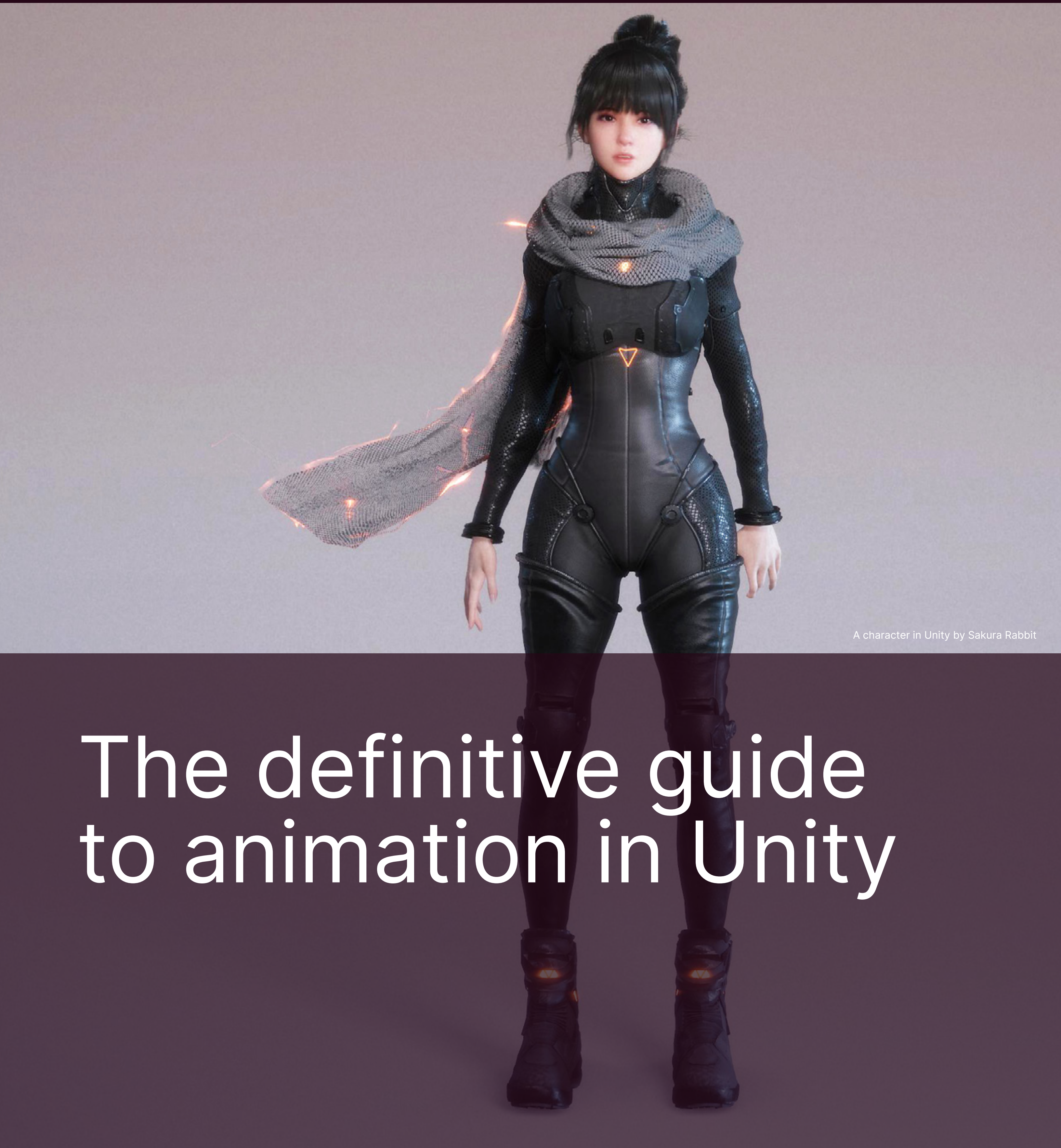
UNITY 2022 LTS EDITION → E-BOOK


A character in Unity by Sakura Rabbit

The definitive guide to animation in Unity

Contents

Introduction

A screen shot from the Unity animated short, *Enemies*

Animation is a crucial part of game design, with many games now featuring animated content that's close in visual quality to that of cinematic films.

Animators carry the story of a game forward through characterization and visual storytelling. They are responsible for character rigging, posing, and keyframing animation sequences. As the role of the animator continues to evolve in game development, so do the animation tools and workflows in Unity.

This guide is for both experienced and new animators who are interested in understanding how to use the dedicated animation tools, methods, and techniques available in Unity's animation system.

You'll learn about how to import animations into Unity, the generic and humanoid animation types, animate and rig in the Editor, create animated cutscenes with Timeline, advanced animation options, and more.

Whether you're creating photorealistic scenes, or hand-painted, cell-shaded, or cartoon visuals, Unity can help you bring your animations to life.

An animated character reacting to the environment with runtime rigging

Contributors

Pete Jepson is a professional tutor who has taught creative media subjects like film making, animation, photography, graphic design and game design for the past decade. With ten years of experience in Unity, he has multiple design courses on Udemy.com.

Unity Contributors

Dave Hunt is a technical artist for animation and rigging at Unity in Copenhagen. He is passionate about empowering artists and animators with reproducible pipelines and seamless interoperability. He worked in game development for 18 years in the Seattle area before joining Unity in 2017. He contributed to games from the Halo and Destiny series at Bungie as leader of the Rigging Tech Art team, while also teaching part-time in the animation capstone program at the University of Washington.

Foreword by Dave Hunt

Animation seems to me to be one of the truest forms of creative expression. When something is well animated it can create such a believable illusion of life. In his now-classic book, *Illusion of Life: Disney Animation,* Frank Thomas wrote about how he, Ollie Johnston, and their colleagues in the early years at Disney pioneered and codified the fundamental principles of animation that are still practiced by professional animators today.

I discovered the joy of animation as a kid, with my own stop motion projects of toy robots and figures made of clay and wood. The exhilaration of seeing my ideas come to life drove me to take on more ambitious projects; eventually it led me into the computer animation capstone program at the University of Washington. Little did I know that my choice of study would lead to a meeting with Frank Thomas at his home in Burbank, California, where he personally reviewed my student animation project. I will never forget the youthful glow in his eyes at age 91, while he investigated and articulated just what makes an animated dog seem so much like a real dog. He was constantly observing life and taking note of the beautiful subtleties of how creatures move so that he could integrate them into his animated works.

Since that encounter with a true master, I've felt compelled to share the joy of animation with everyone. I've been lucky to have the opportunity to teach animation in the capstone program at University of Washington for over 15 years, while also working day jobs as an animator and technical artist in game development since 2000. And I think the creative joy that comes from animation is the same, whether you're working at a AAA game studio or on an indie game jam with a small team of friends.

I'm so glad to collaborate with the authors of this e-book and help both newcomers and professionals unlock the doors to animation in Unity, where interactive animation presents great opportunities for creative expression.

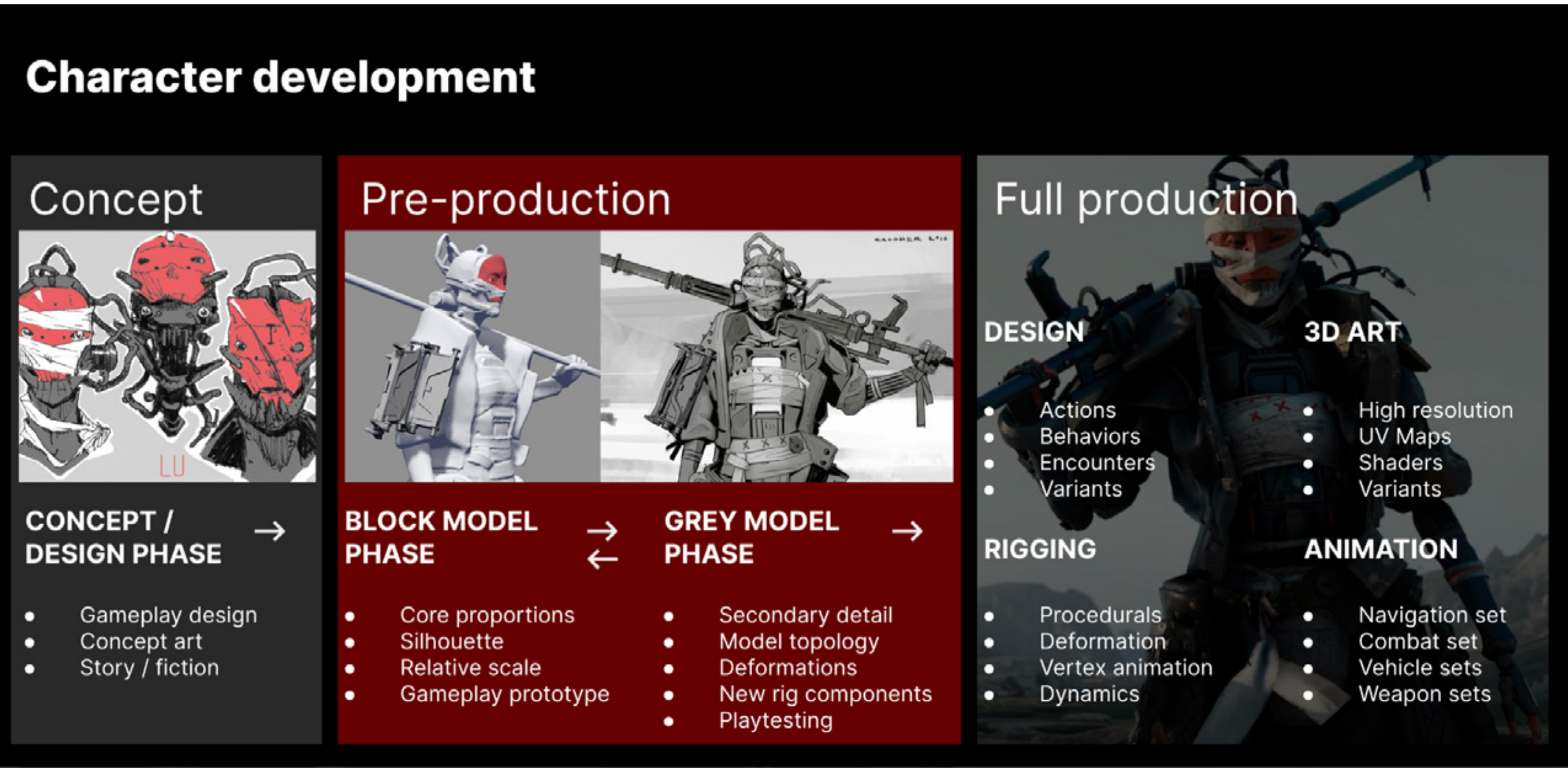
I remember well when my best friend Jay and I first downloaded the third-person platformer tutorial with Lerpz the frog character, for Unity 3. We had a book with a chapter that included the few lines of code we needed to take player input and move the character forward while playing the walk animation clip. And with that we were off to create our own stories and worlds and interactive experiences, with only our imagination as limits. We were both veteran animators and game developers at the time yet we still benefited from that brief introduction that helped us to get started in this new context. I hope that you too will find what you need within these pages to bring the tools of Unity animation within your command, and I would be so excited to see what kinds of creative works you can bring to life.

Creating animation for ambitious productions

Many games incorporate a rich variety of characters, twists and turns in their storyline, multifaceted environments – all of it often created by an equally diverse team of developers and artists. Such ambitious projects will benefit from a solid animation pipeline that provides content creators with the tools that make sense for the project, enabling them to manage potentially thousands of animations.

The process of moving from character concept to modeling, rigging, and animating is a well-trod sequence for character creation in game development. But it's not perfect and in general, artistic productions are rarely a straightforward process!

For example, a character concept could include legs that are too long to create a natural-looking locomotion cycle, or a composition of the arms and torso that doesn't allow the character to hold their weapon in a realistic way. Ideally, the character creation process should allow for non-destructive iteration on character designs. Dave Hunt and Forrest Söderlind give us a glimpse of how they structured their work in their GDC 2015 talk. They developed a system that worked well for their large-scale production. This system included using a modular, procedurally-controlled rig building framework in Maya that enabled gameplay, storytelling and worldbuilding to come together efficiently. You can see some of the key steps in their process in the image below. This workflow could be done in Unity as well.



Character development

Concept

CONCEPT / DESIGN PHASE →

- Gameplay design
- Concept art
- Story / fiction

Pre-production

BLOCK MODEL PHASE ⇄

- Core proportions
- Silhouette
- Relative scale
- Gameplay prototype

GREY MODEL PHASE →

- Secondary detail
- Model topology
- Deformations
- New rig components
- Playtesting

Full production

DESIGN

- Actions
- Behaviors
- Encounters
- Variants

3D ART

- High resolution
- UV Maps
- Shaders
- Variants

RIGGING

- Procedurals
- Deformation
- Vertex animation
- Dynamics

ANIMATION

- Navigation set
- Combat set
- Vehicle sets
- Weapon sets

Main stages of character development, as show in the GDC 2015 talk, character images are from the Unity demo Adam

Enabling animators

In game development it's often the technical artist that enables efficient collaboration between artists, developers, and designers.

Scripting, rigging and technical art are disciplines that can allow a larger production to maintain thousands of files with automated processes.

Here are a few general tips for organizing your animation files, but ultimately every project is unique and the goal should always be to make a pipeline that works for your production.

— **Naming conventions**: Characters are made up of many objects, geometry, bones, and accessories. A good practice is to standardize names so everyone in the team knows how to navigate the hierarchies. Find a good balance between simplicity and readability. Standardized, easy-to-understand naming can also be useful if custom tooling has been created for the animators on your team.

— **Scene organization**: If you are an animator planning to work directly in Unity, consider making a sandbox scene and/or working in prefab mode for common gameplay animations.

— **Asset version tracking and automation:** The AssetPostprocessor class in Unity enables you to run code upon asset import or apply automated import settings with Presets. This can be useful to efficiently verify that assets comply with the team's standards, enabling you to focus more on the actual content creation.

— **Mock up in Unity and use the FBX Exporter:** Designers can mock up animations and cinematic sequences in Unity with systems like Timeline that make it efficient to express general intent and timing. Prototyped animations can then be exported via the FBX Exporter to the animator's preferred DCC software for further polishing.

— **Visualizers and custom Editor tools**: Unity offers great flexibility through Editor tooling, allowing you to create custom interfaces for your needs, like visual control rigs, or other tools that make it easier for artists to use many of the animation tools' APIs in Unity.

— **IK in Unity with the Animation Rigging package:** Enabling runtime rigging enables your characters to be in contact with the game world. Sweeping a hand across a surface or gripping and turning a door handle are just a couple of examples of subtle movements that require modifications in the bone chain to make them look realistic. Unity's Animation Rigging package will allow you to create these detailed movements, making it a great addition to your character creation process.

Unity's animation system

If you are new to Unity, the one of the best sources for learning how to navigate in, and use, the Editor is with Unity Learn tutorials, specifically, the three Pathways courses: Unity Essentials, Junior Programmer and Creative Core.

The Unity Manual is also an important resource and includes a comprehensive section on Animation. The following short intro is from the manual.

Unity's animation system provides the following features:

— Easy workflow and setup of animations for all elements of Unity including objects, characters, and properties

— Support for imported animation clips and animation created within Unity

— Humanoid animation retargeting – the ability to apply animations from one character model onto another

— Simplified workflow for aligning animation clips

— Convenient preview of animation clips, transitions and the interactions between them

— Management of complex interactions between animations with a visual programming tool

— Animating different body parts with different logic

— Layering and masking features

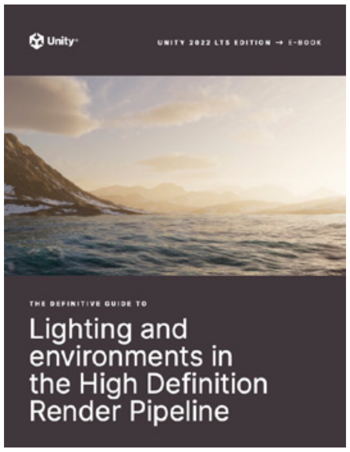
The screenshots in this guide are made using the Universal Render Pipeline (URP), Unity's multiplatform rendering solution built on top of the Scriptable Render Pipeline (SRP) framework. URP is one of three render pipelines available in Unity, the other two being the High Definition Render Pipeline (HDRP) and the Built-In Render Pipeline.

Learn more about URP, HDRP, and Unity toolsets for artists, technical artists, and designers in the following e-books:

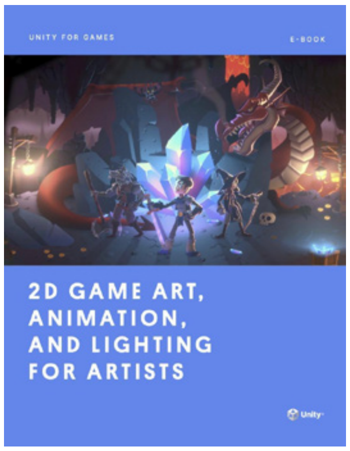
Download

Unity
UNITY 2022 LTS EDITION → E-BOOK
INTRODUCTION TO THE
UNIVERSAL RENDER PIPELINE FOR ADVANCED UNITY CREATORS

Download


Unity
UNITY 2022 LTS EDITION → E-BOOK
THE DEFINITIVE GUIDE TO
Lighting and environments in the High Definition Render Pipeline

Download

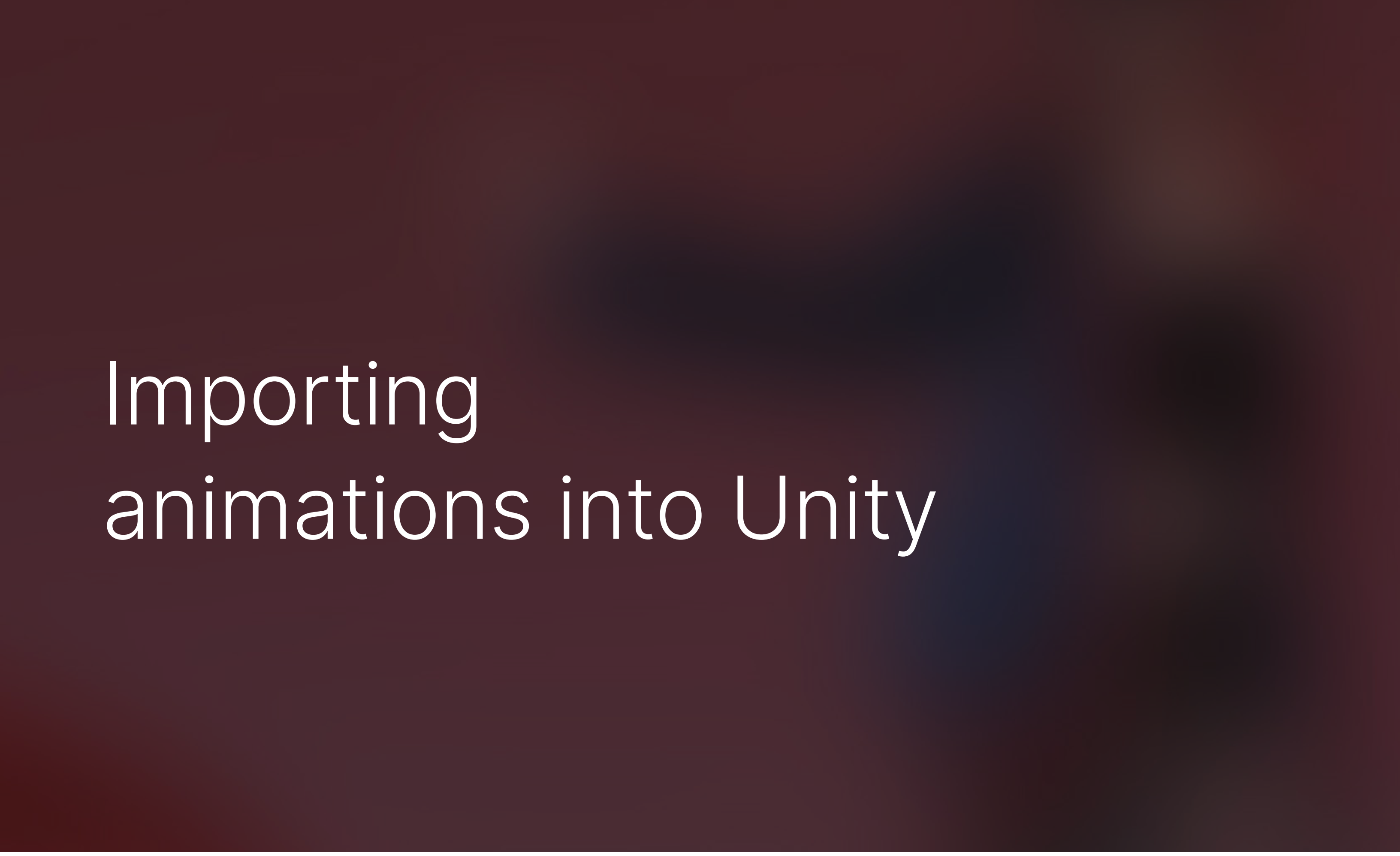


Download

UNITY FOR GAMES
E-BOOK
THE DEFINITIVE GUIDE TO CREATING
ADVANCED VISUAL EFFECTS IN UNITY
2021 LTS EDITION
Unity

Download

Download



Importing animations into Unity

Many studios use software like Autodesk Maya or Blender to create animations that can then be imported into Unity.

Large professional projects often use motion capture, employing actors to perform the motions that will result in realistic movement in the content. Actors wear motion capture suits that have markers spaced across them at key points aligned to a 3D skeleton. High-speed infrared cameras capture the actor's motion, with the data then passed into software like Autodesk MotionBuilder. Face motion is captured in the same way, with markers spread across the actor's face to record realistic facial movement.

After this, you can import the animations into Unity.

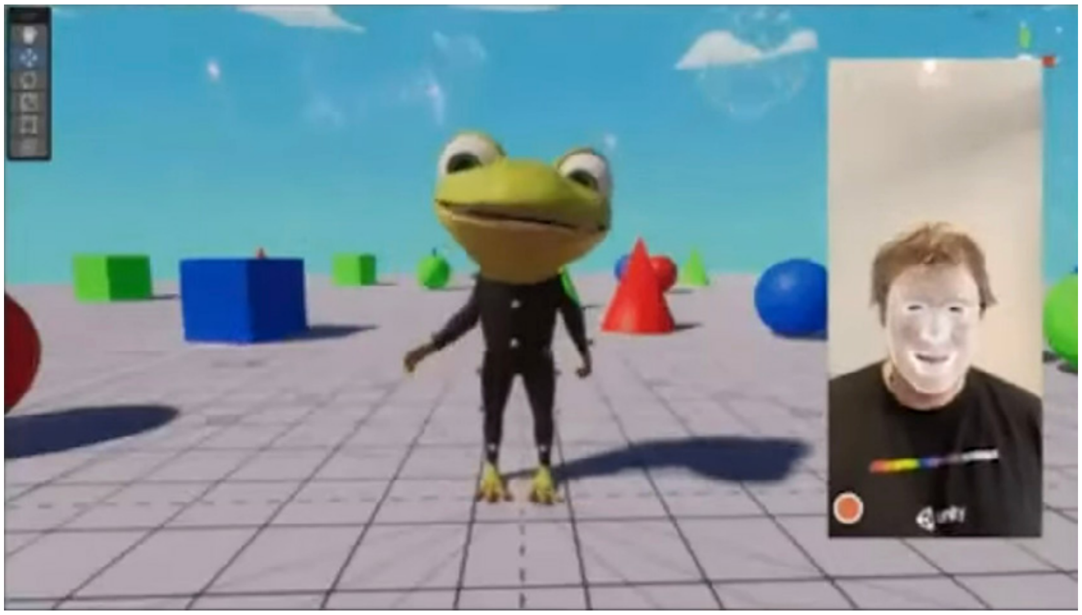
Unity's demo team in Stockholm used motion capture to create the animated shorts *Adam*, *The Heretic*, and *Enemies*.

Realistic human face and body movement in games and other real-time 3D content often relies on motion capture.

In this blog post, animator Krasimir Nechevsky discusses the process of previs, motion capture, and animation for Unity's *Adam*.

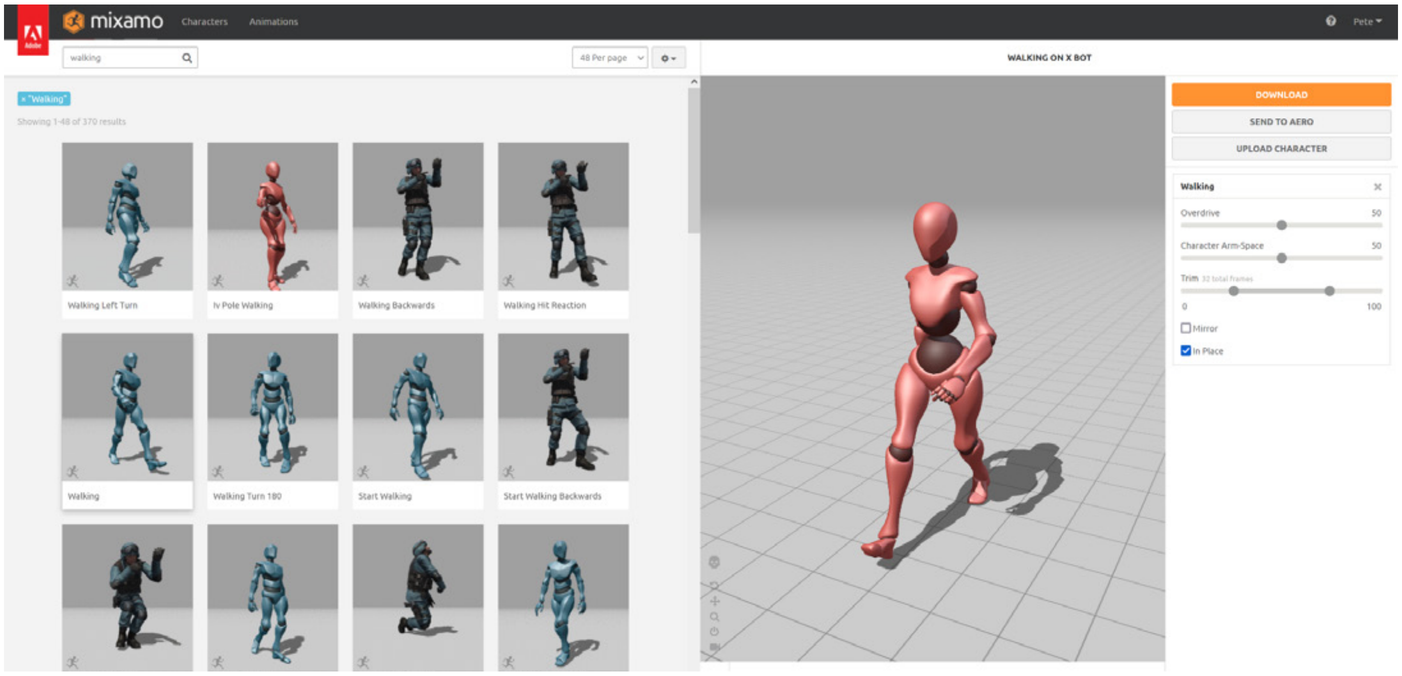
A behind-the-scenes shot from the making of the Unity short *Adam*

You can also use Unity's ARKit, together with an Apple iPhone, to capture facial movements and expressions to control a character's facial rig and mouth movement. See the Unity project The Realtime Rascals for a demonstration of this method.



Live face capture to control a character's face

Motion libraries



Walking

Overdrive 50

Character Arm-Space 50

Trim 0 100

☐ Mirror

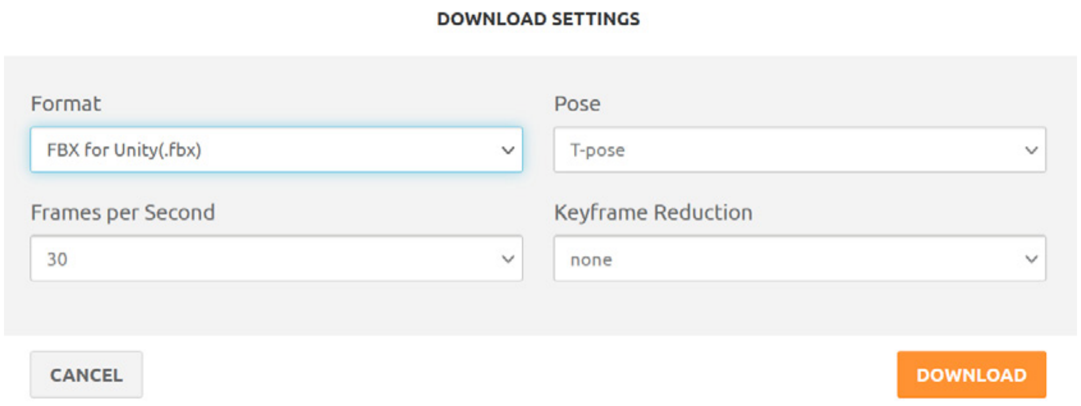
☑ In Place

Adobe Mixamo motion library

If you don't have access to a motion capture suit and motion capture software, you can use premade animated clips from motion capture libraries. These can save you time and be effective for animations on a smaller budget, but you probably won't achieve the same level of fidelity possible with the performance of a real actor.

Adobe Mixamo and Reallusion ActorCore are motion library websites that feature thousands of small motion clips that can be downloaded into your 3D software of choice, including Unity.

In addition to having a range of free characters to choose from, you can upload your own character. If each character has a different skeletal structure the animations that play on one of them might not play on another, unless they are using a humanoid animation type. If you plan to use a humanoid animation type ensure that you download the T pose to begin with, as this is necessary for the humanoid setup (this is covered in a later section of the guide).



DOWNLOAD SETTINGS

Format: FBX for Unity(.fbx)

Pose: T-pose

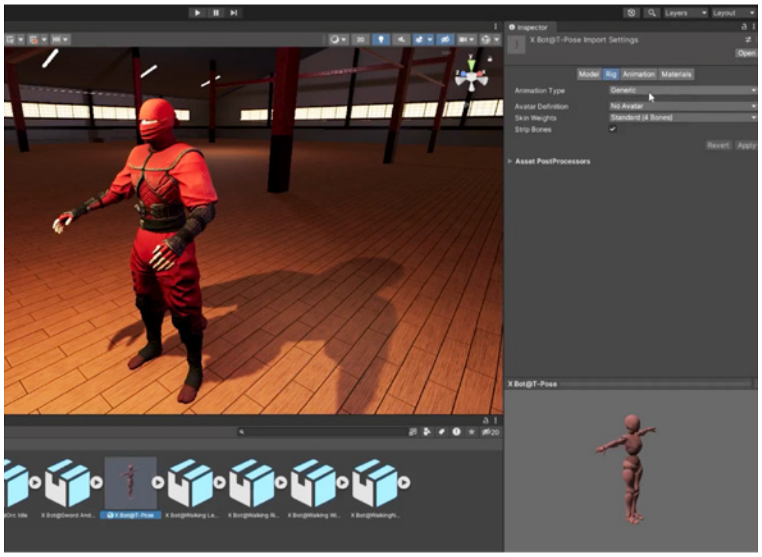
Frames per Second: 30

Keyframe Reduction: none

When you download the models from Mixamo you need to select the FBX for Unity format and T-pose settings.

The Unity Asset Store is a great place to find premade animations. Search the animation category and download packages to Unity through the Package Manager. The animations should be set up automatically and ready to use.

In addition to premade animations you can also download ready-made tools to enhance the quality of your project. Sakura Rabbit, whose character and animation work is seen in the image above, uses Unity Asset Store packages like Final IK, MagicaCloth, and Face Capture.



Video tutorial

Work with humanoid animations

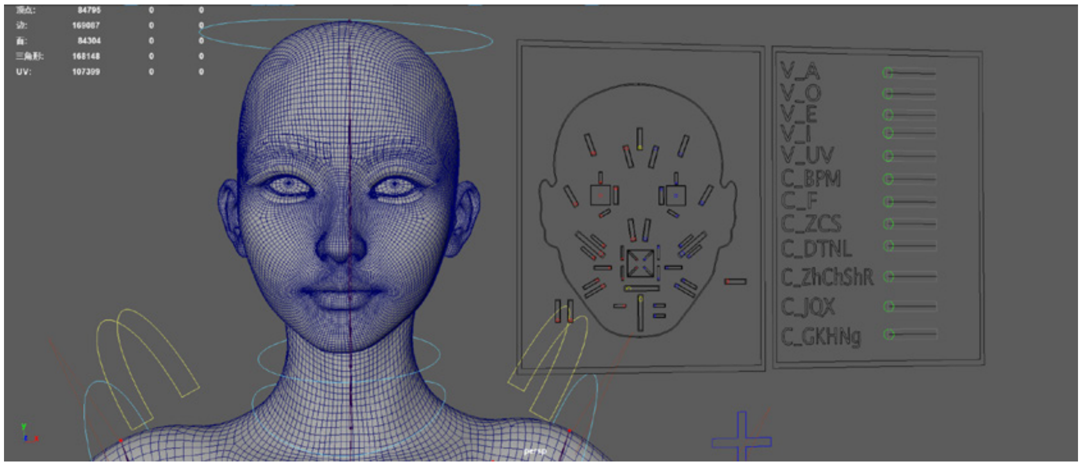
This video shows how to import downloaded animations into Unity and set them up as a humanoid animation type using root motion. Other workflows it covers include:

- Modifying animations using animation rigging, including IK.
- Using layers to combine animations.
- Setting up attributes to control animation flow.
- Setting up a character controller.
- Setting up AI characters.

Exporting animations from Autodesk Maya

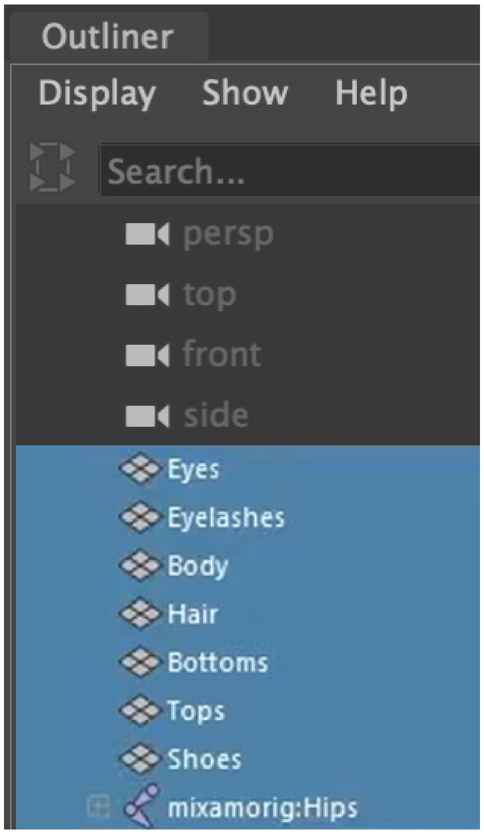
The Maya IK Handle can't be exported from Maya to Unity. Any motions controlled by the Maya rig must be baked to an animation clip before export. Maya does this automatically for you with the send to Unity option.

However, the automatic export option will make assumptions that might not apply to every situation. Often at large studios, a technical artist will use scripting to create a custom tool for exporting that uses the method best suited to their team.



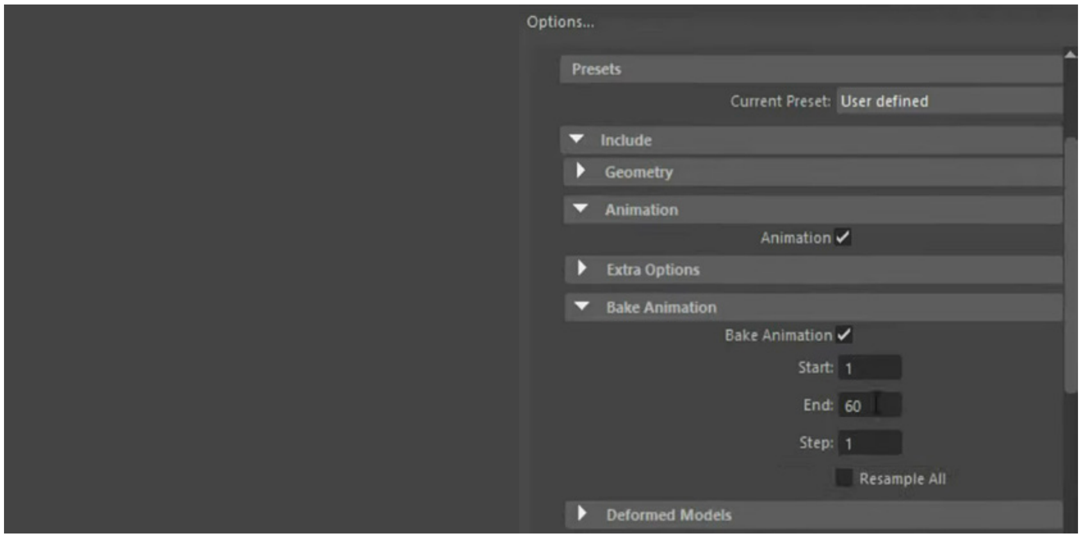
A rigging setup in Maya from a character created by Sakura Rabbit.

For more detailed control, follow these steps to manually export your character:



Selecting the full character in Maya

1. Select the character you want to export.
2. In the outliner click on hips to select the root node of the skeleton.
3. Go to **Select > Hierarchy** to select all the connected bones.
4. While holding Shift, select all the mesh objects for the character.
5. Go to **File > Export Selected**. Choose to export as an FBX file.
6. In the options you can now choose if you want to export animations. If you leave this unchecked it will export the model only. If your character is in T pose then it will be easier to set up a humanoid animation type in Unity.
7. Be sure to select the option to bake the animation if you've opted to export an animation. You can set the start frame and end frame. This bakes all motions to a keyframed animation file that will be exported alongside the 3D mesh.



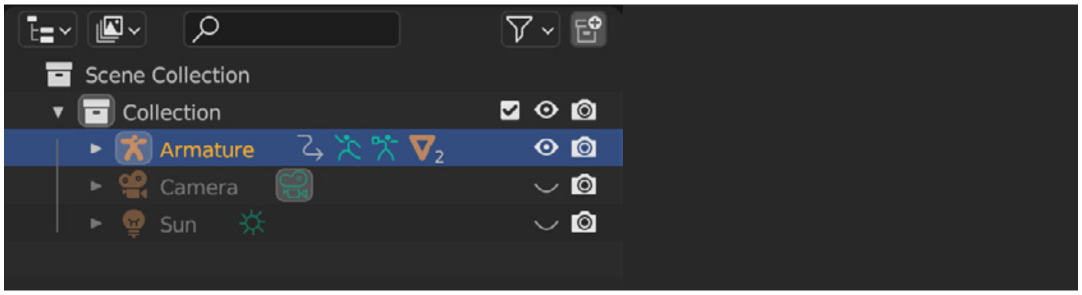
Current Preset: User defined
Animation ✓
Bake Animation ✓
Start: 1
End: 60
Step: 1
Resample All

Baking animations on export in Maya

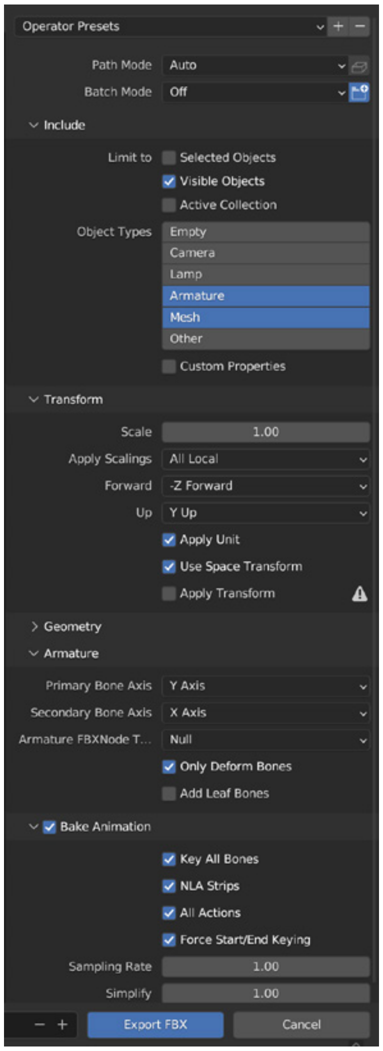
Exporting animations from Blender

Follow these steps to export from Blender:

1. Switch off any objects you don't want to export by clicking on the visibility (eye) icon that appears alongside the object. The character you want to export should be the only object still visible.



