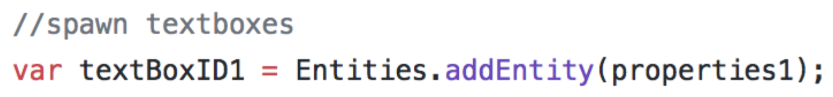
Chapter 6

Spawning/Deleting Entities

Spawning and deleting entities can be done either by using the create app or via script. It is easier to do so in the create app, however, it is likely in a simulation you will want things to appear and disappear on queue. To achieve this effect you will need to use a script. To spawn an entity via script you first must define the entity properties. Different entity types have slightly differing properties. **It is important to note that you are only able to spawn basic entities via script (not items found on the marketplace or imported from elsewhere).** After the properties are set you can add the entity to your domain.



```
//Properties of message1
var properties1 = {
        type: "Text",
        text: text1,
        position: position1,
        rotation: rotation,
        name: "option1",
        dimensions: {x:1, y:0.1, z:0.1},
        script: "atp:/textBoxScript.js"
};
```



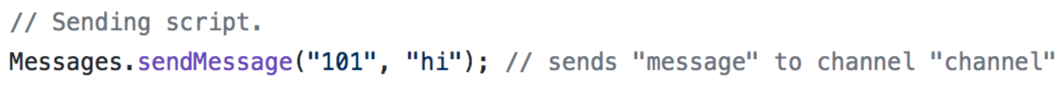
```
//spawn textboxes
var textBoxID1 = Entities.addEntity(properties1);
```

There are a handful of other entity properties you can define as well (ex: if it is grabbable and/or triggerable, ect.)

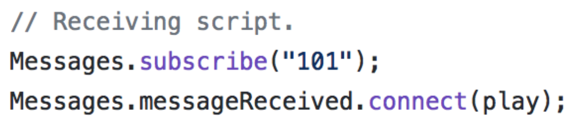
Chapter 7

Messaging System

The Messages API in High Fidelity allows scripts to subscribe to different channels. Scripts that are subscribed can have functions called by external scripts in response to specific triggers. Persistent scripts do not have the capability to respond to triggers directly, but all other script types do. This API allows different types of scripts to communicate with one another. Often this API will be used when a user response should trigger an animation (or something in the environment to change).



```
// Sending script.
Messages.sendMessage("101", "hi"); // sends "message" to channel "channel"
```

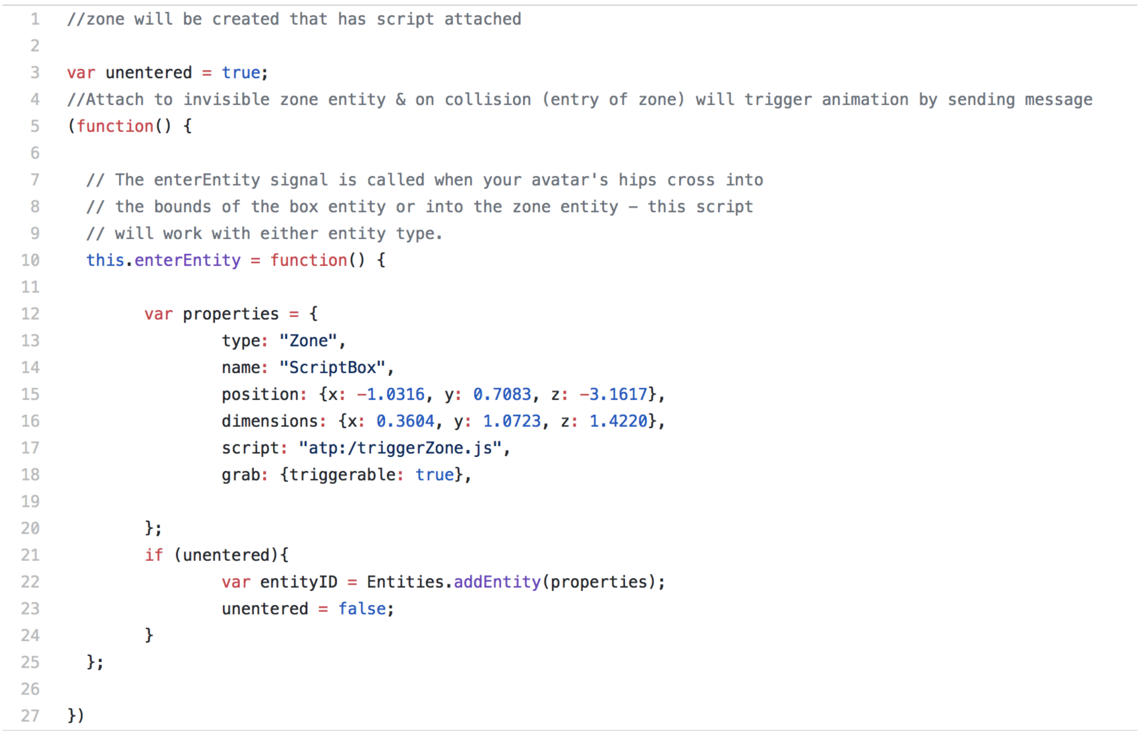


```
// Receiving script.
Messages.subscribe("101");
Messages.messageReceived.connect(play);
```

