



IMPORTANT

The Archery Motion pack requires the following:

[Motion Controller](#) v2.8 or higher

Mixamo's free [Pro Longbow Pack](#) (using Y Bot)

Importing and running without these assets will generate errors!

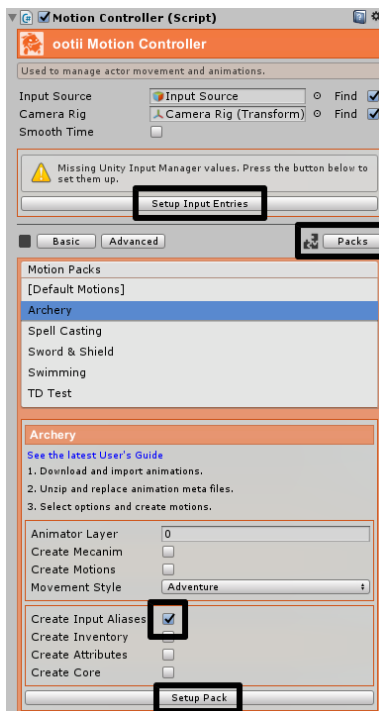


Demo Quick Start

[YouTube Setup Video](#)

This quick start is simply to get the demo up and running in a self-contained project.

1. Start a new Unity 2017 Project.
2. Download and import the [Motion Controller](#) asset.
3. Download and import the [Archery Motion Pack](#) asset.
4. Download Mixamo's [Pro Longbow Pack](#) using "Y Bot" (see this [flow](#)).
5. Unzip Pro Longbow animations to project's Mixamo folder.
...\Assets\ootii\Assets\MotionControllerPacks\Archery\Content\Animations\Mixamo
6. Unzip AnimationMeta.zip from pack's "Extras" folder to project's Mixamo folder (see [steps](#)).
...\Assets\ootii\Assets\MotionControllerPacks\Archery\Content\Animations\Mixamo
7. Let Unity import the animations and meta data. Then, close and re-open Unity.
8. Open the "demo_AMP" demo scenes
...\Assets\ootii\Demos\MotionControllerPacks\Archery\Scenes\
9. Setup input entries on the Packs tab.



10. Press play.

DEFAULT CONTROLS

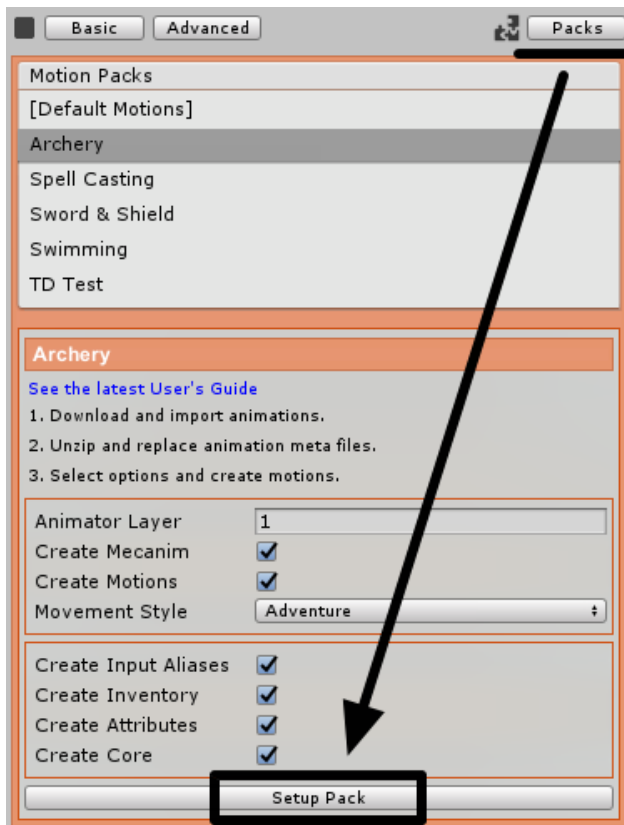
WASD = Move
Mouse = Rotate
LShift = Run
2 = Equip the bow
RMB = Aim the bow
LMB = Fire an arrow
LMB + RMB (held) powers up the bow



Custom Quick Start

This quick start assumes you've worked with the Motion Controller and that you have a Motion Controller enabled character in your scene already.

1. Open your MC enabled project and scene.
2. Download and import the [Archery Motion Pack](#) asset.
3. Download Mixamo's [Pro Longbow Pack](#) using "Y Bot" (see this [flow](#)).
4. Unzip Pro Longbow animations to project's Mixamo folder.
...\Assets\ootii\Assets\MotionControllerPacks\Archery\Content\Animations\Mixamo
5. Unzip AnimationMeta.zip from pack's "Extras" folder to project's Mixamo folder (see [steps](#)).
...\Assets\ootii\Assets\MotionControllerPacks\Archery\Content\Animations\Mixamo
6. Let Unity import the animations and meta data. Then, close and re-open Unity.
7. Open the scene.
8. Setup motion pack on the Packs tab.



9. Bow motions are added to the Advanced tab.

DEFAULT CONTROLS

WASD = Move
Mouse = Rotate
LShift = Run
2 = Equip the bow
RMB = Aim the bow
LMB = Fire an arrow
LMB + RMB (held) powers up the bow



Foreword

Thank you for purchasing the Archery motion pack!

I'm an independent developer and your feedback and support really means a lot to me. Please don't ever hesitate to contact me if you have a question, suggestion, or concern.

I'm also on the forums throughout the day:

<http://forum.unity3d.com/threads/motion-controller.229900>

Tim

tim@ootii.com

Contents

Overview.....	5
Motion Packs	10
Bows	18
Arrow	20
Motions	22
Frequently Asked Questions.....	25
Mixamo Animation Download.....	37



Overview

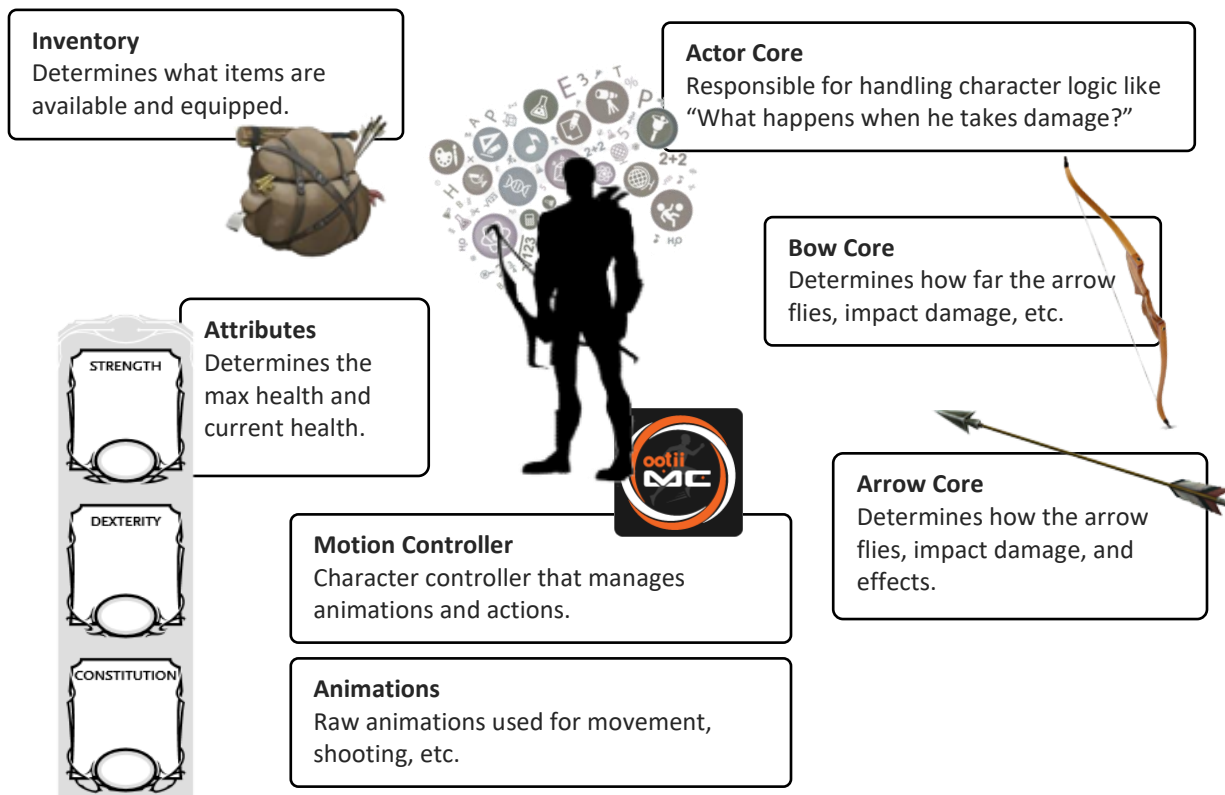
This asset is a bit different from my other assets as it's an add-on pack to the Motion Controller. So, the Motion Controller is required for this asset to work. It also requires Mixamo's free Pro Longbow Pack animations. When those are combined with this asset, your character gains the ability to equip, shoot, and react to bows and arrows.

I've tried to create this asset in a very modular way. This way, you can use it as-is or customize it to work with your game. For example, you can use my basic inventory system or replace it with something like Inventory Pro. In addition, you can use my basic arrow or create an arrow with special effects. There's a lot of things you can tweak to get exactly what you want.

If you think this add-on is missing a key feature or option, let me know.

Key Components

In order to support different attribute systems, different inventory systems, different arrow types, etc., I created this add-on with multiple components. This is a quick overview and I'll go into them in more detail:

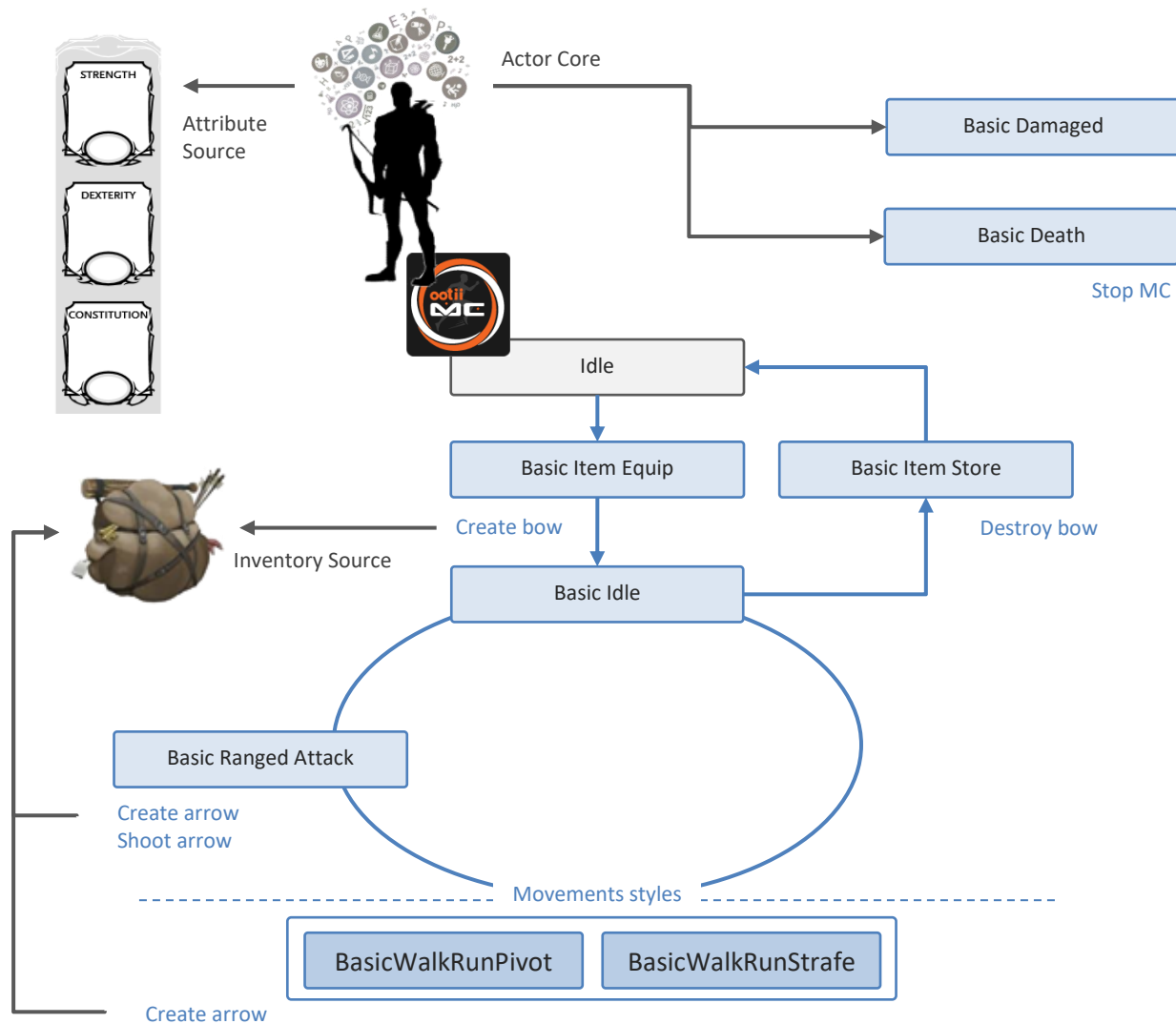


Other than the Motion Controller, each component can be replaced to fit your game's specific needs.