Turn off visibility for items you don't want to export



Operator Presets

- Path Mode: Auto
- Batch Mode: Off
- Include
  - Limit to: [ ] Selected Objects; [✓] Visible Objects; [ ] Active Collection
  - Object Types: Empty, Camera, Lamp, Armature (selected), Mesh (selected), Other
  - [ ] Custom Properties
- Transform
  - Scale: 1.00
  - Apply Scalings: All Local
  - Forward: -Z Forward
  - Up: Y Up
  - [✓] Apply Unit
  - [✓] Use Space Transform
  - [ ] Apply Transform
- Geometry
- Armature
  - Primary Bone Axis: Y Axis
  - Secondary Bone Axis: X Axis
  - Armature FBXNode T...: Null
  - [✓] Only Deform Bones
  - [ ] Add Leaf Bones
- [✓] Bake Animation
  - [✓] Key All Bones
  - [✓] NLA Strips
  - [✓] All Actions
  - [✓] Force Start/End Keying
  - Sampling Rate: 1.00
  - Simplify: 1.00

Export FBX | Cancel

Blender export options for animated characters

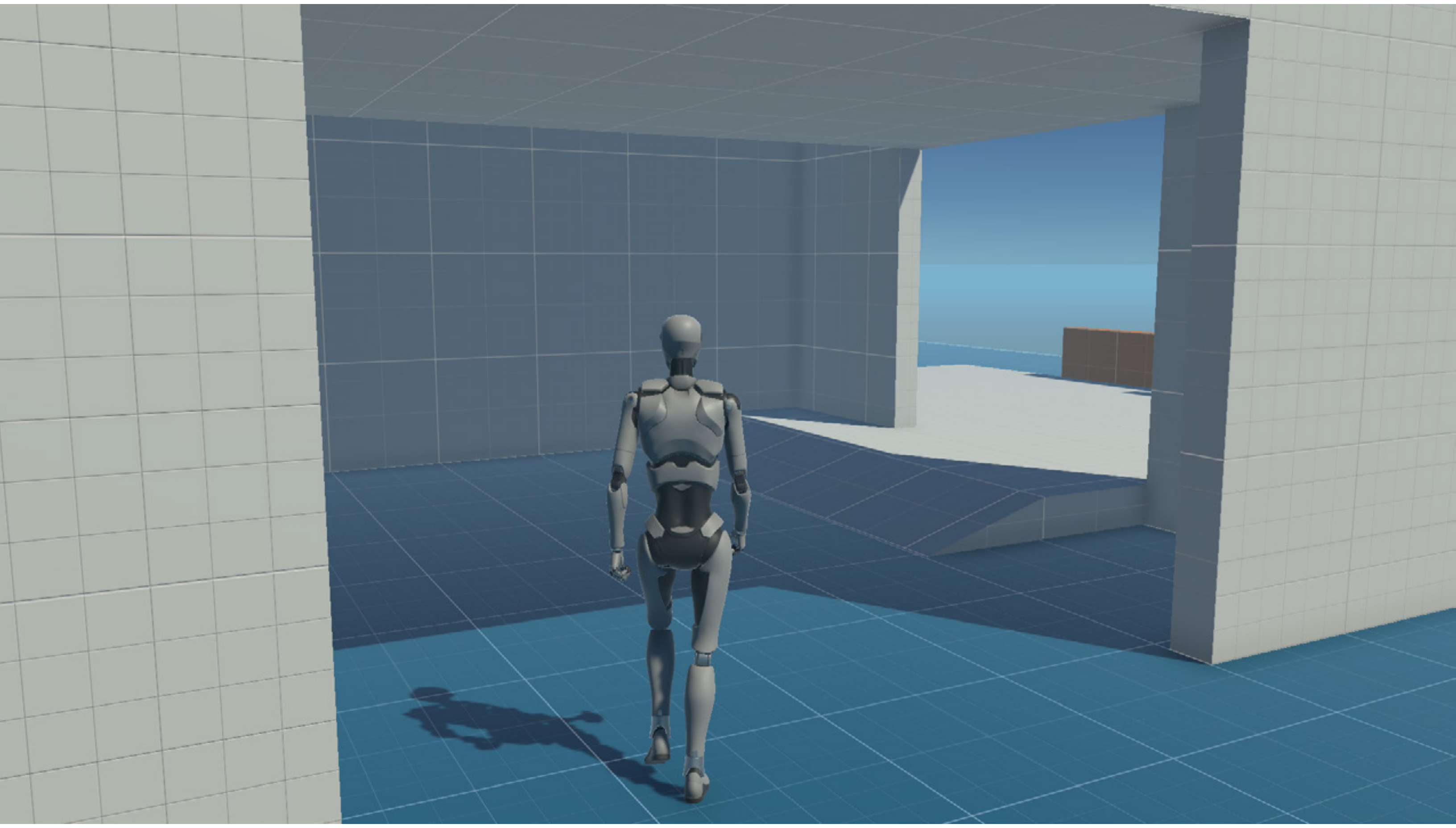
2. Select the Armature, but don't select any control rigs as they can't be exported from Blender to Unity.

3. Go to **File > Export > FBX.**

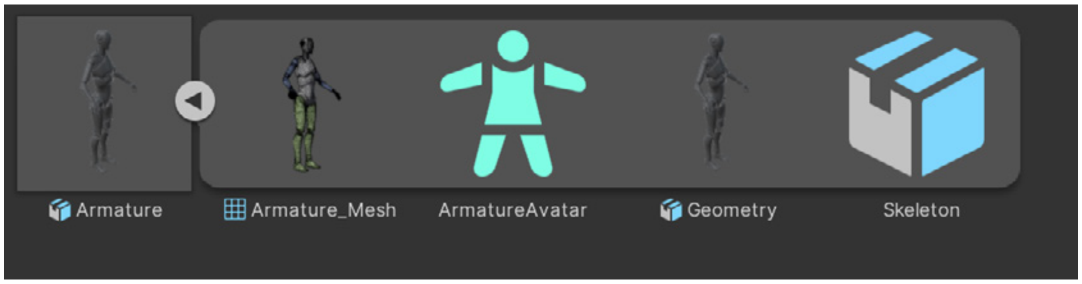
4. In the FBX export options, choose Visible Only; this ensures the control rig and any other items don't get exported in the file. Select the armature and mesh object types only (hold Shift to select multiple options).

5. Under **Armature**, choose **Only Deform Bones** and uncheck **Add Leaf Bones**. The deform bones are the skeletal bones that deform the mesh. If the **Bake Animation** box is unchecked it will export the mesh only. Clicking on Bake Animation will also export the animations that will be baked down to keyframes in animation clips, including any actions that are set by the control rig. Any animation strips in the non-linear editor will be exported as separate embedded animation clips.

Importing animations into Unity

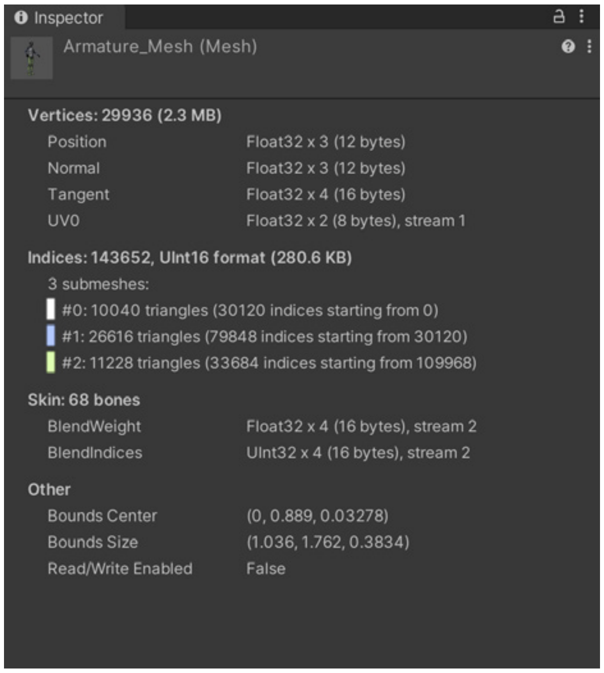

Animated character imported into Unity

FBX is the preferred format for 3D models that have embedded animations. You can import FBX files into Unity by dragging them directly from your computer's file browser into the Editor's Project window.



Structure of an FBX file (from Unity's Starter Assets - ThirdPerson CharacterController package)

The FBX file will have a number of embedded components and properties, including mesh, geometry, materials, skeleton, avatar, and animations.



Inspector

Armature_Mesh (Mesh)

Vertices: 29936 (2.3 MB)

Position	Float32 x 3 (12 bytes)
Normal	Float32 x 3 (12 bytes)
Tangent	Float32 x 4 (16 bytes)
UV0	Float32 x 2 (8 bytes), stream 1

Indices: 143652, UInt16 format (280.6 KB)

3 submeshes:

- ⬜ #0: 10040 triangles (30120 indices starting from 0)
- 🟦 #1: 26616 triangles (79848 indices starting from 30120)
- 🟩 #2: 11228 triangles (33684 indices starting from 109968)

Skin: 68 bones

BlendWeight	Float32 x 4 (16 bytes), stream 2
BlendIndices	UInt32 x 4 (16 bytes), stream 2

Other

Bounds Center	(0, 0.889, 0.03278)
Bounds Size	(1.036, 1.762, 0.3834)
Read/Write Enabled	False

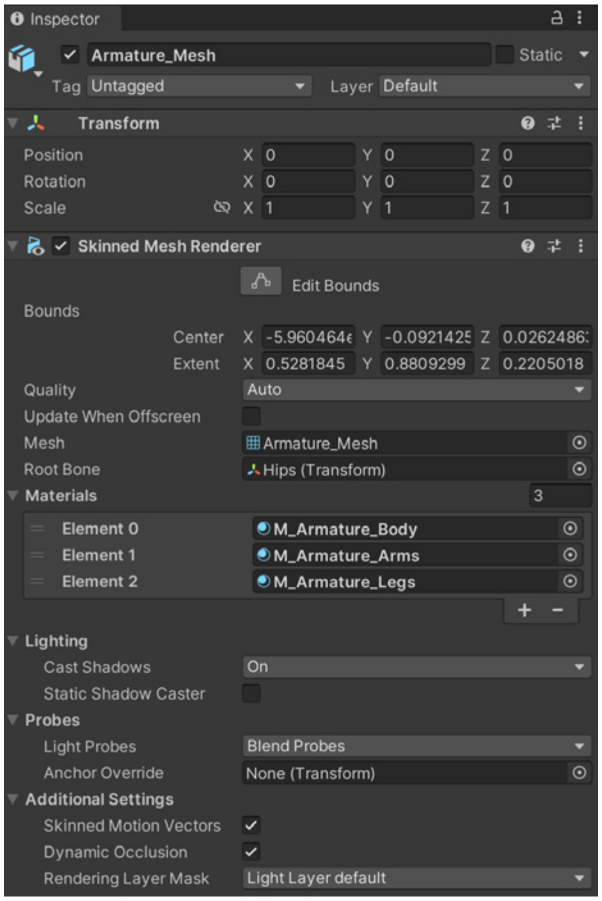
Mesh data for the FBX file

Mesh

A mesh is 3D model data that defines the shape of an object. In Unity, a Mesh asset holds all the vertices, normals, blend shapes and UV data of a model.

Geometry

Geometry is the physical model powered by the mesh. This contains a Skinned Mesh Renderer component that defines how this looks in the scene and is skinned to the skeleton, so it moves as the bones of the skeleton move. Materials can be assigned to the geometry in the Skinned Mesh Renderer component.



Inspector

✓ Armature_Mesh — Static

Tag: Untagged — Layer: Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

✓ **Skinned Mesh Renderer**

Edit Bounds

Bounds

	X	Y	Z
Center	-5.960464ε	-0.0921425	0.0262486:
Extent	0.5281845	0.8809299	0.2205018

Quality	Auto
Update When Offscreen	
Mesh	Armature_Mesh
Root Bone	Hips (Transform)

Materials 3

Element 0	M_Armature_Body
Element 1	M_Armature_Arms
Element 2	M_Armature_Legs

Lighting

Cast Shadows	On
Static Shadow Caster	

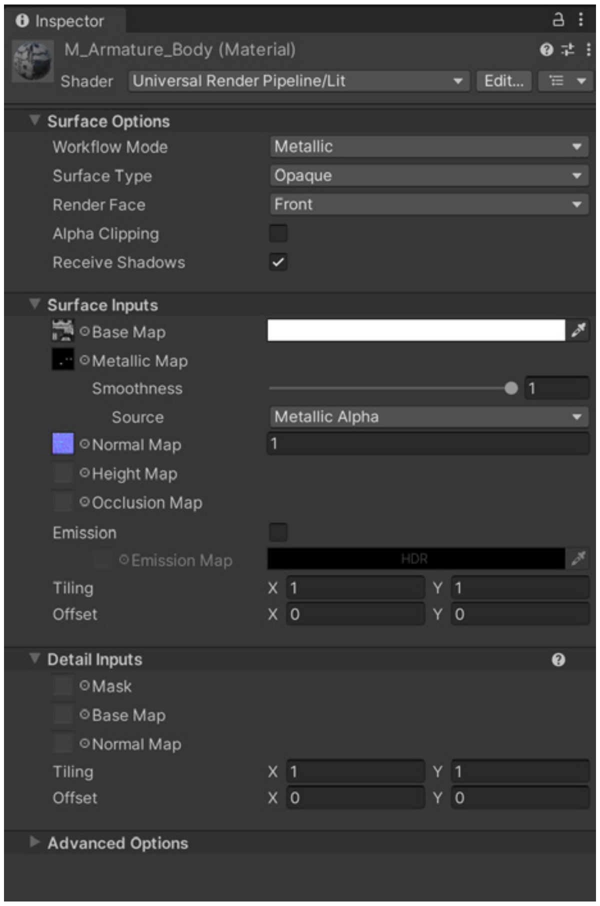
Probes

Light Probes	Blend Probes
Anchor Override	None (Transform)

Additional Settings

Skinned Motion Vectors	✓
Dynamic Occlusion	✓
Rendering Layer Mask	Light Layer default

Geometry for an FBX file



Inspector

M_Armature_Body (Material)

Shader Universal Render Pipeline/Lit Edit...

Surface Options

Workflow Mode	Metallic
Surface Type	Opaque
Render Face	Front
Alpha Clipping	☐
Receive Shadows	✓

Surface Inputs

Base Map	
Metallic Map	
Smoothness	1
Source	Metallic Alpha
Normal Map	1
Height Map	
Occlusion Map	
Emission	☐
Emission Map	HDR
Tiling	X 1 Y 1
Offset	X 0 Y 0

Detail Inputs

Mask	
Base Map	
Normal Map	
Tiling	X 1 Y 1
Offset	X 0 Y 0

Advanced Options

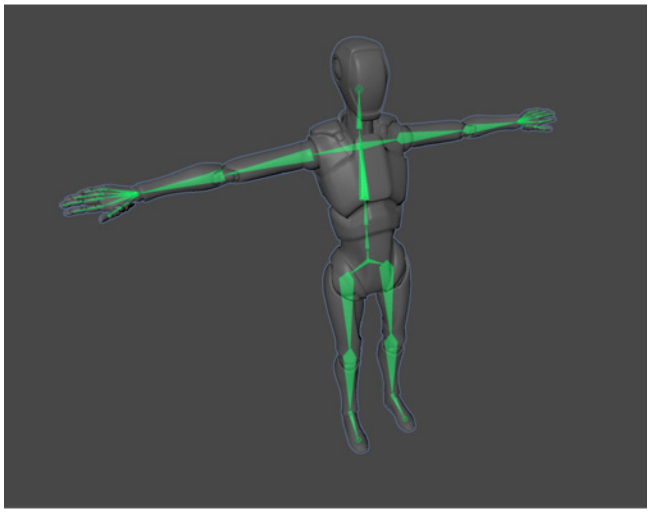
Material settings for an FBX file

Materials

Materials are assets that define how the surface of an object appears when rendered and use a Unity shader to combine textures which are applied to the geometry to define the look of the model in the scene.

Skeleton

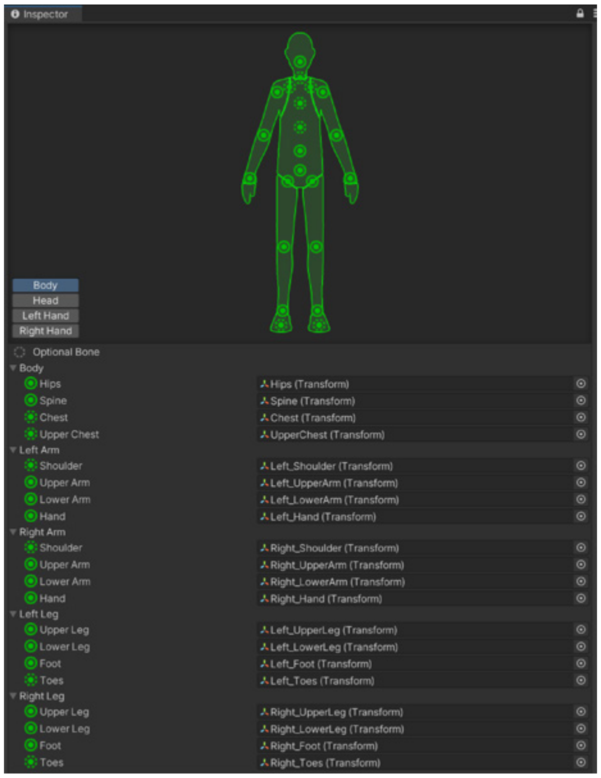
The skeleton is a series of bones (at least 15 for a biped character), that are skinned to the mesh to animate the movement of the character in the scene. The skeleton is hidden by default in the scene and game views. You can access it in the Hierarchy. If you want to visualize the skeleton, use the bone renderer explained in the Animation Rigging section later in this e-book.



A skeleton that is used to animate the FBX file

Avatar

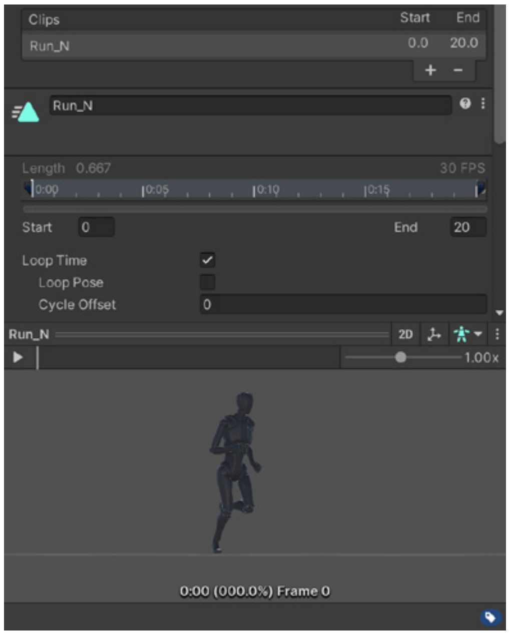
The avatar defines the bone layout for a character which defines how a character or object can be animated using the Unity Animator system. Generic avatars can be non biped structures suitable for animals and creatures.



Optional Bone

Body	
Hips	Hips (Transform)
Spine	Spine (Transform)
Chest	Chest (Transform)
Upper Chest	UpperChest (Transform)
Left Arm	
Shoulder	Left_Shoulder (Transform)
Upper Arm	Left_UpperArm (Transform)
Lower Arm	Left_LowerArm (Transform)
Hand	Left_Hand (Transform)
Right Arm	
Shoulder	Right_Shoulder (Transform)
Upper Arm	Right_UpperArm (Transform)
Lower Arm	Right_LowerArm (Transform)
Hand	Right_Hand (Transform)
Left Leg	
Upper Leg	Left_UpperLeg (Transform)
Lower Leg	Left_LowerLeg (Transform)
Foot	Left_Foot (Transform)
Toes	Left_Toes (Transform)
Right Leg	
Upper Leg	Right_UpperLeg (Transform)
Lower Leg	Right_LowerLeg (Transform)
Foot	Right_Foot (Transform)
Toes	Right_Toes (Transform)

Humanoid avatar layout defining the bone names for an FBX file



Clips	Start	End
Run_N	0.0	20.0

Run_N

Length 0.667 30 FPS

0:00 0:05 0:10 0:15

Start 0 End 20

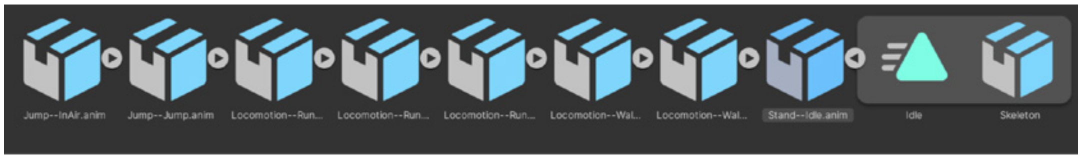
Loop Time ✓

Loop Pose

Cycle Offset 0

Run_N

Animation clips embedded in the FBX file



Animation-only FBX file

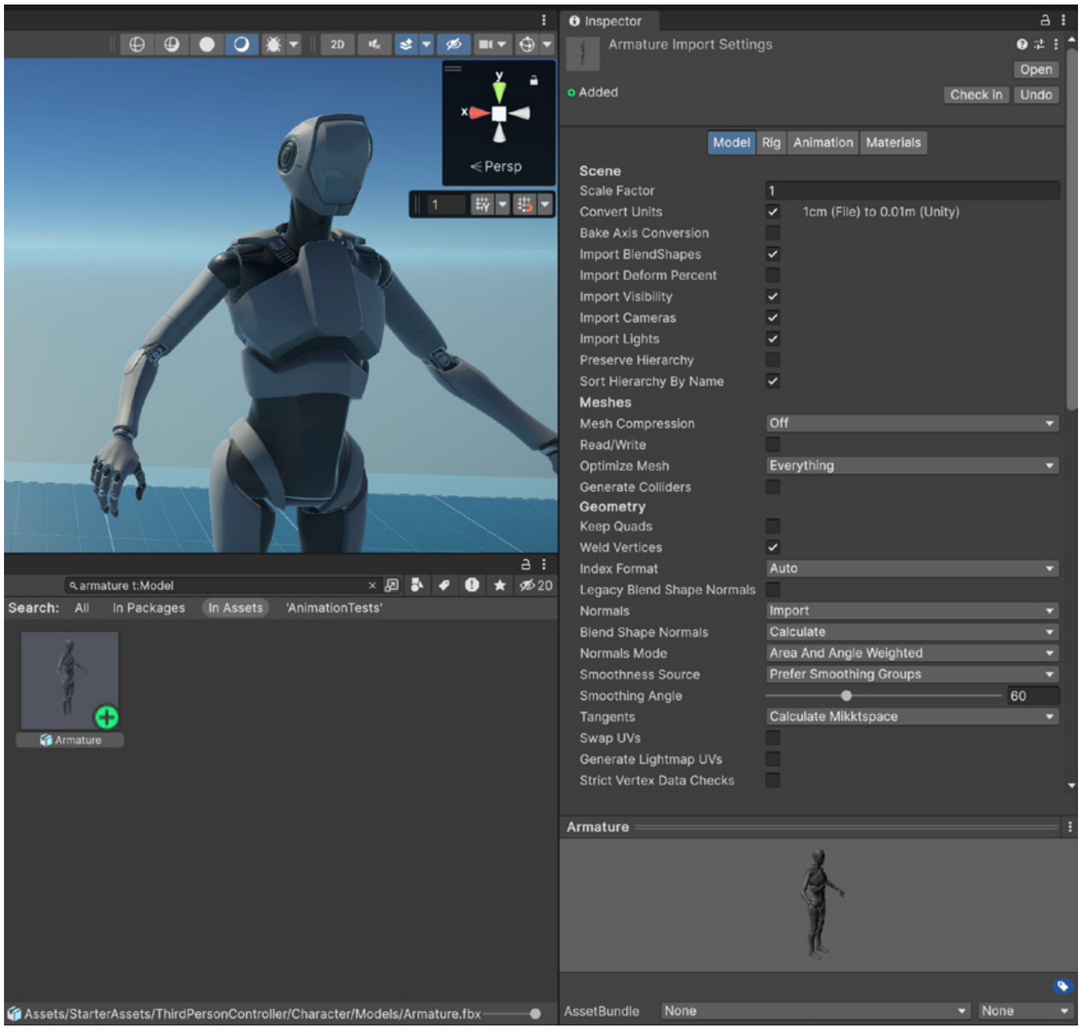
Animations

FBX files can store multiple embedded animation clips. Each animation clip stores keyframed motion data that animates the skeleton of the character.

If you export an FBX file without geometry (but with animation) it will contain a skeleton and animation clip, indicated with a triangular icon.

FBX Import Root object

The FBX Import Root object is the main FBX object that contains the settings for the FBX file.



Scene
Scale Factor 1
Convert Units ✓ 1cm (File) to 0.01m (Unity)
Bake Axis Conversion
Import BlendShapes ✓
Import Deform Percent
Import Visibility ✓
Import Cameras ✓
Import Lights ✓
Preserve Hierarchy
Sort Hierarchy By Name ✓
Meshes
Mesh Compression Off
Read/Write
Optimize Mesh Everything
Generate Colliders
Geometry
Keep Quads
Weld Vertices ✓
Index Format Auto
Legacy Blend Shape Normals
Normals Import
Blend Shape Normals Calculate
Normals Mode Area And Angle Weighted
Smoothness Source Prefer Smoothing Groups
Smoothing Angle 60
Tangents Calculate Mikktspace
Swap UVs
Generate Lightmap UVs
Strict Vertex Data Checks

Assets/StarterAssets/ThirdPersonController/Character/Models/Armature.fbx

Import settings for the FBX file

More resources from the Unity Manual:

— Model tab

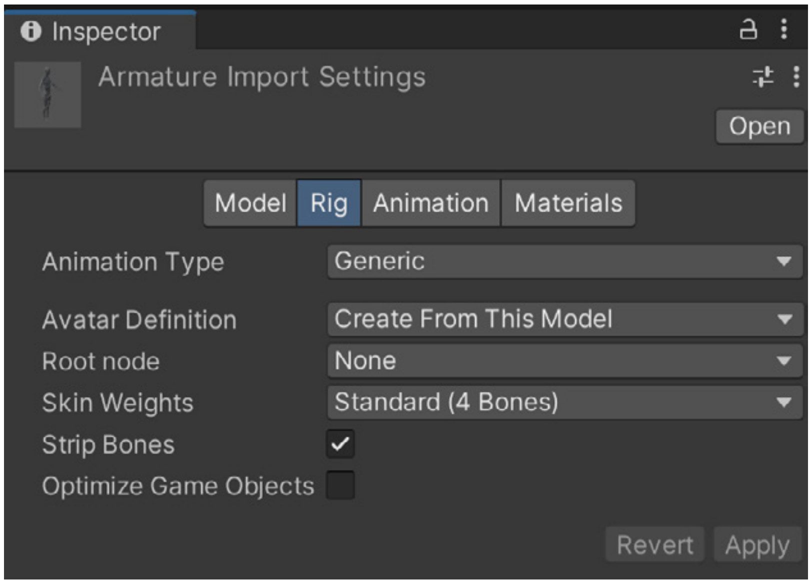
— Rig tab

— Animation tab

— Materials tab

Generic animation type

Unity uses two main animation types to control character animations, Generic and Humanoid. Humanoid provides a universal structure that can be applied to all biped characters. Generic is specific to each character's skeletal structure. The animation curves are position, rotation, and scale, and normalized muscle space.



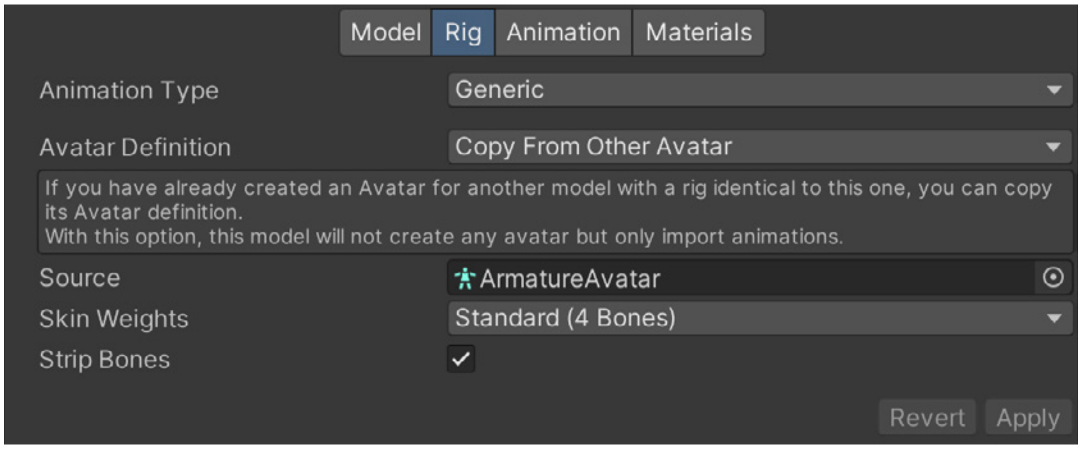
Animation Type: Generic
Avatar Definition: Create From This Model
Root node: None
Skin Weights: Standard (4 Bones)
Strip Bones: ✓
Optimize Game Objects: ☐

Rig settings for the FBX file

The generic animation type can be used for animators who want to keyframe animations directly inside Unity using forward kinematics (FK), and in cases where it's unnecessary to share animations between characters with different skeletal structures.

If you import your own FBX file it will be set to **Generic** by default. This animation type is unique to that character and its skeletal layout. However, other animations for characters that have a different skeletal setup might not work with this character when using a generic animation type, but generic animations will still work with characters that have the same skeletal layout.

You can change its animation type at any point from the drop-down menu in the Rig tab of the FBX file. You can also create an avatar from this model if it does not already have one. It's best if you can create avatars from models in the T or A pose.



Animation Type: Generic
Avatar Definition: Copy From Other Avatar

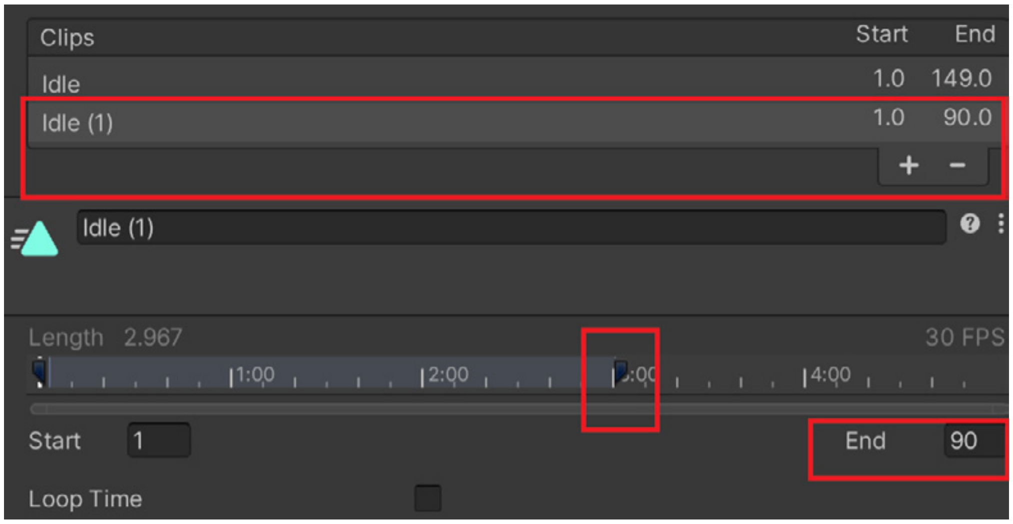
If you have already created an Avatar for another model with a rig identical to this one, you can copy its Avatar definition.
With this option, this model will not create any avatar but only import animations.

Source: ArmatureAvatar
Skin Weights: Standard (4 Bones)
Strip Bones: ✓

Animation-only FBX file rig setup

Animation-only FBX files will also use a generic animation type by default. They will need to reference the avatar from the FBX character file to ensure the animations are matched correctly to that character's skeleton.

You can define animation clips in the **Animation** tab of the FBX import settings by clicking the **+** icon beneath the clips section. Sometimes there are multiple motions within the same clip, in which case it's necessary to separate each one into their own clip. If nothing is defined, the clip is named Take01.



Clips	Start	End
Idle	1.0	149.0
Idle (1)	1.0	90.0

Idle (1)

Length 2.967 30 FPS

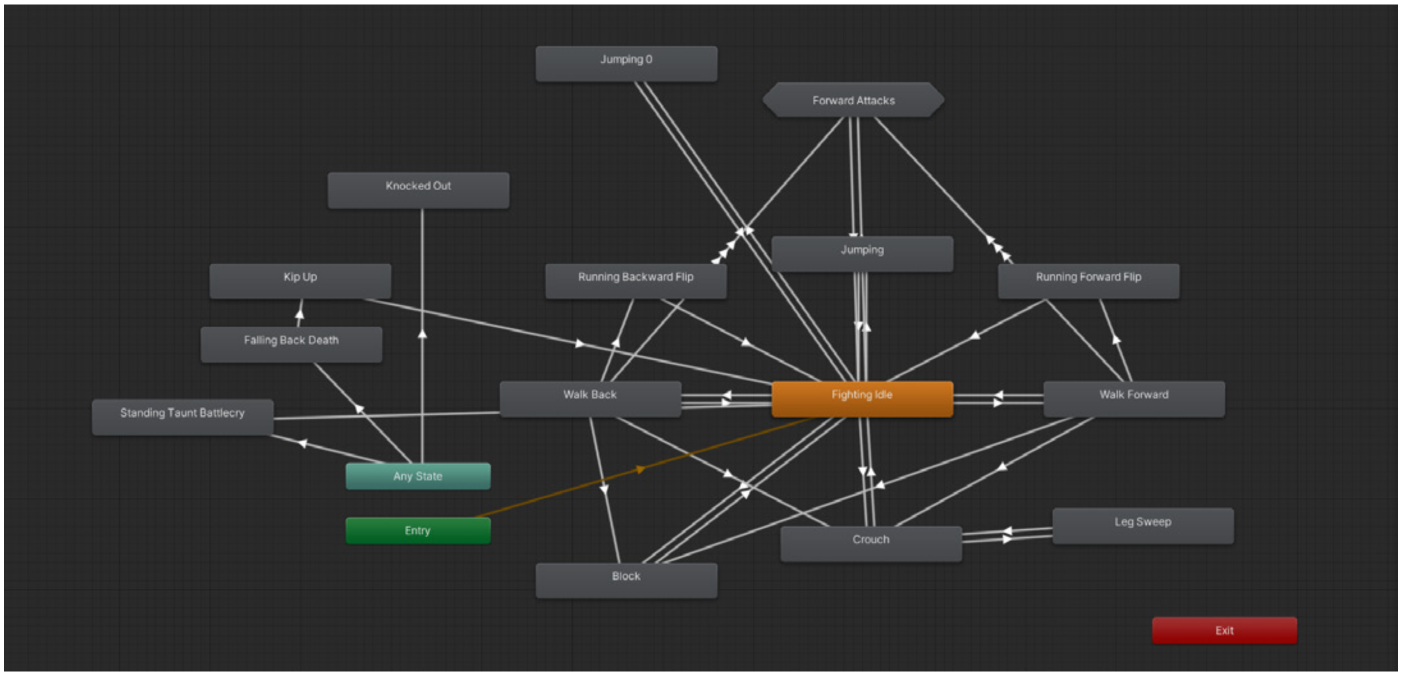
Start 1
End 90
Loop Time

Duplicating and adjusting animation lengths

You can change the timing of the clip by adjusting the in/out markers, or typing a frame number into the **Start** or **End** slot. Click **Apply** to import the animation clips to the character prefab.

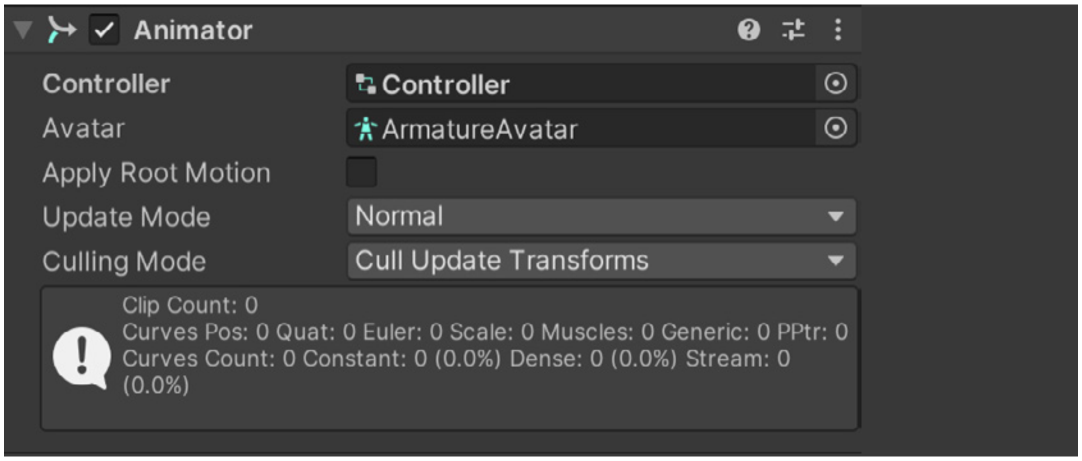
You also have the option of setting the clip to loop as well as setting a cycle offset if you find the loop is not seamless. Adjusting that by a few frames can ensure the first and last keyframe of the clip match up.

Animator Controller



An animation state machine in Unity

An Animator Controller is responsible for storing and driving all animations for a character. Right-click in the Project tab and create an Animator Controller via **Create > Animator Controller**. The character will need an Animator component to hold the Animator Controller.



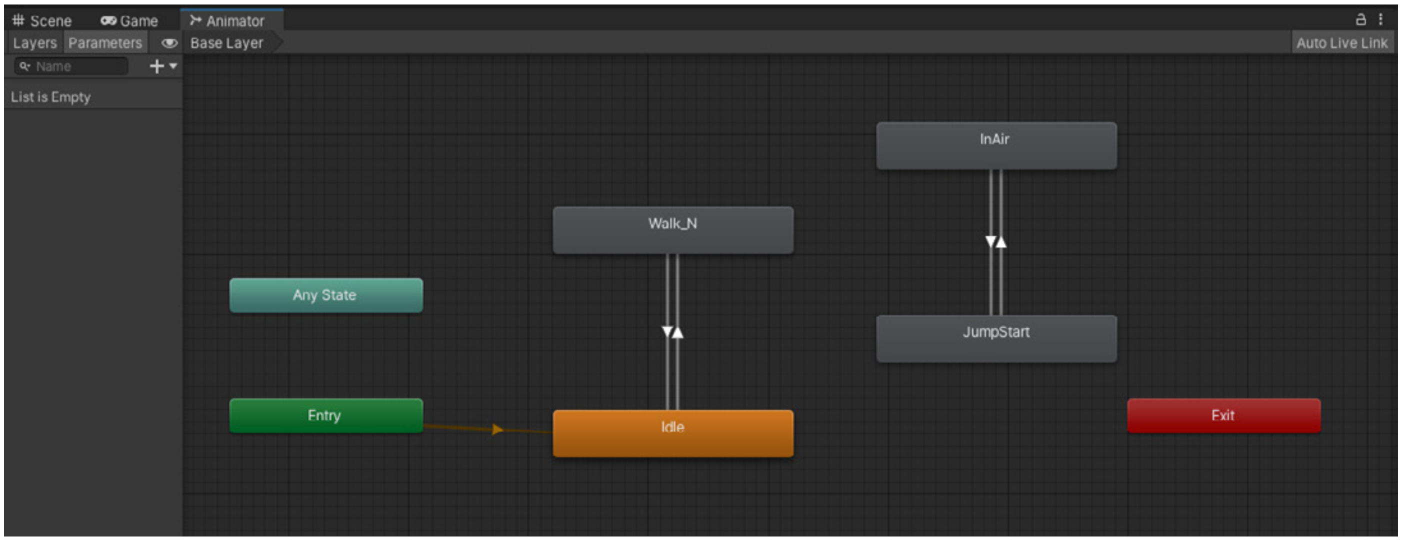
Controller	Controller
Avatar	ArmatureAvatar
Apply Root Motion	
Update Mode	Normal
Culling Mode	Cull Update Transforms

Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0 PPtr: 0
Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 0 (0.0%)

Animator component

A quicker way to achieve this is to drag a clip from the Project view onto the character prefab in the Hierarchy. This adds an Animator component to the character and the animation clip to an Animator Controller.

