CS354p Lab 5: Working with Animations

This lab will focus on extending the a playable character you created in Lab 4 by connecting animations to the State Machine you created. Thus, this lab will have some C++ development, but we will largely be working in Blueprints and with Assets, so this will be a good time to familiarize yourself with those systems, if you are not confident yet in your ability to use the UE5 Editor/work directly with assets.

Getting Started (Complete before the Lab)

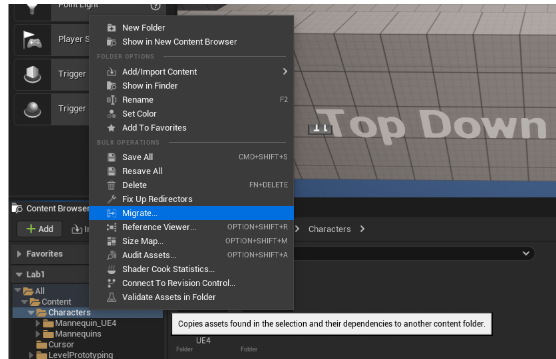
You can choose whether to attempt migrating your Lab4 to this new project, Lab5, or just copy what you had in Lab4 to a separate folder for Lab5. I personally just copied it, because the project is small and I didn't want to bother with migration. That said, if you feel confident in your understanding of UE4 build systems, it's possible to migrate assets, update the C++ module, and fix the derivations in Blueprint. The following resources will give you some context:
`https://isaratech.com/ue4-how-to-rename-an-unreal-engine-project-with-sources-files/`
`https://docs.unrealengine.com/4.26/en-US/Resources/SampleGames/ARPG/MigratingContent/`
`https://unrealxeditor.wordpress.com/2015/05/28/tip-renaming-c-classes-without-breaking-your`

Adding a Camera to Character

Now you will add a camera setup to `Lab5Character`. This means including a Camera Component and a Spring Arm Component that acts as a boom to better control it and handle physical interactions with the camera if necessary. This will be very similar to what we did in Lab3.

You may need to rebuild intermediate data, but once that's working and you see the changes updated in the Blueprint, congratulations! This is basically all we're doing with C++ in this lab! We do need to add a proper mesh in our `Lab5CharacterBP` though.

Since we aren't making our own animation assets from scratch, we'll migrate the Character assets from Lab1 to our project. This will also give you some ideas of how to build out your animation state machine since it includes the premade ones. To perform a migration, open up Lab1 and right click on Content -> Characters. Select "Migrate." You can choose specific assets or just grab them all. We are going to be working with Manny, but the same principles will apply to Quinn if you want both. Select your Lab5's Content folder for the target directory then wait for the files to copy over.



Once this is done, you can open `Lab5CharacterBP`, select the Mesh Component and under the Mesh group in Details, select "SKM_Manny" as the Skeletal Mesh. This attaches a mesh and working rig to your Character. Align the mannequin mesh that appears within the Viewport with the Capsule Component so that it will handle collisions in a plausible way. Confirm this is working in play-in-editor (PIE).

Creating an Animation Blueprint

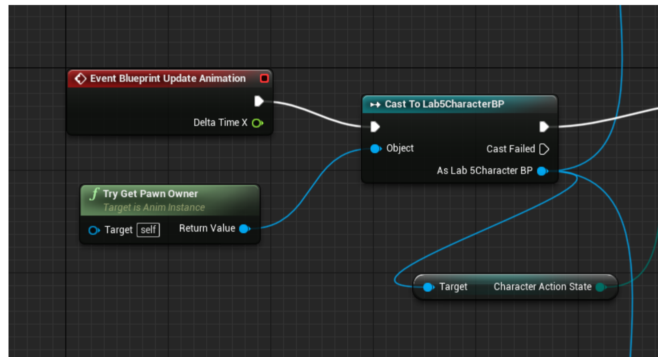
We now need to add animations. You may notice they have a very nice animation blueprint included as ABP_Manny and ABP_Quinn. We are going to create a somewhat simplified version of these using our Character State Enums created in Lab4.

To do this, access the Content folder and right-click to create a new asset. We want a `Animation->AnimationBlueprint`. Select SK_Mannequin as the skeleton and name it "Lab5CharacterAnimBP." Then go back into `Lab5CharacterBP`. Select the Mesh, and within Details, go to the Animation group. Set Animation Mode to "Use Animation Blueprint" and select "Lab5CharacterAnimBP" from the drop down menu to select the Animation Blueprint you just created. You are pretty much done working in `Lab5CharacterBP` unless you need to improve your event logic (very likely) to handle animations more gracefully, but don't worry about that until you're debugging the animations.

To create animations, we'll need to build out two types of functionality in the Animation Blueprint: Event Graph functionality to receive updates from `Lab5CharacterBP`, and Anim Graph where we build out the actual State Machine and transition functionality. You can work on them in any order (or in conjunction), and you are welcome to look up additional tutorials/examples to help in this process. Just be aware that we're using `Lab5Character`'s state setup we created within our enumeration, so you'll be using that information as your primary source of state information.

