



This project has received funding from the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101092889, Topic HORIZON-CL4-2022-HUMAN-01-14

SHARESPACE

Embodied Social Experiences in Hybrid Shared Spaces



Project Reference No	101092889
Deliverable	D4.3. Data-Driven Animation of SHARESPACE Avatars
Workpackage	WP4: Interaction-Aware Avatar Animation and Rendering
Nature	DEM (Demonstrator)
Dissemination Level	PU - Public
Date	5 June 2024
Status	V1.0
Editor(s)	Stéphane Donikian, Yann Pinczon du Sel, Sébastien Maraux, Bastien Arcelin (Golaem)
Involved Institutions	Golaem, UM, CYENS, INRIA
Document Description	This document illustrates the workflow developed to animate avatars from the different streamed data.

CONTENTS

List of Tables.....	4
1 Introduction.....	5
1.1 Purpose of the document.....	5
1.2 Structure of the document.....	5
2 Overview of the approach.....	6
3 Preparation of the scene in Maya.....	8
3.1 Introduction.....	8
3.2 MotionStreaming Behavior.....	8
3.3 Simulation Export.....	10
4 Unreal Project.....	12
4.1 Introduction.....	12
4.2 Golaem Simulation.....	12
4.3 Pixel Streaming BVH.....	13
4.4 BVH to Golaem Simulation.....	16
4.4.1 Introduction.....	16
4.4.2 Configuration.....	17
4.5 Audio to Golaem Simulation.....	20
4.5.1 Introduction.....	20
4.5.2 Audio2Rig Blueprint parameters (standard configuration).....	21
4.5.3 Audio2Rig blueprint Init detail.....	21
4.5.4 Audio2Rig blueprint inference detail.....	22
4.5.5 Audio2Rig blueprint debug detail.....	22
4.5.6 Audio2Rig blueprint Find Skeletal Mesh detail.....	23
4.5.7 Audio2Rig blueprint Set Morph Target detail.....	23
5 Testing the application.....	24
6 Modifying the project.....	25
6.1 Creating a new character.....	25



- 6.2 Adding a character in a project.....29
- 6.3 Modifying the streamed BVH data before animation.....30
- 7 Retargeting animations for the Sport Scenario33
 - 7.1 Preparing a character file for animation conversion33
 - 7.1.1 Initial setup.....33
 - 7.1.2 New morphology on the same skeleton34
 - 7.2 Converting the animation.....35
 - 7.3 Preparing a character file for replay36
 - 7.4 Replaying the animation on another character38
 - 7.5 Export the retargeted animation39
- 8 Conclusions41
- Appendix 1: Unreal tips and tricks.....42

LIST OF TABLES

Table 1: List of Abbreviations4

Table 1: List of Abbreviations

Term / Abbreviation	Definition
Mocap	Motion Capture
VR	Virtual Reality
XR	eXtended Reality
HMD	Head Mounted Display
GDA	Golaem Digital Asset
GCHA	Golaem CHAracter file
GMO	Golaem MOTion file
BVH	BioVision Hierarchy

1 INTRODUCTION

1.1 PURPOSE OF THE DOCUMENT

This deliverable presents the direct one-to-one animation of the SHARESPACE virtual human, suitable for the direct animation of L1 avatars. Facial expressions of the avatar are animated using a combination of lip syncing driven by the participants' speech and gaze control driven by eye tracking data when those data will be available. The avatar animation is driven by encoded motion capture data (WP3). For the peloton scenario, motion capture of the bike should also be provided.

1.2 STRUCTURE OF THE DOCUMENT

This document is structured as follows:

- Section 2 of this document provides an overview of the different steps needed to animate an avatar in an Unreal application.
- Section 3 is dedicated to the preparation of the scene in Maya.
- Section 4 is devoted to presenting the creation and configuration of the scene, in Unreal.
- Section 5 explains how to test the application with live or emulated streams.
- Section 6 illustrates how to create and add a new character in the project and also how to modify the streamed mocap data before the animation to integrate for example the result of the Cognitive Architecture (WP5).
- Section 7 explains how to retarget animations for the Sport Scenario.
- Section 8 concludes this document.

2 OVERVIEW OF THE APPROACH

To illustrate the creation and animation pipeline, we have developed a SHARESPEACE sample project available for all SHARESPEACE partners that may need it. The purpose of this documentation is to cover only the parts that are specific to the SHARESPEACE project, as this is based on the use of Golaem existing commercial products.

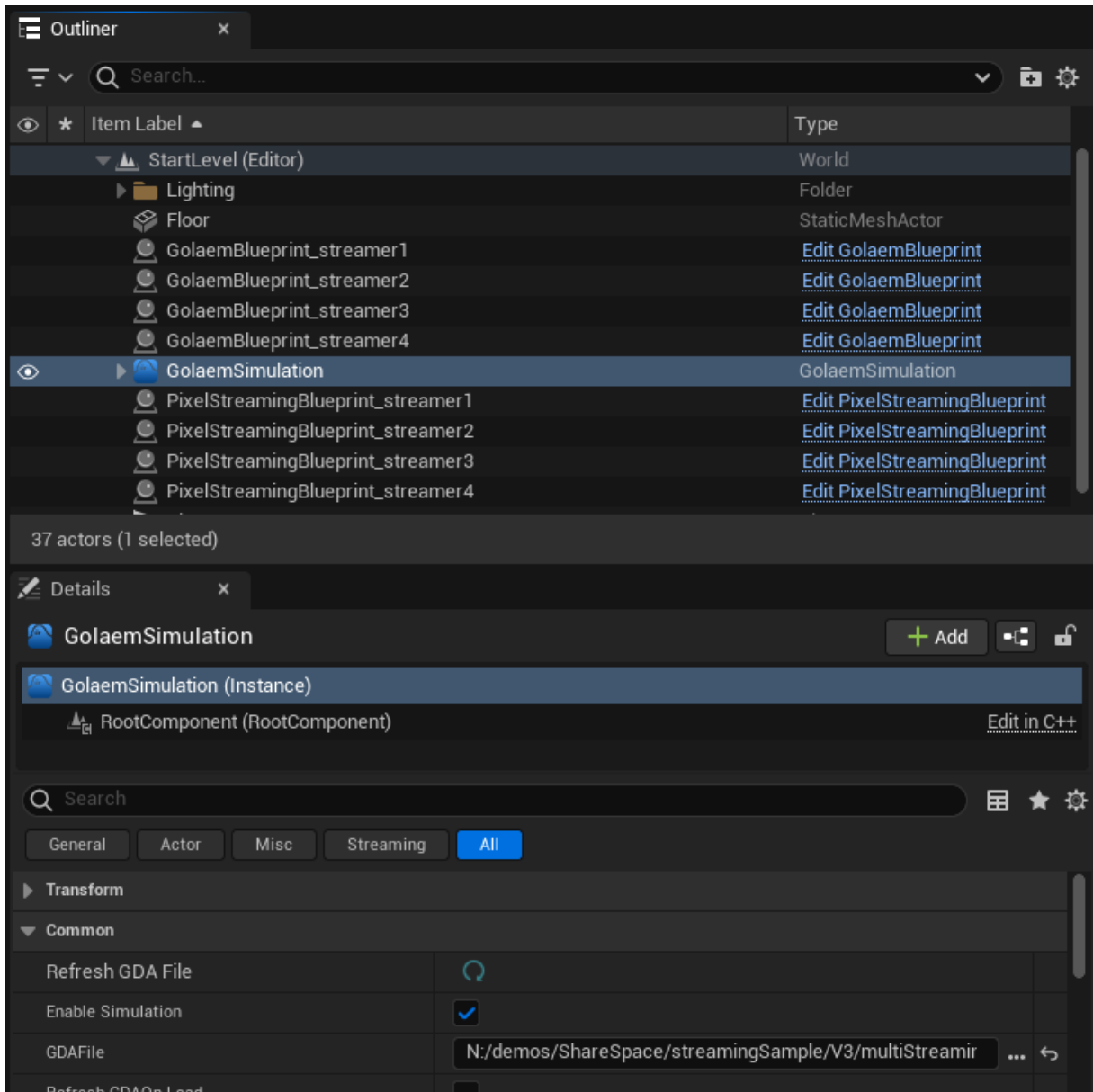
- The usage of Golaem for Maya to prepare and export a GDA file are documented here: <https://golaem.com/content/doc/golaem-crowd-documentation/golaem-crowd>
- The usage of Golaem For Unreal to replay the simulation in Unreal is documented here: <https://golaem.com/content/doc/golaem-crowd-documentation/golaem-unreal>
Note that the Sharespace sample project replays a simulation in Unreal (not a cache): <https://golaem.com/content/doc/golaem-crowd-documentation/overview-7>, which works through the export of a GDA file (<https://golaem.com/content/doc/golaem-crowd-documentation/golaem-engine-exporter>)

The SHARESPEACE sample project comes with:

- The GolaemCrowd maya plugin that is specific to SHARESPEACE (includes the bvhStreamingBehavior that is not publicly available) – ***This is useful only for SHARESPEACE partners that need to edit the simulation;***
- The corresponding GolaemForUnreal plugin - ***This is useful only for SHARESPEACE partners that need to create new simulations;***
- The maya scene that is used to create the simulation – ***It contains the simulation, and it is useful for all SHARESPEACE partners;***
- The unreal project that is ready to use and customize – ***It contains the unreal project that replays the simulation, and it is useful for all SHARESPEACE partners.***

To launch the simulation:

- Extract the *GolaemForSharespace_V5* unreal project and the *GolaemMayaScene* somewhere on one disk of your computer;
- Add the **GLMCROWD_UNIT=1** environment variable in your system - <https://golaem.com/content/doc/golaem-crowd-documentation/through-operating-system>;
- Launch the *GolaemForSharespace_V5/SHSParticipant unreal project*;
- In Unreal, in the GolaemSimulation node, edit the GDAFile attribute to point to the local location on one disk of your computer of the *GolaemMayaScene\GDA\GDASceneWith5Actors\GDASceneWith5Actors.gda* file, as seen in the following screenshot:



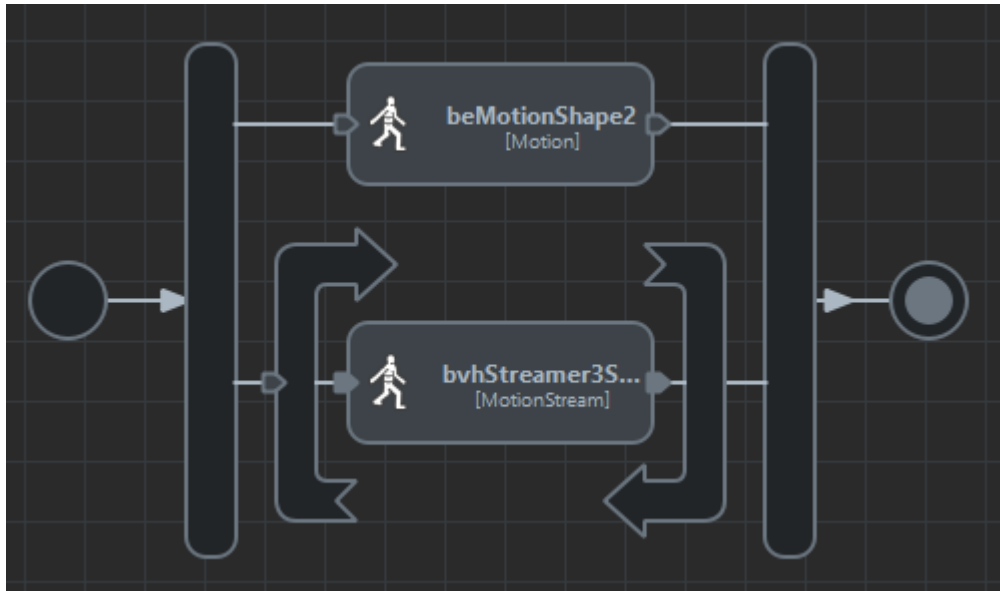
- Hit the play button, and start streaming.

3 PREPARATION OF THE SCENE IN MAYA

3.1 INTRODUCTION

It is necessary to use Golaem for Maya to build the simulation within Maya:
<https://golaem.com/content/doc/golaem-crowd-documentation/golaem-crowd>

The behavior graph for each streamer might look like that:



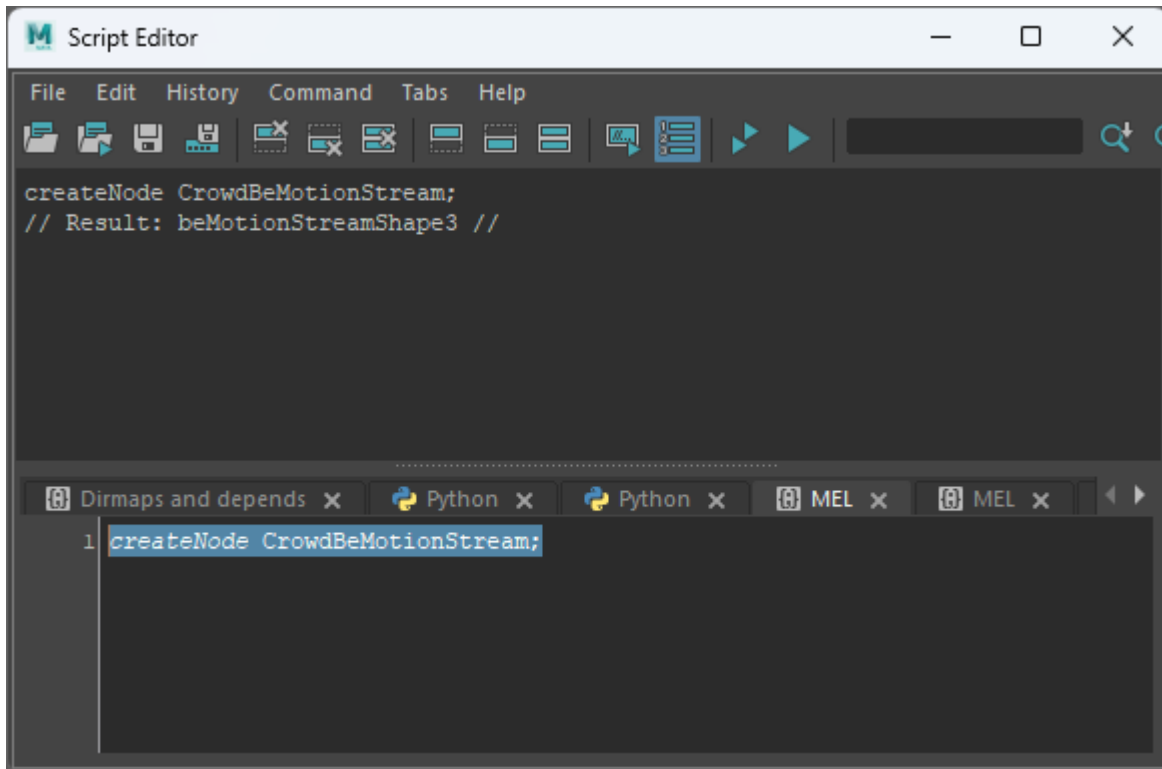
With:

- A default animation that is played when no streaming is connected;
- A BVH streaming behavior that plays the streaming animation with a higher priority than the default behavior, and with start and stop triggers that are set whenever a streamer is connected/disconnected.

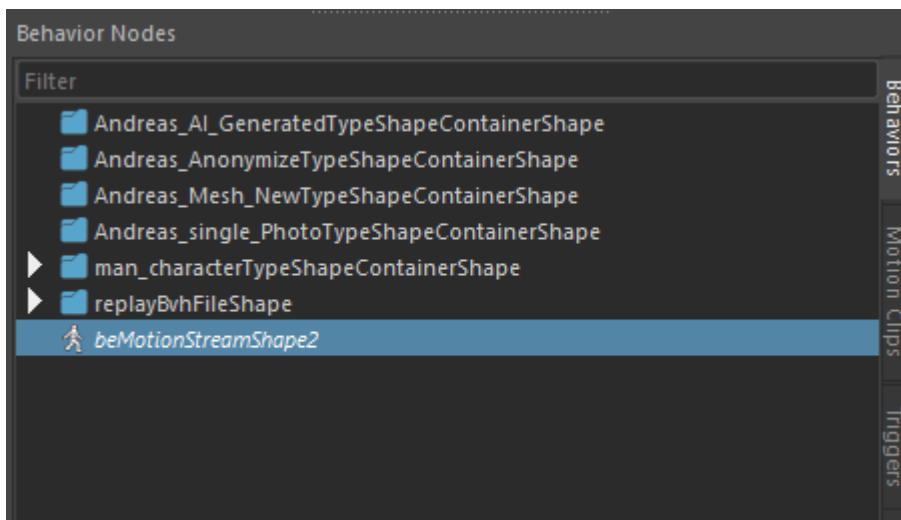
3.2 MOTIONSTREAMING BEHAVIOR

When an entity needs to use the streaming capability, use the hidden MotionStreaming behavior by typing the following commands in the Script Editor:

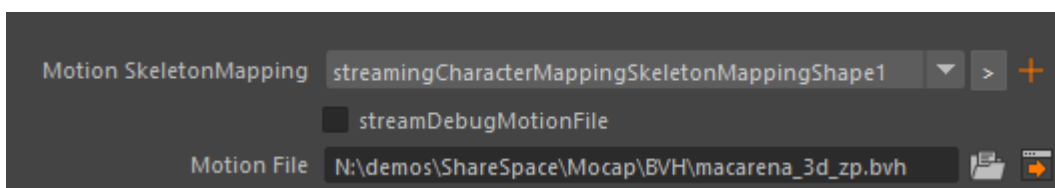
```
createNode CrowdBeMotionStream;
```

The motion will be created and available in the Behavior Nodes panel of the Golaem behavior editor. It can now be dragged and dropped in the behavior of your choice.



