Ultimate Movement System

About The Asset

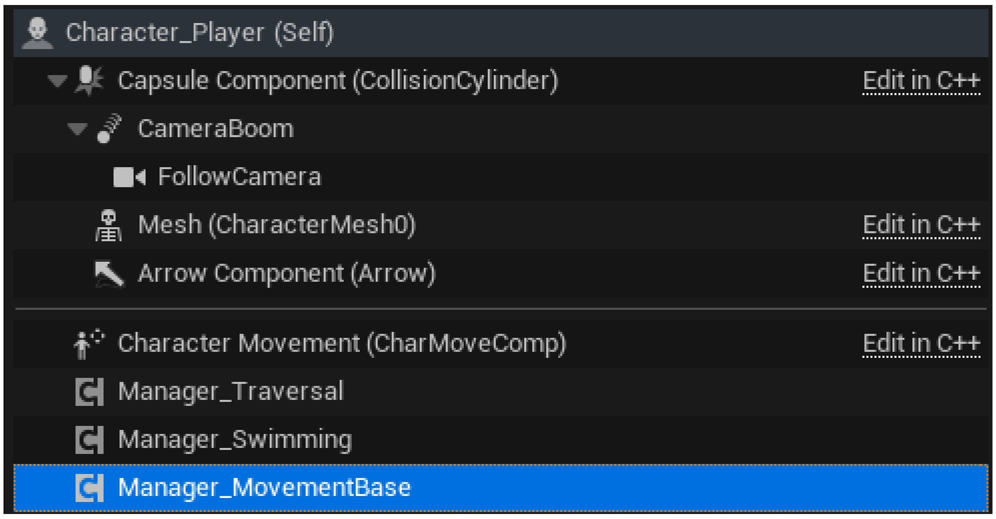
Ultimate Movement System is a replicated character movement constructor. You can create and modify your own movement types with a single component. Completely Game-driven and adapted animations with distance matching, orientation warping, and IK.

You can easily create new movement types, movement models, overlay states, stances, poses, and different gait profiles according to your game needs. All can have specific speeds, acceleration, deceleration, friction, and rotation rates (Check out the last image).

The goal of this asset is to have a good-looking movement system with limited resources. The example content uses only 2 animations for Forward and Backward locomotion and still, 8-way locomotion is possible with orientation warping. Different movement types will be added to further updates.

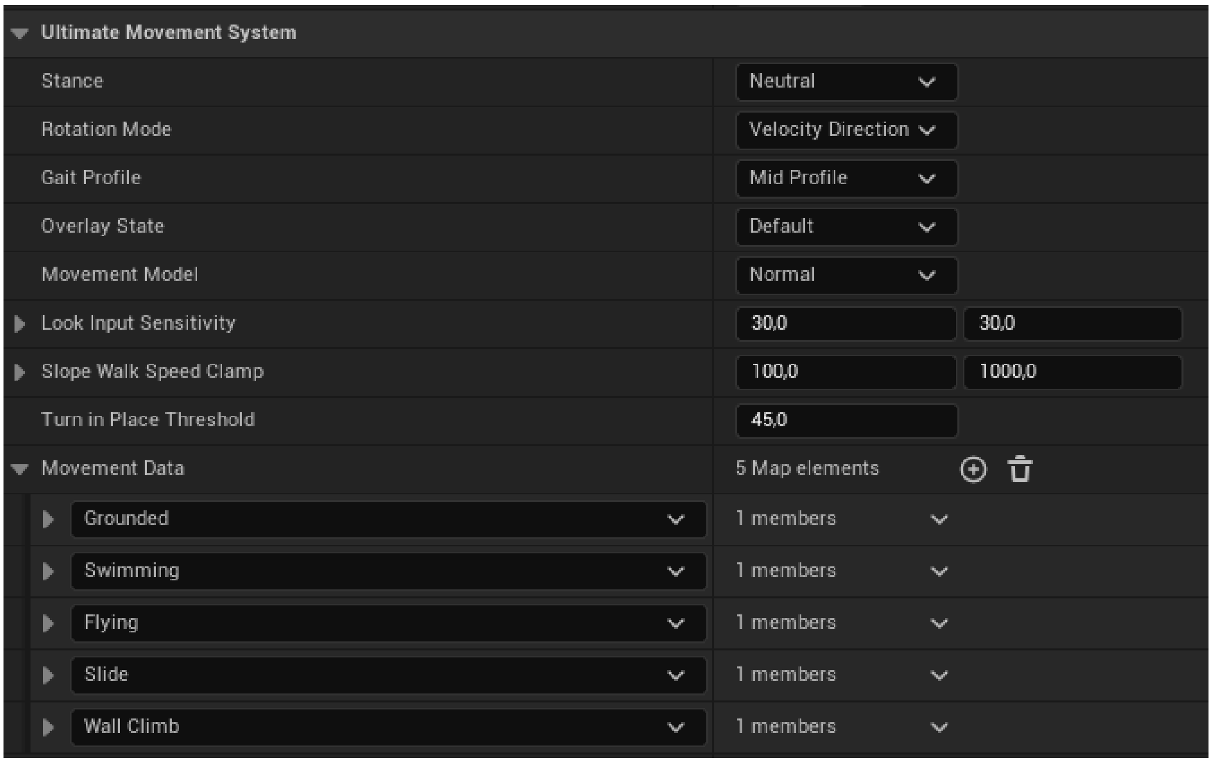
Customizing The Movement System Base

All of the accessible variables are exposed in the components that you have in the character. Overall it will handle the initialization process automatically. You can find the default values by clicking onto the Manager_MovementBase in the character blueprint.



The exposed variables on the details panel are self-explanatory, you can change the default stance, rotation mode, gait profile, overlay state and movement model by the enumerations.

You can also change the things like look input sensitivity, slope walk speed (basically max and min walk speed on slope) and turn in place threshold.

