Research results

To create a dark souls 1 inspired character controller, I will need to attempt to create it of similar quality. To achieve this, I will have to research some methods & Tools. These are:

- Root motion
- Animation notifies

Chosen solutions

Having done research into these topics, I will use / do these things for the dark souls controller:

Overall solution:

I want to use **root-motion** and the **specialized rig** where the control bones aren't on the same hierarchy as the rig for clean foot placement in animations. I am **using enumerators to choose data** inside of state notifies so that it can be **stored in a centralized space** within the weapon.

Using Enhanced input mapping

I experimented with enhanced input mapping to see if it was a tool worth using. I have found in my research that It has, due to the ability of automatically normalizing 2D input for movement. The modifiers on the inputs themselves have been found very useful as well.

I'm using enhanced input because it automates modifies like tap and holds. The default input method will also get deprecated in the next update to unreal 5, so my project would not be able to get updated to this patch without breaking.

Solution:

Use enhanced input to create the input events.

Pros:

- Can make use of the Tap, Hold & Combo modifiers to easily create events for these actions.
- Can make use of the input binding to a 2d vector and make use of its radial normalization to create a clear direction for movement.

Cons:

- Not familiar and a small learning curve.

Use Root motion doesn't name the armature

I setup root motion in unreal and It took me quite a bit of effort to fully implement. But in the process of this implementation, I've learned fully how to set this up within unreal. However, I will only use it for actions. Using root motion for full movement would lead to a designer un-friendly workflow and would make my feature less desirable for other designers.

I'm using root-motion, and because of this, I have no choice but to apply this solution. It's either this or use maya.

Solution:

Use root motion animation, But don't name the armature in the rig and have it be scaled in blender with FBX unit scaling to make sure that it's allowed to have an animated root.

Pros:
- Allows for root motion.
- Root-motion allows for animation with movement which increases the quality of animation by a lot

Cons:
- Using root motion can be designer un-friendly
- Unnamed armature is slightly inconvenient.

Accurate feet placement by removing control bones from hierarchy.

This experiment resulted me removing all control bones from the exported skeleton. This made for a clean skeleton. Because of this I can make animations with feet that line up with the ground accurately because the control bones don't have to be parented to the root bone.

I cannot think of a reason to not use this in my current project. The only time where I wouldn't want to have this, is if I didn't make use of root-motion and then it would be a minor upgrade.

Solution:
Don't put the feet control bones on the same bone hierarchy as the root and this allows for accurate foot ground placement. This was possible by not exporting the control bones, by selecting "only deform bones" to export.

Pros:
- Large increase in animation quality due to feet accurately placed on ground.
- Easy to animate foot position due to them not following root bone.
- Exported skeleton gets cleaned up because the control bones don't get transferred to unreal.

Cons:
[There is no con, this is a direct improvement to my workflow and rig]

Modular animation notifies with a centralized data point

I figured out that it's possible to create variables (and not set them) in animation notifies. They are able to be instance editable variables, which means that a single animation is actually able to hold data on it's own.

I have chosen the solution I have to make sure that the important damage and stamina variables are able to be rapidly iterated upon more effectively than if they were stored in a centralized place. Also, if they weren't, it would be much harder to make them weapon specific.

Chosen solution:
Enumerators are created that allow the user to "select" what data to use for what animation. This allows for storing data in a centralized place (Weapon).

Pros:

- Animation now instead of holding individual data "select" data, which makes it easy to balance and adjust.

Cons:

- It will be harder for animations to hold unique data properties.

~~Solution:~~
Make the user set the individual data of every animation inside of the animation itself.

Pros:

- Animations hold individual data which allows for a large amount of modularity.

Cons:

- Data and variables aren't stored in a centralized place, which would be very tedious for iteration.

Optimizing creation of notify tracks

Tim helped me find a solution that allowed me to copy over notify tracks through just copying the notifies, but this also means that I won't be able to copy over the names.

Having names for each track allows for better organization and an easier time explaining it to others. However, it is much slower to create tracks and it would effectively slow-down my workflow, So I would rather not have names.

Solution:
I will use the method of copying over animation notifies to transfer the tracks to.

Pros:

- Creation of notify tracks will be much faster

Cons:

- Not able to name and sort them.

Experiments summary & Take-aways

Experiment 1: Enhanced input mapping

For my experimentation I've messed around and attempted setting up inputs with different modifiers to see what I could create with it.

