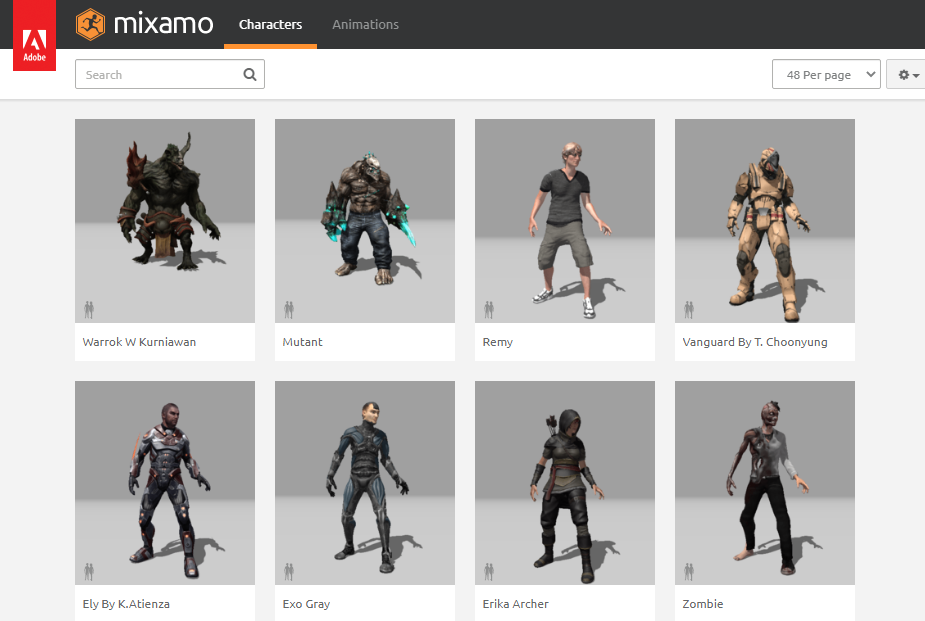
# CS3540 – Activity Third-Person Controller



In this activity, you will practice animating 3D characters in Unity and turn your FPS controller that you had been using so far into a simple third-person controller.

## What to do

Follow along with the instructions on the following pages.

## What to submit

Submit a screen recording of your animated character (similar to the sample provided below) with the intelligent camera behavior implemented in this activity. You can use Zoom to record your screen. Instead of a screen recording, you could also publish your scene for WebGL and share a public link to your scene hosted on itch.io/Unity Play. Pick whichever is easier for you. Make sure your itch.io/Unity Play release is playable (i.e., we can see/control the character).

**You should also submit your ThirdPersonController.cs script, which you will be implementing in Part 2.**

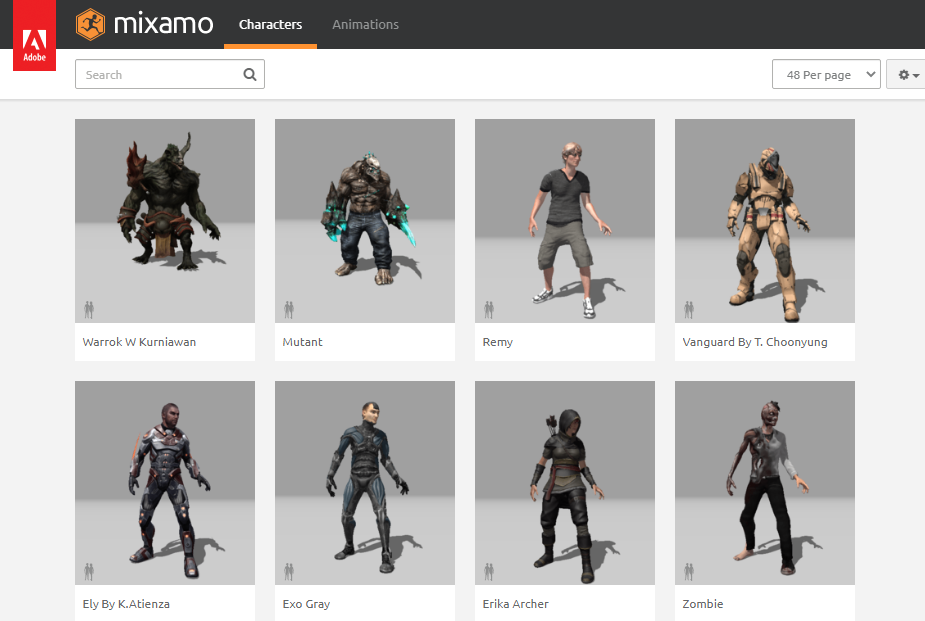
# Part 1 Simple Third-Person Controller

This week we covered how to use Mixamo to animate 3D characters. You will combine that experience with what you already have from last week to make a simple third-person controller. The end product of Part 1 will look like this:

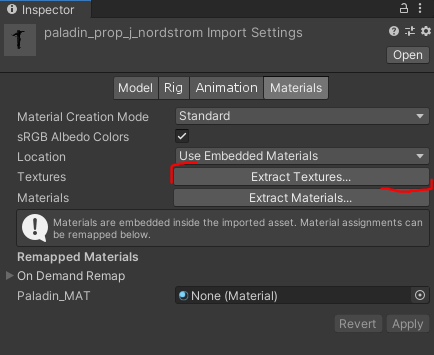
* <https://www.screencast.com/t/GbpDPeQncJ>

## What to do

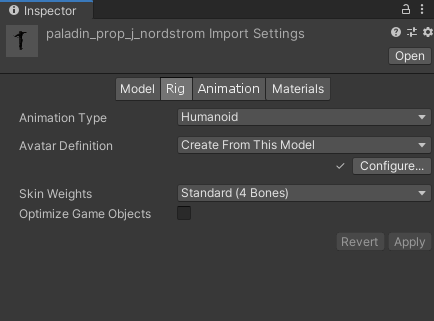
1. Go to mixamo.com and download **a new character** (any character you like) and some animations. At a minimum, you will need a T-pose, an idle animation, a walking animation, and a jumping animation.



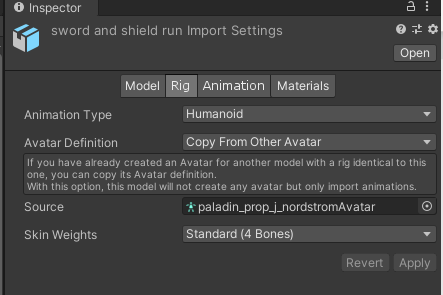
1. Duplicate your scene containing your FPS controller.
2. Import your character and animations into Unity.
3. When you first import the main character (the T-pose), it may not have the textures applied. As mentioned in the lecture videos, the textures are embedded into the characters. You will need to extract textures manually, which will enable you to use them in your project. To do so, select the T-pose in the **Project** **window** and go to **Inspector > Materials**. Here, click the **Extract Textures** button and save the textures.



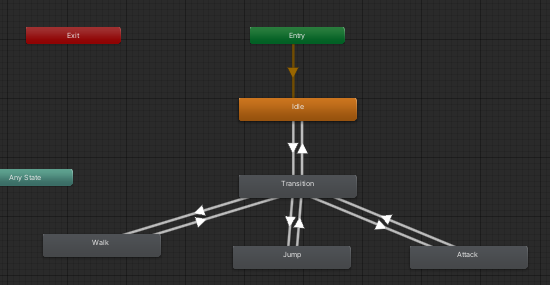
1. Having watched the videos, you should be able to configure the character and animations, but it wouldn’t hurt to have some reminders. 😊 For this main T-pose, you will want to create a **Humanoid** **avatar** **from** **the model** itself.



1. For subsequent animations, you will want to **copy the avatar from the main T-pose avatar** that you just generated.



1. Once all animations are ready, create an animation controller and add transitions among the animations. By default, the character should begin in the Idle state. When the player moves the character (using WASD or arrow keys), the character should be in the Walking state, meaning the walking animation should be played. When the space key is pressed, the character should play the jumping animation. Here is an example of what this could look like (note that the Attack animation is not required):



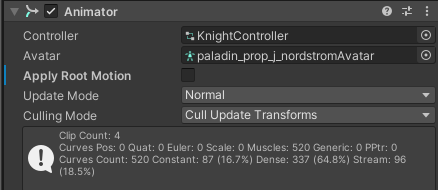
1. For player and camera control, you should use the FPS controller scripts that we used before. Simply locate the PlayerController.cs and MouseLook.cs script files in the Project window. Attach the former to your **character’s prefab** and the latter to **the camera**. Don’t forget to add a Character Controller to the new character model (which is what the PlayerController.cs script relies on for movement). The camera should be a child object of the character (similar to the FPS controller).



1. Because the goal of this activity is to create a simple third-person controller, you will want to position the camera accordingly to give a third-person perspective to the player. Refer to the sample video again.