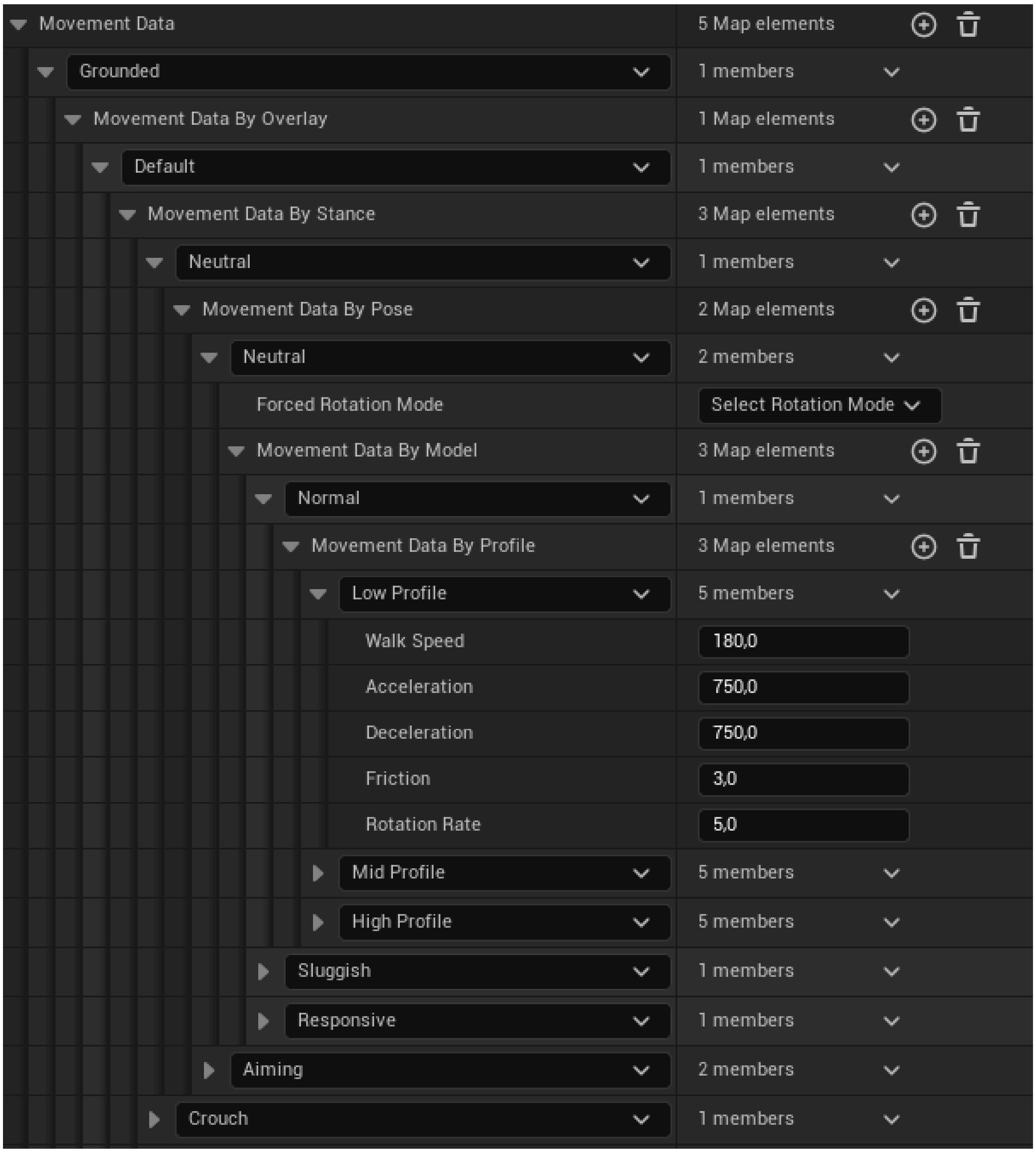


Ultimate Movement System	
Stance	Neutral
Rotation Mode	Velocity Direction
Gait Profile	Mid Profile
Overlay State	Default
Movement Model	Normal
Look Input Sensitivity	30,0 / 30,0
Slope Walk Speed Clamp	100,0 / 1000,0
Turn in Place Threshold	45,0
Movement Data	5 Map elements
Grounded	1 members
Swimming	1 members
Flying	1 members
Slide	1 members
Wall Climb	1 members

But the most important part is the **MovementData** mapping. In this mapping you are able to create different movement properties like walk speed, acceleration, deceleration, friction, and rotation rate based on the movement type, overlay, stance, pose, movement model, and gait profile.

Also, it’s important to have a movement data for your custom state. You should add it to the mapping in order to your custom movement logic work. You can see the example mapping in the image below.

