Marika Salli

3D Game Character Animations in Unreal Engine 5: Creation and Implementation

Bachelor's thesis

Bachelor's Degree in Culture and Arts

Game Design

2023

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Culture and Arts
Author(s)	Marika Salli
Thesis title	3D Game Character Animations in Unreal Engine 5: Creation and Implementation
Commissioned by	Self-commissioned
Year	2023
Pages	55 pages
Supervisor(s)	Marko Siitonen

ABSTRACT

Unreal Engine 5 was released in 2022, and with its new features Lumen and Nanite making high rendering quality more obtainable with lower hardware requirements, it will likely become a very popular game engine in the near future.

Animations are a crucial part of most games, and the new engine version introduced many new animation features with a focus on real-time animation. So far there is little literature sources of the subject, and while Unreal Engine's documentation offers a good manual of how to use different features, it can be challenging for a new user to comprehend when and why to use them, and how they differ. The objective of this thesis was to answer these questions and define a workflow for animating a 3D game character.

The different animation features and their usage were first explored through Unreal Engine's documentation and community tutorials. Next, the features were tested by animating a game character for a game project and different methods were compared.

The study managed to answer the questions of when and why to use each feature, and a suggested workflow was created in a form of a chart. Functional game character animations were created and implemented to a game as the productive result of the thesis.

Keywords: Unreal Engine 5, animation, 3D, character, game development

CONTENTS

1 INTRODUCTION

Producing animations is a crucial part of almost any 3D game, and Unreal, one of the most used game engines, has many tools to implement animations both internally and externally.

Unreal Engine 5 is the newest version of the engine released in 2022 (Unreal Engine 2022). It introduced ground-breaking new technologies called Lumen and Nanite, which make it possible to render higher quality graphics than earlier with lower system requirements, most likely making Unreal even more popular engine than before. Unreal Engine also introduced Meta Humans, bringing realistic human characters to game developers with little effort required. Other notable new features are World Partition, Meta Sounds, Virtual Shadow Maps, Temporal Super Resolution and Chaos Physics, which all contribute to a better quality in game production.

Unreal Engine 5 also brought new features to animation production: Full Body IK, Machine Learning Deformer and Motion Warping. Real-time animation has been given a lot of emphasis. With Unreal's numerous animation features, it can be challenging for a new user to comprehend the usage of them. While Unreal's own documentation acts as a manual of *how* the features are used, it offers little to no knowledge of *when* and *why* the user would want to use each feature in game development.

The purpose of this production-based thesis is to explore the different 3D animation features in Unreal Engine 5, experiment how, when, and why to use them and find out what are the best practices and techniques to use in different situations. To achieve these goals, animation features are first researched and presented, then tested and compared, and finally used to produce character animations for a game project the author is currently working on.

Given the topical nature of the subject, little literature exists of it so far. Main information sources used in this thesis are therefore Unreal's own documentation together with community tutorials. The thesis assumes the reader has basic

knowledge of game development, animation, and rigging. Therefore, common terms and practices related to these are only discussed briefly.

2 3D GAME ANIMATIONS IN GENERAL

Animation is a vital part of almost any game. Characters and objects need dynamic movement to look convincing and lively. In video games, there are actually two sections of animation: game animations and cinematics (Beane 2012). While cinematics are animated with the same fundamentals as animations films, game animations require a slightly different approach, as they need to function within the gameplay and respond to changing situations like different environments or combination of several actions. Players typically like their characters to respond quickly, which means the movement has to begin without delay or anticipation when a key is pressed or a mouse button clicked. Game animations are also limited by hardware capability, although less and less significantly with modern engines.

A 3D game object, also called mesh, is animated by first creating drivers, often referred to as bones, skeleton or joint chains, inside the mesh. Each vertex of the mesh is then given a value of how much its position should be affected by each bone. This process is called skinning. Bones are assigned controllers, which the animator can use to move each bone. This combination of skeleton and controllers tied to a mesh is called a rig.

There are three different ways to animate the mesh (Beane 2012). The first one is hand-keyframed animation, where the animator creates a different pose at each keyframe by moving the controllers. The second method is motion capture, where a real-life actor is filmed, and the motion is transferred to the mesh. The third method is procedural animation, where the mesh is programmed to move according to a set of rules.

Many of the 12 principles found in 2D animation also function in 3D game animation. In game animation, a sense of weight and balance are especially

important (Beane 2012). As mentioned before, game animations have to be responsive to player actions and collisions with objects and environment (Totten 2021). Characters' shapes and sizes should not change too much, as this can create problems with collisions, and anticipation and extra movement before the actual action are limited as the character has to feel responsive. The staging principle applies poorly in 3D game animating, and as 3D games often allow the player to rotate the camera at will, animations have to look good and readable from several angles. Follow-through actions are important, but they have to be quick in order to not slow the game play down. The slow-in/slow-out principle is situational; it works but requires editing the animation graph curves in the software, and in some cases, there are not enough frames to use it as more frames create a bigger file size. Arcs, secondary action, timing, exaggeration, solid drawing and appeal work equally well in games as it does in films, both 3D and 2D.

3D game animations have traditionally been made in 3D software such as Maya or Motion Builder, but Unreal Engine 5 lets the user animate in real-time within the engine, which can make the workflow faster and more efficient as the animator can see how the animation looks immediately and make the needed corrections without having to go back and forth between different programs. Real-time animation technology is advancing quickly at the moment and Unreal Engine is bringing the new techniques accessible even to indie developers (Business of Animation 2023). Apart from in-game animations, real-time animation also makes it possible to have the game character communicate with audience in marketing events, and cinematics creation faster by cutting the rendering time.

3 ANIMATION FEATURES IN UNREAL ENGINE 5

Unreal Engine 5 is a software that can be used for animation in games but also for cinematics and other visual works. User can animate characters, objects, effects, cameras, materials, crowds, and flocks. This chapter will go over the main animation features the engine has and briefly explain how they are used.

3.1 Control Rig

Control Rig is a feature within Unreal Engine 5 which allows the user to create a rig for a mesh fast and easy inside the engine instead of an external 3D software (Unreal Engine 2022). The rig can be further adjusted in Control Rig Editor, where the user can add custom controls, channels, and other manipulators. Once the rig is finished, it can be used to animate the mesh in the engine using other animation features such as Sequencer.

The advantage of using Control Rig is that it allows the user to animate characters directly inside the engine, which makes the workflow faster compared to having to go back and forth between game engine and an external animation software every time an animation needs to be tweaked (Smart Poly 2022). It is important to note that the mesh needs to already have a skeleton and be skinned before being imported to Unreal Engine in order to animate it. Controls or other rig functions on the other hand are not needed if the plan is to use Control Rig.

Importing a mesh into Unreal Engine creates 3 assets in the content browser: a Skeletal Mesh, Physics Asset and Skeleton. A Control Rig can be created by right-clicking the Skeletal Mesh in the Content Browser (found at the bottom of the screen by pressing *Ctrl* + *Space*), and then selecting *Create* – *Control Rig* (Unreal Engine 2022). This creates a Control Rig Asset in the same folder. Double-clicking this new asset will open Control Rig Editor.

In the Control Rig Editor user can create controls for each bone by going to the Hierarchy tab and right-clicking a bone, then selecting *New* – *New Control* (Figure 1) (Unreal Engine 2022). New bones and nulls (Unreal's name for control spaces) can also be created from this same menu. The new control is by default an FK (forward-kinematics) control, and it will have the same placement and orientation as the bone it controls. This new control can then be renamed and moved out of the hierarchy to keep things better organized and to ensure correct functionality. Control shape, colour and transforms can be changed in the Details panel on the right-hand side of the screen.

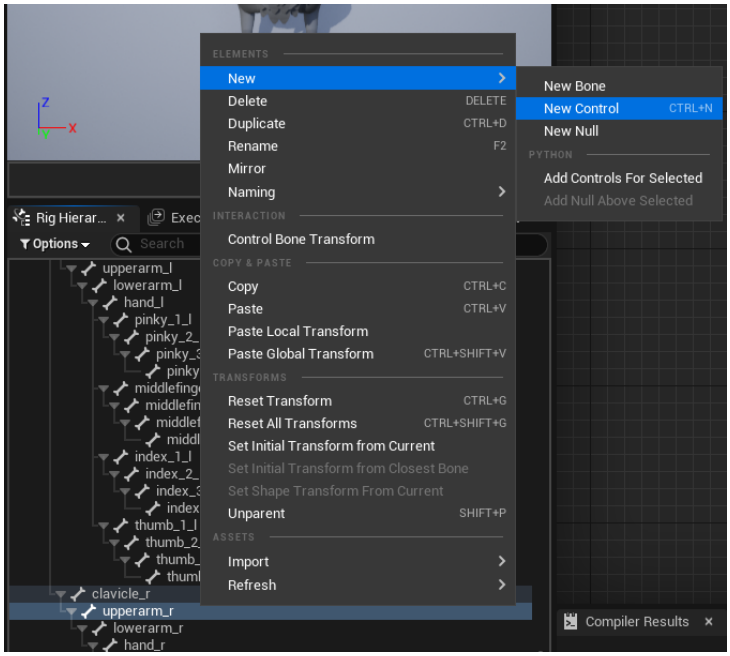

Figure 1. Creation of a new control.

In order to make the control actually drive the bone, it needs to be referenced in the Rig Graph, located in the middle of the Control Rig Editor, by clicking and dragging it to the graph and then selecting *Get Control* (Unreal Engine 2022). The same must be done to the bone, but instead select *Set Bone*. Next, the nodes must be connected: *Transform* pin from the control to the *Value* pin of the bone, and also *Execute* pin of the *Forwards Solve* node to the *Execute* pin of the bone node (Figure 2).

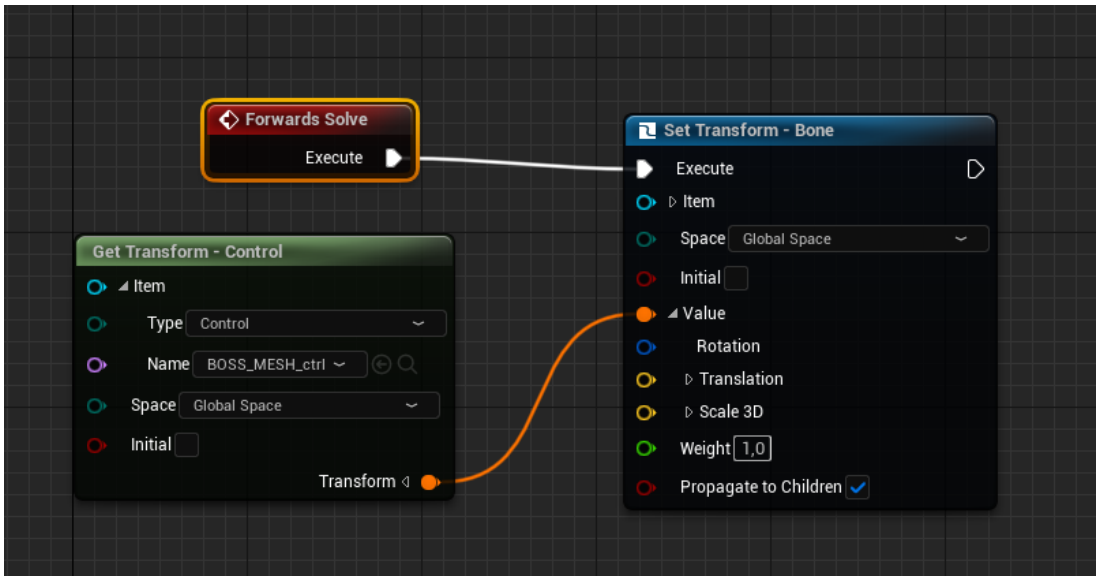

Figure 2. Making a control to drive a bone.

Creating IK controls can be done by right-clicking the Rig Graph and then searching and choosing *Basic IK* (Smart Poly 2022). In this new node (Figure 3) there are *Item A, Item B* and *Effector Item*, which represent the bones of the IK chain with Item A being the first bone and Effector the last. Desired bones are chosen from the dropdown menus on each slot. A control for this new IK chain can be created the same way as for the FK controls, but in the Rig Graph its *Transform* pin will be connected to the *Effector* pin of the IK node instead of a bone node. To control which way the middle joint (for example elbow or knee) of the IK chain points, another control needs to be created and then connected to the *Pole Vector* pin of the IK node. Finally, the *Execute* pin must be connected to this new node similar to the FK controls.

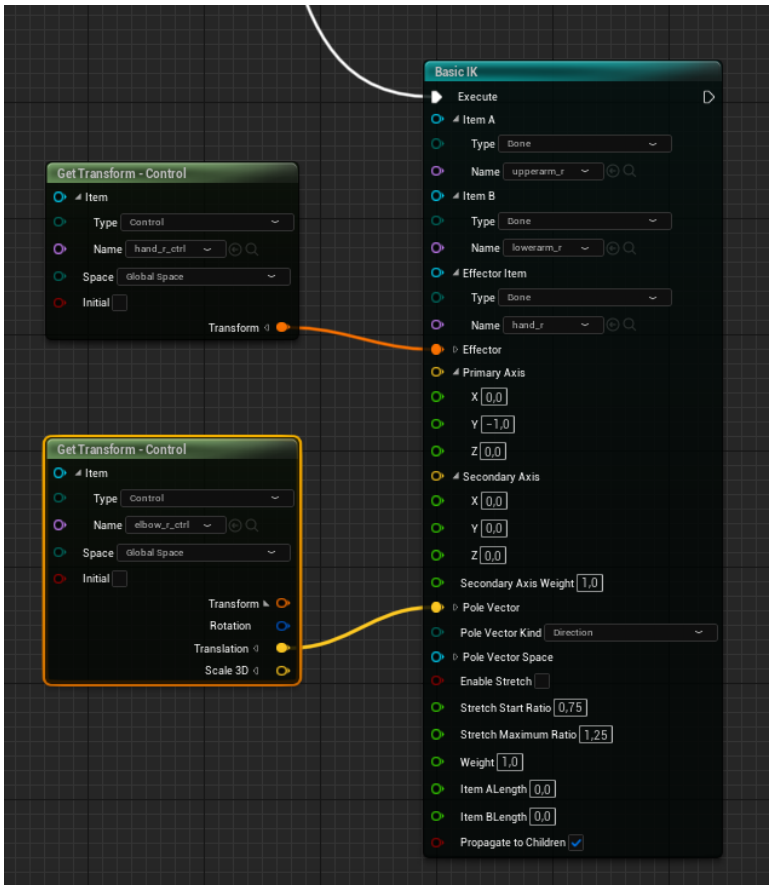

Figure 3. Creation of IK controls for a Control Rig.