Process Flow

Just like the motions that come with the Motion Controller, the Archery Pack is composed of several motions. These motions (blue rectangles below) work together to create the full range of capabilities.



If you don't need a bow motion, want to change how a motion works, or want to add an additional bow motion... you do it just like any other motion. These bow motions are just custom motions that take advantage of Mixamo's Longbow animations.

I've created this pack based on how I think archery should work. As you can imagine, there are lots of ways we could do it and your game may be different than mine. If you want the motions to act differently, you are welcome to change these motions or shoot me an email and there may be a feature or option I can add.

For determining which bow to create and which arrow to shoot, we use an "Inventory Source". To manage health, we use an "Attribute Source".



Motion List

These are the motions included with the Archery Motion Pack. This is just a brief description and I'll go into the properties of each motion later.

Basic Idle: Simple idle animation with bow in hand.

Basic Item Equip: Unsheathe animation that determines the right bow to equip and then equips it.

Basic Item Store: Puts the bow away and returns to the normal (non-bow) idle.

Basic Walk Run Pivot: Movement motion with bow in hand. This movement style is more "Adventure" style and if use with an orbiting camera, allows the character to walk towards the camera.

Basic Walk Run Strafe: Movement motion with bow in hand. This movement style is more "Shooter" style and always has the character facing forward. It supports strafing.

Similar to the standard WalkRun... motions, you would only have WalkRunPivot or WalkRunStrafe enable. Not both.

Basic Ranged Attack: Motion used to fire the arrow.

Basic Damaged: Animation used when the actor is hit by an arrow.

Basic Death: Animation used when the actor is killed by an arrow.

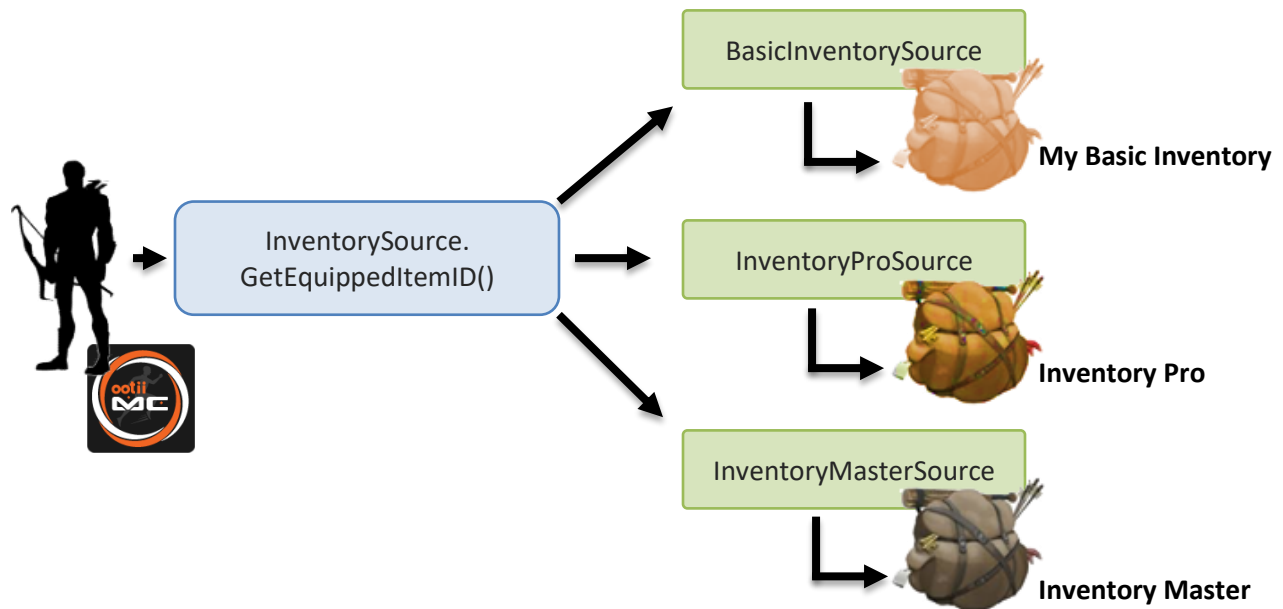


“Sources” Introduction

There’s lots of inventory solutions on the asset store. Some developers use Inventory Pro, some use Inventory Master, and some create their own solutions. So, I have to create a way to support them all.

To do this for inventories, attributes, input, etc., I’ve come up with the concept of “sources”. Basically, a source is a bridge between my asset and every other asset. Just like I did with Input Sources, I’ve “Inventory Sources” and “Attribute Sources”.

Let’s look at how this works:



The archery motions know there’s an “Inventory Source” on the character. It doesn’t care if it’s a “Basic Inventory Source”, an “Inventory Pro Inventory Source”, or an “Inventory Master Inventory Source”. An archery motion can then ask the “Input Source” for the arrow type that is in the quiver.

Depending on the inventory source type, it asks the actual inventory solution for what’s in the quiver. The resource path is returned to the archery motion and it creates and shoots the arrow.

This is exactly how Input Sources work and how Attribute Sources work too.

Interfaces

All this is capable because of interfaces.

To learn more about interfaces, here are some resources:

[Input Sources and Interfaces](#)

[Interfaces in C# \(For Beginners\)](#)

[Unity Interfaces](#)



“Cores” Introduction

A “core” represents the heart-beat of a character or object. Typically, this is the MonoBehaviour whose Update() function controls the object. For this pack, we have three specific cores:

Actor Core – This core will be a component on your characters and has OnDamaged() and OnDeath() functions. It allows the actor to respond to arrow hits and other events.

The Actor Core implements the IActorCore interface. This way, you can use the interface with your own “core” (if you have one).

Bow Core – The Bow Core is a component on the bow prefab and defines things like Draw Power and Impact Power. In this way, you could have two different bows with unique capabilities.

This core is what actually creates and launches the arrow.

Arrow Core – As a component on the arrow prefab, this core is responsible for testing for collisions and applying damage on impact. It also knows how to self-destroy the arrow when it expires.



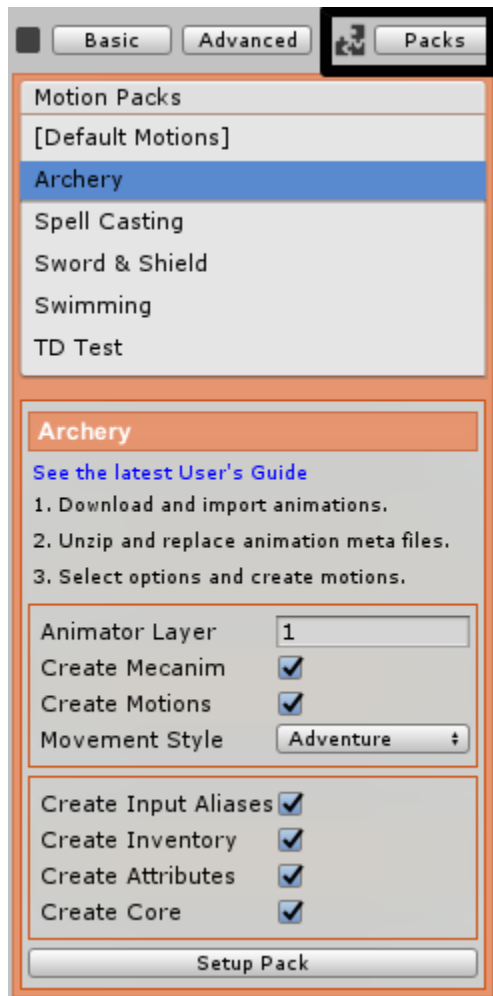
Motion Packs

The Motion Controller UI has changed slightly and now includes a button for “Packs”.

When the “Packs” button is pressed, the Motion Packs imported into the project will be listed. Selecting a pack will provide detailed information about the pack.

In this case, we select the “Archery” pack and the Archery Pack details are listed.

Each pack may show different options. For the Archery Pack, I’ll go through each option. When the “Setup Pack” button at the bottom is clicked, the checked options are run.



Animator Layer – Determines which layer we’ll put the equip, store, and attack motions on. This allows for equipping and shooting while walking.

Create Mecanim – This option is used to create/recreate the Mecanim sub-state machines that are used by the pack’s motions. This is exactly what we do with normal motions. I just create a short cut here.

Create Motions – This is the object that actually creates the bow motions. Based on the **Movement Style** options, certain motions will be enabled and disabled (see below).

Create Input Aliases – When checked, all the required input aliases used by the motions are created in Unity’s Input Manager. This only needs to be done ones for the whole project.

Create Inventory – If you don’t have an inventory solution, this option will create a “Basic Inventory” component and add it to your character.

Create Attributes – If you don’t have an attribute solution for things like health, this option will create a “Basic Attributes” component and add it to your character.

Create Core – If you don’t have an Actor Core on the character, one

will be created for you.

Typically, you’ll just leave all these options checked before pressing “Setup Pack”. However, as you customize your character and include other assets you may no longer need my basic inventory solution.

The following pages show what is created by the pack (assuming all options are checked):



Motion List

With the motions created, if you go back to the “Advanced” tab, you’ll see the new bow motions listed and setup based on the options you’ve checked.

Motions			
Motions determine how your actor will move, rotate, and animate.			
Motions			
Basic Idle		0	<input checked="" type="checkbox"/>
Basic Damaged		30	<input checked="" type="checkbox"/>
Basic Death		100	<input checked="" type="checkbox"/>
Basic Walk Run Pivot		5	<input checked="" type="checkbox"/>
Basic Walk Run Strafe		7	<input checked="" type="checkbox"/>
Basic Item Equip		20	<input type="checkbox"/>
Basic Item Store		8	<input type="checkbox"/>
Basic Ranged Attack		15	<input type="checkbox"/>
Bow - Dodge		13	<input checked="" type="checkbox"/>

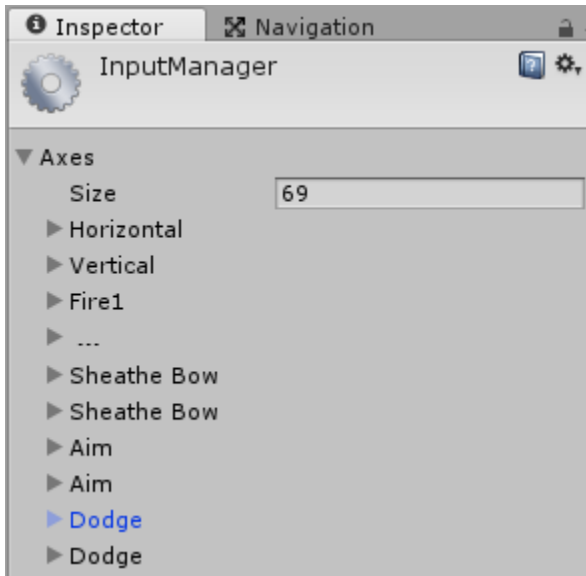
Once setup, you can treat them like any other motion. You can disable them, remove them, activate them using AI, etc. They are motions just like any other motion.

I did a quick overview of the motions earlier and I’ll detail them later. I just wanted to show that the motions get created.



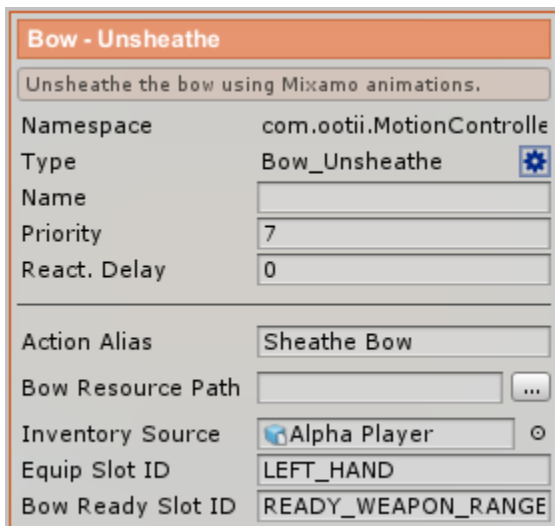
Inputs

For the most part, I use Unity's standard input for moving and controlling the character. However, some new input entries are added to support things like aiming and sheathing the weapon.



For all these extra entries, there is a 'keyboard' entry and an Xbox controller entry.

Following my Motion Controller and Input Source standards, each motion that requires a special key (like Bow – Unsheathe), I allow you to change the input entry. I call these Action Aliases and they trigger the motion. For example, Bow – Unsheathe uses the “Sheathe Bow” entry you see above.



Using this approach, you can change the input entry the motion uses by setting the “**Action Alias**” or changing the key or button that the alias represents.

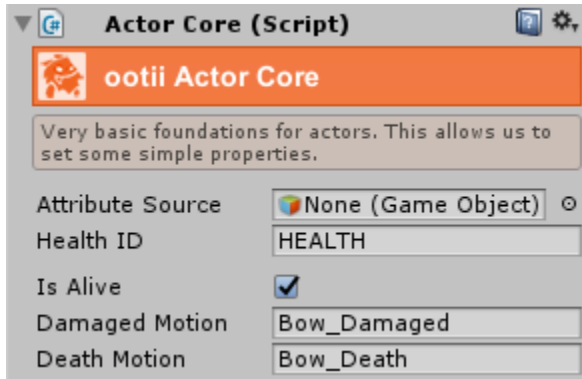
How you change the key or button depends on the input solution you're using. For standard Unity Input Manager, you'd just select the 'Edit | Project Settings... | Input' menu item in Unity.



