Updates

Version 1.1

This latest update includes an 'independent' version of the Adventure Camera. This means it does not require an Adventure Controller and instead uses a raw Transform as an anchor as well as several other properties.

I didn't use an inheritance model, but instead cloned the code. I did this to avoid confusion and keep you from having to add references to the Adventure Controller.

This release also fixes a small bug when it comes to the pitch of the camera.

Overview

Action/Adventure games like Tomb Raider, Uncharted, Assassin's Creed, and Batman Arkham Origin are huge. As Remi Lacoste (the Lead Camera Designer for Tomb Raider 2013) put it, "The camera is the narrator and acts as the soul of the game."

When creating a AAA quality game, having a "third-person camera" isn't good enough. In fact, when you dissect Tomb Raider's camera, you'll see that the camera seamlessly transitions between several different types of camera behaviors. It's this attention to detail that we've created in the Adventure Camera & Rig.

The Adventure Camera & Rig is a multi-behavior camera built specifically for quality 3rd Person Action/Adventure games. Use it as a basis for your custom camera system or out-of-the-box to kick start your own adventure!

Note: The code and tips in this package can be used with any type of camera.

Components

The Adventure Camera & Rig isn't just a camera, it includes the following pieces that work together to create a full experience:

Adventure Camera Rig – Houses the standard camera. If the camera is the lens, this is the dolly.

Adventure Camera Controller – Represents the player and controls movement

Mecanim Animator – Fully detailed Animator tree containing movements such as walk, jog, run, strafe, walk backwards, idle, etc.

Note: All models and animations come from Unity's Raw Mocap data for Mecanim. These assets can be easily replaced with your custom ones.

Debug Logger – A class designed to write debug information to the console, screen, and out to files. Included as a bonus just for you! ☺

Object Pool – A super-fast object caching class used to keep your game from having to reallocate objects over and over. Use it with any C# class you want.

Features

The Adventure Camera & Rig supports the following features:

- Physics based spring camera for smooth movement
- 360-degree rotatable view of the character
- View rotation limits can be enabled, set, and disabled
- Walking, jogging, running, walking backwards, walking sideways, and idling
- Smooth transitions between idles, jogs, and runs
- 180 and 135-degree pivoting
- Exploration Stance for moving through the environment
- Melee Stance for keeping the avatar facing forward
- Targeting Stance for using ranged weapons
- Modern 3rd Person follow behavior where the player rotates around the camera
- Traditional 3rd Person follow behavior where the camera rotates around the player
- 1st Person targeting behavior for precise ranged combat
- Basic collision detection so the camera won't move through objects
- Out-of-the-box support for Xbox 360 gamepad and keyboard/mouse

In addition to the code and assets defined, we'll provide you with a detailed code walk-through to help you understand how the system works. If you want to modify the camera, use the walk-through to guide you.

The walk-through also includes tips on creating other types of cameras and using the Mecanim Animator.

Usage

To use the Adventure Camera & Rig in your scene, follow these simple steps:

1. Create a new project or open an existing one

Follow the standard practice for creating or opening a scene. Nothing special here.

2. Add this package to your project

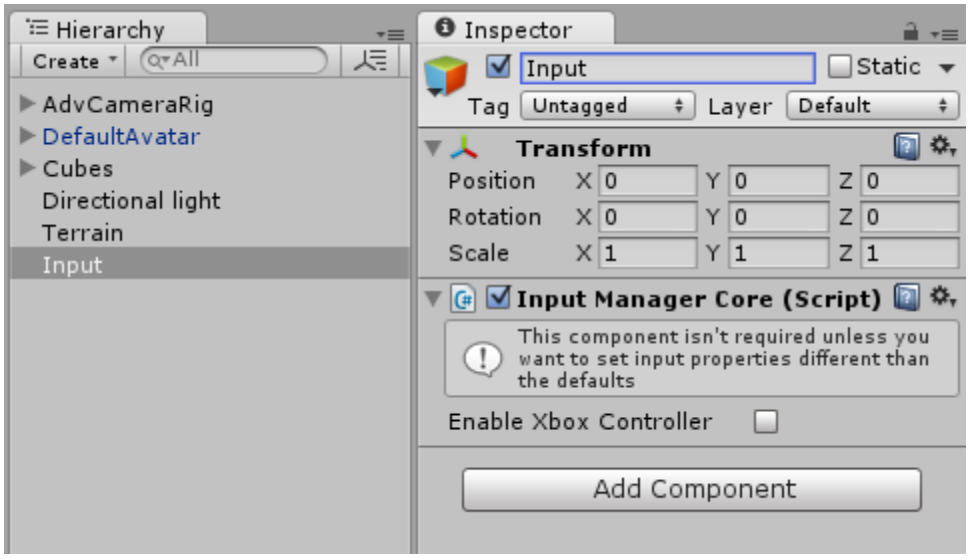
This package contains the camera rig, controller, Animator, and other code you'll need to get up and running. You can keep all these assets in the folders they are currently located or move them as needed.