The Basic IK node has a section for primary and secondary axes. These tell the software which way the joints are supposed to bend, and it's important that they are set the same way as the joint orientations, as otherwise the model will twist

and break uncontrollably. Often it seems to work best to only set the primary axis to correspond the joint orientation and simply leave secondary axis to 0.

Another useful node to set up is Distribute Rotation (Figure 4). It can be used for long joint chains to give each joint an equal rotation as the first joint of the chain, thus making animation a lot quicker for long flexible structures such as spines and tails (Smart Poly 2022). The node is created by right-clicking the Rig Graph and searching and selecting it. Then each bone in the chain must be chosen to *Items*, similar to IK node, then the controls *Rotation* pin is connected to the *Rotation* pin of the new node, and finally the *Execute* pin is connected.

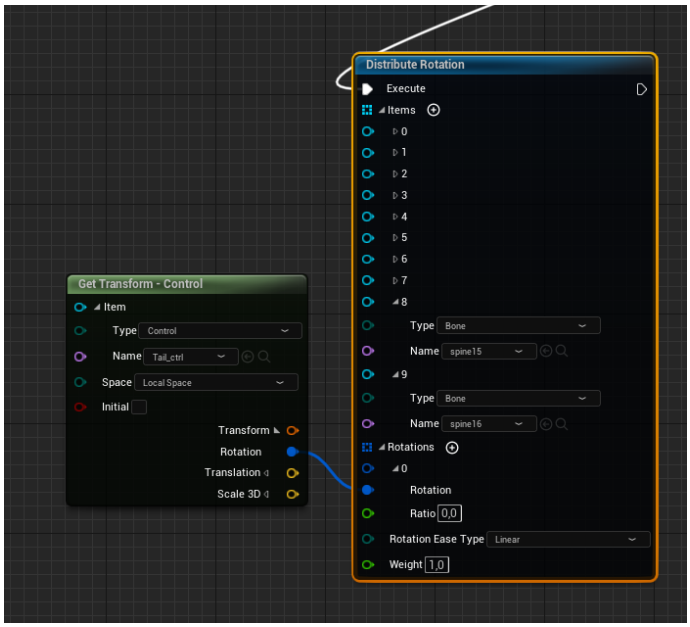

Figure 4. Distribute Rotation node set up.

Full Body IK is a new feature inside Control Rig. It is a procedural adjustment tool which can be used to help position character's hands and feet to be in a certain place, for example touch the ground (Unreal Engine 2022). Full Body IK is a node and is created the same way as other nodes. It has a *Root* pin that requires a bone, usually pelvis or hips, to be selected. Next, the end bone controls of each limp must be brought to the graph, and their *Transform* pins must get connected to the *Effector* pins as shown in Figure 5.

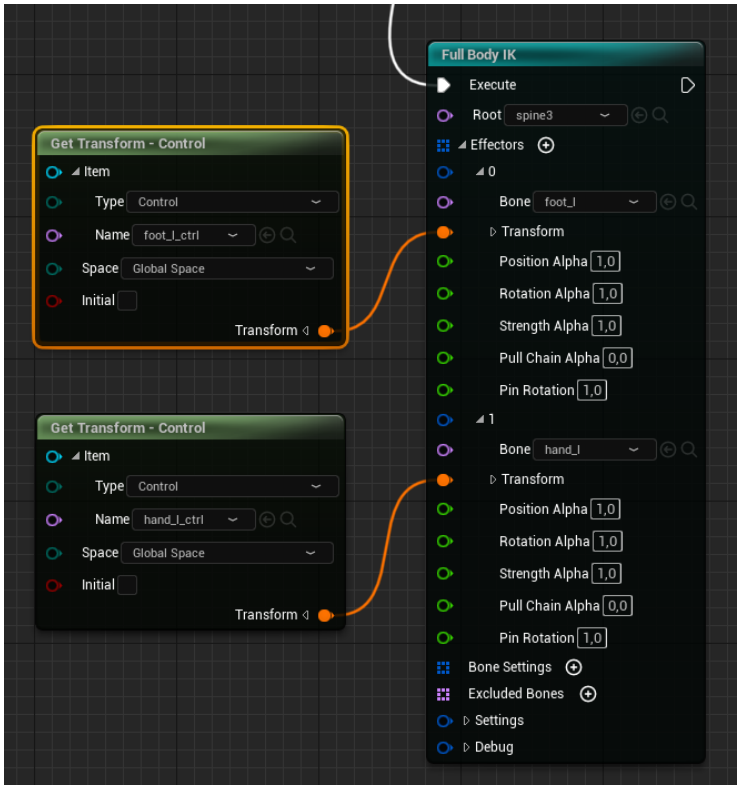

Figure 5. Full Body IK node setup.

Overall Control Rig seems to be a rather quick way to create a rig and is definitely more than enough for at least side characters. I found especially control creation, orientation and positioning faster than in Maya. Control Rig also allows the rig to be animated directly in Unreal Engine's Sequencer which can make the workflow smoother.

3.2 Sequencer

Sequencer is a non-linear editing tool and the primary way of creating cinematics in Unreal Engine 5 (Unreal Engine 2022). It is also used to create Animation Sequences which can be used with Blueprints to implement animations to the gameplay. It utilises timeline, keyframes, layers and curves. Sequencer consists of 2 main parts: Level Sequence Asset and Level Sequence Actor. The former contains tracks, cameras, keyframes and animations while the latter binds its data to a level. Asset is located in the Content Browser while Actor is located in the level.

Level Sequence Asset can be created by clicking Cinematics menu and selecting *Add Level Sequence* (Unreal Engine 2022). This can then be given a name and saved. Level Sequence Actor will be created, and Sequencer Editor (Figure 6) opened automatically after this step has been completed. Dragging a Control Rig asset into the scene will accomplish this same procedure (Bohl 2022).

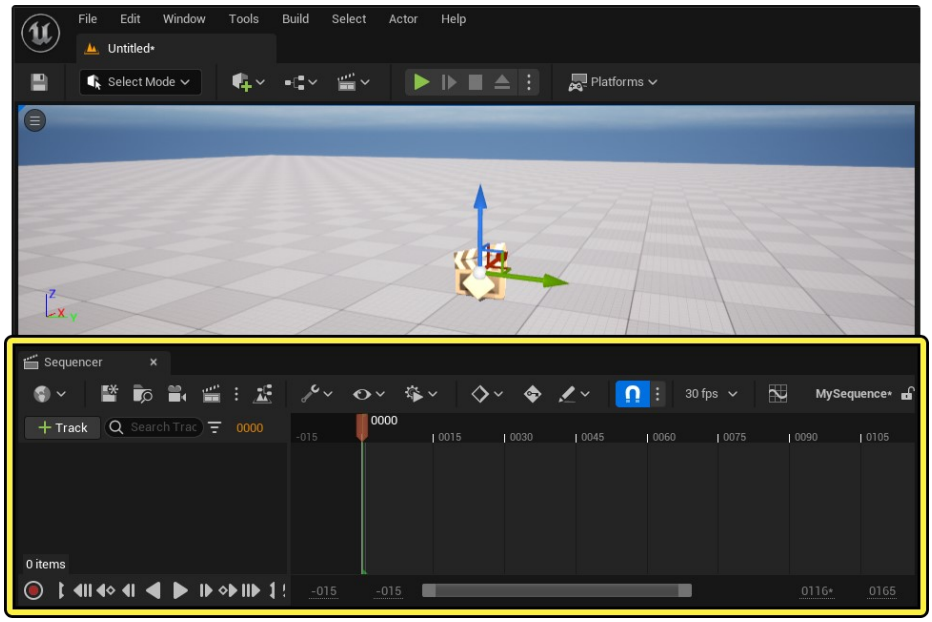

Figure 6. Sequencer editor (Unreal Engine 2022)

Another way to create Level Sequence Asset is to go to the Content Browser and click *Add/Import – Animation – Level Sequence* (Unreal Engine 2022). Level Sequence Actor must be added manually by clicking *Place Actors – Cinematic – Level Sequence Actor* and then dragging it to the scene. Next, the Asset must be bound to the Actor by dragging the Asset onto the *Level Sequence* property within the Actor.

To animate a character in Sequencer, user must first add the character into Sequencer by selecting the character and clicking *Add Track – Actor to Sequencer – Add <character name>* within the Sequencer editor (Unreal Engine 2022). After this, animations are added to the character by clicking *Add Animation* on the *Animation* track (Figure 7). The menu will show all animations on the project which are compatible with the character's skeleton.

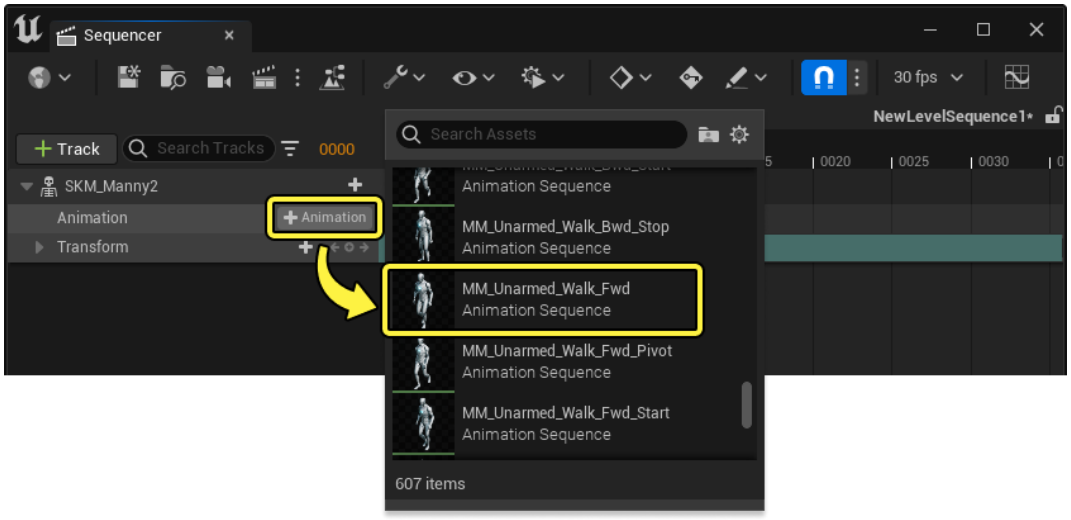

Figure 7. Adding an animation sequence to the Sequencer for editing (Unreal Engine 2022)

A character can also be animated in Sequencer simply by keyframing its transforms, and if it has a Control Rig, then each part of it can be keyframed thus allowing the creation of entirely new animations (Moore 2021). A keyframe is created similarly to other software: first moving to the desired moment on the timeline, then altering the transforms and finally pressing enter. Sequencer also has a traditional curve editor which can be accessed from its top panel's curve icon. There the animation's speed and acceleration can be fine-tuned (Figure 8).

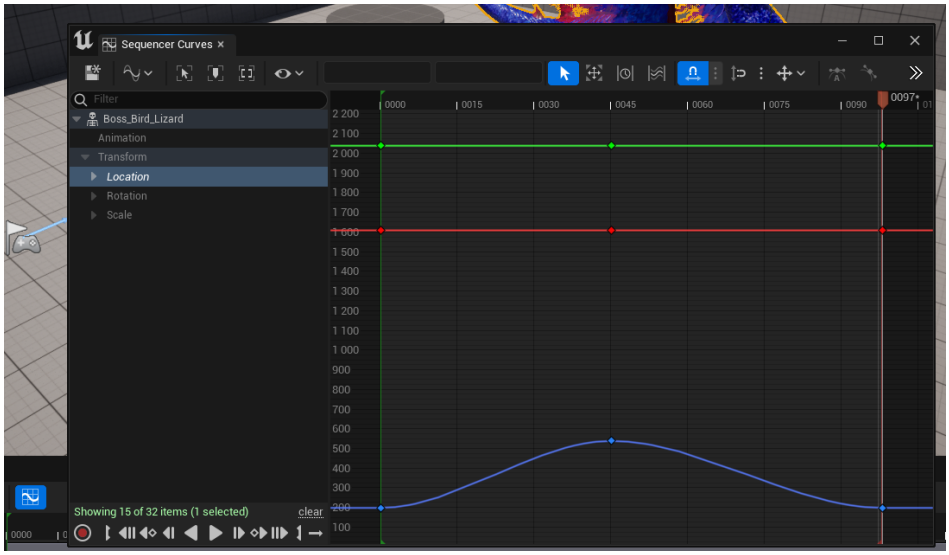

Figure 8. Curve editor inside Sequencer.

Once the animation is finished, it can be exported into Animation Sequence asset to be used with Blueprints (Bohl 2022) or other animation tools. This is done by

right-clicking the character's track in the Sequencer and selecting either *Bake Animation Sequence or Create Linked Animation Sequence.*

3.3 Animation Sequences and Animation Curves

Animation Sequences are animation assets made with Sequencer or external software. Animation Sequence Editor is a tool for editing existing Animation Sequences, Pose assets, Montages or Curves (Unreal Engine 2022). Using Animation Sequence Editor is faster than going to Sequencer if the animation only needs slight adjustments, like cutting a part of it off. It can also be used to alter bone transitions, for example to make a character lean one way for the whole animation duration.

Animation Curve Editor can be used to animate custom float attributes created in Maya (Unreal Engine 2022). Animation curves are typically used to affect scalar material parameters or morph targets.

3.4 Blend Spaces

Blend Spaces are assets that allow the user to blend smoothly between several animations or poses using either one or two-dimensional graph (Unreal Engine 2022). These differ by how they can be used in the editor and Blueprints. The one-dimensional graph is the more commonly used one, as it's simpler to use and usually serves the purpose well enough (Gorka Champion 2022). In addition to regular Blend Spaces, there are special Blend Spaces called Aim Offsets. These can be used to create mesh-space additive poses in order make aiming or look at -animations. The graphs can be referenced with Animation Blueprints and the blending can be set to be controlled by gameplay or other events.