Actor Core

As I mentioned, the Actor Core lives on all your characters and represents the decision making and logic part of your character. When an arrow hits a character with an Actor Core, it's the Actor Core that receives the damage and reacts appropriately.

Note that you don't need an Actor Core if the object is an object that doesn't get damaged or destroyed.



The Actor Core is pretty basic. It has the following properties:

Attribute Source – The component that contains the actor's attributes.

Health ID – String that is the ID of the attribute that holds our health value.

Is Alive – Simple boolean that determines if the object is still active.

Damaged Motion – Name of the motion to activate when damage is taken from an arrow.

Death Motion – Name of the motion to activate when the actor takes so much damage that it should die.

Code Summary

Internally, the Actor Core has a couple of key functions:

OnDamaged() – This function is called by the ArrowCore when it impacts an object that has an ActorCore. This is how the arrow tells the character it has been hit and how much damage is done.

In this function, the ActorCore asks the Attribute Source for how much health exists. If the damage exceeds this health, the character dies.

OnDeath() – This function is called by the OnDamaged function if the damage exceeds the character's health.

Once the death animation finishes, the Motion Controller is disabled.

IActorCore Interface

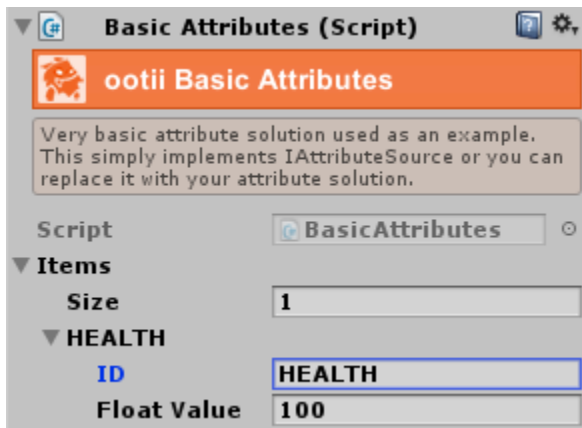
If you have your own "actor heart beat" MonoBehaviour, you can simply add the IActorCore interface to your class and ignore this one. Again, the goal is to be modular and replacable.



Basic Attributes

While we could have simply stored the health attribute inside the Actor Core, I wanted to make sure we were modular enough to support RPG assets that may be managing the attributes.

If you recall from earlier, I talked about Attribute Sources. A source is simply a bridge that I can use to grab attributes regardless of the solution being used. This “Basic Attributes” component is my very very basic version of an RPG attribute asset. It implements the IAttributeSource interface as all Attribute Sources would.



By implementing attributes this way, you can add new attributes to your game. For the Archery add-on, we only care about one attribute:

HEALTH – Current health a character actually has. Remember this is the ID that we used in the Actor Core above.

Code Summary

As an Attribute Source, the following functions are available:

GetAttributeValue() – This function returns the value of the attribute given the name.

SetAttributeValue() – This function sets the value of an attribute given its name.

As you can image, these two functions are used by the Actor Core to get and store the character’s current health. The Actor Core will determine if the character is simply damaged or killed based on the “HEALTH” attribute.



Basic Inventory

While you can use any inventory solution you want, I've included a "Basic Inventory" solution for you. As we've been talking about, any inventory solution you use will need to implement the `IInventorySource` interface. This way, we know how to retrieve information from it.

My "Basic Inventory" solution is very basic, but it is an Inventory Source. My implementation contains three lists; Items, Slots, and Weapons Sets.

The **Items** list contains all the items that the character has in his inventory and some properties for them.

The **Slots** list represents the usage of items.

So, the "LEFT_HAND" slot defines what is equipped in the left hand.

The "READY_PROJECTILE" slot determines what arrow is ready to be used.

In the example to the left, the character has a bow ("Longbow_01") and a set of arrows ("Arrow_01"). Currently he has nothing in his left hand, but his quiver is holding the set of arrows.

The **Weapon Sets** allow you to group items so they can be equipped and stored at the same time. For example, "Weapon Set 2" includes Longbow_01 and Arrow_01. You can imagine that another weapon set may be a sword and shield.

Items

Each item has the following properties:

ID – Simple string that uniquely identifies the item.

Equip Motion – Name of the motion used to equip the item

Store Motion – Name of the motion used to store the item

Instance – Edit-time created instance that is the item

Resource Path – This is a path of a Unity Resource Folder where we can find the prefab that represents the item. We use this value to

create the bow and arrows when the time comes.



To learn more about Unity resources, look here: <https://unity3d.com/learn/tutorials/topics/best-practices/resources-folder>

Slots

Each slot has the following properties:

ID – Simple string that uniquely identifies the slot.

Item ID – ID of the item that is currently in the slot. An empty value means the slot is empty.

Changing Items

When you equip a weapon set (like Weapon Set 2), the weapon set items are equipped. Those weapon set items define which items go into which slot. Hence, the Slots items will be updated.

So, by default the “READY_PROJECTILE” slot is filled with the “Arrow_01” item. The “Arrow_01” item says where the prefab is that defines the arrow to spawn using the item’s “Resource Path” property.

Say you create a new kind of arrow... You would add that as a new item in the Items section and give it an ID like “Arrow_02” and set the Resource Path to your new prefab. Then, in the weapon set item that sets the item in the slot (“READY_PROJECTILE”), you’d change the entry’s “Item ID” value from “Arrow_01” to “Arrow_02”.

If you’re changing arrows another way, you can just change the Item ID in the Slots section directly.

Code Summary

As an Inventory Source, the following functions are available:

GetResourcePath() – Gets the resource path for the specified item.

EquipItem() – Equips the specified item in the specified slot.

GetEquippedItemID() – Gets the item ID of the item in the specified slot.

ClearSlot() – Removes any item from the specified slot.

These functions are used by the Archery motions to determine what bow and arrow should be created and to equip/unequip the bow.

Again, you can use any inventory solution you want. However, that solution needs to implement the `IInventorySource` interface.

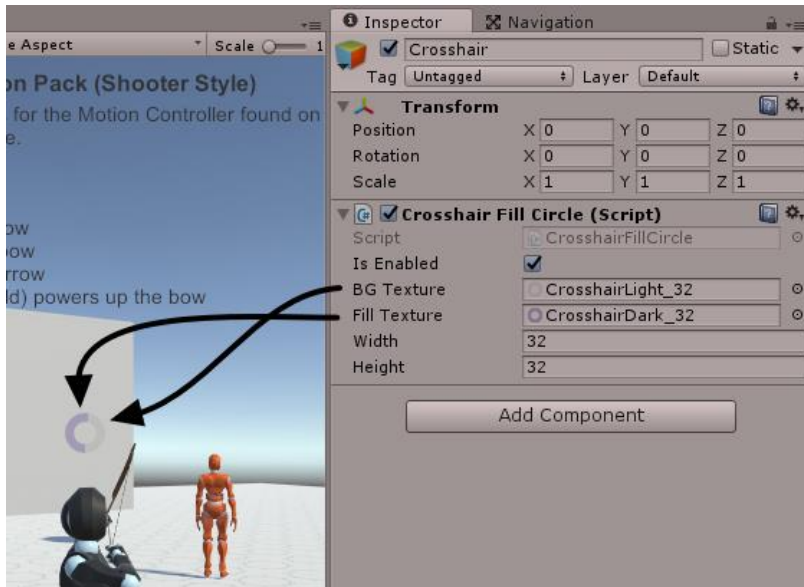


Archery Motion Pack

10/05/2018

Reticle

Adding the reticle to the scene is simply a matter of using the included Crosshair Fill Circle script that I've included.



The BG Texture is the background image for the reticle. To customized it, simply replace the image.

The Fill Texture is what overlays the BG Texture as the bow is being powered. Just replace this texture as well.

The bow motions will enable, disable, and update the reticle automatically.

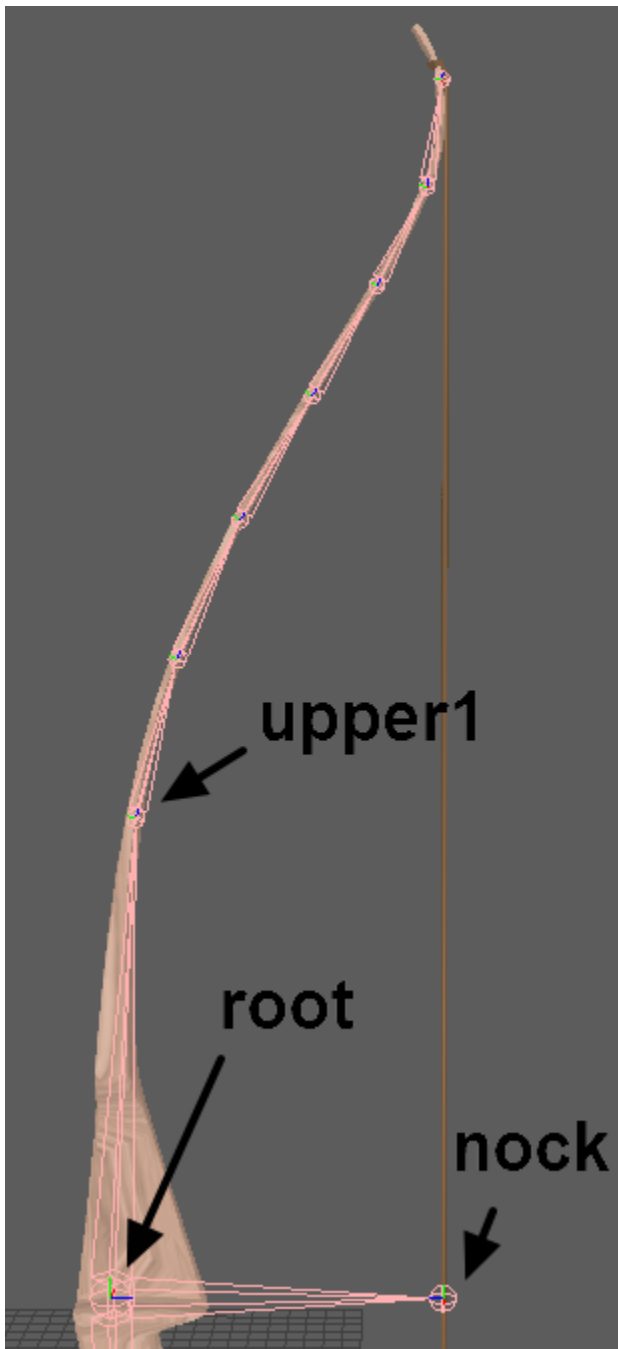


Bows

In the pack, there's a bow included that you can use in your game. You can customize the bow properties or create your own bow following the guidelines I'll go over.

The bow is made up of three parts: the model, the core, and the prefab.

Bow Model



The model is actually a skinned mesh with bones. This way, we can bend the bow as we draw it back. This bow was created, rigged, and skinned in Maya.

We don't need to animate the bow as we'll use IK to bend the joints.

The root of the skeleton is where the arrow rests.

The image to the left shows the bow in Maya and how I placed the bones. There's no specific number of bones needed... But, the more you add, the smoother the curve.

Notice a bone at the center of the string too. That's the "nock" and where I'll anchor the arrow to. So, when we pull the nock back the string and arrow goes back too.

Bone Names

There are 4 important names that need to be used:

"root" is the bone that starts the skeleton.

"nock" is the bone at the center of the string.

"upper1" is the first bone at the top of the bow we'll bend.

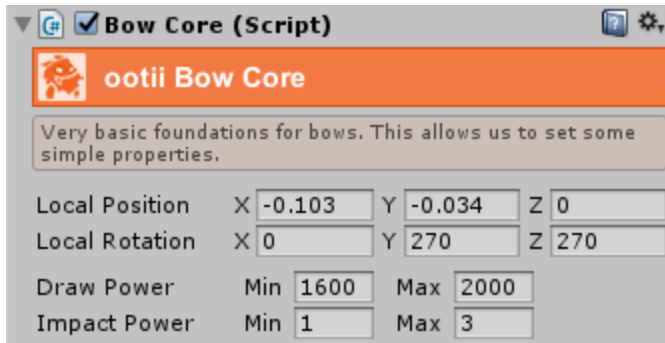
"lower1" (not shown) is the first bone at the bottom of the bow we'll bend.

The Bow Core will look for these transform names in the transform hierarchy.



Bow Core

The Bow Core is the heart-beat of the bow. It controls how the bow bends, the power, etc.



Since we don't know how the character's left hand bone will be created, we include the "**Local Position**" and "**Local Rotation**" values to compensate.

This takes some trial-and-error, but it allows you to offset the bow's position and rotation relative to the left hand bone.

If you're using MountPoints, you won't need to worry about these.

Since the bow has the ability to over-draw (or power up), the draw power and impact power are ranges.

Draw Power determines how far the bow will shoot. This value is the magnitude of the force we put on the arrow before sending it flying. So, if there is no over-draw, the image above would use a value of 1600. If there is 100% over-draw, we'd use a value of 2000.

