THE UNITY APPLICATION SOFTWARE DOCUMENTATION

Table of Contents

List of Figures

List of Acronyms

UI	User Interface
FBX	Filmbox format owned by Autodesk
Xsens	Xsens 3D motion tracking
HUD	Heads-Up Display

1. INTRODUCTION

Movement screens are used across a variety of settings including ergonomic, clinical, and athletic settings to quantify ‘movement quality’ and identify movement patterns that are associated with an increased risk of injury and/or decreased performance. There are many different types of movement screens, with the Functional Movement Screen (FMS) being the most well-known. Each screen has its own unique battery of movements and scoring criteria, but each share one constant: they are all scored using visual appraisal, which is a subjective approach.

Within the literature, there are conflicting results regarding inter- and intra-rater reliability, even within the same movement screen. When looking across studies, this is most likely attributed to the small number of raters being compared within each study (with the majority of studies using only two raters), rater experience, real-time scoring versus scoring from videos, and the qualitative interpretation of reliability measures. When looking within studies, the variability in inter- and intra-rater reliability is thought to be due to the dynamic nature of the movements, the rater’s perspective, and/or rater bias.

Using video data or 3-D reconstructed models from motion capture data to demonstrate a movement could increase the reliability of a movement rating system. Raters in such systems can see the movements from multiple angles and even replay the movements within the boundaries the systems allow. Employing 3-D character models could also minimize bias in rating since the raters are not seeing the person performing the task.

2. APPLICATION DESCRIPTION

The current unity application is designed to help researchers study inter- and intra-rater reliability. There are two types of data that can be used as input for the application: animation files (.fbx) or raw video data. For the animations, a variety of 3-D character models can be used for scoring the movements, the number of replays for each animation can be limited based on the researcher's preferences, and there are free camera movement controllers to help the rater achieve the desired viewing angle of the animation. For raw video data, multiple camera angles can be viewed simultaneously to allow raters to have multiple views of the same movement. For both the video and animation data, the implemented scoring system will save the score, the name of the rater, the animation type and name, and the exact time and date of the submission of the score.

This application comes with local and offline support by default. It is also possible to implement an online version that can be deployed on a Linux Apache server with a MySQL database. All the raters' data and scores will be handled and saved in the database. The description of the online module will be discussed in its dedicated chapter at the end of this documentation.

3. APPLICATION DESIGN

3.1. UI

The application UI components are as follows:

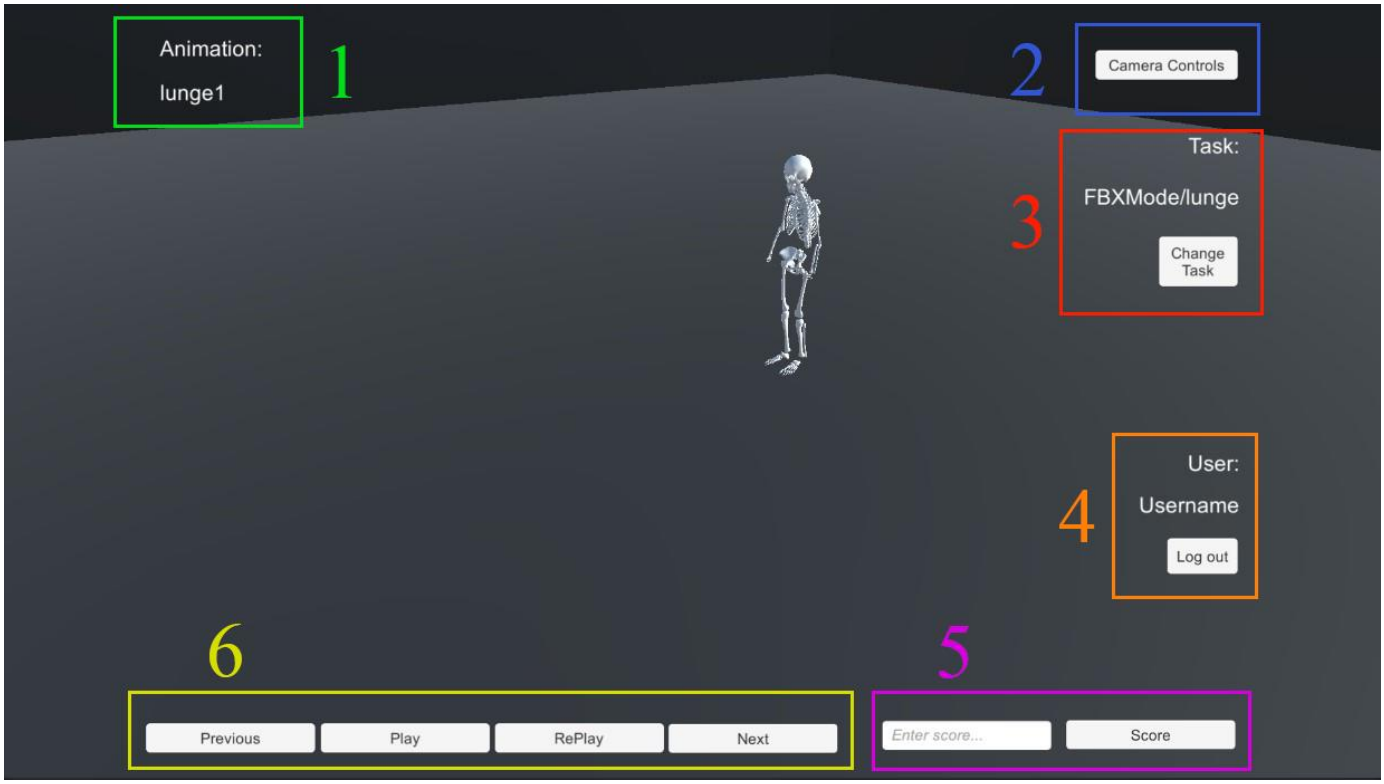

Figure 3-1 The Unity Application Running UI

1. Animation name:

   This component displays the current animation name.

2. Camera Controls:

   This component will show the key bindings for real-time camera controls.

3. Task:

This component shows the current mode and task. By pressing the “Change Task” button the user will be redirected to the mode selection menu and should choose the next task afterward.

4. User:

   This component shows the name of the user. Pressing the “Log out” button will redirect the application to the first menu of the application which is the “Enter your name” menu.

5. Score:

   This component is for the user to type and submit the score for the current animation.

6. Animation controls:

   This component is for animation controls. The rater can play the current animation by pressing the “Play” button. “Next” and “Previous” buttons are responsible for going to the next or previous animation respectively. There is also the “RePlay” button which could be programmed to restrict the number of times a rater can replay a certain animation. This button is not programmed in the stock version of the application.

3.2. Camera

The camera system is designed to let the rater have the proper camera control flexibility around each animation. There are two scripts regarding the camera controls and movements: “Camera Orbit” and “Camera Follow”. “Camera Orbit” contains the script for different camera movements around the character such as zooming, panning, and rotating. “Camera Follow” is the script for locking the camera on the current 3-D character and following it as it moves and plays the animation. Each of these scripts could be disabled from the “MainCamera” game object which is the child game object of “CameraParent”.

The camera controls of the current Unity Application are as follows:

Figure 3-2 The Unity Application Camera Controls

The camera settings for the current Unity Application are as follows:

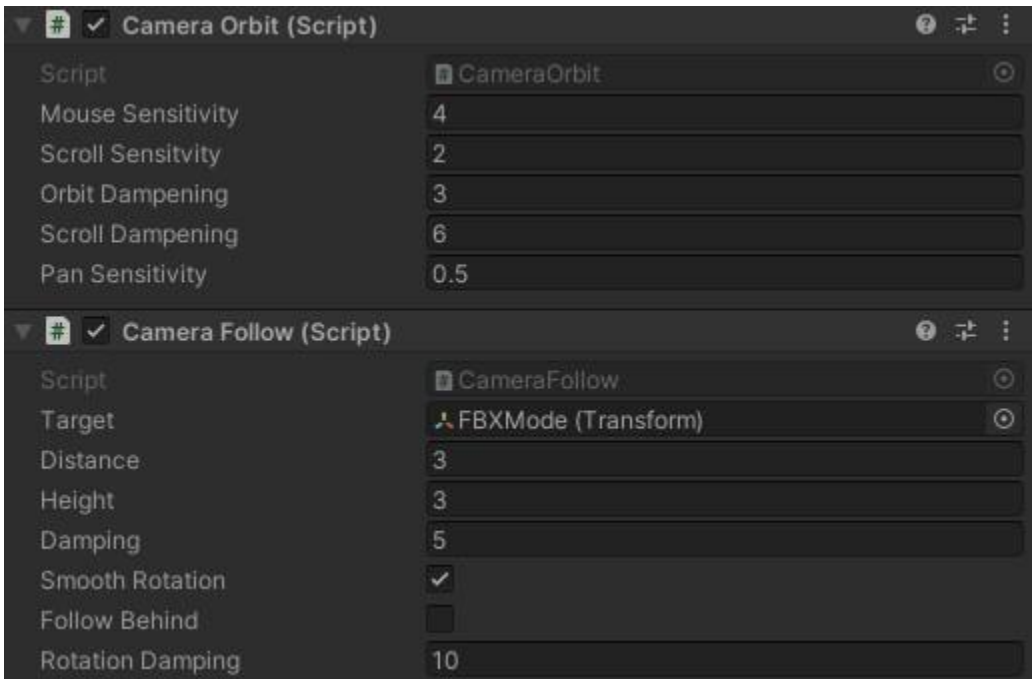

Figure 3-3 The Unity Application Camera Settings

3.3. Scoring

The scoring system in the Unity Application can be either offline or online. This package comes with the offline scoring system by default but an online system is also possible which will be discussed in its own dedicated chapter within this document.

The scoring component consists of “ScoreInputField” and “ScoreButton”. If the score that has been entered into “ScoreInputField” is a valid integer number and it is within the upper and lower bound, which is defined in the “Score.cs” script, then the “ScoreButton” will submit the score. The score will be saved with the following example format:

```
Username: Username/Mode: FBXMode/Animation: lhop1/Score :5/4/27/2022 11:56:23 AM
Username: Username/Mode: XsensMode/Animation: lunge1/Score :9/4/27/2022 11:57:16 AM
Username: Username/Mode: VideoMode/Animation: vertJump1/Score :3/4/27/2022 11:57:26 AM
```

Figure 3-4 The Unity Application Local Score Format

The score will be stored as a .txt file in the application data path. The path will also be printed in the console area of Unity whenever the “ScoreButton” is pressed while in the test play.

3.4. Modes

Three different modes have been implemented for this version of the Unity Application: FBX, Xsens, and Video.

3.4.1. **FBX**: This mode is for a set of animations that have the 3-D character model attached to them. In order to populate this mode with animations, a set of .fbx files are required that have the motion capture animation baked into their corresponding 3-D character

model. These 3-D character models can either be humanoid or generic.

3.4.2. **Xsens**: In this mode motion capture data from Xsens should be used. This mode populates using motion capture data from Xsens without any 3-D character attached to or baked with the animations of the movements. Any humanoid 3-D character model could be assigned to these movements in Unity.

3.4.3. **Video**: This mode needs four different videos for each animation from different angles.

3.5. Tasks

Task is the type of movement that any particular set of animations in any mode form would be. As samples, three different tasks have been provided in the stock unity application to showcase how different tasks work. A user will choose from the task list after they choose their desired mode. Each task contains a set of animations of the same movement. The rater will be responsible to score these movements based on the provided animation/video. Sample tasks in the stock application are L-hop, Lunge, and Vertical Jump. They come in all three modes: FBX, Xsens, and Video.

3.6. Game Scene Hierarchy

In this section, each element in the game scene hierarchy will be presented and explained.

3.6.1. Canvas_LocalLogin

This game object is the login menu canvas and contains the InputField for the username of the rater using the application. The “Proceed” button will call the “Proceed” function inside the “LocalLoginController.cs” script.

3.6.2. Plane and BlackBox

These game objects are the ground and surrounding planes of which the animations will be shown inside. There are no functions assigned to any of these game objects.

3.6.3. Directional Light

A simple directional light to illuminate the scene around the 3-D character model from any of the modes that have one.

3.6.4. CameraParent

This game object holds the “MainCamera” game object which is the main camera of the animation. “MainCamera” holds the two camera scripts, “CameraOrbit” and “CameraFollow”, which will handle the camera controls for camera movement and following of the 3-D character in the scene.

3.6.5. GameScene

The GameScene game object includes the main scene UI elements and 3-D character models. The “Canvas” child game object contains the “GameHUD” game object which holds all the HUD elements for the main scene of the application. Each element has its corresponding functionalities and scripts attached to it.

“FBXMode” and “XsensMode” prefabs are the 3-D character models of their corresponding modes. These prefabs have their own dedicated Animator components to drive the 3-D character model.

3.6.6. LocalOrOnline

This game object is dedicated to the “IsLocal.cs” script which holds the “IsLocal” Boolean to indicate whether the application is in its online mode or not. The stock version only supports the local version so this value is true by default.

3.6.7. LocalAnimController

This game object is dedicated to the “LocalAnimControl.cs” script.

3.6.8. LocalLoginControl

This game object holds all of the login-related scripts. That includes “LocalLoginController.cs”, “LocalModeSelection.cs”, and “LocalTaskSelection.cs”.