To access the controller, open the Animator window via **Window >Animations >Animator** or double-click the Animator Controller in the Project view. You can now drag in the animation clips to assign them to the character. Ensure that the animations match the same animation type set for the character: Generic animations for a generic character, and humanoid animations for a humanoid character.

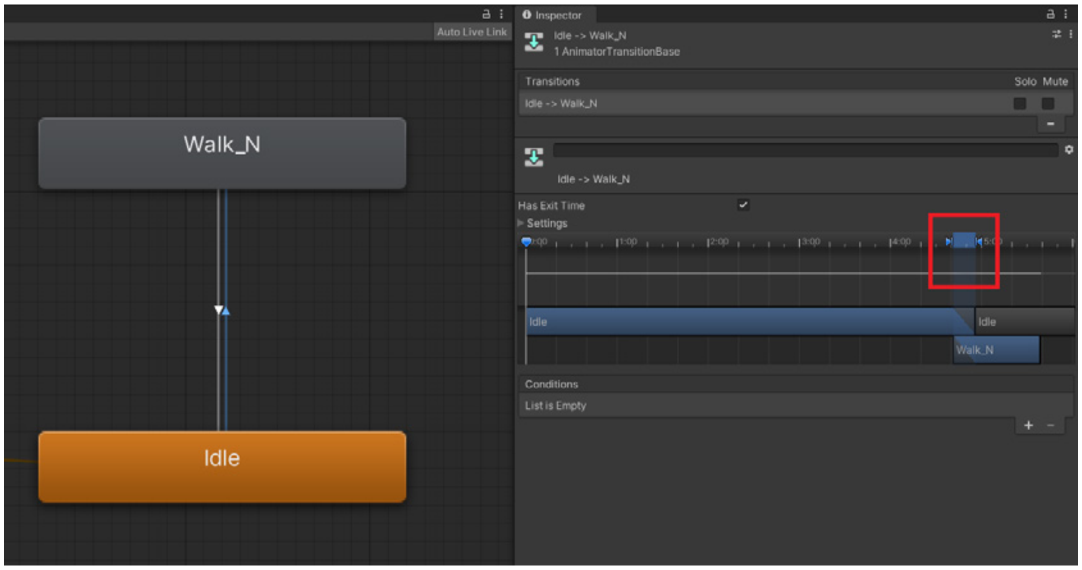


Animator window layout

The orange block seen in the screenshot of the Animator window above, is an animation state **node**. This is the default animation that will play first. Add transitions from one node to the next by right-clicking on a node, choosing **Make Transition**, and moving to the animation node of choice.

In the example above the transitions go from **Idle** to **Walk_N** and back again, creating a constant loop. The two other nodes for the jump have no direct transition to Idle or Walk_N, so they will not play. For an animation to be active it needs a transition from any node that is connected to the default state issuing from the orange node, or a transition from the **Any State** node.

Click on a transition line to see the settings for that particular transition including the conditions for when and how the transition happens.



Blend time for the transition between two animations

The small blue section highlighted by the red box in the image above is the blend time between the two animations. The longer the blend, the smoother the transition. This prevents noticeable pops when changing from one animation to the next.

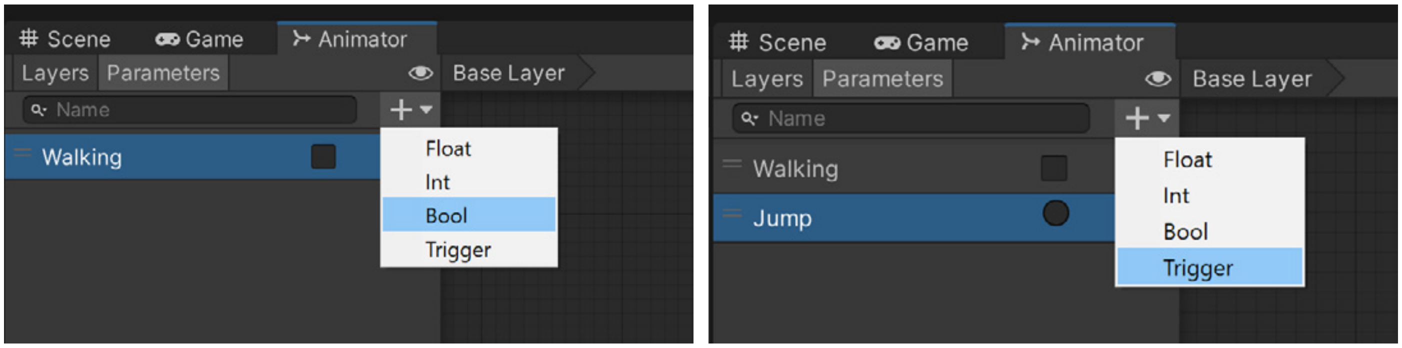
Drag the two blue arrows to extend or shorten the blend time between the two animations.

If you select the option **Has Exit Time**, the first animation will have to play entirely before making the transition to the next. Select this if there are no conditions set.

Parameters

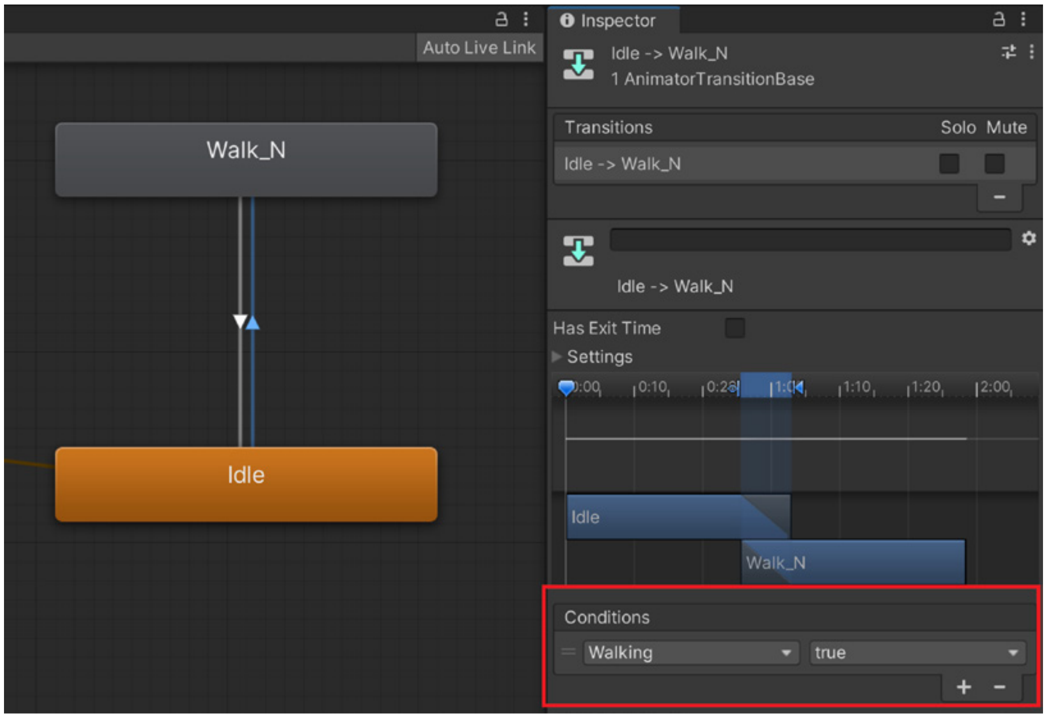
The Animator Controller needs to be interactive so the player can drive animations, for example, by pressing certain buttons. To be interactive, the controller needs parameters. These are variable inputs that can be set as conditions in the animation transitions.

Click the **+** icon to add a Bool to the parameter called Walking. A **Trigger** parameter called **Jump** can also be set.



Creating parameters in the animator window

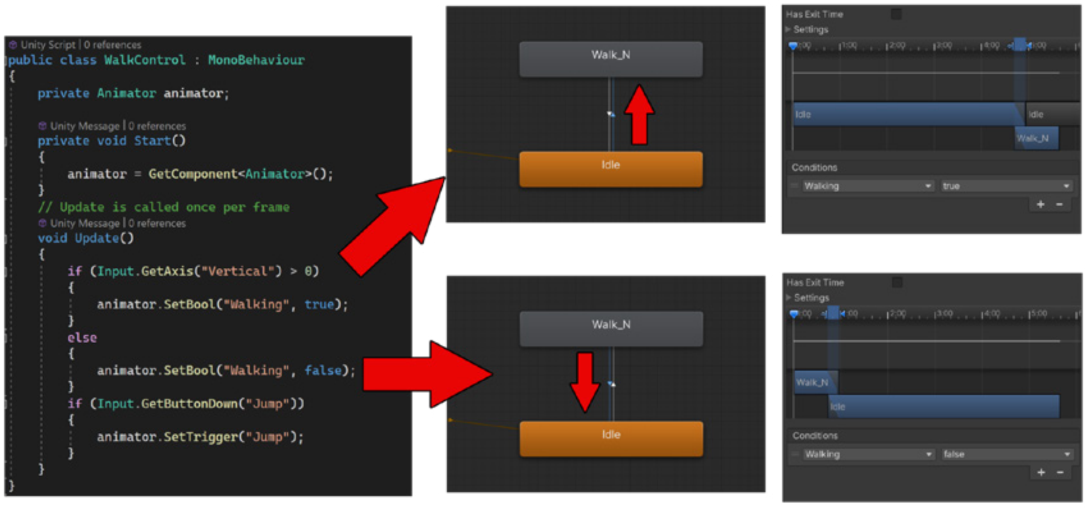
Now in the transition from Idle to Walk_N, the **Walking** condition can be set **true**. Deselect the **Has Exit Time** option and the transition happens immediately without having to play the entirety of the Idle animation.



A bool condition is set from the walking parameter to control when the transition happens.

For the transition from Walk_N back to Idle, the **Walking** condition can be set to **false**.

A C# script is needed to control the parameters for the character.



```
Unity Script | 0 references
public class WalkControl : MonoBehaviour
{
    private Animator animator;

    Unity Message | 0 references
    private void Start()
    {
        animator = GetComponent<Animator>();
    }
    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (Input.GetAxis("Vertical") > 0)
        {
            animator.SetBool("Walking", true);
        }
        else
        {
            animator.SetBool("Walking", false);
        }
        if (Input.GetButtonDown("Jump"))
        {
            animator.SetTrigger("Jump");
        }
    }
}
```

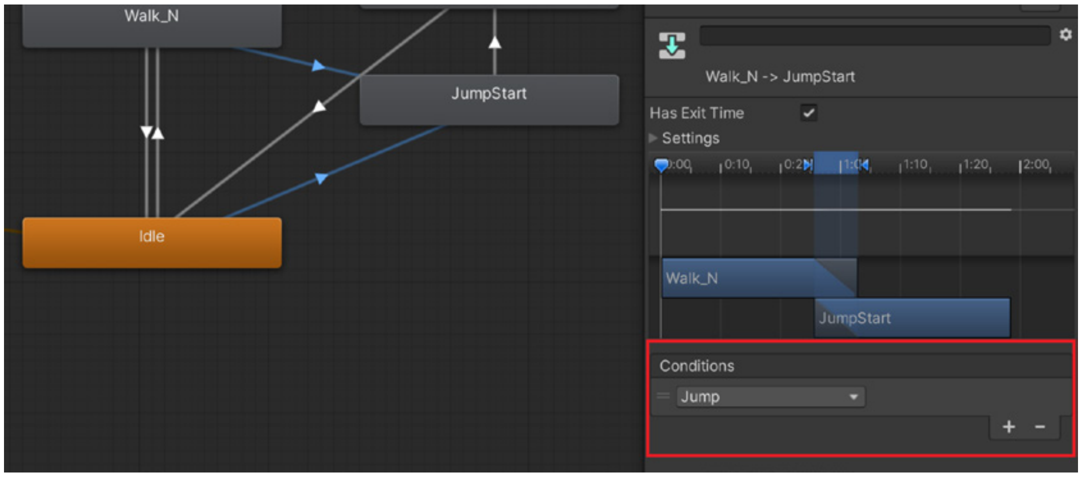
How a script controls the transitions from two nodes in the animator

The script gets the Animator component on the character in order to control it using the *animator = GetComponent<Animator>();* in *Start()*. During the Update method it waits for the player to press the W or Up arrow key (this is vertical input). When this happens the walking bool in the animator is set to **true**. Now the transition from Idle to Walk_N takes place.

It continues to play the Walk_N animation as long as the player is pressing up. When the player stops pressing up, the walking bool in the animator is set to **false**. The transition from Walk_N to Idle takes place and the Idle animation continues to play until the player pushes the Up arrow key again.

The Update method is also waiting for the player to press the jump key (by default this is set to the spacebar). When this happens the animator runs the Jump trigger that plays the jump animation once for every press on the spacebar.

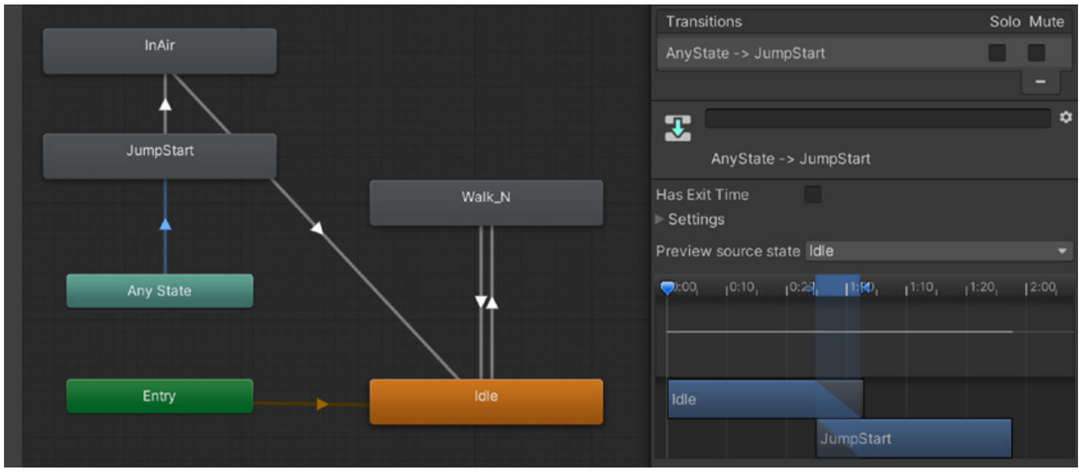
It has two transitions for the jumping animation, from both idle and walking, because you might decide to jump from either state, with the condition of both transitions set to **Jump**.



Walk_N -> JumpStart
Has Exit Time ✓
Settings
Conditions
Jump

A trigger parameter condition is set to control when the jump happens from both Walk_N and Idle.

Another way to achieve this is to have a transition from the **Any State** node. Be aware that this can then be called from any clip in the entire animator. To keep the animation loop active it has a transition back to Idle once completed.

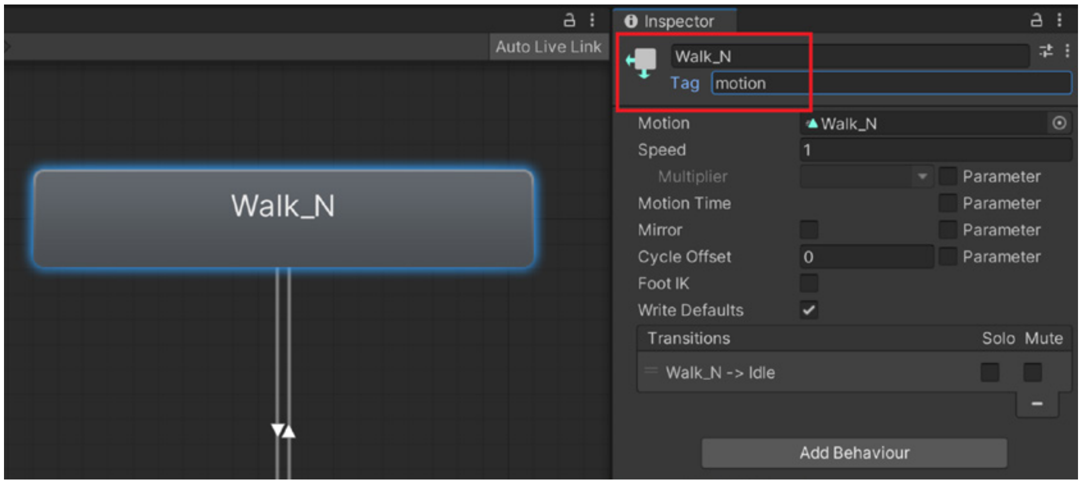


Transitions	Solo	Mute
AnyState -> JumpStart		

AnyState -> JumpStart
Has Exit Time
Settings
Preview source state Idle

Using the Any State node for transitions from any animation node

You can click on the animation node to adjust the play speed to either slow this down or speed it up. You can also assign certain animations tags in the **Tag** field, which are useful when coding. In the example below, the Walk_N animation has a motion tag, but the Idle animation has none. This will allow the code to identify when the character is in motion.



Walk_N
Tag: motion

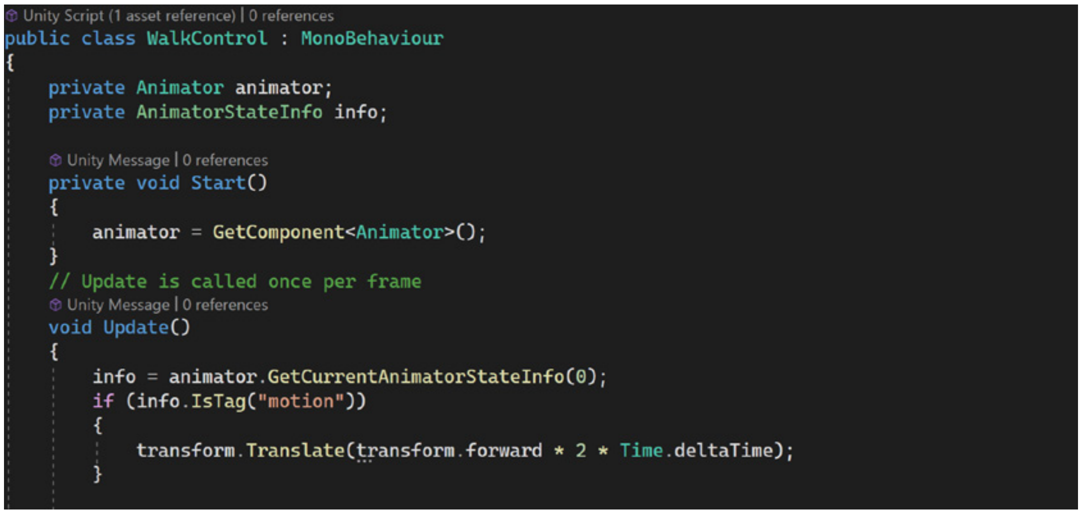
Motion	Walk_N	
Speed	1	
Multiplier		Parameter
Motion Time		Parameter
Mirror		Parameter
Cycle Offset	0	Parameter
Foot IK		
Write Defaults	✓	

Transitions	Solo	Mute
Walk_N -> Idle		

Animation state settings including setting tags

Use an AnimatorStateInfo struct to monitor animations from script. This passes information to the script every frame on the current activity of the animator. In the example below, in the Update method, it gets information from the animator on the base layer, which is layer 0.

If an animation clip with the motion tag plays, it will move the character forward using the `transform.Translate` command.



```
Unity Script (1 asset reference) | 0 references
public class WalkControl : MonoBehaviour
{
    private Animator animator;
    private AnimatorStateInfo info;

    Unity Message | 0 references
    private void Start()
    {
        animator = GetComponent<Animator>();
    }
    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        info = animator.GetCurrentAnimatorStateInfo(0);
        if (info.IsTag("motion"))
        {
            transform.Translate(transform.forward * 2 * Time.deltaTime);
        }
```

Monitoring the animator using code

The AnimatorStateInfo struct can be used to check when animations are playing or transitioning, providing you with a powerful way to time your code with the Animator component.

Layers

You can also use layers to divide up different sets of animations for a character. The different body parts of the character can use different animations, allowing you to combine multiple animations into a new unique animation.

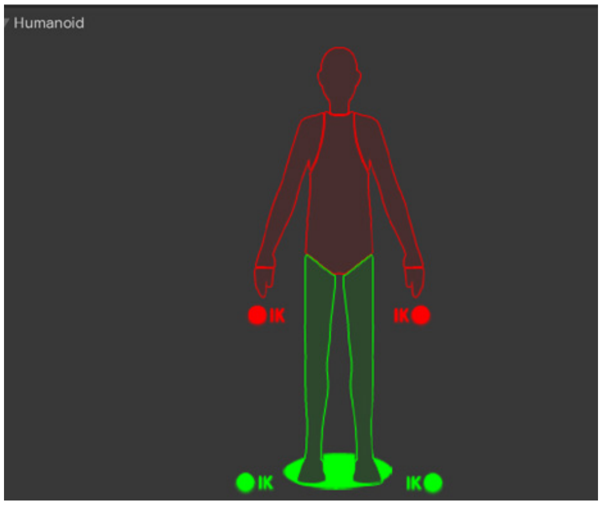
The basic walk animation is on layer 0, called the Base Layer, in the Animator:

Basic walk animation

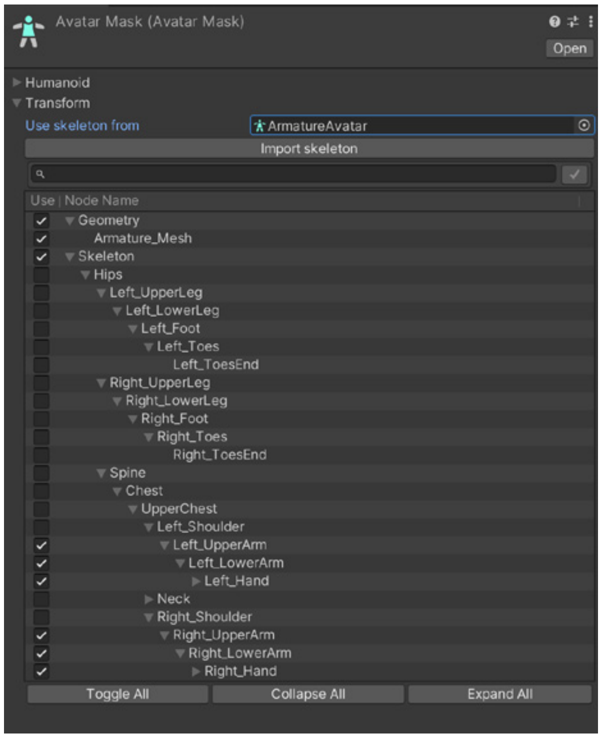
You can use an avatar mask to define which body parts will use the basic walk animation and which will use a different animation. Use them with the layer system to exclude certain body parts affected by specific animations. You can combine motions together, e.g., use the upper body from one and the lower body from another.

To create an avatar mask, right-click in the Project window and choose **Create > Avatar Mask**. In the Inspector you'll see two types to select from depending on the animation type your character is using.

A Humanoid provides you with a visual guide to the character and is only used when your character has a humanoid animation type. Click the body parts in the humanoid section to exclude them (indicated by the parts turning red).



Humanoid avatar mask setup



Avatar Mask (Avatar Mask)

Humanoid
Transform
Use skeleton from: ArmatureAvatar
Import skeleton

Use	Node Name
✓	Geometry
✓	Armature_Mesh
✓	Skeleton
	Hips
	Left_UpperLeg
	Left_LowerLeg
	Left_Foot
	Left_Toes
	Left_ToesEnd
	Right_UpperLeg
	Right_LowerLeg
	Right_Foot
	Right_Toes
	Right_ToesEnd
	Spine
	Chest
	UpperChest
	Left_Shoulder
✓	Left_UpperArm
✓	Left_LowerArm
✓	Left_Hand
	Neck
	Right_Shoulder
✓	Right_UpperArm
✓	Right_LowerArm
✓	Right_Hand

Generic Avatar Mask setup

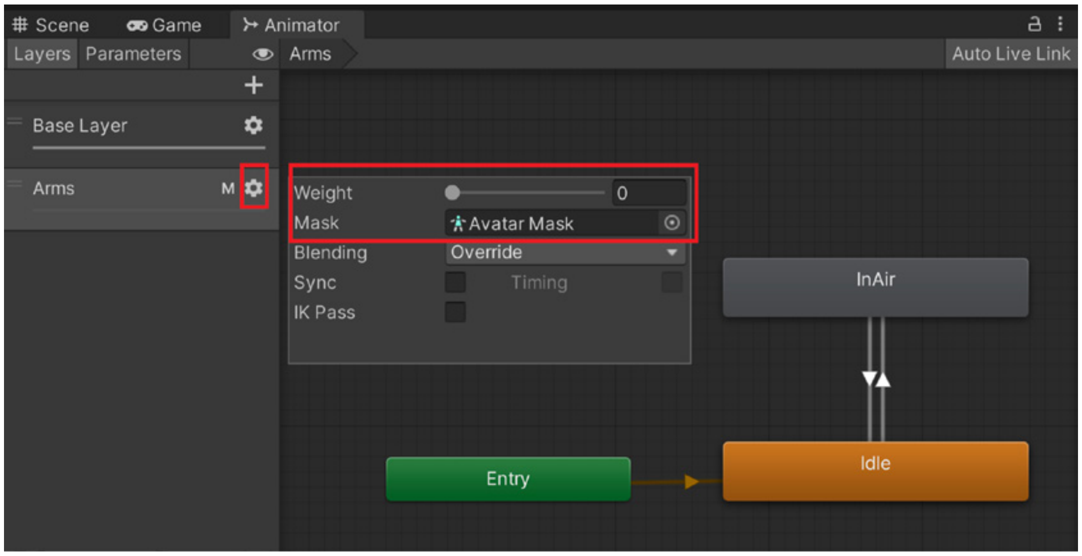
The Transform section provides you with a tree layout of the character's skeletal structure and is used with a generic animation type. It requires you to add the character's avatar and load the skeleton from that avatar.

You can expand the tree to show all bones and then use the toggle button to switch them on or off. Select items to set them to override the base animation. Unselected items are excluded, meaning the base animation will be used on those bones.

Only the character's arms are selected in the example above, so a new animation will drive the arms, while the base animation drives the rest of the character.

A new layer in the Animator is added and uses a jump animation, which has the arms in a lifted position. The transitions are the same as the Base Layer. The Avatar Mask is applied to the Mask slot which is accessed by clicking on the cogwheel Settings icon.

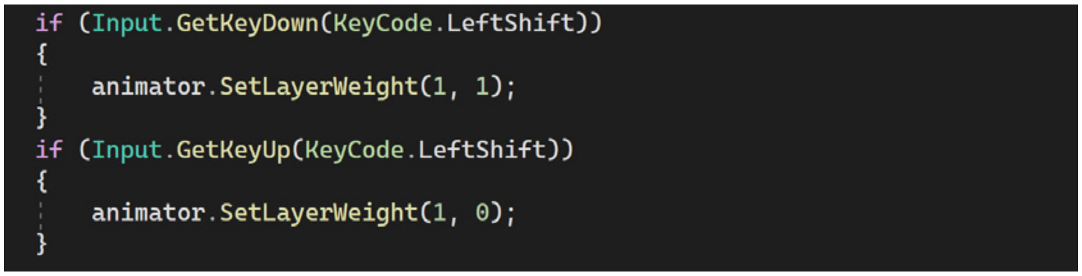
You can control this layer by its weight. At weight 0 it has no effect over the base layer. At weight 1 it has full strength and overrides the bones of the base layer. You can vary the weight to get interesting results and animate to achieve a smooth blend between animations.



Weight: 0
Mask: Avatar Mask
Blending: Override
Sync
Timing
IK Pass

Adding new layers and controlling them with an Avatar Mask

To control layer weight in a C# script you can adjust the parameters in brackets **(layer index, layer weight)**. The code example below shows the Arms layer is at index 1 and has a weight of 1 when you press Shift, and a weight of 0 when you release Shift.



```
if (Input.GetKeyDown(KeyCode.LeftShift))
{
    animator.SetLayerWeight(1, 1);
}
if (Input.GetKeyUp(KeyCode.LeftShift))
{
    animator.SetLayerWeight(1, 0);
}
```

Code to control animation layer weight

During Play mode, when holding Shift and walking, the script combines the two animations.

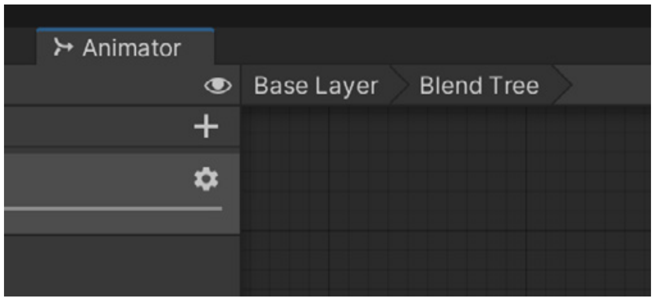
Combined animations using an Avatar Mask

Blend Trees

Use a blend tree when more than two animations need to be blended together dynamically based on input parameters. The Unity-made Starter Assets – ThirdPerson package uses blend trees to control a third-person character motion.

Right-click in the Animator Controller window and choose **Create state > From New Blend Tree** in the context menu to add a blend tree.

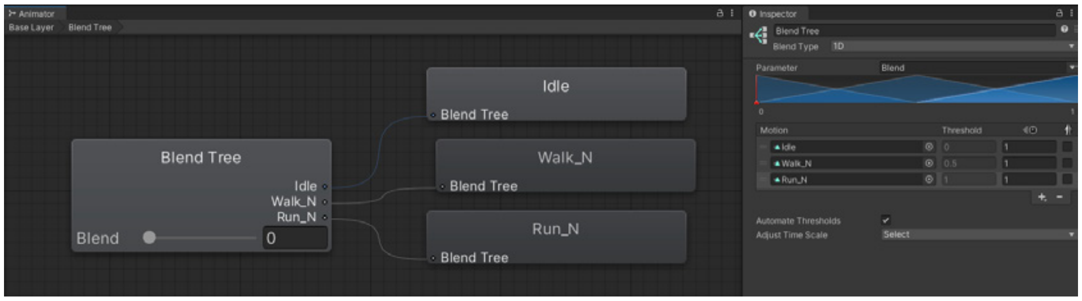
Double-click on this to enter the blend tree setup. If you want to exit the blend tree use the buttons at the top left.



Exit a blend tree by clicking on the base layer.

Click the blend tree node to add the animations that you want to blend together.

The example below blends three animations – idle, walking, and running. This is driven by a float input based on pressing the W or up arrow, which will go from 0.0 to 1.0. The animations will blend together based on that number.

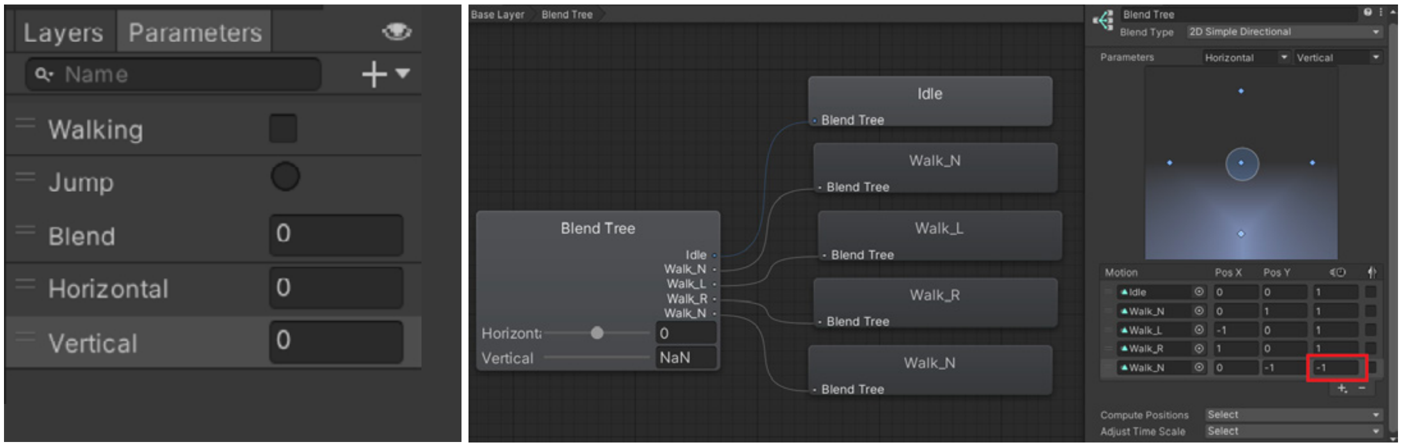


Blend Tree
Blend Type 1D
Parameter Blend

Motion	Threshold		
Idle	0	1	
Walk_N	0.5	1	
Run_N	1	1	

Blend tree setup

Blend trees can be based on a single input (1D blend), or two inputs (2D blend), for example, pressing the WASD keys to control character movement both horizontally and vertically.



Blend Tree
Blend Type 2D Simple Directional
Parameters Horizontal Vertical

Motion	Pos X	Pos Y		
Idle	0	0	1	
Walk_N	0	1	1	
Walk_L	-1	0	1	
Walk_R	1	0	1	
Walk_N	0	-1	-1	

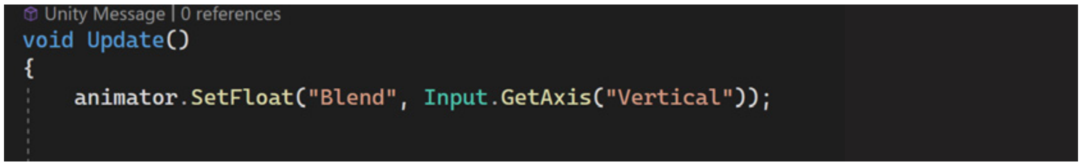
Compute Positions Select
Adjust Time Scale Select

Simple 2D blend tree

The 2D blend tree will need two parameters to drive the different directions. You can set the max values for the pos X (horizontal) and pos Y (vertical).

The image above shows that the backwards walking animation is created by reusing the Walk_N animation set at a playback speed of -1 so it plays in reverse.

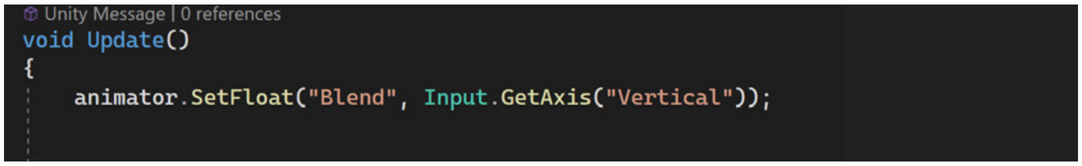
In code you can now pass the input axis of the W and S keys or up and down arrow keys into the animator float parameter (this is `Input GetAxis(“Vertical”))`. This controls the blend for a 1D blend shape.



```
Unity Message | 0 references
void Update()
{
    animator.SetFloat("Blend", Input.GetAxis("Vertical"));
```

Controlling a blend tree with code

A 2D blend requires two inputs for the two parameters.



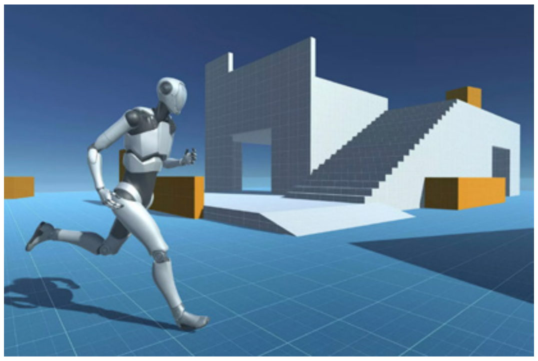
```
Unity Message | 0 references
void Update()
{
    animator.SetFloat("Blend", Input.GetAxis("Vertical"));
```

2D blend code

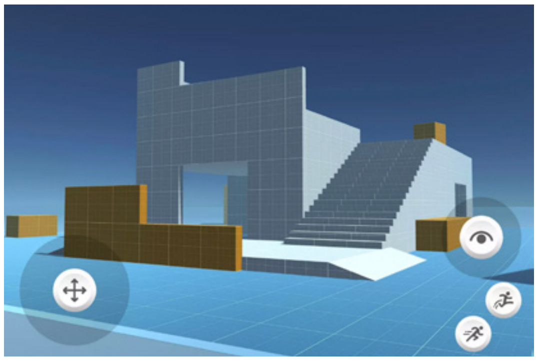
Character Controller

A Character Controller component is used to control a playable character and features a number of properties like a climb slope limit and a step height limit. Typically, a Character Controller component is used for third-person or first-person player control that does not make use of Rigidbody physics.

In addition to the Starter Assets – ThirdPerson package, you can download another Unity-made package, Starter Assets – FirstPerson, from the Unity Asset Store. Both packages provide configured character controllers for quick prototyping of your scenes.



Starter Assets – ThirdPerson



Starter Assets – FirstPerson

Let's go through the main steps for setting up your own character controller for a playable character.

1. Create a new blend tree with three animation clips: A walk or run forward, an idle, and a walk or run backward. Ensure to uncheck the **Automate Thresholds** checkbox. Set the thresholds to -1 for walk backward, 0 for idle and 1 for walk forward.
2. The results should be the following:

    a. Press the W key to set the vertical axis to 1 and make the character walk forward.

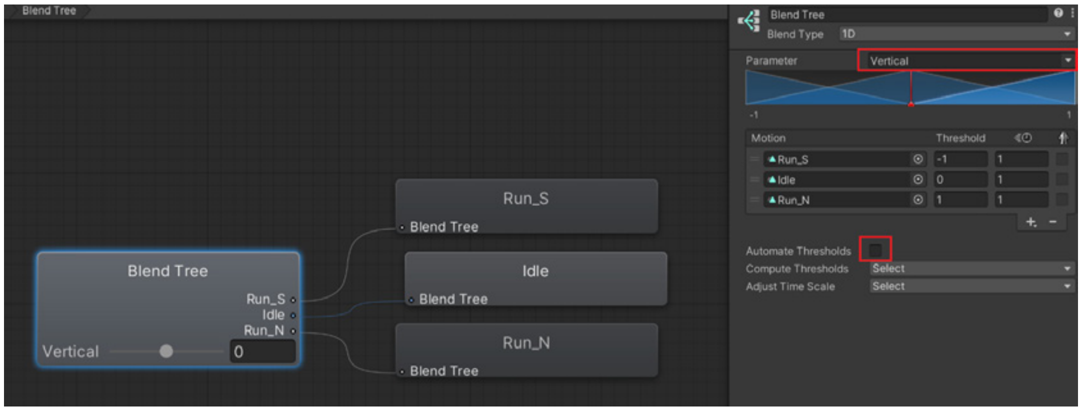
    b. Press the S key to set it to -1 to make the character walk backward.

    c. Press the A key to set the horizontal axis to -1 and make the character walk to the left.

    d. Press the D key to set the horizontal axis to 1 and make the character walk to the right.