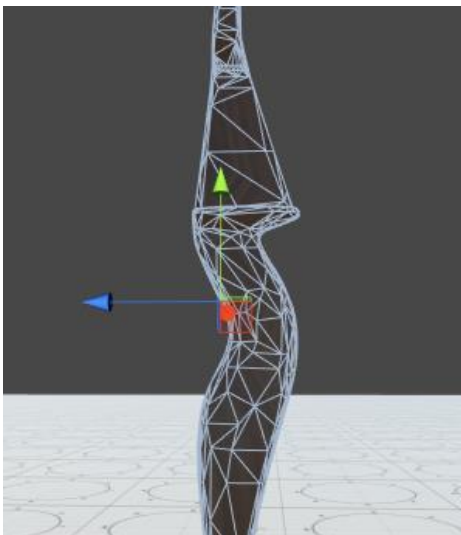
These values are used with the arrow's rigidbody's AddForce() function.:

Impact Power is a multiplier we use when the arrow impacts a rigidbody. This allows the bow itself to add extra force beyond that of the arrow. This is useful for increasing the hit strength of the arrow.

Bow Prefab

The last part is the bow prefab itself. We'll instantiate the prefab when the bow is equipped.

When building the prefab, you want to position the bow skinned mesh so that the prefab's origin lines up with the handle and faces forward. This way it's easier to put it into a character's hand.



As you probably guessed, the prefab will hold the Bow Core and any settings you modify that are specific to the prefab.

By creating multiple prefabs, you could have longbows, short bows, magic longbows that shoot further, etc.

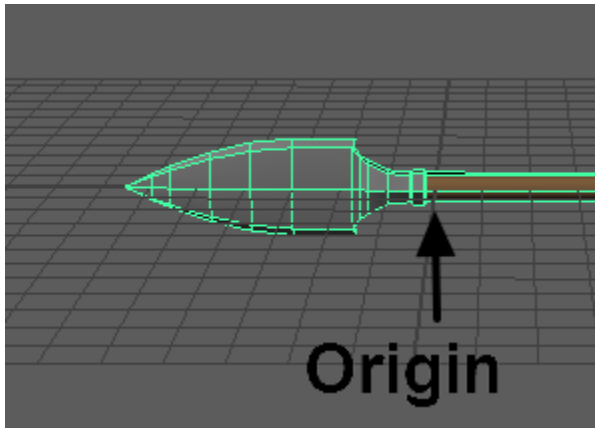


Arrow

As with the bow, I've also included an arrow. You can customize the arrow like you do the bow or simply use mine.

The arrow is also made up of three parts: the model, the core, and the prefab.

Arrow Model



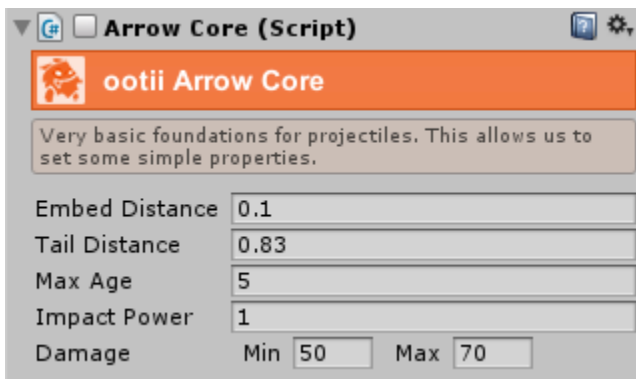
The arrow model is just a static mesh. There's really nothing special about it other than the direction and origin.

In Unity, we want the forward direction to be the forward direction of the arrow.

For the origin, I don't put it at the tip of the arrow-head, but more the back. The reason is that when we embed the arrow in a target, we want it to look like the arrow head is inside the target.

Arrow Core

Like with the Actor Core and Bow Core, the Arrow Core is the heart-beat of the arrow. It's how we detect collision and react when the collision occurs.



Embed Distance is a little extra amount we push the arrow into the target when it collides. I do this since most characters have colliders that are away from the body mesh. This makes it look like the arrow is really inside the head or chest.

As we draw the arrow back, it's the **Tail Distance** that we use to determine how far from the origin that the string should be. Think of it like an offset for pulling the arrow back.

Once the arrow impacts the target, the **Max Age** determines how long the arrow will live for (in seconds). If set to 0, the arrow won't destroy itself.

Impact Power is the base magnitude of the force that will be applied to the target if the target is a rigidbody. This is what causes the target to move under Unity's physics. The final force magnitude will be this value multiplied by the bow's Impact Power.

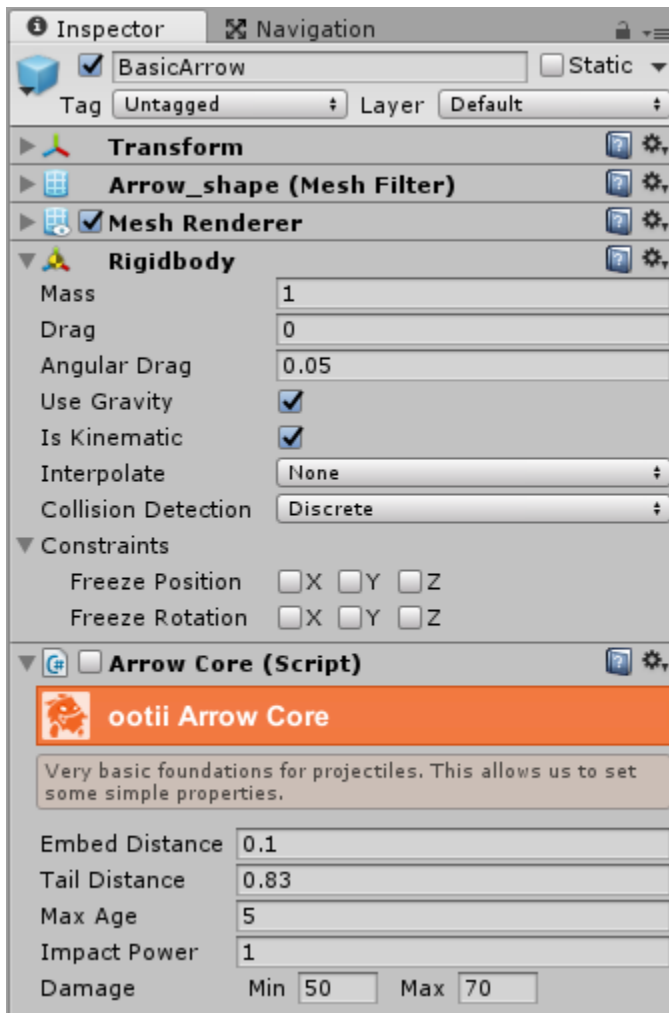


Damage is a range as it uses the bow's over-draw value. Similar to the bow's properties, when not over-drawn, the minimum value is the damage. If the bow is 100% over-drawn, the maximum value is the damage applied.

In both cases, damage is only applied if the target has an IActorCore component.

Arrow Prefab

The arrow prefab needs to include a Rigidbody.



The Rigidbody is actually what propels the arrow forward. When the bow shoots, it will call the Rigidbody's AddForce() method and send the arrow on its way.

When building your own prefab, ensure the Arrow Core component is disabled. I will enable it when we shoot the arrow.

Colliders

Notice the arrow prefab doesn't have any colliders.

Through lots of testing, I found that colliders don't work well with an arrow. Instead, I use a short raycast as the arrow flies.

This raycast gives more accurate and timely information about what is hit, the angle, etc.



Motions

Earlier, I gave a quick description of each motion. Here I'll go into more detail about the motion properties.

Basic Idle

Simple idle animation with bow in hand.

Rotate With Input – Determines if the character rotates as the mouse (right stick) moves.

Rotate With Camera – Determines if the character rotates to match the direction of the camera.

Rotation Speed – Degrees per second to rotate.

Use Pivot Animations – Determines if pivot animations are used while rotating. Since the rotation can be in increments smaller than 90 degrees, the animations may be over-kill. You can use them as desired.

Basic Item Equip

Unsheathe animation that determines the right bow to equip and then equips it.

Action Alias – Input entry that equips the bow

Bow Resource Path – If you don't want to use an inventory source, the resource path to the bow prefab to instantiate.

Inventory Source – Inventory source we'll get the bow resource path from. If left empty, we'll check if the inventory source is on this character.

Equip Slot ID – Slot ID of the inventory source where we'll place the bow.

Bow Ready Slot ID – Slot ID of the inventory source that tells us which bow should be equipped.

Basic Item Store

Puts the bow away and returns to the normal (non-bow) idle.

Action Alias – Input entry that un-equips the bow

Inventory Source – Inventory source we'll use to say we've un-equipped the bow.

Equip Slot ID – Slot ID of the inventory source we will clear.

Basic Walk Run Pivot

Movement motion with bow in hand. This movement style is more "Adventure" style and if used with an orbiting camera, allows the character to walk towards the camera.

Default to Run – Determines if the character runs or walks by default (Action Alias inverts it).

Run Action Alias – Used to trigger the character to run (or walk) based on the Default to Run option



Walk Speed – When greater than 0, overrides root-motion and creates a constant velocity (units per second).

Run Speed – When greater than 0, overrides root-motion and creates a constant velocity (units per second).

Rotation Speed – Degrees per second to rotate.

Basic Walk Run Store

Movement motion with bow in hand. This movement style is more “Shooter” style and always has the character facing forward. It supports strafing.

Similar to the standard WalkRun... motions, you would only have WalkRunPivot or WalkRunStrafe enable. Not both.

Default to Run – Determines if the character runs or walks by default (Action Alias inverts it).

Run Action Alias – Used to trigger the character to run (or walk) based on the Default to Run option.

Walk Speed – When greater than 0, overrides root-motion and creates a constant velocity (units per second).

Run Speed – When greater than 0, overrides root-motion and creates a constant velocity (units per second).

Rotate With Input – Determines if the character rotates as the mouse (right stick) moves.

Rotate With Camera – Determines if the character rotates to match the direction of the camera.

Rotation Speed – Degrees per second to rotate.

Stop Delay – Small delay we add before stopping in order to help with the Mecanim blend tree.

Basic Ranged Attack

Motion used to fire the arrow.

Attack Action Alias – Used to trigger the character to shoot the bow.

Rotate With Input – Determines if the character rotates as the mouse (right stick) moves.

Rotate With Camera – Determines if the character rotates to match the direction of the camera.

Rotation Speed – Degrees per second to rotate.

Horizontal Angle – IK value to offset the yaw (left/right rotation) of the left arm. This is used to help aim the bow.

Vertical Angle – IK value to offset the pitch (up/down rotation) of the left arm. This is used to help aim the bow.

Quick Horizontal Angle – IK value to offset the yaw (left/right rotation) of the left arm when doing a quick shot. This is used to help aim the bow.



Archery Motion Pack

10/05/2018

Quick Vertical Angle – IK value to offset the pitch (up/down rotation) of the left arm when doing a quick shot. This is used to help aim the bow.

Arrow Resource Path – If you don't want to use an inventory source, the resource path to the arrow prefab to instantiate.

Inventory Source – Inventory source we'll get the bow resource path from. If left empty, we'll check if the inventory source is on this character.

Arrow Ready Slot ID – Slot ID of the inventory source that tells us which arrow should be equipped.

Release From Camera – By default, the arrow will shoot from its current position in the bow. However, some animation may make the arrow shoot off on a small angle. If you want to have absolute control over the arrow, shoot it from the camera instead. This is especially useful for over-the-shoulder modes.

Release Time – When in the animation to release the arrow.

Default Draw Force – If no value is set from the bow, the default force to shoot the arrow with.

Basic Damaged

Animation used when the actor is hit by an arrow.

Basic Death

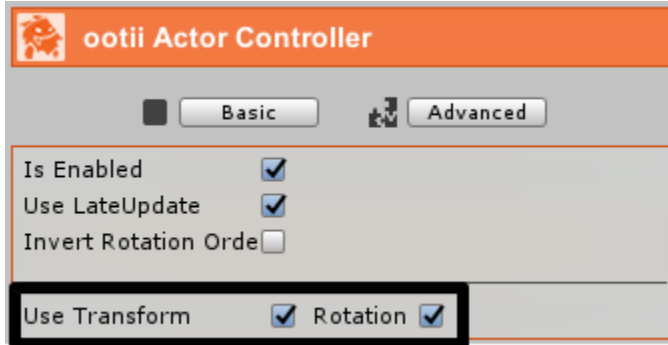
Animation used when the actor is killed by an arrow.



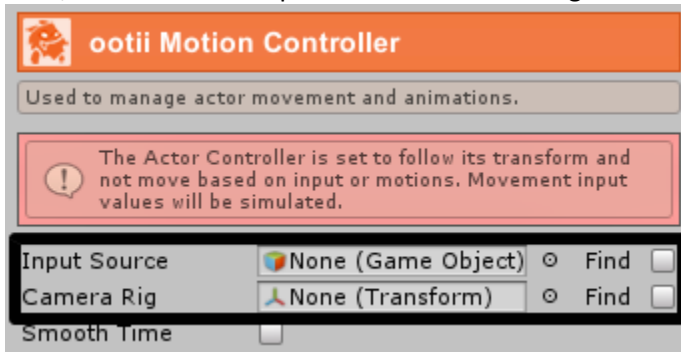
NPCs

The Archery Motion Packs supports NPCs. To do this, you're going to set your NPC up just like your PC. Then, you'll make a couple of additional changes.