3.6.9. LocalScoreController

This game object is dedicated to the “LocalScore.cs” script.

3.6.10. LocalVideoController

This game object is dedicated to the “LocalVideoController.cs” script.

3.6.11. VideoComponents

This game object holds the four different video players for each angle. This includes top left, top right, bottom left, and bottom right with each video player holding its own target texture. Target textures are “Render Texture” which play the video on them. The video player will play any video that is addressed in their “URL” element.

4. APPLICATION IMPLEMENTATION

The first thing that the user or rater is presented with after launching the application is the “Canvas_LocalLogin” menu. After putting the username in the dedicated input field and clicking the “Proceed” button, the “LocalLoginController.cs” script will be called. This script is attached to the “LocalLoginController” game object which is always active after the application is launched.

The second menu is the “Canvas_LocalModeSelection”. The user must select one of the presented modes from the menu list. The “LocalModeSelection.cs” script which is attached to the “LocalLoginController.cs” game object is responsible to handle the mode selection process. The “Video” mode differs from the other two animation modes and the corresponding code is handled differently as well. This is mostly due to the fact that this mode does not have a 3-D character model to drive, instead, it activates the “VideoComponents” game object which holds all the video players.

The next menu is the “Canvas_LocalTaskSelection”. The user will be presented with all the available tasks to choose from. “LocalTaskSelection.cs” script will handle task selection. The code is fairly similar for all the tasks. For each task, the “VideoMode” is checked as an exception since the procedure to activate a video mode for any task is different than the 3-D character model ones.

For any non-video mode (FBX or Xsens in the stock version of the application for instance), the next phase of the application will load the corresponding 3-D character model in the middle of the screen. The camera focus will be on the 3-D character model and the camera follows the model by default. This can be changed by changing the camera script settings in the “MainCamera” under the “CameraParent” game object.

The “Camera Controls” button shows the key bindings for camera control.

The first animation will not start playing until the rater clicks on the “Play” button. The rater can score any animation that is currently present by putting the score in the dedicated input field and pressing the “Score” button. To check the name of the current animation, the animation name is located on the top left of the screen.

The rater can go to the next or previous animations by clicking their corresponding buttons.

In the current stock form of the application, the rater does not hold any constraints regarding the times the rater can replay an animation. This can be changed by utilizing the “RePlay” button. The “Play” button could be inactivated after the first play of the animation and then a replay of any number could be scripted for the “RePlay” button.

To change the mode and the task, the rater can click on the “Change Task” button which will redirect the rater to the “Canvas_LocalModeSelection” menu.

To log out or change the rater, the “Log out” button must be clicked which will bring the application to the first “Canvas_LocalLogin” menu.

5. CUSTOMIZATION

In this chapter, the process of customizing the application to support added and extended animation modes and tasks will be discussed. C# programming skills and being familiar with Unity are required and necessary to customize this application.

5.1. Mode

There are currently three modes available in this unity application package. These modes include FBX, Xsens, and Video.

5.1.1. FBX:

There are multiple ways to customize this mode. First, you can change the animation character; however, the animations must be baked and already attached to a 3-D model for this mode. For any specific body type, there must be a specific prefab in Unity. Each prefab (which contains a specific 3-D character model) should resemble the body type of the person by whom the motion capture data was recorded. This will make sure that the accuracy of each movement is properly reflected in the final animation as assigning different animation files onto different 3-D character models inside Unity could reduce the movement accuracy especially when the rig is generic rather than humanoid.