A Blend Space can be created by clicking *Add* button in the Content Browser and selecting *Animation – Blend Space* (Epic Games 2022). After that the desired skeleton is picked from a list that pops up. This creates a Blend Space asset which can edited in a separate window by double clicking it (Figure 9).

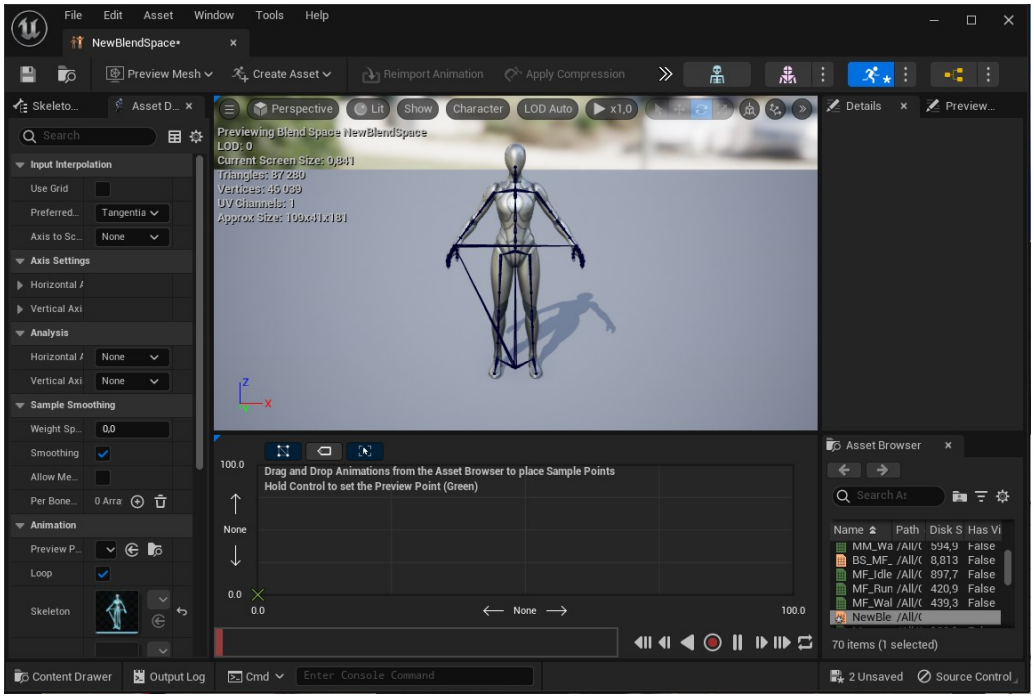

Figure 9. Blendspace editing window (Epic Games 2022)

Animation Sequences can be dragged and dropped to the graph from the Asset Browser on the right (Gorka Champion 2022). The Sequences can be either static poses or loop animations. Once the graph is finished, the Blend Space is put to use on the Animation Graph part of the Animation Blueprint. The correct way to connect the nodes is presented in Figure 10.

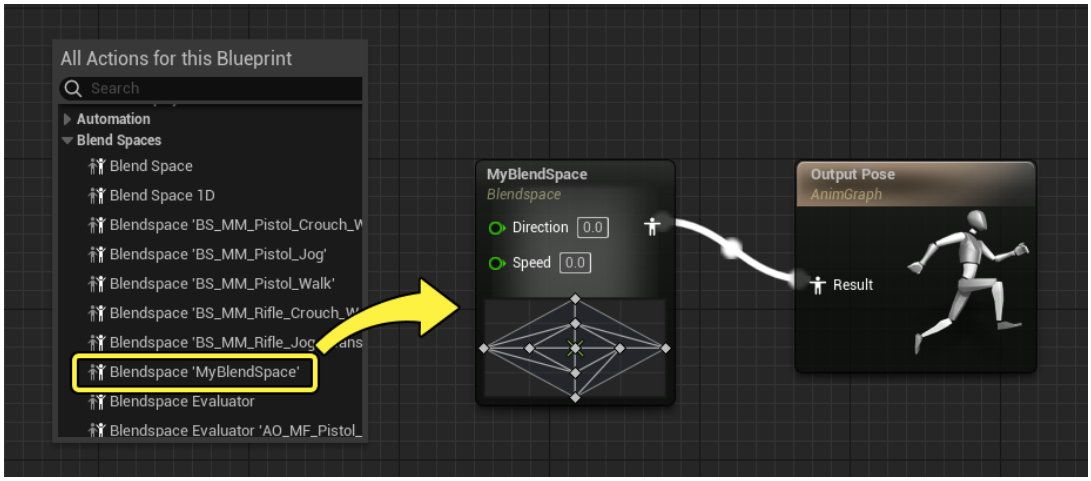

Figure 10. Connecting a blendspace node to the output pose node in animation graph inside an animation blueprint (Unreal Engine 2022)

3.5 Animation Notifications

Animation Notifications are events that happen synchronized with Animation Sequences or Montages (Unreal Engine 2022). They have many uses, for example, making footstep sounds, particle effect spawns and fine-tuning cloth simulations. They are also used for locomotion through Motion Warping.

Creating notifications is done by opening an Animation Sequence or Montage asset and clicking *Notifies* on the timeline (Unreal Engine 2022) (Figure 11). Notifies is a group of tracks, and it has one child by default. More tracks can be added by clicking *Tracks – Add Notify Track.*

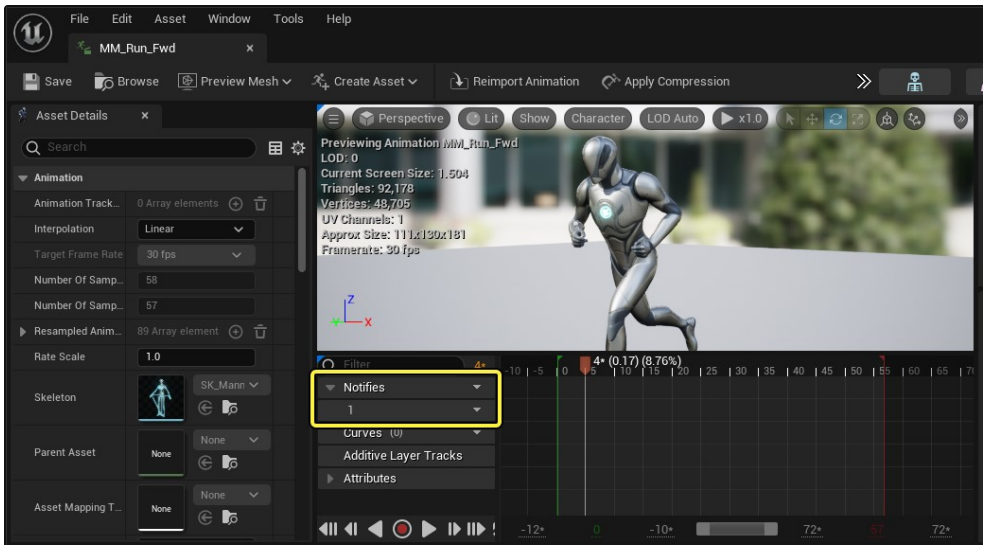

Figure 11. Creation of a new notify (Unreal Engine 2022)

To create a new Notify, the user must right-click a track's timeline graph and select a notify type (Unreal Engine 2022). There are a few different Notifies that can be added: Notify, Notify State, and Sync Marker. The first one is a basic Notify which causes an event to trigger at a specific time. Notify State on the other hand runs for a duration of time, and it has a start, an update, and an end. Sync Marker is a Notifier which informs marker-based animation syncing which is used to blend animations together and synchronizing motion in general.

Once the desired Notify type is selected, it opens a menu of subcategories such as sound or particle effect, which can be dragged to the timeline and edited (Figure 12).

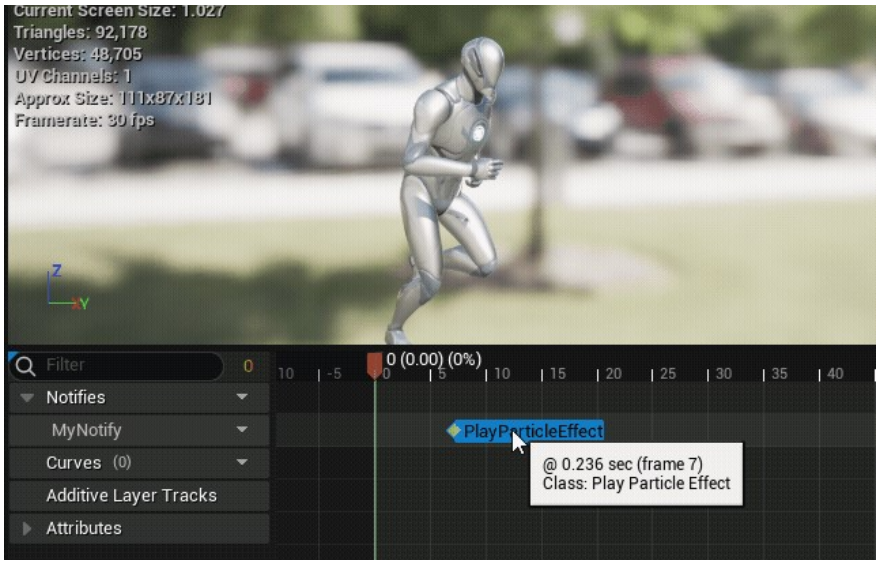

Figure 12. Creating a particle effect notify and placing it on the timeline (Unreal Engine 2022)

When working with Montages, additional Notify types will be visible: Montage Notify, Montage Notify Window, and Disable Root Motion.

3.6 Montage System

Animation Montage system is a tool that allows the user to combine Animation Sequences together into a single asset and control its playback with Blueprints (Unreal Engine 2022). The system can also be used to replicate root motion animations in network games. Montages can be divided to sections which can be played in any order.

Even if there is no need to combine different animations together, PrismaticaDev suggests that using Montages is wise for animations that do not have to loop automatically, so for dodge roll or attack animations for example, while something like running should be done via State Machines which are discussed in Chapter 3.9. (PrismaticaDev 2022).

Montage assets created by right-clicking an Animation Sequence and selecting *Create – Animation Montage* (PrismaticaDev 2022). Double-clicking the new asset will open Animation Sequence Editor which can be used to edit the Montage (Figure 13) (Unreal Engine 2022).

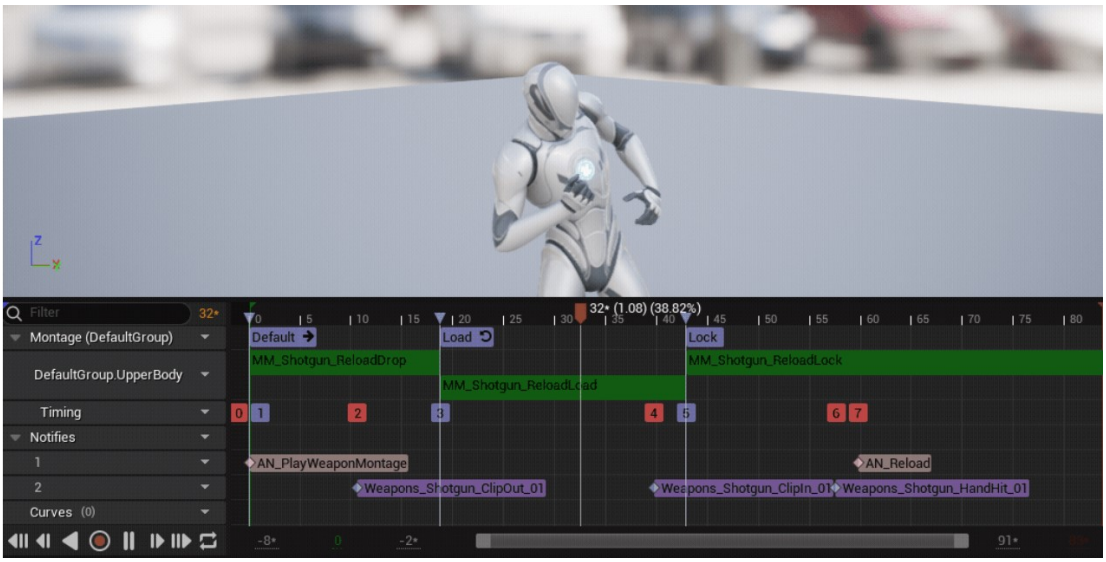

Figure 13. Animation Sequence Editor (Unreal Engine 2022)

Montage sections are created within the editor by right-clicking the top track and selecting *Create New Section* (Unreal Engine 2022). Sections can be moved within the timeline by dragging them from their headings or lines. New animations can be added to the Montage by dragging them from the Asset Browser on the right side and dropping them on the timeline.

3.7 Motion Warping

Motion Warping is a tool to adjust character's root motion to align with targets, such as environment (Unreal Engine 2022) while it performs an animation through Montage. It utilizes Blueprints and Animation Montage workflows.

Motion Warping regions are created by opening the Animation Montage in the Sequence Editor and right-clicking a Notify track, then selecting *Add Notify State – Motion Warping* (Unreal Engine 2022). These regions have start and end times, which should be placed where the Motion Warp is needed to happen. Next, a Motion Warping component must be added to the character Blueprint. This is

done by going to *components – add – Motion Warping* and dragging the node to the event graph. The nodes are connected as presented in Figure 14.

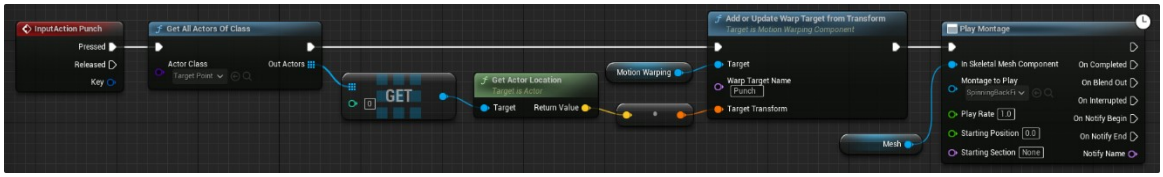

Figure 14. Motion warping set up in the Event Graph (Unreal Engine 2022)

3.8 IK Rig and IK Retargeter

IK Rig is a tool for procedurally adjusting animations by creating Solvers that refine poses of skeletal meshes (Unreal Engine 2022). IK Retargeter can be used to copy animations from one mesh to another with different proportions. IK Retargeter is also used when implementing animation purchased from the marketplace (Epic Games 2022).

