IMPORTANT

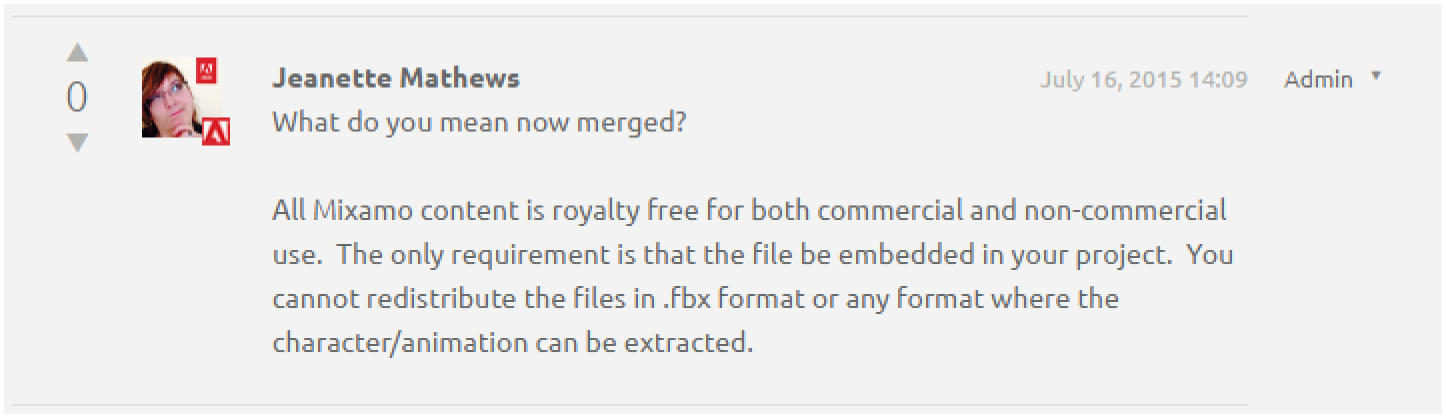
The Spell Casting Motion pack requires the following:

Motion Controller v2.50 or higher

Mixamo's free Pro Magic Pack (using Y Bot)

Importing and running without these assets will generate errors!

Why can't I include Mixamo's animations in this asset?


0

Jeanette Mathews July 16, 2015 14:09 Admin

What do you mean now merged?

All Mixamo content is royalty free for both commercial and non-commercial use. The only requirement is that the file be embedded in your project. You cannot redistribute the files in .fbx format or any format where the character/animation can be extracted.

https://community.mixamo.com/hc/en-us/community/posts/204433758-commercial-use-

Mixamo and Adobe are kind enough to give us the animations for free. However, I cannot redistribute the FBX or I would be in violation of their licensing agreement.

Demo Quick Start

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

1. Start a new Unity 5.5.0f3 or higher Project.

2. Download and import the Motion Controller asset.

3. Download and import the Spell Casting Motion Pack asset.

4. Download Mixamo's Pro Magic Pack using "Y Bot" (see this flow).

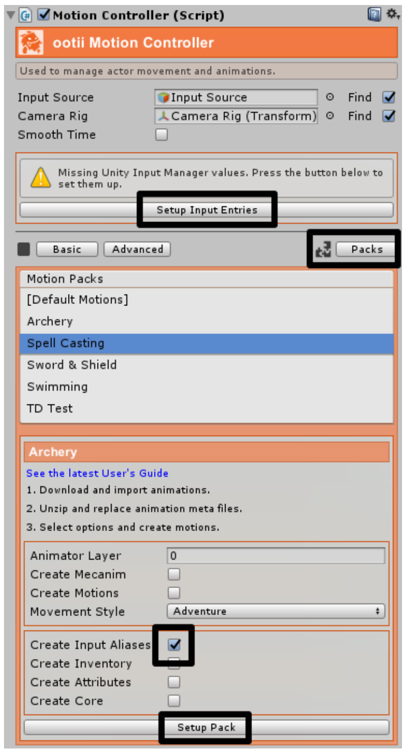
5. Unzip Pro Magic animations to project's Mixamo folder.
...\Assets\ootii\MotionControllerPacks\SpellCasting\Content\Animations\Mixamo

6. Unzip AnimationMeta.zip from pack's "Extras" folder to project's Mixamo folder (see steps).
...\Assets\ootii\MotionControllerPacks\SpellCasting\Content\Animations\Mixamo

7. Let Unity import the animations and meta data. Then, close and re-open Unity.

8. Open the "demo_SpellCasting" demo scene.
...\Assets\ootii_Demos\MotionControllerPacks\SpellCasting\Scenes\

9. Setup input entries on the Packs tab.



Motion Controller (Script)

ootii Motion Controller

Used to manage actor movement and animations.

Input Source	Input Source	Find ✓
Camera Rig	Camera Rig (Transform)	Find ✓
Smooth Time	☐	

Missing Unity Input Manager values. Press the button below to set them up.

Motion Packs
- [Default Motions]
- Archery
- Spell Casting
- Sword & Shield
- Swimming
- TD Test

Archery

See the latest User's Guide
1. Download and import animations.
2. Unzip and replace animation meta files.
3. Select options and create motions.

Animator Layer	0
Create Mecanim	☐
Create Motions	☐
Movement Style	Adventure
Create Input Aliases	✓
Create Inventory	☐
Create Attributes	☐
Create Core	☐

10. Press play.

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 (in Unity 5.5.0f3 or higher).

2. Download and import the Spell Casting Motion Pack asset.

3. Download Mixamo's Pro Magic Pack using "Y Bot" (see this flow).

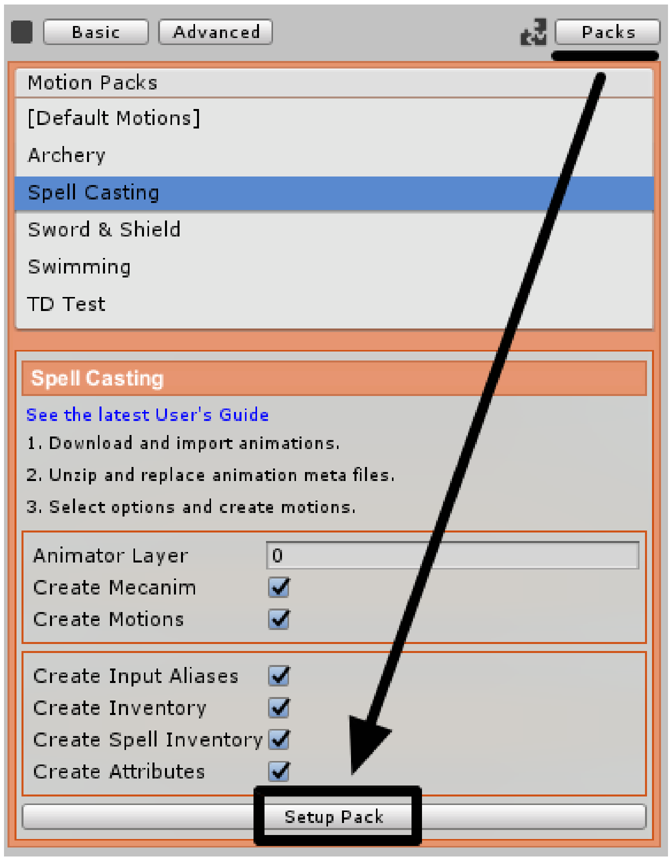
4. Unzip Pro Magic animations to project's Mixamo folder.
...\Assets\ootii\MotionControllerPacks\SpellCasting\Content\Animations\Mixamo

5. Unzip AnimationMeta.zip from pack's "Extras" folder to project's Mixamo folder (see steps).
...\Assets\ootii\MotionControllerPacks\SpellCasting\Content\Animations\Mixamo

6. Let Unity import the animations and meta data. Then, close and re-open Unity.

7. Open the scene.

8. Setup the motion pack on the Packs view.



1. Download and import animations.
2. Unzip and replace animation meta files.
3. Select options and create motions.

Animator Layer: 0
Create Mecanim: ✓
Create Motions: ✓
Create Input Aliases: ✓
Create Inventory: ✓
Create Spell Inventory: ✓
Create Attributes: ✓

9. Motions are added to the 'Advanced' view.

Foreword

Thank you for purchasing the Spell Casting 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:

Tim
tim@ootii.com

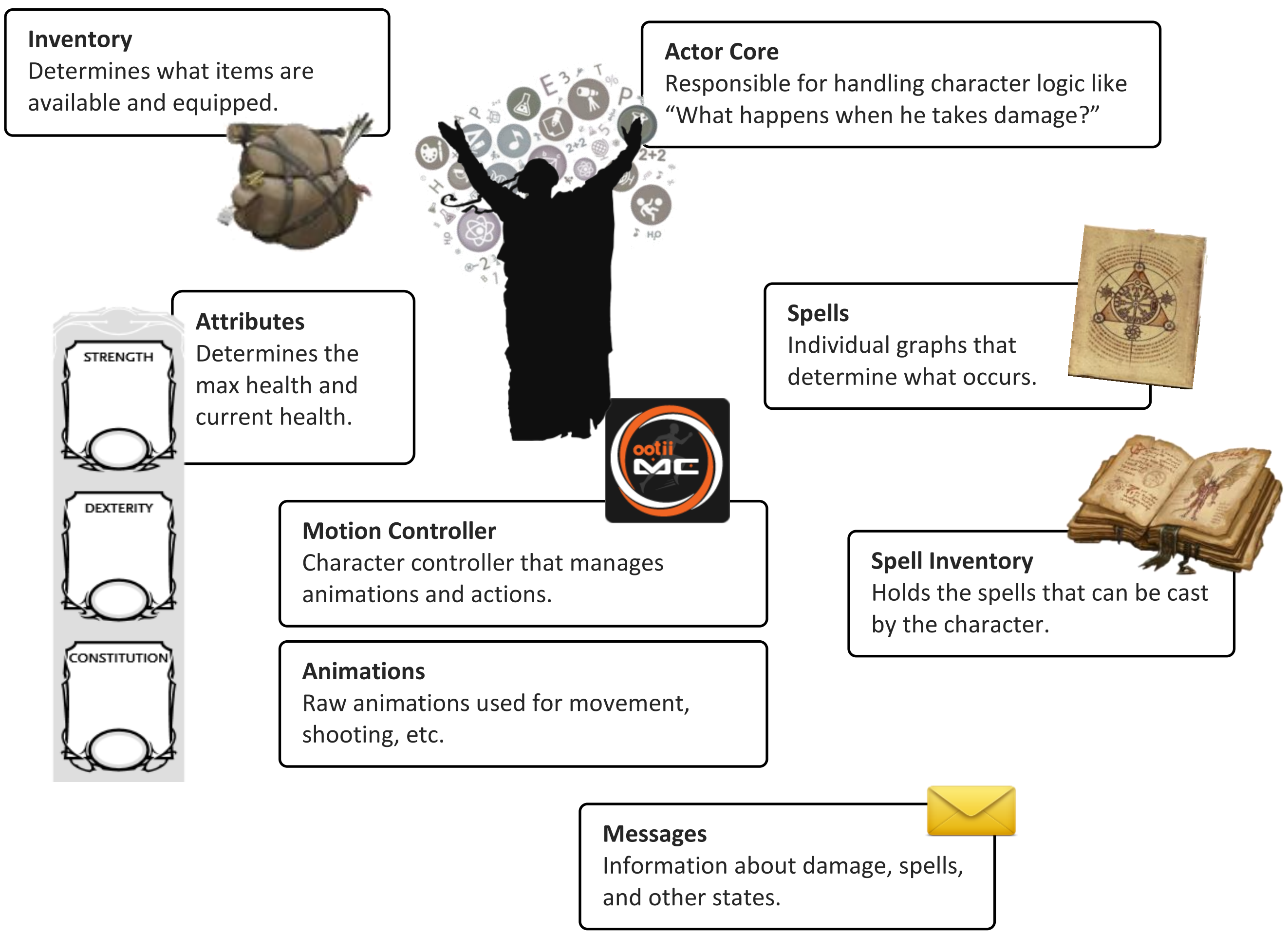
Overview

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 Magic Pack animations. When those are combined with this asset, your character gains the ability to cast spells, react to spells, etc.