3. Setup additional Inputs

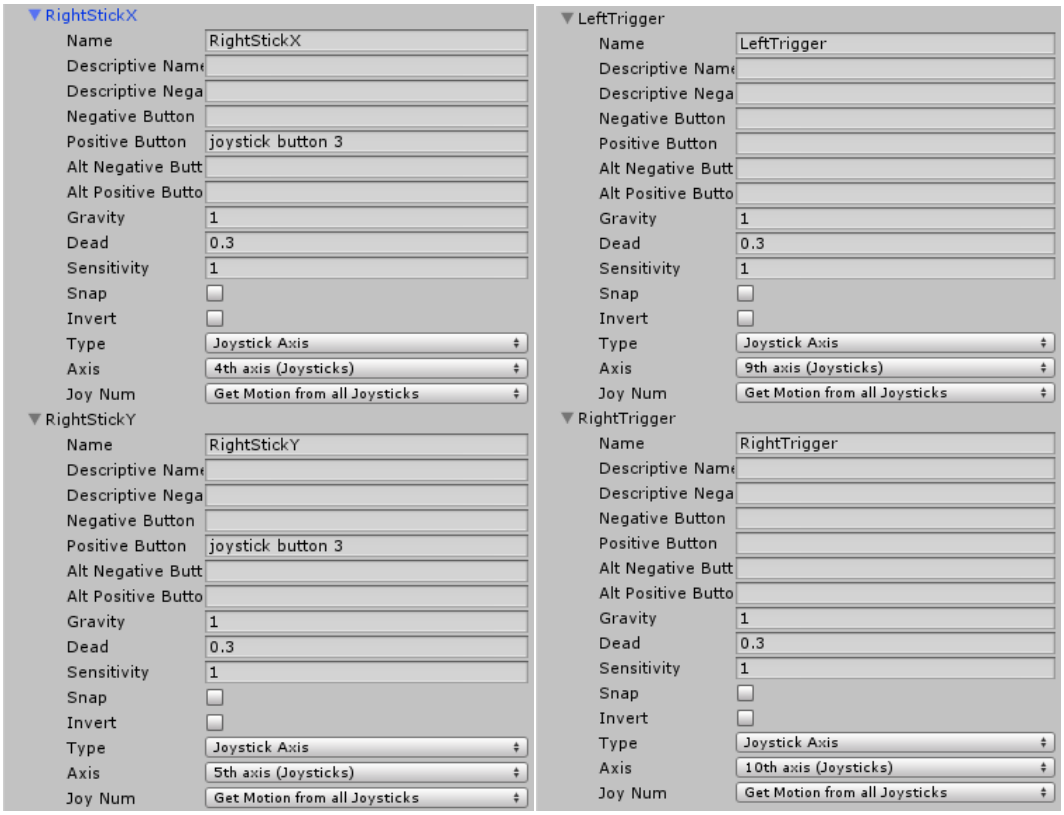
The ootii InputManager class in the package supports the keyboard, mouse, and the Windows version of the Xbox Controller. By default, the Xbox controller is disabled.

If you don't want the Xbox controller enabled, move on to step 4.

If you wish to enable the Xbox controller, add the InputManagerCore script (`ootii/Framework_v1/Code/Input`) to a game object in your scene and set the 'Enable Xbox Controller' flag there. This will keep you from having to modify code and keep the value during updates.



In the Unity InputManager (Edit | Project Settings | Input), add the following entries. These will allow the code to support viewing and zooming.



4. Add an avatar to the scene

In this package, you'll find the sample Unity character in the following folder: ootii/AdventureCamera/Demos/Models/Characters. Drag this avatar into your scene or add your own.