Chapter 8

Trigger Zones

Trigger zones are useful when you want to trigger a reaction in the environment, but do not want the user to know that their action is the cause.



```
//zone will be created that has script attached

var unentered = true;
//Attach to invisible zone entity & on collision (entry of zone) will trigger animation by sending message
(function() {

  // The enterEntity signal is called when your avatar's hips cross into
  // the bounds of the box entity or into the zone entity - this script
  // will work with either entity type.
  this.enterEntity = function() {

        var properties = {
                type: "Zone",
                name: "ScriptBox",
                position: {x: -1.0316, y: 0.7083, z: -3.1617},
                dimensions: {x: 0.3604, y: 1.0723, z: 1.4220},
                script: "atp:/triggerZone.js",
                grab: {triggerable: true},

        };
        if (unentered){
                var entityID = Entities.addEntity(properties);
                unentered = false;
        }
  };

})
```

This script is attached to a zone entity. When a user enters the zone this script will spawn a zone entity in a different location with a trigger script attached to it. This will only occur the first time the zone is entered.

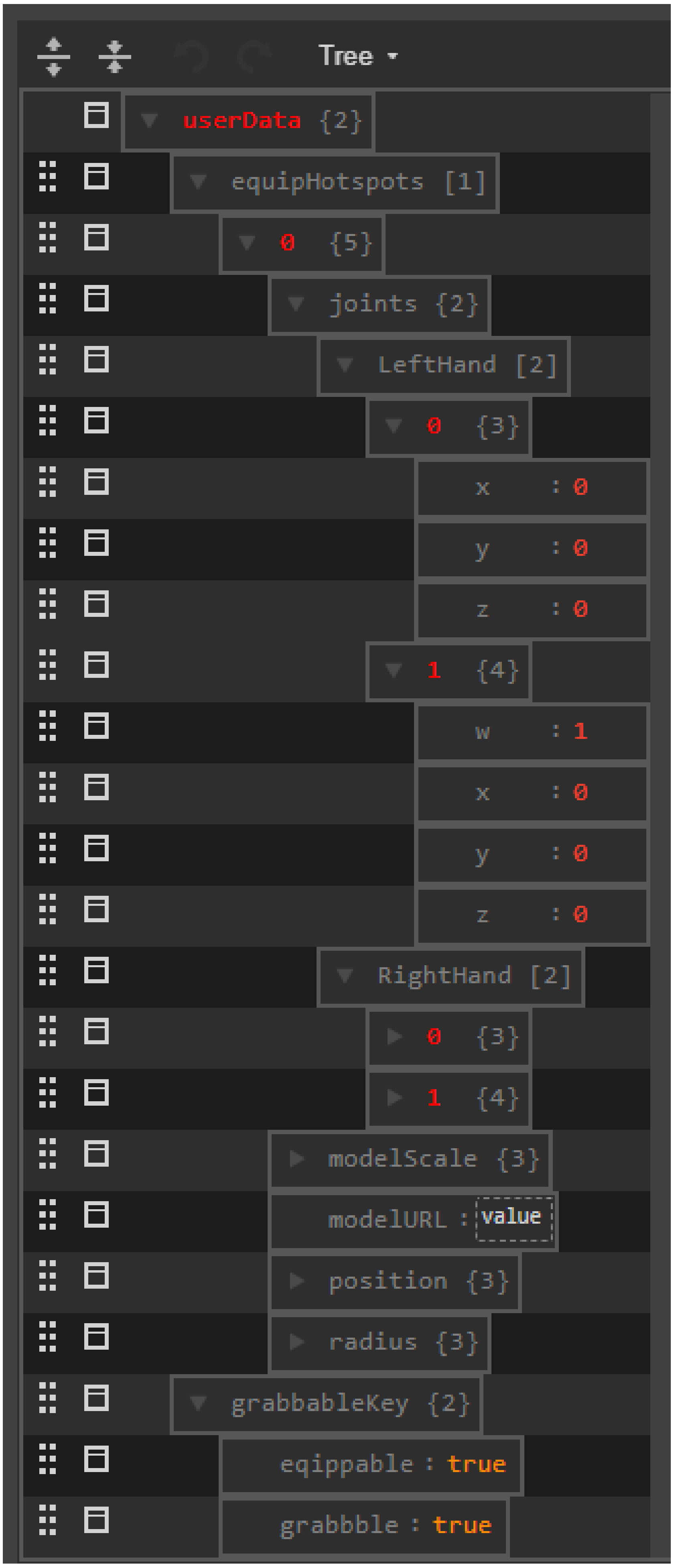
Chapter 9

Wearables

Unfortunately, user's are unable to spawn Market Place items via script. To get an item to become wearable after a user click, we have to manipulate the entity's behavior tree, which is accessible only through the create app. In the create app, navigate to behavior and then user data. You can modify data here as a tree or as code.

If you are dealing with a wearable spawned from the marketplace you will first want to disconnect this from your avatar. To do this go into the create app and set their parent index to 0.

Problem: Although it certainly must be possible, we were unable to have wearable objects in the environment attach to any part of the user's avatar other than the wrists.



Tree

```
userData {2}
  equipHotspots [1]
    0 {5}
      joints {2}
        LeftHand [2]
          0 {3}
            x : 0
            y : 0
            z : 0
          1 {4}
            w : 1
            x : 0
            y : 0
            z : 0
        RightHand [2]
          0 {3}
          1 {4}
      modelScale {3}
      modelURL : value
      position {3}
      radius {3}
  grabbableKey {2}
    eqippable : true
    grabbble : true
```

Chapter 10

Importing 3D Assets