1. *Make use of the Tap, Hold & Combo modifiers to easily create events for these actions.*
2. *Make use of the input binding to a 2d vector and make use of its radial normalization to create a clear direction for movement.*

Experiment 2: Root motion handling

For this experiment I attempted to export a root motion animation and view and use it in-engine. Doing this I ran into issues, I've fixed it and it lead to these take-aways:

1. *The armature, when named, in blender is actually secretly the root bone and the owner of all the scales.*
2. *Blender to unreal plugin isn't available for my current version of blender*
3. *Root motion with blender is possible through the scaling up of the root bone in unreal.*

Experiment 3: Feet placement for root motion animation

This experiment I set out to remove the foot control bones from the root hierarchy and succeeded. I did this through a feature that makes me able to only select bones to export that deform the mesh. This has also cleaned up exported rig a bit.

1. *Exporting deform bones only, allows control bones to not be part of the root hierarchy.*

Experiment 4: Modular animation notifies

This small experiment forced me to think about how I will store my data relating to the animations. This was because I figured out that you can use instance editable variables on animation notifies. To me this meant I could store data in animations themselves, but this caused a problem. The data wasn't centralized and thus hard to adjust. I mitigated this problem by making the instance editable variable be an enumerator that "selects" out of a pool of data. This data is now stored in the weapon, in a centralized place where the user can easily adjust it.

2. *Instance editable notifies open up possibilities by having animations hold data instead of separate blueprint / actor.*
3. *Instance editable data won't be in a centralized place so I have to beware with adding too much.*

Experiment 5: Optimizing creation of notify tracks

I wanted to make creating notify tracks more efficient and hopefully be able to organize them and set them in animations. But this wasn't possible, and the solution is just copying notifies to create a number of tracks desired.

1. *Copy over animation notifies to copy over the tracks.*
2. *This doesn't copy over the names.*

Experiment 1: Enhanced input mapping

Enhanced input is Unreal 5.1's new way of handling input. It's a powerful tool that automates creating custom forms of inputs, like tapping, combos and more.

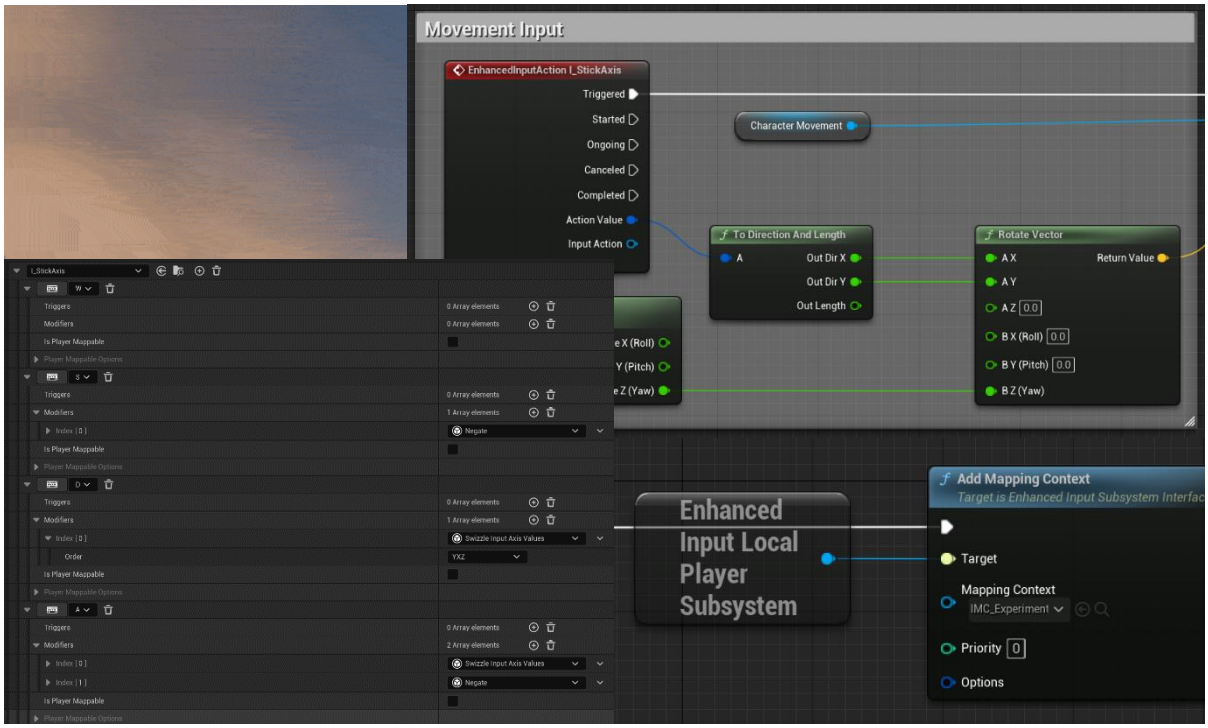
Experiment goal

I want to have a 2d vector as an output for the movement input so that I can then immediately use it for direction. I also want to see if I can create input presets for combos, tapping and holding.