The stream behavior is configured like a motion behavior, except for the motion file part, which will come from the streaming instead. The streaming part cannot be tested directly within Maya, but a debug BVH file can be played to check that it is configured correctly:



When replaying a BVH, it is important to configure the Motion SkeletonMapping to configure how the motion's skeleton will be interpreted by Golaem.

For a better understanding of this part, the documentation of the Motion Skeleton Mapping can be checked here:

<https://mayacrowd.com/content/doc/golaem-crowd-documentation/motion-skeleton-mapping>

3.3 SIMULATION EXPORT

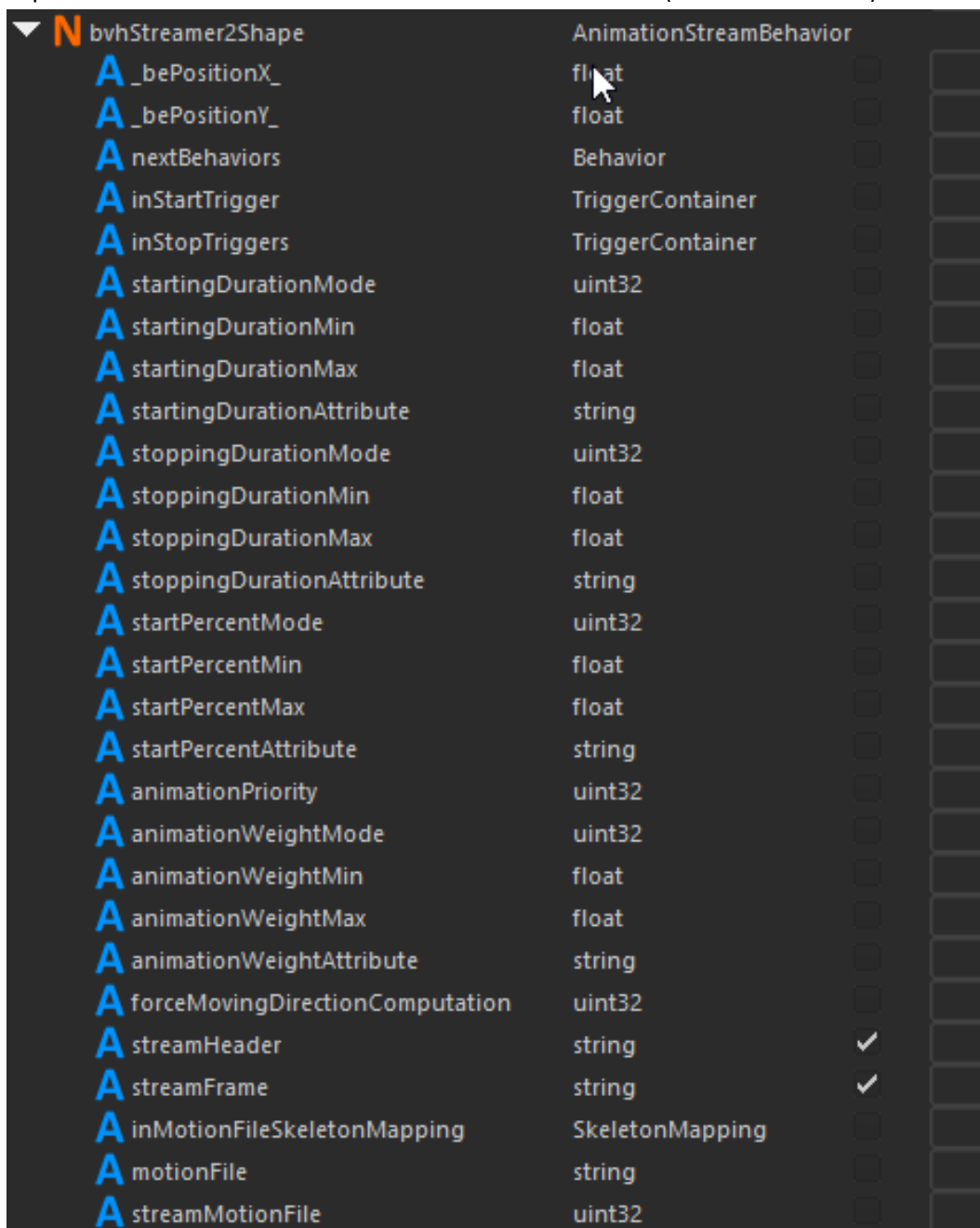
The simulation should then be exported using the GDA exporter:

<https://mayacrowd.com/content/doc/golaem-crowd-documentation/golaem-engine-exporter>

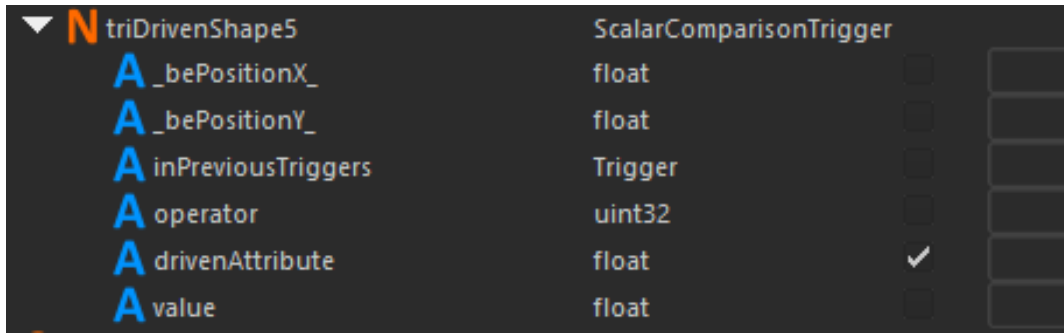
Make sure to set some attributes as public to be able to set them in the Unreal project

(<https://golaem.com/content/doc/golaem-crowd-documentation/gda-attributes-window>):

- Export the BVH streamHeader and streamFrame attributes (for each streamer)



- Export the triggers driven attributes (both start and stop for each streamer):



Important notice:

In the case of the multistreamer sample, it is important to name exported behaviors and triggers in a similar way for each streamer to allow duplicating and configuring the blueprint scripts once in unreal. For instance, here the behaviors and triggers are named:

- *streamerXbvhhBeh*
- *streamerXStartTrigger*
- *streamerXStopTrigger*

Which will export the GDA attributes in:

- *streamerXbvhhBeh.streamHeader*
- *streamerXbvhhBeh.streamFrame*
- *streamerXStartTrigger.drivenAttribute*
- *streamerXStopTrigger.drivenAttribute*

Once in Unreal, it is easy to separate each streamer by simply concatenating the streamer name with each attribute.

4 UNREAL PROJECT

4.1 INTRODUCTION

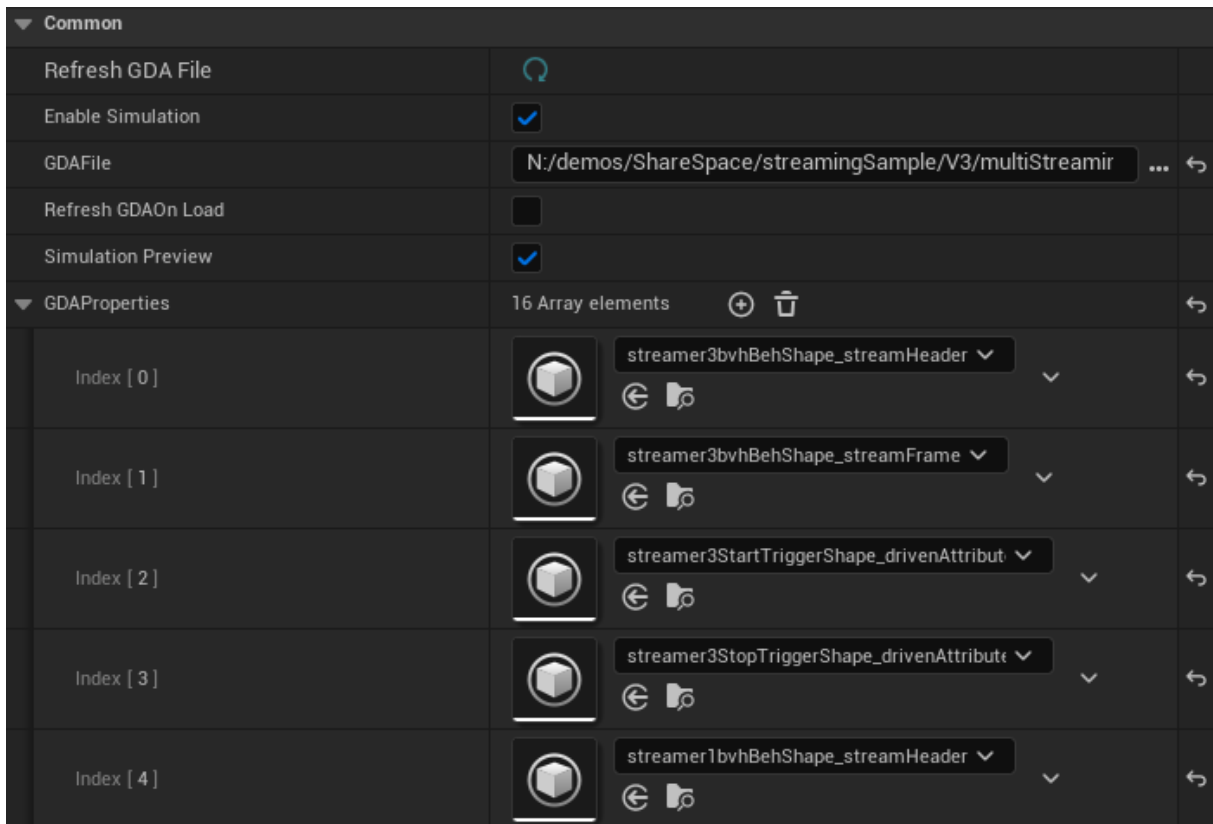
The Unreal scene is made of three main parts:

- The GolaemSimulation node, that replays the GDA file prepared in the previous step;
- The pixel streaming blueprint, that connects to the streamed animation;
- The Golaem blueprint that connects the Golaem simulation with the streaming.

4.2 GOLAEM SIMULATION

The following tutorial explains how to configure a Golaem Simulation replaying the GDA file into Unreal: <https://golaem.com/content/doc/golaem-crowd-documentation/overview-7>

Once correctly loaded, the Golaem simulation node should show the public GDA properties that can be overridden:



4.3 PIXEL STREAMING BVH

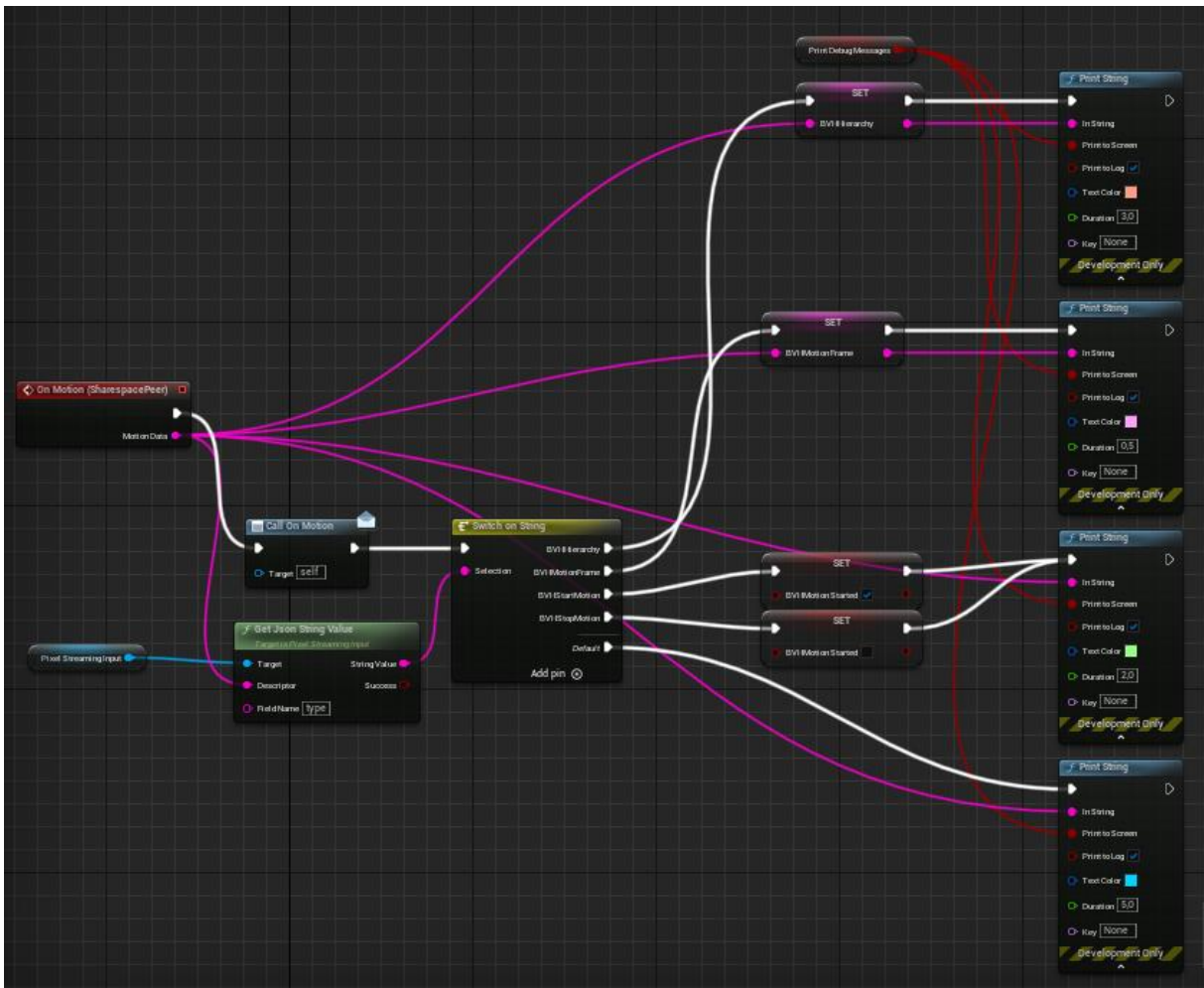
Each streamer is a different instance of the same Pixel Streaming blueprint, configured to connect with a configured streamer name (streamer1, streamer2, streamer3, streamer4):

The screenshot displays the Unreal Engine interface. At the top, a list of actors is shown: PixelStreamingBlueprint_streamer1, PixelStreamingBlueprint_streamer2, PixelStreamingBlueprint_streamer3, PixelStreamingBlueprint_streamer4, and PlayerStart. Each instance has a corresponding 'Edit PixelStreamingBlueprint' link.

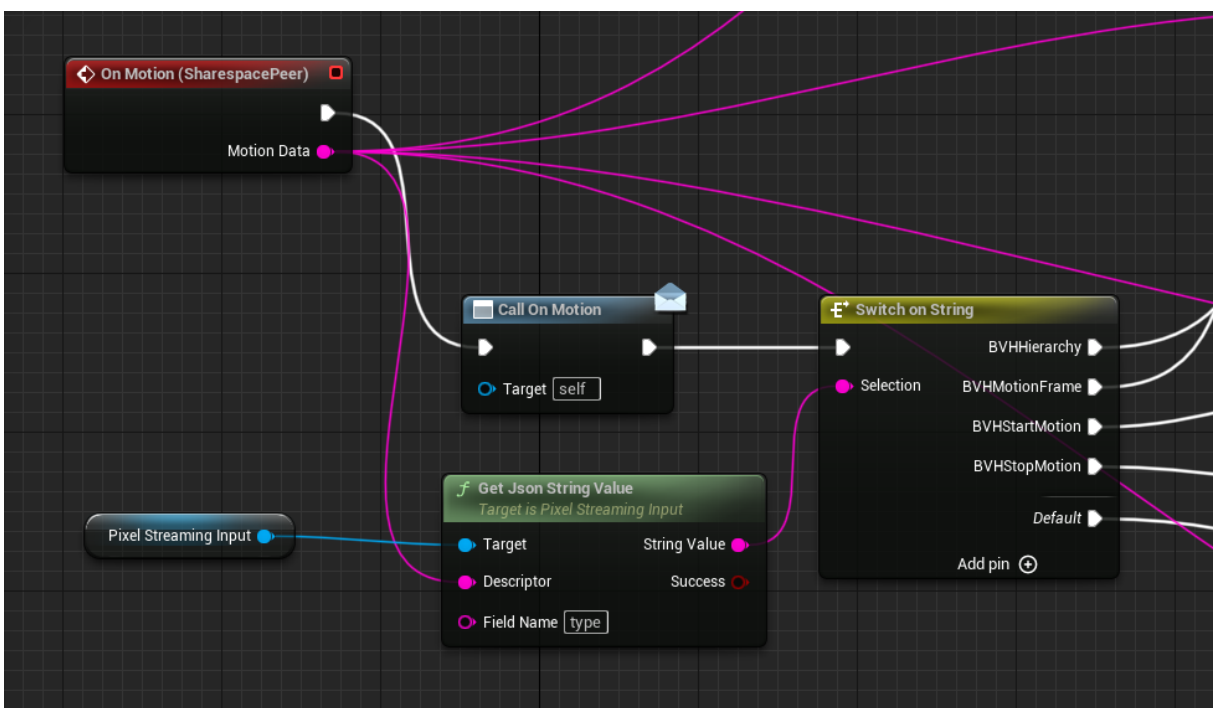
Below this, a 'Details' panel is open for 'PixelStreamingBlueprint_streamer1'. The panel shows the actor's name and a search bar. Below the search bar are tabs for 'General', 'Actor', 'Misc', 'Streaming', and 'All'. The 'Transform' section is expanded, showing location (0,0,0), rotation (0,0°,0,0°), and scale (1,0,1). The 'Default' section is also expanded, showing the following configuration:

Property	Value
Server URL	wss://dev-pixelstreaming.openrainbow.io:443/
Streamer	streamer1
Print Debug Messages	<input type="checkbox"/>
Print Start Stop Messages	<input checked="" type="checkbox"/>