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 Attributes component or replace it with something else. In addition, you can use my special effects or your own. There's a lot of things you can tweak to get exactly what you want.

Key Components

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



Inventory
Determines what items are available and equipped.

Actor Core
Responsible for handling character logic like "What happens when he takes damage?"

Attributes
Determines the max health and current health.

Spells
Individual graphs that determine what occurs.

Motion Controller
Character controller that manages animations and actions.

Spell Inventory
Holds the spells that can be cast by the character.

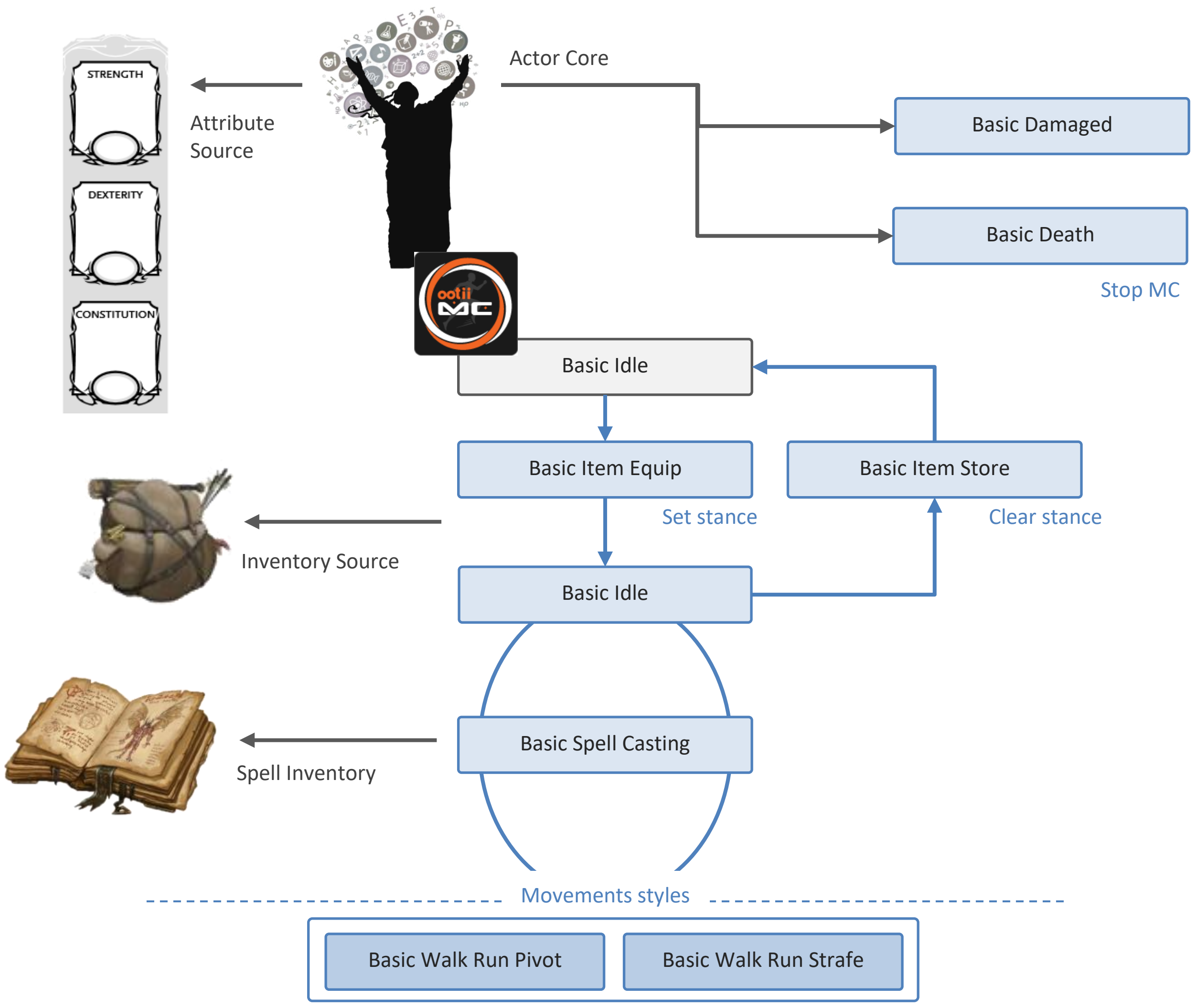
Animations
Raw animations used for movement, shooting, etc.

Messages
Information about damage, spells, and other states.

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

Motion Flow

Just like the motions that come with the Motion Controller, the Spell Casting 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 spell casting motion, want to change how a motion works, or want to add an additional motion... you do it just like any other motion. These motions are just custom motions that take advantage of Mixamo's spell casting animations.

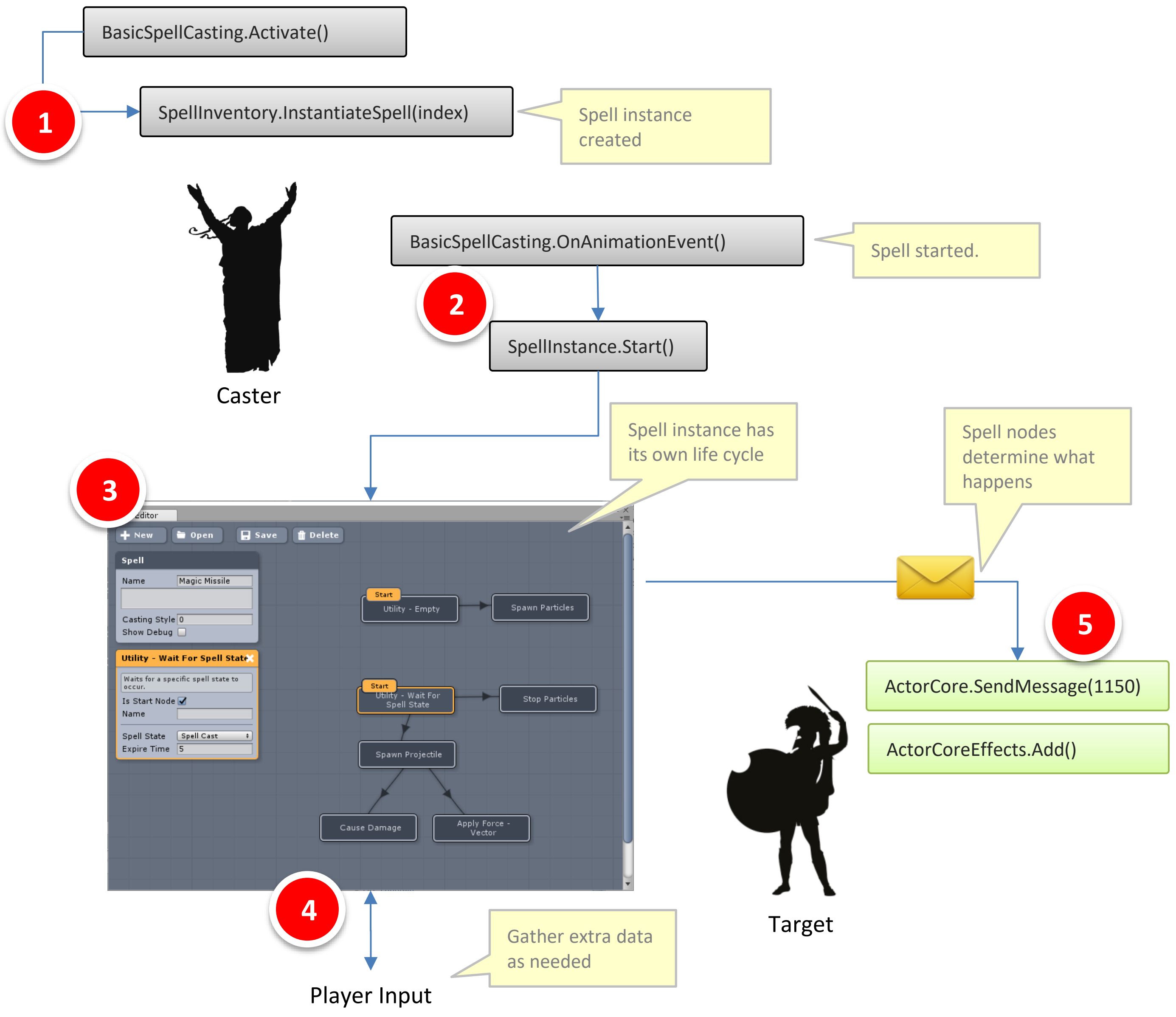
I've created this pack based on how I think magic should work. As you can imagine, there are lots of ways we could do it and your game may be different than mine. If you want the motions to act differently, you are welcome to change these motions or shoot me an email and there may be a feature or option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

Spell Casting Flow

In the end, the Spell Inventory is what actually creates and casts the spell. The BasicSpellCasting motion really just manages the caster's animation and tells the Spell Inventory when to start.

Technically, you could totally ignore the motion and cast spells through the Spell Inventory. Defining the flow is a little tricky because the flow is driven by the spell's node graph and every spell can be different.

Here's the basic flow.


BasicSpellCasting.Activate()
1
SpellInventory.InstantiateSpell(index)
Spell instance created
Caster
BasicSpellCasting.OnAnimationEvent()
Spell started.
2
SpellInstance.Start()
Spell instance has its own life cycle
Spell nodes determine what happens
3
Editor
New
Open
Save
Delete
Start
Utility - Empty
Spawn Particles
Start
Utility - Wait For Spell State
Stop Particles
Spawn Projectile
Cause Damage
Apply Force - Vector
5
ActorCore.SendMessage(1150)
ActorCoreEffects.Add()
Target
4
Player Input
Gather extra data as needed

Spell

Name: Magic Missile
Casting Style: 0
Show Debug: ☐

Utility - Wait For Spell State

Waits for a specific spell state to occur.
Is Start Node: ✓
Name:
Spell State: Spell Cast
Expire Time: 5

Spell Casting Stance

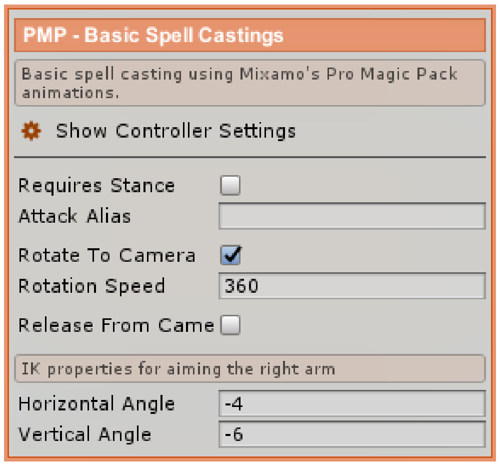
For the Spell Casting Motion Pack, entering a specific stance isn't required. That's because spell casting can occur even if you're not in a "spell casting stance".

For example, you can be in a basic idle, cast a spell, and go back to the basic idle.

Or, you could set the spell casting stance. In this approach, you would be in a spell casting idle, cast a spell, and go back to the spell casting idle.

This way, you can have a sword & shield equipped and still cast a spell.

You can control this using the "Requires Stance" property on the PMP – Basic Spell Castings motion.



PMP - Basic Spell Castings

Basic spell casting using Mixamo's Pro Magic Pack animations.