Both whole scenes and objects (including avatars) can be found on various sites online and imported into High Fidelity.

10.1 Avatars

Avatars must meet certain benchmarks to work in High Fidelity.

- Rigged in T-pose
- FBX file
- Has textures

It is likely you will need to download and use High Fidelity's Avatar Exporter for Unity.

Note: Even with all of these requirements met it is not guaranteed your avatar will work as expected.

More information can be found here.

10.2 Scenes/Objects

If you import a scene/object into High Fidelity and it is unexpectedly grey that means you need to semi-manually apply the appropriate textures. For this you will need 3D Max software, when exporting make sure to check the "embed materials" box.

Chapter 11

Walkthrough of our Code

All of our code for this project can be split up into three types of script:

- Entity scripts
- Spawner scripts
- engineCore.js

The engineCore.js script contains all the central controlling code, interface-type spawner scripts create entities with specific entity scripts attached to them and specific settings, and the entity scripts allow for user interaction.

11.1 Entity scripts

Most entity scripts follow this basic pattern:

They must be wrapped entirely in a function like so:

```
(function()
{
    //code here
});
```

In order to store their own id, a variable must be declared at the beginning, and the following function should be in the code to perform the operation:

```
this.preload = function(entityID)
{
   _selfEntityID = entityID;
};
```

They should subscribe to a channel to receive messages from other scripts. As an example, most if not all of our entity scripts have some function to self delete on a certain command so that interactive elements can disappear after their purpose is served. Some message listeners may serve other purposes, such as toggling a flag to allow the entity to be interactive or not.

Lastly, they should have some way for the user to interact with them, be it an "on click event listener" such as the following for text boxes and other clickable objects...:

```
this.mousePressOnEntity = function()
{
    //On click, do this
};
```

or an event listener for when users enter a zone, such as this:

```
this.enterEntity = function()
{
    //When user enters the zone, do this
};
```

Of course, there are exceptions to these rules, such as our instructionBoxScript.js script, which lacks any sort of user interactivity, instead it merely displays instructions and deletes alongside the accompanying entities.