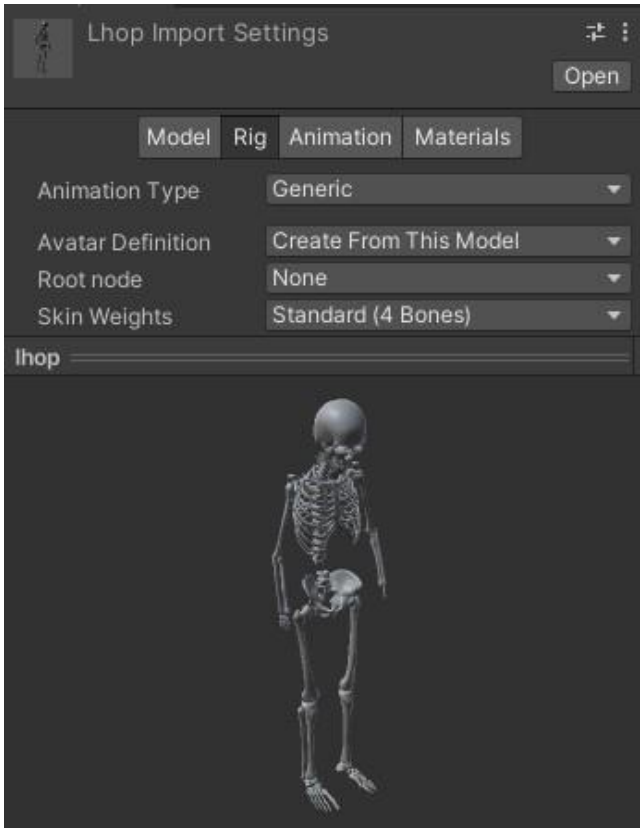

Figure 5-1 FBX prefab in the GameScene

The provided FBX samples in this application are from a single 3-D character that has animation information for three different tasks. However, a group of 3-D characters could be inserted as prefabs under the “GameScene” game object. Then a random sequence of animations from the animation pool could be played when a specific task is chosen.

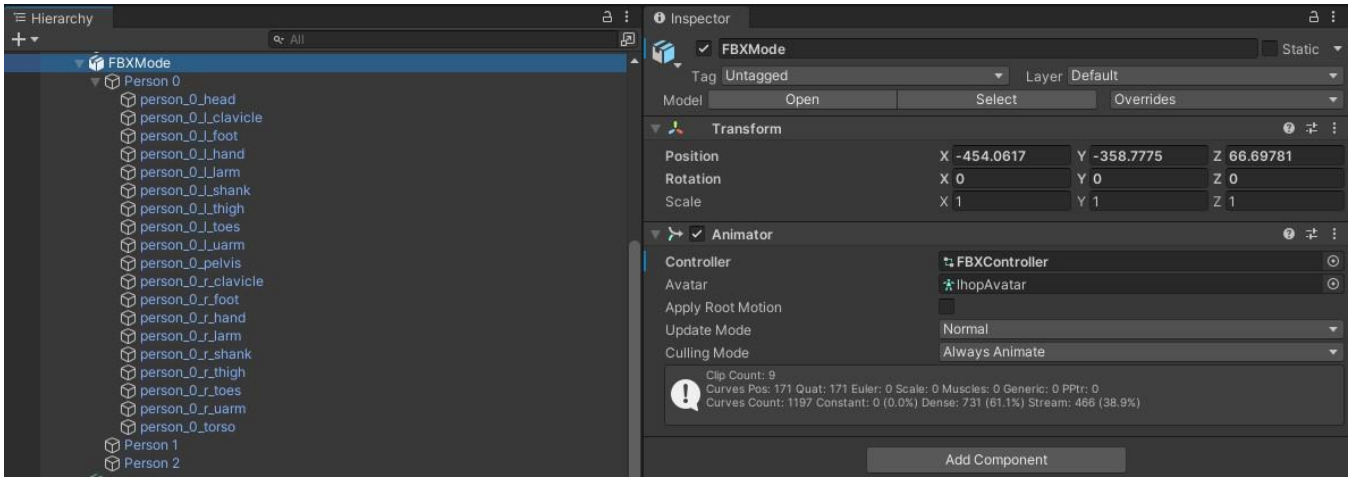

Figure 5-2 Sample of FBX file settings for an L-Hop task

There is a folder under the “Assets” called the “AnimationControllers”. This folder contains the Animator Controller components for all the different modes. Each mode only needs one Animator Controller no matter how many prefabs are corresponding to that mode. In order to customize and add or remove animations from FBX mode, open the “FBXController” from the “AnimationControllers” folder. Then add all of the new animations into the state machine area of this Animator Controller. Name each animation state accordingly (include task and body type in the name for instance). For the next step, the “LocalAnimControl.cs” script must be opened and the animation lists must be updated as they were added to the state machine of the Animator Controller. This would be inside the “Start()” function of this script. Remember that the sequence of the animations added to the lists is the exact sequence they will be shown up in the application.