IK Retargeter can be used by right-clicking the animation sequence wanted to be copied and selecting *Retarget Animation Assets – Duplicate and Retarget Animation Assets* (Epic Games 2022). This will open a new window where the source and the target can be selected in the upper right corner and new assets named below it. Further down there's a *Retarget* button which will finish the job and close the window.

IK Rig and IK Retargeter can also be accessed by clicking *Add* button in the Content Browser and selecting *Animation – IK Rig*.

3.9 Animation Blueprints

Animation Blueprints are visual scripting tools that control Skeletal Mesh animations during gameplay or simulation (Unreal Engine 2022). Animation Blueprints can be created by two methods. One is to go to the Content Browser and right-click the desired Skeletal Mesh, then select *Create – Animation Blueprint.* An alternative way is clicking *Add* button in the Content Browser,

selecting *Animation – Animation Blueprint* and choosing the correct Skeletal Mesh from a dropdown menu.

In order to have Blueprints affect a character, they must be assigned to that character in the Skeletal Mesh's properties: *Animation Mode* has to be set to *Use Animation Blueprints* and *Animation Class* must be set to the correct Blueprint asset created earlier (Unreal Engine 2022).

Animation Blueprints can be managed in their own editor, which works the same way as Unreal's general Blueprint Editor and can be opened by double-clicking the Blueprint asset (Unreal Engine 2022). The editor consists of several menus and 3 different graphs: Event Graph, Animation Graph (Figure 15) and State Machines Graph. Event Graph is used to create Blueprint-based logic to manage node properties and variables. Animation Graph is for pose-based logic and is used to define the character's pose for each frame. State Machine Graph is for state-based logic and is used for locomotion. These graphs are used together to achieve final animation outcome.

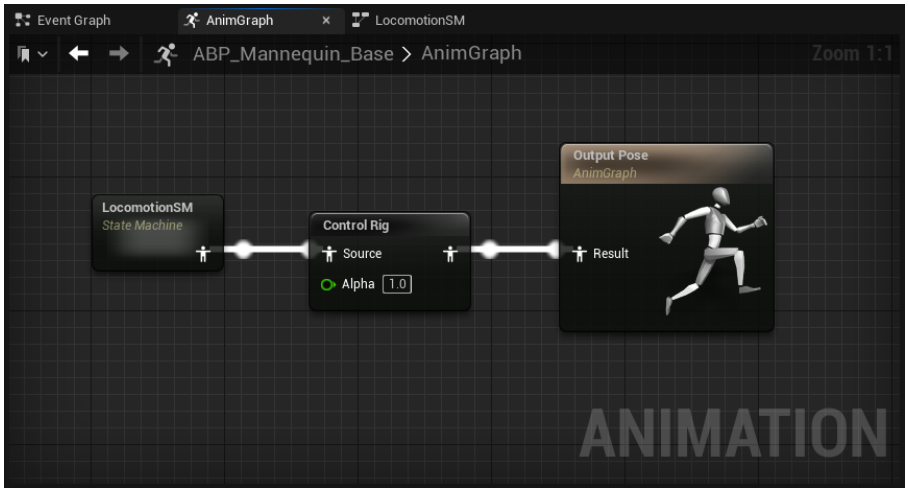

Figure 15. An example of an animation graph (Unreal Engine 2022)

3.10 Sockets

Sockets are attachment points on a Skeletal Mesh which can be used to animate weapons or other objects the character needs to hold (Unreal Engine 2022) or attach effects and other actors to it. Sockets can be created by accessing the

Skeleton Tree and right-clicking the bone the socket is supposed to be attached to (Figure 16). Sockets can be made visible in the animation editor viewport, translated in position, scale and rotation, and copied and pasted.

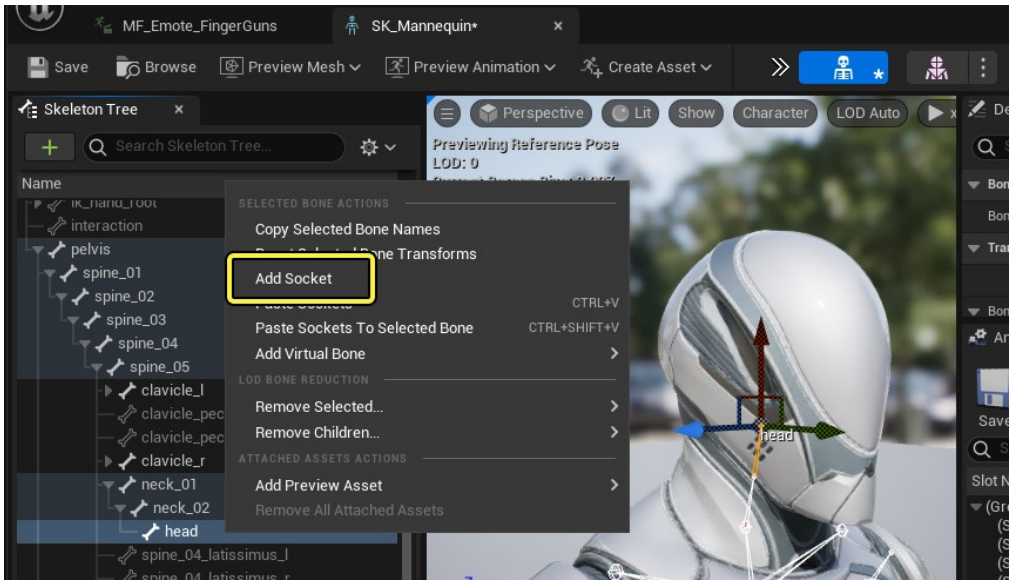

Figure 16. Creation of a socket (Unreal Engine 2022)

There are 3 different types of attachments to the sockets: Basic Level Attachment, Basic Blueprint Attachment and Dynamic Attachment (Unreal Engine 2022). With the Basic Level Attachment, the object will be attached to the socket by default in the corresponding level. Basic Blueprint Attachment will make the object be attached by default in the Skeletal Mesh, thus affecting all levels. Dynamic Attachment is useful when an object needs to be attached and detached during gameplay.

3.11 Live Link

Live Link is a plugin which provides a common interface to transfer animation data into Unreal Engine from external sources such as Maya, Motion Builder or motion capture systems (Unreal Engine 2023). With Live Link the user can see changes made into keyframe animations, or a character moving identically to a mocap actor, in real-time.

After installing the plugin, Live Link client can be accessed by clicking the *Window* drop-down menu and selecting *Live Link* (Unreal Engine 2023). There the source (external software), subject (camera, character etc. in the external software) and other settings can be modified. Next, the character that needs to be animated is set as a Mesh Component in an Actor Blueprint, and also an Animation Blueprint is made for it (Rokoko 2022). The needed node connections in these Blueprints vary by which source software is being used. The Actor Blueprint is then placed in a level and if everything is set up correctly, the character should begin moving accordingly while in game mode. This motion can be recorded and baked into an Animation Sequence.

4 ANIMATIONS FOR THE PROJECT TEAR OF DEATH

Tear of Death is an upcoming 3rd person action game. Combat was designed to consist of aesthetic gymnastic moves, magic and throwing blades. The goal at this stage was to produce the needed animations for the game's prototype with one room boss fight.

The player character, Daphne, is a seasoned fighter and flexible athlete. She performs movements familiar from gymnastics and controls her throwable blades with magic. The animation process was expected to be demanding due to a number of versatile movements, with quality and performance raising concerns. The model (Figure 17) was created using Zbrush, Substance Painter, Blender and MetaHuman Creator. Her 6 identical throwing knives (Figure 18) were created with Zbrush and Substance Painter. A list of her animations included in this thesis is presented in Table 1.

Figure 17. Playable character

Figure 18. Throwing knife

Table 1. Needed animations for the playable character and the used methods. Almost all animations also required the usage of Animation Blueprint.

Animation	Used methods
Idle	IK Retargeting/Swapping skeleton
Running	IK Retargeting, 1D Blendspace, Swapping skeleton
Jumping	IK Retargeting/Swapping skeleton
Falling	IK Retargeting/Swapping skeleton
Crouching	Importing, IK Retargeting, 1D Blendspace
Melee attack	Control Rig, Sequencer, Montage
Throwing	Control Rig, Sequencer, Montage, 2D Aim Offset Blendspace, Layers
Catching	Control Rig, Sequencer, Montage
Drawing and sheathing a weapon	Sockets, Control Rig, Sequencer, Montages, Animation Notifies
Roll	Control Rig, Sequencer, Montage
High Jump	Control Rig, Sequencer
Handspring	Control Rig, Sequencer, Montage
Vaulting	Control Rig, Sequencer, Motion Warp, Montage

4.1 Animation Blueprint & Idle, Running, Jumping and Falling

Using MetaHuman Creator provided the character with a skeleton which is by default similar to the default character, often referred to as Mannequin, skeleton in Unreal Engine 5. Therefore, it was decided that animations for idle, running, falling, and jumping would be created simply by retargeting Unreal's default 3D character's existing animations onto my player character.

This was first accomplished by following Jobutsu's tutorial "Metahuman as third person – Unreal Engine 5.1" which guides how to use a MetaHuman as playable character inside Unreal's 3rd person template (Jobutsu 2022). First, the default character's Blueprint, called 3rd Person Character Blueprint, was set to be the parent of my MetaHuman character's Blueprint and then the default character's Event Graph nodes were copied over to my character's Event Graph. These new nodes were put to action by connecting them to the start node. Next, my character's transforms were zeroed out, the default character made invisible and live retargeting turned on. Retargeting settings were tweaked slightly to adjust my character's arm movements.

While the outcome was brilliant, the disadvantage of using this method of retargeting animations was that any new animations, such as higher jump alternative, had to be made on the 3rd Person Character Blueprint. This blueprint had some features that caused changes to leg animations be impossible as legs would be just locked in place during the new animations. This problem was solved by disconnecting control rig node in the Animation Blueprint, which however caused the feet to no longer touch the ground seamlessly when character was standing on uneven surfaces. Another problem, which is explained thoroughly in Chapter 4.4, occurred when creating sockets for the weapon animations.

As an alternative method, creating movement Blueprints from scratch and copying the animations was tried. Since this project was blank, idle, walk, run, and jump Animation Sequences were obtained by first adding the Unreal 3rd person pack to this new project.

Since MetaHuman mesh consists of two separate parts, body and face, retargeting the animations made for Unreal 4's Mannequin, such as crouching in Chapter 4.3, looked problematic. Pixel Prof (2021) advices to first export the animations from Unreal, reimport them with skeleton set to MetaHuman's skeleton and then change the Bone Translation Retargeting from *Animation* to *Skeleton* for every bone except root. While this fixes scaling problems, there is still a problem with the character's shoulders and head being in odd positions (Figure 19). Fixing this by retargeting the animations to Unreal 5 Mannequin before the export was attempted, and it solved the problem. It was also discovered that actually the export and reimport are unnecessary; the skeleton can be swapped inside Unreal by right clicking the Animation Sequence and selecting *Replace Skeleton*.

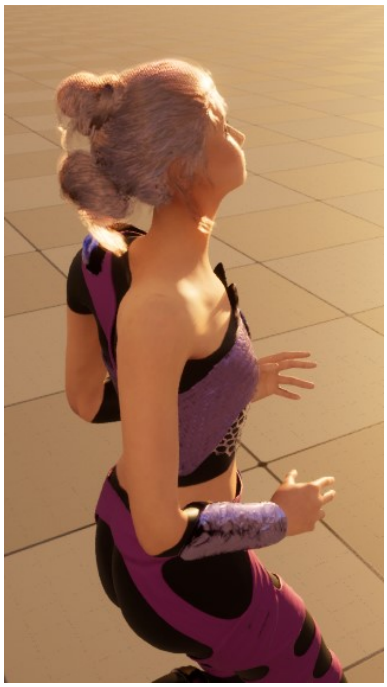

Figure 19. Character's shoulders and head are in odd positions.

Animations were working fine, but this method requires a lot more programming than simply replacing the character in Unreal's 3rd Person Character Template. Therefore, a third method was tried, this time by following Your Sandbox's tutorial "Replace Character with Metahuman in UE5. IK foot issue solved" (Your Sandbox 2022). In this tutorial, MetaHuman's Mesh Components (torso, legs, feet etc.), along with LODSync, in the MetaHuman's Blueprint are copied and pasted into

3rd Person Character's Blueprint and parented under mesh. Translations are zeroed out and Mannequin mesh is replaced by MetaHuman's body mesh. Next, each Mesh Component is assigned with Mannequin's skeleton instead of the MetaHuman skeleton they were using originally.

Unfortunately, Unreal crashed when Mannequin's skeleton was being assigned to the MetaHuman Mesh Components. This was attempted a second time with the same outcome. Further investigation in Unreal community's forums hinted that this might be a bug in the latest engine version, 5.1, which I was using for my project, and therefore this method had to be abandoned. It is likely that even if this method would have worked, the same problems seen with the first used method would have followed.

The project was continued with the second solution presented, where Blueprints would be created from scratch. Eric V. Tuber claims in his tutorial "(NEW UE 5.1) How to replace your Metahuman as a ThirdPerson character in Unreal Engine 5.1" that this is actually the method professional game developers should be using instead of the other methods, which are directed towards hobbyists or film makers to get their characters moving quickly with minimal effort (Eric V. Tuber 2022).

So, after Character Blueprint that controls movement, camera, inputs, etc. was made, an Animation Blueprint was created as described in Chapter 3.9. The State Machines inside Animation Graph were made largely accordingly to Gorka Champion's tutorial "Unreal Engine 5 RPG Tutorial Series - #2: Locomotion - Blendspace, Crouching and Procedural Leaning!" (Gorka Champion 2023). The first State Machine was called "Locomotion", and it contained States for idle and walk/run. Idle animation was connected to the Output Animation Pose node inside the State, and walk/run Blend Space was connected respectively inside its State. The Blend Space node has by default a *speed* pin on it, which was dragged out and promoted to a variable. Next, inside the transition from idle to walk/run state, the speed variable was connected to a node that checks whether it is above 0, which was then connected to result node. Respectively, transition

the other way was made to check that speed is equal or less than 0. Finally, in Animation Graph a cache was made between the State Machine and Final Output Pose node to store its data. Figure 20 shows the Locomotion State Machine and connections inside idle and walk/run States.