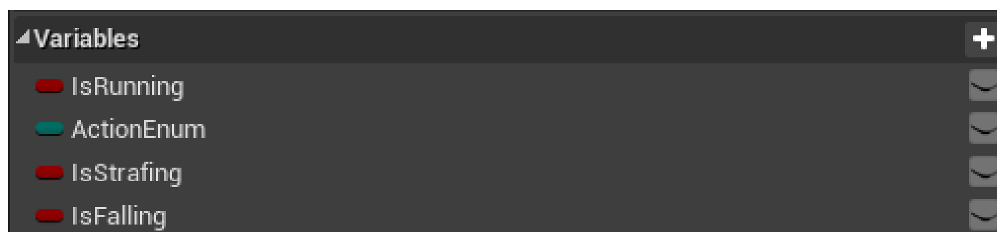
Animation Blueprint Event Graph

To get started, you will first need to access data in `Lab5CharacterBP` from the Animation Blueprint's `Update Animation Event`, which is in the Event Graph tab of the Animation Blueprint. It will look something like this:



Notice that I'm accessing the Pawn's Owner, then casting it to `Lab5CharacterBP`. If it succeeds, I go ahead and grab `Character Action State`, which is the current state of `Lab5CharacterBP` (this event is called on Tick). I can now switch on `Character Action State` to update the logic in my Animation FSM.

To do this, I need to create several variables within this Animation Blueprint, so that I can determine transitions (as I can't access the original Blueprint variables from within Anim Graph). The variables I created for this are listed here:

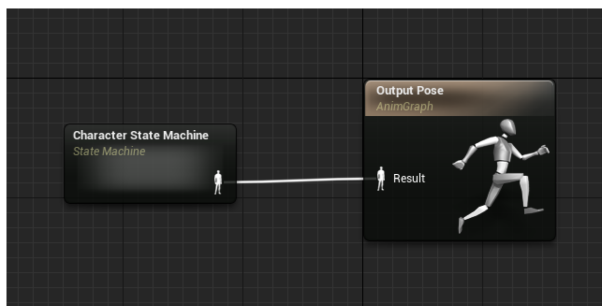


This is by no means the only way to approach building your state machine, but it demonstrates one of many ways to distinguish walking versus running, forward motion versus strafing, when the character is falling, and after the landing animation during Jump state has finished. ActionEnum is of type `ECharacterActionStateEnum`, and I am just setting ActionEnum's values (in `Lab5CharacterAnimBP`) to match Character Action State (in `Lab5CharacterBP`), so I can use that information in `Lab5CharacterAnimBP`.

You'll also want to ensure your booleans are updated correctly in the Event Graph, but you can also come back to that after starting on the FSM itself in Anim Graph if you're not sure about the logic.

Animation Blueprint Anim Graph

Within the Anim Graph, you will build out a hierarchical FSM. You can start by right clicking on the Anim Graph and typing "state machine." The highest-level of this will look like this:



It can get much more complicated if you want better blending, but we'll just go with this for now. Inside this State Machine, the Entry point should lead to the Idle state, which you will create by right-clicking and creating a State called Idle. Double-click on Idle to set the outputted animation pose. We're not going to do anything fancy — just create "Play MM_Idle" or "Play MF_Idle." Once this is in place, you should see your BP Character mesh idling.

Other high-level states to create are Jump, Movement, and Interact. If you are unsure how to create states and transitions, please ask or feel free to look up a tutorial, but the basics are: right-click to create a new state, left-drag from one state to another to create a transition. Take some time to play around with this and determine the transitions that will allow Jump from both Idle, and Movement, but not Interact, and Interact from Idle only.

If you click on the transitions, you can set the condition that will allow this transition. Within transitions, you can access Action Enum and your boolean values to set up the necessary logic. Within the State itself, you can either set the animation pose from the animation files already associated with your Mannequin (e.g. `PlayNameOfAnimation`), or you can create another sub-State Machine. We will be creating a sub-State Machine for both Jump and Movement. The Movement state machine should handle the differences in Run, Walk, and Strafe. Run should playing the running animation if the Character speed is above some value of your choosing, and both Walk and Strafe will play the walk animation depending on Character speed and if the Character is more strafing than moving forward respectively. You do not need to worry about exiting from the sub-State Machine – that will be handled with the transitions between Movement and the other states.

For the Jump state machine, you will chain together two animations: Jump_Start and Jump_Loop. Jump_Start will play upon entering the Jump sub-State Machine then transition as it completes into Jump_Loop. Jump_Loop will play as long as the Character is in the "Falling" state (something you can grab from the Character's MovementComponent in the Event Graph). When the Character lands, you will return to Idle or Move.

As always, please feel free to ask if you are confused about any of the above information. Working with Blueprints is a separate but highly technical skill, so it may take some experimentation. This will hopefully give you a better understanding of FSMs, as well as an appreciation for artists and designers who are able to make cool systems that look good!

Submission

After you're satisfied, collect some video footage showing the system in action. Also screenshot exciting parts of your code, and submit these files plus the project code via your GitLab account and include a link to your video via Youtube. Link your repositiory as your Canvas submission.