Movement Data

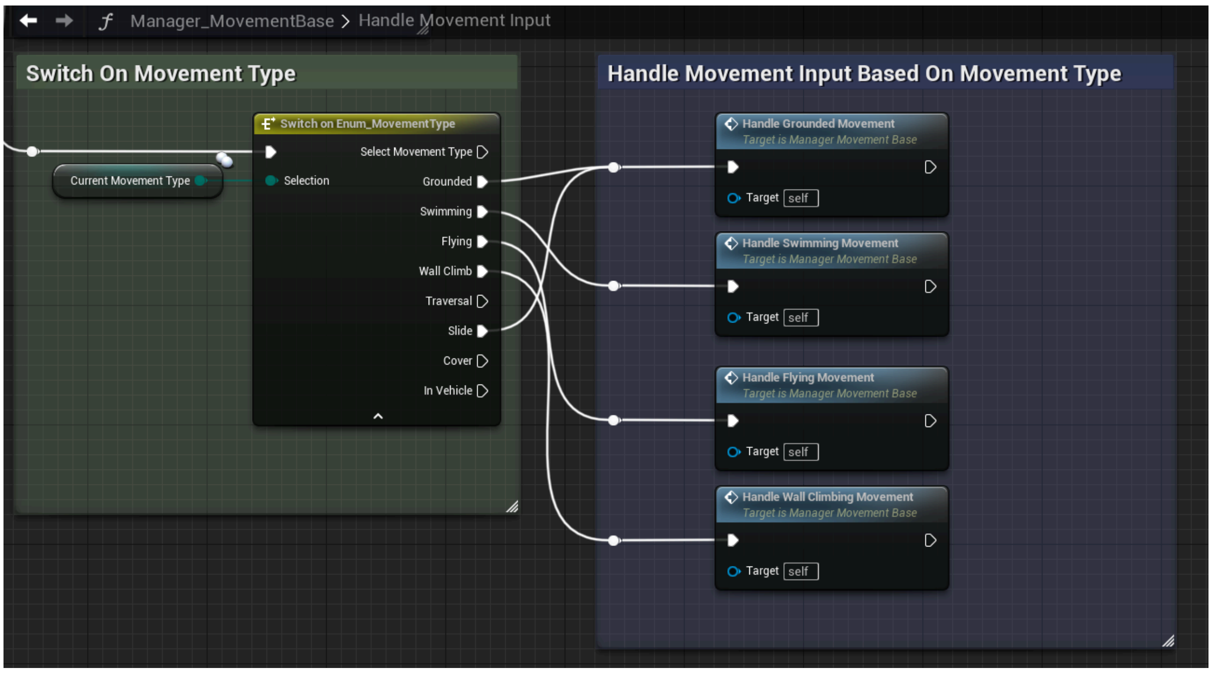


Movement Data	5 Map elements
Grounded	1 members
Movement Data By Overlay	1 Map elements
Default	1 members
Movement Data By Stance	3 Map elements
Neutral	1 members
Movement Data By Pose	2 Map elements
Neutral	2 members
Forced Rotation Mode	Select Rotation Mode
Movement Data By Model	3 Map elements
Normal	1 members
Movement Data By Profile	3 Map elements
Low Profile	5 members
Walk Speed	180,0
Acceleration	750,0
Deceleration	750,0
Friction	3,0
Rotation Rate	5,0
Mid Profile	5 members
High Profile	5 members
Sluggish	1 members
Responsive	1 members
Aiming	2 members
Crouch	1 members

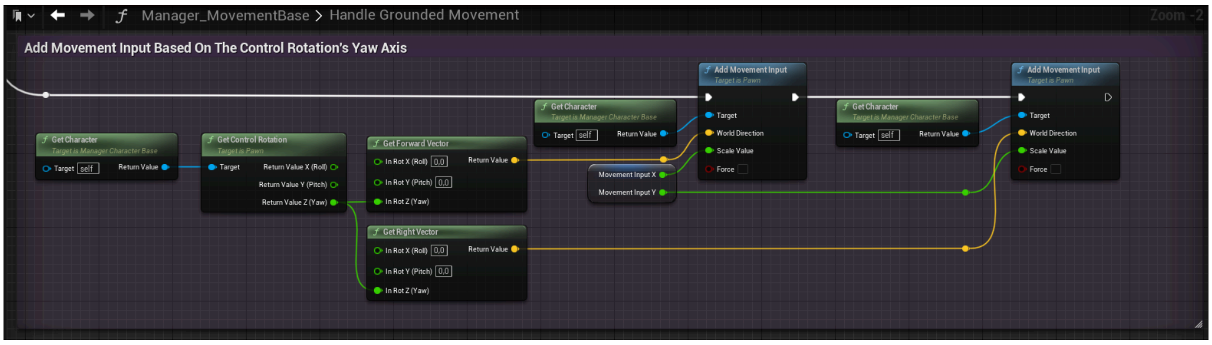
You don’t have to implement for every state you have, but you need at least one per state like default states for example.

Creating a Custom Movement Type

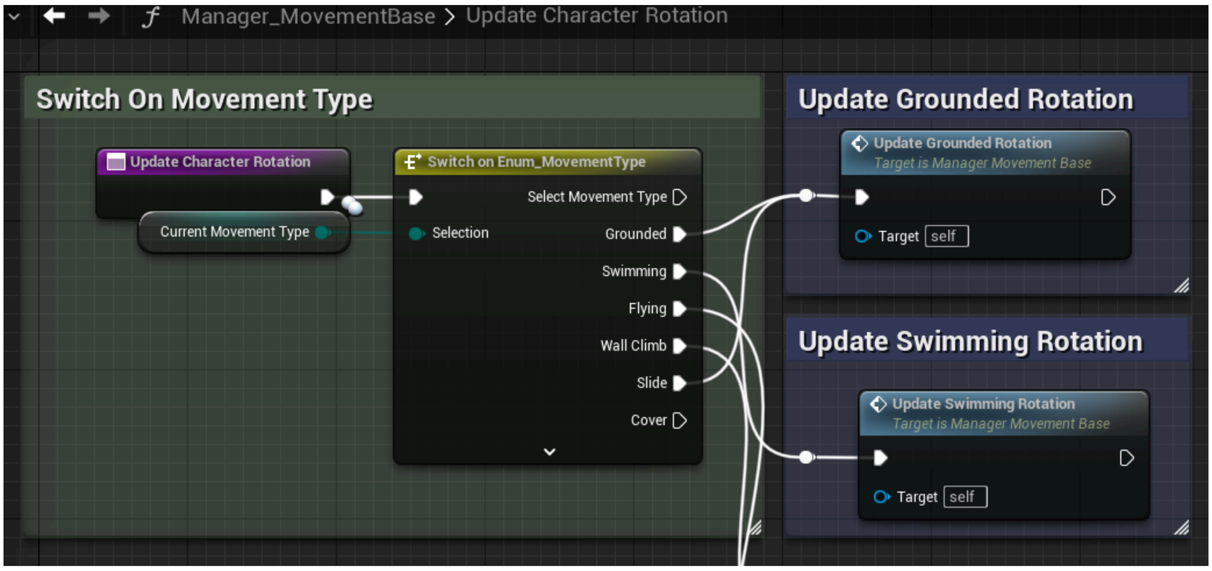
After you add your custom data to the related enumeration blueprint (Movement type, overlay, stance, etc.) you will be able to see them in the movement base component. First thing you need to do is setting up the movement input handler. You can duplicate one of the existing functions and pin to the custom output like the image below;


Manager_MovementBase > Handle Movement Input
Switch On Movement Type
Handle Movement Input Based On Movement Type
Switch on Enum_MovementType
Select Movement Type
Current Movement Type
Selection
Grounded
Swimming
Flying
Wall Climb
Traversal
Slide
Cover
In Vehicle
Handle Grounded Movement
Target is Manager Movement Base
Target self
Handle Swimming Movement
Target is Manager Movement Base
Target self
Handle Flying Movement
Target is Manager Movement Base
Target self
Handle Wall Climbing Movement
Target is Manager Movement Base
Target self