Resources

Documentation – Link

Results



It looked quite intimidating at the start but was manageable and understandable when messing around with it. I created a simple "combo" tester with tap inputs and a string for when the player is holding with time held shown when held.

Take-aways

Enhanced input allows for easy creations of verbs. It is a little finicky to set up but does allow for a huge amount of customizability.

- Make use of the Tap, Hold & Combo modifiers to easily create events for these actions.
- Make use of the input binding to a 2d vector and make use of its radial normalization to create a clear direction for movement.

Experiment 2: Root motion handling in Unreal

Experiment goal

Dark souls uses root motion for its animation and movement & actions. I want to know how to export a proper root motion animation and have it perform movement in unreal. I will create a simple test animation that has root motion to show off that I can use it in unreal, hopefully not running into problems.

Resources

Root motion documentation – Link
Tutorial of preventing root bone addition – Link
Blender to Unreal live training - Link

Results

Figure 1: Root motion doesn't work

Figure 2: Attempted uniform scaling x100 and it broke the root motion.

Figure 3: Fixed the root motion by scaling up the root bone.

Notes during testing

At the beginning I imported an animation and turned on the root motion. This didn't work. After messing around with the skeleton with Tim, we managed to figure out that the root bone I had in blender wasn't the same as the root in unreal. This Is an issue with blender surrounding the naming of the armature.

If the armature has a custom name, it will take that and create a separate root bone, making me unable to use root motion animation.

if the armature is named armature, it will be deleted in unreal. allowing me to create root motion animation. but this will then make the character super tiny, since blender holds all scale data in the new armature bone, which will then be deleted.

I can scale up the animation uniform scale, since multiplying by 100, will make it the exact same as its original size. but this will also scale up the distance of the root motion movement, making me dash extremely fast.

I tried to find a plugin for blender, but the one created by Epic is deprecated. I then found a found another plugin, but it wasn't supported for blender 3.3. Link

~~After reading a thread and getting nowhere I attempted to see if I could scale up the bones except for the root bone. This wasn't possible, but I realized that scaling up the bones didn't make them cross larger distances, so I just scaled up the root bone and this fixed all problems!~~

The previous solution somehow didn't work when repeated, so I messed around in blender exporting just to see that I could turn on FBX unit scaling as an export setting. This I can easily measure and use as an export for the size, and it fixes everything!

Take-aways

- The armature, when named, in blender is actually secretly the root bone and the owner of all the scales.
- Blender to unreal plugin isn't available for my current version of blender
- Apply scaling in the export settings and use FBX units scale and then set it to scale 20 to have same size in unreal.

Experiment 3: Feet placement for root motion animation

Experiment goal
The animation workflow is very flawed right now due to the feet being parented to the root bones, which moves for root motion animation. I need to somehow be able to not have the feet move when the root moves.

Resources
Thread - Link

Results

Notes during testing
Feet don't line up due to moving of feet with root bone.
attempted to make child of separate from root bone.

Removed foot control bones from root hierarchy, this didn't allow me to import in unreal.

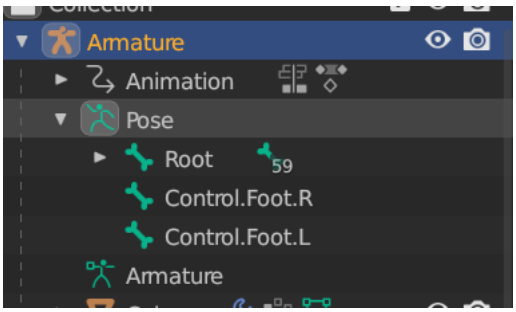
~~After doing further research, I realized that the feet need to be bound on the master hip bone instead of the root bone. in this way, when it is done animating, I can just move the root and the feet will line up.~~

This didn't work.

After trying literally every modifier in the world, I resorted to trying to remove foot control from the hierarchy and somehow **not have them export**. which worked by not setting them as deform. this made them not included during the export, allowing me to animate the feet on the floor!

This allowed me to import the mesh, because the bones that weren't in the hierarchy weren't exported.

Take-aways
- Exporting deform bones only, allows control bones to not be part of the root hierarchy.