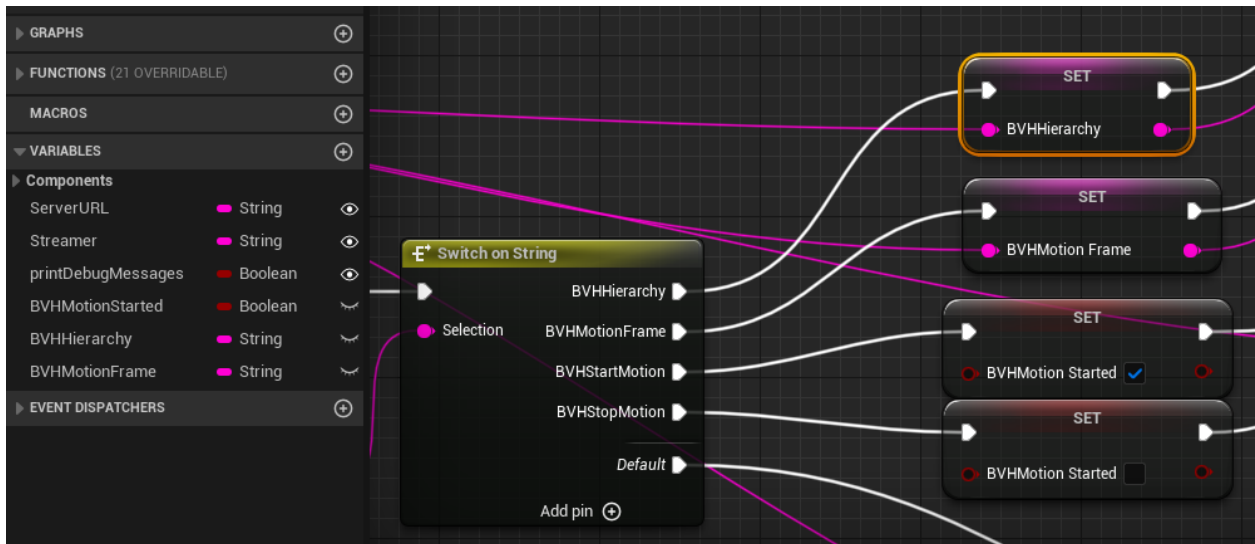
The Pixel Streaming blueprint looks like this:



More in details, it is mainly the stream receiver that check the BVH message type and switch depending on it:



And depending on the received type, it simply fills different variables:



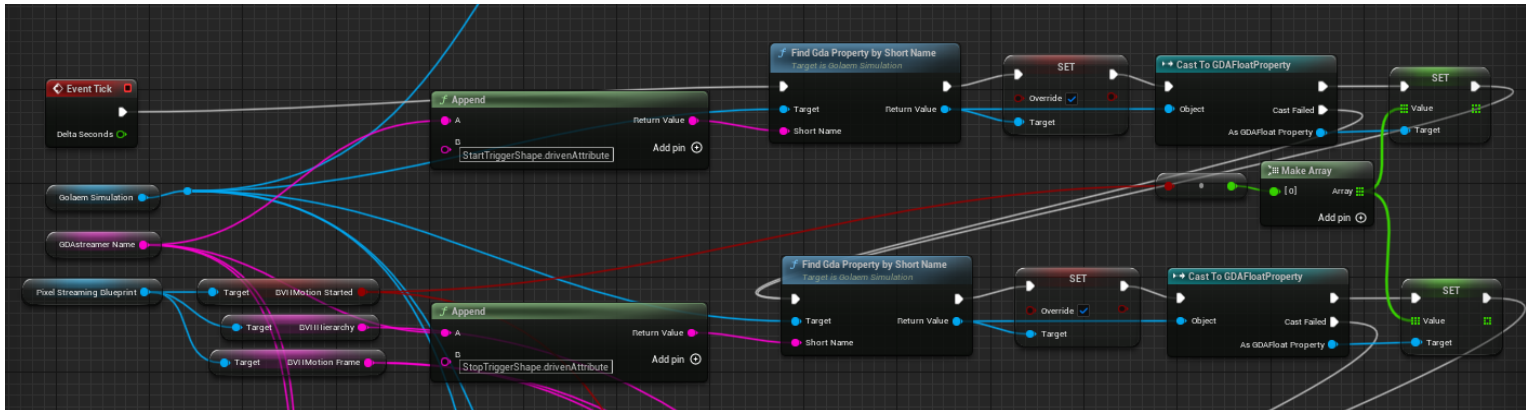
- *BVHMotionStarted* will be set to true when a *BVHStartMotion* message is received, and back to false when a *BVHStopMotion* message is received;
- *BVHHierarchy* will hold the content of the last *BVHHierarchy* message that was received;
- *BVHMotionFrame* will hold the content of the last *BVHMotionFrame* message that was received.

4.4 BVH TO GOLAEM SIMULATION

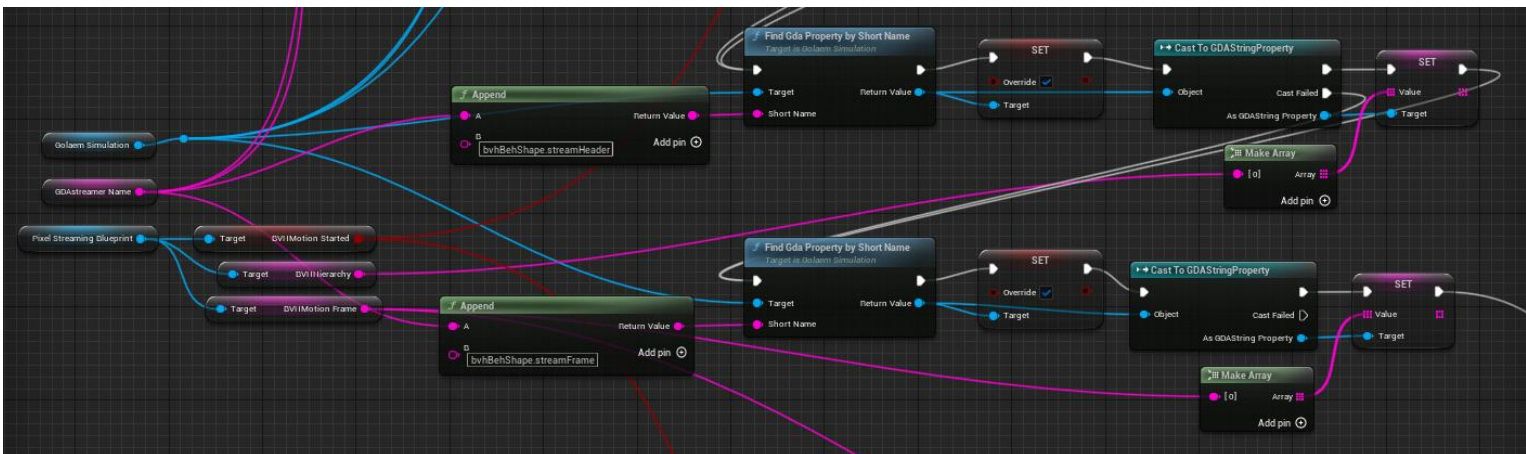
4.4.1 Introduction

The link between the Pixel Streaming BVH and the Golaem Simulation is done in the Golaem blueprint. Its purpose is to bring the variables that were stored in each blueprint in the Golaem GDA properties that were prepared for this purpose during the preparation of the scene in Maya.

This is the part that links the *BVHMotionStarted* variable to the start and stop triggers that were defined in the Maya scene, and exported in the GDA:

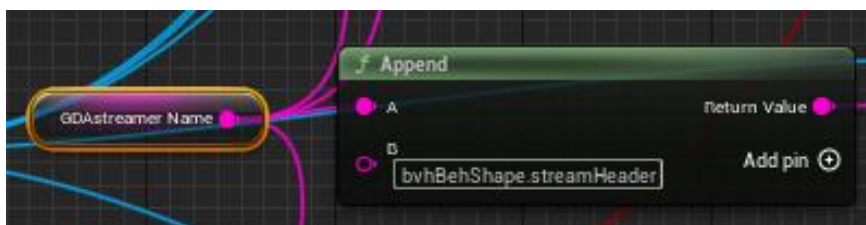


This is the part that links the *BVHHierarchy* and *BVHMotionFrame* variables to the streaming behavior that was defined in the Maya scene and exported in the GDA:



For each part, the variables are used through a reference to the pixel streaming blueprint instance of the entity that is configured. Each entity in the scenario needs to see its streaming blueprint linked to the Golaem simulation with the correct GDA attribute name.

As each GDA attribute name was prefixed with the steamer name in the Maya scene, it is now easy to concatenate a public blueprint variable (to configure with the steamer name) with the end of the GDA attribute:





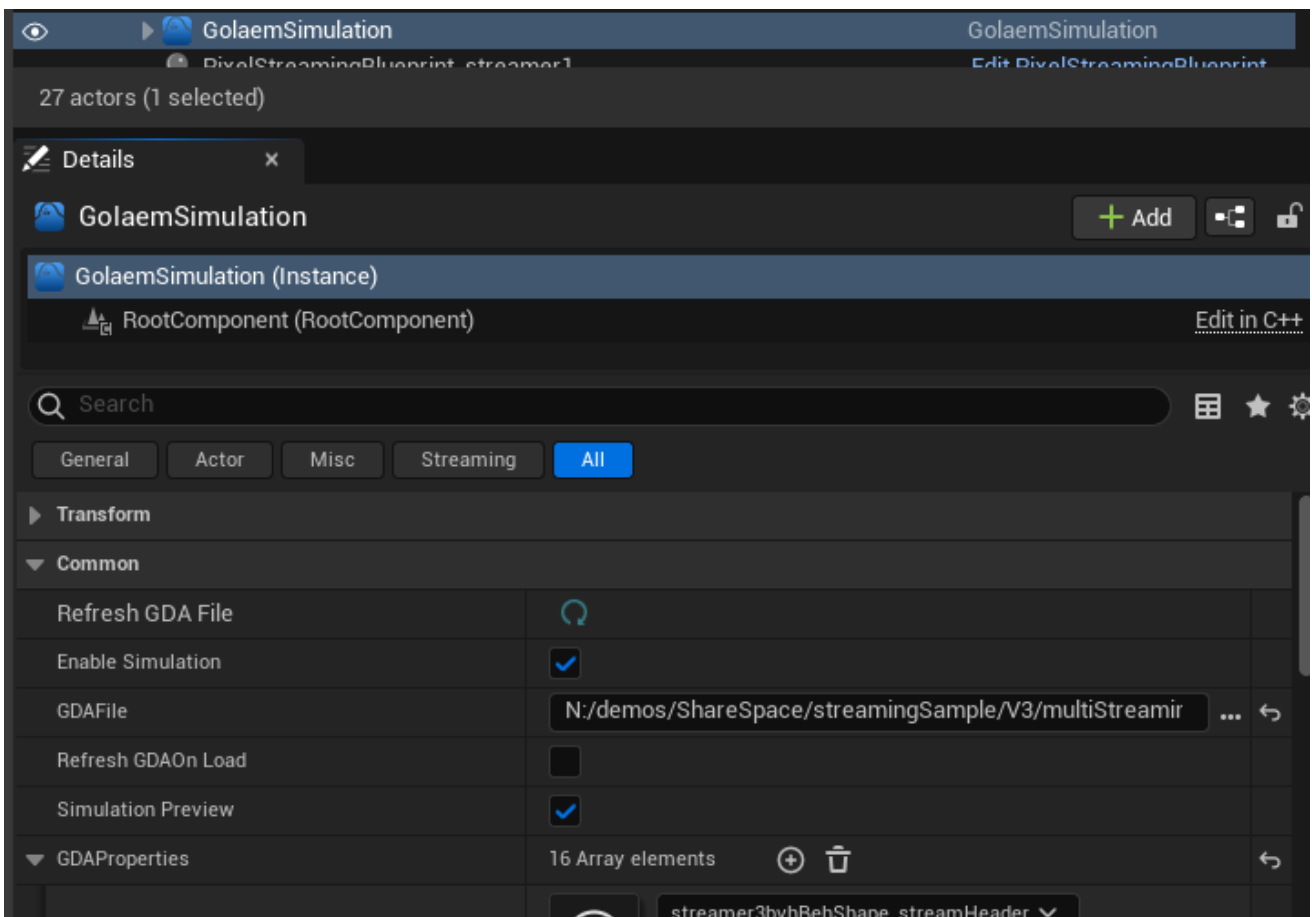
This is used everywhere a GDA attribute has to be found, which allows each instance of the blueprint to be correctly configured with:

- The Golaem simulation;
- The pixel streaming blueprint instance;
- The streamer name.

Note that each GDA attribute that is configured needs to be configured with the override value to true first.

4.4.2 Configuration

The GolaemSimulation node has to be configured with the correct GDA file:





Each instance of the pixel streaming (one per replay actor) needs to be configured with the correct Server URL and Streamer name:

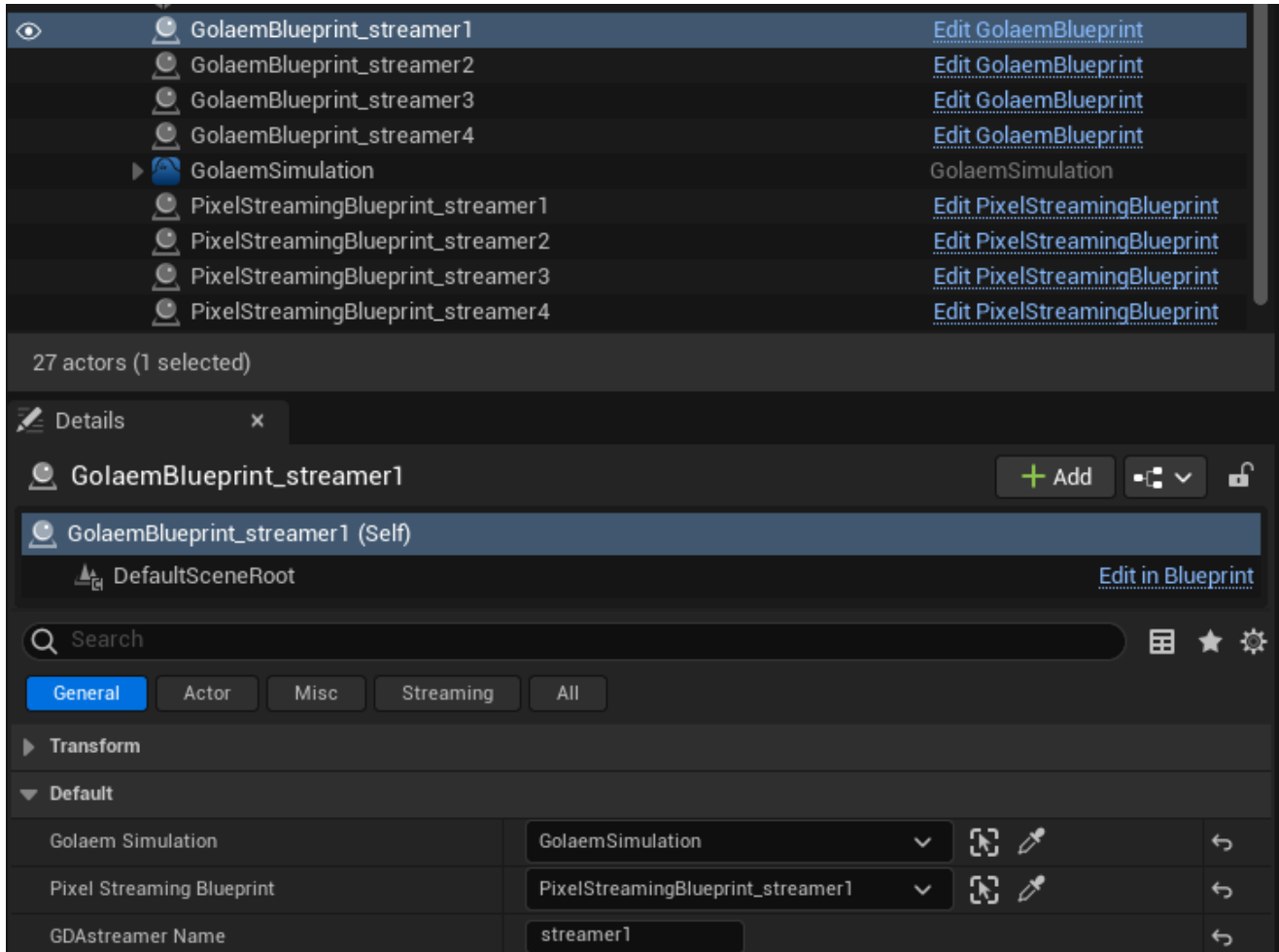
The screenshot shows the Unreal Engine interface. At the top, the Hierarchy panel lists several actors: PixelStreamingBlueprint_streamer1, PixelStreamingBlueprint_streamer2, PixelStreamingBlueprint_streamer3, PixelStreamingBlueprint_streamer4, and PlayerStart. Each actor has a corresponding 'Edit PixelStreamingBlueprint' link. Below the Hierarchy panel, the Details panel is open for 'PixelStreamingBlueprint_streamer1'. It shows a search bar, tabs for 'General', 'Actor', 'Misc', 'Streaming', and 'All', and a 'Transform' section with fields for Location (0,0,0), Rotation (0,0°,0,0°), and Scale (1,0,1,0). The 'Default' section contains the following configuration:

Property	Value
Server URL	wss://dev-pixelstreaming.openrainbow.io:443/
Streamer	streamer1
Print Debug Messages	<input type="checkbox"/>
Print Start Stop Messages	<input checked="" type="checkbox"/>

Configuration of the pixel streaming blueprint instance

Each instance of the Golaem blueprint has to be configured with the correct:

- Golaem Simulation;
- Pixel Streaming Blueprint instance;
- Streamer name.