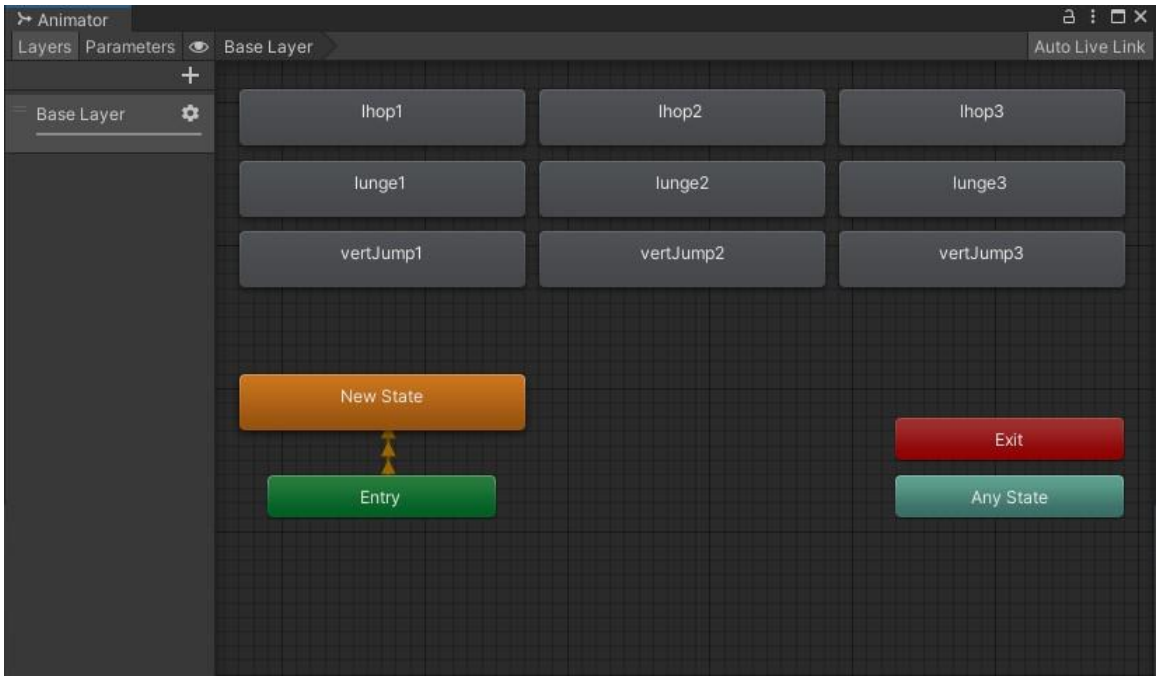

Figure 5-3 Adding the animations to each corresponding list after putting the animation files in the Animator Controller

The “LocalAnimControl.cs” script is unique across the application and it handles all of the modes. This means that all the animation tasks must be exactly the same for all of the modes, otherwise, the same script would not work across the modes.

```
25      void Start()
26      {
27
28          lhopList.Add("lhop1");
29          lhopList.Add("lhop2");
30          lhopList.Add("lhop3");
31
32          lungeList.Add("lunge1");
33          lungeList.Add("lunge2");
34          lungeList.Add("lunge3");
35
36          vertJumpList.Add("vertJump1");
37          vertJumpList.Add("vertJump2");
38          vertJumpList.Add("vertJump3");
39
40          currAnimList = lhopList;
41
42          currAnimIndex = 0;
43      }
```

Figure 5-4 Animator tab for FBX Animator Controller. Body type is not specified in the names since there is only one body type in this sample

If extra FBX prefabs are added to the scene, the “Controller” element under the “Animator” component for that prefab must be filled with “FBXController” from the “AnimatorControllers” folder.

The mechanism to change the prefab for different body types must be implemented in the scripts to swap the current prefab with the proper one in real-time. Remember to also change the camera settings to point at the new prefab as well.

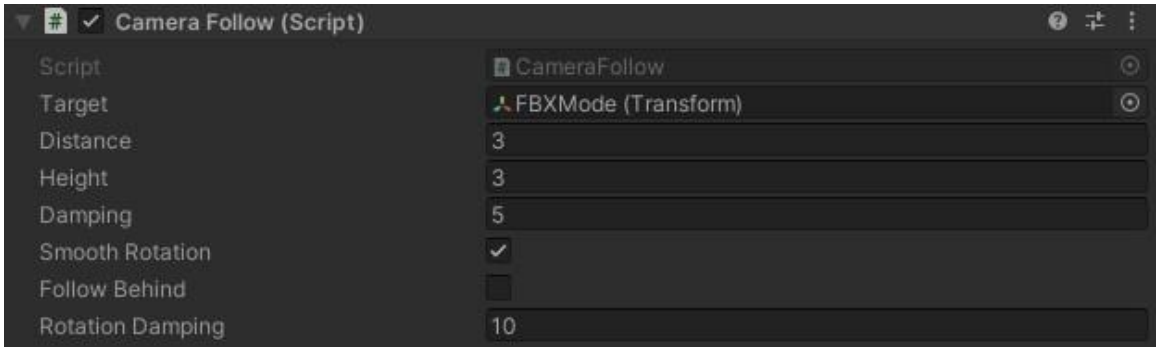

Figure 5-5 Camera Follow target must match the selected prefab

5.1.2. Xsens

In many ways, Xsens mode is similar to the FBX mode and almost all of the instructions to customize FBX would also apply to Xsens mode as well. So, in this section, the differences between Xsens and FBX will be discussed.