First, ensure the Actor Controller is set to use the transform to determine movement:



Next, ensure that no Input Source or Camera Rig is set or found automatically:



Finally, the NPC requires a "Combatant" component added along with the other components for the NPCs:



Code

With those changes, the NPC is ready to be controlled. You can use Node Canvas, Behavior Designer, or any of the other AI drivers... or you can use code to control your NPC.



Archery Motion Pack

10/05/2018

In the asset, I include a demo and some NPC code to look at and use.

- Demo: Assets\ootii_Demos\MotionControllerPacks\Archery\Demos\Scenes\demo_NPC_Shooter
- AI Code: Assets\ootii_Demos\MotionControllerPacks\Archery\Demos\Scenes\AMP_NPC_Controller

Besides the normal movement logic, the real code that allows the NPC to fire an arrow is this:

```
CombatMessage lMessage = CombatMessage.Allocate();  
lMessage.ID = CombatMessage.MSG_COMBATANT_ATTACK;  
lMessage.Attacker = gameObject;  
lMessage.Defender = Target.gameObject;  
  
mMotionController.SendMessage(lMessage);  
CombatMessage.Release(lMessage);
```

When the NPC archer has the bow equipped, sending this “ATTACK” message will have him fire towards the Defender.

You can also set a direction to fire instead of the defender. Simply set the “HitDirection” vector3 variable in the message instead of the defender.



Frequently Asked Questions

Below is a list of how-to and responses to questions that I get (or expect to get). Hopefully they help provide some quick insight and tutorials.

Pre-Purchase

Can I use this with NPCs?

Yes. See the section titled [NPCs](#).

Can I use this with Inventory Pro, Rewired, or other non-ootii assets?

Yes. I've built this (and the MC) to be very modular. You can simply replace pieces as needed.

Movement

How do I make the movement speed faster?

If you want to use the root-motion, you can increase the speed of the animations in the Animator (specifically BasicWalkRunPivot, BasicWalkRunStrafe, and BasicWalkRunTarget).

If you don't want to use root-motion, just change the "Walk Speed" and "Run Speed" properties of the motions. However, the more you move away from the animation's speed... the more your character will look like they are ice skating.

When aiming, my character won't rotate left or right...

This can happen when using the Adventure Camera Rig. When in aiming/targeting mode, the camera won't rotate left or right. Instead, the character does the left/right rotation and the camera rotates up/down. I do this so you can have different characters that rotate at different speeds.

In this case, go to the "Basic Ranged Attack" motions and uncheck "Rotate With Camera". Then check "Rotate With Input". Now, the character will drive the camera's rotation.

Do my NPCs have to use the MC to react to arrow hits?

No.

When the arrow hits, it looks to see if the target has an "IActorCore" based component attached to it.

In my examples, "ActorCore" is used and it tells the MC to play the damaged or death animation.

You can totally change that by creating a custom MonoBehaviour that implements IActorCore. Then in the OnDamaged() and OnDeath() functions, you can do anything you want. You don't need the Actor Controller or Motion Controller at all.



Archery Motion Pack

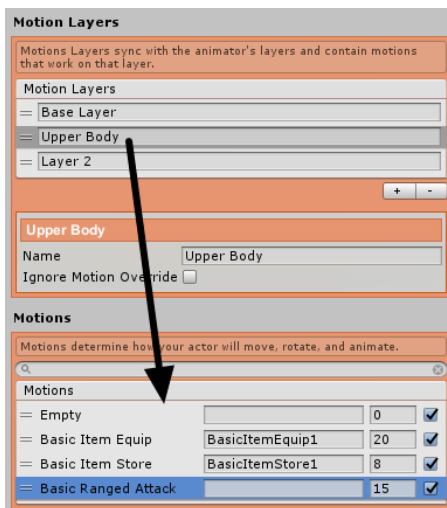
10/05/2018

How can I have my character move and shoot at the same time?

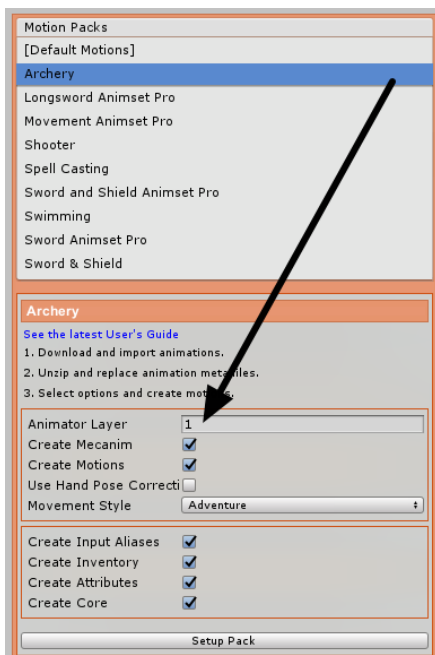
It's important to remember that animation controllers are built based on the animations and not the other way around. So, if the animations don't support a feature, an animation controller won't either.

Moving and shooting at the same time requires full-body animations that move and shoot or two animations that are capable of blending together (one for lower body movement and one for upper body shooting).

To enable this, we need to move our "Basic Ranged Attack" to the second motion layer as well as a version of "Basic Item Equip" and "Basic Item Store". The last two allow us to equip and store the bow while walking as well.

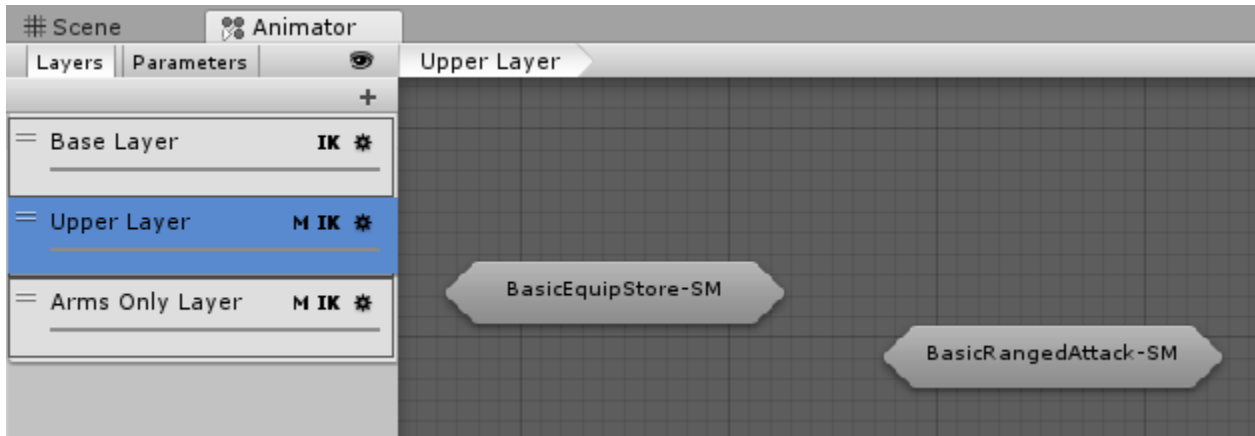


I can setup this flow automatically in the "Packs" view when you change the "Animator Layer" to "1" and press the "Setup Pack" button.





When you press the “Setup Pack” button, I’ll create the Mecanim sub-state machines on the second animator layer (index 1) and add the animations.



Just ensure that if you had the “Basic Ranged Attack” motion on the first Motion Controller layer that you disable it.

Camera

Can I use the Camera Controller?

Absolutely.

Some setting that make this work really well:

1. On the ACR, set the Targeting Alias to match the BasicWalkRunStrafe’s Activation alias (“Camera Aim”)
2. On the ACR, uncheck Targeting As Toggle
3. On the AMP, check the BasicWalkRunStrafe’s Rotate With Input
4. On the AMP, uncheck the BasicWalkRunStrafe’s Rotate With Camera
5. On the AMP, change the BasicWalkRunStrafe’s Rotation Speed to 180
6. On the AMP, check the BasicRangedAttack’s Rotate with Input
7. On the AMP, uncheck the BasicRangedAttack’s Rotate with Camera
8. On the AMP, check the BasicRangedAttack’s Release From Camera

Can I use my own camera system?

Absolutely.

This asset doesn’t control the camera at all. It does use the camera’s direction to help aim and move, but that is simply a transform that I access.

You’ll have to play around with the motion properties to get the kind of behavior that you’re looking for.



How do I change the camera position when aiming?

That's really a function of your camera system. Let's say aiming requires the right-mouse-button to be held (which it does be default), thank your camera system would want to move or swap the camera when the right-mouse-button is held.

If you do change the view of your camera, you may find that using the "Release from Camera" option on the Bow – Basic Attacks motion is a useful option.

How do I keep the crosshairs always on?

On the "Crosshair Fill Circle" component in your scene, check the "Is Enabled" checkbox.

Then, on both the "Bow – Walk Run Target" and "Bow – Basic Attack" motions, uncheck the "Manage Crosshair" checkbox.

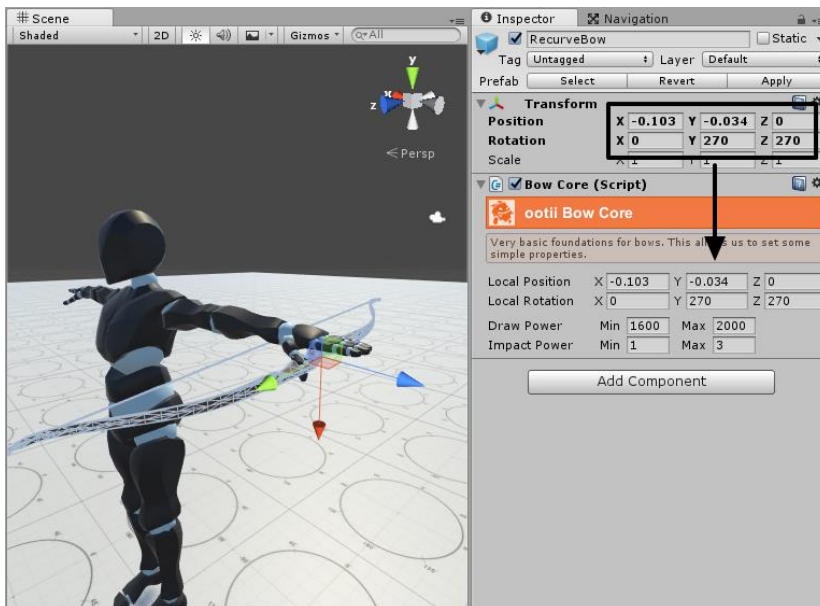
Bow

How do I make the bow shoot shorter or further?

Change the Draw Power on the Bow Core that your bow's prefab contains.

My bow isn't place in the left hand correctly. How do I fix this?

You will use the Bow Core's Local Position and Local Rotation properties to offset the bow. This is a little bit of trial-and-error. So, I'd put my character in the scene, parent the bow to the left hand, and use the transform values to get what you want. Those values then go into Local Position and Local Rotation:



How do I make a new bow?

Go [here](#).



Why isn't my custom bow bending?

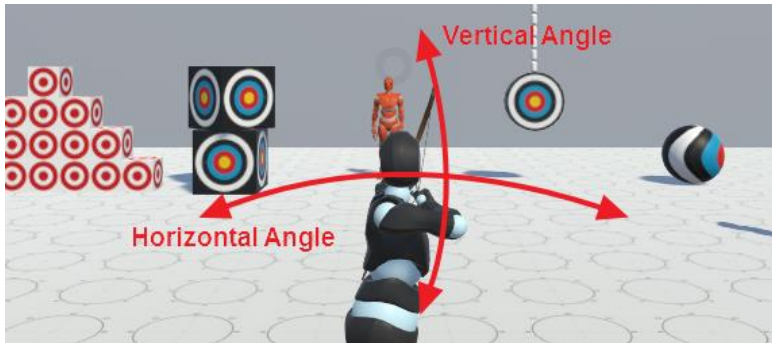
Ensure you include the right bone names ("upper1" and "lower1").

When I shoot, the arrow goes off to the side. How can I fix this?

There's a couple of ways you can do this.

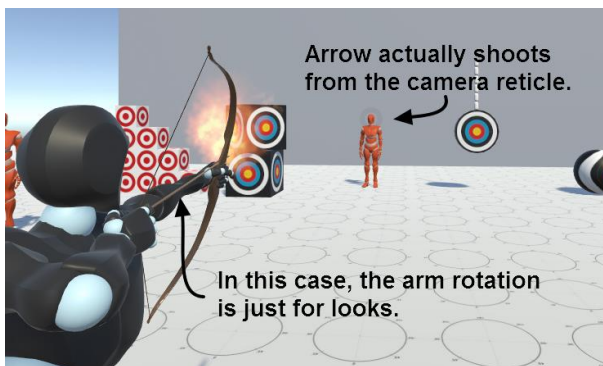
1. You can tweak the placement of the bow in the left hand. See the FAQ entry above for placing the bow. If you're only using a single character, this may be a simple option.
2. Another simple option is to set the "**Horizontal Angle**" and "**Vertical Angle**" on the "Bow – Walk Run Target" and "Bow – Basic Attack" motions.

These values rotate the left shoulder so that bow aiming can be tweaked. By using positive and negative values, you can change how the left arm aims for each individual character.



3. If you're using more of a first-person kind of view. The best thing to do is to actually shoot the arrow from the camera instead of the bow. Use the "**Release from camera**" option on the "Bow – Basic Attack" motion.