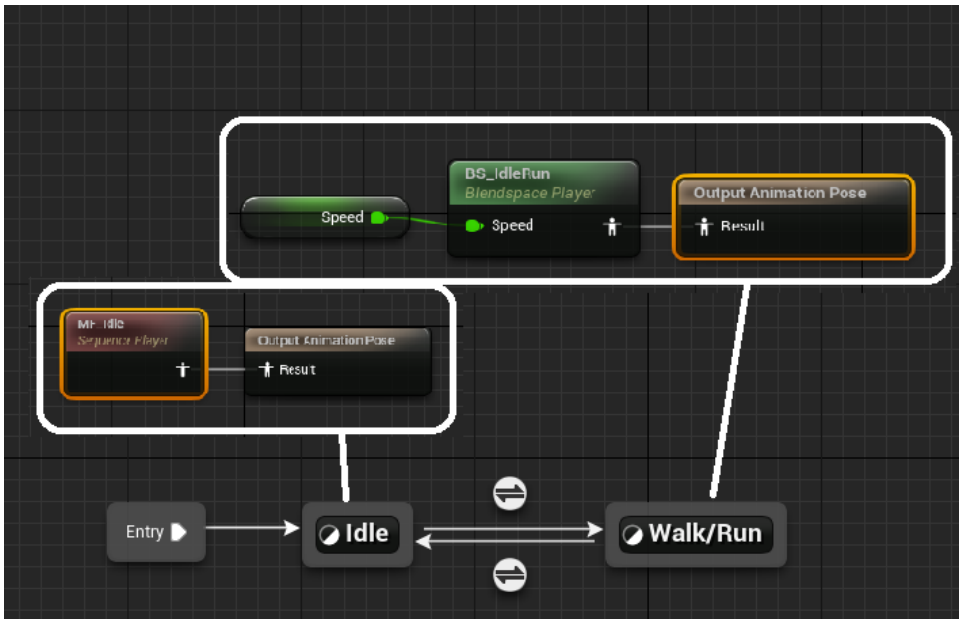

Figure 20. Locomotion State Machine.

Next, another State Machine was made which was called “Main”, as it was planned to contain all the other animations the character shall be using. A Default Slot, which apparently is useful for Animation Montages, was added between the State Machine and the Final Output Pose. A State called “Unarmed Locomotion” was created and its Output Animation Pose was linked to the cache made earlier. After that, Event Graph was tinkered to pass the character’s velocity on to the State Machines as shown in Figure 21.

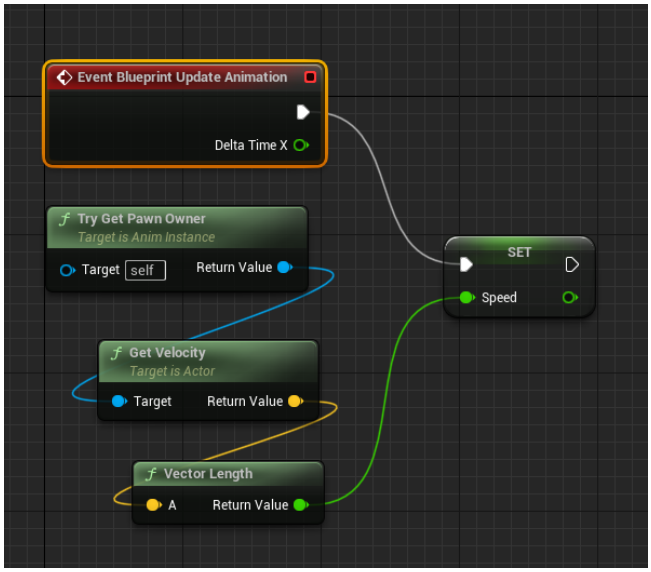

Figure 21. Animation Blueprint's Event Graph is set to pass character's velocity on to the State Machines.

The process continued by creating States for jumping and falling. These were made exactly the same way as Unreal's Mannequin has them, with a State Alias called "Is Falling" handling the transitions, which makes the graphs cleaner. A new vector variable was made to track the vertical movement and it was connected as shown in Figure 22 inside the transition to jump from the State Alias. Transition from Jump to Falling was set to work by automatic rule, which makes the transition happen as soon as the animation has finished playing. This was done by merely clicking a checkbox at the transition Detail panel.

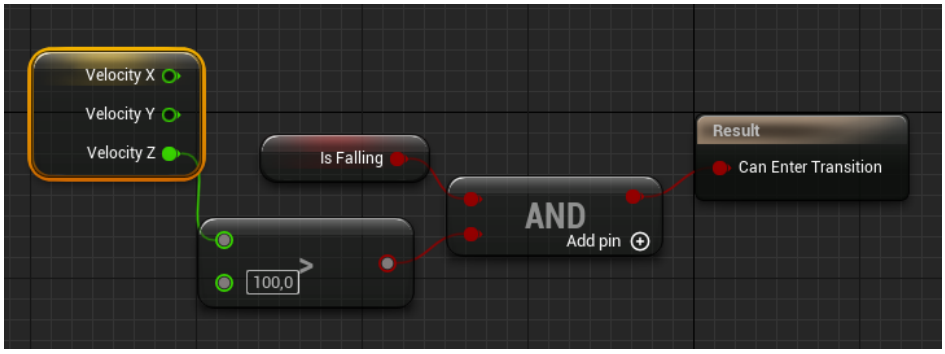

Figure 22. Nodes inside transition from Is Falling to Jump.

Landing was made by first making a State Alias called "Is Landing" and a transition from it to a State called "Land", which was put to transition into Unarmed Locomotion state. This transition was made to happen by the automatic

rule presented earlier and the transition from the state alias to Land was set to happen if boolean variable “Is Falling” is false. The completed State Machine is presented in Figure 23.

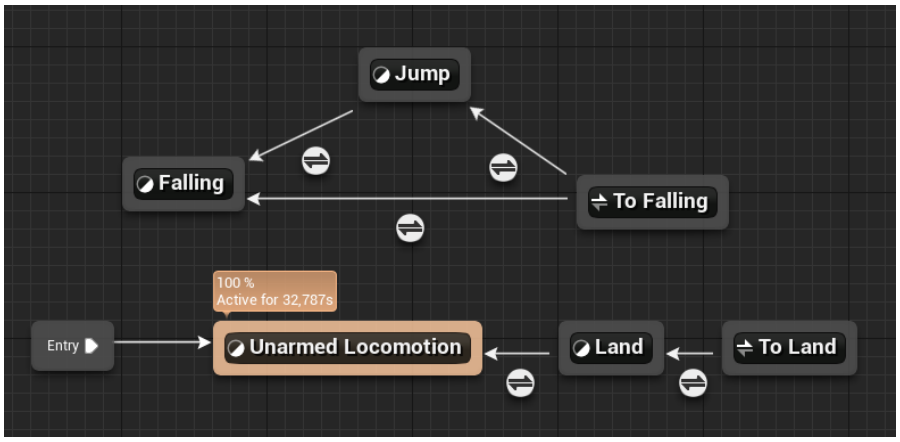

Figure 23. Main State Machine completed.

Since my landing animation was made as an additive animation on top of idle/run animations, it needed an extra node along with the Locomotion cache to play correctly. The setup is presented in Figure 24.

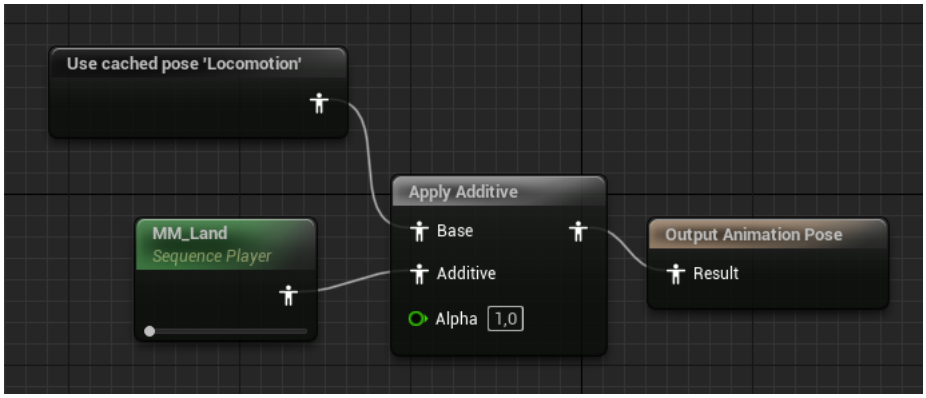

Figure 24. Node setup inside the State “Land”.

Event Graph was updated with new nodes as shown in Figure 25. Once everything was done, the animations were working perfectly.

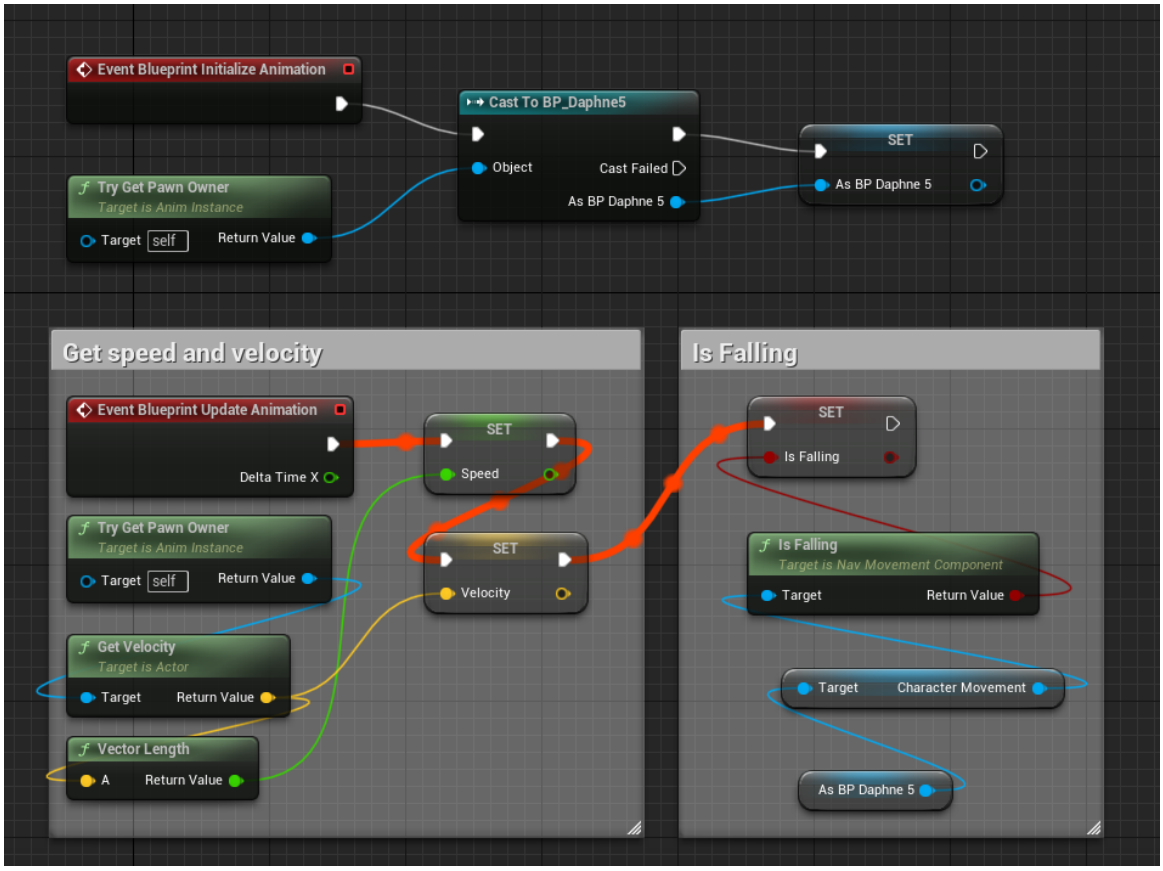

Figure 25. Final Event Graph.

4.2 High Jump

The character was designed to also have a higher jump alternative inspired by gymnastic moves. Creating a new Animation Sequence with the Sequencer tool by tweaking the basic jump Animation Sequence was chosen as the method. This was thought ensure that the character's movements blend better with other animations since the start and end poses could be kept the same as they were.

This method required baking the Animation Sequence into a Control Rig in Sequencer with a keyframe every second (Unreal Fest 2022). The middle keyframes were deleted and new ones created at the highest point of the jump, shown in Figure 26. This method gave the best-looking Animation Sequence so far, and it worked seamlessly when tested by replacing the default jump with it.

Figure 26. In the process of making the high jump animation in Sequencer. This was done with the Unreal mannequin as it used the same skeleton and blueprint as our character and there was uncertainty whether it works to animate with my character at this point of the project.

High jump was implemented by making a new State to the Main State Machine. It was connected the same way as basic jump, and a new boolean variable called “isHighJumping” was created. This was connected to the transition as shown in Figure 27. The Event Graph was updated as shown in Figure 28 to link it to the Character Blueprint which controls movement.