    The blend tree will now match the correct motion to the vertical and horizontal axes.

3. Create a float parameter called `Vertical` to control the motion and set it as the parameter for the 1D blend, as seen in the image below:



Blend Tree

Blend Type 1D

Parameter Vertical

Motion	Threshold	
Run_S	-1	1
Idle	0	1
Run_N	1	1

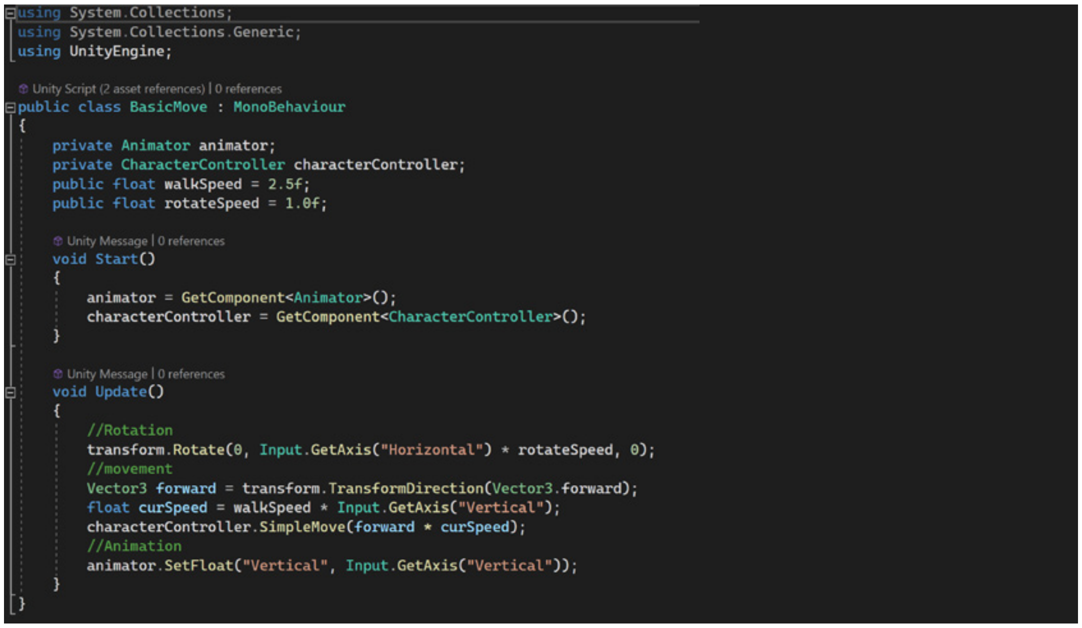
Automate Thresholds ☐

Compute Thresholds Select

Adjust Time Scale Select

Blend tree animations for a playable character

This will require a script to control character movement, using a simple character controller move.

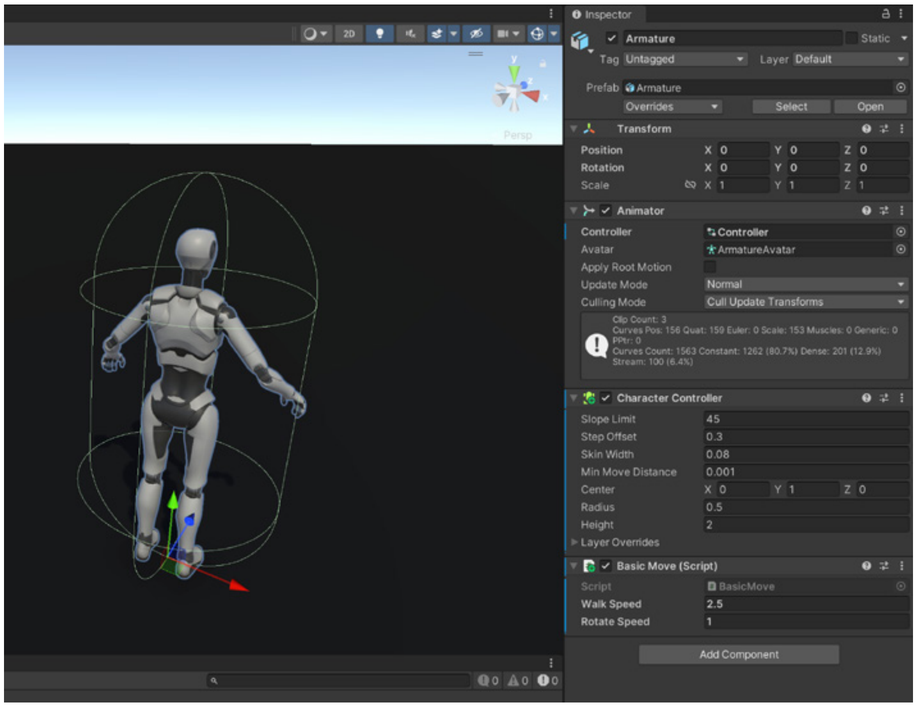


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (2 asset references) | 0 references
public class BasicMove : MonoBehaviour
{
    private Animator animator;
    private CharacterController characterController;
    public float walkSpeed = 2.5f;
    public float rotateSpeed = 1.0f;

    Unity Message | 0 references
    void Start()
    {
        animator = GetComponent<Animator>();
        characterController = GetComponent<CharacterController>();
    }

    Unity Message | 0 references
    void Update()
    {
        //Rotation
        transform.Rotate(0, Input.GetAxis("Horizontal") * rotateSpeed, 0);
        //movement
        Vector3 forward = transform.TransformDirection(Vector3.forward);
        float curSpeed = walkSpeed * Input.GetAxis("Vertical");
        characterController.SimpleMove(forward * curSpeed);
        //Animation
        animator.SetFloat("Vertical", Input.GetAxis("Vertical"));
    }
}
```

Basic move script to control player movement and rotation

4. Add the script and a Character Controller component to your character.
5. Adjust the Step Offset to allow the character to walk up steps. If the character needs to climb very steep steps make this number higher (however, if it is too high the character may be able to step over small walls).
6. Adjust the Slope Limit properties to allow the character to walk up sloping surfaces. 45 degrees is the default value, however, you can set this higher for the character to climb steeper surfaces. 90 degrees or higher will mean the character can walk up the sides of buildings.

7. To raise the character controller to match the center of the character set the Center Y value to 1.
8. The height can then be set to 2 to match the character's height. If your character is taller, you can make the height value greater.
9. Change the radius value to adjust the width. For wider characters, set this to 1 or above; for thin characters, set it to 0.3 or below.



Armature — Static
Tag: Untagged — Layer: Default
Prefab: Armature — Overrides — Select — Open

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

Animator

Controller	Controller
Avatar	ArmatureAvatar
Apply Root Motion	
Update Mode	Normal
Culling Mode	Cull Update Transforms

```
Clip Count: 3
Curves Pos: 156 Quat: 159 Euler: 0 Scale: 153 Muscles: 0 Generic: 0
PPtr: 0
Curves Count: 1563 Constant: 1262 (80.7%) Dense: 201 (12.9%)
Stream: 100 (6.4%)
```

Character Controller

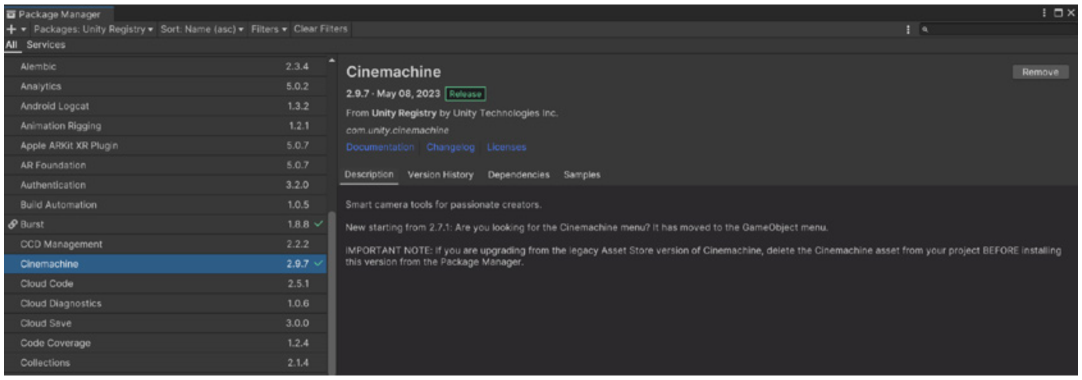
Slope Limit	45
Step Offset	0.3
Skin Width	0.08
Min Move Distance	0.001
Center	X 0 Y 1 Z 0
Radius	0.5
Height	2
Layer Overrides	

Basic Move (Script)

Script	BasicMove
Walk Speed	2.5
Rotate Speed	1

Character controller component added to a player character

Cinemachine is a Unity asset that simplifies the process of creating dynamic camera systems for games and simulations by providing advanced camera controls, procedural animation, and smart camera behaviors. We will cover it in more detail later on in the book that covers Cinemachine. For now install Cinemachine via **Window > Package Manager,** in the Unity Registry menu.



Alembic	2.3.4
Analytics	5.0.2
Android Logcat	1.3.2
Animation Rigging	1.2.1
Apple ARKit XR Plugin	5.0.7
AR Foundation	5.0.7
Authentication	3.2.0
Build Automation	1.0.5
Burst	1.8.8 ✓
CCD Management	2.2.2
Cinemachine	2.9.7 ✓
Cloud Code	2.5.1
Cloud Diagnostics	1.0.6
Cloud Save	3.0.0
Code Coverage	1.2.4
Collections	2.1.4

Cinemachine
2.9.7 · May 08, 2023 Release
From **Unity Registry** by Unity Technologies Inc.
com.unity.cinemachine

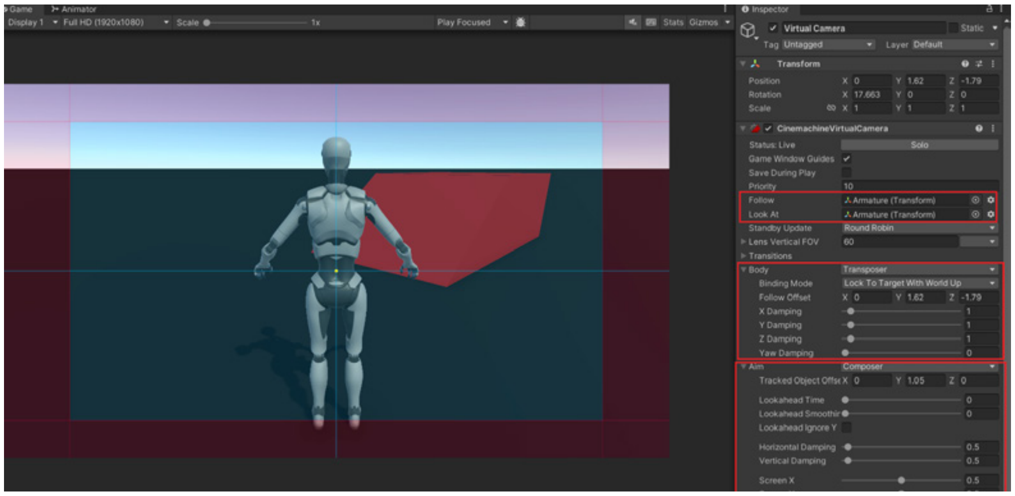
Smart camera tools for passionate creators.

New starting from 2.7.1: Are you looking for the Cinemachine menu? It has moved to the GameObject menu.

IMPORTANT NOTE: If you are upgrading from the legacy Asset Store version of Cinemachine, delete the Cinemachine asset from your project BEFORE installing this version from the Package Manager.

Cinemachine package from the package manager

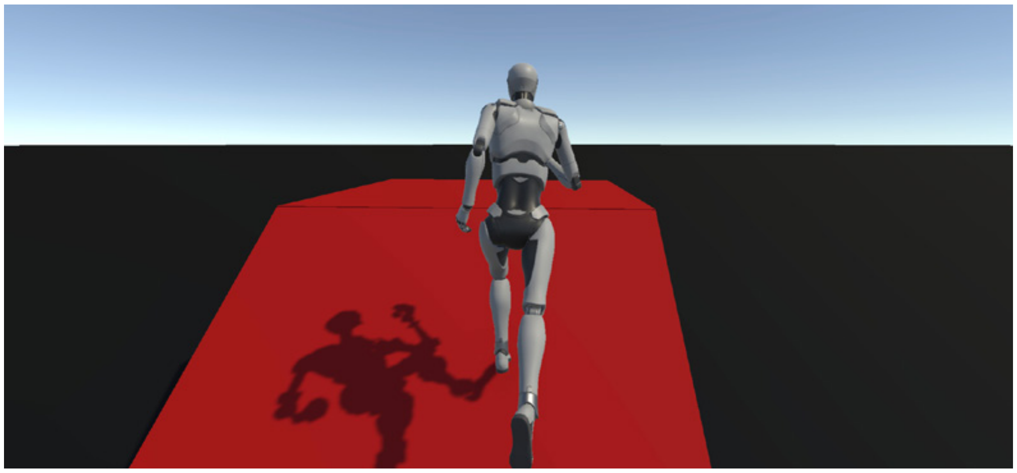
Add a player follow camera via **GameObject > Cinemachine > Virtual Camera**. Add the character to the **Follow** and **Look At** fields, and adjust the values to set up the camera view in the Scene view.



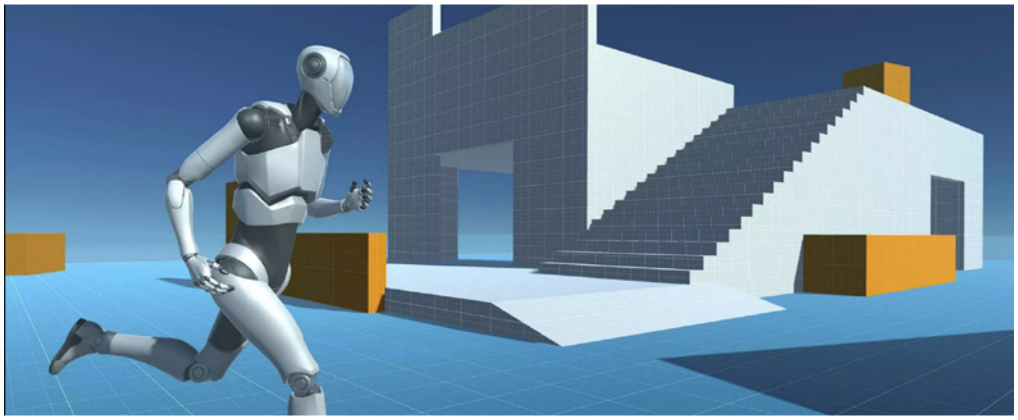
```
Virtual Camera                      Static
Tag Untagged          Layer Default
Transform
Position        X 0        Y 1.62   Z -1.79
Rotation        X 17.663   Y 0      Z 0
Scale           X 1        Y 1      Z 1
CinemachineVirtualCamera
Status: Live              Solo
Game Window Guides        ✓
Save During Play
Priority                  10
Follow                    Armature (Transform)
Look At                   Armature (Transform)
Standby Update            Round Robin
Lens Vertical FOV         60
Transitions
Body                      Transposer
  Binding Mode            Lock To Target With World Up
  Follow Offset           X 0      Y 1.62   Z -1.79
  X Damping               1
  Y Damping               1
  Z Damping               1
  Yaw Damping             0
Aim                       Composer
  Tracked Object Offse X  0        Y 1.05   Z 0
  Lookahead Time          0
  Lookahead Smoothir      0
  Lookahead Ignore Y
  Horizontal Damping      0.5
  Vertical Damping        0.5
  Screen X                0.5
```

Cinemachine virtual camera settings for a follow camera

Now when you play the game, you can control the character with WASD or arrow keys.



Playable character with a character controller



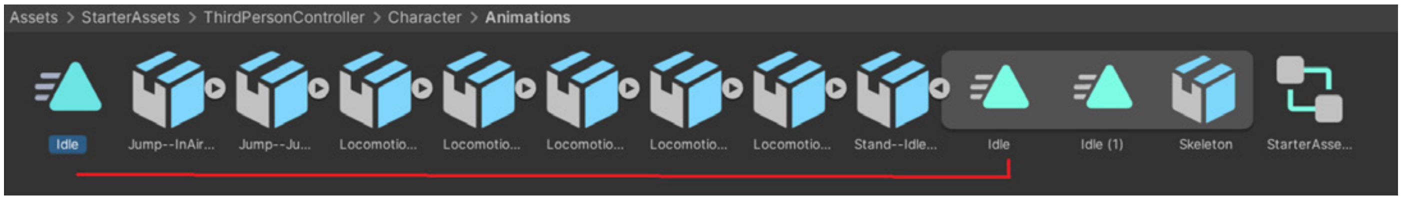
Third person starter assets pack

Animating in Unity

Any object can be animated directly in Unity. Open the Animation view by going to **Window >Animation >Animation**.

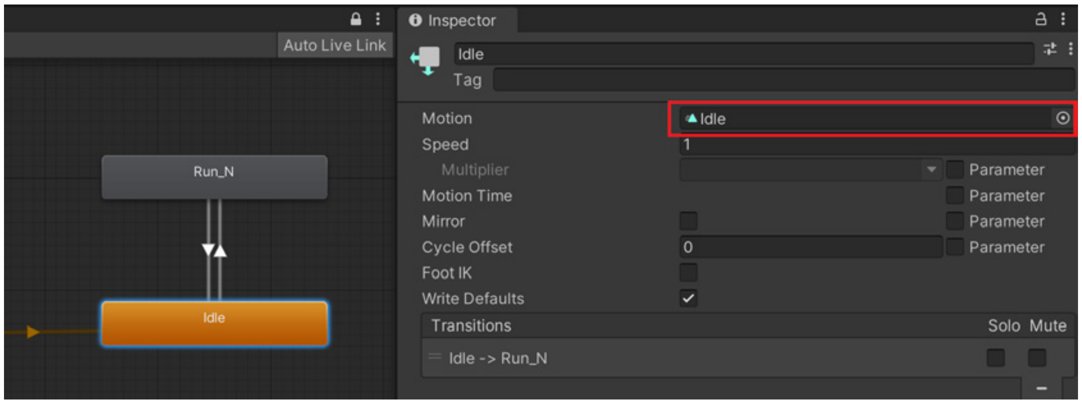
Animations embedded within FBX files are read-only. You need to duplicate the clips in order to edit them, but be aware that once you duplicate a clip, you can no longer change its animation type (you'll need to access the original clip in the FBX file to change its animation type).

Select the animation and duplicate it via **Ctrl + D** or **Edit >Duplicate**.



Duplicating animations

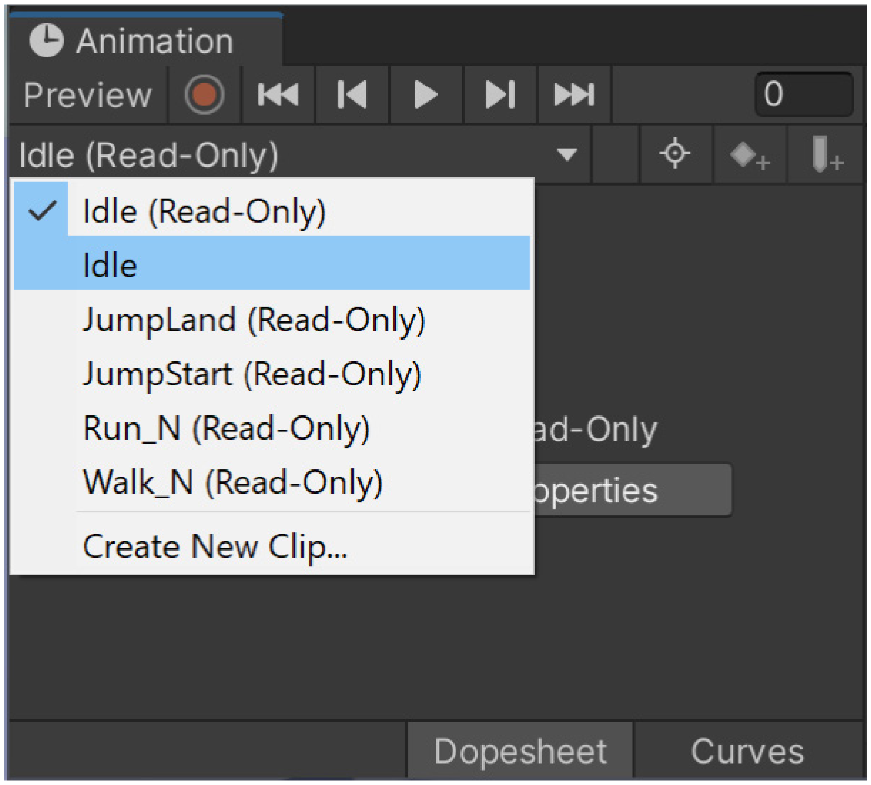
Add the duplicated animation to the character's Animator Controller to edit it. Click the Idle node and drag the new animation file into the **Motion** field.



Motion	Idle	
Speed	1	
Multiplier		Parameter
Motion Time		Parameter
Mirror		Parameter
Cycle Offset	0	Parameter
Foot IK		
Write Defaults	✓	

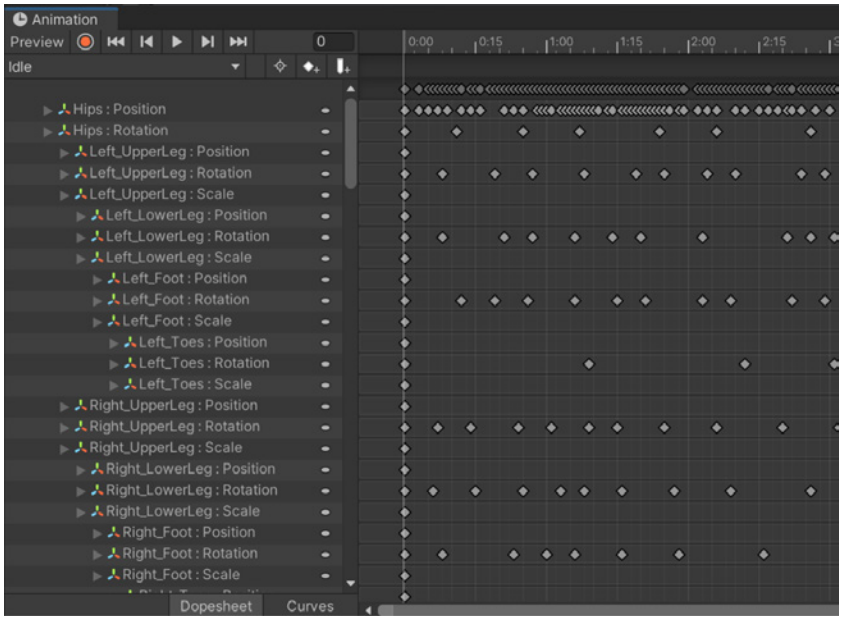
Replace animation clips by dragging the new clip into the Motion slot.

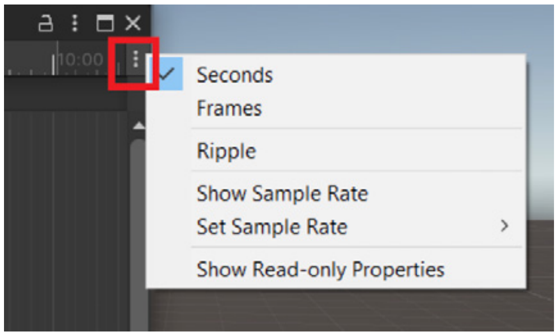
With the character selected you'll have access to the animations inside the Animator Controller directly from the Animation view. These can be selected using the drop-down menu from the left panel. Notice the duplicated clip is editable whereas the embedded ones are read-only.



You need to duplicate embedded animations to modify them.

You can also create new animation clips from the same drop-down menu by selecting **Create New Clip**.



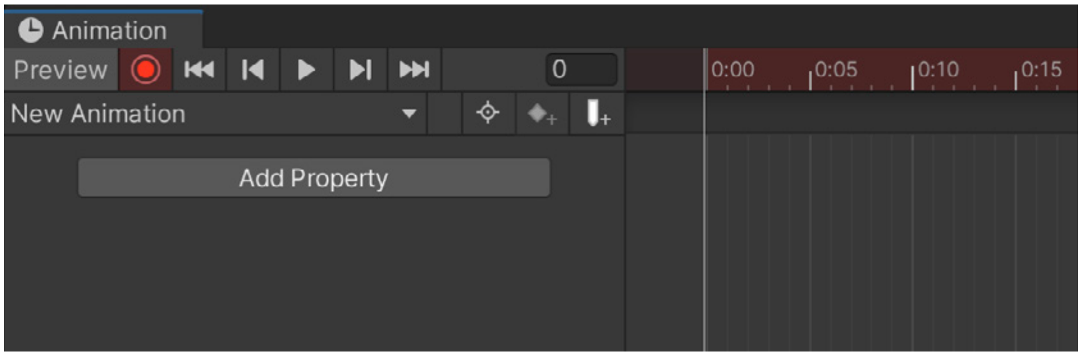


Viewing keyframes (top left), and changing animation options (top right)

Select keyframes by clicking on them. You can delete, move, duplicate, or edit them; to modify them, switch on the Record button.

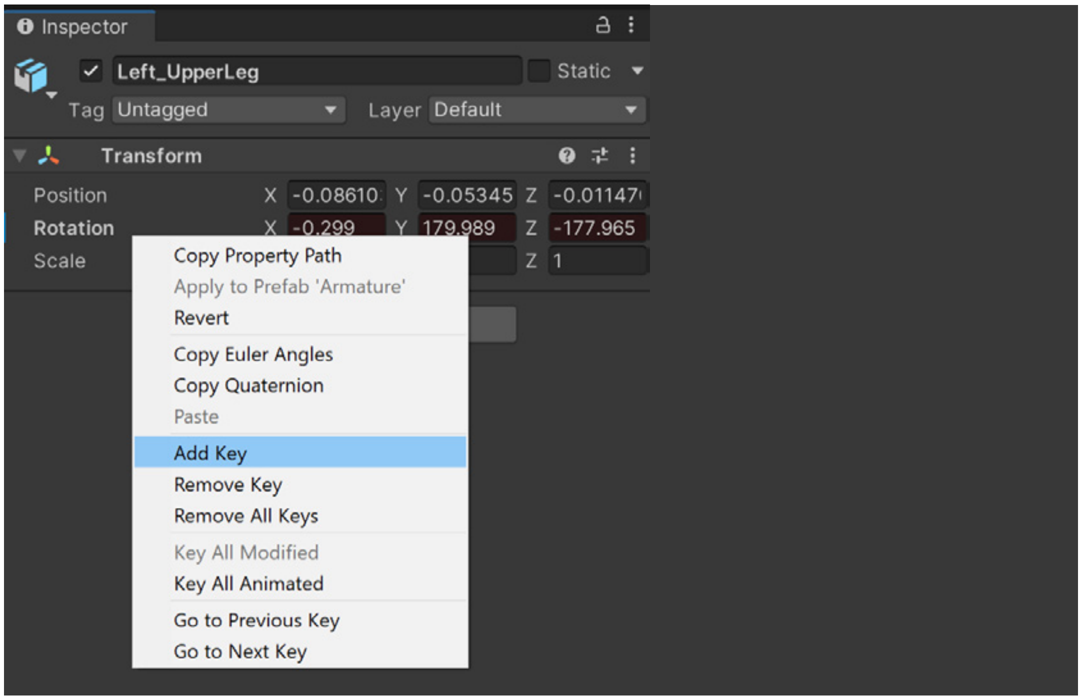
Click the icon of three dots at the top right of the Animation window to change the options. You can view the timeline in seconds or frames or change the sample rate from 30fps to 60fps, or vice versa.

Create your own forward kinematic (FK) animations in a new clip. A new clip will be blank with no keyframes set. Press the Record button to begin setting keyframes.



A new clip has no keyframes. Press record to begin creating keyframes.

To set a keyframe, select a bone in the Hierarchy, right-click at the rotation in the Inspector and choose **Add Key**. This sets an initial keyframe.



Inspector

Left_UpperLeg | Static
Tag Untagged | Layer Default

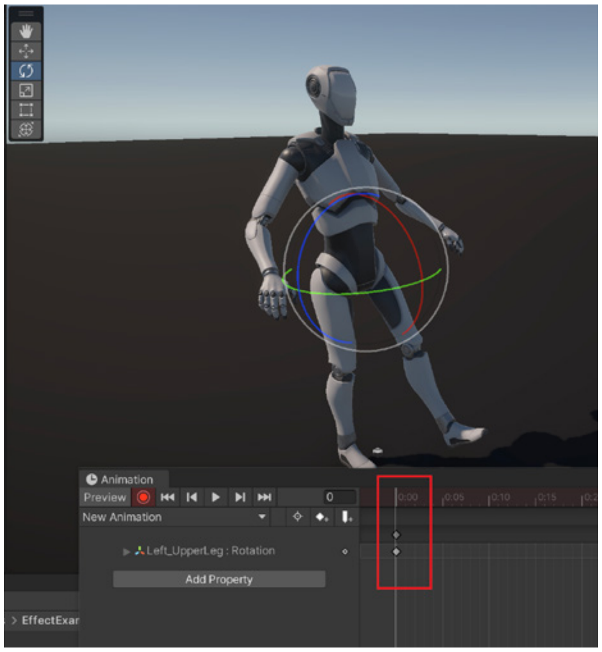
Transform

Position X -0.08610 Y -0.05345 Z -0.01147
Rotation X -0.299 Y 179.989 Z -177.965
Scale Z 1

Copy Property Path
Apply to Prefab 'Armature'
Revert
Copy Euler Angles
Copy Quaternion
Paste
Add Key
Remove Key
Remove All Keys
Key All Modified
Key All Animated
Go to Previous Key
Go to Next Key

Right-click next to the Rotation field to add a key for the bone.

Bones are animated between keyframes, so this is useful when you want a bone to stay at the same position over time.

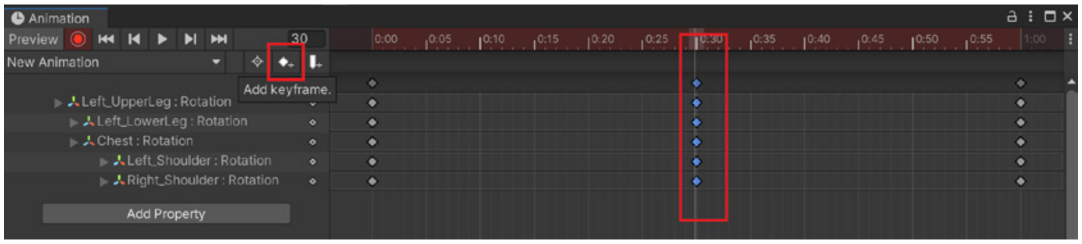


Rotate a bone to set a keyframe.

For example, if you want the leg to remain still for two seconds, you add a key at 0, move forward to 2 seconds and add another key. Now the leg will remain still. You can then begin rotating after that point.

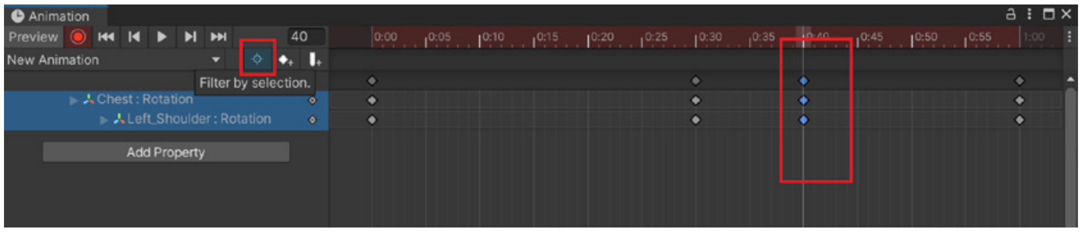
Use the rotate tool in the Scene view to begin rotating the bone to create a new keyframe.

You can also use the **Add keyframe** button to add keyframes at the selected frame for all bones in the animation.



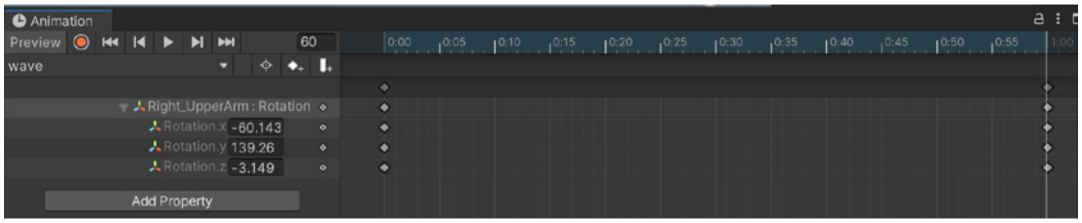
The Add keyframe option adds new keyframes for all bones at the set frame.

If you want to set keys for certain bones only, select the bones in the list and click the **Filter by selection** button. Now when you press the Add keyframe button it only adds keys for those bones. This can be quicker than manually right-clicking on the rotation of each bone and adding a key.



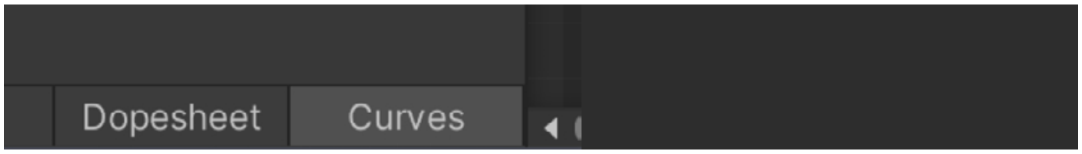
Filter by selection to add keys to selected bones only.

You can manually adjust the values for Rotation, Transform, and Scale on the X, Y, and Z axes separately, as well as typing in specific values.



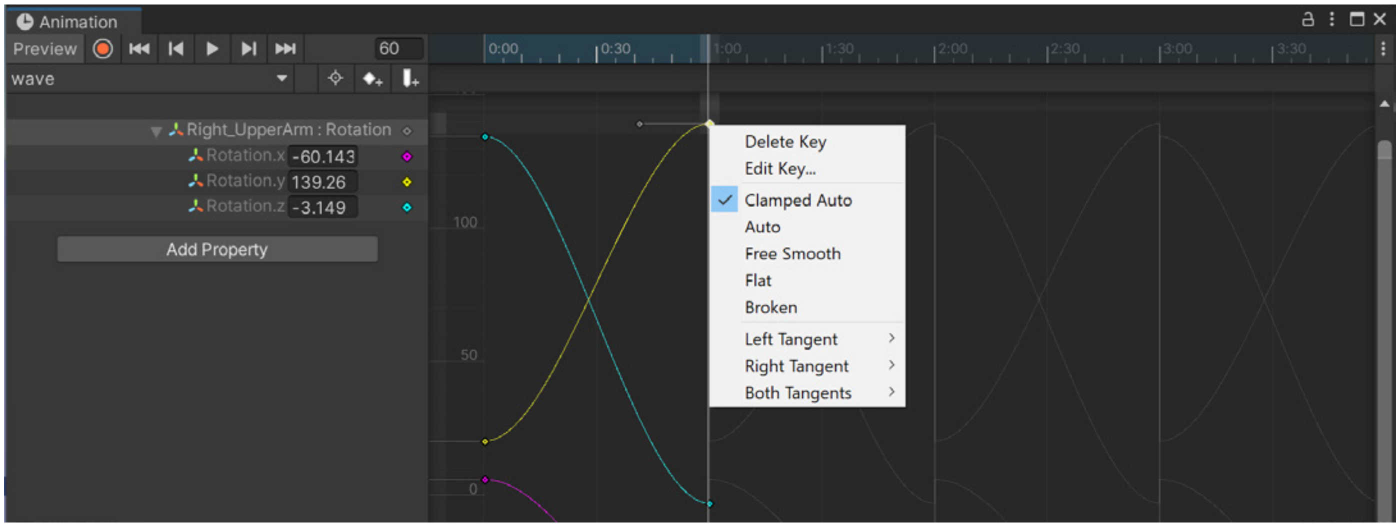
Keyframing X,Y, and Z axes

At the bottom of the Animation window you can change from a **Dopesheet** view (that shows the keyframes), to the Curves view to click individual keyframe curves to edit or move them.



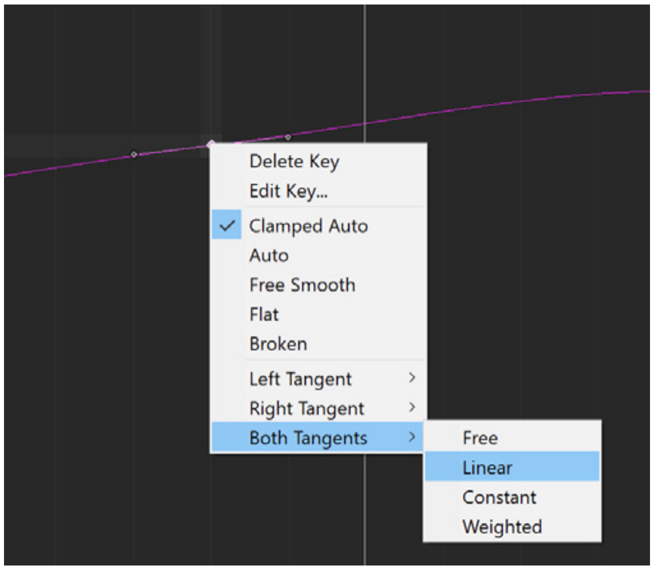
Change between Dopesheet (keyframes) and Curves using the buttons at the bottom.

Right-clicking allows you to change the shape of the curve, altering its easing between start and end positions, as well as using the bezier handle to set a curve.



Curves editing

By default curves are set to **Clamped Auto** which adds an ease in and ease out to the motion between keyframes. It will start slow, speed up in the middle, and slow down again toward the end, providing a fluid smooth motion.

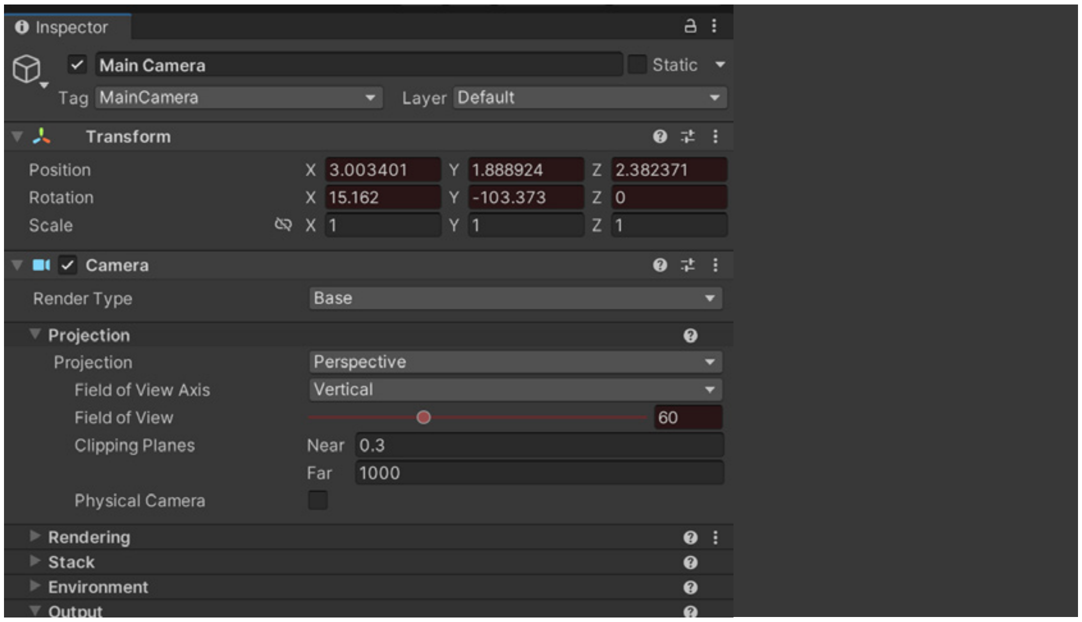


Removing curves from a keyframe by choosing a Linear tangent

To remove curves from the motion to ensure there is no easing between keyframes, right-click the keyframe, choose both tangents and set it to **Linear** (this can be customized to just have a linear flow from the left or right as well).

Animate any object and its components, switch items on or off, modify values in a material or the intensity of lights, or position the field-of-view of a camera using animations.

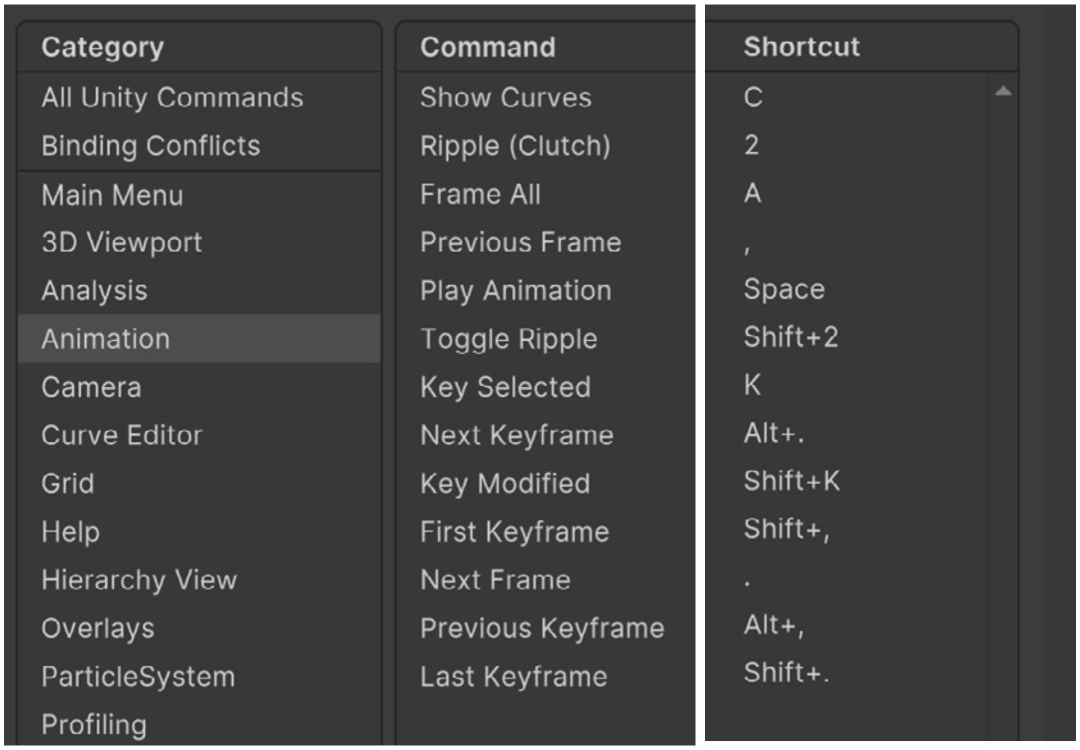
Animated settings appear in red in the Inspector and in blue when coming out of Record mode.



Items that have been animated turn red or blue, like the Position and Rotation values in this image.

Useful shortcuts for the Animation window

To display shortcuts for the Animation window go to **Edit > Shortcuts**. Choose **Animation** from the list:



Category	Command	Shortcut
All Unity Commands	Show Curves	C
Binding Conflicts	Ripple (Clutch)	2
Main Menu	Frame All	A
3D Viewport	Previous Frame	,
Analysis	Play Animation	Space
Animation	Toggle Ripple	Shift+2
Camera	Key Selected	K
Curve Editor	Next Keyframe	Alt+.
Grid	Key Modified	Shift+K
Help	First Keyframe	Shift+,
Hierarchy View	Next Frame	.
Overlays	Previous Keyframe	Alt+,
ParticleSystem	Last Keyframe	Shift+.
Profiling		

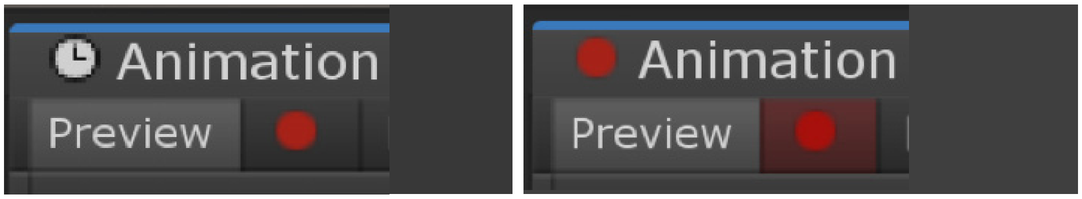
Useful animation shortcuts on Windows

Animation Window zoom controls for Windows

Frame selected keys/curves	F
Frame all curves	A
Zoom in/out	Mouse wheel (or) Drag Alt+Right click
Zoom vertical	Drag Alt+shift+Right click
Zoom horizontal	Drag Alt+ctrl+Right click

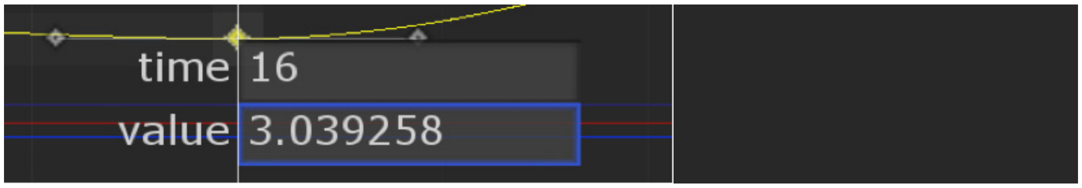
Shortcuts for navigating the Animation window on Windows

Preview vs Recording mode



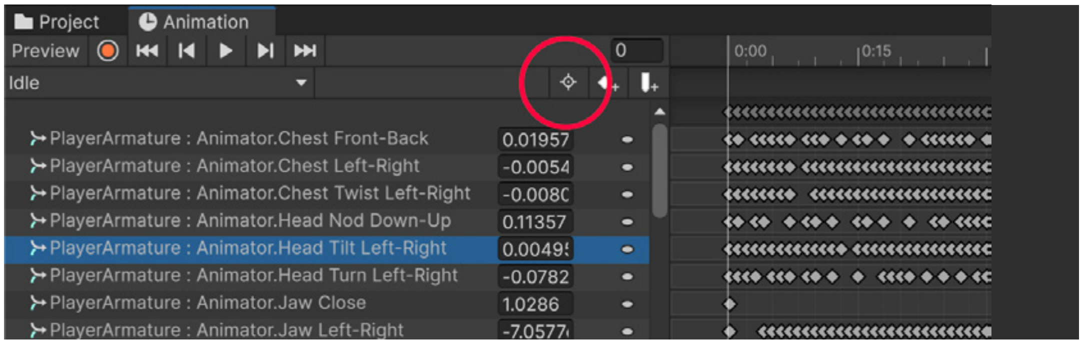
Preview lets you set candidate keys on properties you need to manually keyframe. Recording will automatically add all modifications to the clip.

Setting exact values for keyframes


time 16
value 3.039258

In the Curves view press Enter with one or more keyframes selected. A pop-up dialog will appear where you can enter exact value and time values (note that this does not modify tangents).

Filter by selection



PlayerArmature : Animator.Chest Front-Back 0.01957
PlayerArmature : Animator.Chest Left-Right -0.0054
PlayerArmature : Animator.Chest Twist Left-Right -0.008C
PlayerArmature : Animator.Head Nod Down-Up 0.11357
PlayerArmature : Animator.Head Tilt Left-Right 0.0049!
PlayerArmature : Animator.Head Turn Left-Right -0.0782
PlayerArmature : Animator.Jaw Close 1.0286
PlayerArmature : Animator.Jaw Left-Right -7.0577

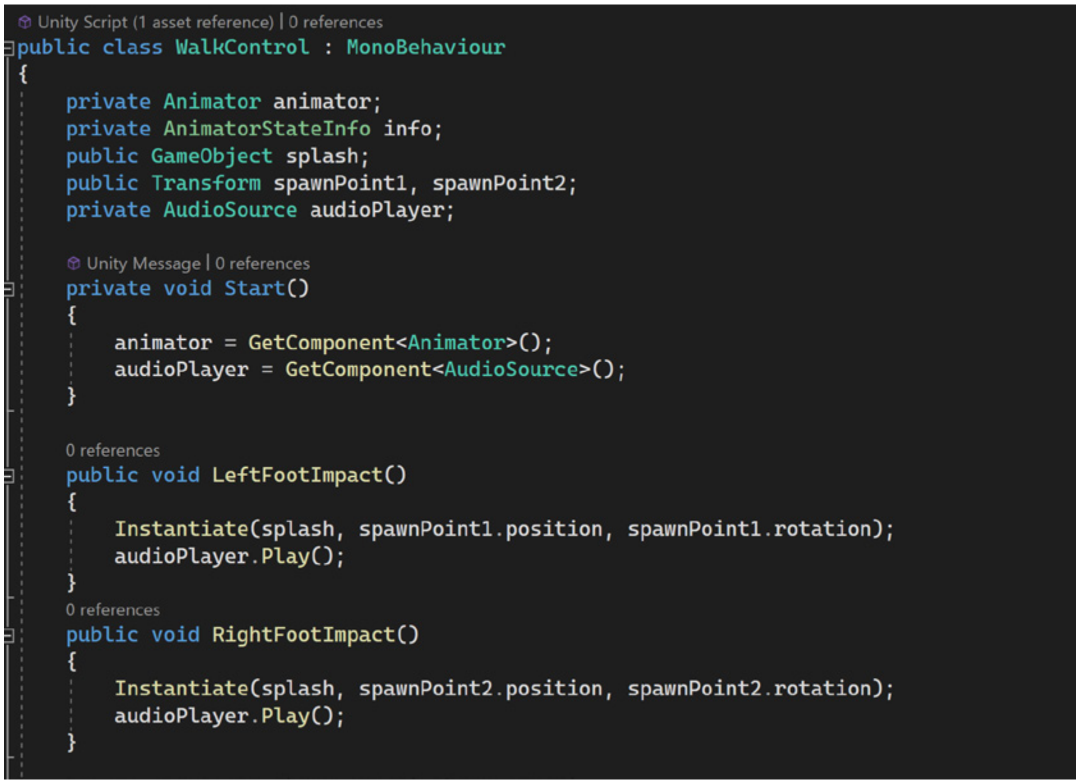
This is the icon for the **Filter by** selection button:

This shows the properties of only selected objects in the Scene view. When this is deactivated, all properties in the Animation Clip will be shown.

Animation events

Events can be called from an animation that trigger a function to run in a C# script. These functions can be in any script attached to the GameObject.

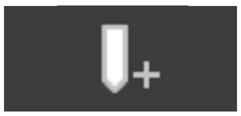
In the example below there are two custom functions that will spawn in a water splash and play a footstep sound when the left foot or right foot makes impact with the ground.