Players won't notice the difference and aiming will be based on the center of the camera. Then, you can use the properties I mention in option #2 to simply move the arm for looks.

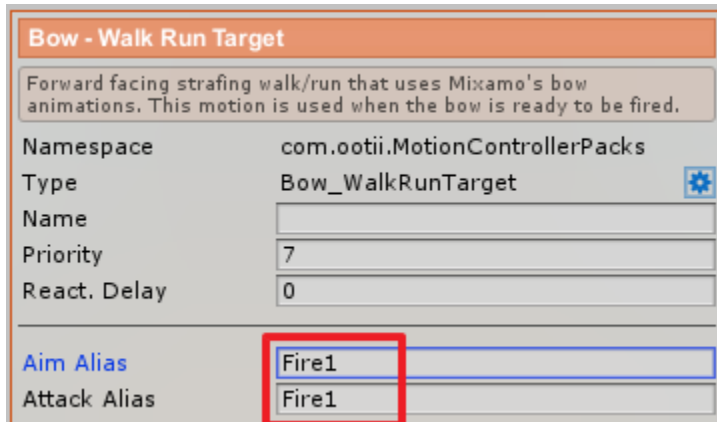


To help with testing the direction of the bow, check the "Bow – Walk Run Target" motion's "Show Debug" option. That will shoot a ray in the direction of the arrow. This way, you'll see where the arrow will head.



How can I use the left-mouse-button for aiming and shooting?

In both the “Bow – Walk Run Target” and “Bow – Basic Attack” motions, you want to set the “Aim Alias” and “Attack Alias” properties to the same value.



It can get a bit tricky as the code needs to know if you’re trying to aim or fire. So, I suggest that you use different keys for aiming and firing. However, you don’t have to.

Arrow

How do I make the arrow do more damage?

Change the Damage properties that are on your arrow’s prefab.

How do I make a new arrow?

Go [here](#).

Why isn’t my custom arrow aligned on the bow correctly?

1. Make sure your arrow faces along the forward direction.
2. Make sure your Arrow Core’s Tail Distance property accurately represents the distance between the arrow’s origin and it’s back.

How do I create fire arrows?

This is really a matter of putting a particle effect on the arrow’s prefab. As the arrow moves, the particle effect source will as well.

I’ve actually included a “FireArrow” prefab to show this.



Archery Motion Pack

10/05/2018



If you want to get more advanced, you could create a new “FireArrowCore” class that inherits from “ArrowCore”.

In this class, you can override the OnImpact() function to enable particle effects, add explosive forces, add extra damage, etc.

Just remember to call base.OnImpact() so the rest of the logic continues.



Inventory

How do I create an Inventory Source for Inventory Pro?

The process is the same for any inventory asset, but I'll write these steps for Inventory Pro.

1. Create a new class and call it "InventoryProSource".
2. Use this code as a template and put it in your new class:

```
using UnityEngine;

namespace com.ootii.actors.Inventory
{
    /// <summary>
    /// Inventory Pro Inventory Source for use with ootii.
    /// </summary>
    public class InventoryProSource : MonoBehaviour, IInventorySource
    {
        /// <summary>
        /// Some motions will use this to determine if they should test
        /// for activation or allow the inventory source to drive activation.
        /// </summary>
        public virtual bool AllowMotionSelfActivation
        {
            get { return true; }
        }

        /// <summary>
        /// Instantiates the specified item and equips it. We return the instantiated item.
        /// </summary>
        /// <param name="rItemID">String representing the name or ID of the item to equip</param>
        /// <param name="rSlotID">String representing the name or ID of the slot to equip</param>
        /// <param name="rResourcePath">Alternate resource path to override the ItemID's</param>
        /// <returns>GameObject that is the instance or null if it could not be created</returns>
        public virtual GameObject EquipItem(string rItemID, string rSlotID, string rResourcePath = "")
        {
            // Add Inventory Pro specific code;

            return null;
        }

        /// <summary>
        /// Instantiates the specified item and equips it. We return the instantiated item.
        /// </summary>
        /// <param name="rSlotID">String representing the name or ID of the slot to clear</param>
        public virtual void StoreItem(string rSlotID)
        {
            // Add Inventory Pro specific code;
        }

        /// <summary>
        /// Retrieves the item id for the item that is in the specified slot. If no item is slotted, returns an
        empty string.
        /// </summary>
        /// <param name="rSlotID">String representing the name or ID of the slot we're checking</param>
        /// <returns>ID of the item that is in the slot or the empty string</returns>
        public virtual string GetItemID(string rSlotID)
        {
            string lItemID = "";

            // Add Inventory Pro specific code here;

            return lItemID;
        }
    }
}
```



Archery Motion Pack

10/05/2018

```

    /// <summary>
    /// Retrieves a specific item's property value.
    /// </summary>
    /// <typeparam name="T">Type of property being retrieved</typeparam>
    /// <param name="rItemID">String representing the name or ID of the item whose property we want.</param>
    /// <param name="rPropertyID">String representing the name or ID of the property whose value we
want.</param>
    /// <returns>Value of the property or the type's default</returns>
    public virtual T GetItemPropertyValue<T>(string rItemID, string rPropertyID)
    {
        T lValue = default(T);

        string lPropertyID = rPropertyID.Replace(" ", string.Empty).ToLower();

        // Resource Path is used to grab the path to the prefab (string)
        if (lPropertyID == BasicInventory.Properties[0])
        {
        }
        // Instance is used to grab the pre-created item (GameObject)
        else if (lPropertyID == BasicInventory.Properties[1])
        {
        }

        return lValue;
    }
}

```

3. Each function needs to be customized to work with Inventory Pro. I don't know Inventory Pro, but I would expect it has the concept of items and slots. Add code that performs the needed function.

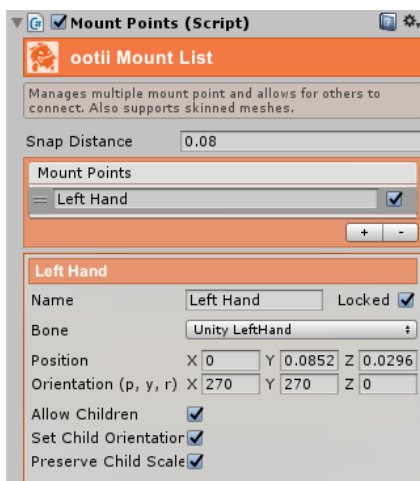
4. Once that's done, you can use Inventory Pro instead of my Basic Inventory solution.

Can I use Mount Points with the Archery Motion Pack?

Absolutely.

Since I can't assume users of AMP will have Mount Points, I have to build it like they don't. So, there's a couple of things you need to do to enable Mount Points.

1. Place Mount Points on your character and setup a mount point for the left hand:



I use the name "Left Hand".



2. Open the BasicInventory.cs file and uncomment the first line. You're changing the code from this:

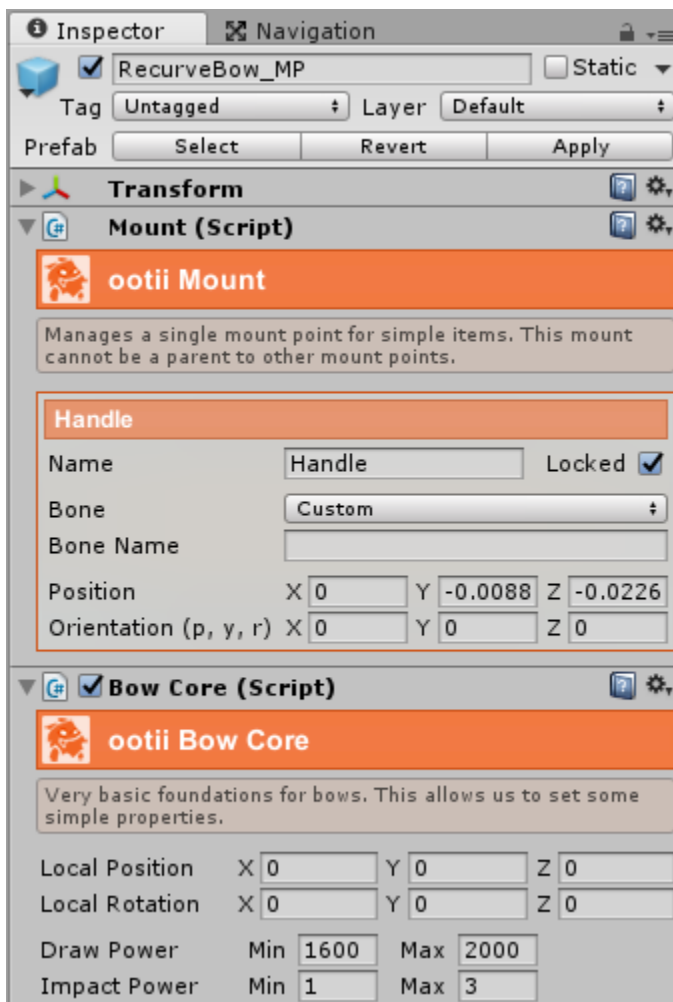
```
///#define USE_MOUNT_POINTS
```

to this:

```
#define USE_MOUNT_POINTS
```

3. When setting up your inventory, you'll want to use a bow resource that include a mount point. For example, I've included two prefabs: RecursiveBow and RecurseBow_MP. The second one includes a mount point.

I use the mount point name of "Handle".



When you do this, you won't be using the 'Local Position' and 'Local Rotation' properties of the Bow Core. Instead, you'll be using Mount Points.

This is better as it handles different bows and different characters in a more generic, flexible, and consistent way. To learn more about Mount Points, check out its [Asset Page](#) on the Unity Asset Store.

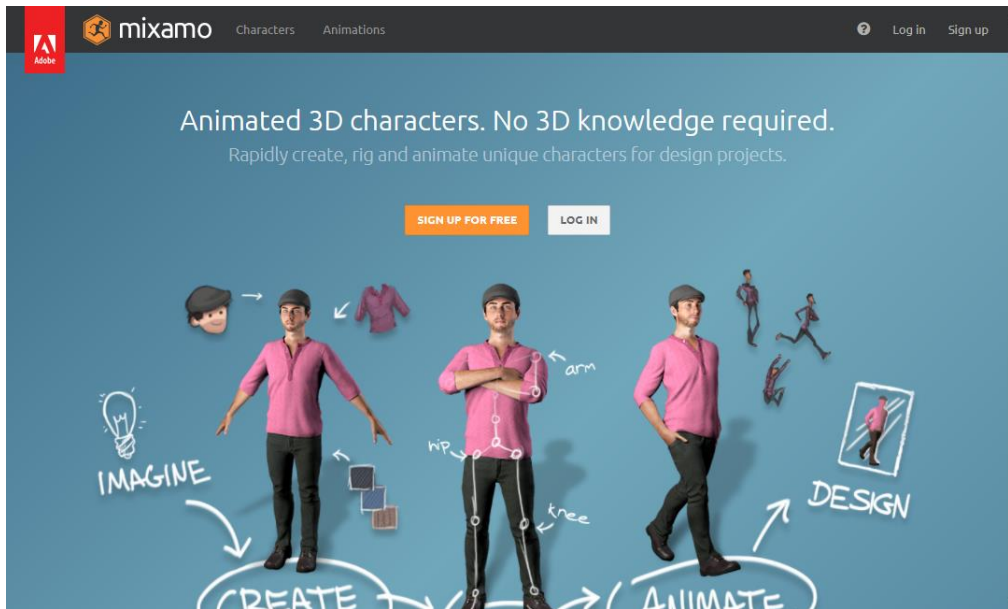


Mixamo Animation Download

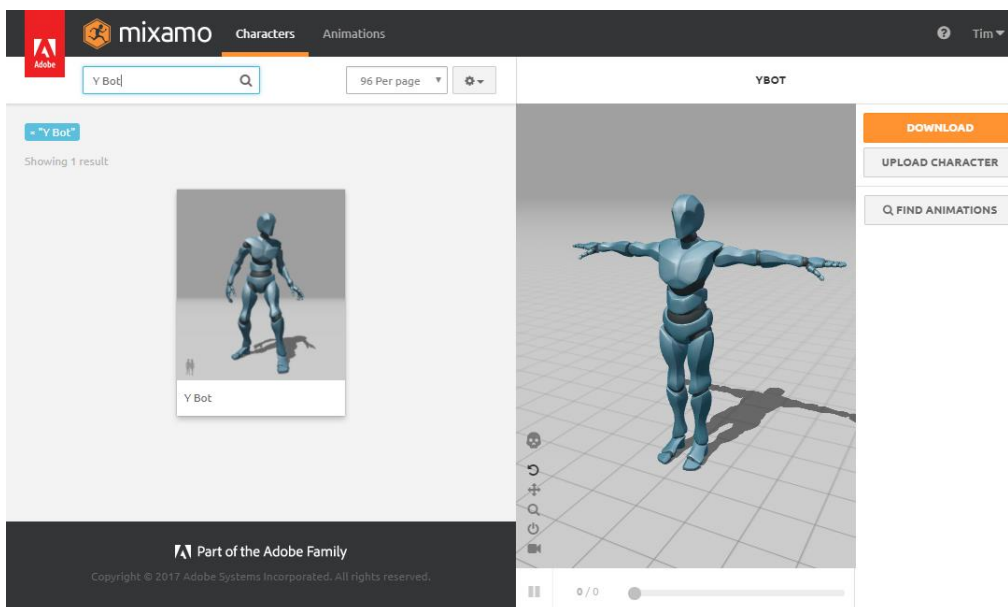
Mixamo updated their site and changed the flow. In addition, they change file names. This flow represents the updated process for getting the animations.