11.2 Spawner Scripts

Our spawner scripts are interface scripts that, as the name suggests, are responsible for spawning entities into the environment.

All spawner scripts must subscribe to a messaging channel first, so they can be instructed to spawn entities when desired. They all must have javascript objects that contain properties of entities to spawn. Once the objects are all declared and initialized with their parameters, the spawner script can connect a function to the messageReceived signal. This function will simply have a set of function calls to add the spawner's entities to the environment.

You can find a detailed list of the parameters for entities here.

11.3 engineCore.js

The flagship script of our program's fleet is the engineCore.js script. This assignment client script contains most of the core logic of the VR experience.

It begins by declaring a set of javascript objects to represent the states. Each state object contains a string transition and four option objects. Each option contains a dialogue string, an animation URL, and an audio URL.

Once all the states are declared, the engine defines a mapping for the transition strings to the states. This is needed because the states must all be declared already when the transitions are declared, so placeholder strings act as keys for the map, allowing bidirectional flow for states if desired.

Following this is initial setup, setting the active state to the first state, and declaring any stateless URLs for the startup. initFlag is set to true and the counter is set to 0 for animation purposes, and the script is subscribed to the "engine" channel to listen for messages from itself and from other scripts.

For now, it sends a message out to the shelfWatchSpawner to spawn all shelfWatch entities. It is worth noting that if there exist shelfwatches already in the scene, it will spawn duplicates. Take care to delete all non-locked entities in the environment before restarting the domain.

11.3.1 Setting the Scene

The settingScene function is then called, starting the experience. This function was rather rushed, so it is worth noting that it could use a rewrite for modularity and readability. Everything involved with it is stateless, and the actual FSM doesn't start until it's done. It begins by playing the clerkIdle animation, which is just a recording of the clerk standing still for 1 second. This ensures that the clerk is in the store from the beginning, and can be used to tell when the experience is ready to begin, as a clerk will appear when everything is working.

It continues by subscribing to 4 different messages. When the cashierZone is triggered by a player approaching the clerk, the clerk will play an animation. Once done, a message is sent to a script to spawn a set of watches to be selected from. Once selected, the script attached to the watch messages the engine, and the engine will play an animation on the clerk again, then trigger the paymentOptionSpawner script to spawn text boxes allowing the player to choose a payment option. Once selected, the entity script on the selected text box signals the engineCore to continue, and then a final clerk animation plays.

The user will then follow the prompts of the clerk's animations and go to the other room of the store to pick up their watch, which has been signalled by the engineCore to be highlighted and clickable. Once clicked, it will self-delete to simulate being placed in the player's bag. This will then spawn a threshold zone from the zoneSpawner script.

Lastly, once the player goes through the threshold back into the main room of the store, the thresholdZone will message the engineCore to trigger the final part of the settingScene function: the initFlag variable is reset to true to allow the manager to spawn in a different location, and the manager's first animation URL is set to the current animation URL. Then, the manager's first animation plays and the FSM begins.

11.3.2 The Finite State Machine

The executeState function is the first function called. If the initFlag variable is true, then it just plays the current animation. Otherwise, it waits for any audio to finish playing first, then plays the current animation. Either way, once the animation is complete, it calls the menuSpawner function, passing all the option dialogues from the current active state. The menu spawner function then parses these strings and sends them to the messageSpawner script, which parses them further and spawns a set of text boxes. The dialogue is then read out by the audio from each current option's audioURL. Once finished, the message sent from the audio player function allows the text boxes to become interactive. When the player selects a text box, the animation associated with that option plays, and then the text boxes delete and the state transition occurs, spawning more text boxes to allow the player to respond to the animation that just played.

This behavior continues until the final state, which currently is set to transition back to state 1. The engine should be modified to instead reset all starting entities and replay the behaviors from the settingScene function.

Chapter 12

Known issues

- Upon loading a domain, High Fidelity seems to take a random amount of time to execute assignment client scripts. The reasons for this are unknown, but repeated restarts of both the domain and physical machine seem to be a minor remedy.
- If an assignment client script is set to send messages to an interface script, and the assignment client script begins running before the domain is completely loaded, it may send the message before the user enters the domain. If this happens, the interface script will not receive the message and another restart will be needed.
- Due to reset function not being implemented, entities have to be manually deleted through the create menu between domain resets.