```
Unity Script (1 asset reference) | 0 references
public class WalkControl : MonoBehaviour
{
    private Animator animator;
    private AnimatorStateInfo info;
    public GameObject splash;
    public Transform spawnPoint1, spawnPoint2;
    private AudioSource audioPlayer;

    Unity Message | 0 references
    private void Start()
    {
        animator = GetComponent<Animator>();
        audioPlayer = GetComponent<AudioSource>();
    }

    0 references
    public void LeftFootImpact()
    {
        Instantiate(splash, spawnPoint1.position, spawnPoint1.rotation);
        audioPlayer.Play();
    }
    0 references
    public void RightFootImpact()
    {
        Instantiate(splash, spawnPoint2.position, spawnPoint2.rotation);
        audioPlayer.Play();
    }
```

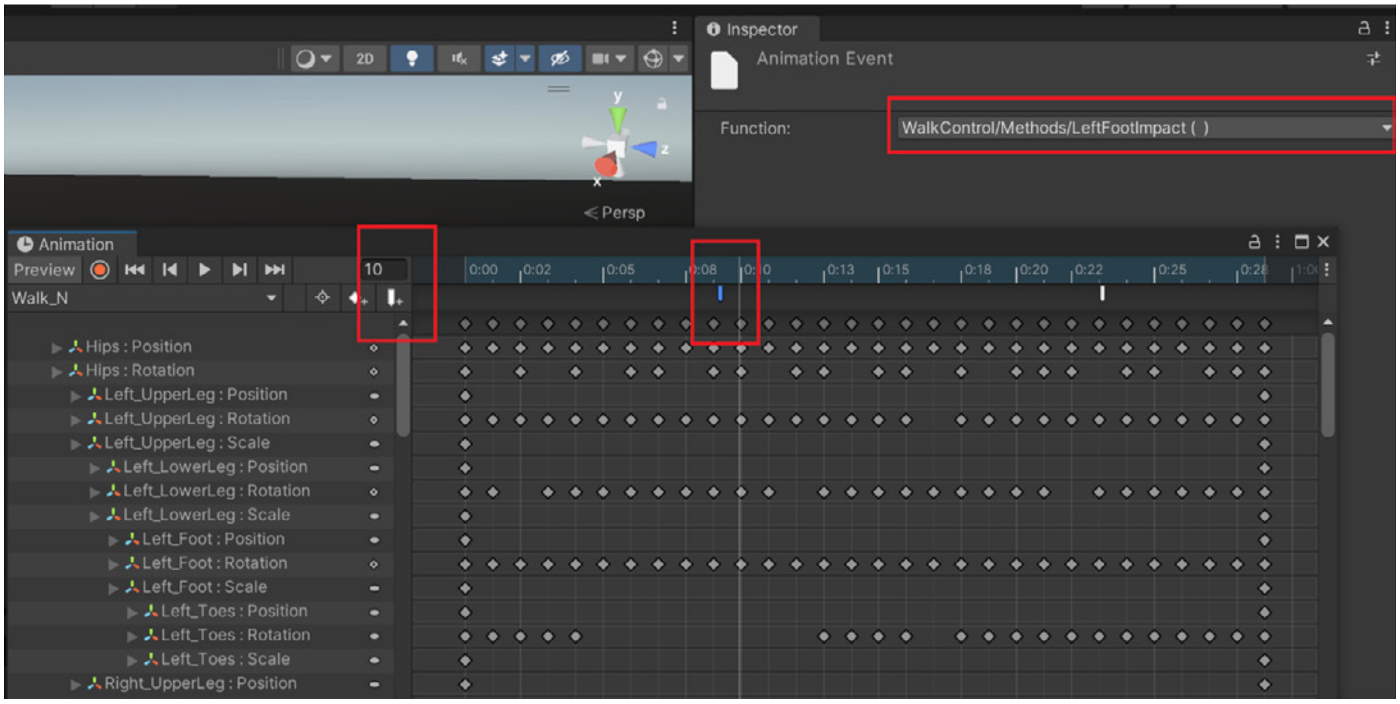
Custom functions to be triggered by animation events

In the Animation window you can time the events to the correct place in the animation. Add an event by clicking on the event icon.



Event icon to add a new event

In the Inspector it will prompt you to select a function from a script that you want to trigger.



Function: WalkControl/Methods/LeftFootImpact ( )

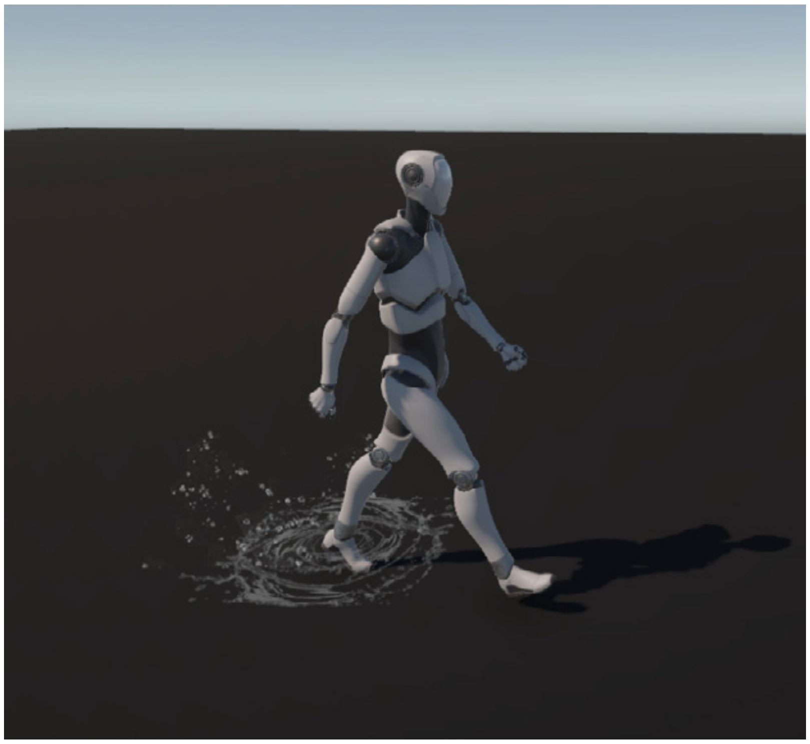
Walk_N

10

Hips : Position
Hips : Rotation
Left_UpperLeg : Position
Left_UpperLeg : Rotation
Left_UpperLeg : Scale
Left_LowerLeg : Position
Left_LowerLeg : Rotation
Left_LowerLeg : Scale
Left_Foot : Position
Left_Foot : Rotation
Left_Foot : Scale
Left_Toes : Position
Left_Toes : Rotation
Left_Toes : Scale
Right_UpperLeg : Position

Adding events to an animation

When you play the animation the result should be splash effects spawning in as the feet hit the puddle, with footstep sounds timed to the walk cycle.



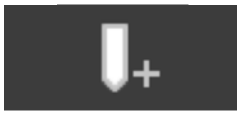
Event-driven code during an animation

Advanced animation features

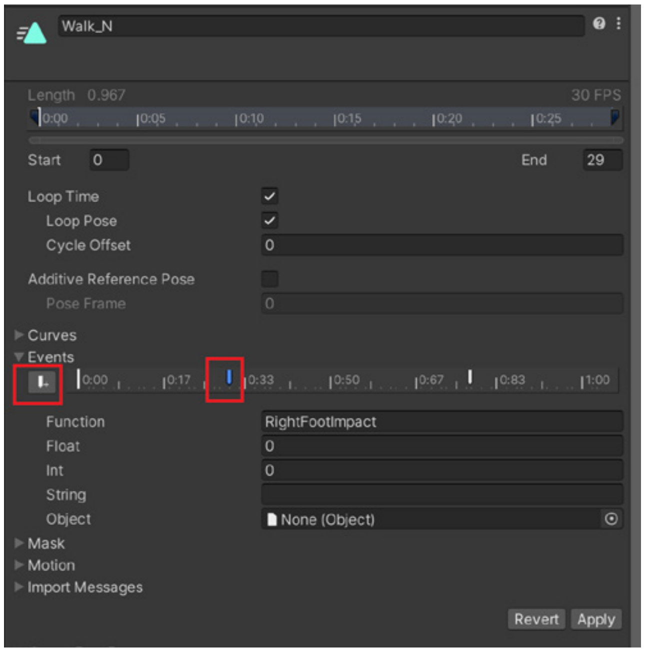
Events in read-only animations

Events can also be added to a clip that is read-only, which is embedded within an FBX file. If a clip has not been duplicated out of the FBX file it will not be editable in the Animation window.

Open the Events section in the animation section of the FBX file. You can now move to the timing of the clip, watching the preview window at the bottom of the screen. Click **Add Event**.


Event icon to add a new event

This requires you to manually type in the name of the function in the script. There are also options to add required parameters such as integers, floats or strings.


Walk_N

Length 0.967 30 FPS

Start 0 End 29

Loop Time ✓
Loop Pose ✓
Cycle Offset 0

Additive Reference Pose
Pose Frame 0

Curves
Events

Function RightFootImpact
Float 0
Int 0
String
Object None (Object)

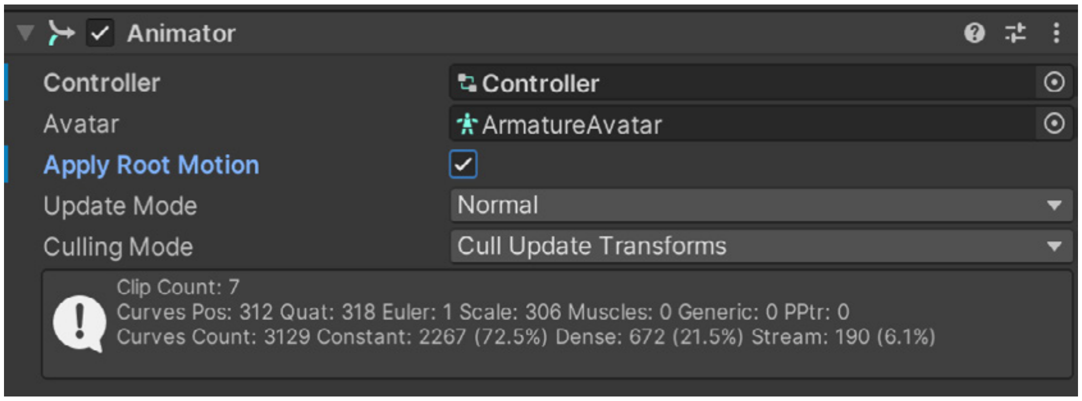
Mask
Motion
Import Messages

Adding events to read-only clips

Root motion

Many clips are in-place animations, where the character stays in one place while the walking animation plays. You can control the character movement using script.

Some clips contain movement keyframes that move the character in 3D space. For example, in a walk animation the character will walk forward for the duration of the clip. However, once the clip loops back to the beginning, the character will also jump back to the origin point, creating an unusual look.



Controller: Controller
Avatar: ArmatureAvatar
Apply Root Motion: ✓
Update Mode: Normal
Culling Mode: Cull Update Transforms

```
Clip Count: 7
Curves Pos: 312 Quat: 318 Euler: 1 Scale: 306 Muscles: 0 Generic: 0 PPtr: 0
Curves Count: 3129 Constant: 2267 (72.5%) Dense: 672 (21.5%) Stream: 190 (6.1%)
```

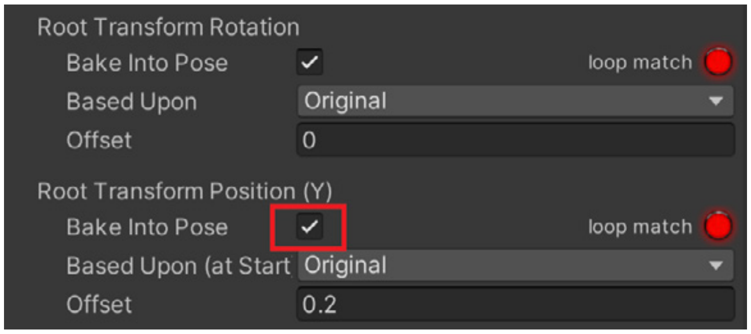
Apply root motion for the Animator component

In the Animator component there is an option to apply root motion. Checking this box ensures that, as the clip loops, it will continue from its current position, instead of jumping back to its original position. This results in the character walking forward realistically.

Root motion can be effective as the movement is timed with the keyframes of the character. As the feet make contact with the ground the character moves forward in a realistic way and avoids the foot sliding issue that can happen when moving a character with code.

Root motion is suited to complex movement, like a boxer character with tricky, multi-punch combos, and feet quickly moving back and forth, and left and right.

Clips containing jumping animations can be useful for cutscene animations, where you can control a character's position using Unity's Timeline. For jumping animations in actual gameplay, bake the Y position of the character into the clip to prevent the character from floating into the air.

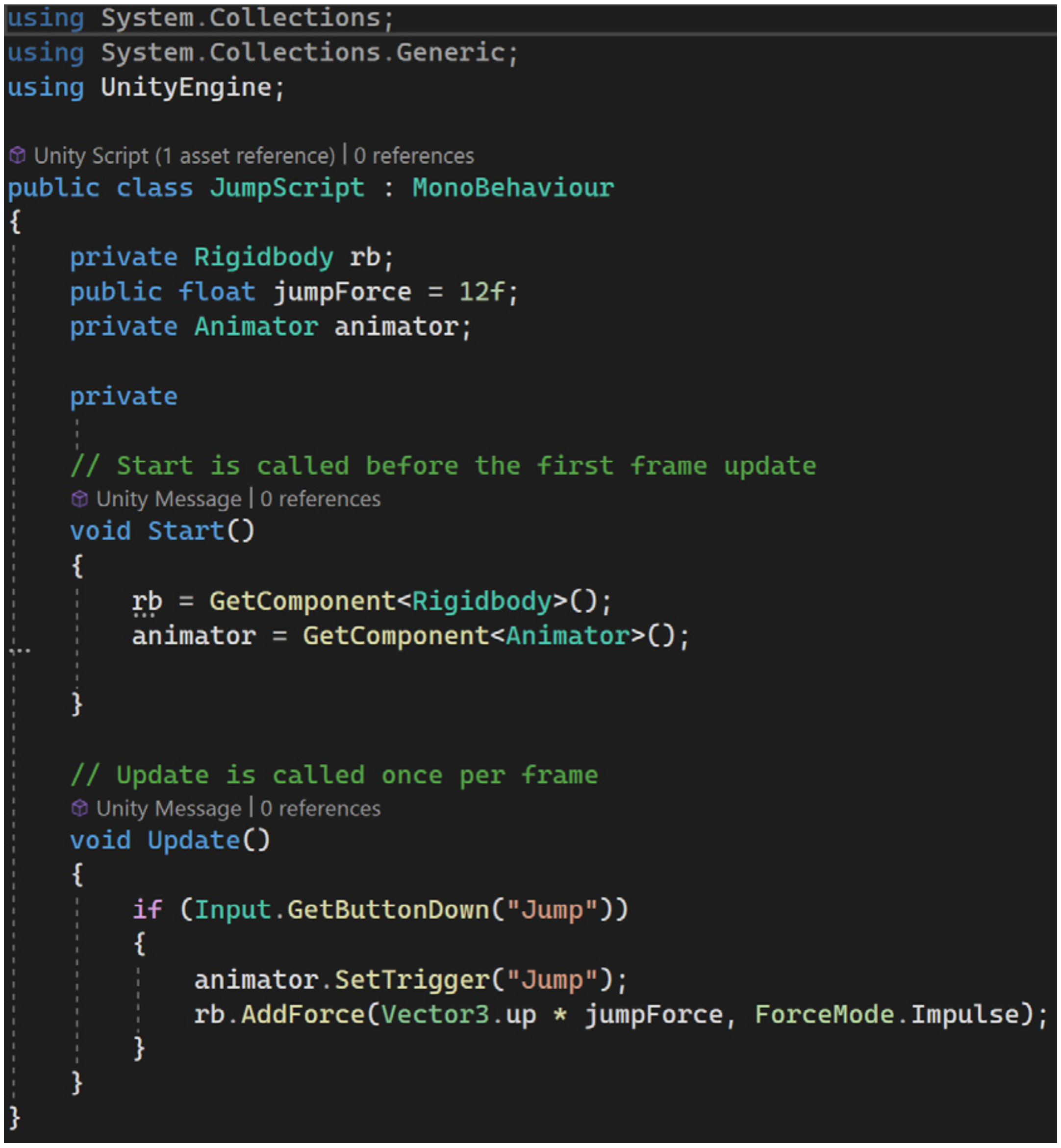


Root Transform Rotation
Bake Into Pose: ✓ loop match ●
Based Upon: Original
Offset: 0

Root Transform Position (Y)
Bake Into Pose: ✓ loop match ●
Based Upon (at Start: Original
Offset: 0.2

Bake the Y position for root motion jumping clips.

You can control the character’s jump using a script. Use Rigidbody.AddForce for the best jumping results.



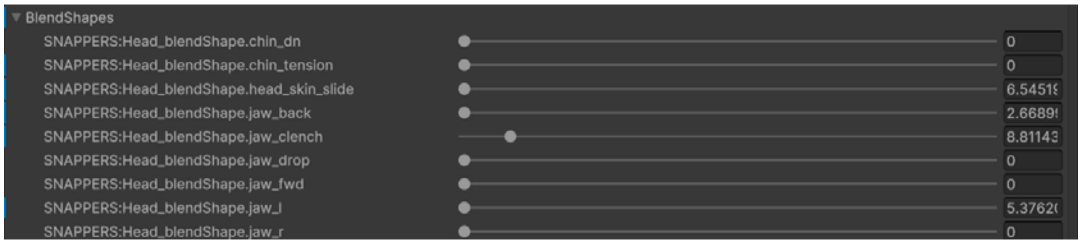
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class JumpScript : MonoBehaviour
{
    private Rigidbody rb;
    public float jumpForce = 12f;
    private Animator animator;

    private

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        rb = GetComponent<Rigidbody>();
        animator = GetComponent<Animator>();

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (Input.GetButtonDown("Jump"))
        {
            animator.SetTrigger("Jump");
            rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
        }
    }
}
```

Controlling a character’s jump using rigidbody in code

Blend shapes

Blend shapes, sometimes referred to as morph targets in other 3D software tools, are vertex deformations to the character mesh shape that can be blended with the original mesh shape.

Blend shapes are baked into the FBX file and can be tweaked in the Inspector. The sliders seen in the below image can be adjusted to set the weight of the blend, where 0 has no weight and 1 is full weight. Multiple sliders can be mixed together to achieve new blend morphs.

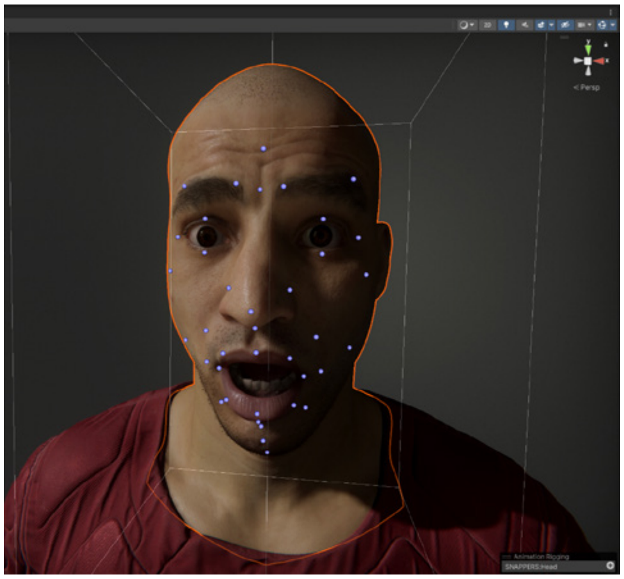


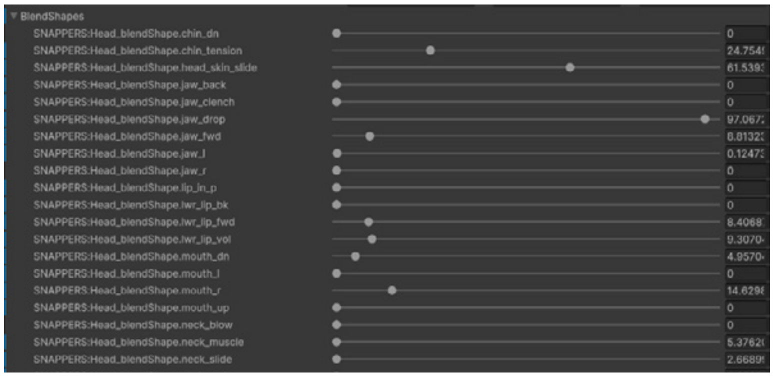
▼ BlendShapes

SNAPPERS:Head_blendShape.chin_dn	0
SNAPPERS:Head_blendShape.chin_tension	0
SNAPPERS:Head_blendShape.head_skin_slide	6.5451
SNAPPERS:Head_blendShape.jaw_back	2.6689
SNAPPERS:Head_blendShape.jaw_clench	8.81143
SNAPPERS:Head_blendShape.jaw_drop	0
SNAPPERS:Head_blendShape.jaw_fwd	0
SNAPPERS:Head_blendShape.jaw_l	5.3762
SNAPPERS:Head_blendShape.jaw_r	0

Blend shape sliders in the Inspector

Facial rig blend shapes are often used for more advanced characters, such as Gawain from *The Heretic*. The spots on the face can be dragged around in real time to create a variety of expressions. This is combined with a digital human shader which uses wrinkle maps for adding realistic wrinkles to the face as it deforms.





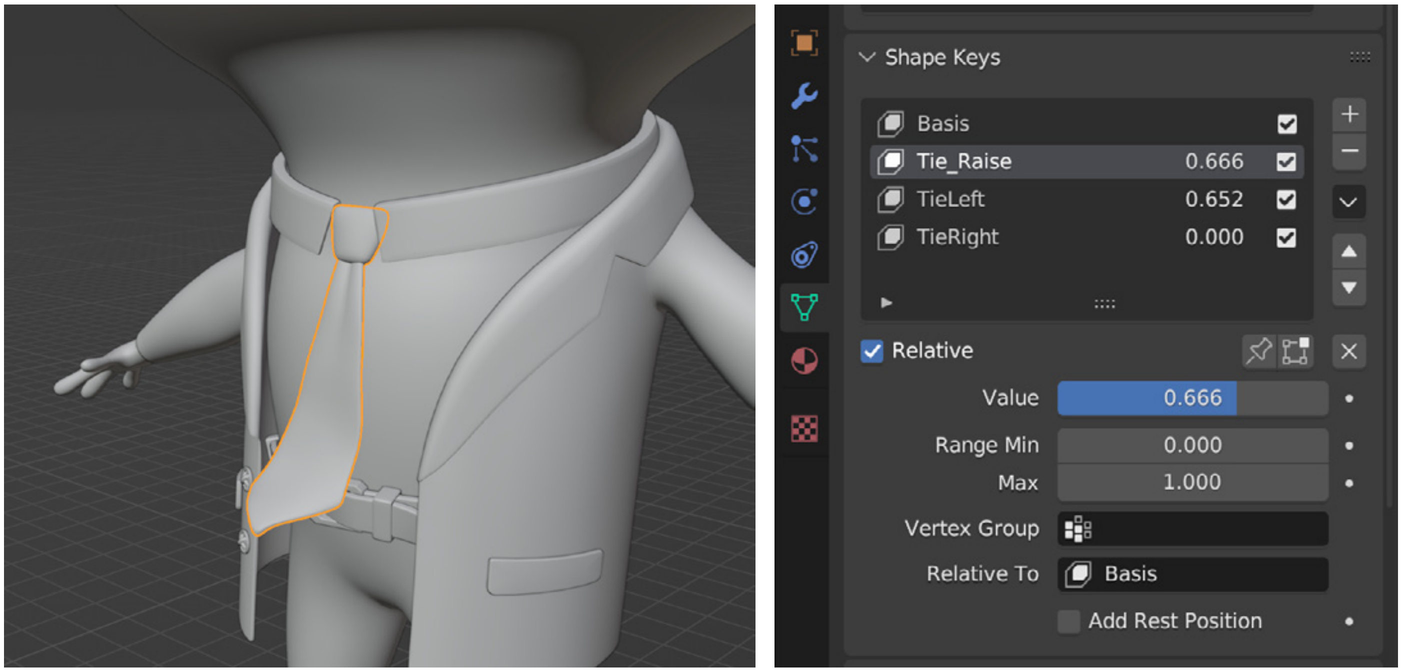
▼ BlendShapes

SNAPPERS:Head_blendShape.chin_dn	0
SNAPPERS:Head_blendShape.chin_tension	24.754
SNAPPERS:Head_blendShape.head_skin_slide	61.539
SNAPPERS:Head_blendShape.jaw_back	0
SNAPPERS:Head_blendShape.jaw_clench	0
SNAPPERS:Head_blendShape.jaw_drop	97.067
SNAPPERS:Head_blendShape.jaw_fwd	8.8132
SNAPPERS:Head_blendShape.jaw_l	0.1247
SNAPPERS:Head_blendShape.jaw_r	0
SNAPPERS:Head_blendShape.lip_in_p	0
SNAPPERS:Head_blendShape.lwr_lip_bk	0
SNAPPERS:Head_blendShape.lwr_lip_fwd	8.4068
SNAPPERS:Head_blendShape.lwr_lip_vol	9.3070
SNAPPERS:Head_blendShape.mouth_dn	4.9570
SNAPPERS:Head_blendShape.mouth_l	0
SNAPPERS:Head_blendShape.mouth_r	14.629
SNAPPERS:Head_blendShape.mouth_up	0
SNAPPERS:Head_blendShape.neck_blow	0
SNAPPERS:Head_blendShape.neck_muscle	5.3762
SNAPPERS:Head_blendShape.neck_slide	2.6689

Blend shapes on the digital human character's face from the Unity demo *The Heretic*

Different expressions created through blend shapes

You can also add blend shapes to items of clothing to create secondary motion in Unity. These types of blend shapes can be created in 3D software such as Blender or Maya.

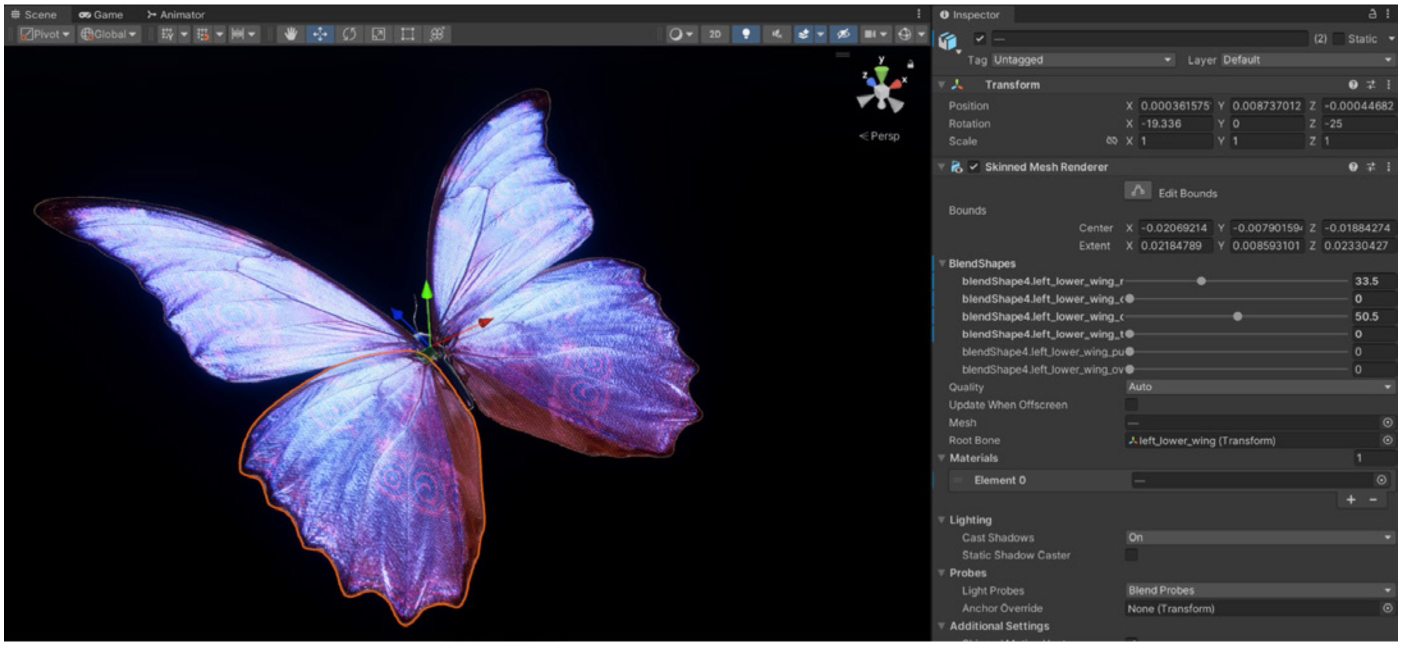


Basis		✓
Tie_Raise	0.666	✓
TieLeft	0.652	✓
TieRight	0.000	✓

✓ Relative
Value 0.666
Range Min 0.000
Max 1.000
Vertex Group
Relative To Basis
Add Rest Position

Shape keys created in Blender to animate a tie moving

Butterfly wings are another example of using blend shapes to create motion for an animated butterfly..