The following is a step-by-step approach on how to download the Mixamo animations for **Y Bot**.

1. Go to www.mixamo.com and login



2. Go to the "Characters" link at the top and search for "Y bot"

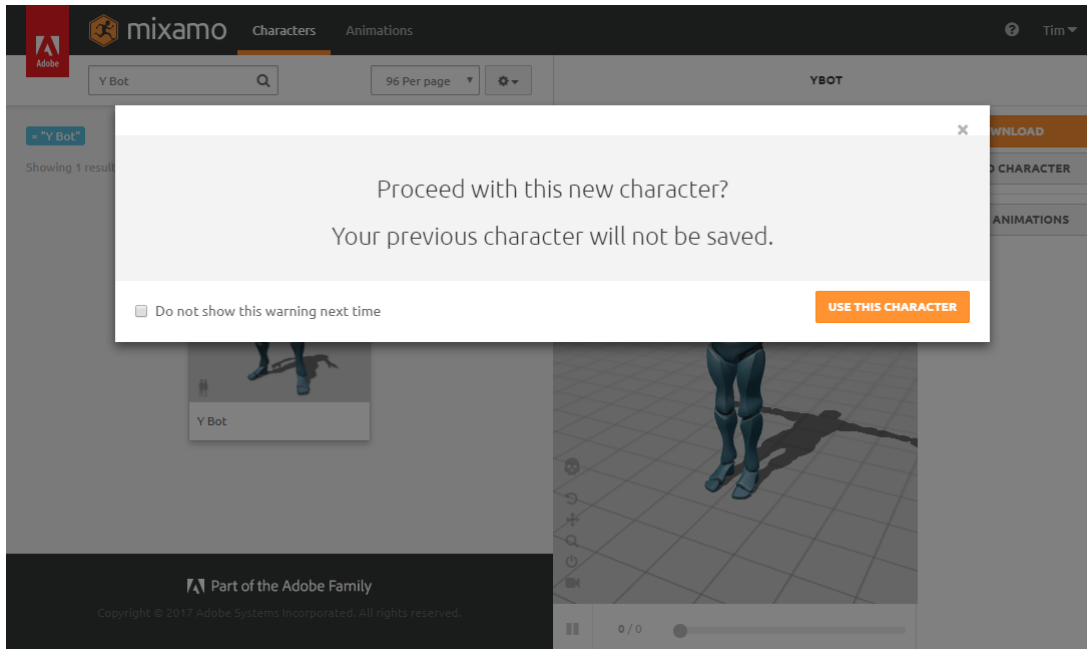




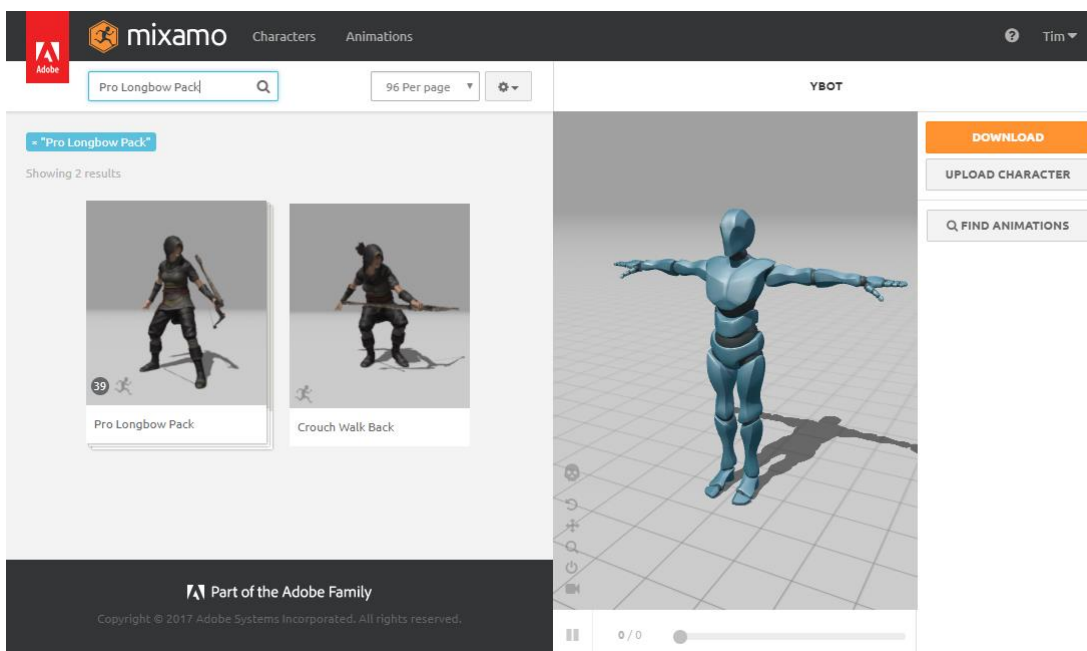
Archery Motion Pack

10/05/2018

3. Select the “Y Bot” character and press “Use This Character”. This will make the Y Bot your default character for the selected downloads.