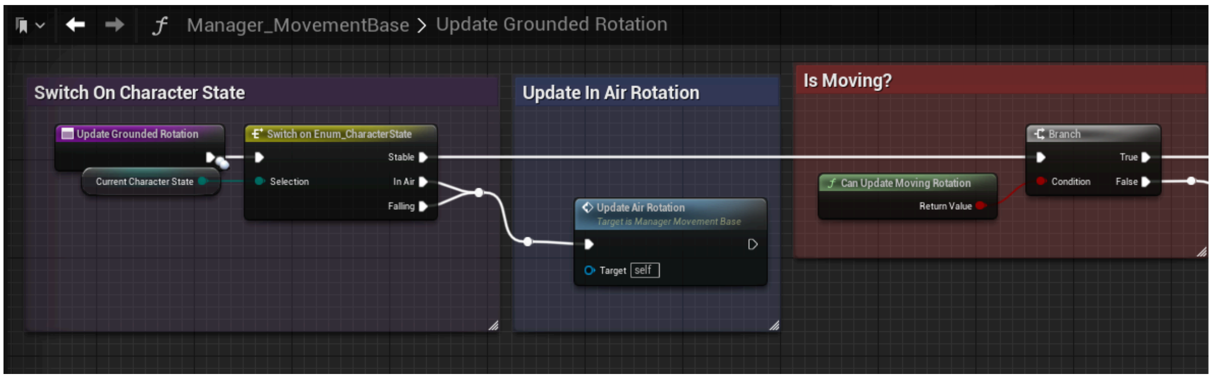
Take HandleGroundedMovement function for example, it's really basic and just updates the character velocity by adding movement input based on the control rotation.



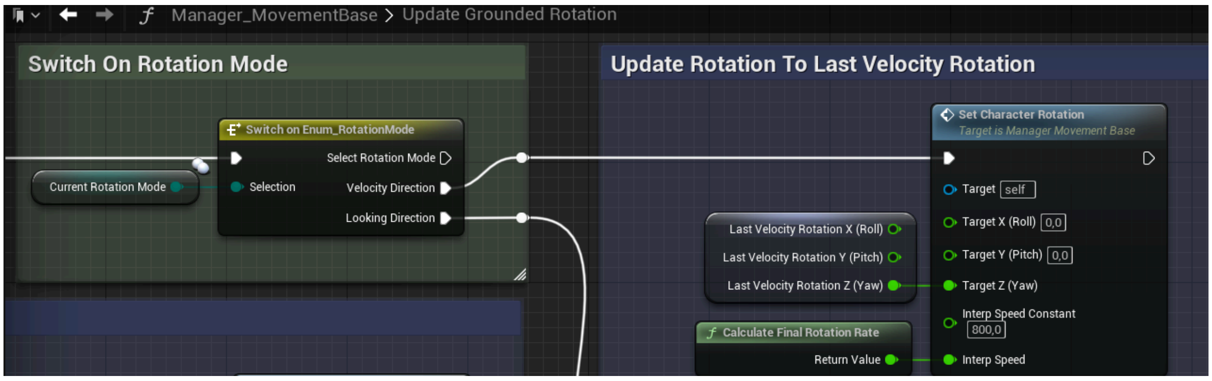
Same goes to rotation as well, You should create a new custom function called **Update(YourMovementMode)Rotation** and add it to the UpdateCharacterRotation function like the examples below;



Different from the movement input, the function you have for the rotation may or may not be built different. For example if you go to the example Update_GroundedRotation function you will see a bunch of enumeration switchers and different checks.

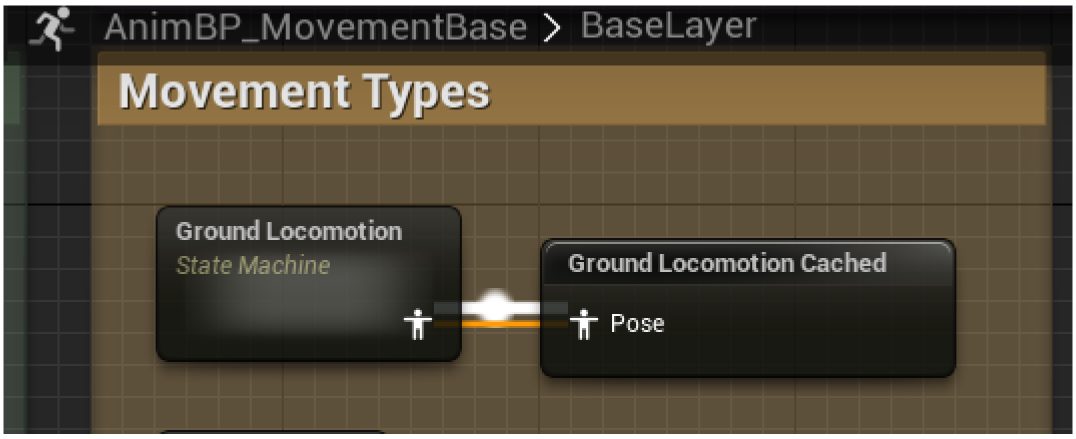


These are all movement type related and acts based on the given conditions. The important function you need to call after your logic implemented is the **SetCharacterRotation**