Show Controller Settings

Requires Stance	☐
Attack Alias	
Rotate To Camera	✓
Rotation Speed	360
Release From Came	☐

IK properties for aiming the right arm

Horizontal Angle	-4
Vertical Angle	-6

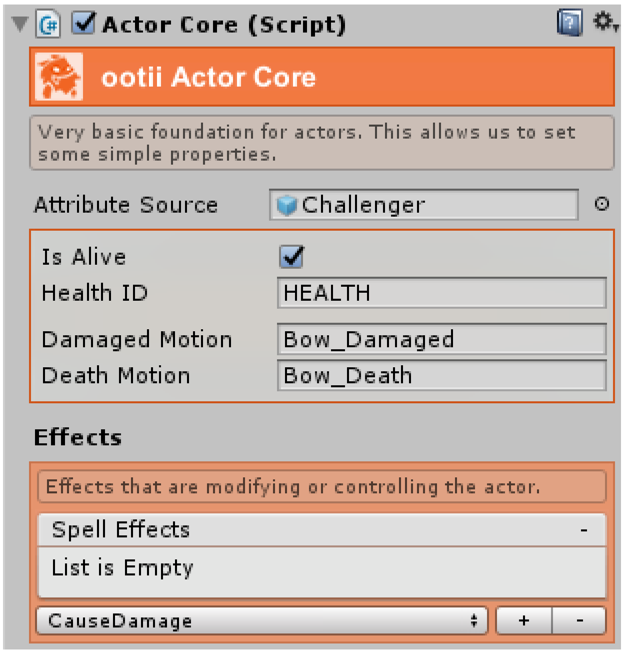
If you're not wanting to use the spell casting stance, you don't really need a Basic Inventory.

Key Components

Actor Core

The Actor Core lives on all your characters and represents the decision making and logic part of your character. When a spell 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.



Very basic foundation for actors. This allows us to set some simple properties.

Attribute Source: Challenger

Is Alive: ✓

Health ID: HEALTH

Damaged Motion: Bow_Damaged

Death Motion: Bow_Death

Effects that are modifying or controlling the actor.

Spell Effects

List is Empty

CauseDamage

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.

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

Effects – This is a dynamic list of conditions or effects that are active on your character. This is how we implement things like damage-over-time, flames on the character, etc.

Code Summary

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

OnDamaged() – This function is called by the weapon when it impacts an object that has an ActorCore. This is how the weapon 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.

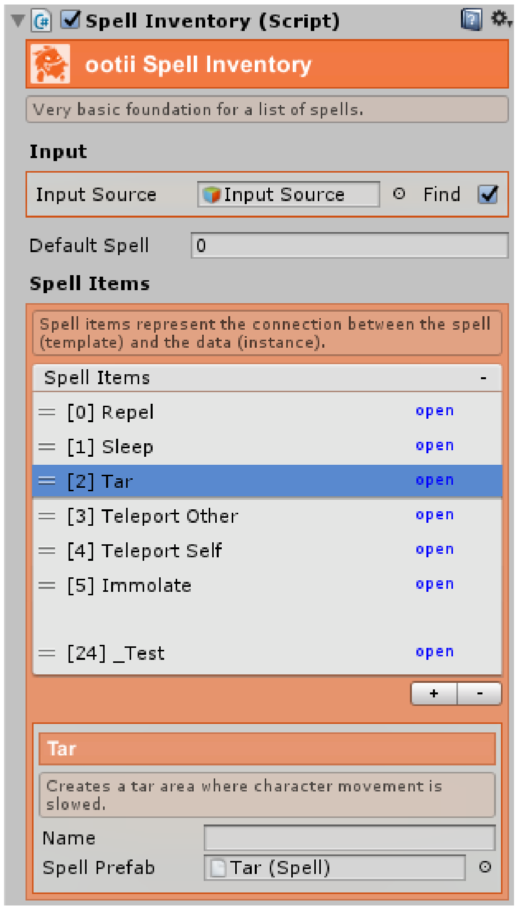
IActorCore Interface

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 replaceable.

Spell Inventory

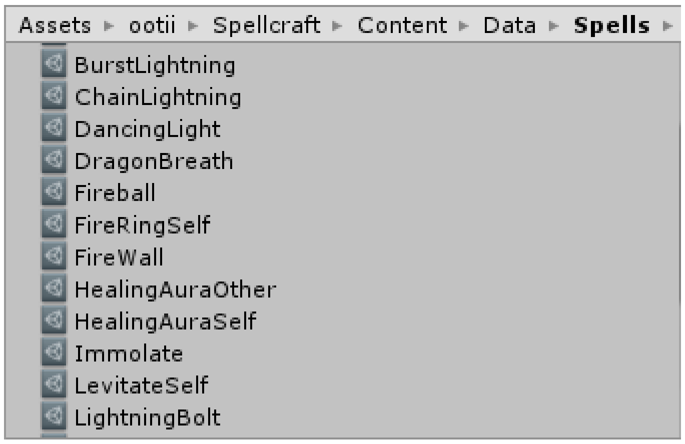
The Spell Inventory is used to hold the collection of spells the actor knows about. It is also used to instantiate a spell and activate it.

The component is found under the '`Components | Scripts | com.ootii.Actors.Magic | Spell Inventory`' menu.


Spell Inventory (Script)
ootii Spell Inventory
Very basic foundation for a list of spells.
Input
Input Source: Input Source ⊙ Find ✓
Default Spell: 0
Spell Items
Spell items represent the connection between the spell (template) and the data (instance).
Spell Items
[0] Repel open
[1] Sleep open
[2] Tar open
[3] Teleport Other open
[4] Teleport Self open
[5] Immolate open
[24] _Test open
Tar
Creates a tar area where character movement is slowed.
Name:
Spell Prefab: Tar (Spell)

When an actor will be able to cast spells, you simply need to add an entry to the list and drag the spell from the Project view into the "Spell Prefab" property.

Spells are typically found here:


Assets ► ootii ► Spellcraft ► Content ► Data ► **Spells** ►
BurstLightning
ChainLightning
DancingLight
DragonBreath
Fireball
FireRingSelf
FireWall
HealingAuraOther
HealingAuraSelf
Immolate
LevitateSelf
LightningBolt

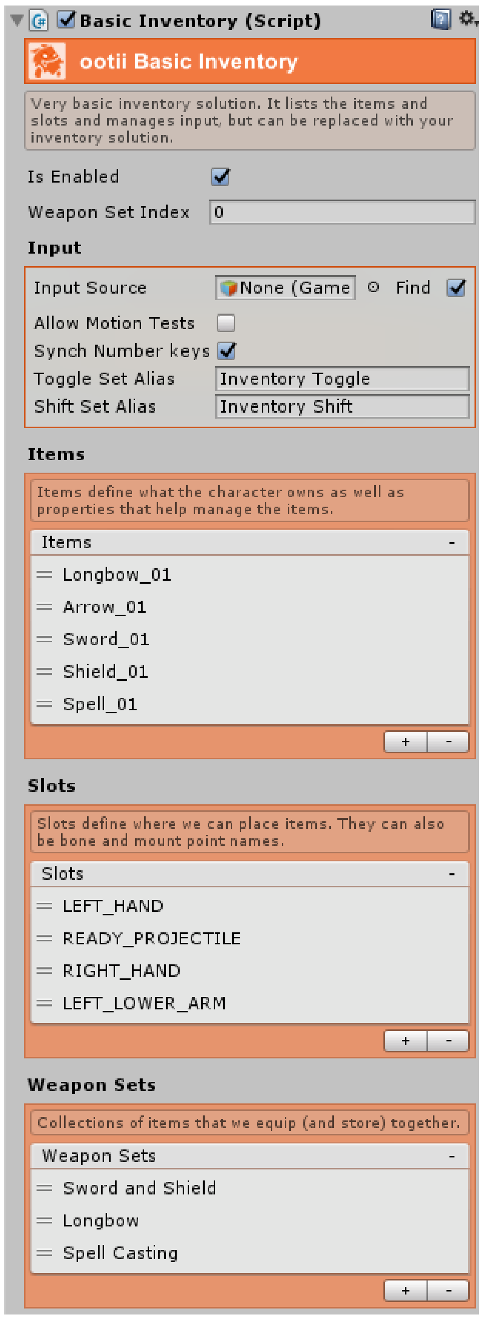
However, they can be placed anywhere in the project.

The Spell Inventory really isn't that advanced. Other than the list, the key functionality is in the `InstantiateSpell()` function. This function creates an instance of the spell based on the index and then updates that spell over time. Once the spell status is set to 'COMPLETED', the spell instance is removed.

Basic Inventory

As mentioned earlier, if you're not using the spell casting stance than you don't really need the Basic Inventory. I use it to set the stance in a consistent way.

The reason for this is that the Weapon Sets define an item to hold. That item has a motion used to equip it. That motion sets the stance. This is how we do it for the Archery Motion Pack, the Sword & Shield Motion Pack, and now the Spell Casting Motion Pack.



Very basic inventory solution. It lists the items and slots and manages input, but can be replaced with your inventory solution.

Is Enabled ✓
Weapon Set Index 0

Input
Input Source None (Game ⊙ Find ✓
Allow Motion Tests ☐
Synch Number keys ✓
Toggle Set Alias Inventory Toggle
Shift Set Alias Inventory Shift

Items
Items define what the character owns as well as properties that help manage the items.
Items: Longbow_01, Arrow_01, Sword_01, Shield_01, Spell_01

Slots
Slots define where we can place items. They can also be bone and mount point names.
Slots: LEFT_HAND, READY_PROJECTILE, RIGHT_HAND, LEFT_LOWER_ARM

Weapon Sets
Collections of items that we equip (and store) together.
Weapon Sets: Sword and Shield, Longbow, Spell Casting

While you can use any inventory solution you want, I've included a "Basic Inventory" solution for you. 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 "RIGHT_HAND" slot defines what is equipped in the right hand.

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

In the example to the left, the character has a sword ("Sword_01") and a shield ("Shield_01").

The **Weapon Sets** allow you to group items so they can be equipped and stored at the same time. For example, "Sword and shield" includes Sword_01 and Shield_01. You can imagine that another weapon set my be a bow and arrow.

Item

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.

Local Position – When mounted, the position relative to its parent. When Mount Points is used, this value is ignored.

Local Rotation – When mounted, the rotation relative to its parent. When Mount Points is used, this value is ignored.

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

Slot

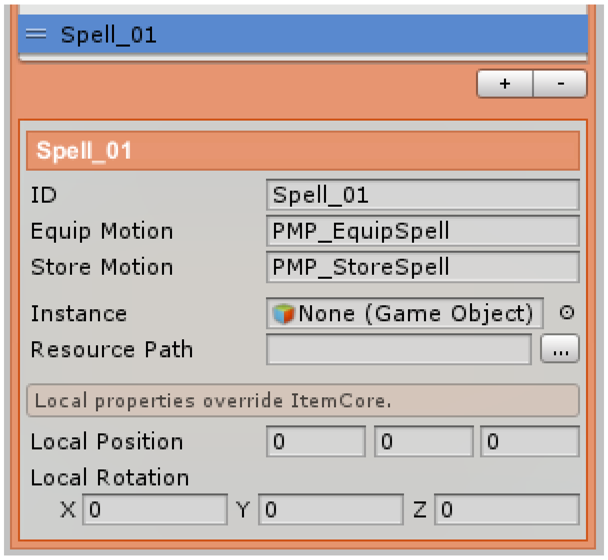
Each slot has the following properties:

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.

Spell 01

In the image above, there's an item: Spell 01.


Spell_01

Spell_01

ID	Spell_01
Equip Motion	PMP_EquipSpell
Store Motion	PMP_StoreSpell
Instance	None (Game Object)
Resource Path	

Local properties override ItemCore.