Xsens mode holds the humanoid animation data without any 3-D character model attached. Therefore, a humanoid 3-D character model is required to be able to drive the animation data. Unity supports any humanoid rig to drive the Xsens animation data. Just make sure that the “Animation Type” under the “Rig” menu for the prefab is set to “Humanoid”. This mode is not possible without a humanoid 3-D character model. A default Humanoid 3-D character model is provided in this application as a sample.

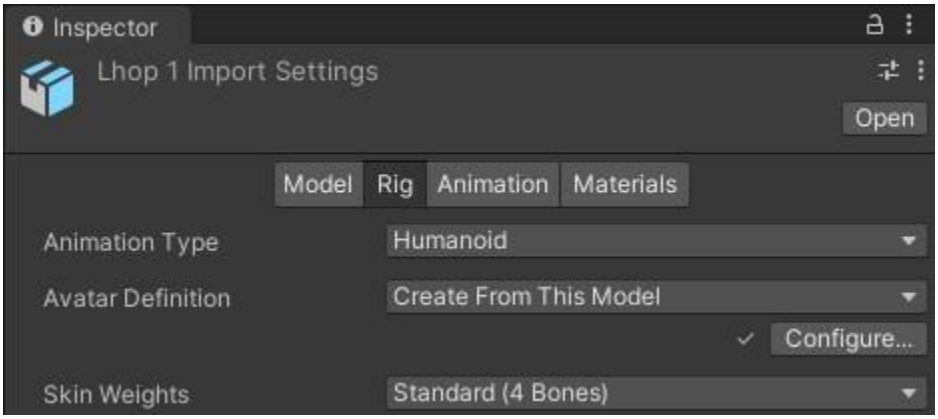

Figure 5-6 Rig settings for an L-Hop Xsens file sample

3-D humanoid characters can be added to the scene and they should each hold an “Animator” component. The “Controller” element must be assigned with the “XsensController” from the “AnimatorControllers” folder.

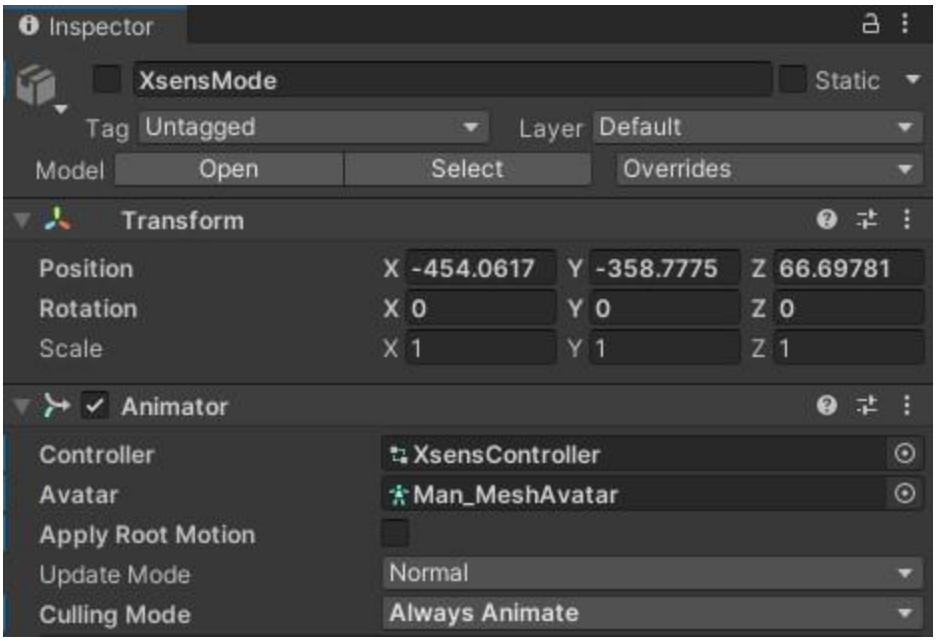

Figure 5-7 XsensController is assigned as the controller for the XsensMode prefab

Populating the animation files into the state machine happens exactly the same as in the FBX mode.

5.1.3. Video

The Video mode is vastly different than the other modes since it does not have a 3-D character model as it plays 2D videos from a set of files instead.