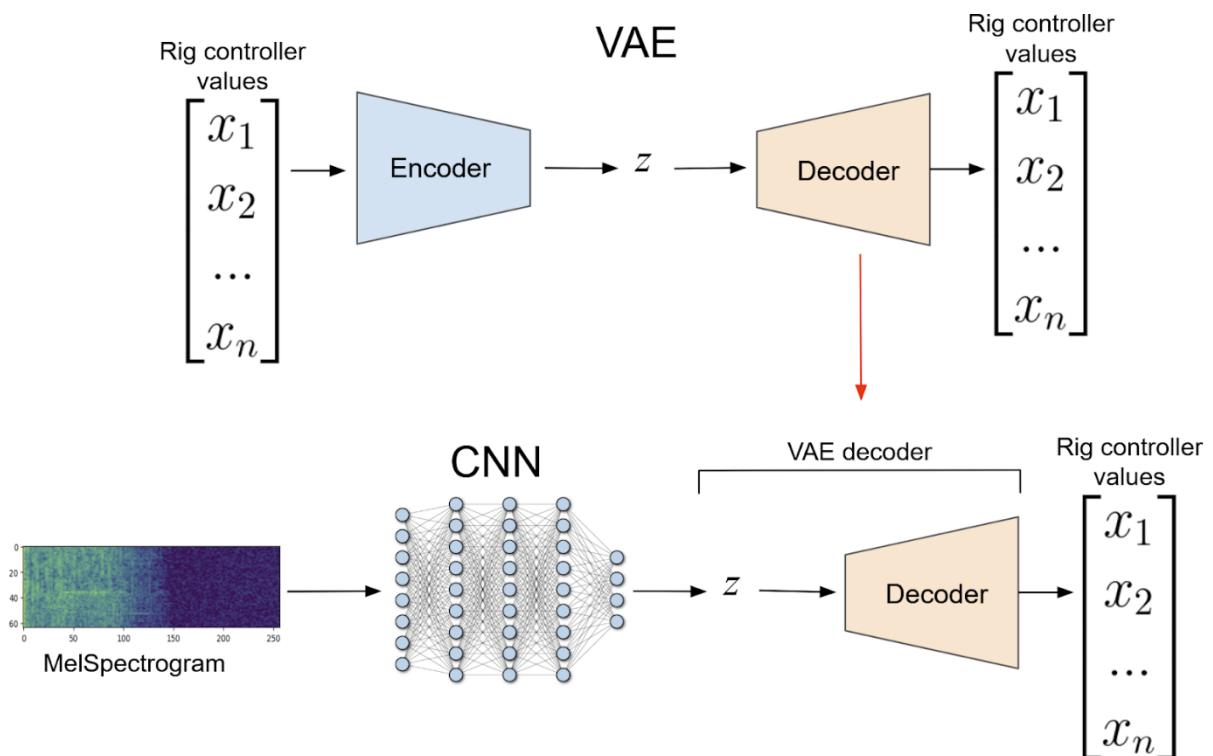


Configuration of the Golaem blueprint instance

4.5 AUDIO TO GOLAEM SIMULATION

4.5.1 Introduction

Audio lipsync is achieved via a deep learning model and Pytorch standard (beta) integration in Unreal. It is adapted for a real-time application from the work that will be presented at SIGGRAPH this year¹. It takes audio as input and outputs rig parameters to animate the avatar. The audio is transformed into a MelSpectrogram before being fed to the neural network. Two networks are used for inference. A Convolutional neural network (CNN) is used to encode audio information into a latent space, and the decoder of a trained Variational Auto-Encoder (VAE) is used to predict rig controllers values. The VAE is trained as a first step, to learn a compressed (or latent) representation of the rig controller's values parameter space.



LipSync deep learning model training architecture. First, the VAE is trained to encode and reproduce rig controller values. Then, the VAE decoder is used in another network to predict the rig controller values from a latent space, z. This latent space is inferred from the MelSpectrogram by the CNN.

To train the CNN, we generated animation on rig controllers using Polywink and audio recordings of different exercises of the Health Scenario in Spanish Language. The network is then trained to generate facial rig animation predicting a latent representation fed to the trained VAE decoder. Hence, Polywink rig controllers are directly inferred from the audio. We extracted the audio processing as well as the trained neural networks to perform live inference.

The CNN needs a 200 ms audio window to predict the current rig controllers values. This window starts 100 ms before “present” and goes 100 ms in the future. Therefore a delay parameter has been added to the sharespace audio component to match the 100 ms of data required “in the future”.

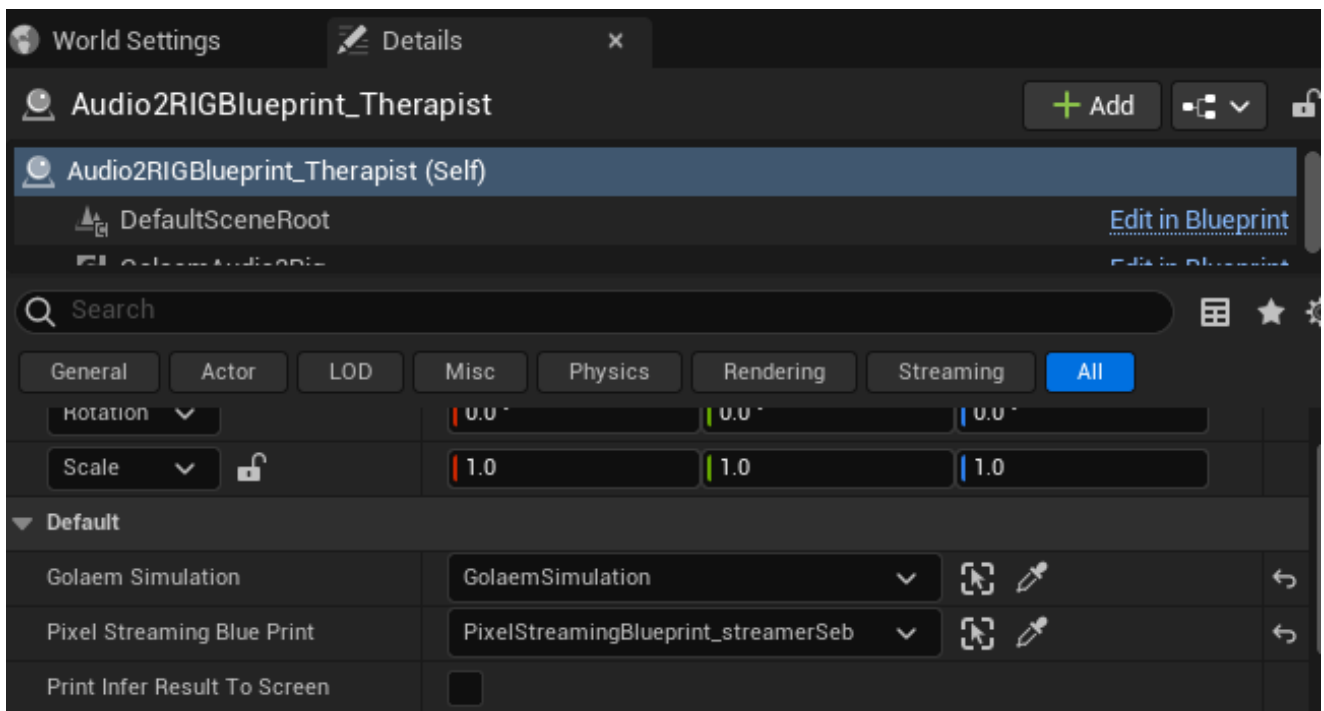
¹ Bastien Arcelin and Nicolas Chaverou. 2024. *Audio2Rig: Artist-oriented deep learning tool for facial animation*. SIGGRAPH '24 Talks: ACM SIGGRAPH 2024.

The SharespaceParticipant plugin has been augmented by Golaem to output the received audio data. It adds to the USharespaceAudioComponent some features to output the stream PCM data as Unreal data (buffer, rate and channels). It also handles a live audio delay which will be explained below. The PixelStreaming Blueprint has not been modified as the data is stored in the component and requested in the Audio2RIG blueprint.

The Unreal Project uses the Unreal 5.3 “Unreal python editor script” plugin, and the “Python Foundation Packages” which includes Pytorch, required for the inference.

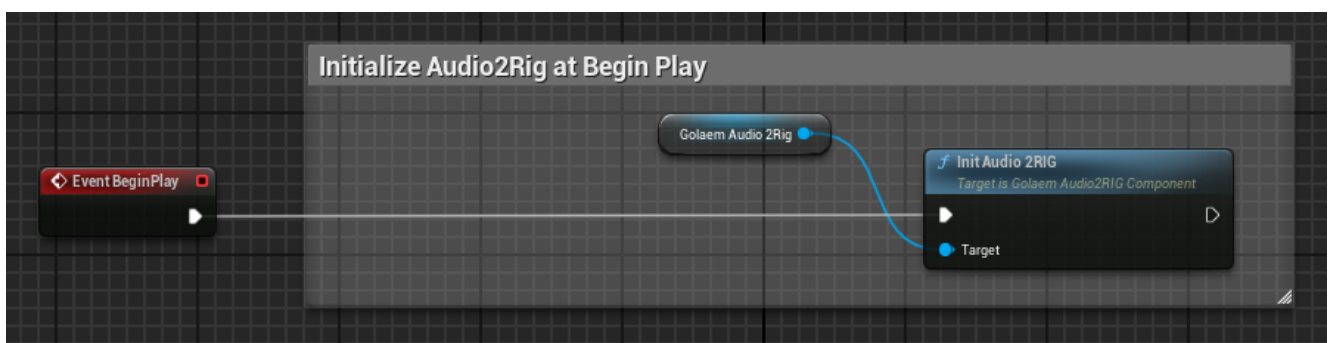
4.5.2 Audio2Rig Blueprint parameters (standard configuration)

The configuration of the Audio2Rig blueprint is straightforward, and needs a link to the Golaem Simulation and a link to the pixel streaming blueprint of the character to be lipsync animated.



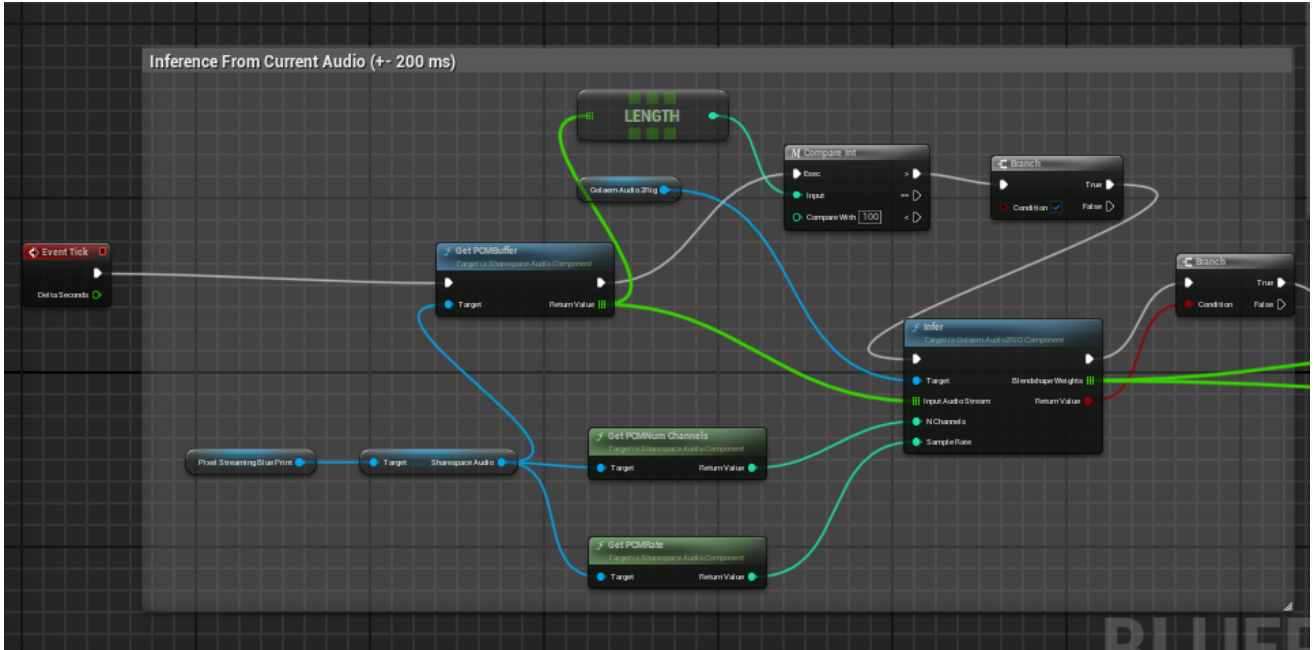
4.5.3 Audio2Rig blueprint Init detail

The Audio2Rig init consists in a single call to the init method on the component, called on Event Begin Play Unreal event. It will take care of initializing the PyTorch model.



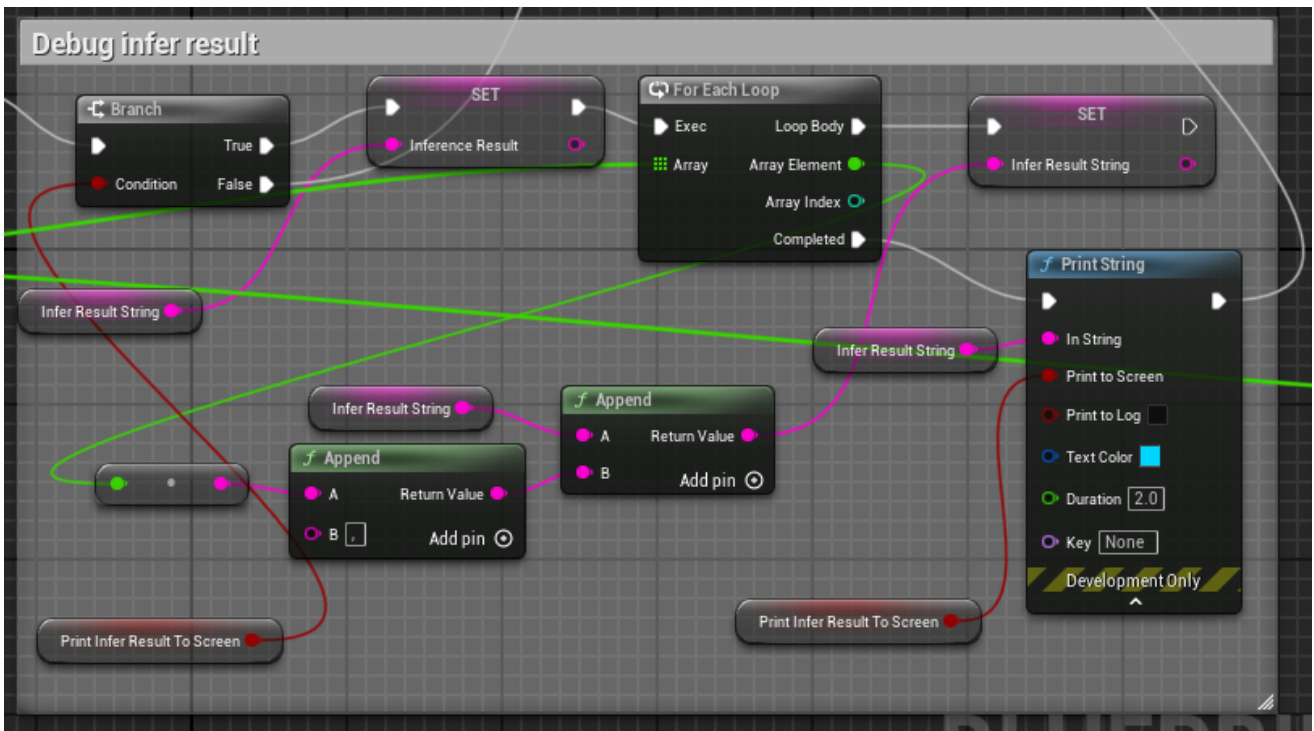
4.5.4 Audio2Rig blueprint inference detail

The inference part is called by the “Event Tick”. It will call the SharespaceAudioComponent to get PCM data and then call the Audio2Rig component for inference. A test has been added to only call inference if sufficient data has been received.



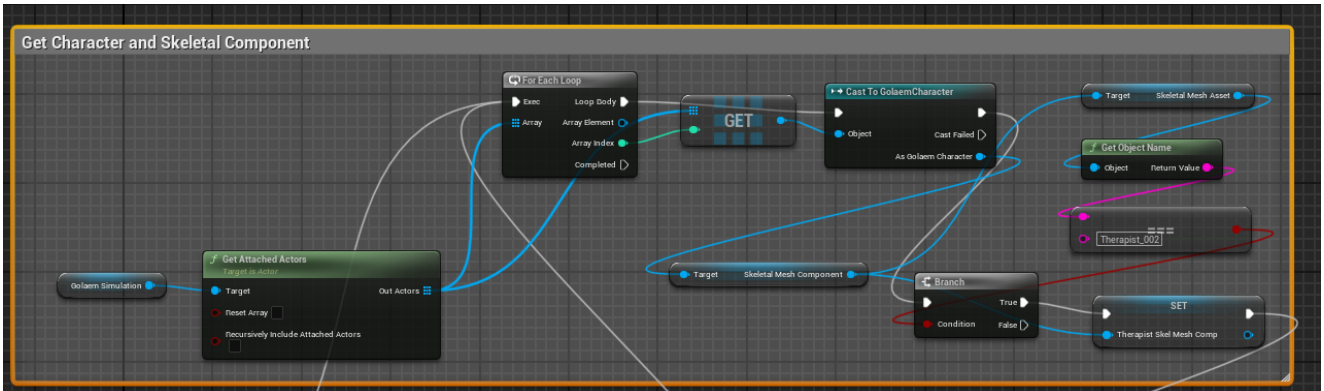
4.5.5 Audio2Rig blueprint debug detail

The blueprint also have a debug print part to display the received phonemes activation array (a list of 14 floats) which will help debug.