Local Position	0	0	0
Local Rotation	X 0	Y 0	Z 0

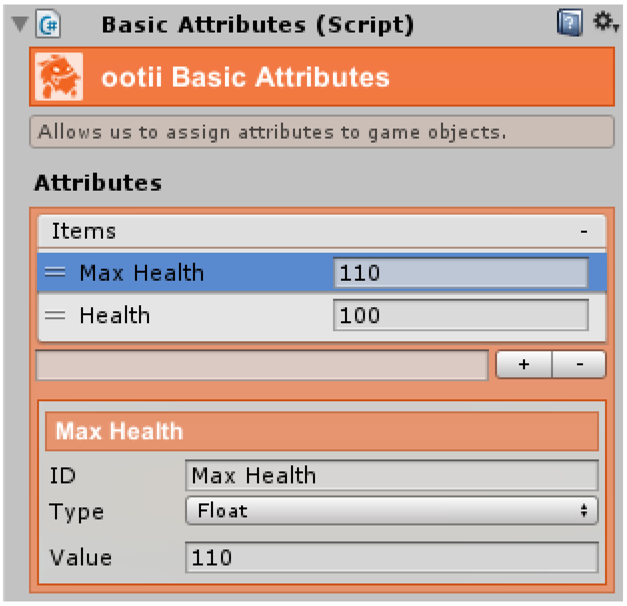
What really matters here is the "Equip Motion" and "Store Motion" properties as these are the motions activated when the weapon set equips the "Spell_01" item.

If you notice, I don't have an "Instance" or "Resource Path" set. That's because I don't actually have a visual object for this. You could, but I really just care about setting the stance through the "Equip Motion".

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.

This "Basic Attributes" component is my very basic version of an RPG attribute asset. It implements the IAttributeSource interface as all Attribute Sources would.



Basic Attributes (Script)

ootii Basic Attributes

Allows us to assign attributes to game objects.

Attributes

Items	
Max Health	110
Health	100

Max Health

ID: Max Health

Type: Float

Value: 110

By implementing attributes this way, you can add new attributes to your game. For the 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<T>() – This function returns the value of the attribute given the name.

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

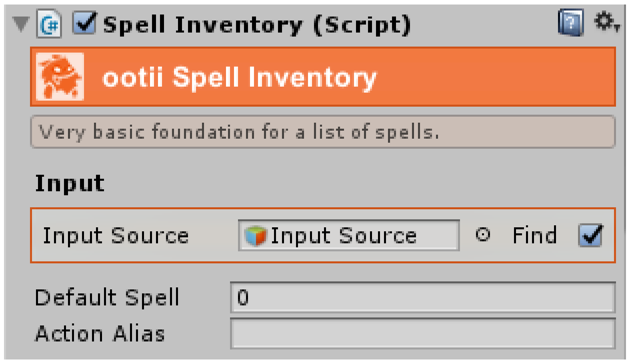
As you can image, these two functions are used by the Actor Core to get and store the character's current health. The Actor Core will determine if the character is simply damaged or killed based on the "HEALTH" attribute.

Casting Spells

At the base of casting spells is some simple code. The reason for this is that it gives you the most flexibility when making your game. For example, you could use a spell bar, separate buttons, cast with a mouse click, etc.

Spell Inventory

The Spell Inventory contains this code so that you can cast with a single click. However, I don't typically use it as it's too basic. However, you can.



Spell Inventory (Script)

ootii Spell Inventory

Very basic foundation for a list of spells.

Input

Input Source: Input Source, Find ✓

Default Spell: 0

Action Alias:

By setting the "Action Alias" to an input entry like "Fire1", the spell identified by the Default Spell index will be cast.

I don't like this approach as it only casts the default spell. However, it may be a good way to test your spells.

Basic Code

This is the root code for casting a spell. It's pretty simple and can be used anywhere. For example, in Node Canvas you would create an action that wraps the code above. In Behavior Designer and PlayMaker you would do the same thing.

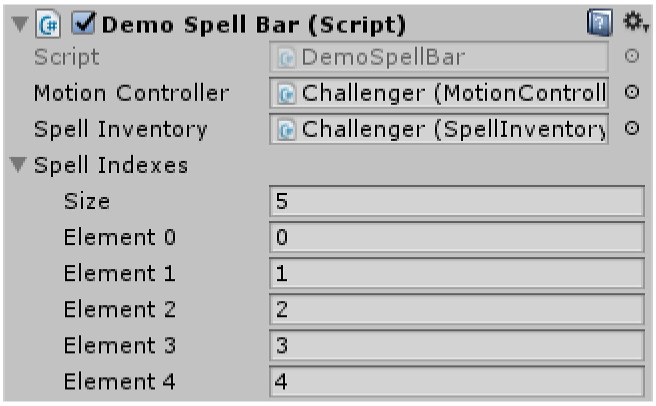
If you're using Unity's UI, you would have a UI button trigger a function on a component that wraps the code.

```
int lSpellIndex = 0;
BasicSpellCasting lCastMotion = MotionController.GetMotion< BasicSpellCasting>();
if (!lCastMotion.IsActive && (!lCastMotion.RequiresStance || MotionController.Stance ==
EnumControllerStance.SPELL_CASTING))
{
    MotionController.ActivateMotion(lCastMotion, lIndex);
}
```

The first line defines the index of the spell in the Spell Inventory.
The second line grabs the motion that will animate the caster and cast the spell.
The third line ensures that we're not casting a spell if we're not supposed to.
The fifth line activates the motion.

Demo Spell Bar

In the 'Assets\ootii_Demos\MotionControllerPacks\SpellCasting\Scenes' folder, I've included code for a sample spell bar. It's called DemoSpellBar.cs. It's not pretty, but it works.



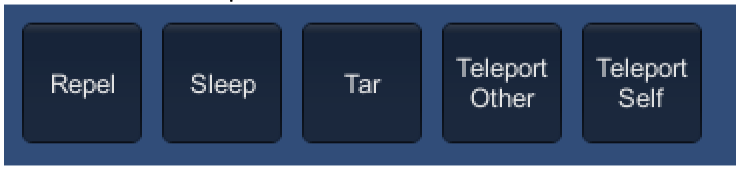
Demo Spell Bar (Script)

Script	DemoSpellBar
Motion Controller	Challenger (MotionControll
Spell Inventory	Challenger (SpellInventory
Spell Indexes	
Size	5
Element 0	0
Element 1	1
Element 2	2
Element 3	3
Element 4	4

By adding the component to your scene, you can specify which Spell Inventory indexes will be listed.

In the example, I'm list the first five spells in the Spell Inventory.

The results is this spell bar:



By clicking a button, the spell will be cast.

Code

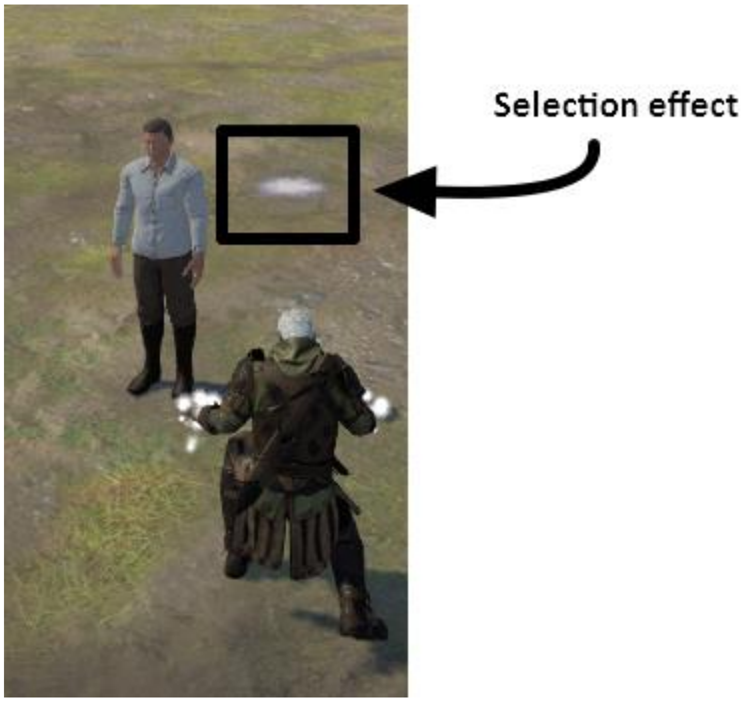
You can see the code for the DemoSpellBar isn't that complicated. We're just creating standard Unity GUI Buttons and then calling the code that I mentioned above.

```
public class DemoSpellBar : MonoBehaviour
{
    /// <summary>
    /// Motion Controller that will cast the spells
    /// </summary>
    public MotionController MotionController = null;

    /// <summary>
    /// List of spells to cast
    /// </summary>
    public SpellInventory SpellInventory = null;

    /// <summary>
    /// Indexes that controll the UI
    /// </summary>
    public List<int> SpellIndexes = new List<int>();

    void OnGUI()
    {
        float lWidth = 60f;
        float lHeight = 60f;
        float lSpacer = 10f;

        for (int i = 0; i < SpellIndexes.Count; i++)
        {
            int lIndex = SpellIndexes[i];

            string lName = SpellInventory._Spells[lIndex].Name.Replace(" ", "\n");

            if (GUI.Button(new Rect(10f + ((lWidth + lSpacer) * i), 10f, lWidth, lHeight), lName))
            {
              BasicSpellCasting lCastMotion = MotionController.GetMotion< BasicSpellCasting>();
              if (!lCastMotion.IsActive && (!lCastMotion.RequiresStance || MotionController.Stance ==
              EnumControllerStance.SPELL_CASTING))
              {
                  MotionController.ActivateMotion(lCastMotion, lIndex);
              }
            }
        }
    }
}
```

Targeting Spells

Some spells like Teleport require a character or position to be targeted. This is done through the camera.

Once the spell is cast, a selection effect will appear and move with the center of the camera.

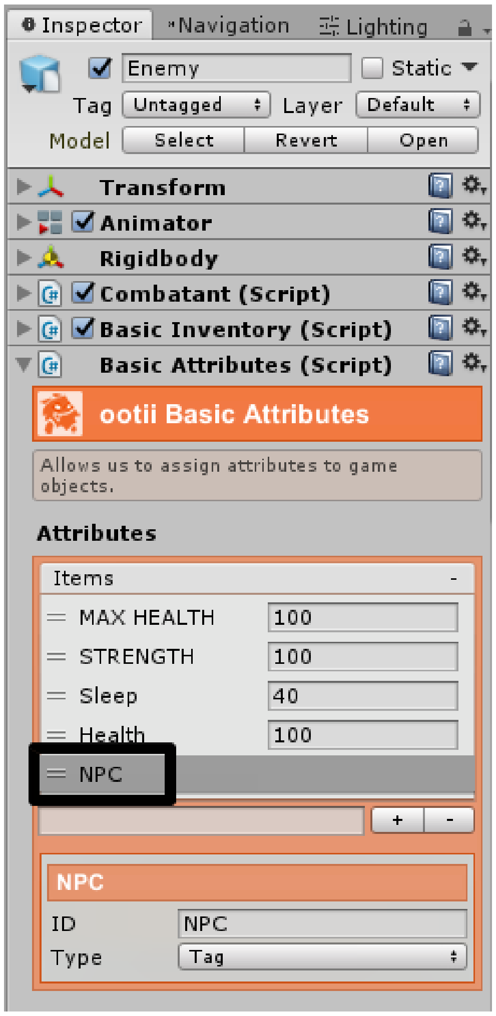


Once you've select a spot, press the 'fire' button and the position or target will be selected and the spell will continue.

NPCs

Not all of the spells I created will work with NPCs. That's because NPCs don't select ground points with the mouse or use the camera to select targets. So, some spells may have to be built specifically for NPCs.

Some spells (like Magic Missile and Chain Lighting), I've built as an example of how spells could be used with PCs and NPCs. In these cases, we use the Basic Attributes component to determine if the spell caster is an NPC:



Enemy Static
Tag Untagged Layer Default
Model

Allows us to assign attributes to game objects.