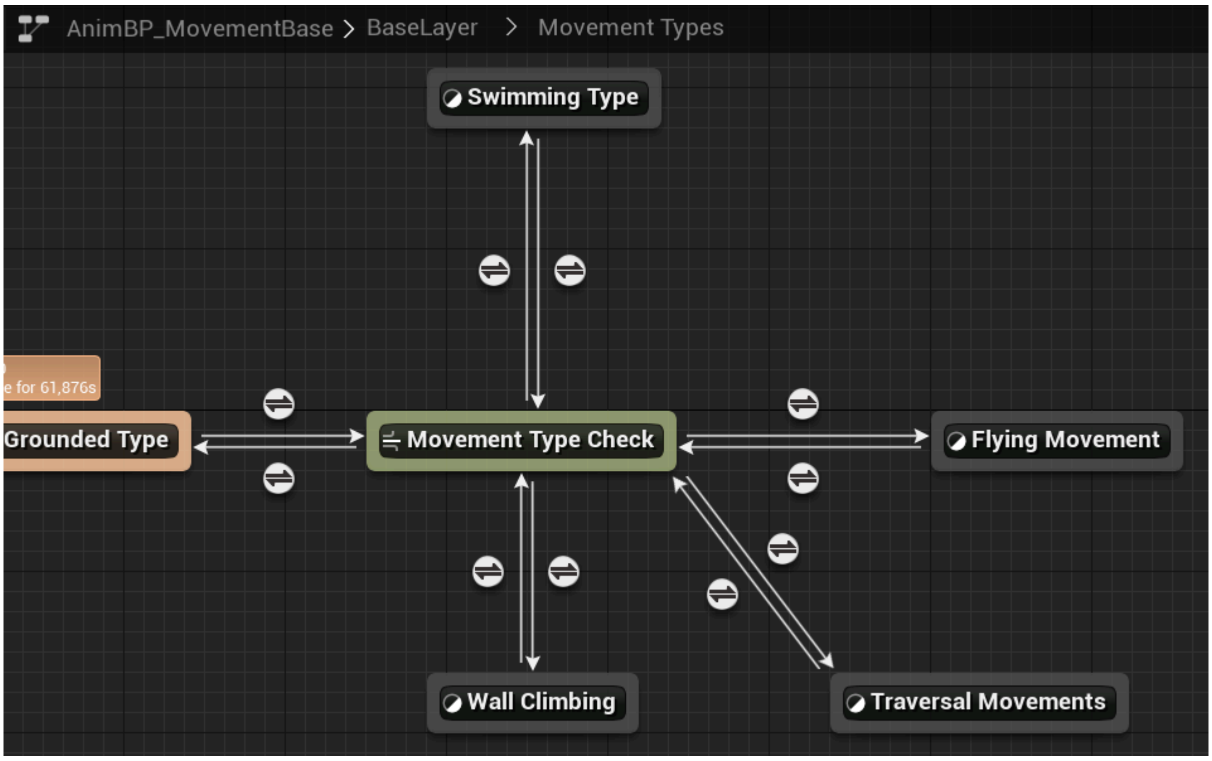

Manager_MovementBase > Update Grounded Rotation
Switch On Rotation Mode
Switch on Enum_RotationMode
Select Rotation Mode
Current Rotation Mode
Selection
Velocity Direction
Looking Direction
Update Rotation To Last Velocity Rotation
Set Character Rotation
Target is Manager Movement Base
Target self
Last Velocity Rotation X (Roll)
Last Velocity Rotation Y (Pitch)
Last Velocity Rotation Z (Yaw)
Target X (Roll) 0,0
Target Y (Pitch) 0,0
Target Z (Yaw)
Interp Speed Constant 800,0
Calculate Final Rotation Rate
Return Value
Interp Speed

It will automatically lerp the current character rotation to target value you have. Also important to connect the CalculateFinalRotation to InterpSpeed so it will take effect from the MovementData's RotationRate variable.

Then you will also add new state machines to the animation blueprint as well. As you can find examples in the **BaseLayer** you can create new state machines and cache them to use in the MovementTypes statate machine.



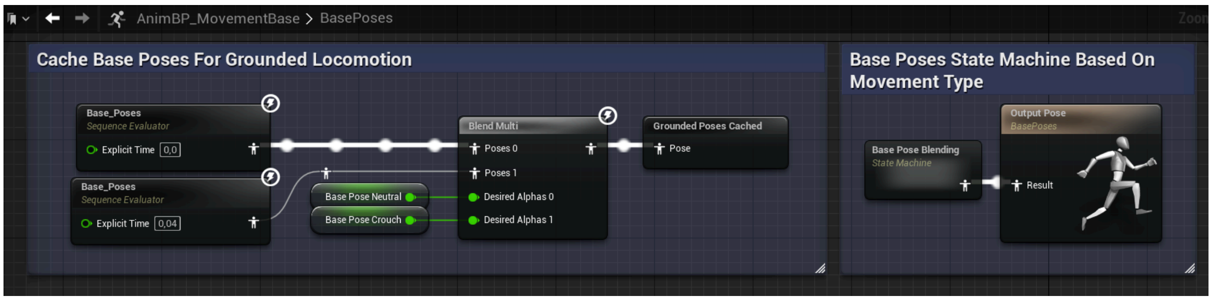
Then you can create a new state in the MovementTypes and add the conditions as the others. Basically checks the movement type enumeration value.



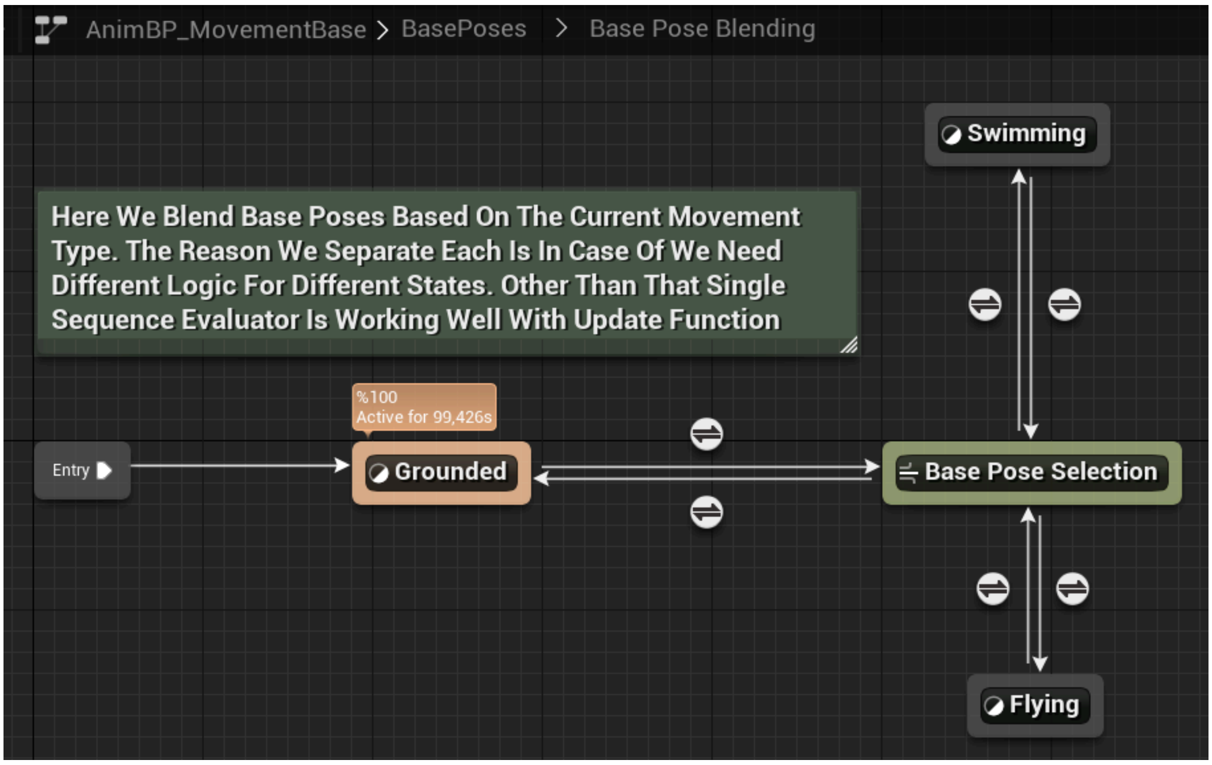
AnimBP Implementation Tips

Other applications such as pose or movement model is straight forward and they will just follow the movement data that you put into. But for a custom stance, you will have to modify the animation blueprint and add the new states based on your requirement.