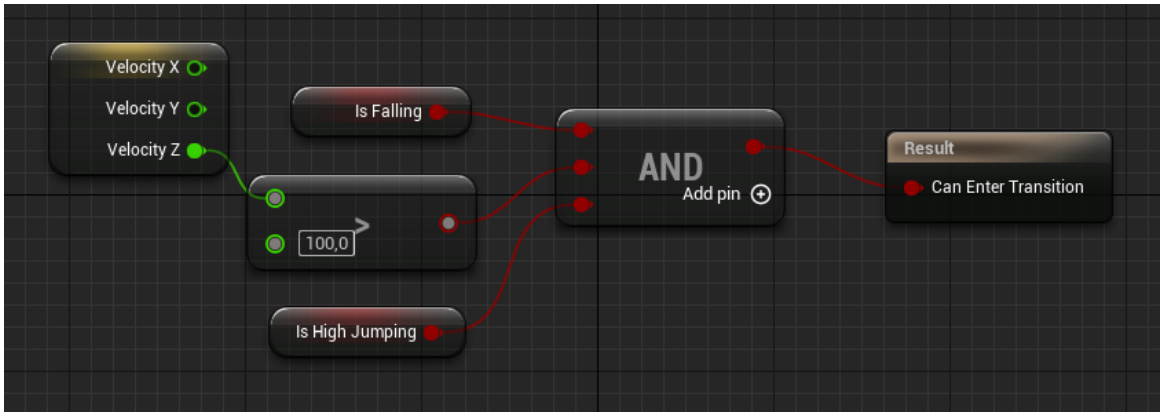

Figure 27. Connections inside the transition from “To Falling” to “High Jump”. In the transition of basic jump, a not boolean node was added between “Is High Jumping” and the “and” node to make it behave the opposite way.

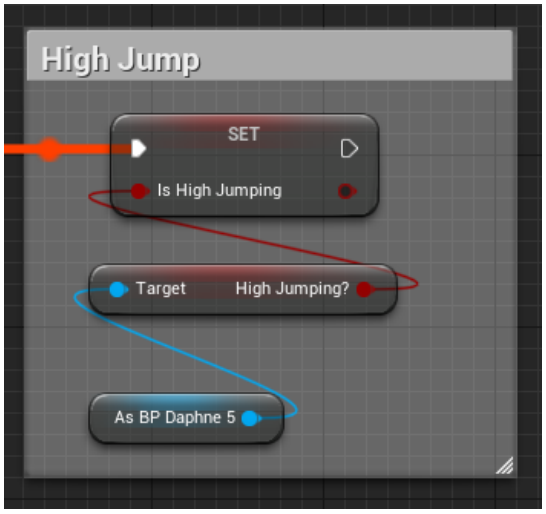

Figure 28. Connections of High Jump in the Event Graph inside Animation Blueprint. “High Jumping?” is a new variable which was made in the Character Blueprint (BP Daphne 5), and it tells the Animation Blueprint whether the key to activate High Jump was pressed or not.

4.3 Crouching

As crouching is a very common yet simple movement with little variation possible, it was decided that the animation would be imported from an external source and retargeted to the character. A suitable animation was found on Unreal’s Marketplace, as part of MCO Mocap Basics pack made by MoCap Online.

Since the animations had been made for the Unreal 4 Mannequin’s skeleton, the process started by retargeting them to Unreal 5 Mannequin’s skeleton, as described in Chapter 3.8, which was then replaced by MetaHuman skeleton. A 1D Blendspace was made to blend between idle and moving crouch animations and then this Blendspace, along with a couple of new nodes and a bool variable, were connected to the idle-walk-run State Machine inside the Animation Blueprint as shown in Figure 29. The Event Graph was updated similar to the High Jump.

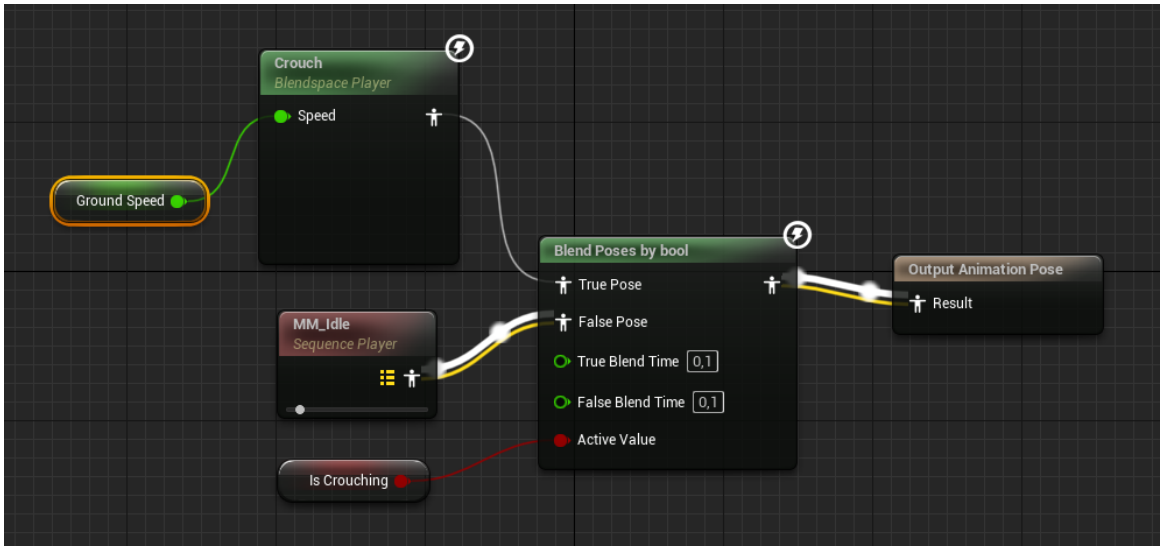

Figure 29. Crouch Blendspace connected inside idle State in the idle-walk-run State Machine.

The imported animation package included both animations with root motion and without. Initially the one without it was chosen to be used and it worked perfectly. For experimental reasons the process was repeated with the animations where root motion was included. This resulted in the character moving forward more than was intended and getting further away from the camera, so it was established that root motion should not be present in this type of animations. It was also discovered that animations that have root motion can be fixed into not having it by ticking *Enable Root Motion* and *Force Root Lock* boxes in the animation settings.

4.4 Knives and sheathes

The next task was to create an animation for unsheathing and sheathing each of the character's 6 knives she carries attached to her thighs and make the game understand which one to unsheathe when.

The first thing to do was to create Sockets where the weapons would be attached to when they are sheathed. Since my character was using Unreal Mannequin's Blueprint at this point, the question was whether the Sockets should be attached to Mannequin's skeleton or the MetaHuman's skeleton. It did not seem to be possible to enable my character's armor to be visible when creating Sockets to Mannequin's skeleton which was a critical problem as weapons would need to be

placed exactly on top of the sheathes on my character's armor. Therefore, the Sockets were made on the MetaHuman skeleton and Blueprints would simply be redone to my character if it would not work otherwise.

Sockets were created as described in Chapter 3.10. It was discovered that the dagger would have to be coded to spawn in the MetaHuman's Blueprint instead of the parent 3rd Person Character Blueprint. However, the Socket would indeed need to be put on the Mannequin's skeleton. As predicted, this made the correct knife placement impossible, as even if the transform values were copied over from the MetaHuman's skeleton Sockets, they would place incorrectly due to the transforms being relative to the Mannequin skeleton which is taller than my MetaHuman character. Even if my character did not have sheathes, placing the weapons accurately close to the body would be difficult.

In standing position, it looked like the placement issue would have been somewhat doable by editing the Socket placement with trial-and-error mentality until satisfactory result was reached. However, when crouching the issue was more visible and not even related to the usage of Mannequin's skeleton, as the knife would transform exactly the same as the bone it is parented to, while the sheathes were skinned to follow the thighs which was a result of blending skin weights between several different bones. Attaching the Socket to different thigh bones was tried to see if any of them would work better, but to no avail. The issue is demonstrated in Figure 30.

Figure 30. Knives do not overlap with the sheathes correctly.

Four solutions were come up with. First, the easiest one – surrender to the same solution – as most video games do and delete sheathes. The second solution would be to detach the sheathes and handle them similar to the knives with Sockets. This solution would also allow the character to have differently shaped weapons as upgrades or alternatives later in the game. The third solution would be to add extra bones to the skeleton and skin the knives to follow them, and then animate them to be in correct places in each movement. This was thought to give the best-looking result but would limit the character to one weapon type for the entire game. Fourth solution would be to add the thigh driver bones to the armor skinning and make the sheathes follow only them, similar to Sockets.

Solution number two was chosen and after the sheathes were detached from the armor in Blender, the Socket creation process continued. This is also the point where the project was switched from the 3rd Person Template to a new project with custom Blueprints as explained in Chapter 4.1. This method worked well as shown in Figure 31.

Figure 31. Character crouching with weapons following seamlessly.

4.5 Sheathing and Drawing Weapons

In order to hold a weapon on her hand, the character needed a new Socket for each of her hands. They were made the usual way. After that, an actual Animation Sequence for the sheathing was needed for each weapon. These were made using Control Rig and Sequencer, and the same animation was used for both sheathing and unsheathing as that seemed to work fine.

The animations were initially made so that the character starts and ends the animation from A-pose, with the thought that it would make them blend better, but this was later discovered to be trivial, as Unreal would blend the animations in any case. Fingers were not animated to grab the weapon yet, as the plan was to use additive animation to accomplish the grip whenever the character would be holding a weapon yet performing other animations simultaneously. The grip was indeed accomplished by Bone Layers; first the Event Graph was set up the same way as with crouching and high jump, with a new boolean called “Holding a weapon”, and then the Animation Graph was updated with a Layered Blend per Bone node, described more in depth in the next paragraph. Additionally, a new State Machine was connected to control the grip locomotion, and the grip animation itself was created in Sequencer.

The animations for drawing and sheathing were turned into Montages as recommended by Unreal RPG Mastery (2022) and PrismaticaDev (2022), and an Animation Notify was added on the timeline, so that a weapon Blueprint could change the weapon's attachment to the hand Socket at the right frame. Next, Bone Layers were used to make the lower body follow other animations while upper body would be performing drawing or sheathing animation. This was done by adding a saved cache for the Main State Machine in the Animation Graph, then adding Use Cached Pose node to the graph and connecting that with Layered Blend per Bone. Then, the Default Slot node was duplicated, and the copy was set to use Upper Body Slot. Layered Blend per Bone node's configuration was tinkered by adding array elements to branch filter accordingly to the bones desired to be blended, as shown in Figure 32. Finally, the nodes were connected as shown in Figure 33 and the Montages were set to use the new Upper Body Slot.

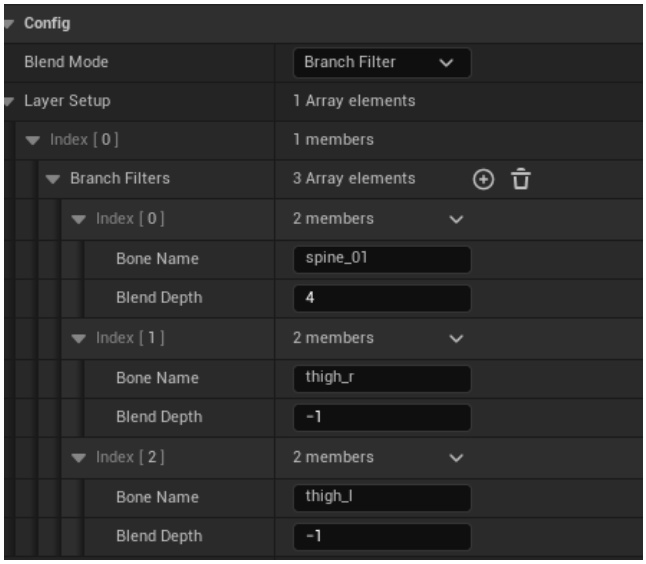

Figure 32. Layered Blend per Bone -node's configuration settings.

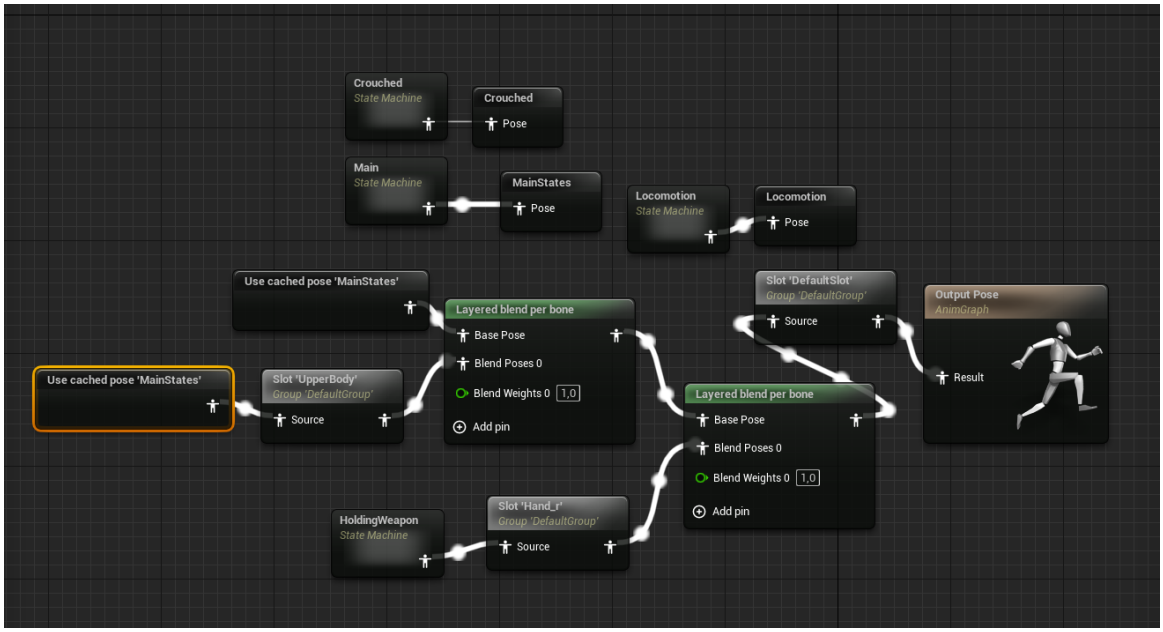

Figure 33. Animation Graph as a whole at this point of the project.

The resulting drawing and sheathing animations were satisfactory when the character was idle, running or jumping, but showed issues with crouching as character's hands were cutting through her legs. Since altering the leg animations was not an option assuming the crouch position was needed, it seemed that making a separate animation for drawing and sheathing in crouch position would be the best, if not the only course to take. This worked, and after a bit of finetuning in Sequencer, the animations were good in all poses. The new Montages were set to use the same Upper Body Slot as the regular sheathing Montages, and therefore no changes to the Animation Graph were needed.

4.6 Melee attack, throwing and catching

Attack animations were made next. Two variants were thought to be needed for the melee attack to look realistic, one swinging from right to left and the other from left to right. Punching animation was also needed in case the character ran out of knives.

