CS378 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 UE4 Editor/work directly with assets.

Getting Started (Complete before the Lab)

You can choose whether to attempt migrating your Lab 4 to this new project Lab 5 or just reconstruct what you had in Lab 4 for Lab 5. Both approaches are hassles, so emotionally prepare yourself if you want to get a taste of how migration works. I personally just recreated it, because the project is small enough, 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-project/

“Can I just copy the project to a directory called ‘Lab 5’ and leave the classes named ‘Lab 4’ you ask?” I’d really prefer you do it one of the above ways as you’ll get more practice and therefore have a better understanding of what’s going on, but yes, you can do this. Just know that I will know.

Regardless of how we all got here, please download these files: https://www.cs.utexas.edu/~theshark/courses/cs378/assignments/Mannequin.zip and drop the unzipped folder into your project’s Content folder. You should see the files appear in the Content Browser if you did it correctly.

Adding a Camera to Character

Once you’ve successfully updated your project, 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. I would highly recommend setting your UPROPERTY for both to EditAnywhere and BlueprintReadWrite to make the parameters more accessible and allow for camera tuning within 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.

Within Lab5CharacterBP, select the Mesh Component and under the Mesh group in Details, select “SK_Mannequin” 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 to make it look better. In the Content folder next to your Lab5CharacterBP asset, right-click to create a new asset. We want a Animation->AnimationBlueprint. Name it “Lab5CharacterAnimBP” and 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_C” 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 to help in this process (or reference the “ThirdPerson_AnimBP” that came with the mannequin files. 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. It will look something like this:

![Event Graph](image)

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:

![Variables](image)

This is by no means the only way to approach building your state machine, but I found it convenient 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. The highest-level of this will look like this:

![Character State Machine](image)

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. Other high-level states 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 three animations: Jump_Start, Jump_Loop, and Jump_End. 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 play the Jump_End animation.

To ensure this final animation plays (rather than transitioning directly back into the Idle state, which should happen by default with you exit Falling state assuming your Lab5CharacterBP is working correctly), you will have to use Animation State Events. You can create these events in the Anim Graph then add them as Events in the Animation Blueprint’s Event Graph. By updating some of your Animation Blueprint variables based on animations completing, you can delay some of the animation state transitions to reduce a little of the state complexity. As a hint, this is what my End Jump State looks like:

![Animation State](image)

Note that there are many many ways to implement these sorts of state machines, so your solution does not need to look like mine by any means. Also I didn’t do a particularly good job blending the Jump_End, so if yours doesn’t look good either, don’t worry about it — that’s the artist’s job to fix it!

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 repository as your Canvas submission.