Attributes

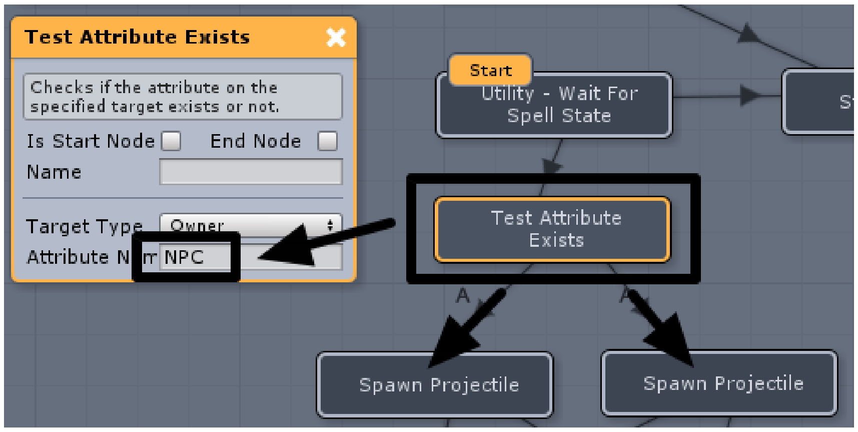
Items

MAX HEALTH	100
STRENGTH	100
Sleep	40
Health	100
NPC	

NPC
ID NPC
Type Tag

By adding the tag 'NPC', spell nodes can be used to direct the flow based on whether the tag was found or not.

The 'Test Attribute Exists' node does this well:



Test Attribute Exists

Checks if the attribute on the specified target exists or not.

Is Start Node End Node
Name

Target Type Owner
Attribute Nam NPC

Stock Spell Descriptions

Attract

This spell causes character and objects within a radius to be pulled towards the caster. The strength of the pull is determined by the power set in the "Apply Force – Spherical" node.

Burst Lightning

When cast, the player uses the camera to look at a target. That target will glow when valid. Pressing the 'fire' button will cast lightning at the target. The lighting with then jump to other nearby targets, damaging them all.

Chain Lightning

Similar to Burst Lightning, the player will cast the spell and use the camera to select a target. Pressing the 'fire' button will cast lighting at the target and that lighting will then jump from one target to another.

Dancing Lights

Once cast, a light will be created in front of the caster that will move with the player. After a period of time, it will fade.

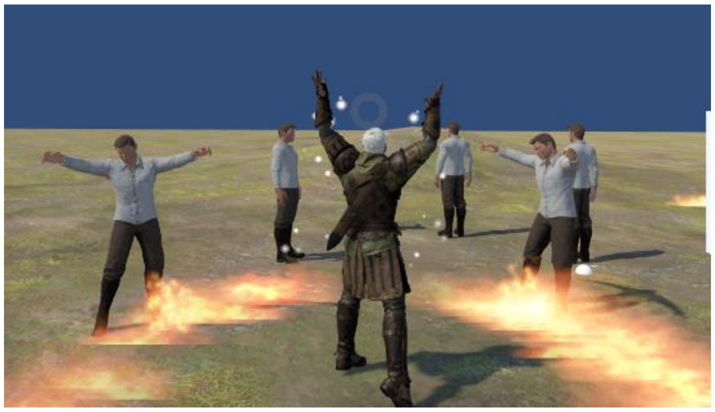
Dragon's Breath

Dragon's Breath allows the caster to breathe flames from his mouth. Characters caught within the flames will catch fire and take damage over time.

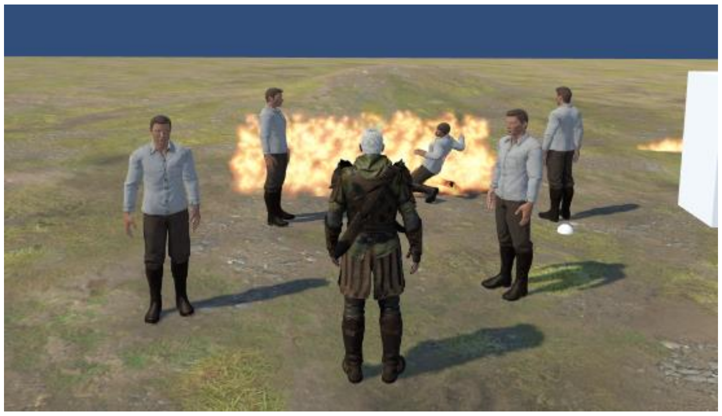
Fireball

When cast, a projectile is shot forward. If it impacts something before expiring, the fireball will explode and damage everyone in the area of effect.

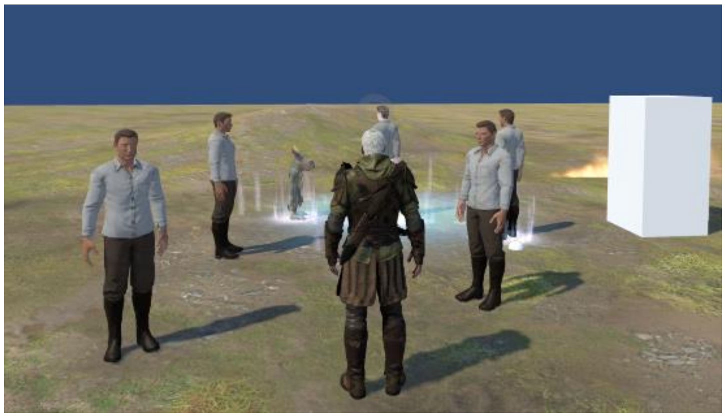
Fire Ring Self
Fire Ring creates a ring of fire around the cast. Anyone caught in the ring will be pushed back. Those characters that cross the fire will catch fire and take damage.



Fire Wall
Once cast, the player will use the camera to select a place on the ground. By pressing the 'fire' button, a small wall of fire will be placed perpendicular to the caster. Anyone who touches the wall takes damage over time.

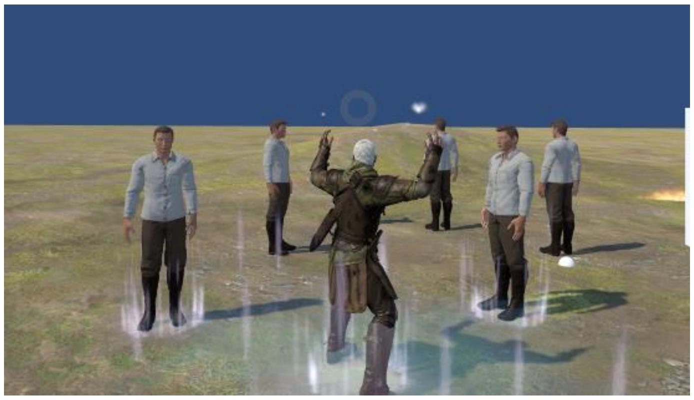


Healing Aura Other
When cast, the player will select a target using the camera and 'fire' button. Using the target as the center, anyone within the radius will be healed over time.



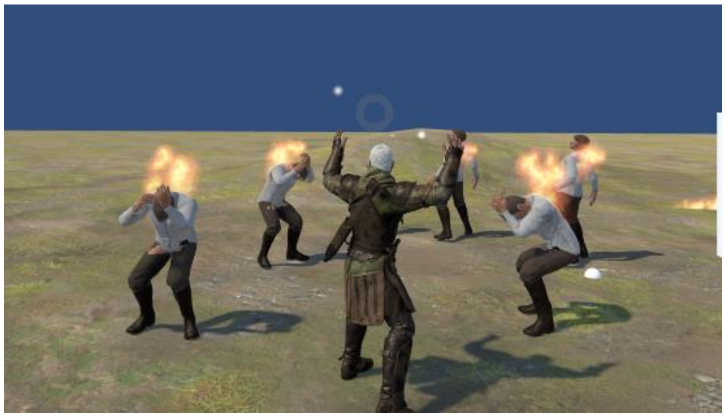
Healing Aura Self

Once cast, anyone within the specified radius of the caster will be healed over time.



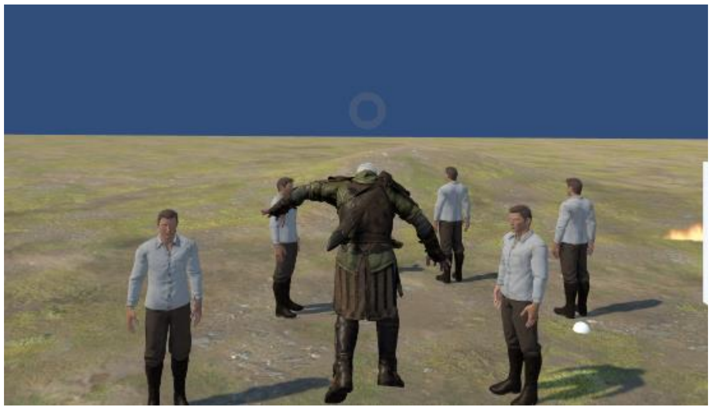
Immolate

Immolate catches anyone within a specified radius of the caster on fire. At the same time, it causes a crippling fear that makes the targets cower and unable to move.



Levitate Self

Once cast, the caster is able to levitate for a short amount of time. While levitating, they can float up and down as well as move laterally as a very slow speed.



Lightning Bolt

This spell launches a bolt of lightning forward. Anyone caught by the bolt will take damage.

Magic Missile

Once cast, a missile is shot forward. When it hits, it will push the target back and cause damage.

Poisonous Fog

Once cast, the player selects an area on the ground using the camera and presses the ‘fire’ button. A cloud of poison will rise and anyone caught in the cloud will take damage.

Repel

Repel is used to push characters and objects away from the caster.

Sleep

Anyone caught within the radius of the spell may lay down and fall asleep. This spell allows characters to have a resistance to the affect.

Tar

When cast, the player uses the camera to choose a target on the ground. By pressing the ‘fire’ button, tar is placed and any movement through the area is severely hampered (including the player’s). When characters leave the area movement goes back to normal.

Teleport Other

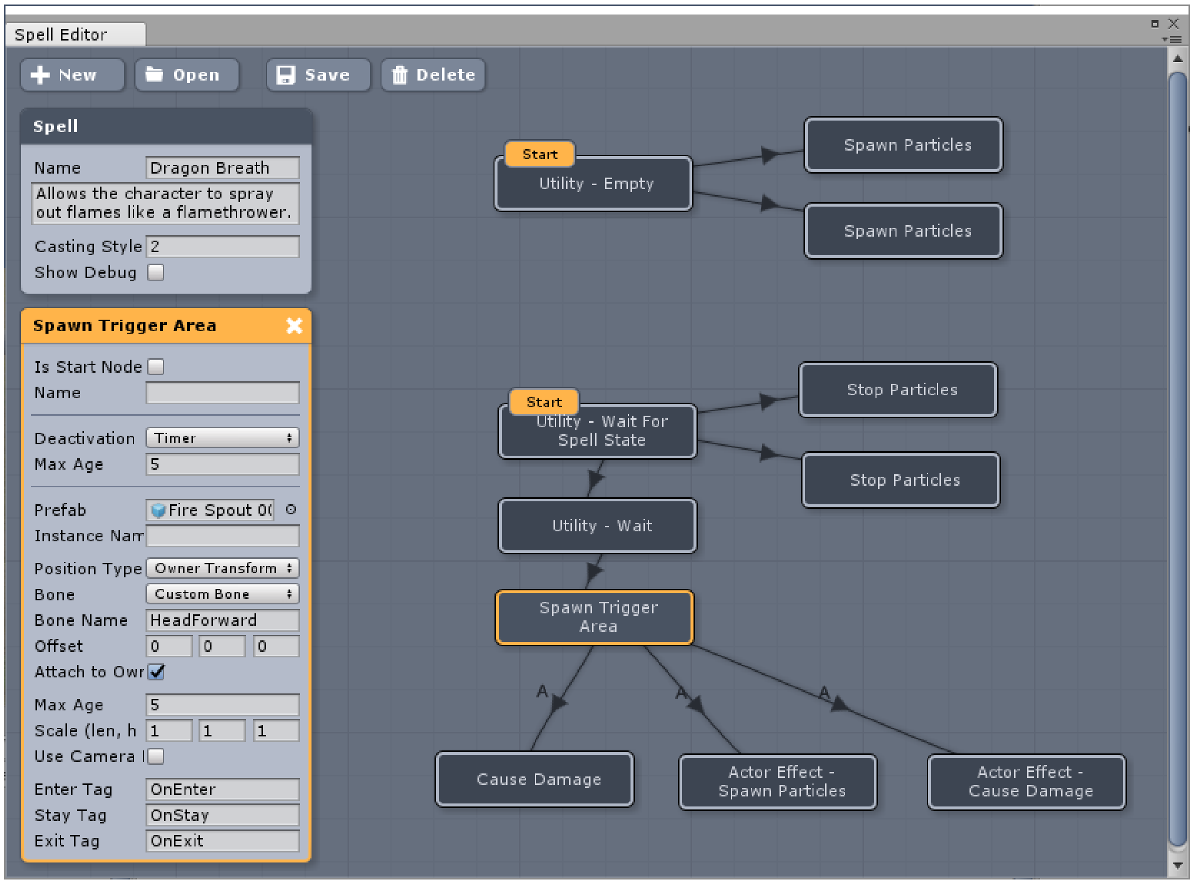
Teleport Other allows the caster to choose a target using the camera. Once selected, the player then chooses an area on the ground with the camera and presses the 'fire' button. The target will be teleported to that new area.