Tag Untagged Layer Default (2) Static

Transform

	X	Y	Z
Position	0.000361575	0.008737012	-0.00044682
Rotation	-19.336	0	-25
Scale	1	1	1

Skinned Mesh Renderer

Edit Bounds

Bounds

	X	Y	Z
Center	-0.02069214	-0.00790159	-0.01884274
Extent	0.02184789	0.008593101	0.02330427

BlendShapes

blendShape4.left_lower_wing_r	33.5
blendShape4.left_lower_wing_c	0
blendShape4.left_lower_wing_c	50.5
blendShape4.left_lower_wing_t	0
blendShape4.left_lower_wing_pu	0
blendShape4.left_lower_wing_ov	0

Quality Auto
Update When Offscreen
Mesh —
Root Bone left_lower_wing (Transform)
Materials 1
Element 0 —
Lighting
Cast Shadows On
Static Shadow Caster
Probes
Light Probes Blend Probes
Anchor Override None (Transform)
Additional Settings

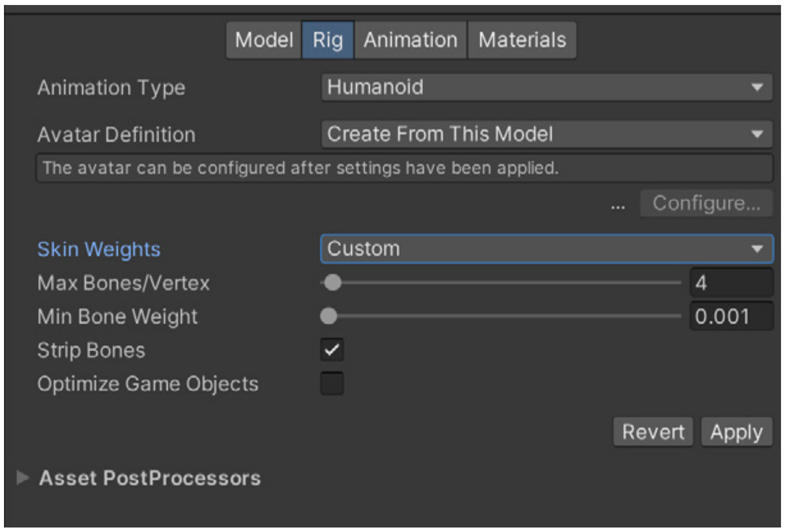
The four parts of the wings can be animated through blend shapes

Humanoid animation type

The humanoid animation is used for animations that need to work on most human characters. Keyframing directly in Unity using forward kinematics (FK) is not possible with a humanoid animation. Animations can be created in other software and imported into Unity. Humanoid animations can be controlled with animation rigging which is covered in a later section.

The humanoid animation type is more complex than the generic one due to features like support for twist bones and leaf bones. It doesn't rely on named bones like the generic animation type, but instead uses reference points for bone positions, which is why humanoid animations work on most human characters.

To set up a humanoid animation type ensure that the character model is in a T or A pose for best target matching. In the Rig section of the FBX model, choose humanoid from the drop-down list and create an avatar from this model. Click **Apply** to change its animation type.

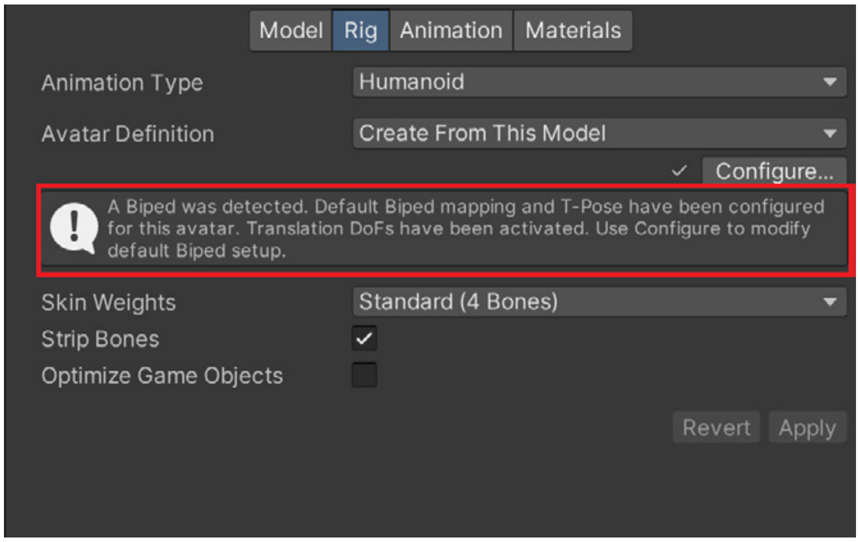


Animation Type: Humanoid
Avatar Definition: Create From This Model
The avatar can be configured after settings have been applied.
... Configure...
Skin Weights: Custom
Max Bones/Vertex: 4
Min Bone Weight: 0.001
Strip Bones: ✓
Optimize Game Objects:

Humanoid rig settings for a character

When you choose the option **Create from this Model** in the **Avatar Definition** field (see the image above and below) then the humanoid conversion process analyzes the skeletal structure of the chosen character. Bones are identified at key positions and placed into slots at key reference points, for example, an upper arm reference point now uses the upper arm bone of this character. It no longer matters what the bone is called, the animation is able to use the reference points to animate the skeleton.

If the conversion is successful you will receive a message in the rig section.

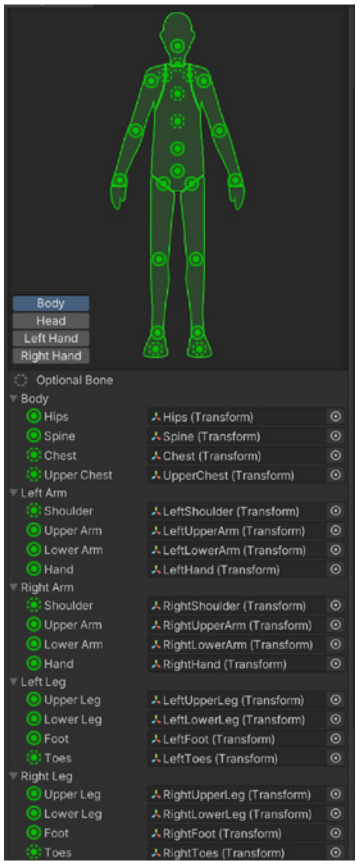


Animation Type: Humanoid
Avatar Definition: Create From This Model
✓ Configure...
A Biped was detected. Default Biped mapping and T-Pose have been configured for this avatar. Translation DoFs have been activated. Use Configure to modify default Biped setup.
Skin Weights: Standard (4 Bones)
Strip Bones: ✓
Optimize Game Objects:

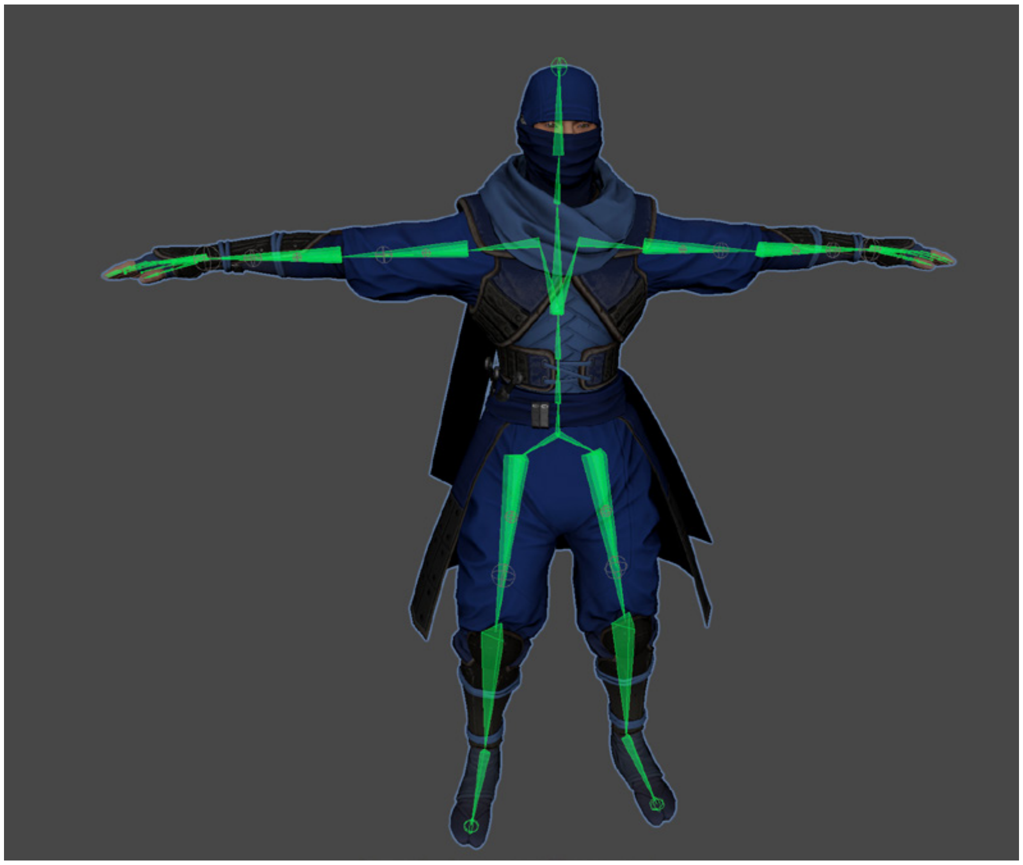
Humanoid conversion is successful

If there are issues with the conversion Unity will highlight which bones could not be located, meaning you might have to try and match the bones manually. If the character's skeleton is too complex it may not be possible to convert to humanoid.

Click **Configure** to manually set up bones for the animation type.



▼ Body	
Hips	Hips (Transform)
Spine	Spine (Transform)
Chest	Chest (Transform)
Upper Chest	UpperChest (Transform)
▼ Left Arm	
Shoulder	LeftShoulder (Transform)
Upper Arm	LeftUpperArm (Transform)
Lower Arm	LeftLowerArm (Transform)
Hand	LeftHand (Transform)
▼ Right Arm	
Shoulder	RightShoulder (Transform)
Upper Arm	RightUpperArm (Transform)
Lower Arm	RightLowerArm (Transform)
Hand	RightHand (Transform)
▼ Left Leg	
Upper Leg	LeftUpperLeg (Transform)
Lower Leg	LeftLowerLeg (Transform)
Foot	LeftFoot (Transform)
Toes	LeftToes (Transform)
▼ Right Leg	
Upper Leg	RightUpperLeg (Transform)
Lower Leg	RightLowerLeg (Transform)
Foot	RightFoot (Transform)
Toes	RightToes (Transform)



The bones are identified at key reference points in the avatar.

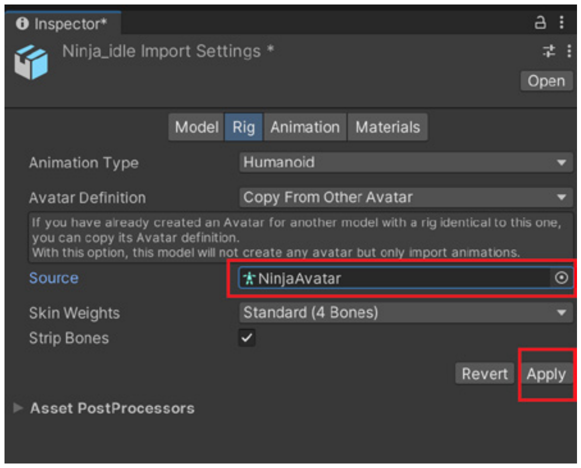
If you use a generic animation on a character with a humanoid animation type it will result in the character going into the setup pose as shown below.

When a generic animation plays on a character with the humanoid animation type, it assumes the setup pose.

The conversion process for the humanoid animation type works best with a character in the T or A pose because the bones are then at a location Unity expects. If a character is in an animated pose, the bones may be at unusual positions, making the conversion more difficult.

Create a humanoid avatar in the T pose, and then use this avatar to convert any animation clip files you import.

For example, when importing animation clips from Mixamo, choose the option **Copy From Other Avatar** in the Avatar Definition field during the conversion process, and choose the avatar from the T pose character.



Animation Type: Humanoid
Avatar Definition: Copy From Other Avatar

If you have already created an Avatar for another model with a rig identical to this one, you can copy its Avatar definition.
With this option, this model will not create any avatar but only import animations.

Source: NinjaAvatar
Skin Weights: Standard (4 Bones)
Strip Bones: ✓

Converting animations to a humanoid rig and using the avatar from the T pose character

It's important that the avatar for the T pose character has the same skeletal structure as the animation clip files, so be sure to import a T pose model alongside animation clips from the software you're using to create or download animations.

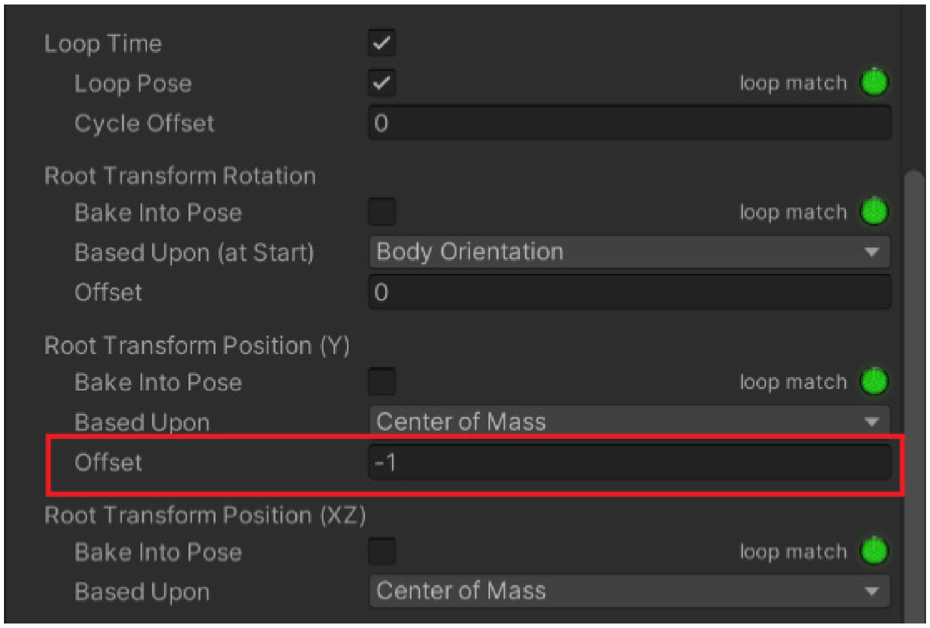
You wouldn't use an avatar from a different character in T pose to convert these clips to humanoid as that would cause issues.

Once the clip has been converted to humanoid animation type it will now work on the original character as well as most other biped characters that use a humanoid animation type.

Imported walk animation working on three characters with different skeletal structures

Humanoid clips have more settings than generic clips, such as adjusting offsets for transform and rotation to match other clips.

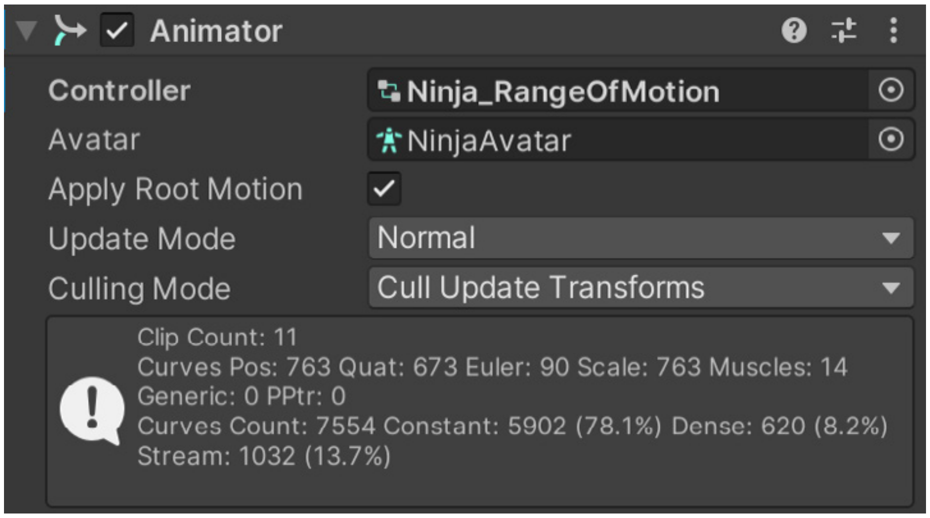
As you can see in the image below the **loop match** option is green for all settings. This means that the first and last keyframes are matching, creating a seamless loop. For Transform this means the character may be animating on the spot, so it doesn't have root motion keyframed into the animation. This type of clip is useful when moving the character using code.



Loop Time ✓
Loop Pose ✓ loop match ●
Cycle Offset 0
Root Transform Rotation
Bake Into Pose loop match ●
Based Upon (at Start) Body Orientation
Offset 0
Root Transform Position (Y)
Bake Into Pose loop match ●
Based Upon Center of Mass
Offset -1
Root Transform Position (XZ)
Bake Into Pose loop match ●
Based Upon Center of Mass

Animation settings and using offsets

If the spots are red this means the keyframes don't match and you will see a pop as the animation begins to loop. Although not all clips are designed to be looped, a red spot means that motion has been keyframed into the animation. This is root motion and when using a humanoid animation with root motion, ensure the **Apply Root Motion** option is checked in the Animator component:

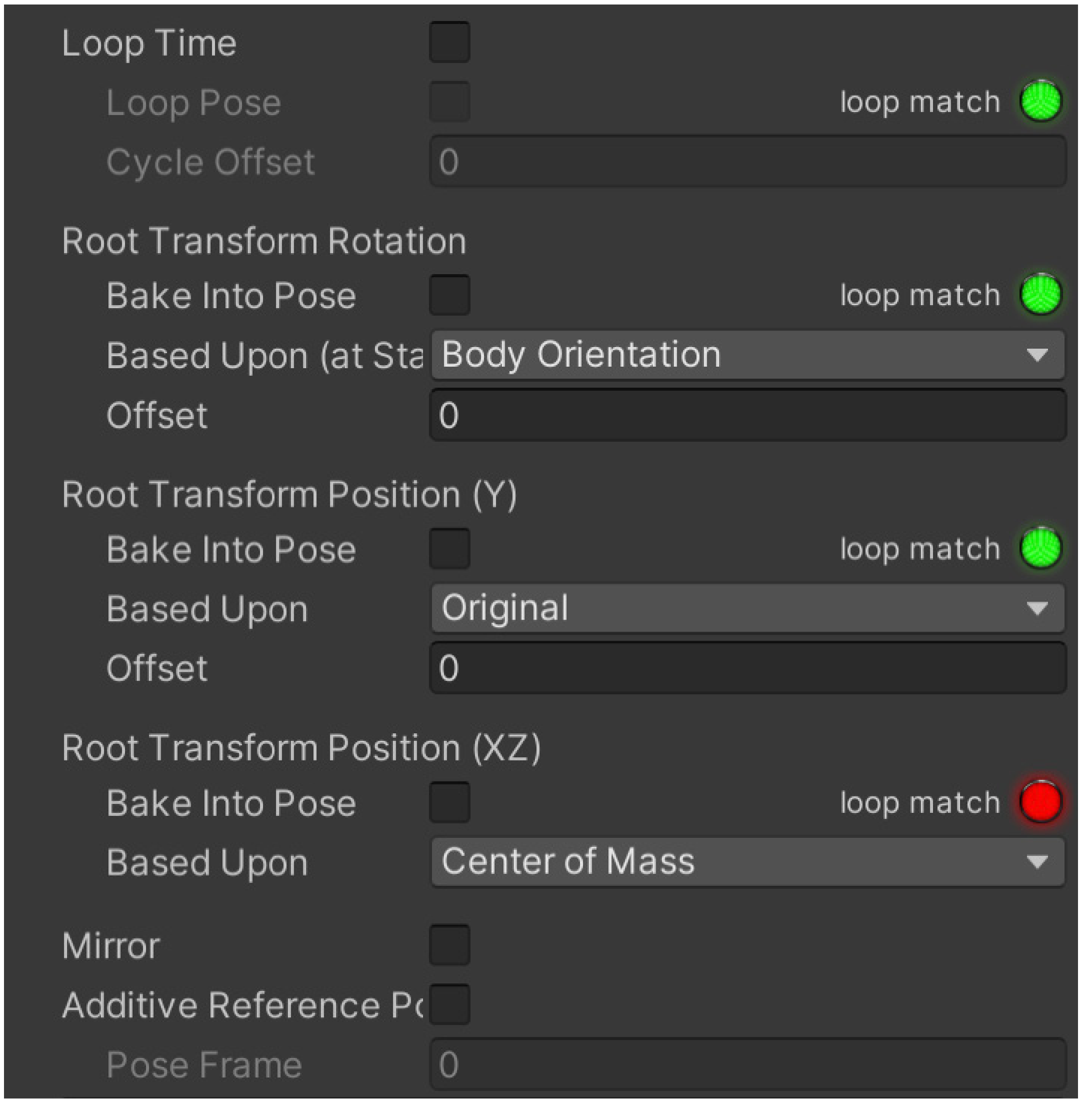


Animator
Controller Ninja_RangeOfMotion
Avatar NinjaAvatar
Apply Root Motion ✓
Update Mode Normal
Culling Mode Cull Update Transforms

```
Clip Count: 11
Curves Pos: 763 Quat: 673 Euler: 90 Scale: 763 Muscles: 14
Generic: 0 PPtr: 0
Curves Count: 7554 Constant: 5902 (78.1%) Dense: 620 (8.2%)
Stream: 1032 (13.7%)
```

Apply root motion to use root motion clips

A character with rigidbody physics can cause conflicts with root motion files. To fix these conflicts you can bake the motion into the clip using the **Bake Into Pose** option.



Loop Time	☐	
Loop Pose	☐	loop match 🟢
Cycle Offset	0	
Root Transform Rotation		
Bake Into Pose	☐	loop match 🟢
Based Upon (at Sta	Body Orientation ▾	
Offset	0	
Root Transform Position (Y)		
Bake Into Pose	☐	loop match 🟢
Based Upon	Original ▾	
Offset	0	
Root Transform Position (XZ)		
Bake Into Pose	☐	loop match 🔴
Based Upon	Center of Mass ▾	
Mirror	☐	
Additive Reference Pc	☐	
Pose Frame	0	

Loop using root motion and the option of baking into the pose

Check this option for the settings **Root Transform Rotation** and **Root Transform Position (Y)**, which is the jump up and down motion. However, to maintain walking forward and backward motions don't check the Bake Into Pose option for the setting **Root Transform Position (XZ)**.

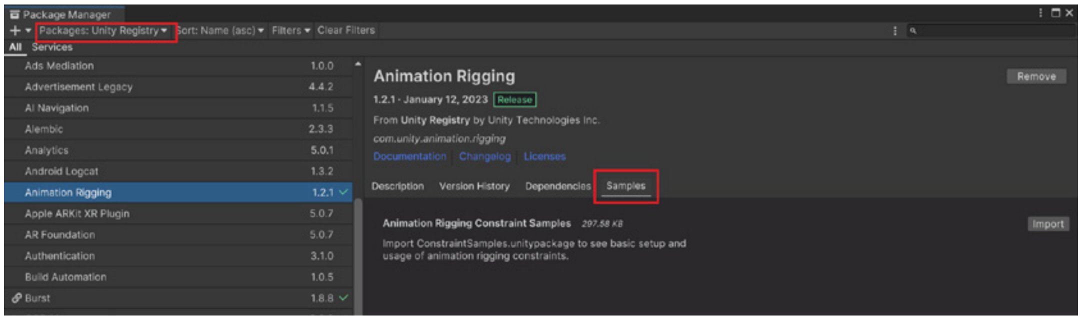
Animation Rigging

In video games, the environment or location of a character will change during gameplay. Often you can't anticipate all the variations you'll need for an animation to make the character move in the world in a believable way. That's why being able to adjust existing animations based on the game variables or specific needs at runtime is key and a time saver.

The Animation Rigging package allows you to create Inverse Kinematic (IK) rigs for any type of character or object inside of Unity. It will work on both generic and humanoid animation types as well as inside the Timeline editor.

Inverse kinematics (IK) is a technique used in animation and robotics to determine the positions and rotations of the joints in a skeletal structure based on the desired position of the end effector, such as a character's hand or foot, allowing for more natural and realistic movement. Imagine you're controlling a character's hand in a game. With inverse kinematics (IK), instead of manually moving each joint of the arm to place the hand where you want it, you simply specify the position you want the hand to reach. The IK system then calculates how to adjust the shoulder, elbow, and wrist joints automatically to achieve that position, resulting in more natural and fluid movement.

Install the Animation Rigging package by going to **Window > Package Manager** and selecting Unity Registry from the top left drop-down. Once you install the package, it's also recommended to download the samples package, which contains example scenes of each rigging constraint so you can see how to implement them in your projects.



Ads Mediation	1.0.0
Advertisement Legacy	4.4.2
AI Navigation	1.1.5
Alembic	2.3.3
Analytics	5.0.1
Android Logcat	1.3.2
Animation Rigging	1.2.1 ✓
Apple ARKit XR Plugin	5.0.7
AR Foundation	5.0.7
Authentication	3.1.0
Build Automation	1.0.5
Burst	1.8.8 ✓

Animation Rigging

1.2.1 · January 12, 2023 Release

From **Unity Registry** by Unity Technologies Inc.

com.unity.animation.rigging

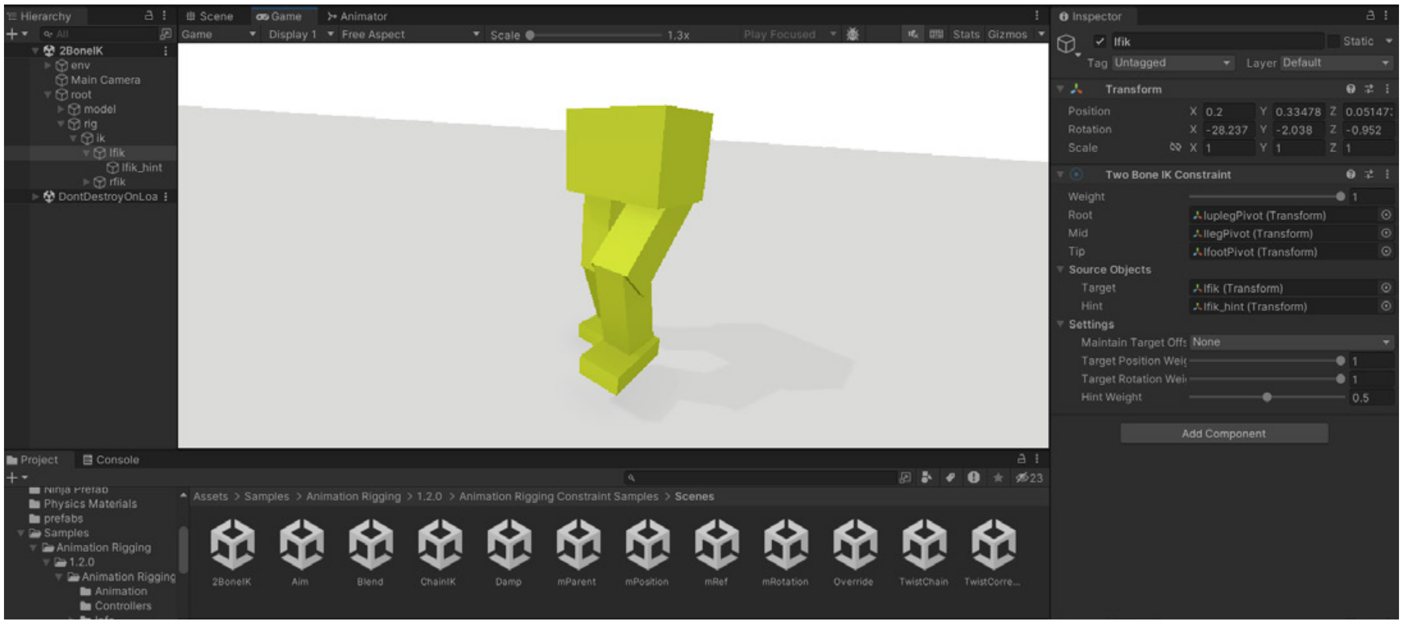
Documentation | Changelog | Licenses

Animation Rigging Constraint Samples *297.58 KB*

Import ConstraintSamples.unitypackage to see basic setup and usage of animation rigging constraints.

Installing Animation Rigging from the Package Manager

The samples are in the Samples folder; preview how they work by entering Play mode and examining their structure in the Hierarchy.



lfik — Static — Tag: Untagged — Layer: Default

Transform

Position	X 0.2	Y 0.33478	Z 0.05147
Rotation	X -28.237	Y -2.038	Z -0.952
Scale	X 1	Y 1	Z 1

Two Bone IK Constraint

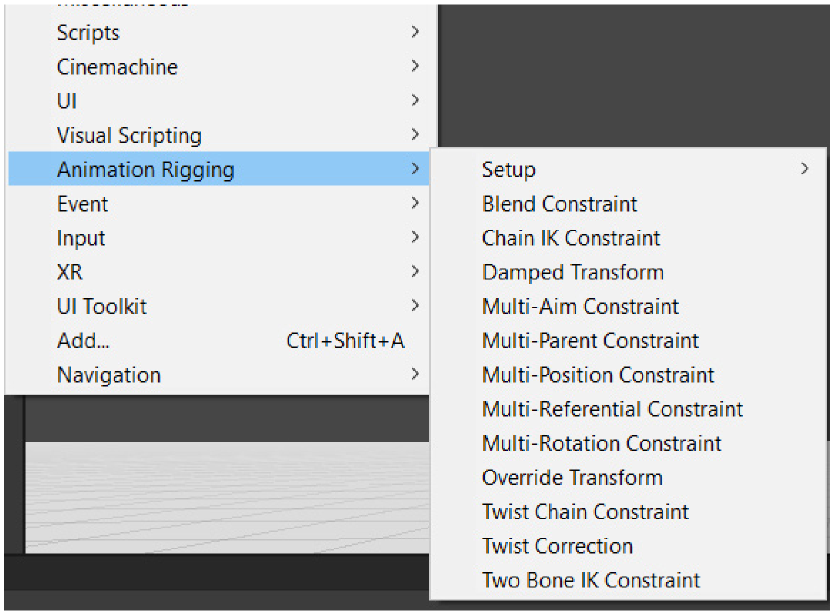
Weight: 1
Root: luplegPivot (Transform)
Mid: llegPivot (Transform)
Tip: lfootPivot (Transform)
Source Objects
Target: lfik (Transform)
Hint: lfik_hint (Transform)
Settings
Maintain Target Offs: None
Target Position Weig: 1
Target Rotation Wei: 1
Hint Weight: 0.5

Add Component

The example scenes demonstrate how each constraint works.

The constraints are based on predefined scripts that can make setting up an IK solution for Unity more easy. The constraints are also modular; instead of using a single IK rig that controls every bone in a character's skeleton, you can mix the individual constraints to suit any character type, including non biped characters such as animals or creatures. You can also turn off parts of the IK rig independently of the rest.

The constraints are available via **Component > Animation Rigging**.

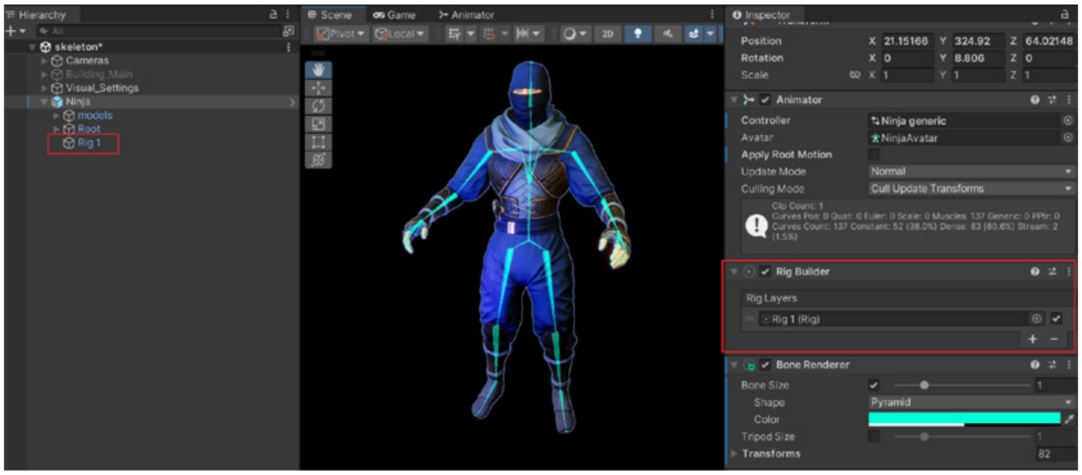


Rigging constraints

Set up an IK Rig

Select your character and go to **Animation Rigging > Rig Setup**. This will add an Animator component and a Rig Builder component. Adding the latter component to this root character GameObject automatically creates a child Rig GameObject of that root, with the default (chanageable) name Rig 1 (see the image below).

You can also display the bone layout by going to **Animation Rigging > Bone Renderer Setup**. This provides a visual representation of the skeletal layout but it's not mandatory for the Rigging package to work. The Bone Renderer component is removed from the rest of the examples in this guide to make the character easier to see.



Position X 21.15166 Y 324.92 Z 64.02148
Rotation X 0 Y 8.806 Z 0
Scale X 1 Y 1 Z 1

Animator
Controller: Ninja generic
Avatar: NinjaAvatar
Apply Root Motion
Update Mode: Normal
Culling Mode: Cull Update Transforms
Clip Count: 1
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 137 Generic: 0 PPtr: 0
Curves Count: 137 Constant: 52 (38.0%) Dense: 83 (60.6%) Stream: 2 (1.5%)

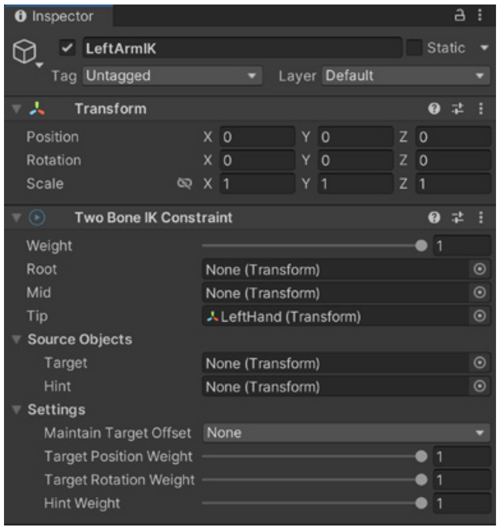
Rig Builder
Rig Layers
Rig 1 (Rig)

Bone Renderer
Bone Size: 1
Shape: Pyramid
Color
Tripod Size: 1
Transforms: 82

IK Rig setup for the ninja character

To add the modular Rig components add an empty "GameObject named Rig 1." You can then choose which constraint to add from the components menu.

A Two Bone IK Constraint is used to add an inverse kinematic constraint to arms, legs, or fingers. Drag the hand from the Hierarchy into the **Tip** slot.



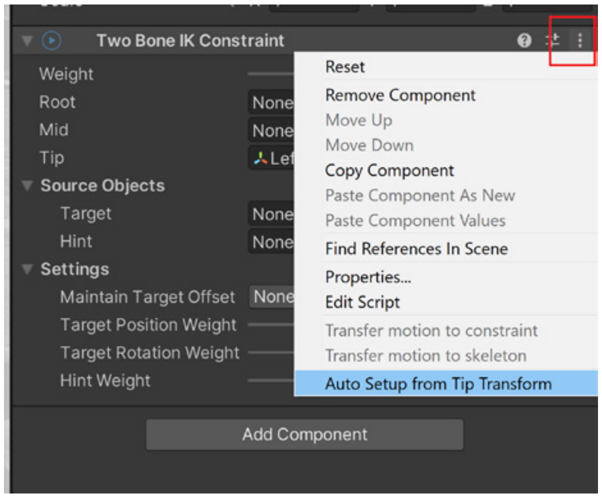
LeftArmIK Static
Tag Untagged Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Two Bone IK Constraint
Weight: 1
Root: None (Transform)
Mid: None (Transform)
Tip: LeftHand (Transform)
Source Objects
Target: None (Transform)
Hint: None (Transform)
Settings
Maintain Target Offset: None
Target Position Weight: 1
Target Rotation Weight: 1
Hint Weight: 1

Add the Tip object to a Two Bone IK constraint

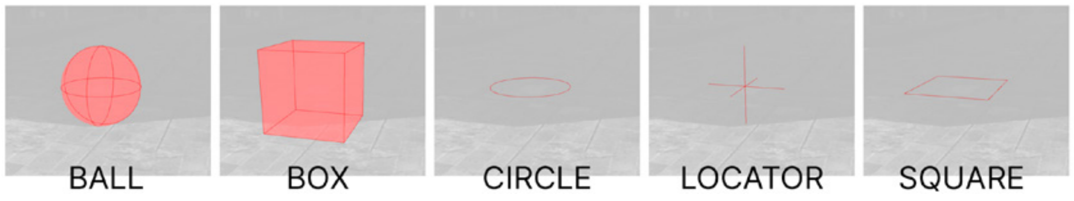
Click the three-dot options icon and choose **Auto Setup from Tip Transform**. This will automatically find the connected bones and add target and hint objects in the correct positions.



Auto Setup from Tip Transform sets up the Two Bone IK Constraint automatically

It will also add effectors to visualize the target and hint objects. The target object will control the hand position and rotation, and the hint object will control the elbow rotation.

There are a variety of effectors to choose from and these can be modified in size, position, rotation, and color.



Types of effectors you use to visualize targets and hints

With both arms using Two Bone IK Constraints they can now be positioned in 3D space. The animation will not affect the arms but will continue to control the rest of the body. This may be useful when having a character leaning against a wall, opening a door, or carrying a box for example.

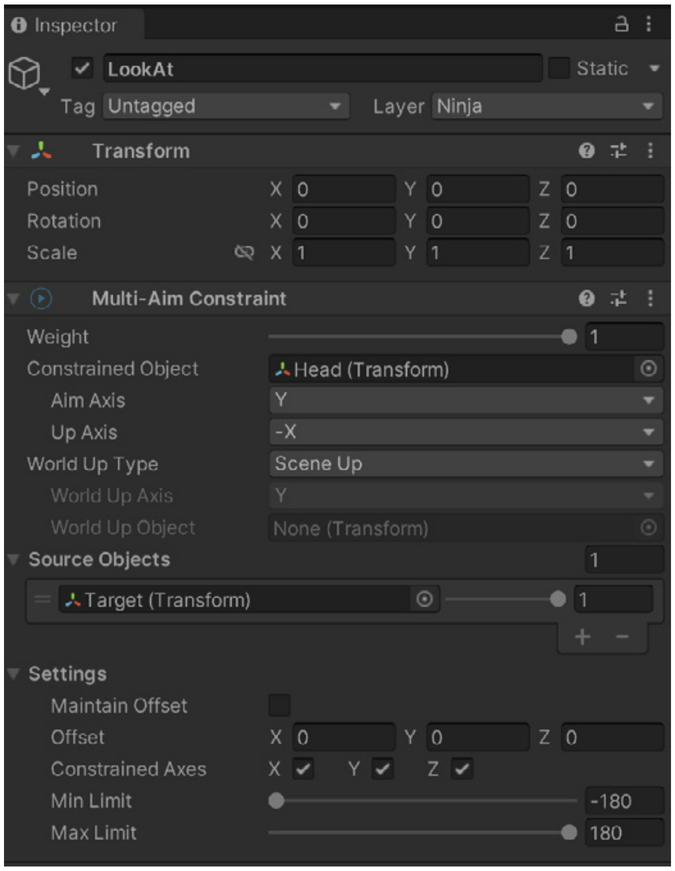
Two Bone IK constraints positioned to enable the character to carry a box

All constraints have a weight slider that affects the overall influence of the targets. At a strength of one the target will completely control the hand position. At a strength of zero it will have no effect over the hand. You can also control the position weight and rotation weight independently.