If you add your own, ensure that it has a 'Humanoid' Rig defined. If you need help, you can follow Unity's instructions: http://docs.unity3d.com/Documentation/Manual/AvatarCreationandSetup.html

4.a. Associate the 'Humanoid' Animator to the controller

Under ootii/AdventureCamera/Demos/Models/Animations, you'll find a Mecanim Animator called 'Humanoid'. Drag this to the 'Controller' slot of the 'Animator' component that was created when you added your avatar.

Note: All models and animations come from Unity's Raw Mocap data for Mecanim. These assets can be easily replaced with your custom ones.

4.b. Add a Unity Rigid Body component to your avatar

See the settings in the picture for step #5 below.

4.c. Add a Unity Character Controller component to the avatar

See the settings in the picture for step #5 below.

Setting the 'Interpolate' property to 'Interpolate' is important for the physics and camera to run smoothly

4.d. Set the Unity tags

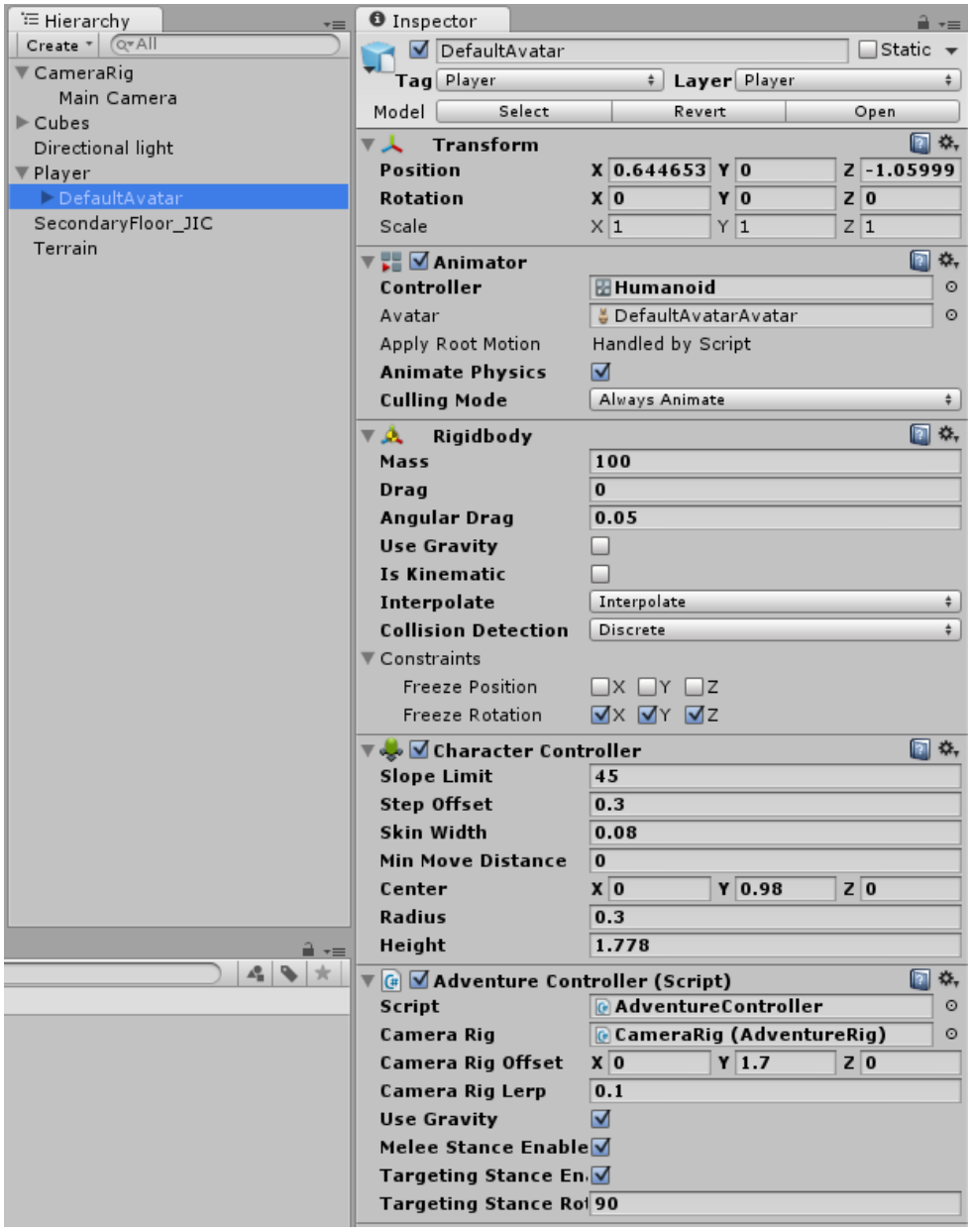
Set the 'Tag' property to 'Player'. This is just a good standard.