In order to achieve a realistic look, all these combat animations were attempted to be created with Motion Capture and Live Link, but for an unknown reason Live Link did not work accordingly, leaving the character sailing around while standing

stiff in an A-pose. It is suspected that there was incompatibility between the software versions and the plugin, as it seemed that the bone names did not get translated correctly between the software. The animations were therefore made by keyframing in Sequencer, and the melee attack was combined with the punching to save time.

The character was supposed to be able to catch and throw the knives with both of her hands, so these animations were made for both sides in Sequencer using the character's Control Rig. They were initially made with only the upper body moving, which worked well in combination with other animations except idle, where it was clearly required that the character would alter her standing position when throwing. Therefore, the animations were altered to also contain leg movement and the Montages made from them were set to use two slots, the Upper Body and the Default Slot, which were then placed in the Animation Graph respectively to enable or disable lower body movement during each animation combination. Animation Notifies were not used this time, as the dagger dispatch was handled entirely through coding.

Editing the leg movement was achieved in the Sequencer. However, as an experiment it was tried in Animation Sequence Editor first, as it was thought to be faster since I already had a base Sequence with the upper body movement. The first issue was that the character's head and shoulders were slightly separated in the editor viewport due to MetaHuman properties, which made estimating correct movement difficult. The second problem was that no controls are visible in this editor, meaning that each bone has to be selected in order to move it, which was troublesome for bones like pelvis that are surrounded by other bones. The most problematic though, was that no IK handles existed, making feet and pelvis transforms slow. It was established that Animation Sequence Editor was mostly suited for trimming animations and making slight changes to the pose during the whole duration of an animation, while Sequencer should be used for actual animation making.

During gameplay testing it became apparent that an additional aiming position, where the character would stay until throwing command was given, was needed before the throw itself. Therefore, the throwing animation was cut to only contain the throw, and the aiming before it was separated into its own Sequence and tweaked to contain slight idle movement so that the character would seem lively during it. The cutting was performed in Animation Sequence Editor and the idle adjustments were made in Sequencer. It was desired that the character would look and aim to the same direction the gameplay camera would be looking, and therefore a 2D aim offset Blendspace was made for the aiming animation. This required a total of 9 aiming Animation Sequences with each one having the character look and aim different way. These were produced by altering the bone transitions of the idle aim Sequence in Animation Sequence Editor.

Melee attacks and parry turned out to be simple. Animations were made in Sequencer, turned into Montages and played with Blueprints. An additional idle in combat animation and State was also created to make combat look more natural.

4.7 Vaulting over obstacles and walls

Next up the character needed ways to get over obstacles and small walls. In order to make the same animation function for different sized obstacles, the plan was to use Motion Warping to control root transforms.

The process began by creating a vaulting animation in Sequencer. The animation was made using Unreal Mannequin to allow modification of the animation later, as it had been discovered during the creation of the combat animations that baking an existing Sequence into Control Rig was no longer working for my MetaHuman after its Blueprint had been turned into a Character Blueprint and parented under mesh to allow programming functions to work correctly. This could have been perhaps solved by better programming, but since using the Mannequin was equally practical as using the real character, it was decided to continue animating with the Mannequin and replacing the skeleton of the Sequences afterwards.

Once the Sequence was finalized, it was opened in the Sequence Editor and *enable root motion* box was ticked to make the animation run in place as was needed for the game animation to function as required with the Motion Warp. At this point it was realized that using the global control to rotate the character's body mid-flight had been a mistake as this movement got disabled. The Animation Sequence was therefore remade by rotating character's pelvis instead. The incorrect global control rotation can be seen in Figure 34 and correct rotation made by pelvis in Figure 35.

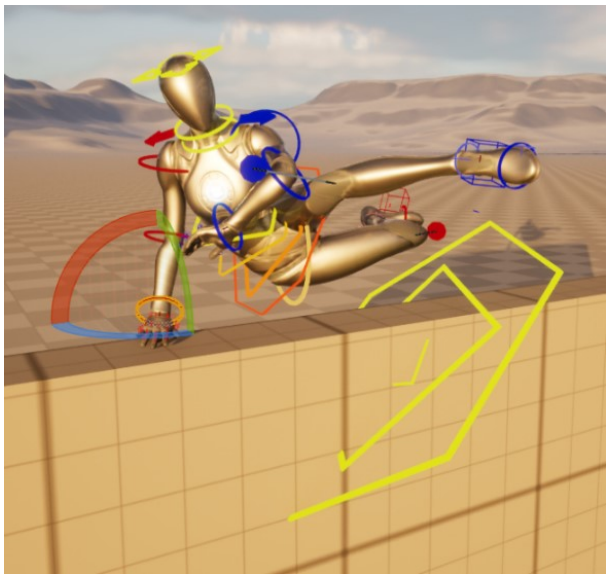

Figure 34. Global control (big yellow control) was used incorrectly to make the character's body rotate.

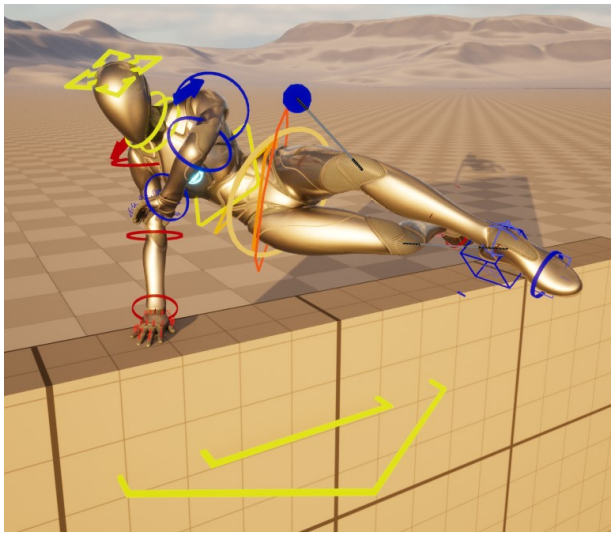

Figure 35. Rotation achieved by rotating the pelvis control (orange angular). Here the pose has also been improved to enhance the forward motion.

Next, the Sequence was turned into a Montage and a Motion Warping Animation Notify-State was added on the timeline for the duration of character touching an obstacle with her hand, and also another for when she is landing to the ground afterwards. A third Notify-State was added in the middle of the Montage as it appeared to make the animation slightly smoother.

Each Notify-State was given a name to access them with the Character Blueprint. There are also several settings that can be changed for each Motion Warp Notify-State. First off Root Motion Modifier, which can be set to *None, Scale*, or *Skew Warp*. Skew Warp was recommended by Gorka Champion (2023), and it seemed to function well with my animation. The two others were tested too, but they both seem to disable the naming option, making it impossible to reference the Notify-States from the Blueprints therefore meaning that they are unusable in this case. The settings also allow to choose to warp translation, warp rotation and ignore z-axis. In my project warp translation was enabled for all the Notify-States, while z-axis was ignored only in the middle one. Rotation was enabled only on the first one.

The Notify-States' final position on the timeline (Figure 36) was determined by testing their functionality in game. The final outcome was pleasant, and the character leaped over different walls with ease.

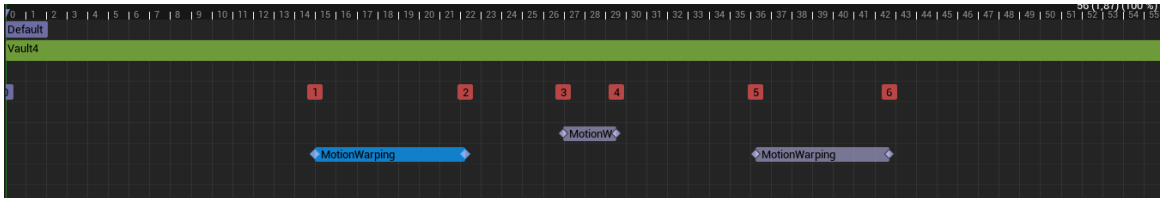

Figure 36. Motion Warp Notify-States positioned on the Montage's timeline.

4.8 Handspring and Roll

The highly athletic character also needed some special moves for dodging and for aesthetic purposes. Motion capture was not an alternative for these animations as there was no actress available who would be capable of performing such high-level acrobatic moves. Therefore, the plan was to make the

animations in Sequencer while using reference picture on the background to ensure realistic and smooth look.

The process was started with backwards handspring, which was keyframe animated in Sequencer using Mannequin's Control Rig with a reference picture placed on a plane behind. It turned out the animation looked better when the reference was not followed exactly but with small modifications in especially how long and high the jumps would be. The start and end of the action were also exaggerated as instructed by one the 12 animation principles, anticipation. Tweaking the timing of each keyframe by moving them on the timeline was also essential to get a realistic feeling to the movement.

The Control Rig is built to have both IK and FK controls for legs and arms, and these can be switched on and off on whichever frame. The animation was first made by having the legs set to IK for the whole duration, but this ended up making problems with the foot roll rotations as my character was flipping full 360 degrees during the animation. All the keyframes of the foot roll controls were removed and the feet rotations were made with foot IK control instead, but this did not seem to fix the problem. Therefore, the animation was remade by using FK controls and only turning on IK when the character had her feet on the ground. Similarly, the arm IK was turned on when she was standing on her arms in the middle frames of the animation. Turning the IK/FK switch created a small clipping movement initially, but this was solved by carefully adjusting the positions of hands and feet to be identical in both modes at the timestamp the switch was set to happen.

Different interpolation modes for the keys were also tested here. Cubic is the default key type, and it creates a smooth curve between keys, thus slowing down the animation close to keys and speeding it up between keys. Linear keeps the same speed throughout the animation, and constant makes the animation "jump" from one key to the next without any smoothness. For my animation there did not seem to be a significant difference between Linear and Cubic, but this is probably dependent on how the animation is made and what is desired. When exporting

the animation from Sequencer into an Animation Sequence, it was discovered that interpolation can be set to either Linear or Step. The first one creates a smooth animation while the latter creates a staggering animation that jumps from frame to frame, which however cannot be seen when playing the Sequence in the Sequence Editor but becomes apparent when a Montage is made and played while testing the game.

Automatic backwards movement was desired when performing the handspring in game. This could have been implemented with Motion Warping similarly to the vaulting, but this time using solely the animation's root motion was tried as a faster solution. As we learned in the crouch chapter, *Enable Root Motion* has to be ticked in the animation Sequence settings to prevent the character from "running away" from the camera and her capsule collider.

A single handspring looked good now, but there was an issue if the player wanted to keep handspringing, as the character would do her slow start and end of the jump between every handspring instead of continuing without stopping. To solve this, a second Montage was made which only contained the middle part of the animation. Looping this however caused the character to gain more and more z axis value with each jump, making her ascend to the skies. Different blending settings were tried to solve the issue. Making the blending start later did stop the z multiplying but caused the animation clip and waver disturbingly. Using the original handspring Montage while inserting two new sections (Figure 37) on it was tried as alternative method. These different sections were set to play accordingly: first section when the loop was supposed to start, and the second when player stopped the handspring. This brought smoother results, although character was still ascending slightly. This issue was fixed by using IK Rig to adjust feet placement, described in detail in Chapter 4.9.

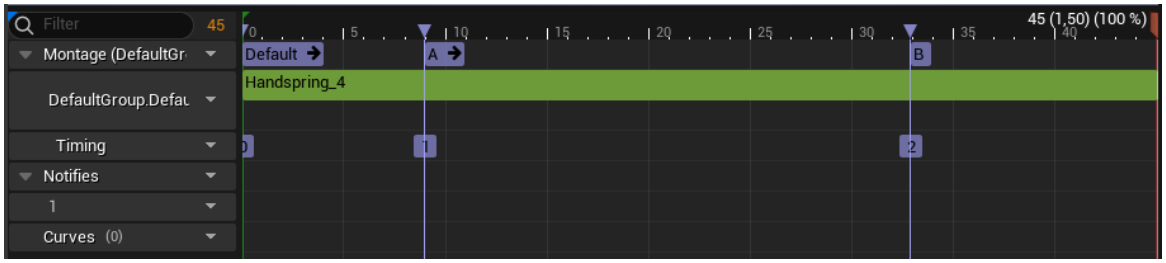

Figure 37. New section (A and B) inserted into the montage.

Making the roll was similar but easier process. There was no need to make it loop and it did not have z axis transform. It seemed to perform well even on up- and downhills and if character fell off from a ramp, she would fall accordingly.

4.9 Feet placement

While the character was standing on uneven surfaces, like ramps or stairs, her feet would not adjust realistically, but instead float partly in the air (Figure 38). Unreal 5's Mannequin uses Control Rig with Foot IK to fix this, and I tried the same. This did fix the problem, but instead the character's feet were locked in place permanently, so running or any other motion was ruined. Investigation revealed that for this to work, the skeleton needs to have Foot IK Bones, which the Mannequin has, but surprisingly MetaHuman does not.

Figure 38. The character's feet are not touching the ground correctly when she is standing on a ramp.

An alternative way to solve the issue was found by following UNF Games tutorial "Foot IK Tutorial in Unreal Engine 5 with Custom Character - Realistic Foot Placement" (UNF Games 2022). The process began by creating an IK Rig for the

character, but my character being a MetaHuman already had it. Hand and leg IK goals existed already by default, but pelvis goal had to be added manually. These goals can be created as children of joints in the rig, and they direct where the joints in question should be placed. Next, a new node called IK Rig was added to the Animation Blueprint before the output pose and the correct IK Rig was selected at the Details tab. To get the foot and pelvis goal position parameters visible in the node, the parameters had to be set to exposed in the IK Rig. These parameters were then promoted to variables.

The setup required some programming in the Animation Blueprint's Event Graph, where Sphere Tracing was utilized to track where the feet were supposed to be placed. The results were quite good for the feet, but unfortunately the hand IKs reacted to the leg movement and gained the same offset as the foot that was on a higher ground. This was fixed by disconnecting the hand goals from the Full Body IK solver used for the feet placement in the IK Rig asset. Feet were floating above ground slightly, but this was fixed by subtracting some z value in the Blueprint.

Figure 39. Feet position adjusts to the ground and also defines the knee and hips joint rotations.