Teleport Self

Teleport Self allows the caster to choose a position on the ground with the camera. By pressing the 'fire' button, the caster will be teleported to that new area.

Custom Spells

The whole premise of the Spell Casting Motion Pack is that you can build your own spells through the Spell Editor.



Spell

Name	Dragon Breath
	Allows the character to spray out flames like a flamethrower.
Casting Style	2
Show Debug	☐

Spawn Trigger Area

Is Start Node	☐
Name	
Deactivation	Timer
Max Age	5
Prefab	Fire Spout 0(
Instance Nam	
Position Type	Owner Transform
Bone	Custom Bone
Bone Name	HeadForward
Offset	0 0 0
Attach to Owr	✓
Max Age	5
Scale (len, h	1 1 1
Use Camera I	☐
Enter Tag	OnEnter
Stay Tag	OnStay
Exit Tag	OnExit

However, the details of that is found in the Spell Builders' Guide:

www.ootii.com/Unity/MotionControllerPacks/SpellCasting/SpellCasting_BuildersGuide.pdf

Frequently Asked Questions

Where is the mana pool?

This asset is about how spells are cast and the affect that they have. It is not about when spells are cast.

For now at least, determining if a spell can be cast based on time, mana, inventory items, etc. is up to the game developer.

Can NPCs cast spells?

Yes. Can they cast all spells out-of-the-box? No.

Teleport for example, requires the player to select a position to teleport to. NPCs can't use the camera to select a position. So, the spell would need to be modified to select an appropriate position for the NPC. Then, the teleport would work.

Spells that don't require selecting a target or selecting a position can typically be cast by NPCs out-of-the-box.

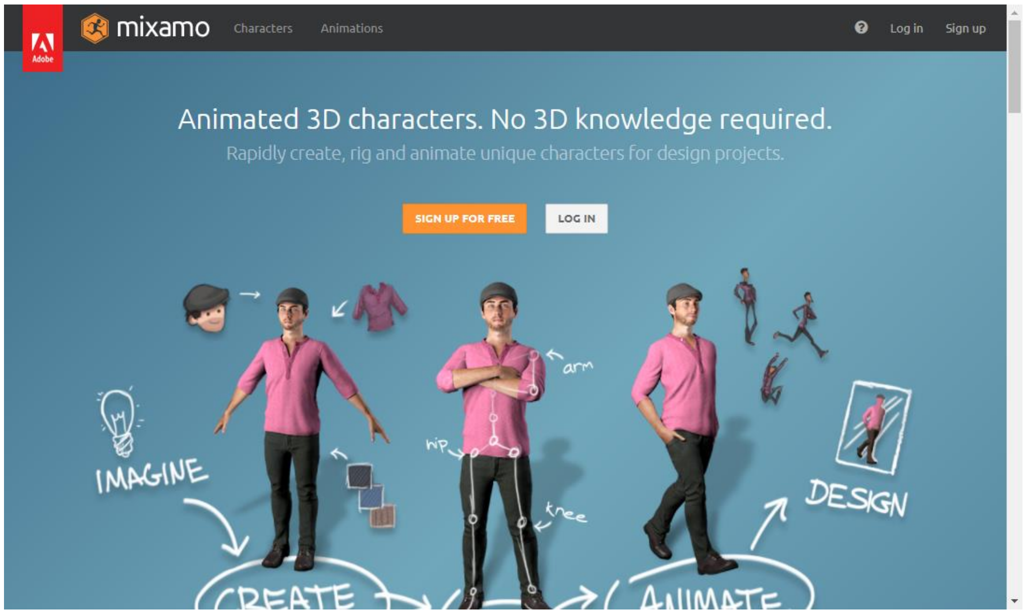
Check out more detail about NPCs here.

Mixamo Animation Download

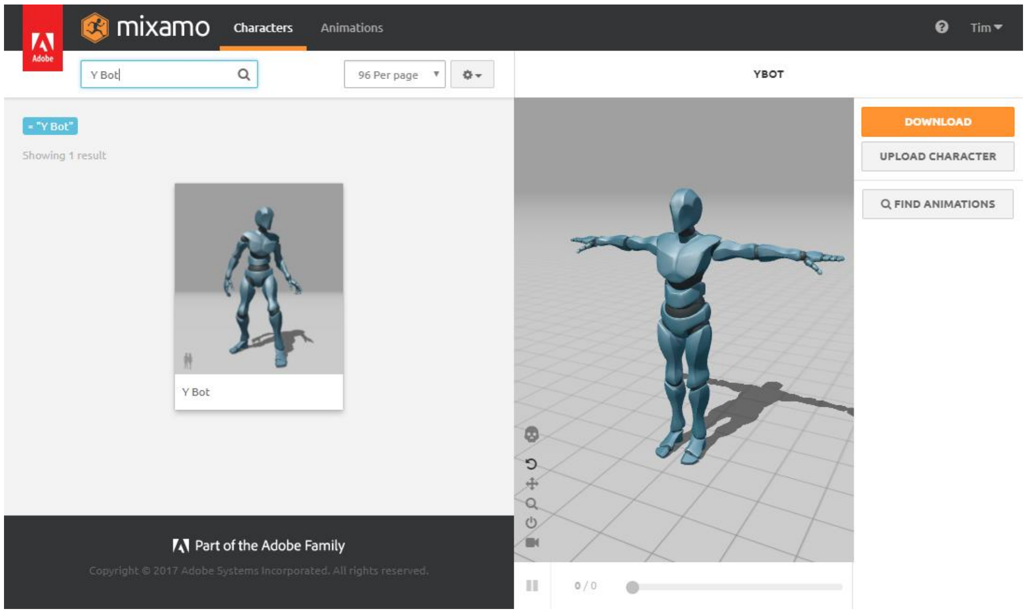
On August 23rd 2017, Mixamo updated their site and changed the flow. In addition, they change file names. This flow represents the updated process for getting the animations.

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

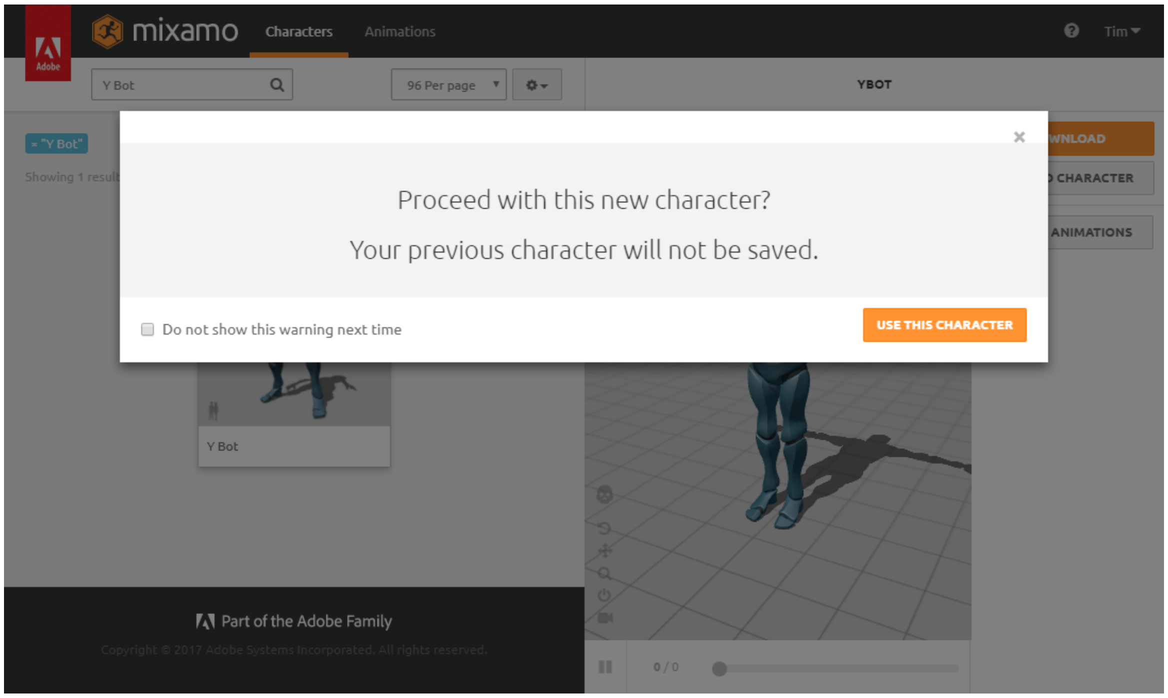
1. Go to www.mixamo.com and login


mixamo
Characters
Animations
Log in
Sign up
Animated 3D characters. No 3D knowledge required.
Rapidly create, rig and animate unique characters for design projects.
SIGN UP FOR FREE
LOG IN
IMAGINE
CREATE
ANIMATE
DESIGN
arm
hip
knee