During Play or Record mode you can move or rotate the target around to affect the position of the character's arms.

You can use a Multi-Aim Constraint to get the character's head to look at a target, like a moving object. The targets are parented to those objects and the character will look at them as they move.

Using the Multi-Aim Constraint to have the character looking at a moving target



Inspector

LookAt — Static
Tag: Untagged — Layer: Ninja

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

Multi-Aim Constraint

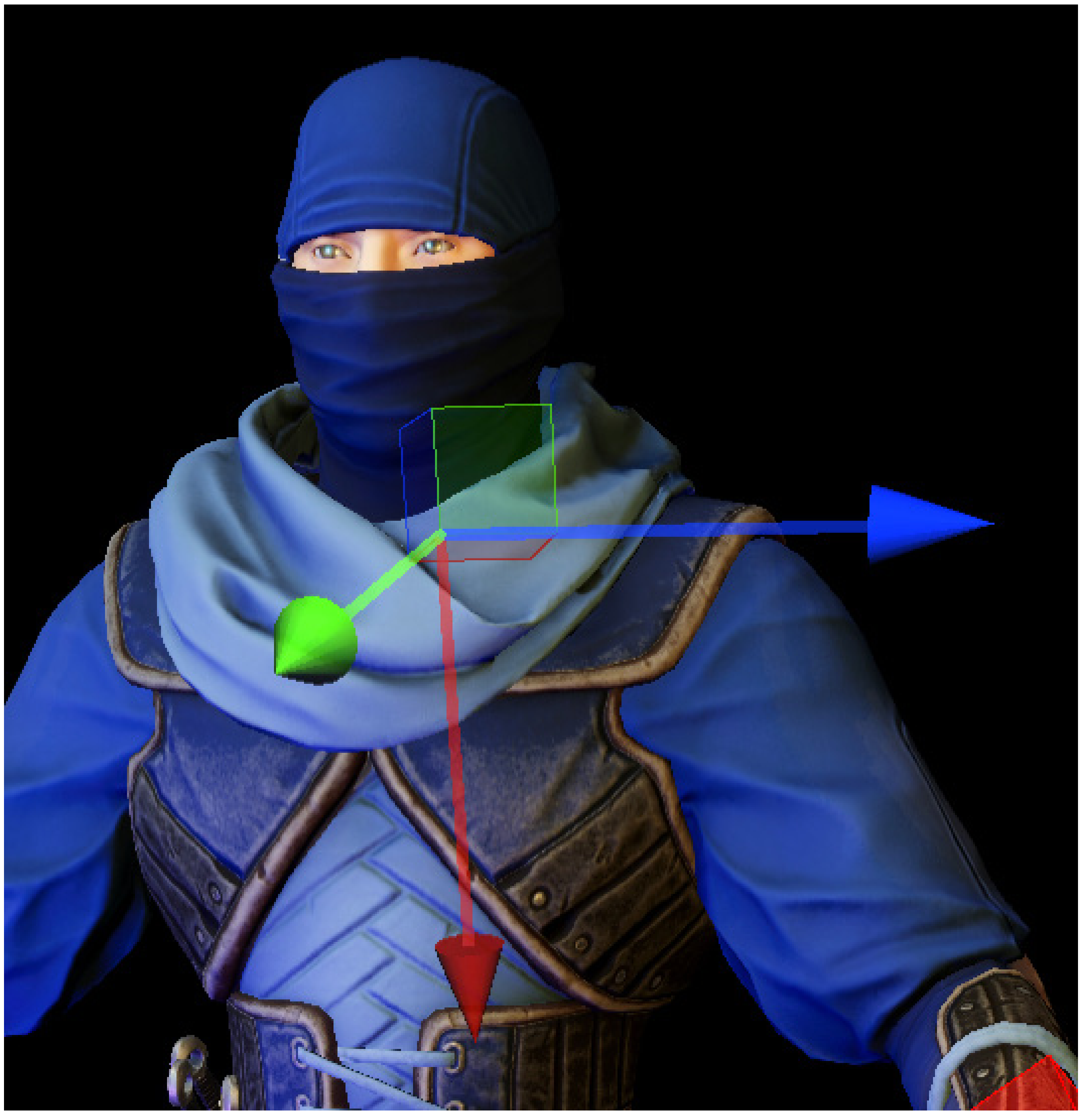
Weight	1
Constrained Object	Head (Transform)
Aim Axis	Y
Up Axis	-X
World Up Type	Scene Up
World Up Axis	Y
World Up Object	None (Transform)
Source Objects	1
Target (Transform)	1

Settings

Maintain Offset	
Offset	X 0 Y 0 Z 0
Constrained Axes	X ✓ Y ✓ Z ✓
Min Limit	-180
Max Limit	180

The Multi-Aim Constraint component using the Scene Up option in the **World Up Type** field to ensure correct rotation

If the head looks like it's rotating incorrectly, check the **Aim Axis** and **Up Axis** direction of the head. Ensure you are in Local space. The head is looking along the Y axis and the Up Axis field is set to - X.

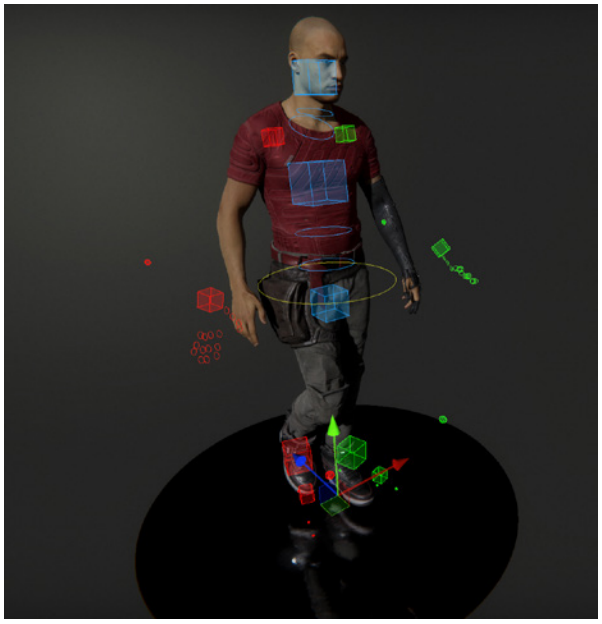

The head is using the Y positive for the Aim Axis and the negative X for the Up Axis.

A minimum and maximum rotation amount can also be set, e.g., you might want to restrict head rotation to -90 and 90.

Multiple Rigs

You can create as many rigs as you like for a character by going to **Animation rigging > Rig setup**. It will create a second rig that gets added to the Rig Builder component. Be aware that any constraints for Rig 1 are calculated independently of Rig 2. Multiple rigs are useful when you want the two or more rigs to be independent of each other.

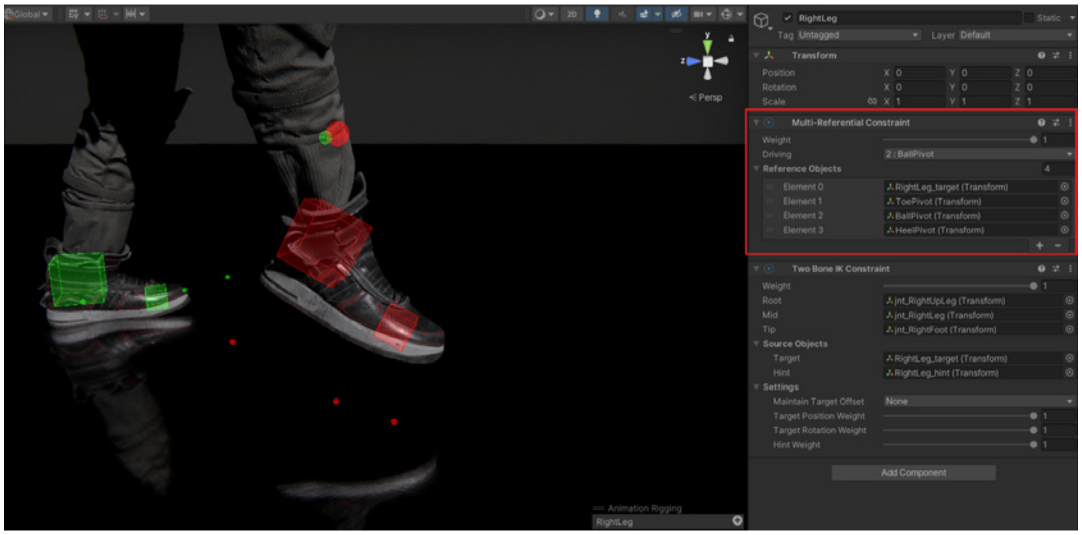
Full-body control rig


An example of a full body control rig

The character Gawain from *The Heretic* is an example of a full body control rig where each body part can be controlled using a range of constraints such as Multi-Parent, Multi-Position and Multi-Aim.

The benefit of this type of set up is the ability to control the character in any given situation. Some of the constraints are IK and will take full control over the character, where some others are overrides that will adjust position and rotations of bones without affecting the underlying animation.

Precise foot or hand positioning is possible with a **Multi-Referential Constraint** that allows the bone positions to be controlled from a specific pivot. In the case of foot positioning, the toes and ball of the foot have been used to set up precise positioning of the foot.



```
RightLeg                                   Static
Tag Untagged              Layer Default
Transform
  Position      X 0    Y 0    Z 0
  Rotation      X 0    Y 0    Z 0
  Scale         X 1    Y 1    Z 1
Multi-Referential Constraint
  Weight                         1
  Driving          2 : BallPivot
  Reference Objects              4
    Element 0      RightLeg_target (Transform)
    Element 1      ToePivot (Transform)
    Element 2      BallPivot (Transform)
    Element 3      HeelPivot (Transform)
Two Bone IK Constraint
  Weight                         1
  Root             jnt_RightUpLeg (Transform)
  Mid              jnt_RightLeg (Transform)
  Tip              jnt_RightFoot (Transform)
  Source Objects
    Target         RightLeg_target (Transform)
    Hint           RightLeg_hint (Transform)
  Settings
    Maintain Target Offset   None
    Target Position Weight   1
    Target Rotation Weight   1
    Hint Weight              1
```

Driving bone pivots using the Multi-Referential Constraint component

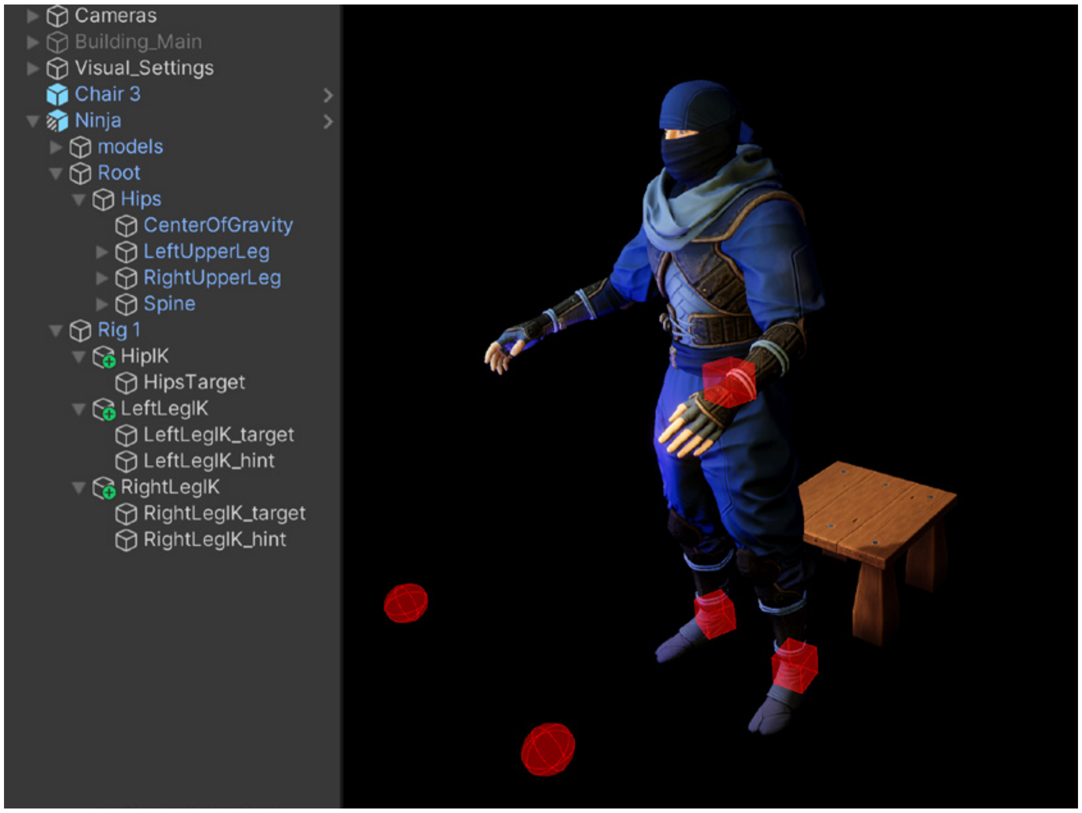
Target points must be set up for each point, in this case the heel, ball, and toe of the foot. These are then added to the constraint along with the right leg target (this is the target that is going to be controlled by the pivot of each point).

During Play or Record mode the driving pivot can be set from the drop-down. Select the target point and rotate to see the foot rotate from that pivot point. This is covered in more detail in this Unite session.

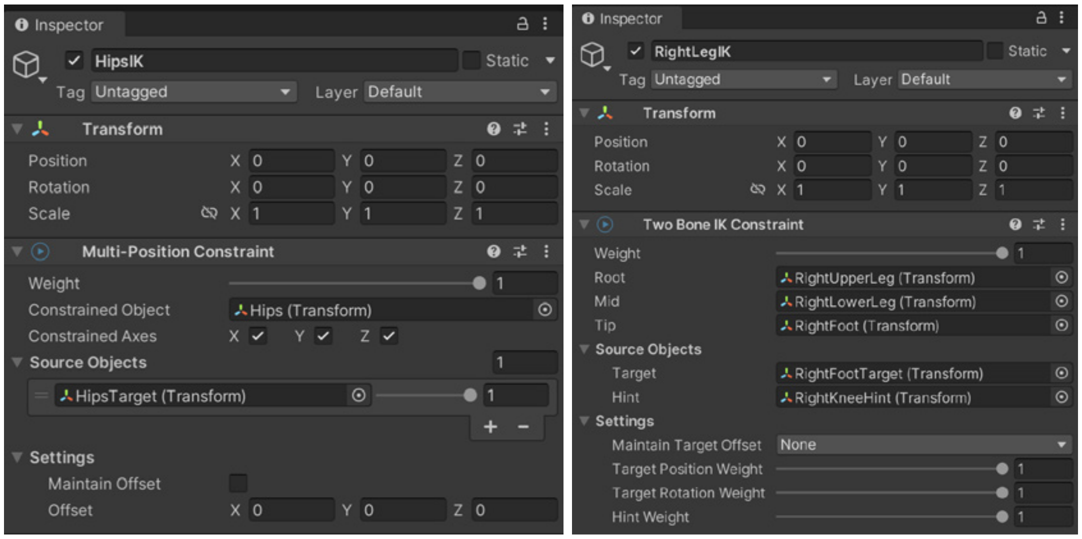
Foot IK

Most animators are familiar with foot IK because it's a key concept in crafting authentic-looking animations. Foot IK reduces foot sliding and feet dipping below the ground. It ensures that feet stay in the correct position, for example, when a character is sitting in a chair.

The example below is a new rig with a hip control, to begin with. Placing this constraint above the two feet constraints is important as IK is calculated in the order it appears in the Hierarchy. So it calculates the hip movement first and then the feet position. If it did it the other way around the feet would be calculated before the hips have moved.



Hips and Legs IK



Inspector

HipsIK — Static

Tag: Untagged — Layer: Default

Transform

Position: X 0, Y 0, Z 0
Rotation: X 0, Y 0, Z 0
Scale: X 1, Y 1, Z 1

Multi-Position Constraint

Weight: 1
Constrained Object: Hips (Transform)
Constrained Axes: X ✓ Y ✓ Z ✓
Source Objects: 1
HipsTarget (Transform): 1

Settings
Maintain Offset: ☐
Offset: X 0, Y 0, Z 0

Inspector

RightLegIK — Static

Tag: Untagged — Layer: Default

Transform

Position: X 0, Y 0, Z 0
Rotation: X 0, Y 0, Z 0
Scale: X 1, Y 1, Z 1

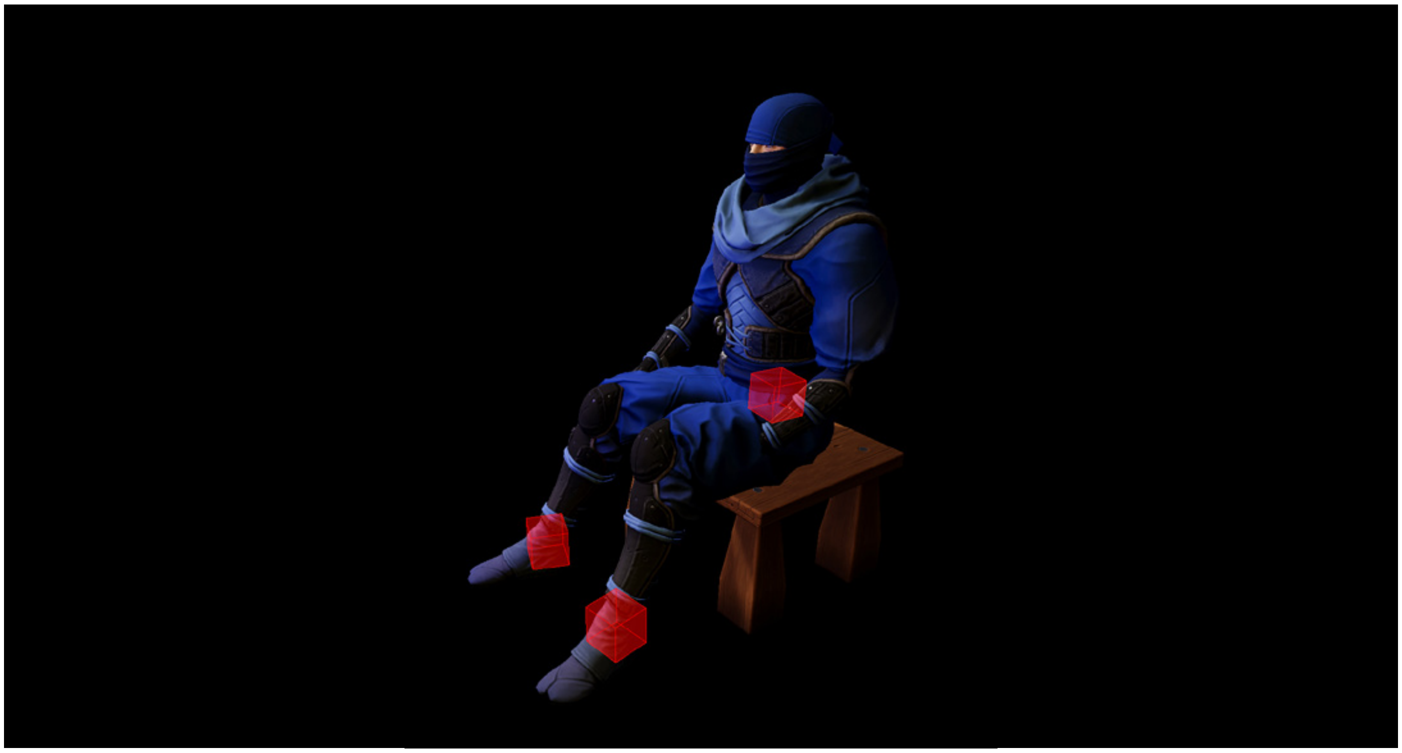
Two Bone IK Constraint

Weight: 1
Root: RightUpperLeg (Transform)
Mid: RightLowerLeg (Transform)
Tip: RightFoot (Transform)
Source Objects
Target: RightFootTarget (Transform)
Hint: RightKneeHint (Transform)
Settings
Maintain Target Offset: None
Target Position Weight: 1
Target Rotation Weight: 1
Hint Weight: 1

Settings for Hips and Legs IK

The two feet use a Two Bone IK Constraint to ensure the correct bending of the knees as the character sits. The feet will remain on the ground the entire time.

During Play mode the hip target can now be positioned to get the character to sit on the chair. The left and right leg hints can be used to adjust the knee positions if needed.



Easy sitting animation using IK with the feet remaining planted on the ground

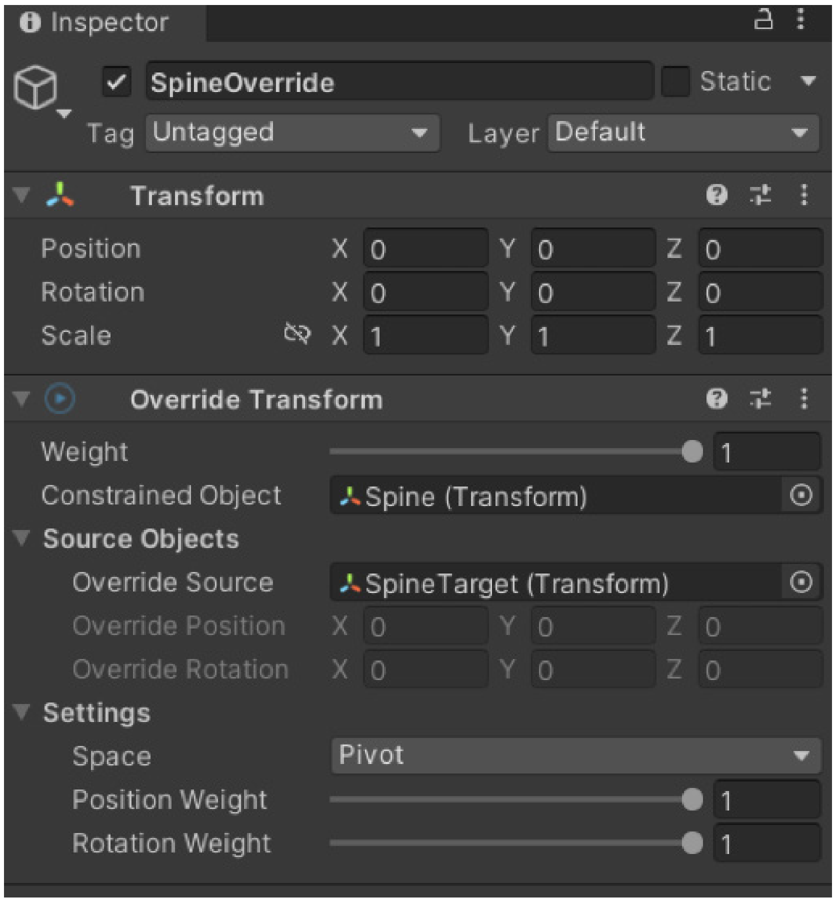
Merging IK with animations

Usually, IK components take full control over a body part, meaning the animations will no longer be run on that part.

The **Override Transform** component, however, can merge animations with IK control and is useful for making small adjustments to existing animations without stopping them from working on different body parts.

To set this up you will need targets set up at each of the bone positions. The best way to do this is to right-click a bone in the character's skeleton, for example the spine, and create an empty GameObject.

You can assign the targets an effector for visualization and drag them down to the rig section. The Override Transform component is added to each of these targets, the bone to modify is added in the **Constrained Object** field, the target itself is added to the **Override Source** field.


Inspector

SpineOverride ✓ Static

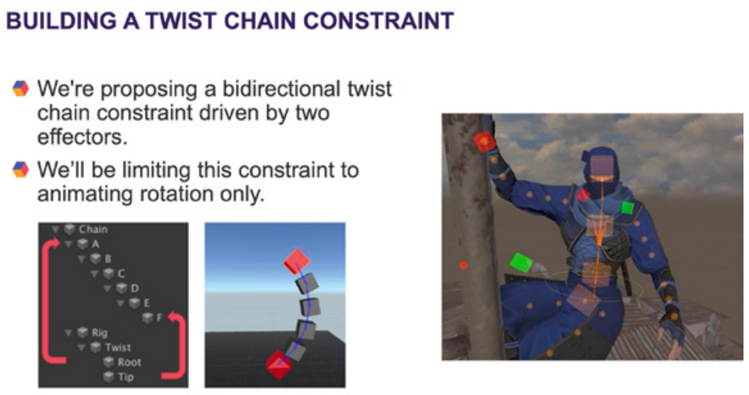
Tag Untagged Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Override Transform
Weight 1
Constrained Object Spine (Transform)
Source Objects
Override Source SpineTarget (Transform)
Override Position X 0 Y 0 Z 0
Override Rotation X 0 Y 0 Z 0
Settings
Space Pivot
Position Weight 1
Rotation Weight 1

Overrides will adjust the bones while the animation is still controlling them.

You can now animate these targets to make adjustments such as spine rotation, shoulder adjustments, neck position, or leg rotations while the animation continues playing. See this video for a good example of this process.



BUILDING A TWIST CHAIN CONSTRAINT

- We're proposing a bidirectional twist chain constraint driven by two effectors.
- We'll be limiting this constraint to animating rotation only.

Watch session

Freeform animation rigging: Evolving the animation pipeline

In this GDC talk Dave Hunt shares how Unity implemented freeform animation in the Animation Rigging package, enabling animators to non-destructively preserve motion content while changing the structure of the control rig without the help of a rigging specialist.

Benefits of using animation rigging

— The easy-to-use IK system is accessible to all team members.

— Teams can work inside Unity to directly iterate on the rig to achieve the desired animations.

— The playback performance is efficient even with multiple constraints on a single character.

— Real-time IK updates for preview and animation purposes help to make iteration and keyframing efficient

— The modular rig system allows you to create custom IK rig setups for biped and non biped characters.

— The modular system makes creating rigs for different types of characters efficient

— Weights on each modular rig section make the system customizable.

— The same rig setup can be copied to different characters.

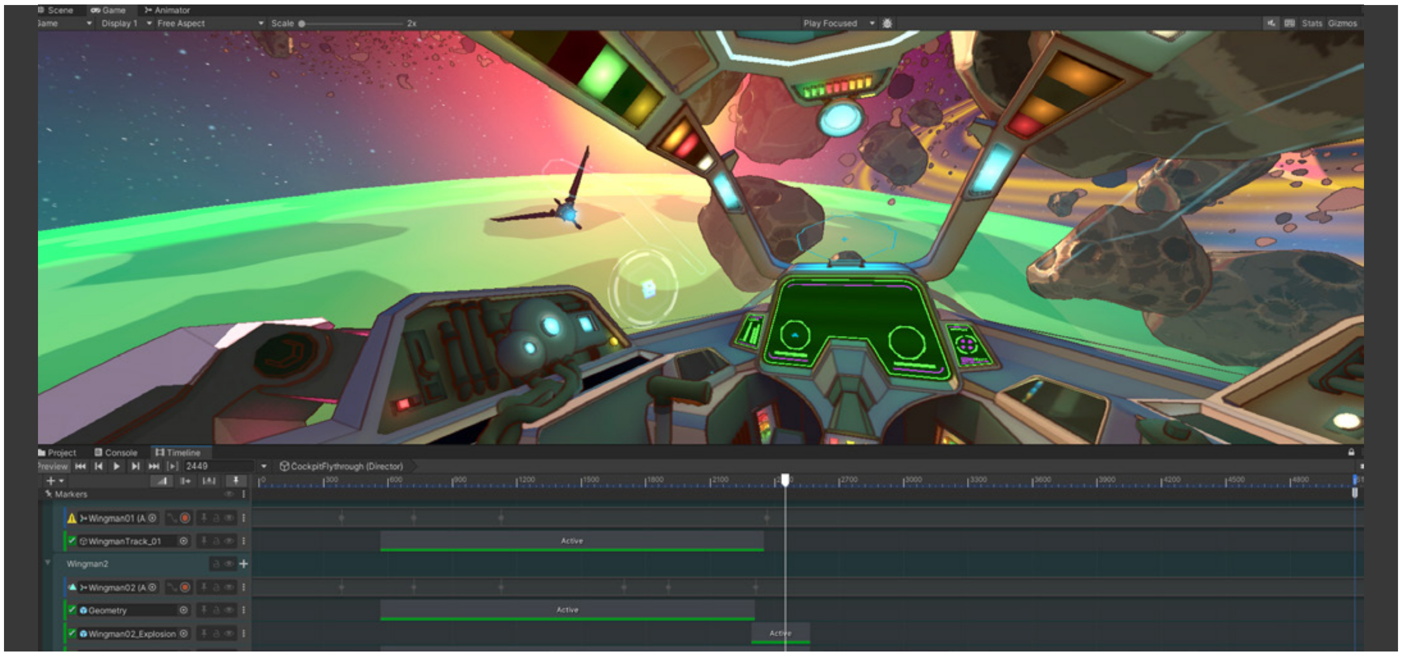
— The code is open source and modifiable by developers.

— Timeline allows for multiple constraint overrides to achieve sophisticated results.

— Bidirectional motion transfer allows you to bake the motion back to the character. skeleton, or to bake motion from a skeleton to an IK control rig

Timeline for animated cutscenes

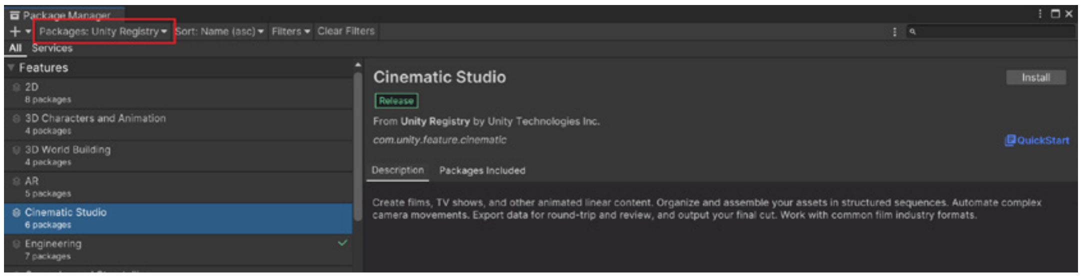
Timeline is an animation editor that looks and works like a video editor, enabling you to create linear animated sequences directly in Unity. You can control animations on all the objects in your scene from a single location, as well as control when the animations will occur relative to all the other objects and characters.



Timeline example from the URP 3D Sample cockpit environment

It works with both generic and humanoid animation types and supports the Animation Rigging package.

In the Package Manager, there are a number of packages available in the Cinematic Studio collection that are useful for creating animated sequences. To install this go to **Window > Package Manager**.



Cinematic Studio

From **Unity Registry** by Unity Technologies Inc.

com.unity.feature.cinematic

Create films, TV shows, and other animated linear content. Organize and assemble your assets in structured sequences. Automate complex camera movements. Export data for round-trip and review, and output your final cut. Work with common film industry formats.

Cinematic Studio package from the Package Manager

The packages included are:

— Timeline: Create playable sequences such as cutscenes

— Sequences: Create movies, cinematics, or any other linear content

— FBX Exporter: Send geometry, lights, cameras, and animation made in Unity to 3D modeling software

— Alembic: Bakes animation data into a file so that you can play the animation in Unity

— Cinemachine: Advanced camera system that solves the complex mathematics and logic of tracking targets, composing, blending, and cutting between shots

— Recorder: Captures gameplay footage as video files or image sequences at optimal frame rates for trailers, tutorials, and other types of content

Animated cutscenes

Traditionally, cutscenes in games were created in software like Autodesk 3ds Max or Maya, or Blender, and then rendered out as video files that were played back during the cutscenes. The video files were huge and needed to be compressed using video encoding, which could diminish their overall quality.

But since the addition of the Cinemachine and Timeline systems to Unity, you can now play back cinematic-quality cutscenes directly inside the game engine with no need for video compression.

Enemies is an animated short created in Unity using Timeline.

Adding a Timeline sequence to your scene

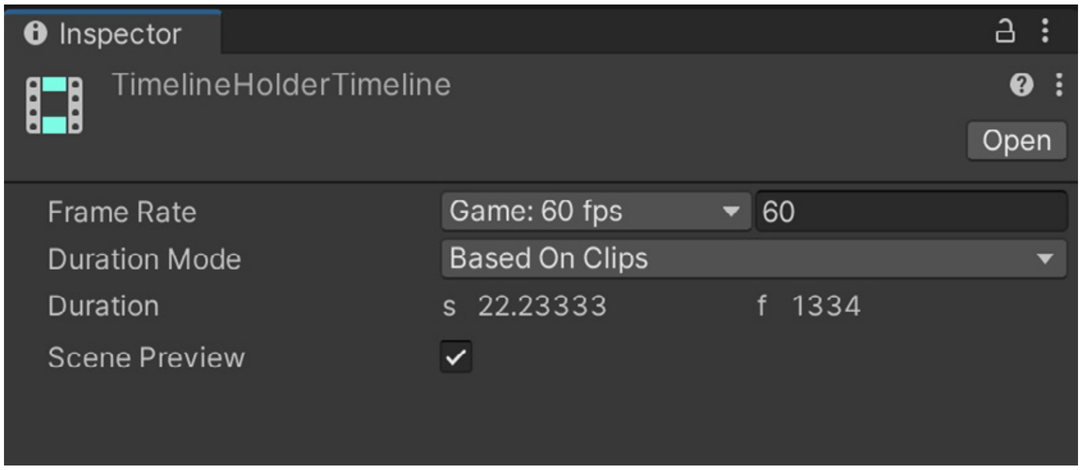
A timeline sequence needs to be included as part of your current scene and stored in a single location. Start by creating an empty GameObject and call it TimelineHolder. Open the Timeline window by going to **Window > Sequences > Timeline**.

Select the TimelineHolder object in the Hierarchy and create a Timeline asset. This will store all the data on the timeline including any animated clips.

It will also create a Playable Director component that binds the timeline to the TimelineHolder GameObject. Now you'll have full access to the track editor when you have TimelineHolder selected in the Hierarchy.

Clicking on the Timeline asset in the Project window allows you to define the properties for the Timeline sequence, including the frame rate, which is 60fps by default, but you can change it to a range of other frame rates.

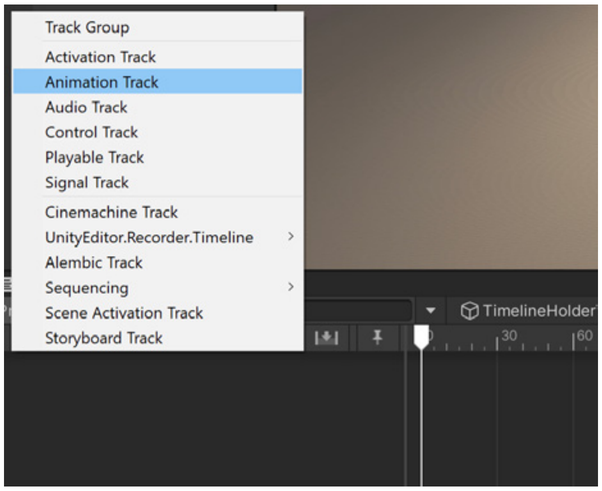
You can also set a specific duration for the Timeline sequence or extend it based on how many clips you add.



Frame Rate: Game: 60 fps, 60
Duration Mode: Based On Clips
Duration: s 22.23333 f 1334
Scene Preview: ✓

Timeline properties

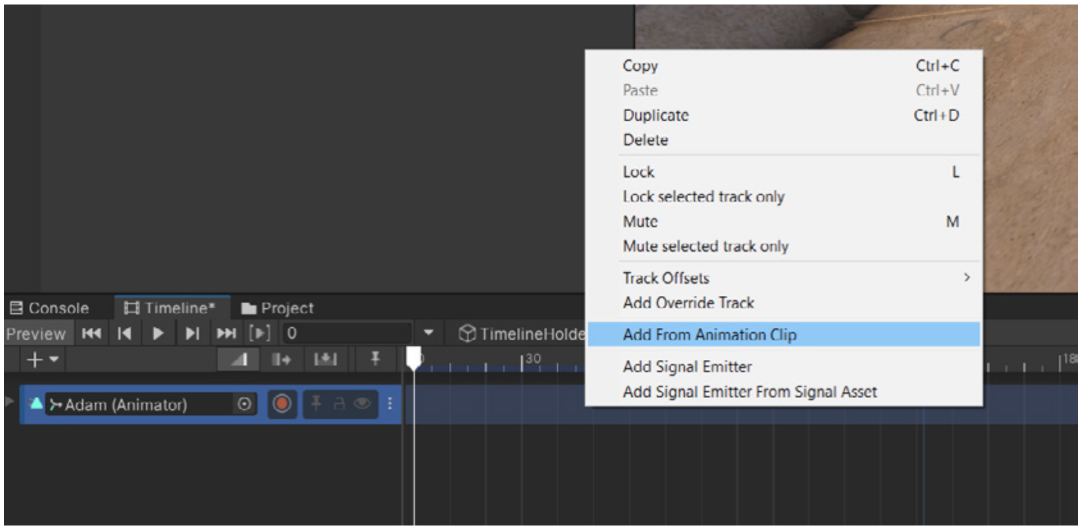
Adding animation clips



Adding an animation track

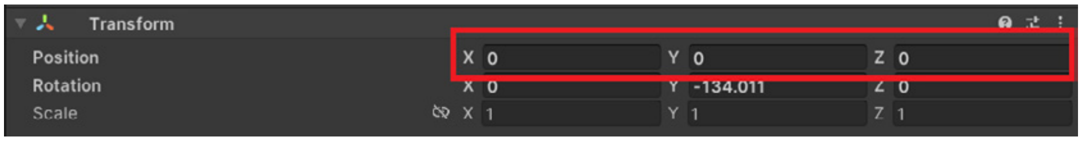
Click the “plus” icon in the Timeline window to create a track and choose the type **Animation Track**.

Drag animation clips directly onto the track and move them into place via click and drag. You can also right-click the track and choose Add from Animation Clip, then select the chosen animation.



Right-click and choose the option **Add from Animation Clip** to choose an animation clip.

Be aware that your character might disappear when you do this if it has a root origin of world 0. This means the character will jump to a world position of 0 on X, Y, and Z and will begin playing the animation there. This might be far away from where you want your character to be.

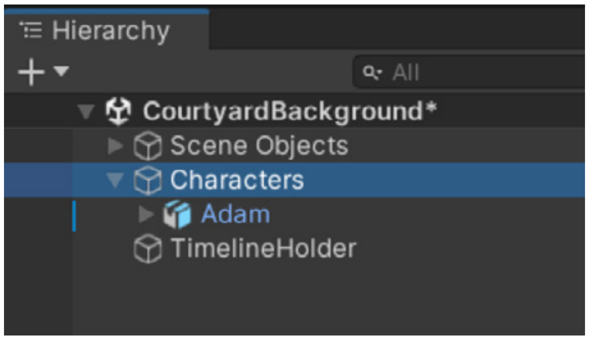


	X	Y	Z
Position	0	0	0
Rotation	0	-134.011	0
Scale	1	1	1

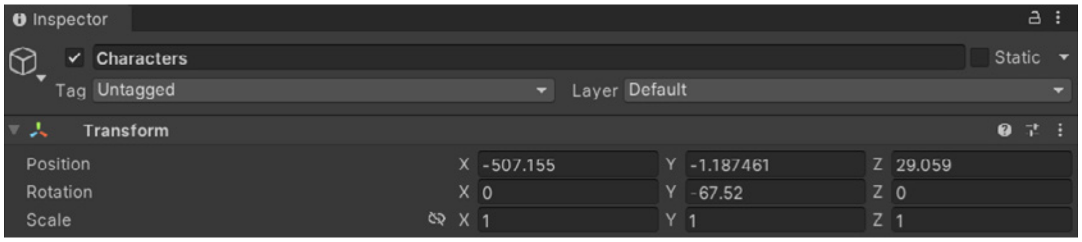
Root origin of 0 moves the character to world position 0

To fix this, create an empty GameObject at the character's current position and place the character inside it to parent it.