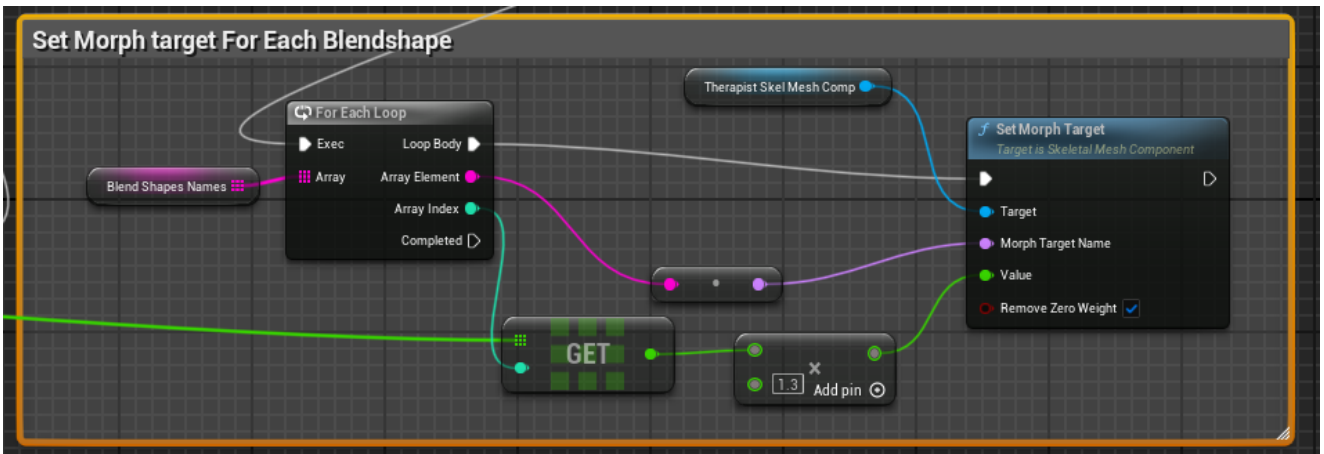
4.5.6 Audio2Rig blueprint Find Skeletal Mesh detail

In this blueprint part, the blueprint loop on all Golaem Simulation attached actors to find the Actor with a Skeletal Mesh configured with the specified name (Therapist_002 in this case)



4.5.7 Audio2Rig blueprint Set Morph Target detail

Finally, the inferred phonem weights are applied to the found skeletal mesh component, on the Morph Targets (also called blendshapes).



5 TESTING THE APPLICATION

It is possible either to use live mocap and audio streams or to emulate them thanks to tools provided by ALE partner.

The streams need to have the same streamer name between the streamer source and the one configured in the Unreal project. In the sample application streamer names are: streamer1, streamer2, streamer 3 and streamer4.

For BVH and audio streaming emulation, for each stream, you need to open a console command and type such type of command:

- Docking system:

```
docker run -it sharespacestreamer:latest -x 443 -w wss://dev-pixelstreaming.openrainbow.io -p streamer1 -S -b ./assets/physiotherapist_22Feb2024_ex6_3d.bvh -a ./assets/exercise6.wav
```

- Standalone streaming tool:

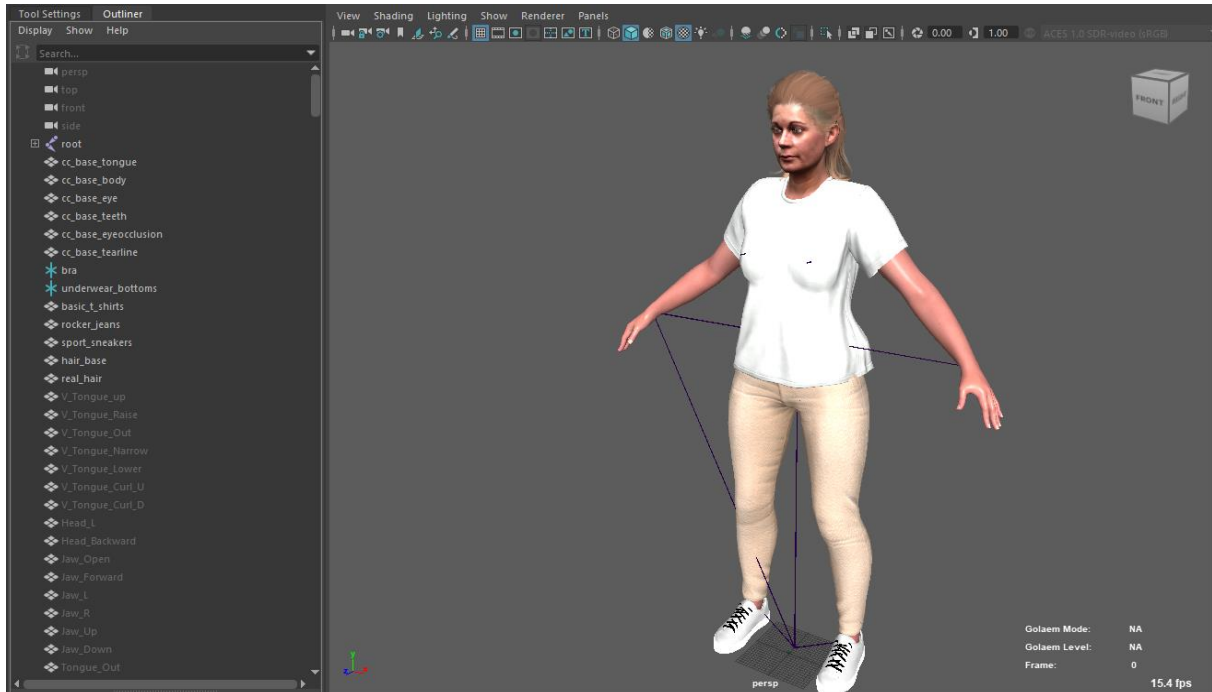
```
.\shs-streamer.bat -x 443 -w wss://dev-pixelstreaming.openrainbow.io -p streamer1 -S --bvhFile .\assets\physiotherapist_22Feb2024_ex6_3d.bvh
```

In this second case, the audio stream comes from the standard audio input of the used computer.

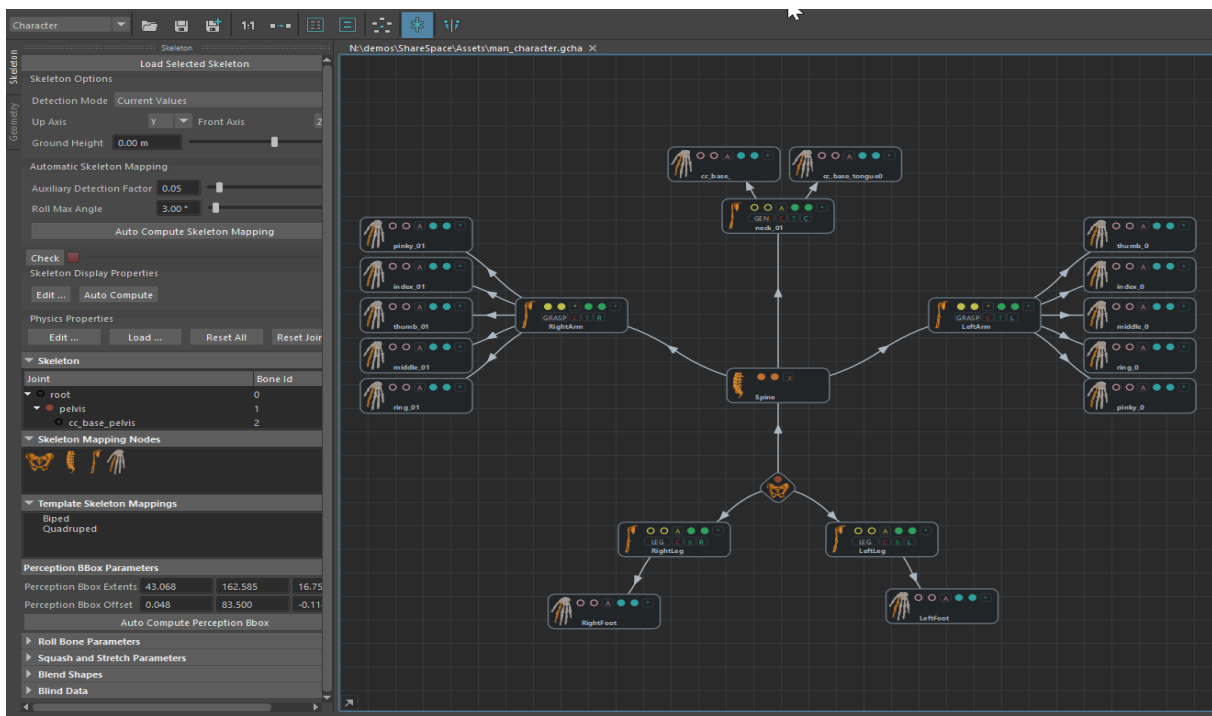
6 MODIFYING THE PROJECT

6.1 CREATING A NEW CHARACTER

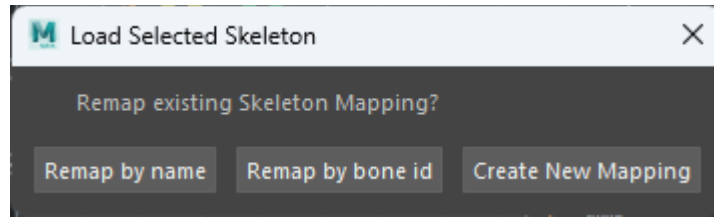
Start by importing the FBX representation file of the character into Maya, and remove any namespace that may exists:



Open the Golaem CharacterMaker and open any existing GCHA from the project (with the same bone hierarchy)



Set the detection mode to current values **Detection Mode Current Values**, select the root node in Maya, and click **Load Selected Skeleton**. When asked, choose to remap the skeleton by bone ID

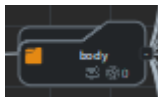


If remapping by bone id is not available, it means that the skeleton is different from the one used for the GCHA. Either make sure to get the same skeleton (try another GCHA, or make sure to use the same options while creating the character), or try to remap by name, but with the risk of obtaining a malfunctioning GCHA at the end.

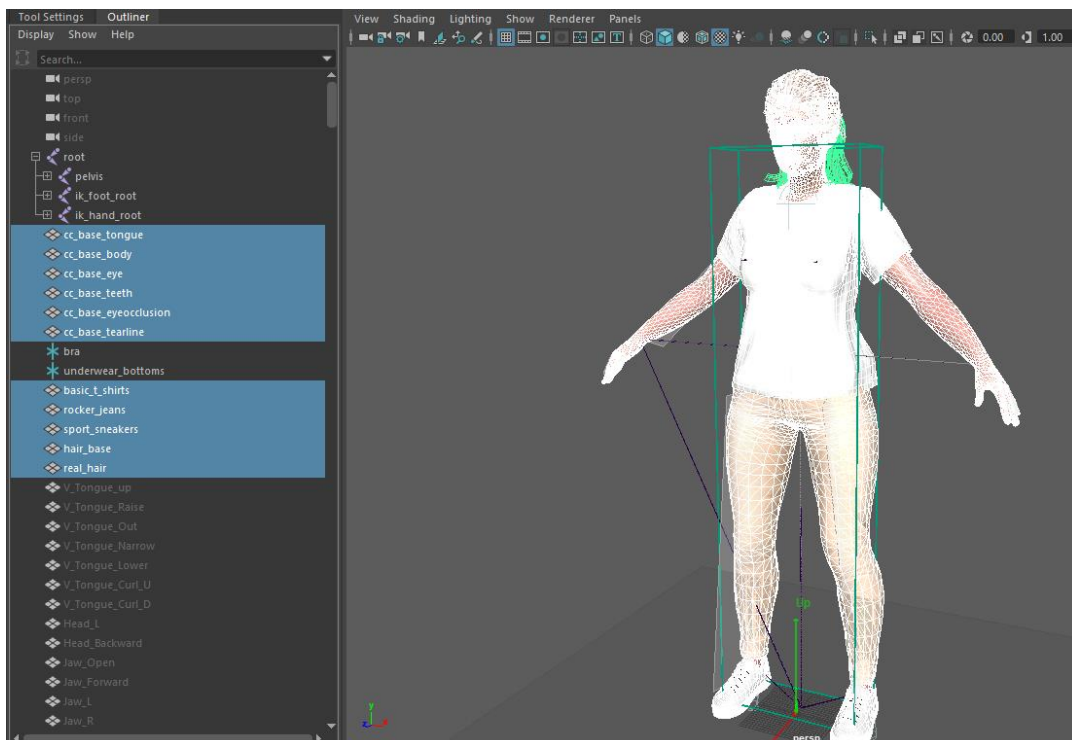
Now switch to the geometry tab.



Make sure to extend all nodes in the view, and delete everything except the container nodes (the ones



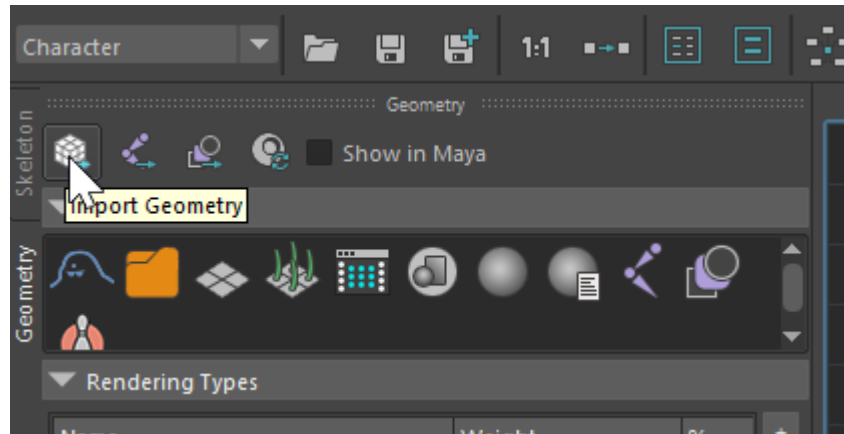
with this icon:). Now select all the geometry in the Maya Outliner:



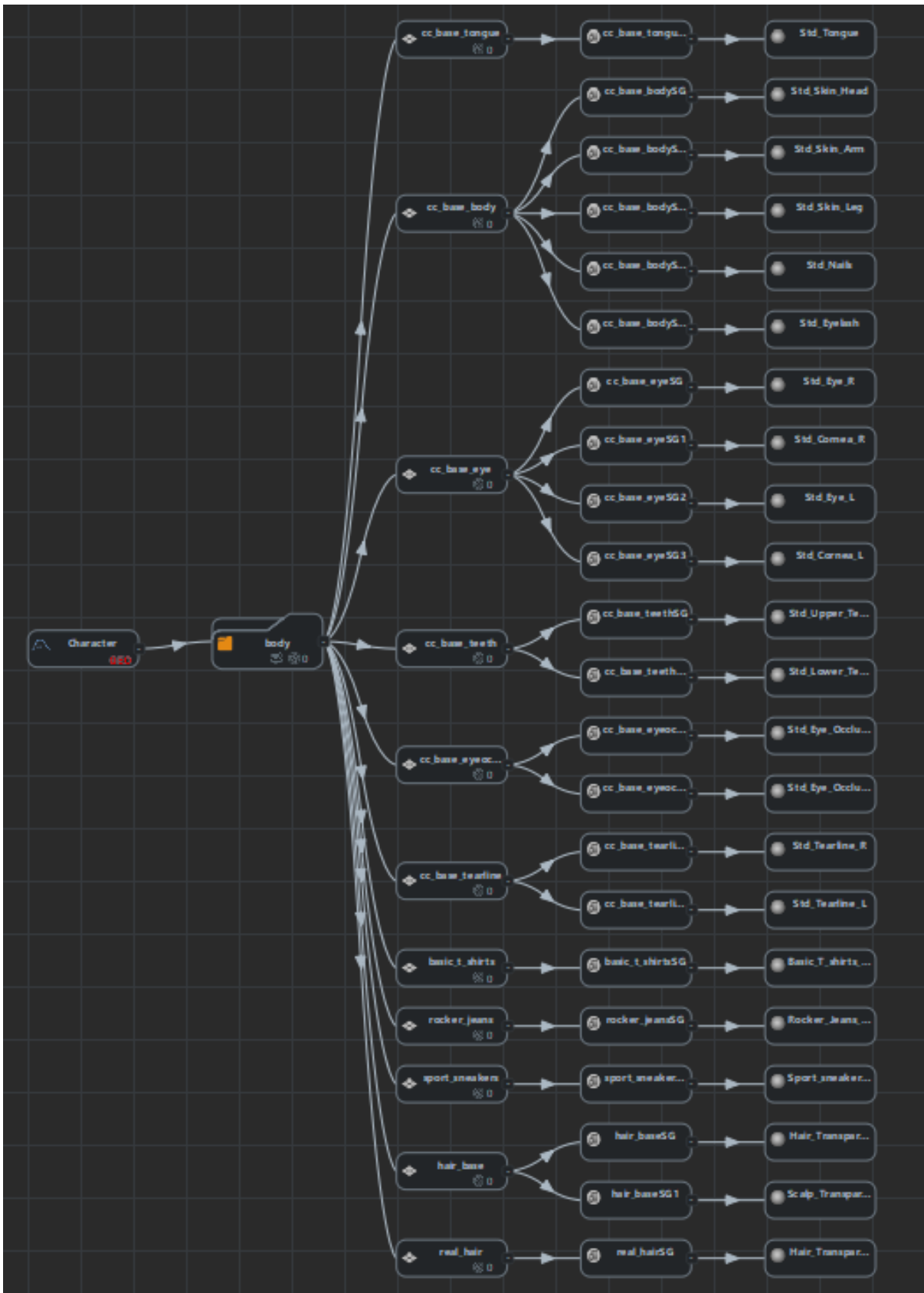
Then select the correct container node in the character maker:

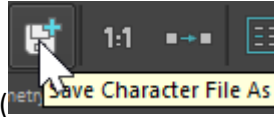


And click the import geometry button:



It should populate the character with its meshes and materials names:



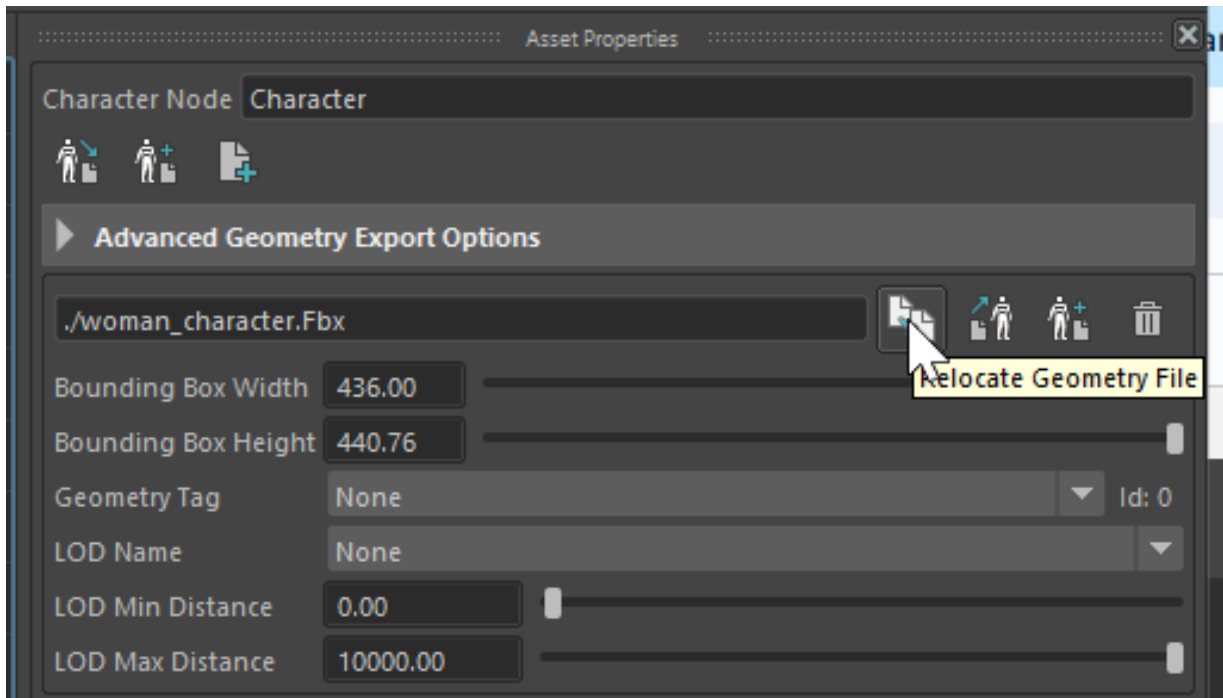


Save the modified character as a GCHA file (



Select the character node:

And relocate the geometry file to the fbx that was used to create this character:



Save the file again, it's now ready to be used.

6.2 ADDING A CHARACTER IN A PROJECT

Adding a character needs to be addressed both in Maya (adding the entities in the scene) and in Unreal (importing the geometry and setting up the blueprints that will allow streaming some bvh).

To add an entity in the Maya scene, follow the golaem Quickstart tutorial here: <https://golaem.com/content/doc/golaem-crowd-documentation/golaem-overview>. The GDA file needs to be re-exported for the simulation to take the new characters into account.

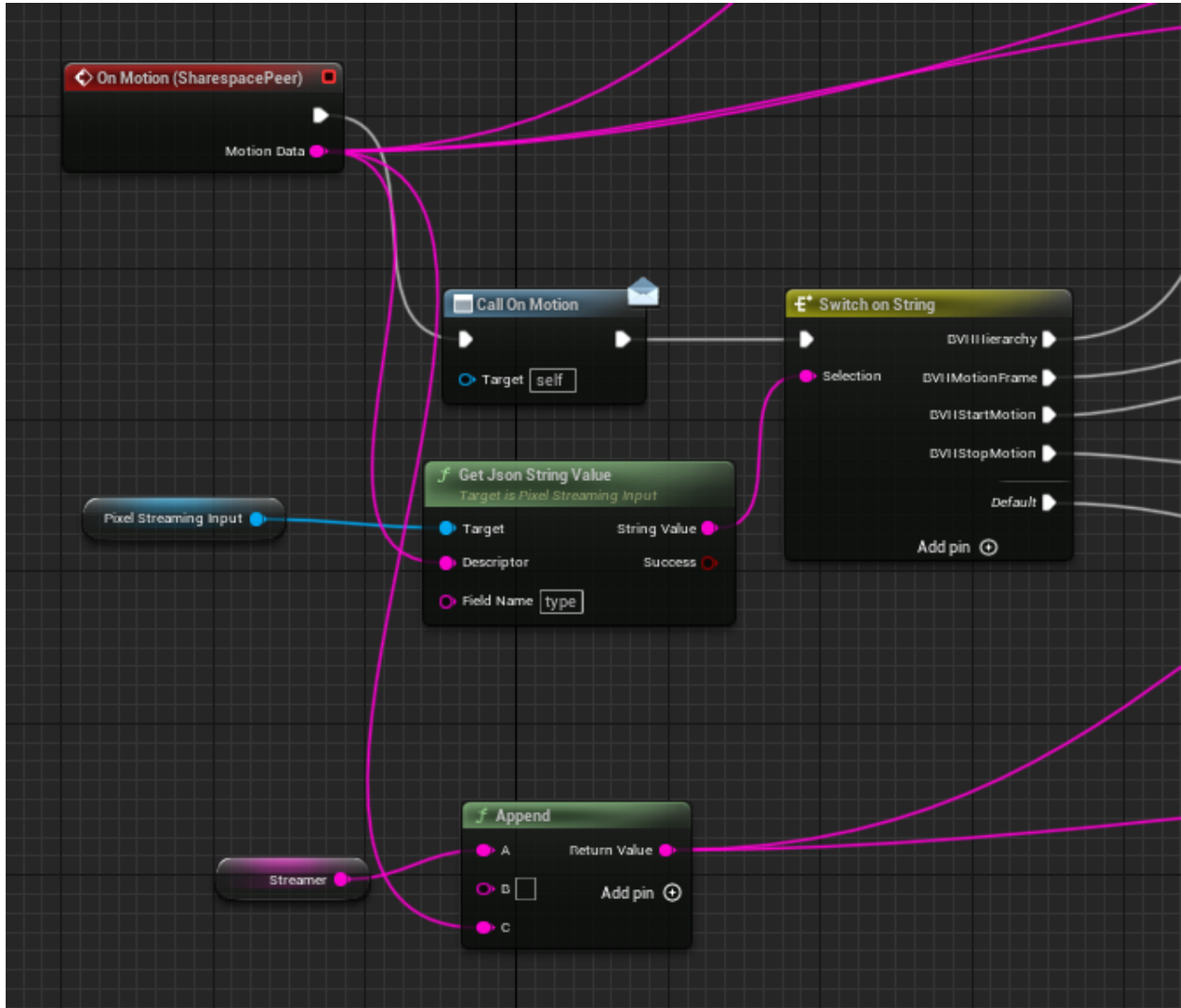
To make sure that an entity added in a GDA file can be correctly replayed in Unreal, it needs to be correctly imported in the Unreal project, making sure that the Unreal Skeletal Mesh node is named as the golaem GCHA file.

To make sure that the bvh streaming will work on the entity, new instances of the *PixelStreamingBlueprint* and the *GolaemBluePrint* needs to be created and configured in the Unreal scene.

6.3 MODIFYING THE STREAMED BVH DATA BEFORE ANIMATION

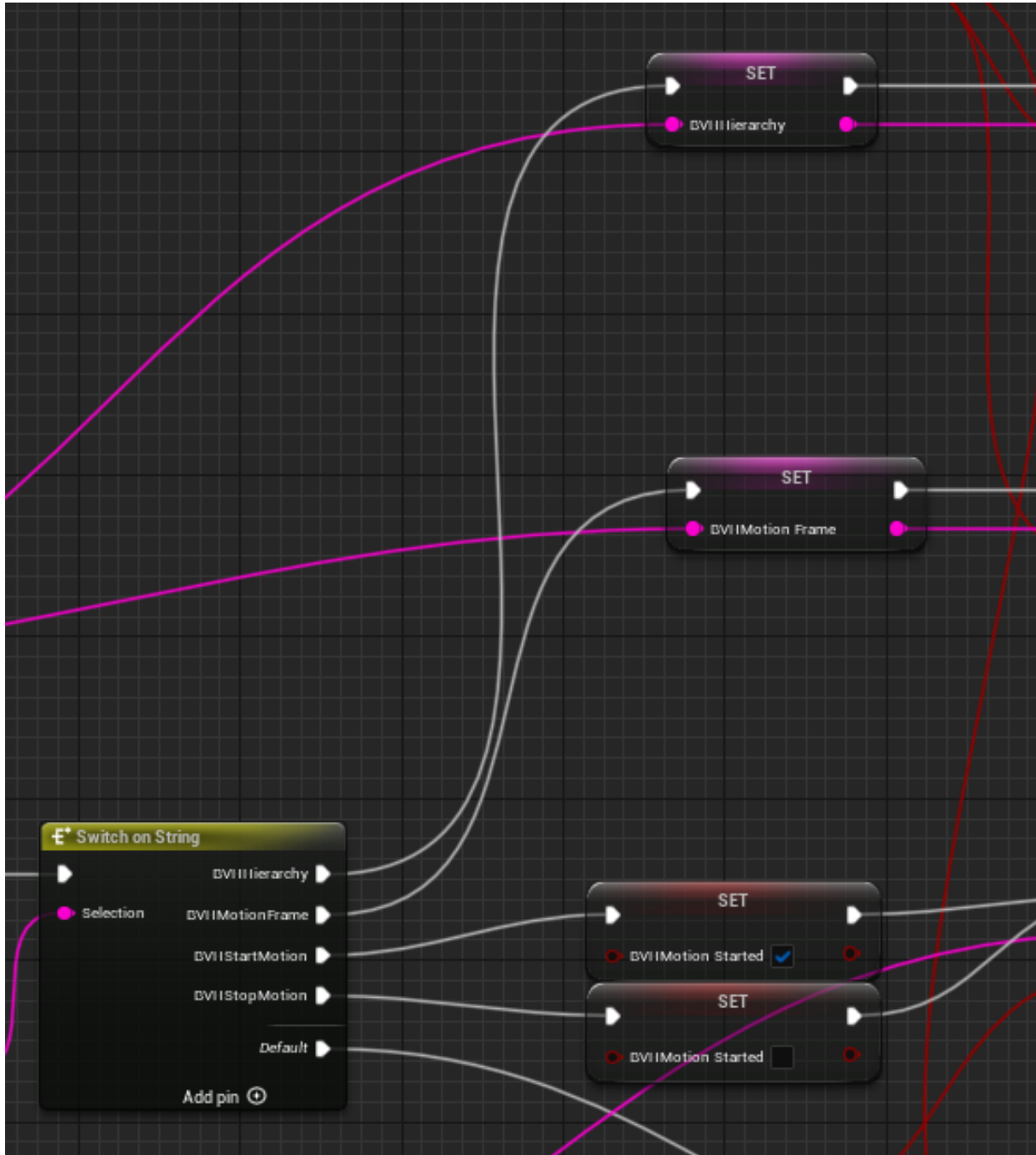
Golaem replays streamed animation data provided by the PixelStreaming plugin. However, it is possible to intercept the streaming data from the PixelStreaming plugin to make some modification before injecting them into the Golaem replay system.

The pixel streaming stream is coming from the Pixel Streaming Input block and the following switch:



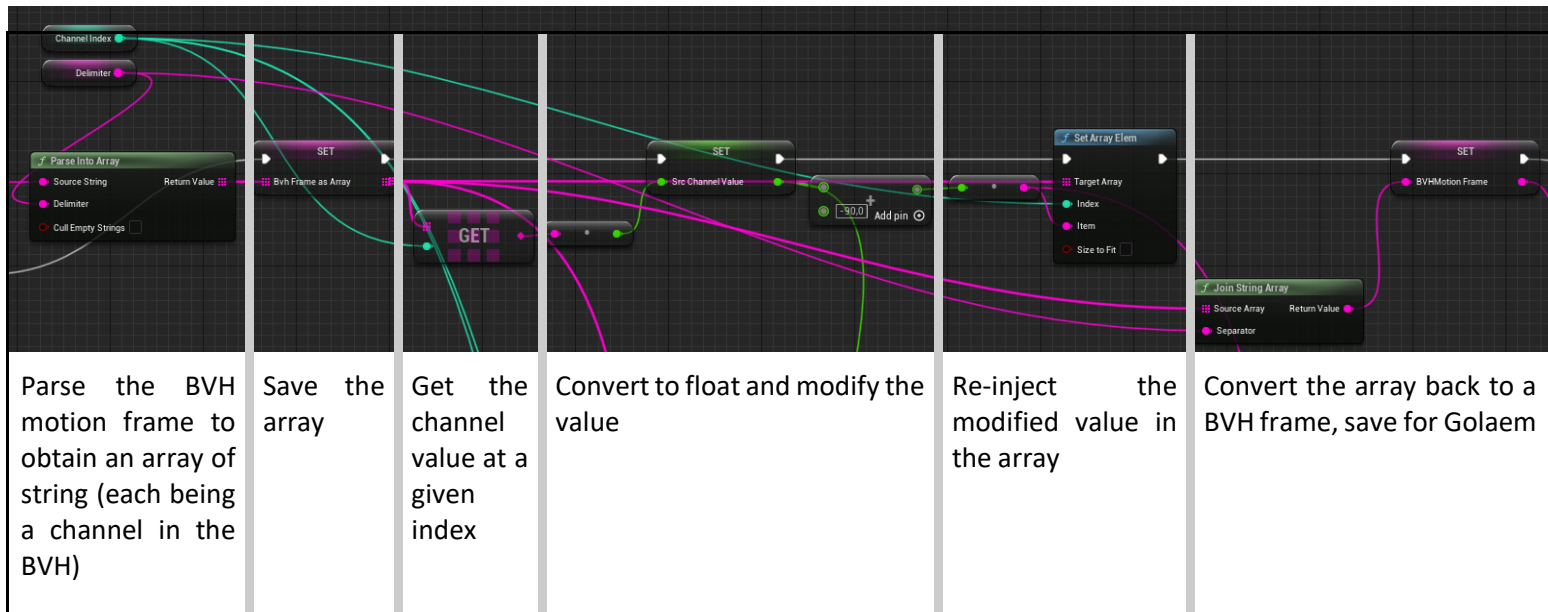
And it is sent to Golaem by setting the corresponding variables:

- BVH Hierarchy
- BVH Motion Frame
- BVH Motion Started



So as long as the string format is kept, it is perfectly possible to modify the BVH motion frame between the pixel streaming stream and the golaem motion replay.

For instance, here is a sample that will get a given channel in the BVH, and add -90 before injecting it back to Golaem:



This approach can be easily used to integrate the Cognitive Architecture developed in WP5.

7 RETARGETING ANIMATIONS FOR THE SPORT SCENARIO

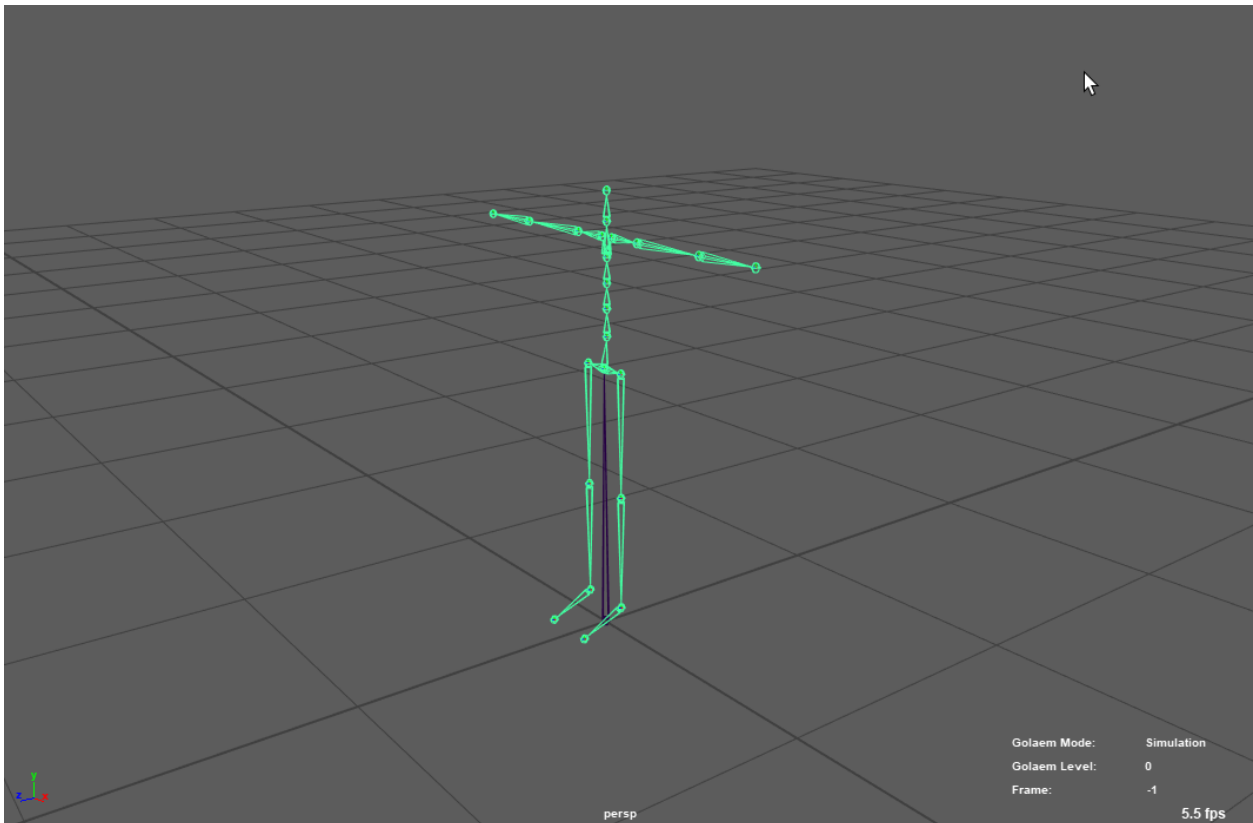
7.1 PREPARING A CHARACTER FILE FOR ANIMATION CONVERSION


Golaem retargeting process works by mapping the skeleton of an animation onto the skeleton of another character on which playing the animation.