Set the 'Layer' property to 'Player'. You'll need to add this layer and ensure that it's in the "User Layer 8" slot. If you use a different slot, you'll need to modify AdventureRig.cs line 608. The "8" will need to be the layer you used instead.

5. Create the Adventure Controller

To do this, drag the 'Adventure Controller' component onto the avatar you added to the scene. You'll find the 'Adventure Controller' under ootii/AdventureCamera/Code/AI/Controllers.

Once you do this, you'll see the following in the inspector:



For a detailed explanation of the settings, see the 'Adventure Controller Settings' section below.

6. Create the Adventure Rig

We're going to place the existing 'Main Camera' into a rig. I like to think of the existing camera as the lens and this new object as the dolly or rig that moves the camera lens.

6.a. Create a GameObject for the rig

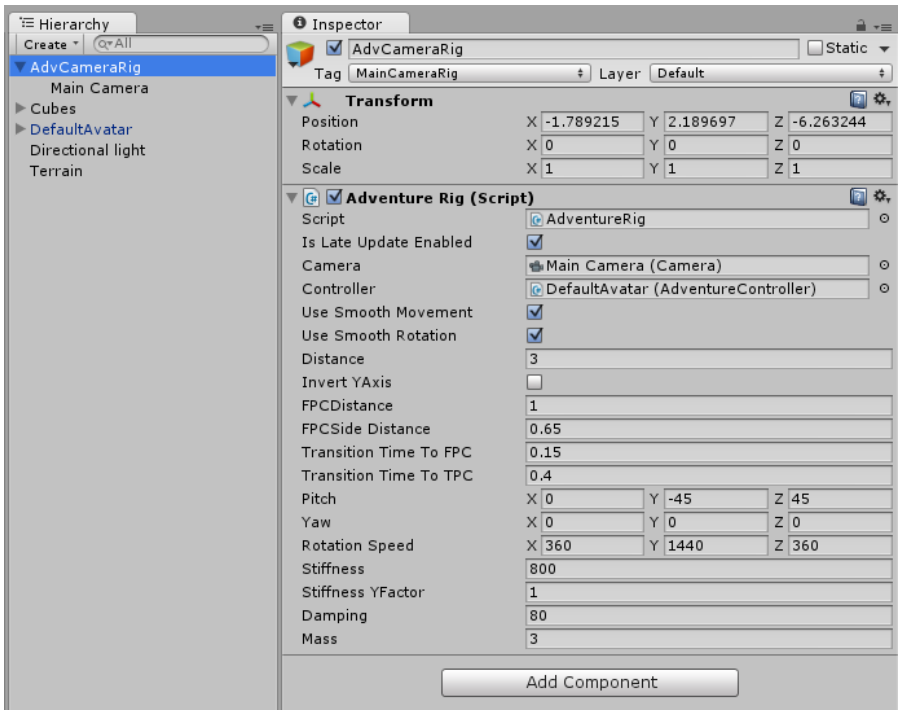
Simply add an 'Empty' game object to your scene using the menu item: GameObject | Create Empty.

Rename this to something like 'Camera Rig' and drag the existing 'Main Camera' onto it. Now the 'Main Camera' is a child of the rig.

6.b. Add the component

Drag the 'Adventure Rig' component onto the 'Camera Rig' you just created. You'll find the 'Adventure Rig' under ootii/AdventureCamera/Code/Cameras.

Once you do this, you'll see the following in the inspector:



For a detailed explanation of the settings, see the 'Adventure Rig Settings' section below.

7. Connect the Adventure Rig and Adventure Controller

In the setting inspectors, you'll see an option for 'Camera Rig' or 'Controller'. Drag the appropriate scene objects into the field for each. This will tell the controller about the camera rig and the camera rig about the controller.

8. Add screen logging (optional)

Included in the Adventure Camera & Rig is the Debug Logger. Use this to write debug info to the console, to the screen, or to a file. To get more information, check out our online documentation:
http://www.ootii.com/UnityDebugLogger.cshtml

To setup the debug logger, simply add the 'Log' component directly to the camera lens. You can find the 'Log' under ootii/Framework_v1/Code/Utilities/Debug.

9. Run the game

That's the meat and bones of it. With that in place, you can now run the game using a keyboard and mouse or Xbox 360 controller.