Experiment 4: Modular animation notifies

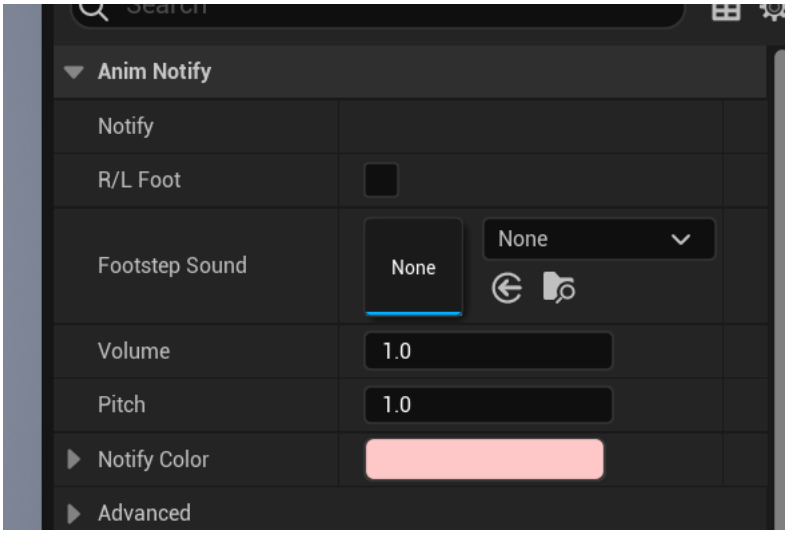
Experiment goal

The dark souls frame data has what looks like a lot of modular animation notifies, because it uses the same notifies in almost all animations. I will do some research to see how modular you can make an animation notify.

Resources

Unreal notify documentation – Link

Results - Example



Notes during testing

After messing around the documentation, there was nothing I could find that helped me further for documentation.

But coincidentally, while already working on a master notify state blueprint, I realized I could add but not set variables (which is interesting) that were owned by the notify state. After which I realized I could just set these variables to be instance editable.

This is great since this means that I can just make notify state which holds variables like enumerators or damage values etc…

I've made a little footstep notify with instance editable variables as an example. \

However, experimenting with this has led me to believe this isn't a smart way to store data since the adjusting of it will have to be done within the animation notify specifically.

What I think is best, is to create enumerators which "select" and put the animation / attack into a type. Which will allow me to create data in a centralized place which makes use of this type.

Take-aways

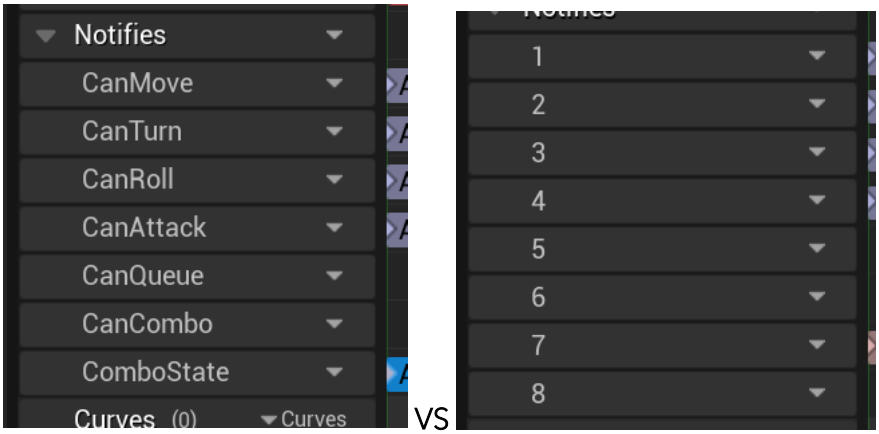
- Instance editable notifies open up possibilities by having animations hold data instead of separate blueprint / actor.
- Instance editable data won't be in a centralized place so I have to beware with adding too much.

Experiment 5: Optimizing creation of notify tracks

Experiment goal
Creating notify tracks and naming them specifically is very time consuming, so I will do some research into trying to see if I can automate / make this process more efficient.

Resources
Unreal notify documentation – Link

Results



Notes during testing
The documentation doesn't touch this topic at all, and there are no reddit post / online threads on it. So weirdly no one seems to be having this issue.

I did learn however that you can rename a notify track much quicker by pressing exactly 3 times on it, instead of naming only on creation of a track.

I sent Tim a message and he had shown me that you can create similar notify tracks if you just copy over the notifies needed for the animation. However, it doesn't copy over the names.

So I had to make a decision between organization or speed.

I chose speed since for me, the order and efficiency matter more than actual naming. However, this does make it less clear where an animation is supposed to be placed.

Tm also told me, I could try and solve this through creating an editor utility widget, but I don't think this problem is big enough and nor do I have the time.

Takeaways
- Copy over animation notifies to copy over the tracks.
- This doesn't copy over the names.