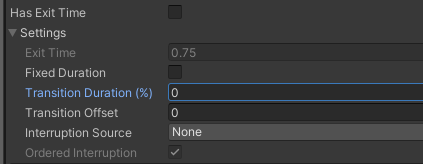
1. With these steps completed, you should be able to view the character from a third-person perspective and move the character around the scene using keyboard input.
2. As you will likely notice, the animations are not being changed yet. **And that’s your task in this OLA.** With most of the code already written, you should figure out where and how to change the animation applied to the character so that it starts walking when the player provides keyboard input, jumps up when the space key is pressed, and stands in idle mode when no input is provided. Your task is to update the PlayerController.cs script to play these animations when appropriate. The Animator class has custom methods for changing animation states, such as [SetInteger](https://docs.unity3d.com/ScriptReference/Animator.SetInteger.html)() and [SetTrigger](https://docs.unity3d.com/ScriptReference/Animator.SetTrigger.html)(). Check out the hyperlinked documentation to figure out what to do. It’s rather straightforward! If you need to refresh your memory of how to apply character animations, refer to the slides from the corresponding week.
3. One thing to note is related to the Animator component attached to the character. You will likely want to uncheck the Apply Root Motion option in order for the Walking animation to work properly. The nature of the walking animation involves character locomotion, which will be a problem due to the way the PlayerController.cs script is implemented.



1. If you want to make this more realistic, it is possible to add different animations like moving backwards, strafing, turning, etc. That said, doing the minimum will be fine for this OLA.
2. Note that this is a simple third-person controller. There are other (and better) ways of doing this.

## Some common pitfalls to avoid:

* Not turning off the Loop Time in animations that shouldn’t be looping (or vice versa).
* Not unchecking the “Has Exit Time” options for the animations (which will delay the transition among animations).
* In addition to unchecking the “Has Exit Time”, you may also need to manually set the transition duration and offset to 0. This depends on the animations you choose to download. Sometimes you need to do this; sometimes you don’t.



* Not changing the Rig properties of the animations.
* Not paying attention to the lecture videos. 😊

You now have a simple third-person controller. Scroll down to proceed to Part 2!

# Part 2 Cinemachine Camera

A picture containing grass, athletic game, sport, tennis

Description automatically generated

In Part 2, we will make our third-person controller look better. This is where we are headed:

* <https://www.screencast.com/t/Jk8lDeFiXDs3>

## Getting Started

1. For starters, duplicate the scene that you just used, where you implemented your third-person controller.
2. Then, move the camera outside the character prefab so that it is a standalone object in the hierarchy. Remember you’d made it a child object of your character for the third-person controller to work.

Text

Description automatically generated

1. **Remove (or disable) the MouseLook.cs script from the Camera object**. The behavior of the camera will be controlled by the Cinemachine controller we are about to add. But first we need to install the package!

## Installing Cinemachine

1. [Cinemachine](https://unity.com/unity/features/editor/art-and-design/cinemachine) is a comprehensive package provided by Unity that allows you to add complex camera behaviors in real time without writing much code manually.
2. To install the package, go to **Window > Package Manager**.
3. Search for “cinemachine” under Unity Registry and install the package:

Graphical user interface, text, website

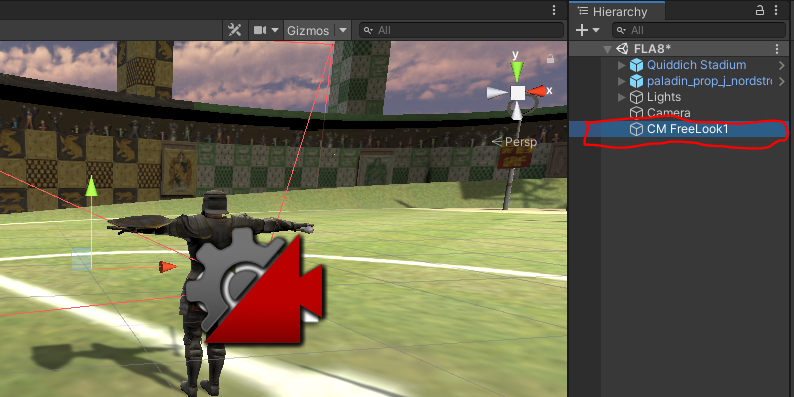
Description automatically generated

1. Once installed, Cinemachine will in the top menu:

Graphical user interface

Description automatically generated

1. This menu enables you to add a variety of cameras that can be used for different purposes. For a third-person controller, a FreeLook Camera is a natural choice. Therefore, go ahead and add one.
2. This will add a new GameObject to your hierarchy, which can also be seen in the scene.