The first thing you need to do is adding a state to BasePoses layer. If it's grounded movement you can add it to the GroundedPosesCached like in the examples so it will be using the cached value. Also important to create new float variables and update them as the examples to keep blending correct.



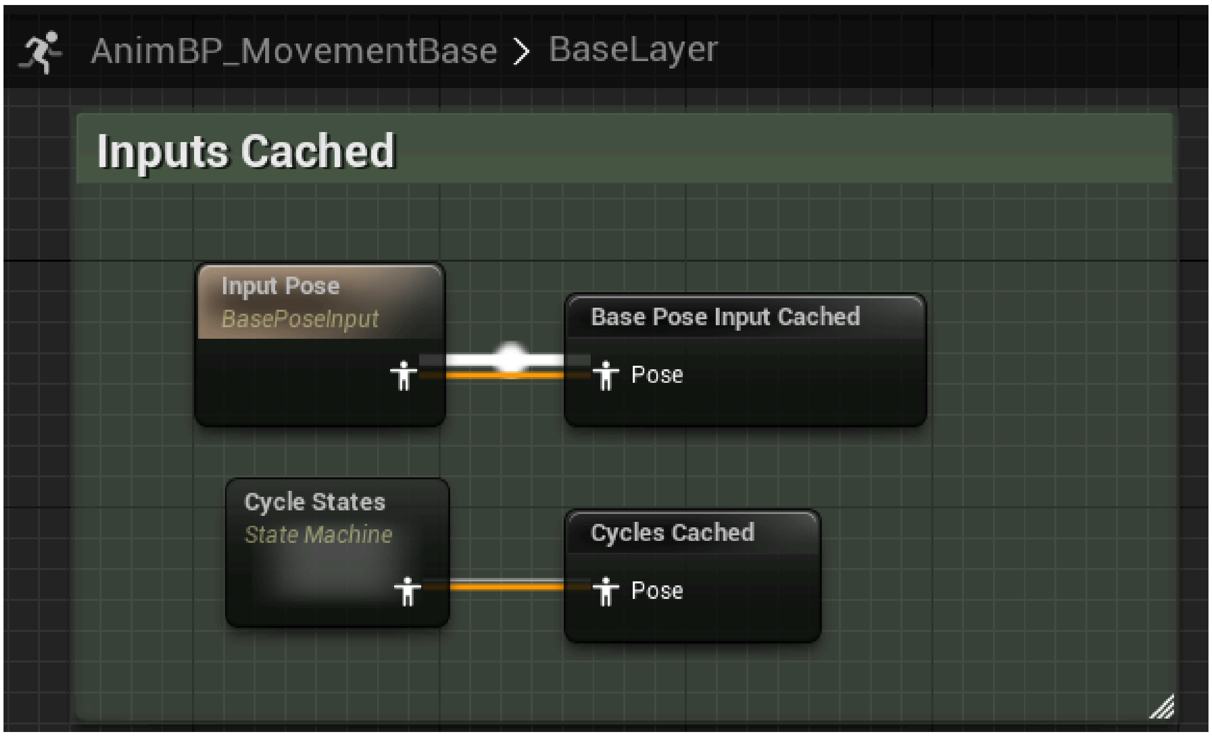
Or you can simply add a new logic to the Base Pose Blending state machine if you are not using the grounded movement type.



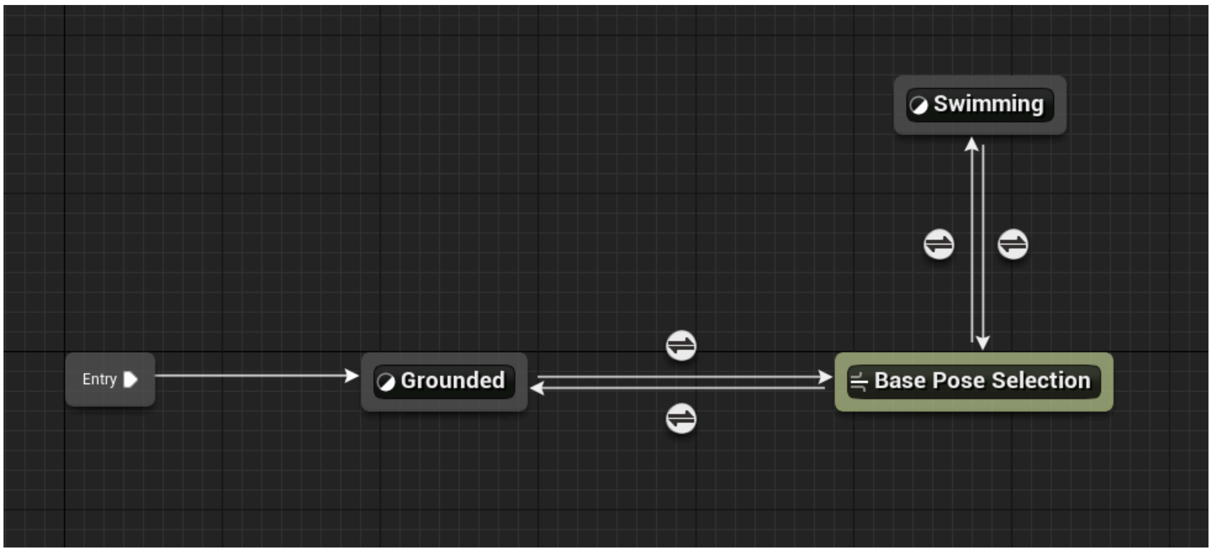
Here We Blend Base Poses Based On The Current Movement Type. The Reason We Separate Each Is In Case Of We Need Different Logic For Different States. Other Than That Single Sequence Evaluator Is Working Well With Update Function

Do not forget to use **Base Pose Input Cached** in the BaseLayer, the custom movement type state machine that you created as the **IDLE** state.