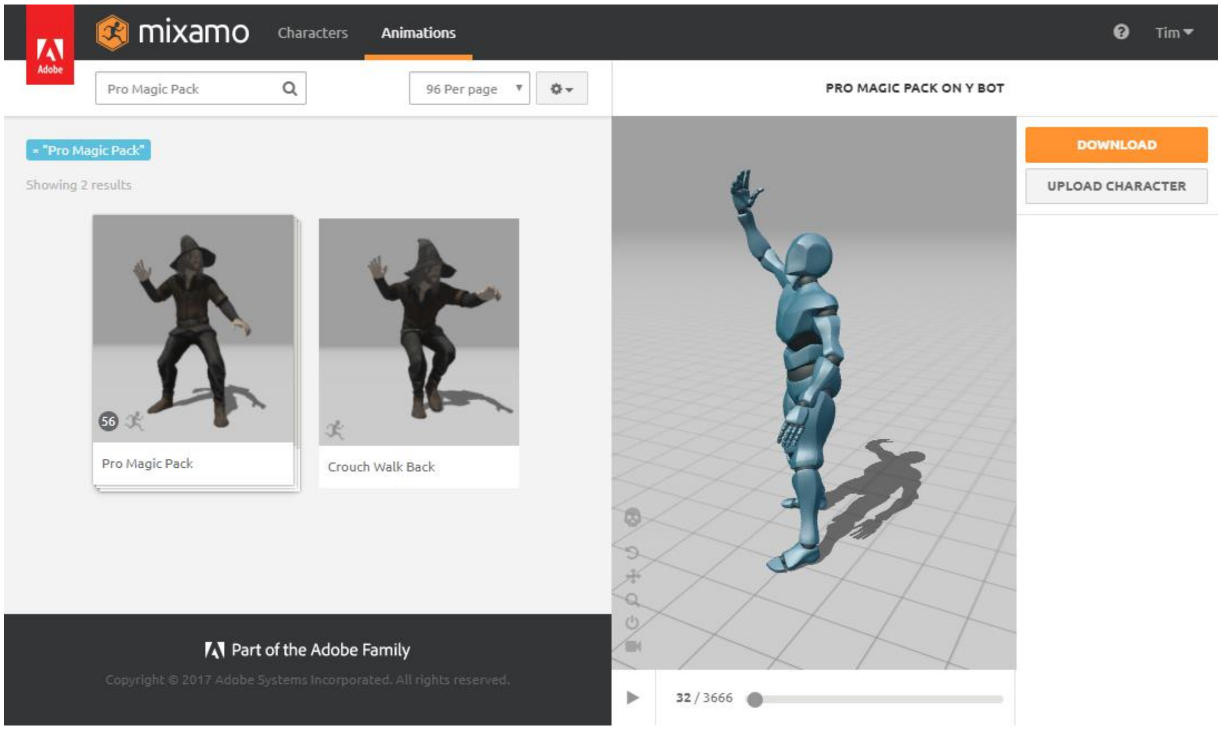
2. Go to the “Characters” link at the top and search for “Y bot”



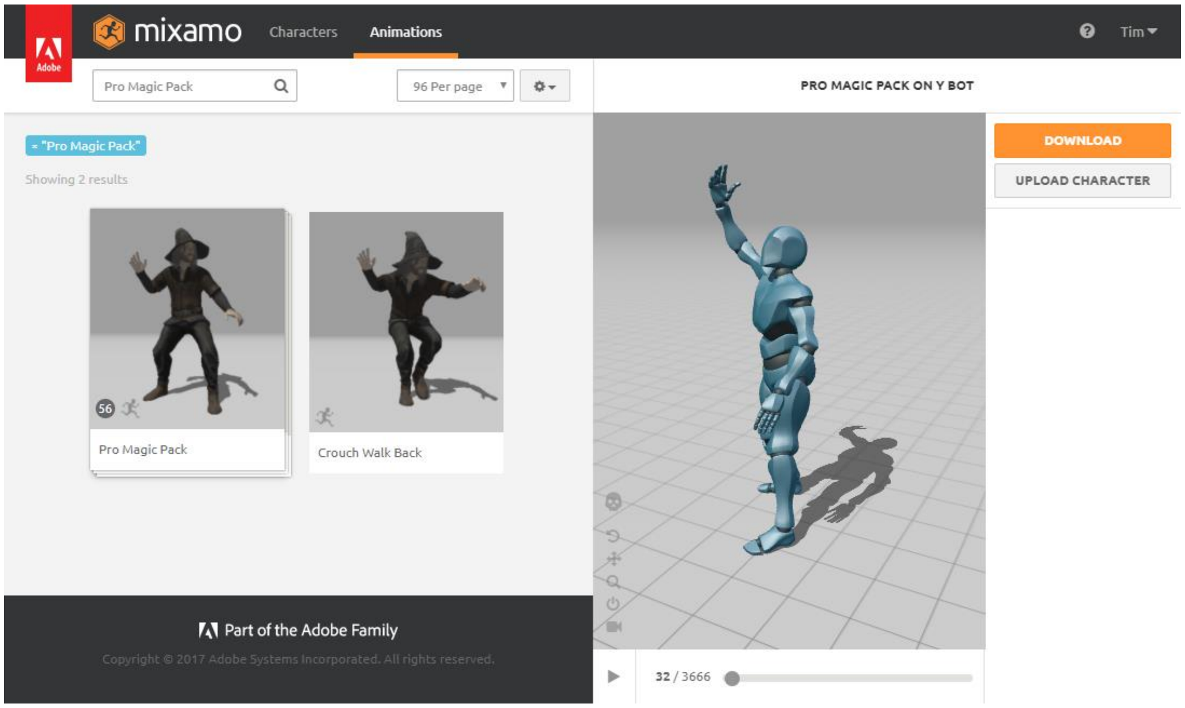
3. Select the "Y Bot" character and press "Use This Character". This will make the Y Bot your default character for the selected downloads.


mixamo
Characters
Animations
Tim
Y Bot
96 Per page
YBOT
"Y Bot"
Showing 1 result
Proceed with this new character?
Your previous character will not be saved.
Do not show this warning next time
USE THIS CHARACTER
Y Bot
Part of the Adobe Family
Copyright © 2017 Adobe Systems Incorporated. All rights reserved.
0 / 0

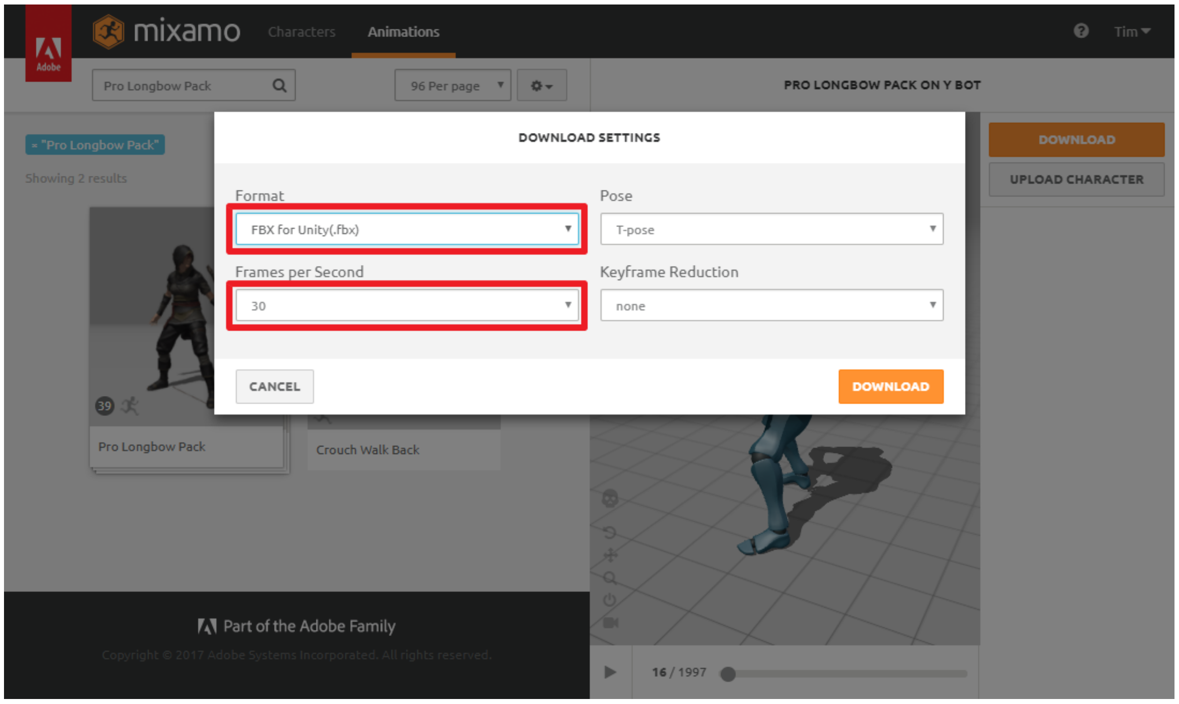
4. Go to the "Animations" link at the top and search for "Pro Magic Pack"



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

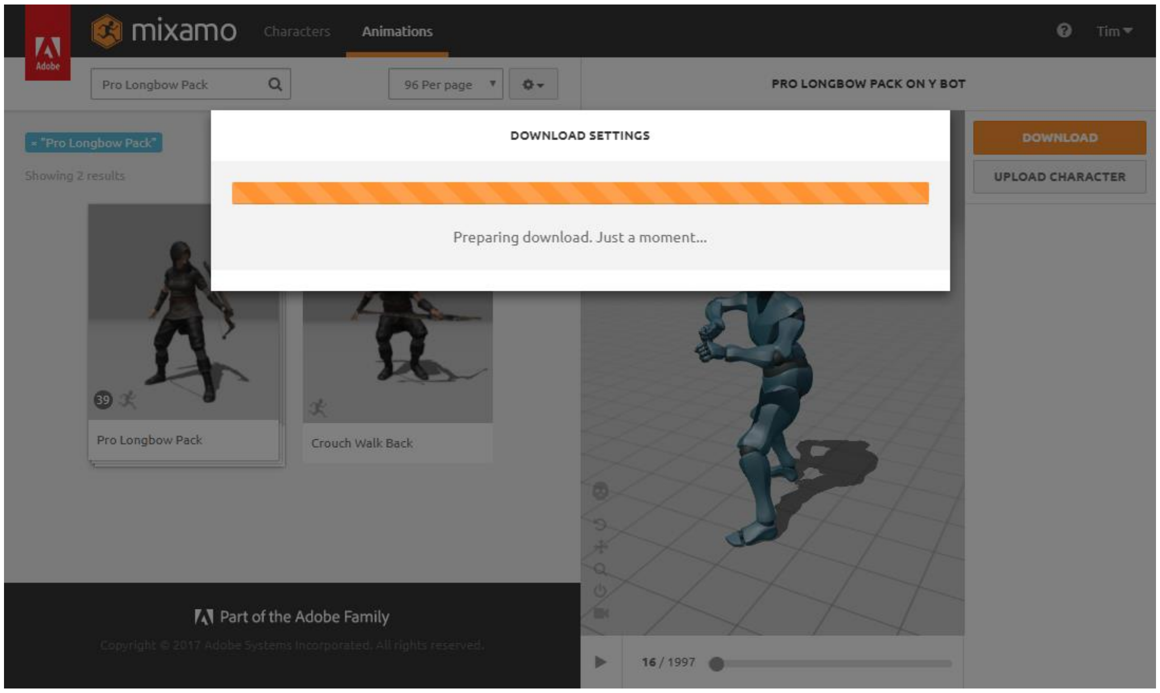


6. Select “FBX for Unity” and “30” Frames per Second. Then, press “Download”.

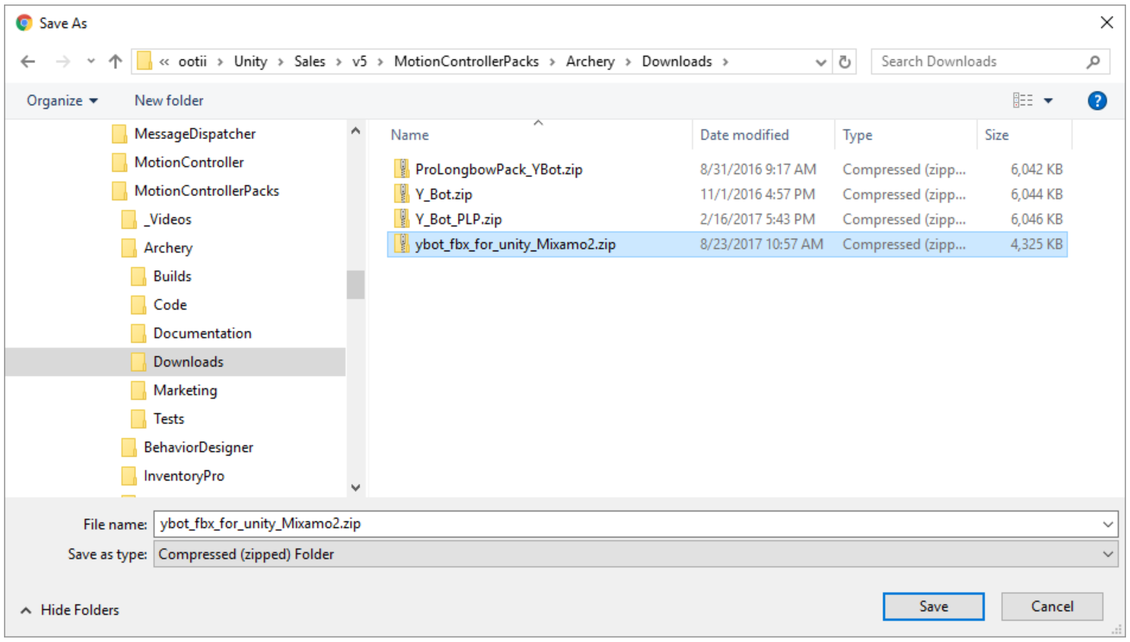

DOWNLOAD SETTINGS

Format: FBX for Unity(.fbx)
Pose: T-pose
Frames per Second: 30
Keyframe Reduction: none