In order to add videos in this mode, a proper naming convention must be agreed upon first. The naming convention which was chosen for the sample files of this application is as follows:

“TaskName”+”TaskIndexNumber”+”TopLeft”.mp4

“TaskName”+”TaskIndexNumber”+”TopRight”.mp4

“TaskName”+”TaskIndexNumber”+”BottomLeft”.mp4

“TaskName”+”TaskIndexNumber”+” BottomRight”.mp4

All of the movement videos must be on the same path in the running system. The path must be added (or edited) in the “LocalVideoController.cs” script.

```
32      void Update()
33      {
34          if (_updateVideoPathFlag == true && LocalTaskSelection.taskSelectedFlag == true)
35          {
36              videoPlayerTopLeft.GetComponent<VideoPlayer>().url = videoFolderPath + LocalTaskSelection.taskSelected + currVideoIndex.ToString() +"TopLeft.mp4";
37              videoPlayerTopRight.GetComponent<VideoPlayer>().url = videoFolderPath + LocalTaskSelection.taskSelected + currVideoIndex.ToString() +"TopRight.mp4";
38              videoPlayerBottomLeft.GetComponent<VideoPlayer>().url = videoFolderPath + LocalTaskSelection.taskSelected + currVideoIndex.ToString() +"BottomLeft.mp4";
39              videoPlayerBottomRight.GetComponent<VideoPlayer>().url = videoFolderPath + LocalTaskSelection.taskSelected + currVideoIndex.ToString() +"BottomRight.mp4";
40
41              _updateVideoPathFlag = false;
42          }
43
44          GameObject.Find("AnimationName").GetComponent<UnityEngine.UI.Text>().text = LocalTaskSelection.taskSelected + currVideoIndex.ToString();
45      }
```

Figure 5-8 Video naming convention and flag checking in LocalVideoController.cs

After the task selection menu, if the video mode has been chosen, two different flags of “LocalTaskSelection.taskSelectedFlag” and “_updateVideoPathFlag” will be assigned the “true” value which will update the path of the movement videos only once. Then by clicking on the next or previous movement selection they get updated one time each as well.

```
0 references
public void LungeSelect()
{
    taskSelected = "lunge";
    taskSelectedFlag = true;
    Canvas_LocalTaskSelection.SetActive(false);

    if (LocalModeSelection.modeSelected == "VideoMode")
    {
        VideoComponents.SetActive(true);
        LocalVideoController.SetActive(true);
    }
}
```

Figure 5-9 LocalTaskSelection.taskSelectedFlag gets updated after choosing any tasks

```
0 references
public void PlayVideoNext()
{
    if (currVideoIndex < taskVideoCount)
    {
        currVideoIndex++;

        _updateVideoPathFlag = true;
    }

}

0 references
public void PlayVideoPrevious()
{
    if (currVideoIndex > 1)
    {
        currVideoIndex--;

        _updateVideoPathFlag = true;
    }
}
```

Figure 5-10 _updateVideoPathFlag gets updated by going to the next or previous video

Every time this happens the script will search for the specific requested videos and put the absolute path of each video in the “URL” element of the “Video Player” component of each of the four video player game objects. Then, it plays all the four movement videos at once.

6. ONLINE VERSION OF THE UNITY APPLICATION

It is possible to implement the same application as a web application that runs on any supported browser. This could help with remote raters that might not be available for an in-person rating session or are not willing to install the Unity application on their system. Another helpful feature of an online application is that updating and customizing it is easier as only one version of the application needs to be updated and it would be the same for everyone.

The online version of the Unity Application needs a server to be deployed upon and a database to handle the login system and the scoring system. The suggested server for deployment is an Apache server on an Ubuntu OS. The preferred database server would be a MariaDB Server which can be administrated by the phpMyAdmin tool.

To use the application in the online mode a log-in system must be implemented so that users can sign up and log in with their credentials. This will help to save and retrieve all the information about each user and rater such as their scores for different movements of different tasks in different modes.

A valid domain address with SSL certification is also needed for proper web application deployment. A domain without SSL certification will rise warnings to users and is not ideal for the application.

Additional information could also be stored in the database beside the required data such as user information and scores for each movement. This information could include the number of times an animation of a movement was replayed or the exact time that an animation was played and the duration between when the animation was played and scored. This could help the researchers assess the rating process in more detail.

The web version of the Unity Application would be compiled in the WebGL platform. This could raise some issues for different operating systems and Internet browsers which does not fully support WebGL. Another concern is the performance fidelity which cannot be fully guaranteed on every system. The performance settings must be chosen appropriately to not overload the average system.