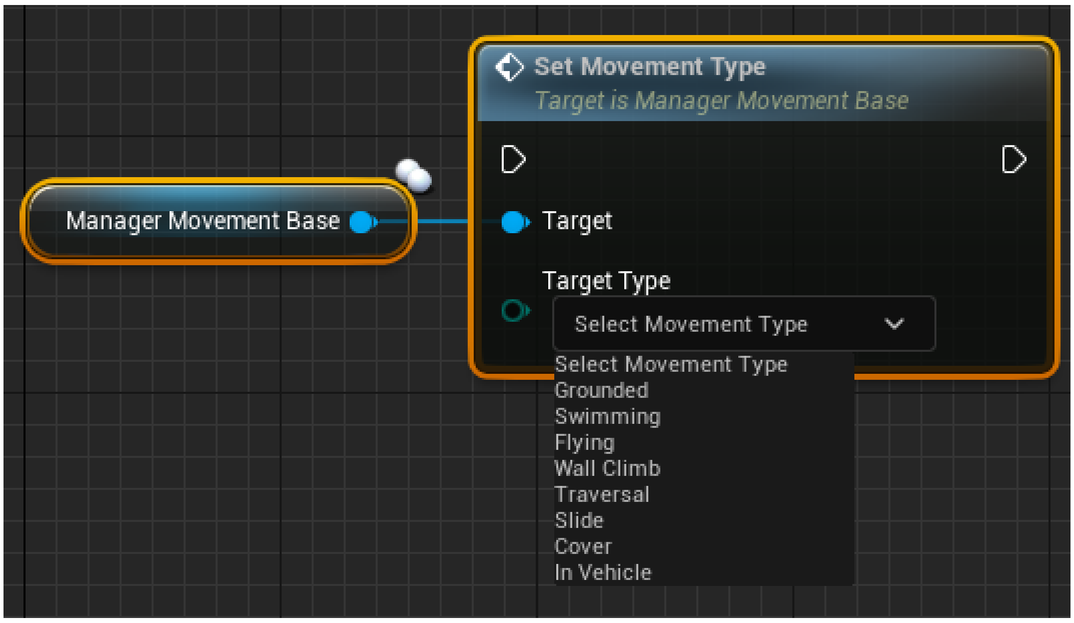
For cycles you can use the Cycles Cached and implement your custom cycle logic to the Cycle States state machine.



Basically you can create a new state in it and it will be used in your **CustomMovementType State Machine** that you created. So BasePoseInputCached is for idle state, Cycles Cached for moving state. You should keep that in mind.



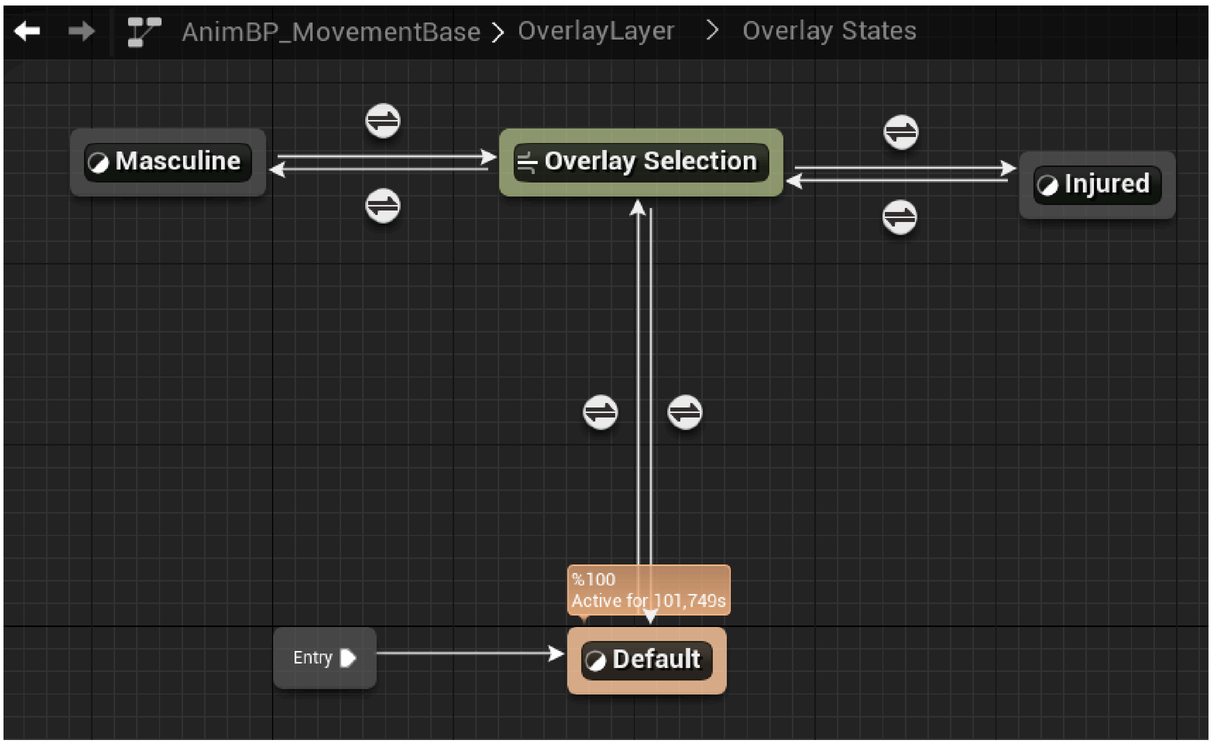
After all these done, and you add your new movement data to the component defaults, it should be all working as you set it up. All you need to do is calling the function to update the movement mode from anywhere you like. It will be using your custom Movement State Machine. It's all upto you how complex it's going to be. You can also check out the examples to get more information.
Here is how you can change the movement type;