7. Wait for the processing to finish



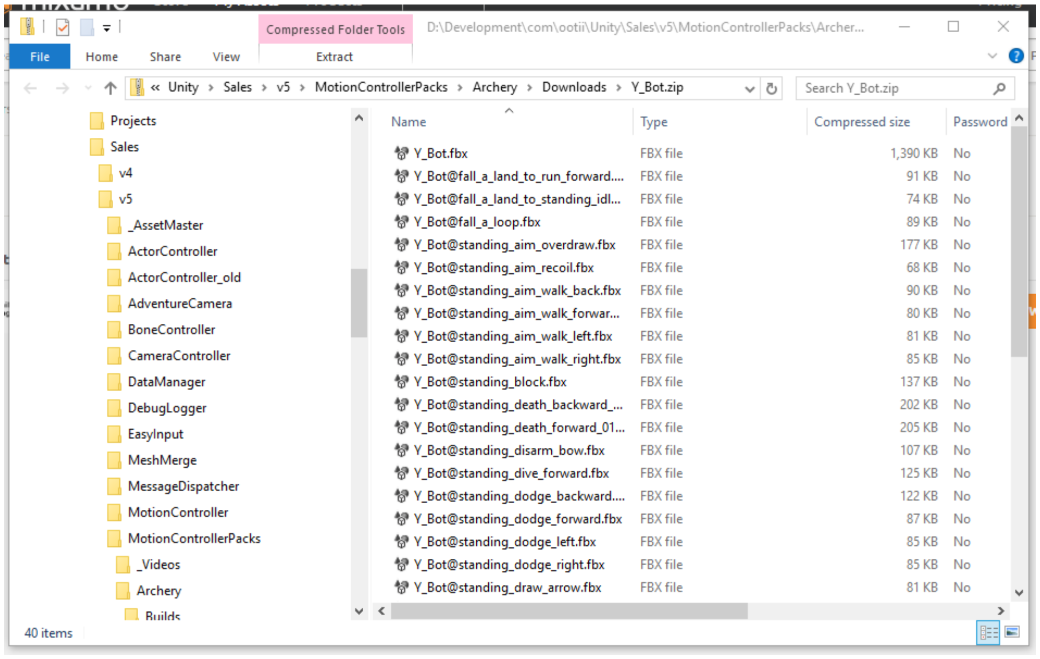
8. Select where to store the animations



Name	Date modified	Type	Size
ProLongbowPack_YBot.zip	8/31/2016 9:17 AM	Compressed (zipp...	6,042 KB
Y_Bot.zip	11/1/2016 4:57 PM	Compressed (zipp...	6,044 KB
Y_Bot_PLP.zip	2/16/2017 5:43 PM	Compressed (zipp...	6,046 KB
ybot_fbx_for_unity_Mixamo2.zip	8/23/2017 10:57 AM	Compressed (zipp...	4,325 KB

File name: ybot_fbx_for_unity_Mixamo2.zip
Save as type: Compressed (zipped) Folder

9. Unzip the contents to “...\Assets\ootii\MotionControllerPacks\SpellCasting\Content\Animations\Mixamo”


Compressed Folder Tools
D:\Development\com\ootii\Unity\Sales\v5\MotionControllerPacks\Archer...
File
Home
Share
View
Extract
« Unity › Sales › v5 › MotionControllerPacks › Archery › Downloads › Y_Bot.zip
Search Y_Bot.zip
40 items

Projects
Sales
v4
v5
_AssetMaster
ActorController
ActorController_old
AdventureCamera
BoneController
CameraController
DataManager
DebugLogger
EasyInput
MeshMerge
MessageDispatcher
MotionController
MotionControllerPacks
_Videos
Archery
Builds

Name	Type	Compressed size	Password
Y_Bot.fbx	FBX file	1,390 KB	No
Y_Bot@fall_a_land_to_run_forward....	FBX file	91 KB	No
Y_Bot@fall_a_land_to_standing_idl...	FBX file	74 KB	No
Y_Bot@fall_a_loop.fbx	FBX file	89 KB	No
Y_Bot@standing_aim_overdraw.fbx	FBX file	177 KB	No
Y_Bot@standing_aim_recoil.fbx	FBX file	68 KB	No
Y_Bot@standing_aim_walk_back.fbx	FBX file	90 KB	No
Y_Bot@standing_aim_walk_forwar...	FBX file	80 KB	No
Y_Bot@standing_aim_walk_left.fbx	FBX file	81 KB	No
Y_Bot@standing_aim_walk_right.fbx	FBX file	85 KB	No
Y_Bot@standing_block.fbx	FBX file	137 KB	No
Y_Bot@standing_death_backward_...	FBX file	202 KB	No
Y_Bot@standing_death_forward_01...	FBX file	205 KB	No
Y_Bot@standing_disarm_bow.fbx	FBX file	107 KB	No
Y_Bot@standing_dive_forward.fbx	FBX file	125 KB	No
Y_Bot@standing_dodge_backward....	FBX file	122 KB	No
Y_Bot@standing_dodge_forward.fbx	FBX file	87 KB	No
Y_Bot@standing_dodge_left.fbx	FBX file	85 KB	No
Y_Bot@standing_dodge_right.fbx	FBX file	85 KB	No
Y_Bot@standing_draw_arrow.fbx	FBX file	81 KB	No

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

10. Continue to the next section...

Animation & Meta Files

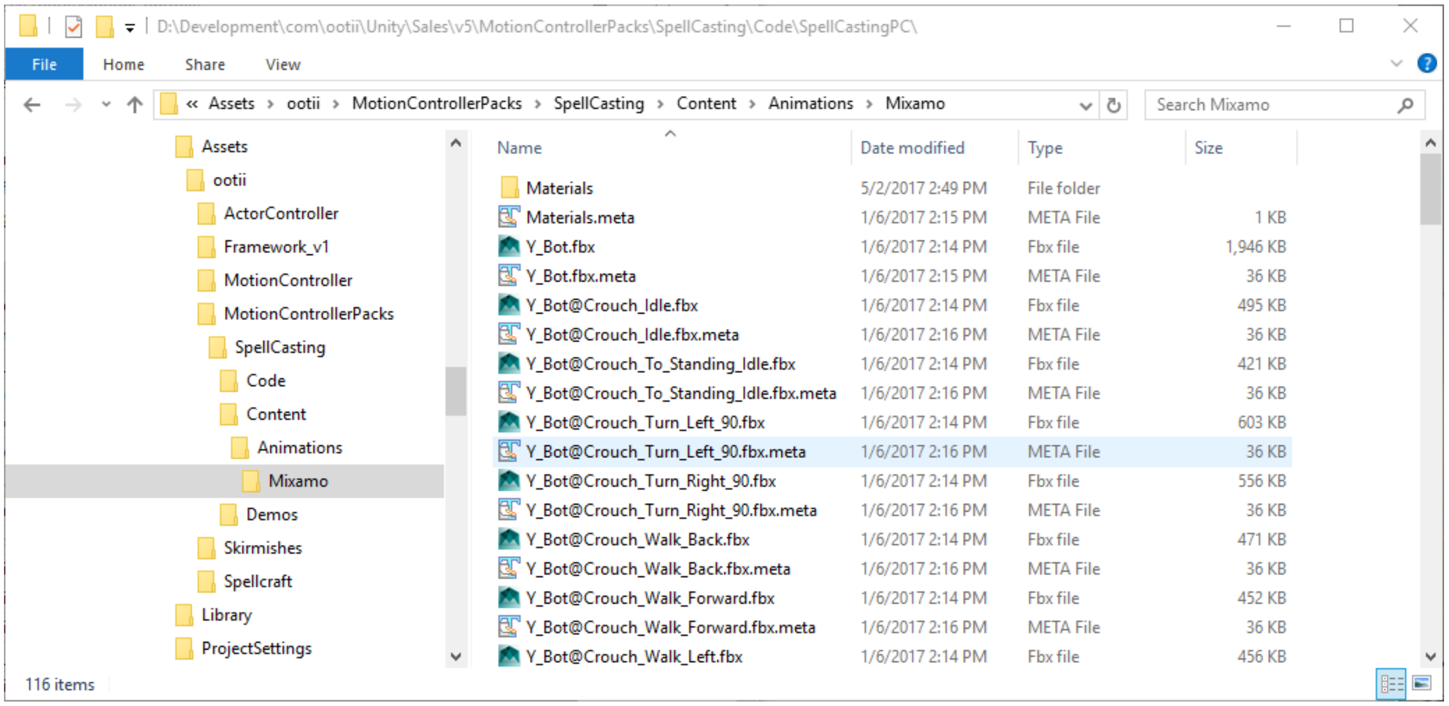
Assets\ootii\MotionControllerPacks\SpellCasting\Content\Animations\Mixamo

When you first import the Spell Casting 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 56 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. Instead, you want the meta files found in the Assets\ootii\MotionControllerPacks\SpellCasting\Extras\AnimationMeta.zip file.

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



Name	Date modified	Type	Size
Materials	5/2/2017 2:49 PM	File folder	
Materials.meta	1/6/2017 2:15 PM	META File	1 KB
Y_Bot.fbx	1/6/2017 2:14 PM	Fbx file	1,946 KB
Y_Bot.fbx.meta	1/6/2017 2:15 PM	META File	36 KB
Y_Bot@Crouch_Idle.fbx	1/6/2017 2:14 PM	Fbx file	495 KB
Y_Bot@Crouch_Idle.fbx.meta	1/6/2017 2:16 PM	META File	36 KB
Y_Bot@Crouch_To_Standing_Idle.fbx	1/6/2017 2:14 PM	Fbx file	421 KB
Y_Bot@Crouch_To_Standing_Idle.fbx.meta	1/6/2017 2:16 PM	META File	36 KB
Y_Bot@Crouch_Turn_Left_90.fbx	1/6/2017 2:14 PM	Fbx file	603 KB
Y_Bot@Crouch_Turn_Left_90.fbx.meta	1/6/2017 2:16 PM	META File	36 KB
Y_Bot@Crouch_Turn_Right_90.fbx	1/6/2017 2:14 PM	Fbx file	556 KB
Y_Bot@Crouch_Turn_Right_90.fbx.meta	1/6/2017 2:16 PM	META File	36 KB
Y_Bot@Crouch_Walk_Back.fbx	1/6/2017 2:14 PM	Fbx file	471 KB
Y_Bot@Crouch_Walk_Back.fbx.meta	1/6/2017 2:16 PM	META File	36 KB
Y_Bot@Crouch_Walk_Forward.fbx	1/6/2017 2:14 PM	Fbx file	452 KB
Y_Bot@Crouch_Walk_Forward.fbx.meta	1/6/2017 2:16 PM	META File	36 KB
Y_Bot@Crouch_Walk_Left.fbx	1/6/2017 2:14 PM	Fbx file	456 KB

Some meta files will not be valid because I'm including the old ones and the new ones.

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.