Figure 39 shows the resulting feet placement on an uneven surface. There are still minor issues with the feet cutting through the ground at ramps, but the result was considered good enough for now.

5 RESULTS

Unreal Engine 5 seems well capable of creation and implementation of animations for different needs. All the goals for my character's movement and animations were met and the outcome was visually and functionally pleasing. It can be said that State Machines, Blendspaces and Montages create the core of game animations, with Notifies and Motion Warping supporting in specific situations. Sockets are needed for a great deal of games, and Animation Blueprint ties everything together. Animation Sequences are the root of everything, and they can be created in engine with Sequencer or imported from an external software.

A suggested workflow, shown in Figure 40, was created according to the findings from Chapter 4. Everything begins by the creation of an animation sequence, which can be achieved in three different ways; keyframe animating or using Live Link together with Sequencer in the engine, or importing from another software. Using any of these ways brings equal results as long as the animation itself is well made. Using Live Link with motion capture is likely the fastest way if gotten to work, but is not suitable for some physically demanding animations, for which keyframe animation is a good option. Animation in the engine requires the character to have a Control Rig.

When animating in Sequencer, the animation can be created so that it has root motion by moving either the global or root control. There was found to be no disadvantage of creating root motion, as it could be easily turned off if needed, and it was useful to have for montages that are supposed to have the character move a default amount during the animation. Key interpolation can be changed from cubic to linear, but my project suggested that this might be trivial for most animations. The start or end pose of the character does not seem to matter in

most cases, as Unreal will blend the animations together in gameplay automatically unless the blending settings are manually changed.

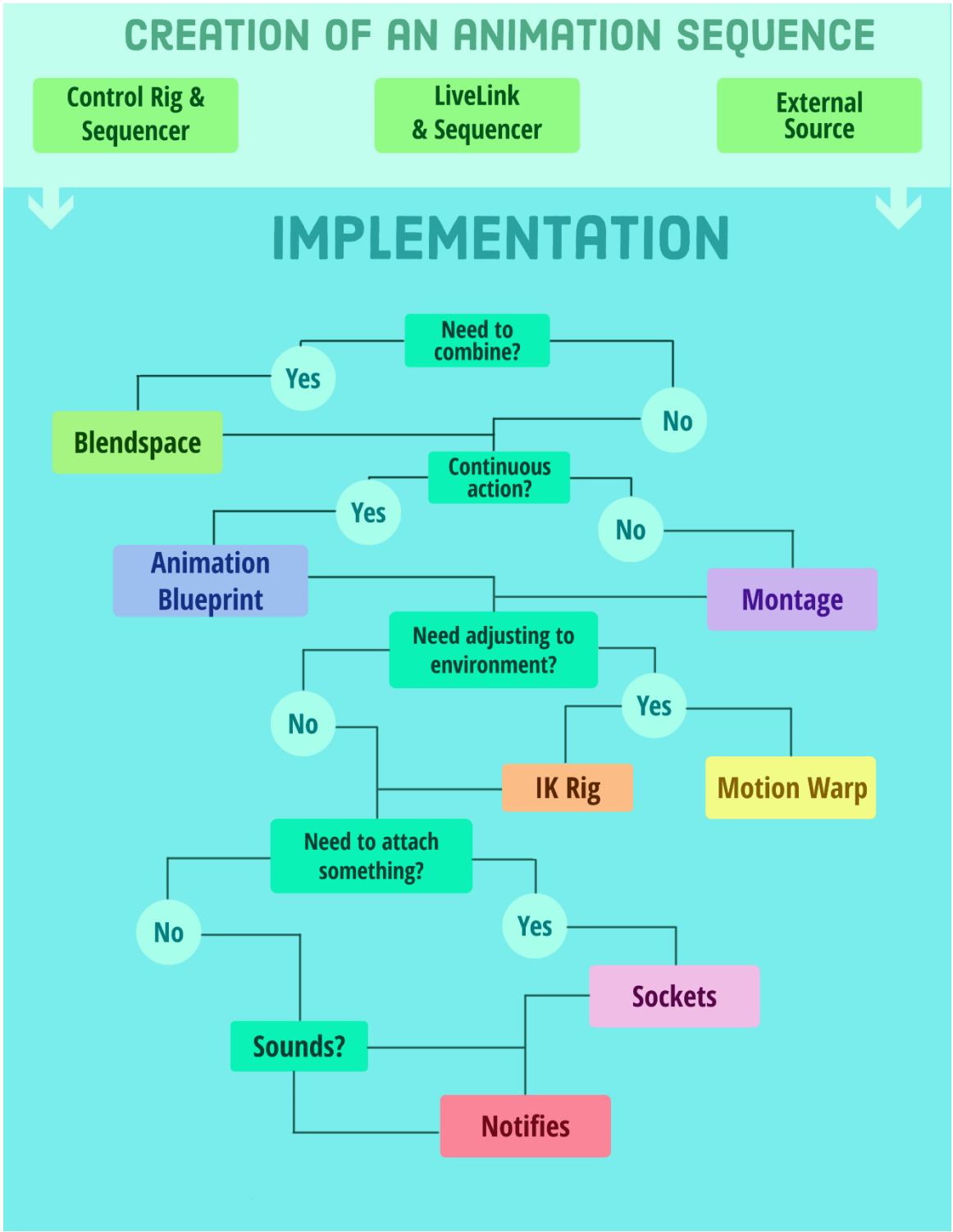

Figure 40. The suggested workflow based on findings of the research.

Once a sequence is obtained, the implementation to the game begins. First thing to consider is whether there is a need to blend several sequences together and make the engine change the motion depending on a value, such as speed or camera angle. For example, running and aiming are typically these kinds of animations. In Unreal Engine this blending is achieved with Blend Spaces. Blend Spaces are easy to make; simply drag and drop a sequence on a graph. Blend Spaces are put into a State Machine similarly to a mere sequence.

The second thing to consider is whether the action is continuous, such as running, or is it only meant to be done once when triggered, such as an attack. Continuous actions are made using State Machines while Montages are recommended for one-time actions, as they can be programmed to play when a key is pressed, or an event triggered. My research suggested that root motion is useful for Montages but not for continuous actions, as the character translation should be controlled by the player in those.

If there is a need to adjust the motion according to the environment or other characters, IK Rig and Motion Warp are useful tools. IK Rig is typically used to control feet and hand end positions, for example to make the feet hit the ground on uneven surfaces. Motion Warp is handy for situations where the character's entire body must adjust and change location during the animation. It is used together with Montages. Both Motion Warp and IK Rig require a fair amount of programming to implement as they require sphere tracing.

When there is a need to attach another object to the character, Sockets are useful. These are made by editing the skeleton of the character and they are tied to a bone's movement. The object can be made to switch from one Socket to another during an animation by programming and using Animation Notifies. Notifies can also be used to play sounds, such as footsteps, and Motion Warp is also a type of Notify.

If an animation only needs to be played on one part of the character, for example upper body, it can be achieved with Layered Blend per Bone -node and different

Slots in the Animation Graph inside Animation Blueprint. Animations can be retargeted from one skeleton to another through IK Retargeting, which requires the skeleton to have an IK Rig, or by using the “replace skeleton” -function, which requires the skeleton’s retargeting options to be set to “animation”.

A summary of different animation features and their typical usage in Unreal Engine 5 is presented in Table 2.

Table 2. A summary of different animation features and their usage in Unreal Engine 5 based on findings of the research.

CONTROL RIG	In-engine rig needed to animate in engine. Can be created to a skeletal mesh that has already been skinned in another software.
SEQUENCER	A tool to keyframe animate or record livelink data into an animation sequence. Can also be used to edit existing animations or to create cutscenes.
BLENDSPACE	Used to blend several sequences together and make the engine change the animation depending on variables. For example walk-run, aiming in different directions.
ANIMATION BLUEPRINT	Controls constant movements through state machines. Used for idle, walk/run, jump or poses character has to stay in, for example parry/block. Also needed to control if montages play in all bones or just some.
MONTAGE	Similar to sequences but recommended for actions that only happen once when triggered. Can also be used to combine several animations. Montages can be triggered by blueprints.
IK RIG	Can be used to adjust feet or hands to the environment. Also needed for IK Retargeting to redirect an animation sequence between different skeletons.
MOTION WARP	A notify state placed on a montage. Used to adjust motion to the environment and objects or characters in it.
SOCKETS	Made on the skeleton of the character. Can be used to attach or detach items, for example weapons.
NOTIFIES	Inserted on montages or sequences. Can be used, for example, to play sounds or trigger blueprints at certain points of the animation. Also needed for Motion Warp.

6 CONCLUSIONS

The thesis managed to answer the research question of how, when, and why use each Unreal Engine 5 animation feature and the productive part succeeded in its goal of creating realistic and athletic animations for a game character. It is a shame that true real-time animation through Live Link motion capture could not be tested, but keyframe and procedural animating were thoroughly experimented on, and many useful observations were made. Implementing the animations remains the same regardless of animation method, so the results of the thesis would not have been affected even if Live Link could have been used.

As the animations were constantly tested in game mode, there are unlikely to be errors in the results, and the thesis should have quite high reliability. As the author is a beginner with Unreal Engine, it is however possible that some features were not used to their full capacity, but given the wide community tutorial research, the thesis should offer a decent overview of the features and their most important functions.

Future research topics could involve further research into more niche uses of the animation features within the engine, and perhaps Live Link especially, together with its different source software.

REFERENCES

Adam Moore. 2021. Unreal Engine 5 Sequencer for Beginners. Available at: https://www.youtube.com/watch?v=YGXrOQ-5ztg&ab_channel=AdamMoore [Accessed 7 Dec. 2022].

Beane, A. 2012. 3D Animation Essentials. Indianapolis: John Wiley & Sons, Inc.

Business of Animation. 2023. Why Real-Time Animation is the Future for 3D Animators. Available at: https://businessofanimation.com/why-real-time-animation-is-the-future-for-3d-animators/ [Accessed 13 Apr. 2023].

Epic Games. 2022. Animation Ecosystem for Game Development | Course. Available at: https://dev.epicgames.com/community/learning/courses/2Lz/unreal-engine-animation-ecosystem-for-game-development/5l7K/unreal-engine-retargeting-mannequin-assets [Accessed 2 Jan. 2023].

Eric V. Tuber. 2022. (NEW UE 5.1) How to replace your Metahuman as a ThirdPerson character in Unreal Engine 5.1. Available at: https://www.youtube.com/watch?v=2hpw0LY2QGE&ab_channel=EricV.Tuber [Accessed 24 Feb. 2023].

Evans Bohl. 2022. How to Animate Characters in UE5. Available at: https://www.youtube.com/watch?v=w9mijf-gKOg&ab_channel=EvansBohl [Accessed 3 Jan. 2023].

Gorka Champion. 2023. Unreal Engine 5 RPG Tutorial Series - #2: Locomotion - Blendspace, Crouching and Procedural Leaning! | Community tutorial. Available at: https://dev.epicgames.com/community/learning/tutorials/r49w/unreal-engine-5-rpg-tutorial-series-2-locomotion-blendspace-crouching-and-procedural-leaning [Accessed 24 Feb. 2023].

Gorka Champion. 2022. How to Make a Simple Blendspace in Unreal Engine 5 | Community tutorial. Available at: https://dev.epicgames.com/community/learning/tutorials/J6Lq/how-to-make-a-simple-blendspace-in-unreal-engine-5 [Accessed 23 Feb. 2023].

Jobutsu. 2022. [NEW] Metahuman as ThirdPerson - UnrealEngine 5.1. Available at: https://www.youtube.com/watch?v=vuhquaUE5HI&ab_channel=Jobutsu [Accessed 15 Feb. 2023].

Pixel Prof. 2021. Quick & Easy Mannequin to Metahuman Retargeting. Available at: https://www.youtube.com/watch?v=sS6j88-76VI&ab_channel=PixelProf [Accessed 23 Feb. 2023].

PrismaticaDev. 2022. Animation Montages | Adv. Anim Application [UE4/UE5]. Available at: https://www.youtube.com/watch?v=2dmqKf6w5J8&ab_channel=PrismaticaDev [Accessed 3 Jan. 2023].

Rokoko. 2022. Rokoko Guide: Updated Unreal Engine Workflow UE4.26 + UE4.27. Available at: https://www.youtube.com/watch?v=0ZSlEopk5zk&ab_channel=Rokoko [Accessed 22 Mar. 2023].

Smart Poly. 2022. Unreal Engine 5 Tutorial | Rig a T-Rex with Control Rig. Available at: https://www.youtube.com/watch?v=C_vrn1n1Uq8&ab_channel=SmartPoly [Accessed 22 Nov. 2022].

Totten, C. 2021. 12 principles for game animation. Game Developer. Available at: https://www.gamedeveloper.com/blogs/12-principles-for-game-animation. [Accessed 13 Apr. 2023].

UNF Games. 2022. Foot IK Tutorial in Unreal Engine 5 with Custom Character - Realistic Foot Placement. Available at: https://www.youtube.com/watch?v=IfU3p80EjQE&t=119s&ab_channel=UNFGames [Accessed 6 Apr. 2023].

Unreal Engine. 2022. Animating in Unreal Engine 5.1 | Unreal Fest 2022. Available at: https://www.youtube.com/watch?v=FgJ1stTScxI&t=2058s&ab_channel=UnrealEngine [Accessed 17 Feb. 2023].

Unreal Engine. 2023. Unreal Engine 5. Available at: https://www.unrealengine.com/en-US/unreal-engine-5. [Accessed 22 Nov. 2022].

Unreal Engine. 2022. Unreal Engine 5 Documentation. Available at: https://docs.unrealengine.com/5.1/en-US/ [Accessed 22 Nov. 2022].

Unreal RPG Mastery. 2022. Soulslike Combat Part 3 - Draw & Sheathe Weapon | Unreal Engine 5. Available at: https://www.youtube.com/watch?v=00e9VKMsBVM&list=PLrlmX34siH59eNHaR0CBoc0N-E__puH5t&index=4&ab_channel=UnrealRPGMastery [Accessed 3 Mar. 2023].

Your Sandbox. 2022. Replace Character with Metahuman in UE5. IK foot issue solved. Available at: https://www.youtube.com/watch?v=rxCWtcArhFU&ab_channel=YourSandbox [Accessed 23 Feb. 2023].