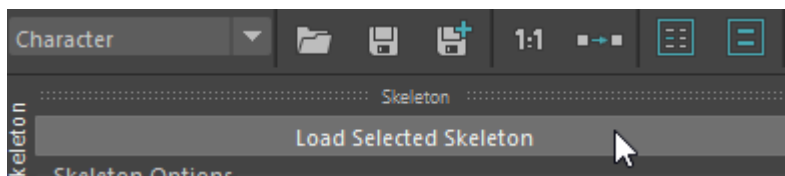
The first step for retargeting is therefore to prepare a Golaem skeleton for the process.

7.1.1 Initial setup

- Load the animation's skeleton in T-pose into Maya



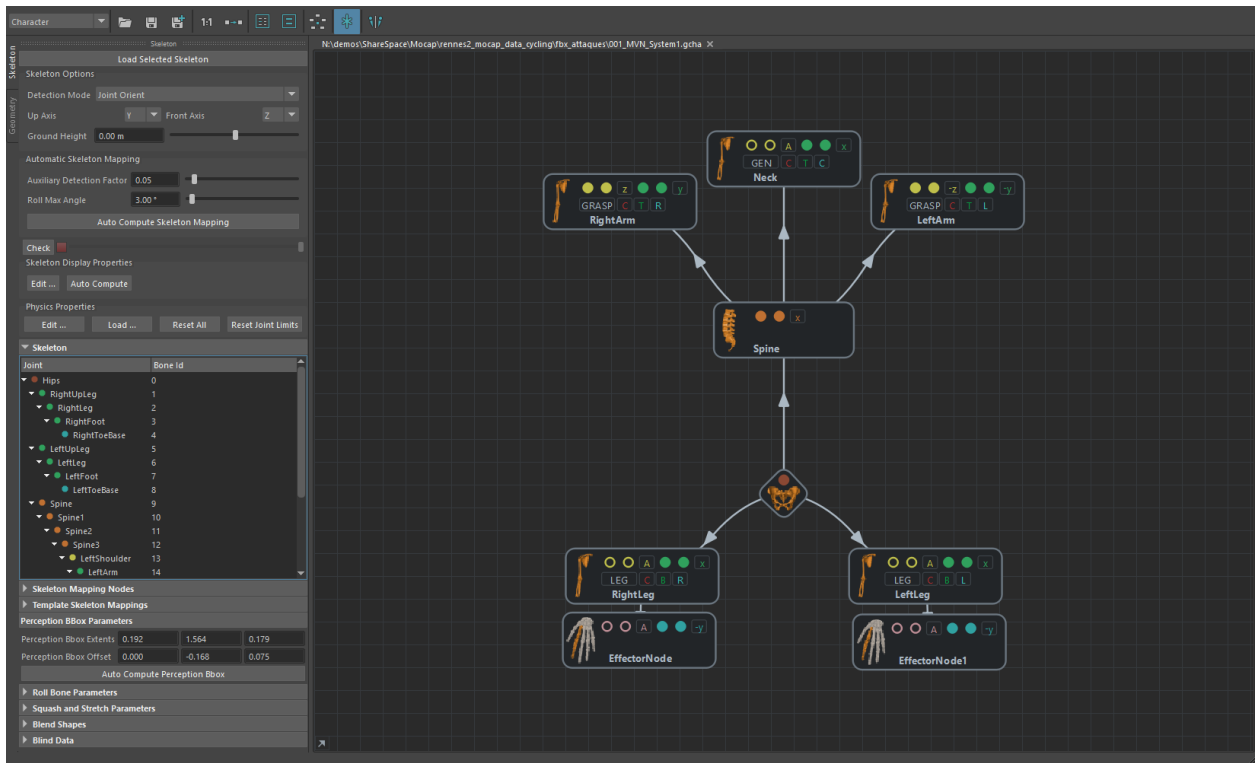
- Open the Golaem character maker , select the root hierarchy for the skeleton and load the skeleton into the character maker



Note that the skeleton T-Pose should have the exact same bone dimensions as in the animation. It's not the case for the 'Reference' bone in the given motion capture, so the 'Hips' bone should be loaded into the character maker rather than the reference bone.

- Use the Golaem Character Maker to map the legs and arms of the skeleton:
<https://golaem.com/content/doc/golaem-crowd-documentation/character-maker-overview>

Here is an example of a correctly mapped skeleton:



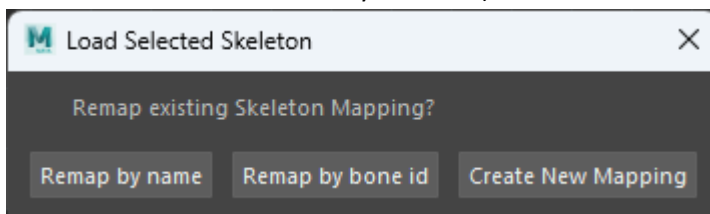
- Save this golaem skeleton mapping as a new GCHA file.

Once done, this file can be re-used for any animation that shares the exact same skeleton (same hierarchy, same dimensions).

7.1.2 New morphology on the same skeleton

If the morphology of an animation’s skeleton changes, but everything else stay the same (same hierarchy, same bone names, ...), then it is possible to use a previously done Golaem skeleton mapping as a base for the new morphology.

- Load the animation’s skeleton in T-pose into Maya
- Open the Golaem character maker, select the root hierarchy for the skeleton and load the skeleton into the character maker
- A dialog box should ask if remapping the skeleton mapping, or create a new one: choose to remap (by bone ID or by name should produce the same results if the skeletons really have the same names and hierarchy of bones):

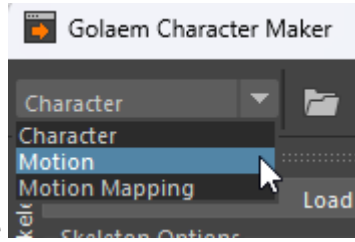


If the ‘Remap by bone id’ option is grayed out, it means that the number of bones in the new skeleton is different than the number of bones in the previous skeleton mapping

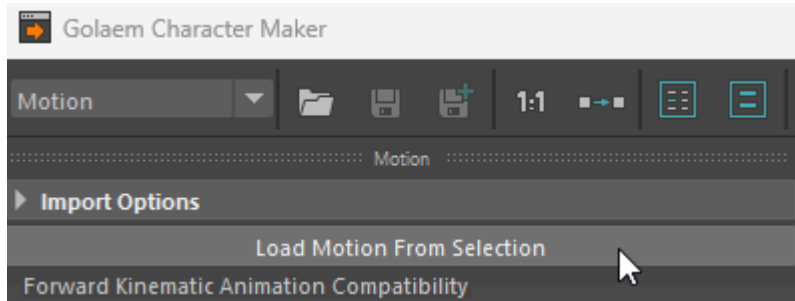
- Save this golaem skeleton mapping as a new GCHA file.

7.2 CONVERTING THE ANIMATION

With the Golaem skeleton mapping for the animation loaded into the Golaem Character Maker, switch

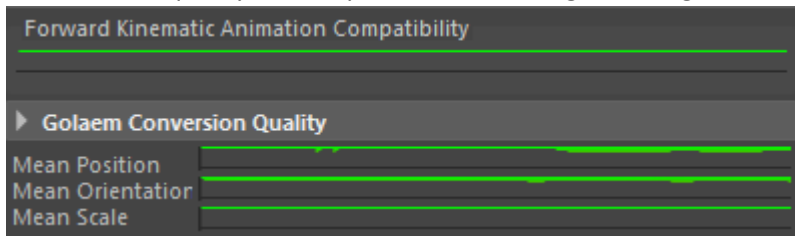


into Motion mode , and once again, select and load the same skeleton



joint :

Conversion quality and potential warning messages should be checked at this step:

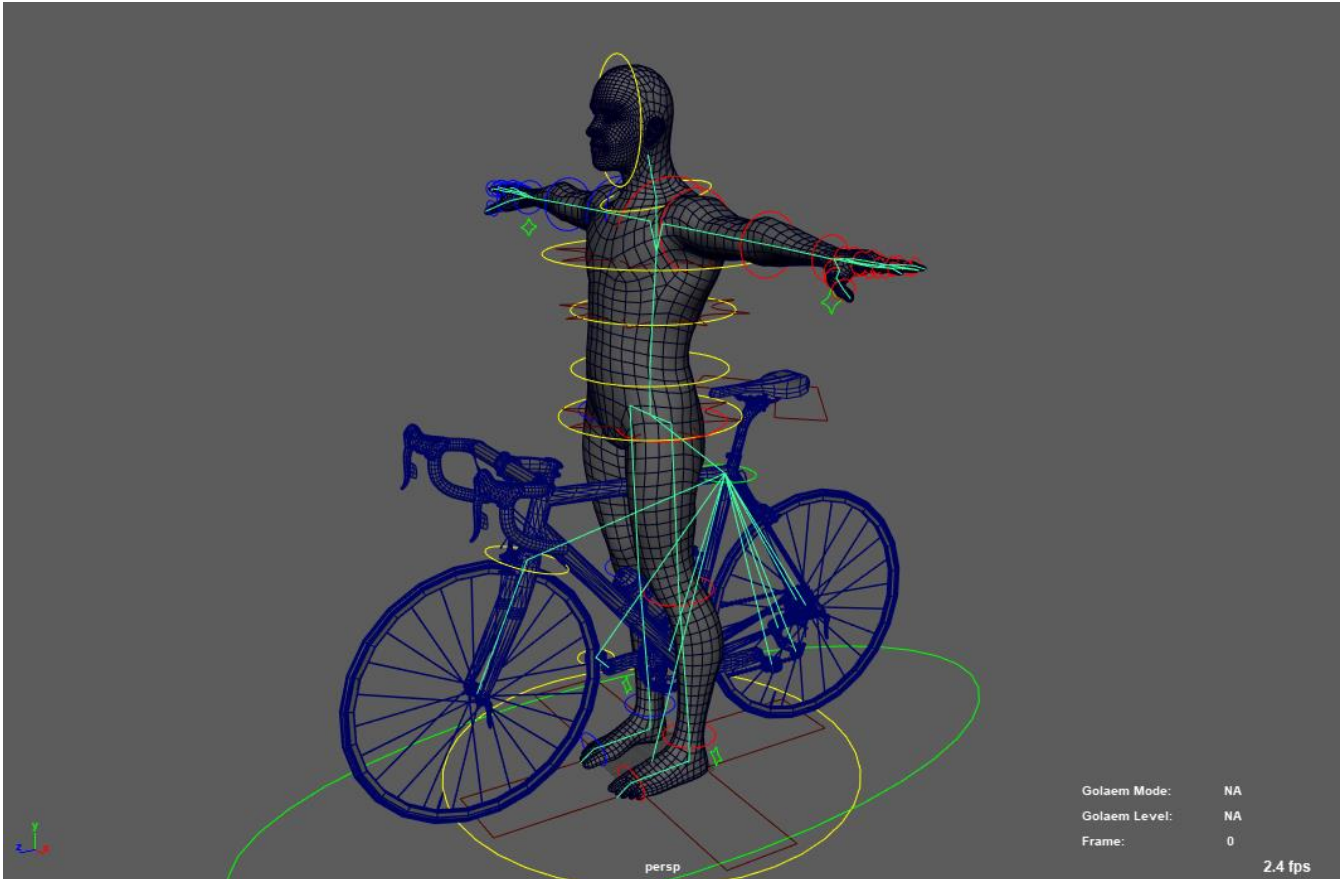



And the animation file can then be saved as a GMO file.

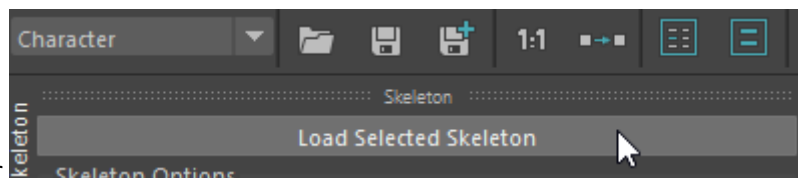
7.3 PREPARING A CHARACTER FILE FOR REPLAY

The replay skeleton should also be converted into a Golaem character file in order to replay the animation.

Load the replay character in T-pose into Maya:



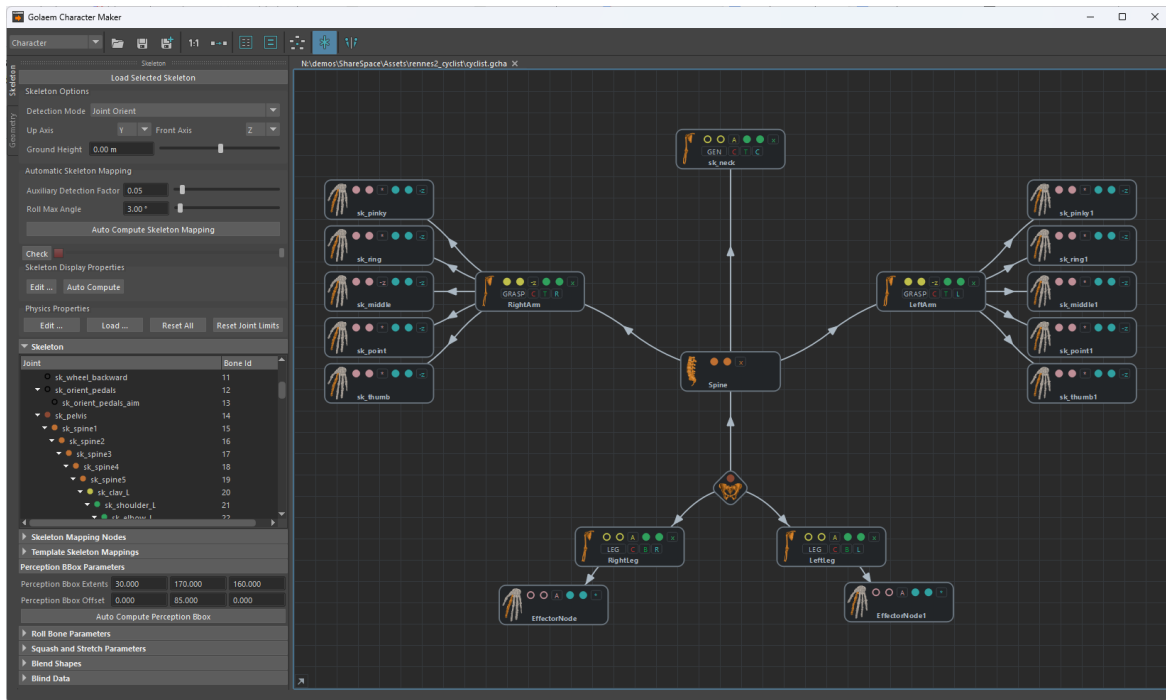
Open the Golaem character maker , select the root hierarchy for the skeleton and load the



skeleton into the character maker

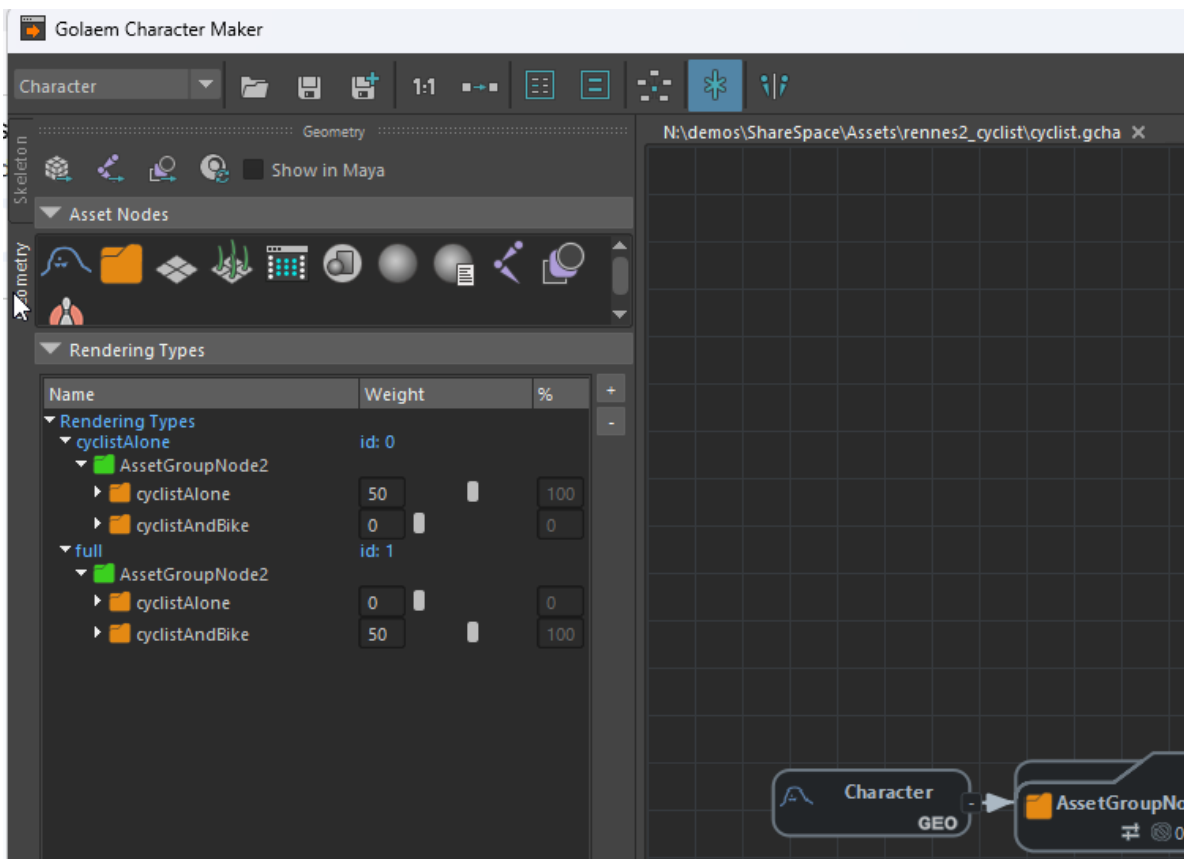
Use the Golaem Character Maker to map the legs and arms of the skeleton: <https://golaem.com/content/doc/golaem-crowd-documentation/character-maker-overview>

Here is an example of a correctly mapped skeleton:



Note that the retargeting process can only work for bones with orientation DOFs, not for joints with translation DOFs (except on the root bone). At the current state, only animations with the cyclist (without the bicycle) were available, so it's recommended to map only the cyclist on this side as well.