1. This is essentially a camera controller, which is why I usually rename it “CameraController.”
2. If you look at this GameObject’s Inspector, you will see that it has a CinemachineFreeLook component, which enables you to change various aspects of this camera controller:

A screenshot of a computer

Description automatically generated with medium confidence

1. After adding this component to your scene, go to the Camera object in the scene and look at its Inspector. You will see that it now has a **CinemachineBrain** component attached. This is what the **FreeLook Camera** uses to tell the main camera in the scene what to do, which is what we will use to tell it to follow the player.

Graphical user interface

Description automatically generated

## FreeLook Camera

The FreeLook camera orbits around its target with three separate camera rigs defining rings around the target, namely **Top**, **Middle**, and **Bottom**. Each rig has its own radius, height offset, composer, and lens settings. Depending on the camera's position along the spline connecting these three rigs, these settings are interpolated to give the final camera position and state. You can view these camera rigs as wireframes when you select the FreeLook component in the hierarchy. The red circles represent the rigs:

A picture containing grass, athletic game, sport, tennis

Description automatically generated

We will look at the rigs in greater detail later, but let’s first make sure the camera can follow the character!

1. To tell the FreeLook camera to follow the player, all you need to do is specify the Follow target in the inspector window. The LookAt target enables you to make sure the camera always looks at and orbits around the player.
2. While you could directly drag and drop your player prefab into these two fields, the best practice for third-person controllers is to add a reference to the shoulders/head of the character. To do so, open the character prefab (by double-clicking it in the hierarchy) and add an Empty GameObject, which we will use as a reference point. Position this GameObject so that it is between the shoulders of the character as shown below:

Graphical user interface

Description automatically generated

1. Once the reference point is added, you can **save** the changes to the prefab and go back to the scene view. Under the prefab in the hierarchy, select the empty GameObject you just added and drag and drop it to the **Follow** and **Look At** fields of the CinemachineFreeLook component.

A picture containing text

Description automatically generated

1. When you select the **CinemachineFreeLook** controller in the hierarchy, you will see that this has added CineMachine guides to your game view as shown below, which can be disabled by unchecking the **Game Window Guides** option in the Inspector.

A picture containing grass, athletic game, sport, outdoor

Description automatically generated

1. With just these minor changes, go ahead and play the game to see what this looks like.
2. You should be able to control the camera using the mouse, but the behavior isn’t optimal yet, which we will fix in the next section.

## FreeLook Camera Settings

1. You may have noticed that the camera moves around the character in an orbit, which is after all what the Cinemachine Camera is designed to do. Most of the time, you will want to modify the orbit a tad to make sure the camera works in the way you want it to.
2. With your Cinemachine FreeLook controller selected in the hierarchy, look at the Inspector window and scroll down to the **Orbits** section of the CinemachineFreeLook component:

Graphical user interface, text

Description automatically generated

1. Here you can modify the height (y-position) and radius of the three rigs of the orbit around the character.
   1. The **height** values are in reference to the transform of the GameObject added to the **Look At** field, which in our case is the shoulder reference point. For a third-person controller, you want to position the TopRig well above the character’s head, the MiddleRig at around the same height as the shoulders, and BottomRig at around the waist level of the character.
   2. The **radius** simply defines the distance from the camera to the reference point. The larger the radius, the farther away the camera will go.
2. The **Binding Mode** defines the coordinate space used to interpret the distance to the target. To keep things short here, you should use “World Space”. You can learn more about all these settings at [this link](https://docs.unity3d.com/Packages/com.unity.cinemachine@2.3/manual/CinemachineFreeLook.html).
3. Here is what my rigs look like with the above settings:

A picture containing grass, sport, athletic game, outdoor

Description automatically generated

1. With these changes applied, play the game again. Hopefully the orbiting behavior looks a lot better now!
2. If you aren’t happy with the speed of the movement of the camera, you could change it using the Axis Control section of the component:

Graphical user interface

Description automatically generated

1. As you can deduce, these define the vertical and horizontal axes for blending between rigs.
   1. For **Y Axis** the Value range is 0 to 1 and represents the blend position between the Top, Middle, and Bottom rigs. The value 0.5 represents the middle rig, which is what you will want to you for a third-person controller.
   2. For **X Axis**, the Value is the angular deviation (in degrees) of the camera from directly behind the target. 0 means that it is centered, which is again what you want to use. After some trial-and-error, I changed the y-axis speed to 4 and x-axis speed to 500. You can definitely use different values.
   3. For more information on these and other settings, I strongly encourage you to check out the documentation linked earlier!
2. If you examine the component closely, you will see that it is possible to modify some further aspects of each rig. In this quick tutorial, we won’t touch those, but you’re more than welcome to play with them. For instance, if your camera is too close to the character (small radius for the rigs), you will likely see some jittery camera movement. To fix that, you can increase the damping values in the corresponding rigs. Did I say we weren’t going into the details of these additional settings?
3. Mind you, there are no hard and fast rules around these settings (for better or worse). You will need to play with them and adjust them to fit your character and your needs.
4. At this point, you should have a rather enhanced camera behavior implemented **without a single line of code**! While this is great, there is one issue with our player controller script; it doesn’t rotate the player! If you remember from previous weeks, we relied on the MouseLook.cs script to rotate the player, which I asked you to remove from the camera at the beginning of the tutorial. In the next section, we will look at how to fix this issue.

## Character Rotation

1. If you look at the MouseLook.cs script, you will notice that it rotates the character based on the mouse input so that the character always moves in the direction that the camera is facing. We will need to modify the PlayerController.cs script from last time to incorporate the same behavior.
2. For this purpose, add a new script “ThirdPersonController.cs”.
3. Copy/paste the contents of the PlayerController class into the body of the new script you just added.
   1. Make sure you **don’t** copy the class definition line (public class PlayerController : MonoBehaviour).
   2. Instead copy everything else within the scope of the class, i.e., between the curly braces {}.
   3. This is the way to duplicate a script in Unity without a lot of headaches. Take my word for it. 😉
4. Save the changes to the ThirdPersonController.cs script.
5. Go back to Unity and select your character in the hierarchy.
6. Uncheck or remove the PlayerController.cs script and add the ThirdPersonController.cs script to your character. Make sure your default settings match what you had in the previous script.

Graphical user interface, application

Description automatically generated

1. When you play the game, nothing will change yet. This means that you’ve kept the original copy of the PlayerController.cs script file, which is in fact a first-person controller. Now you can modify it to produce a third-person controller. Two gems of this class!
2. Under normal circumstances, I would ask you to figure out how to rotate the character on your own during an activity like this. But because you are completing this asynchronously, here is one way to achieve this rotation:

Text

Description automatically generated

1. Your task is to figure out where to put this in the ThirdPersonController.cs script, which should be fairly straightforward.
2. At this point, I sincerely hope this code snippet makes sense. If not, you may want to review some of the earlier lectures.

## Cinemachine Collider

1. As seen in the output video above, my game world has no walls or obstacles near the player (how convenient!). If you do have walls or other obstacles in your scene, you may notice that the camera passes through those obstacles, leading to a problematic view. This is a lot easier to illustrate with an example:

* <https://www.screencast.com/t/zeNlBZYRLx>

1. As you just saw in the demo, we can fix this problem by adding a Collider extension to the Cinemachine camera.

Graphical user interface, text, application, chat or text message

Description automatically generated

1. The Collider extension simply adds a collider to the camera and enables the camera to avoid obstacles.

Graphical user interface

Description automatically generated

1. In so doing you want to ignore collisions with the player if the camera gets too close to them. Therefore, you will want to specify the tag of the player, which is typically “Player”. Make sure your character has that tag too.
2. The full set of attributes and their descriptions are available at the following link:
   1. <https://docs.unity3d.com/Packages/com.unity.cinemachine@2.8/manual/CinemachineCollider.html>
3. Very quickly, for Strategy (which specifies what happens when a collision occurs) “Pull Camera Forward” typically works fine, which essentially refocuses the camera on the reference point along its z-axis.
4. Because the camera is automatically refocused, it is possible to smooth the transition through the Smoothing Time, Damping, and Damping When Occluded attributes. I like the above settings, but feel free to play with them to get the effect you like.

## More on Cinemachine

In summary, the goal of this OLA was to introduce the basics of Cinemachine, with the hope that you will explore it further on your own. Here are some resources on Cinemachine (where I shamelessly borrowed some of the content for this OLA).

This is the official documentation:

* <https://docs.unity3d.com/Packages/com.unity.cinemachine@2.8/manual/index.html>

Here are some tutorials by Unity:

* <https://learn.unity.com/tutorial/cinemachine#5c7f8528edbc2a002053b4ee>
* <https://www.youtube.com/watch?v=537B1kJp9YQ>

\* The video tutorial shows how to achieve the same effect using a Virtual Camera instead of a FreeLook Camera, which is instructive.

*Oh, dear! I spent too much time writing this tutorial. I should have just recorded a video and be done in 10 minutes, instead!*