By default:

WASD or **Left-Stick** moves the avatar

Mouse or **Right-Stick** rotates the view

Right-Mouse-Click or **Left-Trigger** goes into the Targeting Stance

T-Press changes from Exploration Stance to Combat Stance and back.

Adventure Controller Settings

The code is available for you to customize or change the controller as needed. However, if you want to use the camera system out-of-the-box, these settings can help you customize it.

Camera Rig
References the Adventure Camera Rig you setup in step #5

Camera Rig Offset
The 'anchor' position (relative to the avatar) that is the orbit center of the camera. Typically this is the head of your avatar.

Camera Rig Lerp
As the avatar moves up (say in a jump or climb), how quickly the camera moves up to follow it. This allows us to have a different movement feel for a camera going up and down than side to side.

Use Gravity
Determines if we apply gravity during movement. This is applied in the controller's ApplyMovement function.

Melee Stance Enabled
Determines if the controller can enter the melee stance when the player activates it.

Targeting Stance Enabled
Determines if the controller can enter the targeting stance when the player activates it.

Targeting Stance Rotation Speed
Degrees per second the player will rotate at when in the targeting stance.

Notes

It's important to note that the speed of player movement is controlled by the mocap data since we're using Root Motion. While you could change this pretty easily in code, the system will built to use Root Motion.

Non-Targeting rotation speed is dictated by the camera rotation speed (see below).

Adventure Rig Settings

The code is available for you to customize or change the rig as needed. However, if you want to use the camera system out-of-the-box, these settings can help you customize it.

Is Late Update Enabled
Determines if the rig should process during the typical LateUpdate function or if it should wait for something like the controller to call it directly. This is useful if the controller is doing positioning in its LateUpdate function.

Controller
References the Adventure Controller you setup in step #4

Use Smooth Movement
Determines if we use smoothing during linear movement

Use Smooth Rotation
Determines if we use smoothing during rotation. In truth, the 'Use Smooth…' options should stay in synch.

Distance
Determines how far from the orbit center the camera should stay.

Invert YAxis
Allows you to invert the input direction for looking up and down with the camera.

FPC Distance (First Person Camera Distance)
When in the Targeting Stance, this is the new distance the camera stays from the center-point.

FPC Side Distance
When in the Targeting Stance, this is the distance to the right the camera is from the avatar. It creates an 'off-the-shoulder' kind of view.

Transition Time TO FPC
Determines how long it will take for the camera to move from third person to first person position.

Transition Time to TPC
Determines how long it will take for the camera to move from first person to third person position.

Pitch (Value, Min, Max)
These three values represents the camera's pitch (degrees around the x-axis), minimum pitch, and maximum pitch.

Looking straight forward has a pitch of 0. Looking down, creates a negative pitch (in degrees). Looking up creates a positive pitch (in degrees). Set limits here to keep the camera from flipping over the avatar.

Yaw (Value, Min, Max)
These three values represents the camera's yaw (degrees around the y-axis), minimum yaw, and maximum yaw.

Looking straight forward has a yaw of 0. Looking left, creates a negative yaw (in degrees). Looking right creates a positive yaw (in degrees). Set limits here to keep the camera from rotating 360-degrees around the avatar.

To disable the min and max, simply set them to 0.

Rotation Speed (for each axis)
Represents how quickly the camera rotates around the avatar. The rotation is in degrees per second.

Stiffness
This property relates to the physics based spring camera motion.

Imagine there is a spring between where the camera is and where the camera wants to be. The stiffness or tension determines how hard the camera will be pulled to the desired position.

Stiffness YFactor
This property relates to the physics based spring camera motion.

It controls the stiffness on the Y axis. It's a relative value for 'Stiffness'. Meaning 1 = 100% of the 'Stiffness' value and 0 = 0% of the 'Stiffness' value.

Damping

This property relates to the physics based spring camera motion.

Think of this like friction. It determines how long it takes for the camera to start moving and how quickly it will slow down once it is moving.

Mass

This property relates to the physics based spring camera motion.

This is the overall 'weight' of the camera and is used when determining acceleration and velocity.

Code-Walkthrough

This document was meant to give you a basic understanding of the camera and get you up and running quickly. For an understanding of the camera concepts and code, see the 'Code' document.

Support

If you have any comments, questions, or issues, please don't hesitate to email me at support@ootii.com. I'll help any way I can.

Thanks!

Tim