4. Go to the “Animations” link at the top and search for “Pro Longbow Pack”

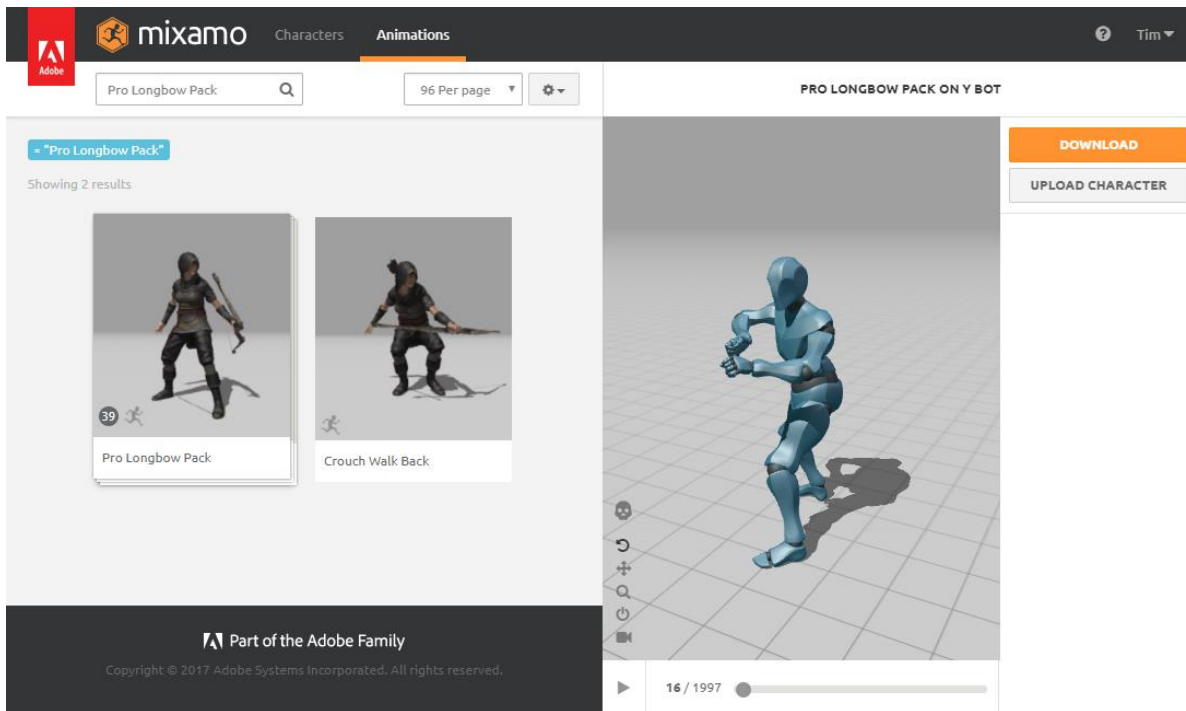




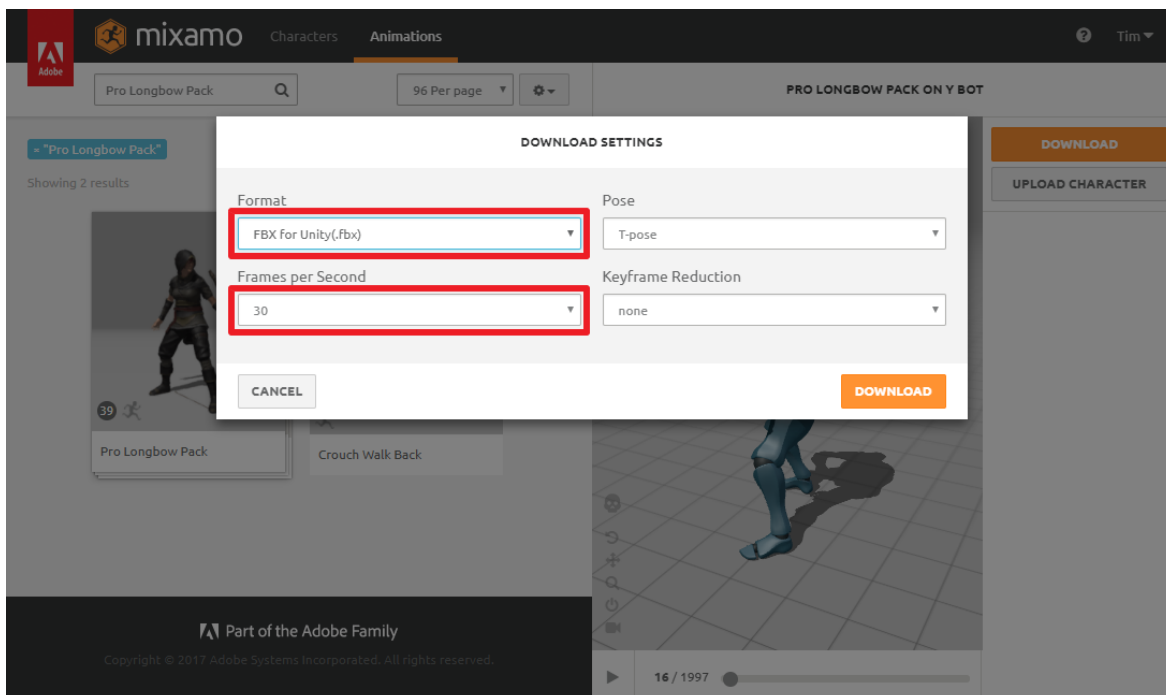
Archery Motion Pack

10/05/2018

5. Click the Pro Longbow Pack and press the “Download” button at the top right.

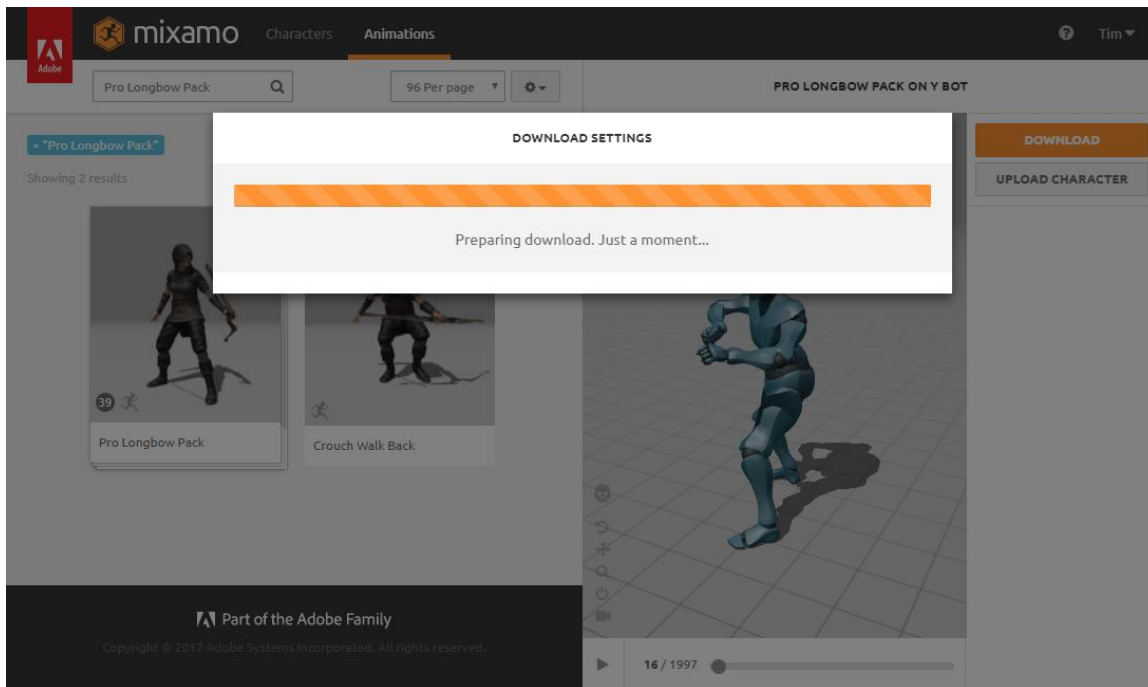


6. Select “FBX for Unity” and “30” Frames per Second. Then, press “Download”.

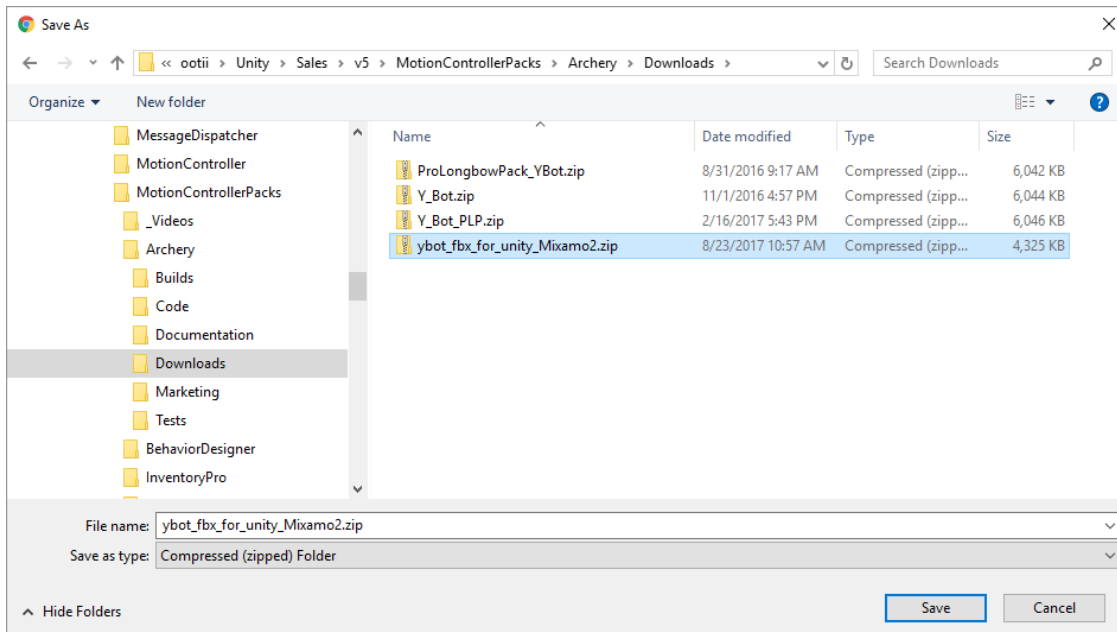




7. Wait for the processing to finish



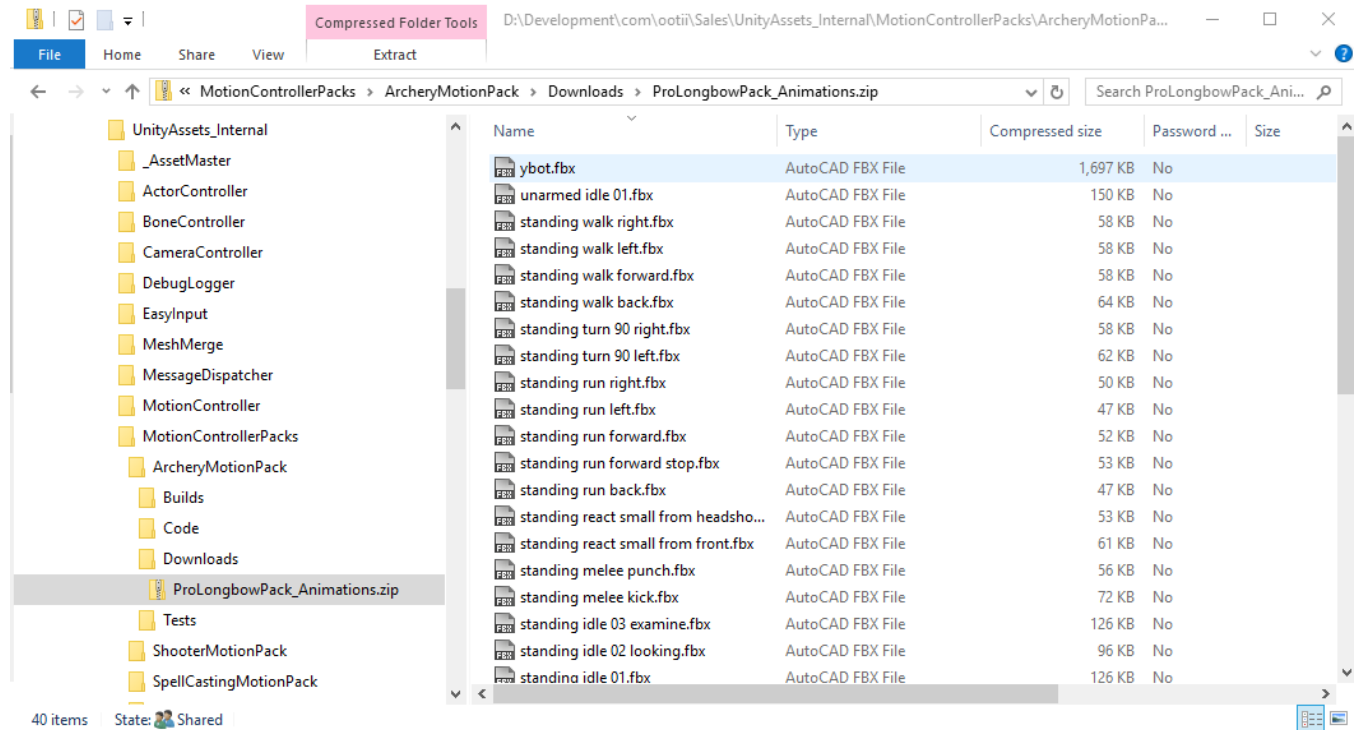
8. Select where to store the animations





12. Unzip the contents to

"...\Assets\ootii\Assets\MotionControllerPacks\Archery\Content\Animations\Mixamo"



NOTE: Your animation names may be different. For example, they may not include "Y_Bot@" or underscores. That's fine.

Once you do that, simply follow the steps from the [Quick Start](#).



Animation & Meta Files

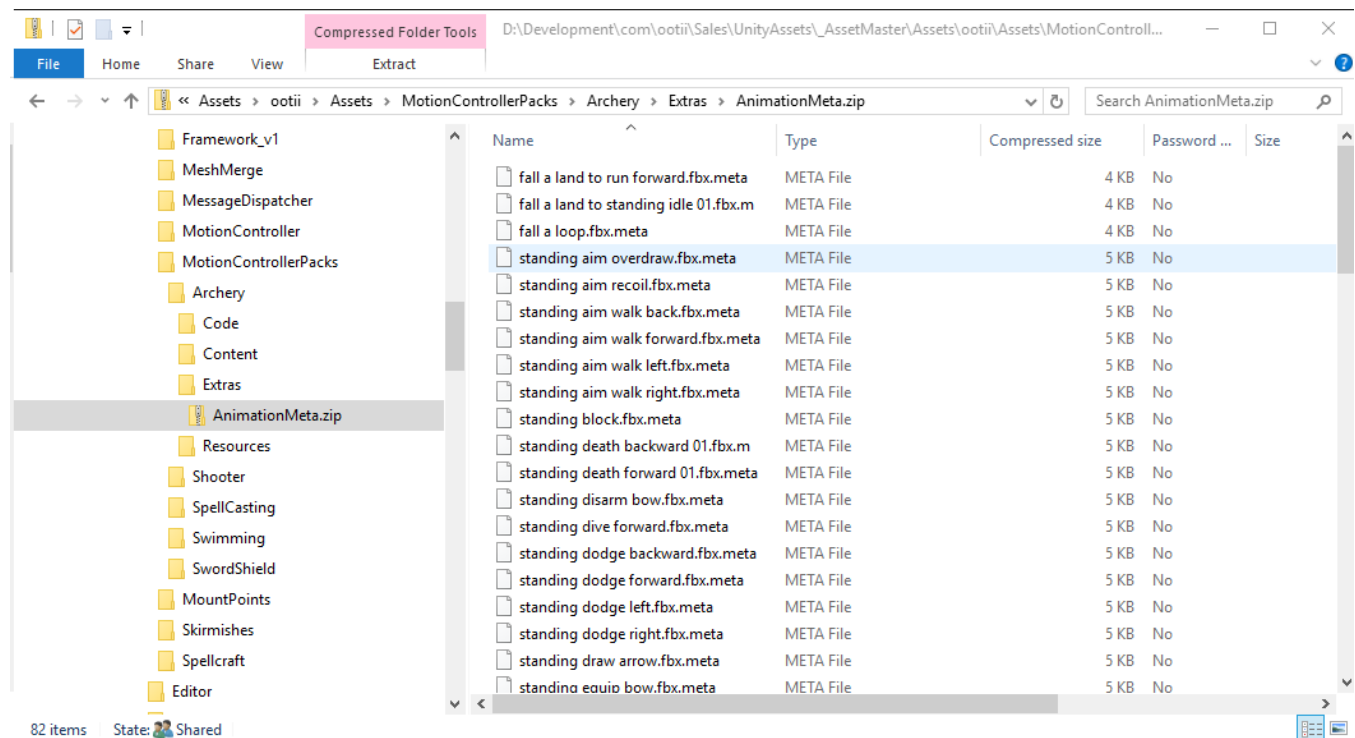
Assets\ootii\Assets\MotionControllerPacks\Archery\Content\Animations\Mixamo

When you first import the Archery Motion Pack, the Mixamo folder will only have a “Materials” folder in it (and the associated Materials.meta file).

Once you download the Mixamo animations, there will be 39 animation files + 1 character file. You’ll place them in that Mixamo folder.

If you click on Unity, it will create a meta file for each file. You don’t want those files.

Instead, you want the meta files found in the
Assets\ootii\Assets\MotionControllerPacks\Archery\Extras\AnimationMeta.zip file.



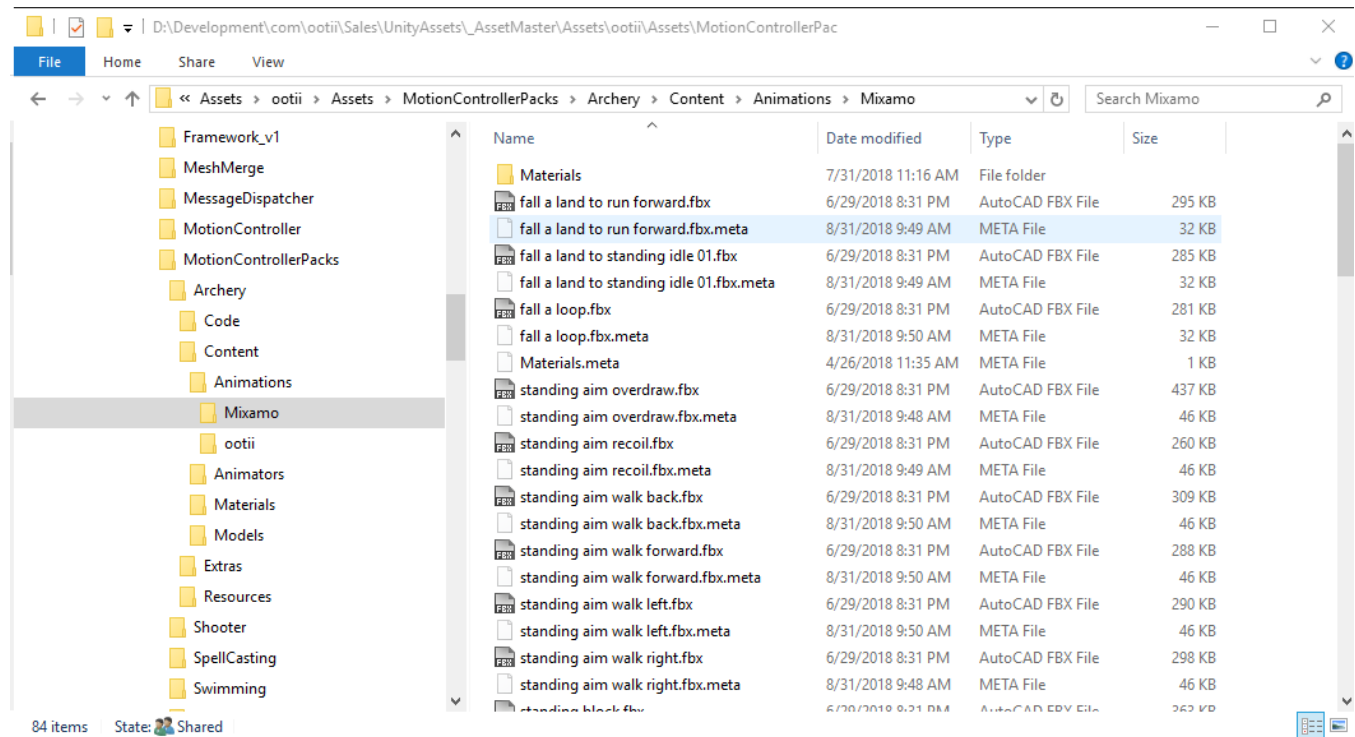
Extract, copy, and paste those 40 *.meta files into the Mixamo folder that we just put the animations. When you click on Unity again, it will reload the meta files.



Archery Motion Pack

10/05/2018

Your Mixamo folder will look like this:



Finally, you will need to close Unity and re-open it. For some reason, Unity needs to do this to update the Animator with these animations.