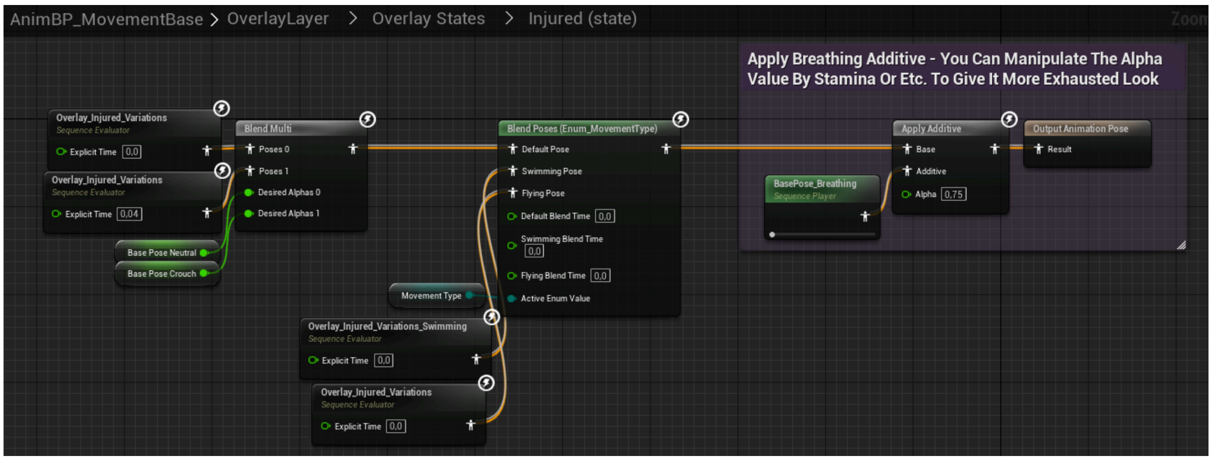
Creating A New Overlay State

Creating a new overlay state is a few steps as the movement type. Right after you add a new value to the Enum_OverlayStates, you should add a new MovementData overlay state to your movement type, could be also for the one which is Grounded Movement Type.

Then all you have to do is adding a new state to the OverlayLayer -> Overlay States as the examples. Conditions are basically checks the overlay state enum.

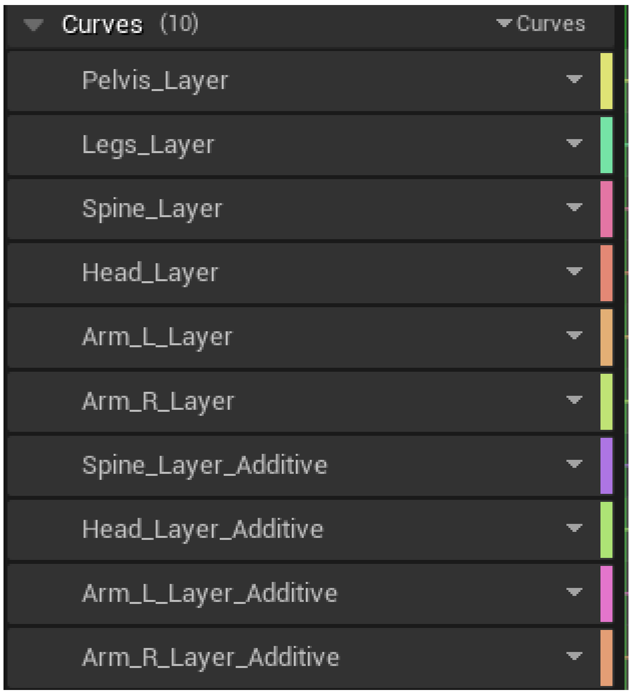


An overlay animation must be a single frame animation. You can add different scenarios to here like in the examples;



Apply Breathing Additive - You Can Manipulate The Alpha Value By Stamina Or Etc. To Give It More Exhausted Look

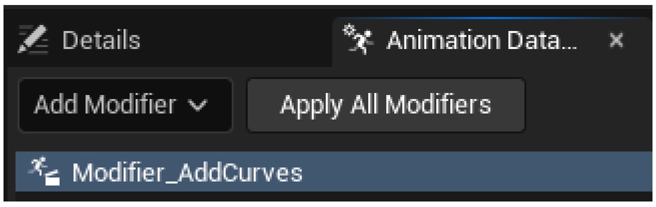
Then you should add Curves to your animations like in the image below;



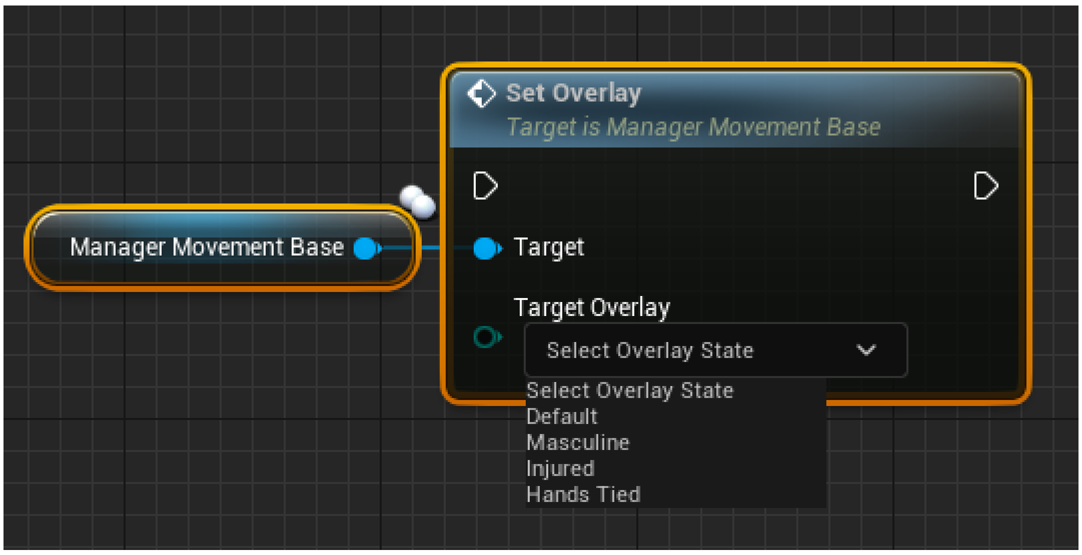
Curves (10)

Pelvis_Layer
Legs_Layer
Spine_Layer
Head_Layer
Arm_L_Layer
Arm_R_Layer
Spine_Layer_Additive
Head_Layer_Additive
Arm_L_Layer_Additive
Arm_R_Layer_Additive

Also good to mention there is a Animation Data Modifier that will add all of the curves for you. You still have to change the values tho.



Then you can change the overlay state whenever you want by calling the function below.


Set Overlay
Target is Manager Movement Base
Manager Movement Base
Target
Target Overlay
Select Overlay State
Select Overlay State
Default
Masculine
Injured
Hands Tied