As this is the replay character, it's also important to configure its mesh. Follow the Golaem documentation to import the geometry and shader in the geometry part of the character maker.



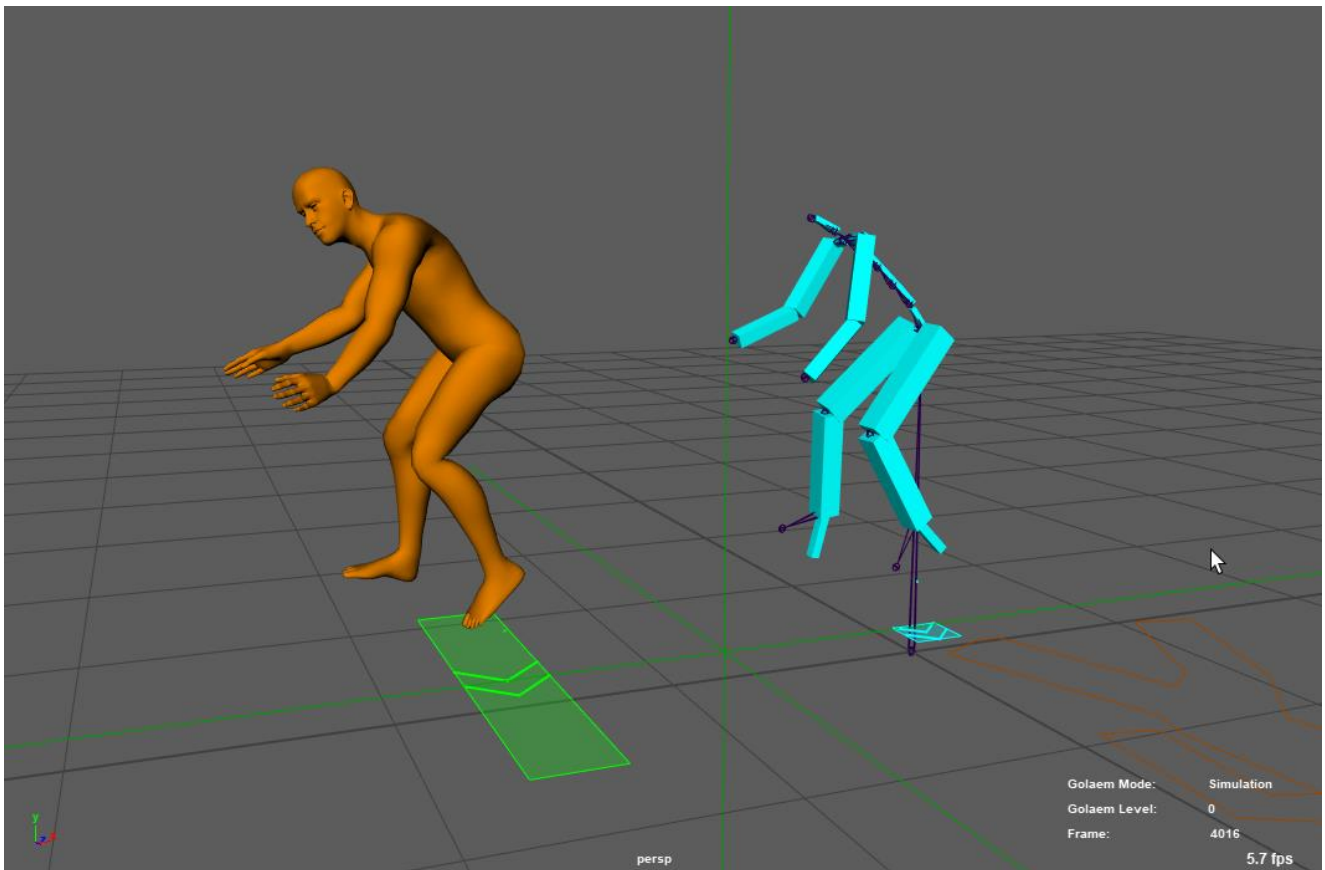
Once done, save this golaem character as a new GCHA file.

7.4 REPLAYING THE ANIMATION ON ANOTHER CHARACTER

Replaying the animation is just a matter of configuring the replay character file (.gcha) on a new golaem entity type, and replay the converted animation (.gmo) onto it.

The Golaem QuickStart should is a good base to configure such a scene: <https://golaem.com/content/doc/golaem-crowd-documentation/quick-start>

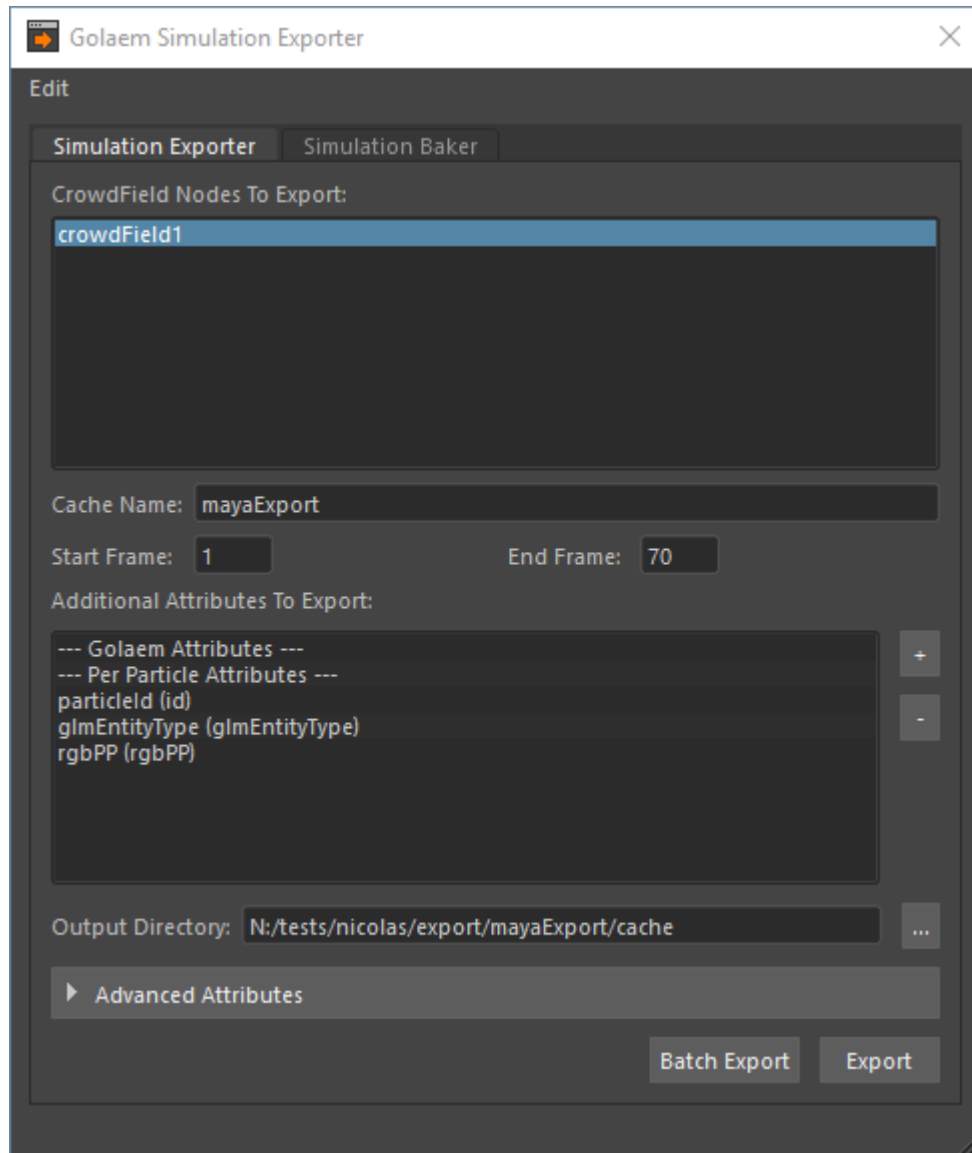
Here is the final scene with the retargeted animation (orange character) and the original animation (cyan skeleton):



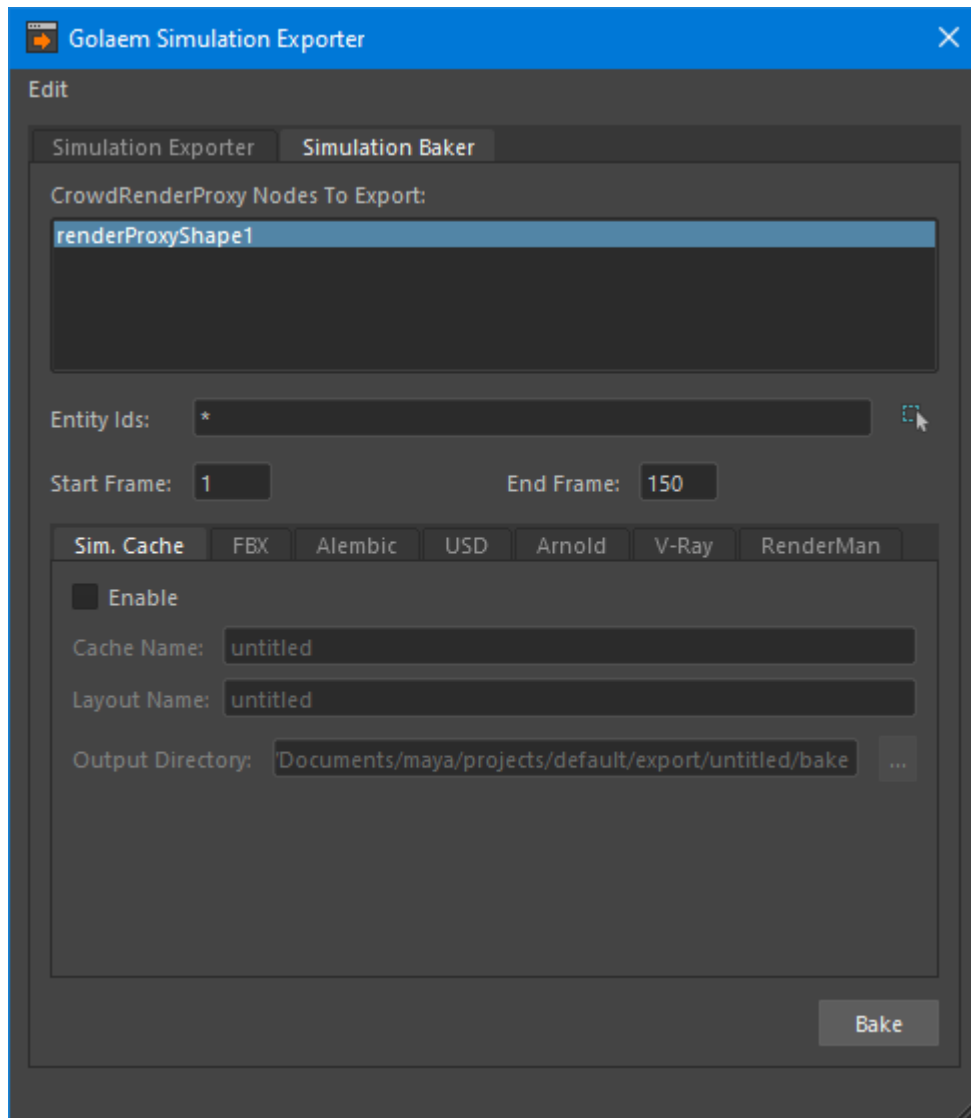
7.5 EXPORT THE RETARGETED ANIMATION

Once an animation is played in simulation, it can be exported as a Golaem cache thanks to the Simulation Exporter:

(<https://golaem.com/content/doc/golaem-crowd-documentation/simulation-export>):



The cache can then be baked into either an FBX / Alembic or USD file using the simulation baker tab on the same tool:



8 CONCLUSIONS

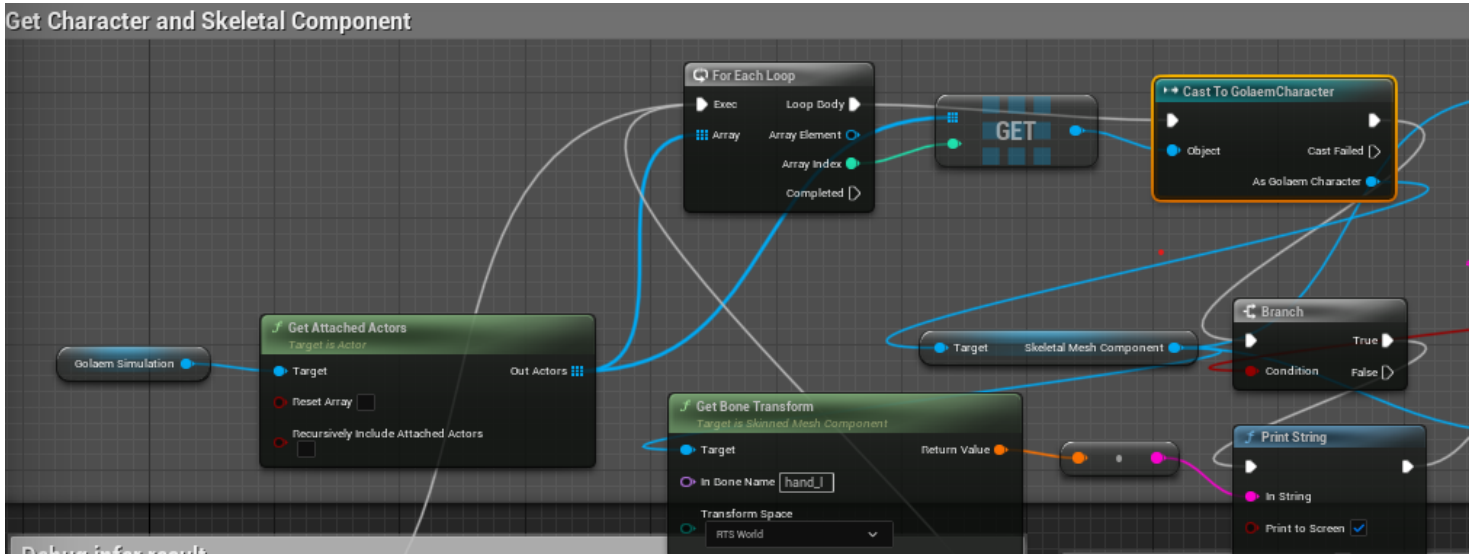
This deliverable has presented the solution developed to animate L1 avatars by using streamed data both for body and face. We also introduced a solution to integrate the Cognitive Architecture to allow the management of L2 and L3 characters, as well as specific needs concerning the Sport Scenario.

A video² is accompanying this deliverable to illustrate body animation from BVH streams on avatars with different morphologies, and facial animation driven by the audio stream. Note that gaze tracking is not yet available, so the eyes of the avatar are currently static.

² <https://www.youtube.com/watch?v=46LtEDNo6xA&t=7s>

APPENDIX 1: UNREAL TIPS AND TRICKS

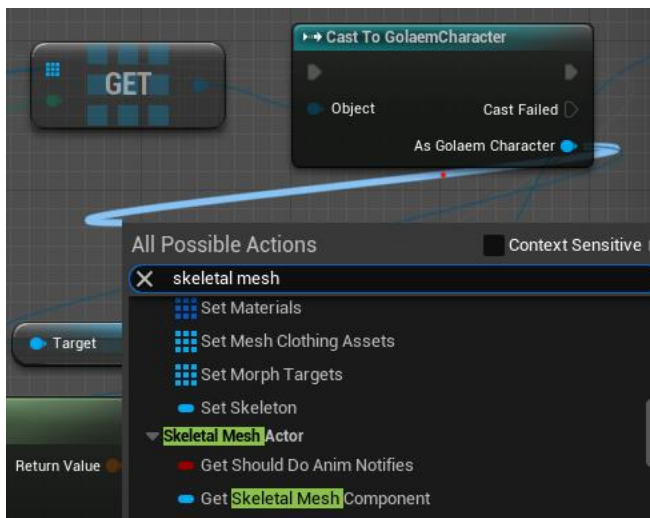
Getting the bone transforms of an actor:



You can get a bone transform from a Skeletal Mesh Actor.

Total path of this script is:

- GetAttachedActors, which takes GolaemSimulation as input;
- For each Actor / index : Cast to Golaem Character (or any Actor having a SkeletalMeshComponent);
- If cast is successful, get the skeletal mesh from the GolaemCharacter



- Call GetBoneTransform on the skeletalMeshComponent.