Child objects are relative to the parent object's position. So, although the child object in this case will still have a Transform position of 0 on X, Y and Z axis, these are no longer the root origin of 0 in world space, but are now the current position of the empty GameObject.



The Adam Character is now a child of an empty GameObject called Characters.



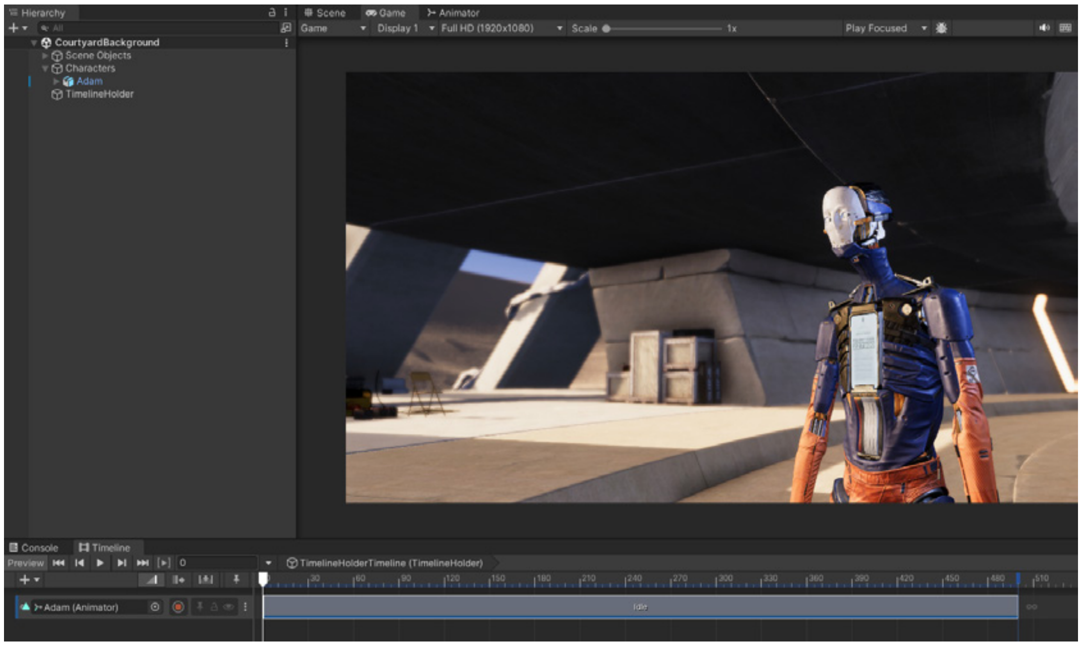
Characters
Tag Untagged
Layer Default

```
Position  X -507.155  Y -1.187461  Z 29.059
Rotation  X 0         Y -67.52     Z 0
Scale     X 1         Y 1          Z 1
```

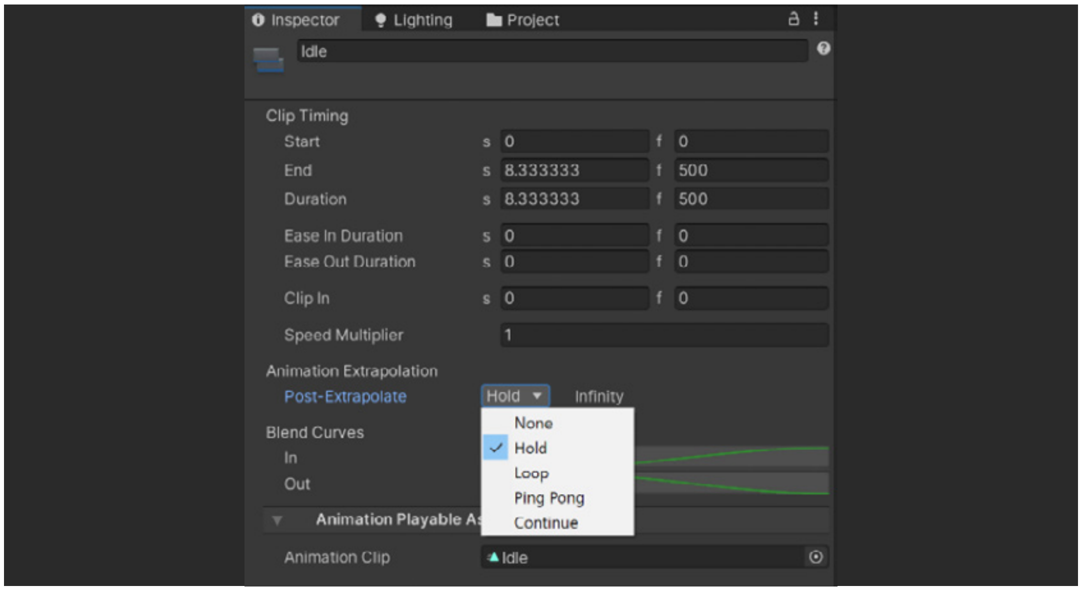
The position of the empty Characters GameObject

Now the Adam character plays the animation at the correct position.

Animation clips appear as solid blocks on the track in the Timeline window that you can drag forward or backward in time across the length of the track.



Animation clip for the Adam character on the timeline and the clip options in the Inspector



Idle

Clip Timing

Start	s	0	f	0
End	s	8.333333	f	500
Duration	s	8.333333	f	500
Ease In Duration	s	0	f	0
Ease Out Duration	s	0	f	0
Clip In	s	0	f	0
Speed Multiplier		1		

Animation Extrapolation

Post-Extrapolate: Hold ▾ Infinity

None
✓ Hold
Loop
Ping Pong
Continue

Blend Curves

In

Out

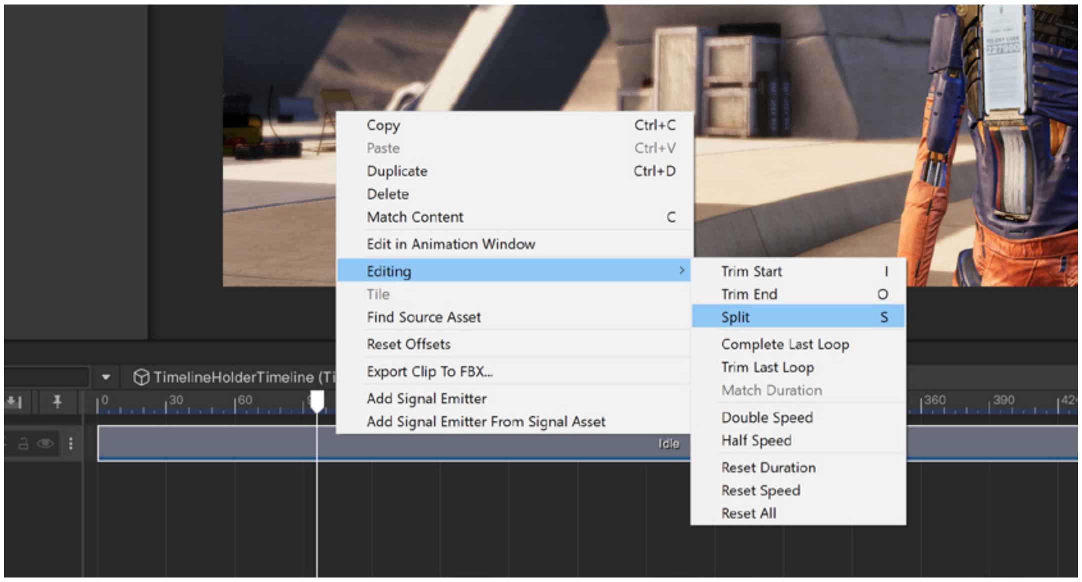
Animation Playable As

Animation Clip: Idle

Post extrapolation for an animation clip

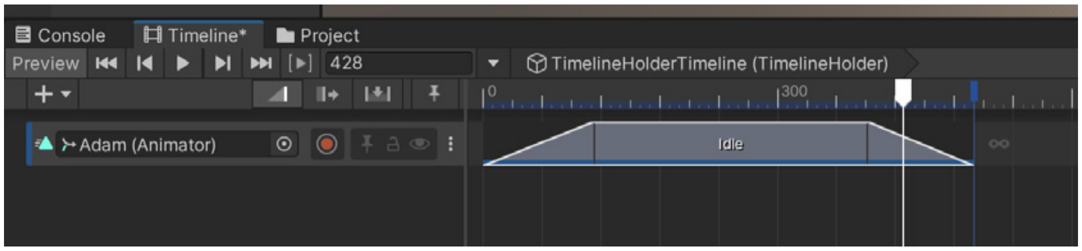
Each clip has a pre- and post-extrapolation method if there are spaces to the left or right of the clip. This determines what happens to the animation before and after it plays. By default it will be set to **Hold**, which creates a still pose from the first or last frame of the animation.

Much like a video editor you can cut or resize the animation clips by right-clicking on the clip and choosing Editing. The shortcut keys are also displayed to make editing more efficient.

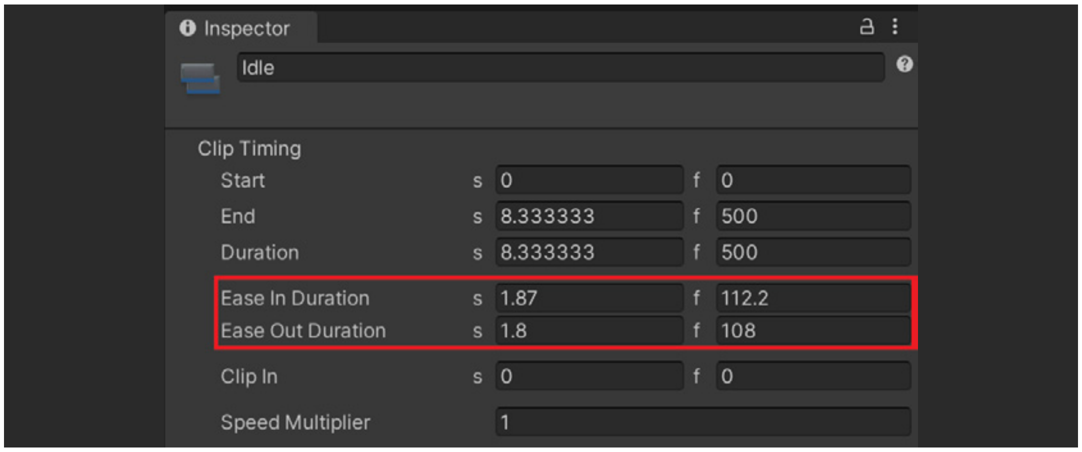


Editing options for animation clips

You can adjust a clip to fade in and out if there are no other clips to the left or right of it. This prevents popping and leads to a gradual change of pose by adjusting the blend in and out values in the Inspector.



Fading the in and out points of an animation clip



Inspector

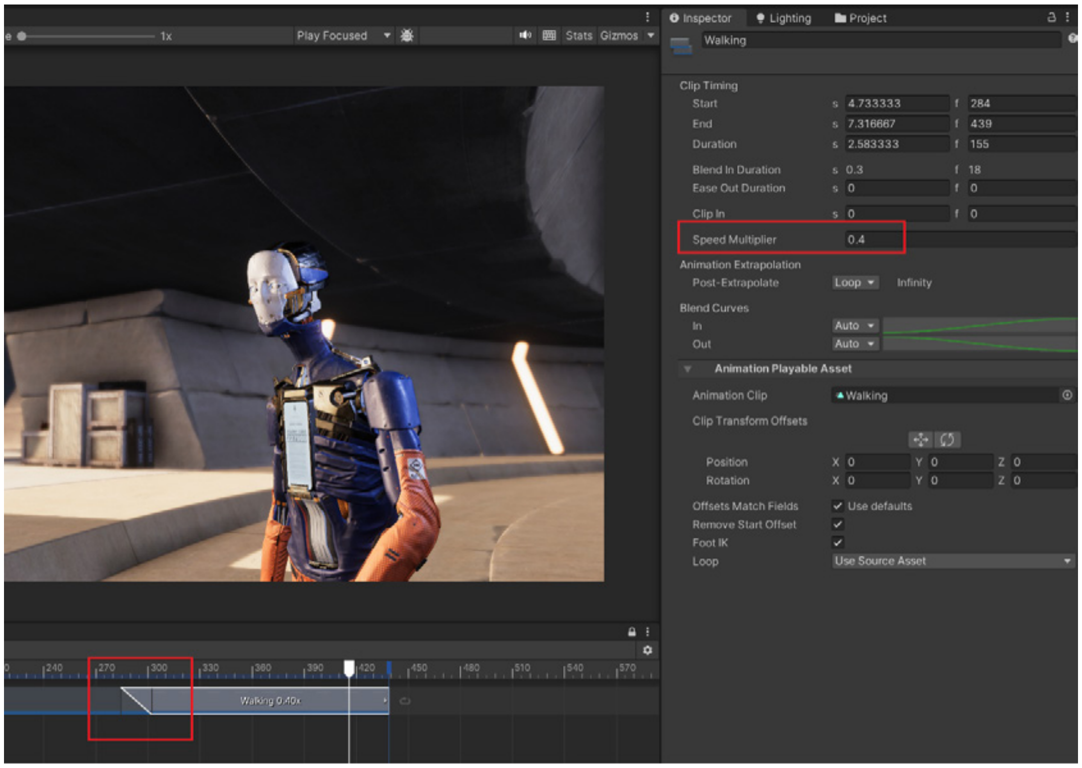
Idle

Clip Timing

Start	s	0	f	0
End	s	8.333333	f	500
Duration	s	8.333333	f	500
Ease In Duration	s	1.87	f	112.2
Ease Out Duration	s	1.8	f	108
Clip In	s	0	f	0
Speed Multiplier		1		

Adjusting the duration of the fade in and out

Adding a second animation allows you to then blend them together; drag the second animation into the first to create a blend overlap.



Walking

Clip Timing

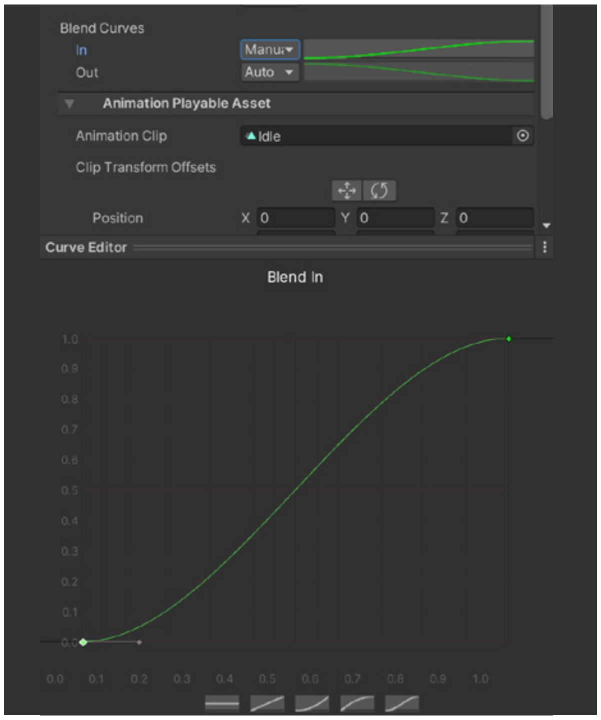
Start	s	4.733333	f	284
End	s	7.316667	f	439
Duration	s	2.583333	f	155
Blend In Duration	s	0.3	f	18
Ease Out Duration	s	0	f	0
Clip In	s	0	f	0
Speed Multiplier		0.4		

Animation Extrapolation
Post-Extrapolate: Loop, Infinity

Blend Curves
In: Auto
Out: Auto

Animation Playable Asset
Animation Clip: Walking
Clip Transform Offsets
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Offsets Match Fields: ✓ Use defaults
Remove Start Offset: ✓
Foot IK: ✓
Loop: Use Source Asset

Blending two animations together on the timeline



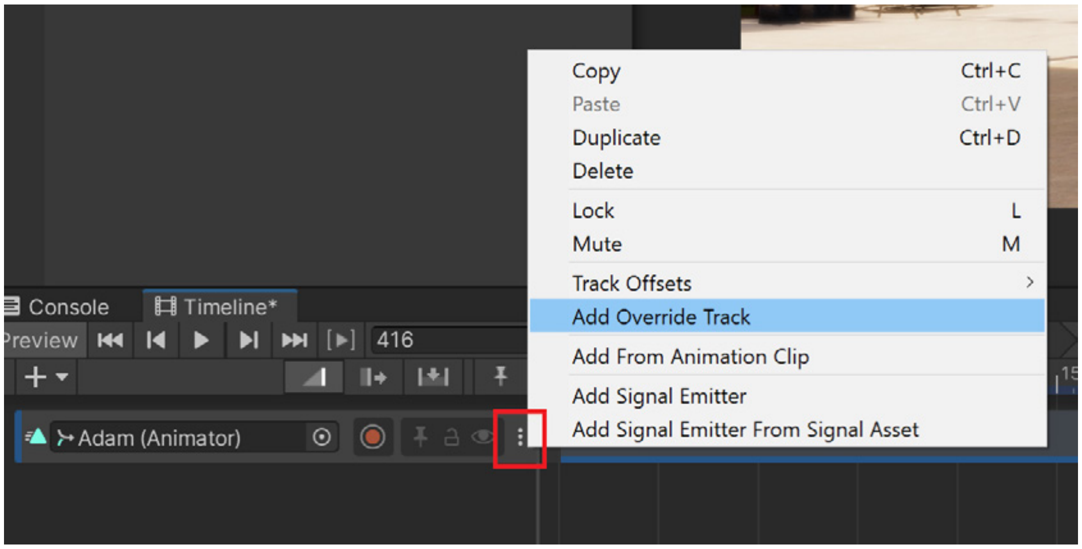
Blend Curves
In: Manual
Out: Auto
Animation Playable Asset
Animation Clip: Idle
Clip Transform Offsets
Position X 0 Y 0 Z 0

Adjusting the in curve of the blend

You also have options to change a clip's playback speed using the **Speed Multiplier** setting. If the speed is set to anything except 1 you'll notice the dotted line above the animation track. Set the blend curves to Manual and click the curve to change the method of easing between the two clips. There are presets at the bottom, or create your own curve by adjusting the bezier handles.

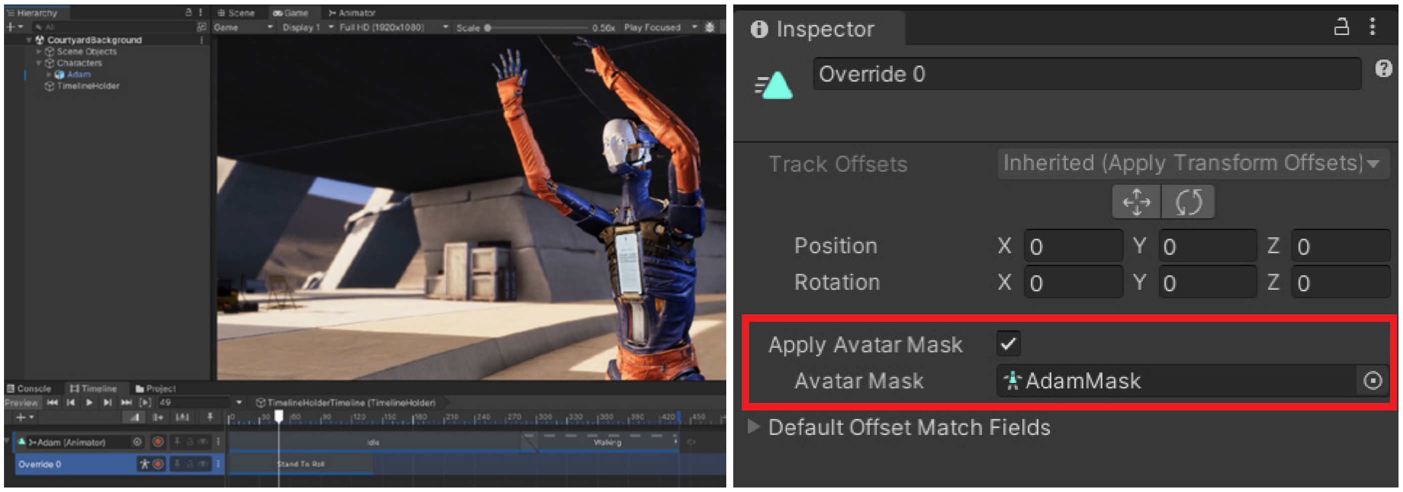
Add track overrides

You can merge two or more animations together in the same way as in the Animator window, by using avatar masks. Click the three-dot track options icon and choose **Add override track**.



Adding an override track to an animation track

Click the track to add an avatar mask. The animations you place on this track will override the bones on the base track, allowing you to create interesting, unique animations.

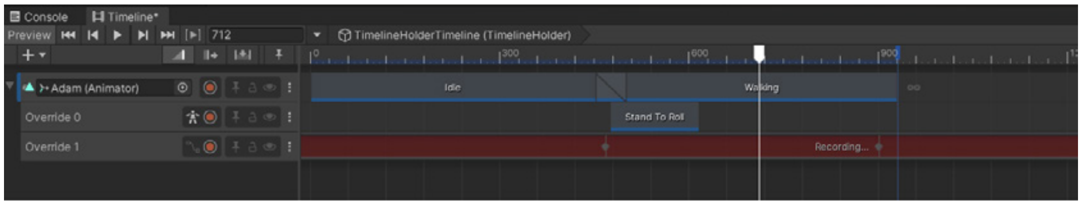


Use an avatar mask on an override track to merge two animations together.

If you are using an in-place animation that doesn't contain root motion you can use an override track to control the position, rotation, or scale of the character.

You can add as many override tracks as you like to an animation track, but if two override tracks control the same bone or bones, then the override track that is furthest down the list will override any that are above it.

Click **Record** on a new override track to add keyframes to an infinite clip as you adjust the position of the character. During recording the track will turn red.



Recording the movement of the character using an override track

Right-click and choose **Convert to Clip Track** to turn keyframes into an animation clip. That gives you access to all the options available for an animation clip including speed, looping, and blend style.

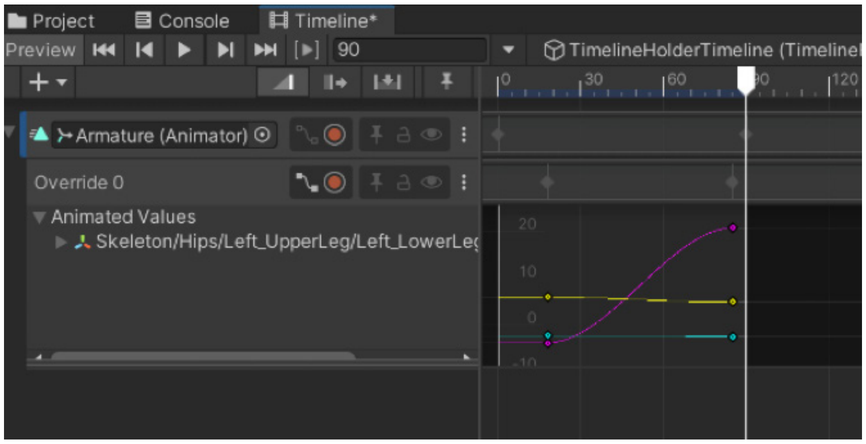
Double-click the animation clip to open the keyframes in the Animation window. From here you can adjust the keyframes and add events in the same way as covered earlier in this e-book.

You can duplicate animation clips by pressing **Ctrl + D** or going to **Edit > Duplicate**. You'll notice that the character position will jump back to its origin when transitioning from the first to the second clip. To fix this, right-click the second clip and choose **Match Offsets to Previous Clip**. Now it will act like an animation that uses root motion, and the character will not jump back to its origin point but will continue from the last position of the previous clip. You can also match offsets to the next clip if there is one to the right of your current clip.

Keyframing

Timeline also allows you to create animations directly from this window. For generic animation types you can use forward kinematics (FK) to adjust the rotation of each bone. Overrides can also be used to adjust and refine the position of bones.

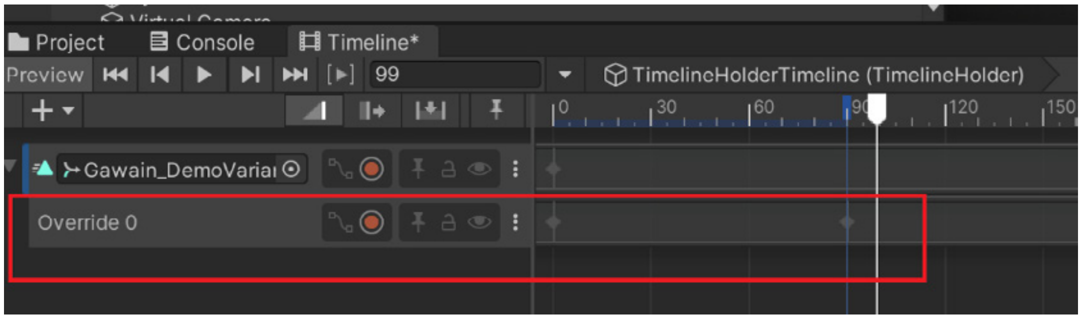
Base animations can also be adjusted with keyframing by using overrides which allow you to customize the animation. You can also access curves here, to refine the easing between keyframes.



Generic rig keyframe overrides for an existing animation and curves adjustment

You can’t animate humanoid animation types using forward kinematic keyframing. Instead, use the inverse kinematics (IK) rig in the Animation Rigging package to animate the position of the targets. See the Animation Rigging section in this e-book.

In order to see the rig update in the scene you need to first set an initial transform keyframe for the parent GameObject that contains the Rig Builder component. As you add override tracks, adjust the position and rotation of the targets and see them update in real-time in the Scene view.

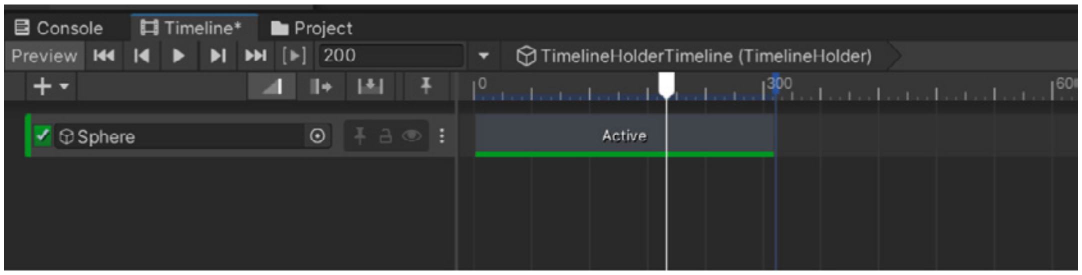


Animating IK rig adjustment in real-time

Track types

Activation track

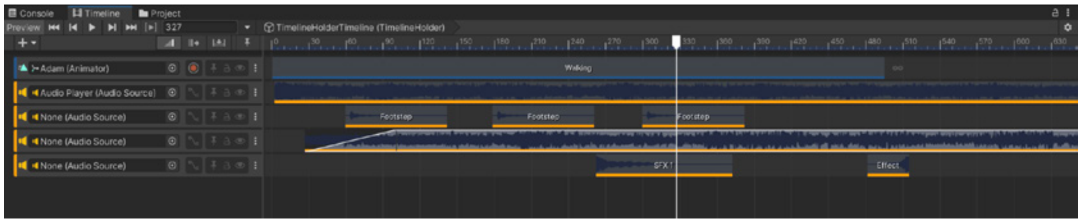
Use the activation track to switch objects on for the duration of the block during the animation.



Active block that switches objects on for the duration of the block

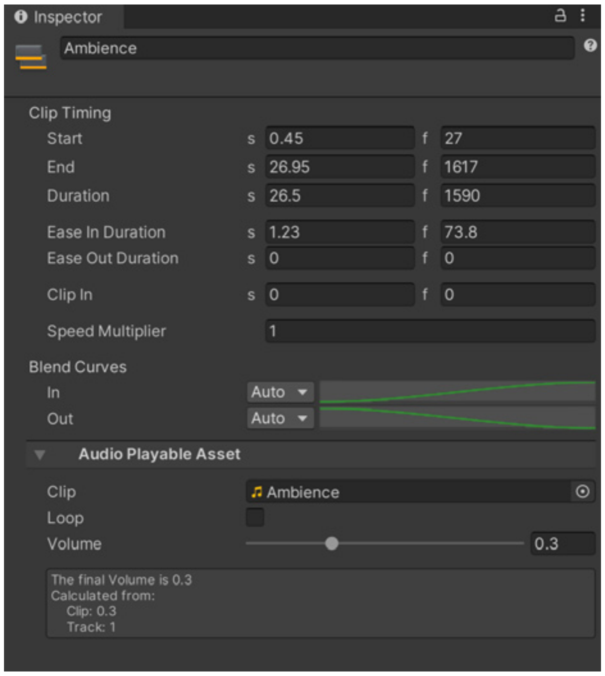
Audio track

Use the audio track to place sound files and music at set points in the animation. You can use multiple audio tracks to mix together a complex audio score.



Use multiple audio tracks to make a rich sound track for a sequence.

If the audio source slot is left empty the sounds and music will play in 2D space. If you place an object with an Audio Source component into this slot you can set 3D sounds to play from specific parts of the 3D world, creating surround sound audio scores.



Inspector

Ambience

Clip Timing

	s		f	
Start	s	0.45	f	27
End	s	26.95	f	1617
Duration	s	26.5	f	1590
Ease In Duration	s	1.23	f	73.8
Ease Out Duration	s	0	f	0
Clip In	s	0	f	0

Speed Multiplier: 1

Blend Curves

In: Auto

Out: Auto

Audio Playable Asset

Clip: Ambience

Loop

Volume: 0.3

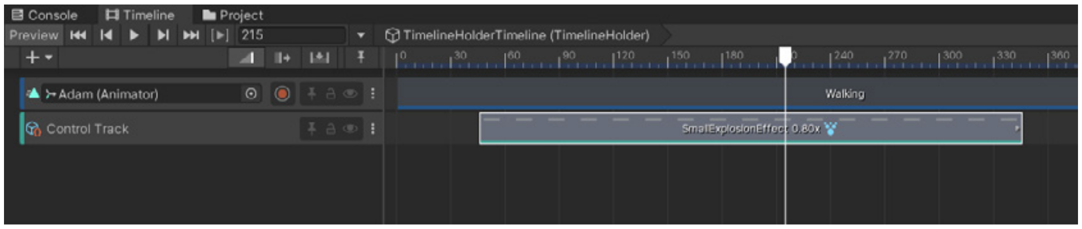
The final Volume is 0.3
Calculated from:
Clip: 0.3
Track: 1

Fading sounds in or out and adjusting the volume

You can fade tracks in or out using the **Ease In Duration** and **Ease Out Duration** settings; a diagonal line appears on the track showing the fade in duration. You can also adjust the volume from the Inspector.

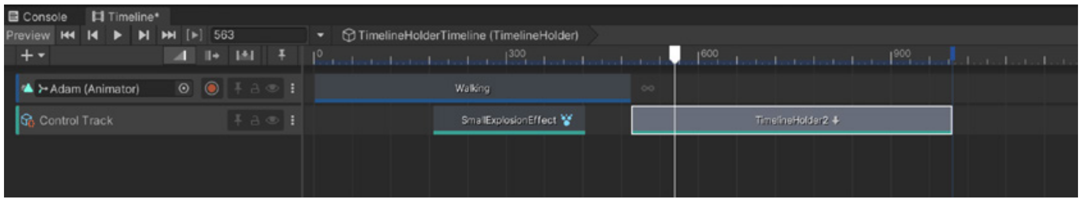
Control track

Use the control track to activate particle effects in your scene, for example, for timing an explosion to happen at a set point in the animation. You can choose what will happen after the particle track ends – it becomes inactive, continues to loop with the active setting, or returns to a dormant state using the Revert setting, and waits to be activated again at a later point.



Controlling particle effects using the control track

Use the control track to orchestrate other timelines from your current timeline. If you have several timeline sequences which you want to begin at set times, you can drag the TimelineHolder GameObjects onto the control track to activate them at the appropriate time.



Controlling multiple timelines using the control track

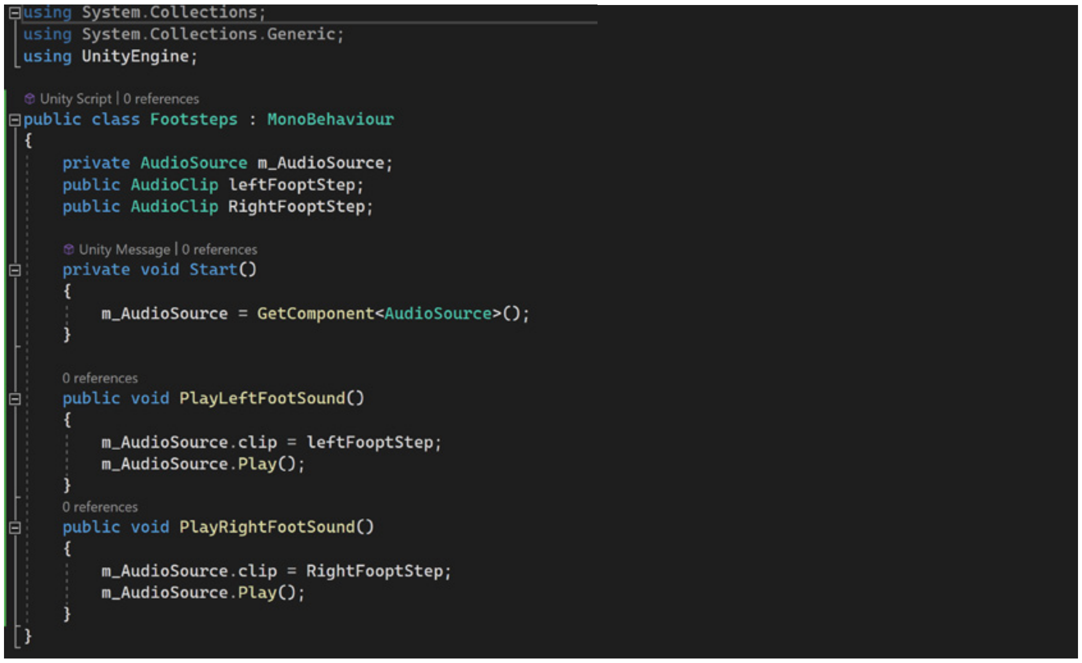
Playables track

The playables track allows you to create your own custom scripts and add them to this track to drive objects and characters during the sequence. A good explanation of this is in this Unity blog post.

Signal track

The signal track works very much like events in the Animation window. You can set a signal event that accesses components on GameObjects and also custom public functions in scripts attached to an object.

To create a footstep sound to match with the character walking, a script can be created with two public functions, one for the left foot and one for the right.

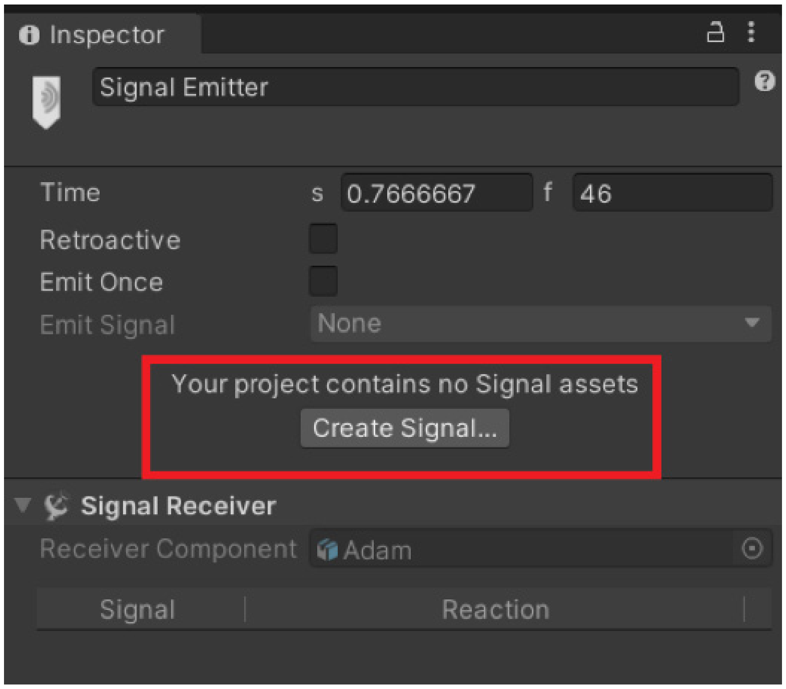


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script | 0 references
public class Footsteps : MonoBehaviour
{
    private AudioSource m_AudioSource;
    public AudioClip leftFooptStep;
    public AudioClip RightFooptStep;

    Unity Message | 0 references
    private void Start()
    {
        m_AudioSource = GetComponent<AudioSource>();
    }

    0 references
    public void PlayLeftFootSound()
    {
        m_AudioSource.clip = leftFooptStep;
        m_AudioSource.Play();
    }
    0 references
    public void PlayRightFootSound()
    {
        m_AudioSource.clip = RightFooptStep;
        m_AudioSource.Play();
    }
}
```

Script to play footstep sounds

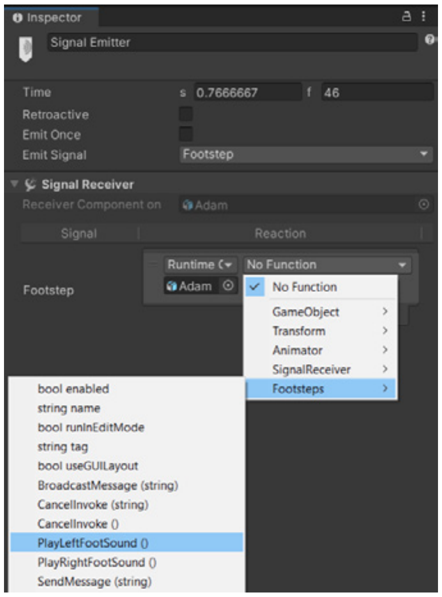
Attach the script to the character along with an Audio Source component to play the sounds.

Now add a signal track in the timeline. Right-click at the point of the first footstep and add a Signal Emitter. It will have a yellow warning, as a Signal Asset does not yet exist. You can create this in the Inspector.



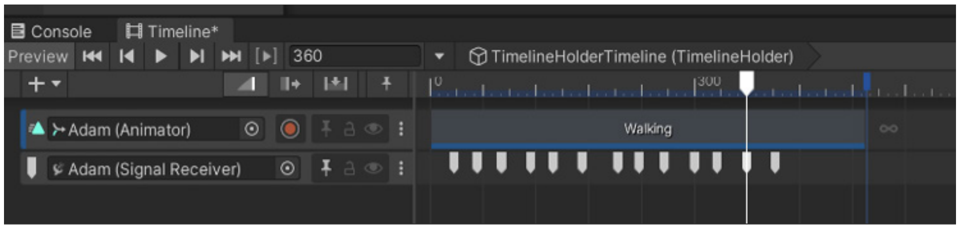
Inspector
Signal Emitter
Time s 0.7666667 f 46
Retroactive
Emit Once
Emit Signal None
Your project contains no Signal assets
Create Signal...
Signal Receiver
Receiver Component Adam
Signal | Reaction

Creating a new signal asset



Inspector
Signal Emitter
Time s 0.7666667 f 46
Retroactive
Emit Once
Emit Signal Footstep
Signal Receiver
Receiver Component on Adam
Signal | Reaction
Footstep | Runtime O | No Function | Adam

From the drop-down you have access to any component attached to the character and you can do multiple tasks. Choosing the script you can find the PlayLeftFootSound and PlayRightFootSound functions and add multiple signals for the left and right footstep sounds.



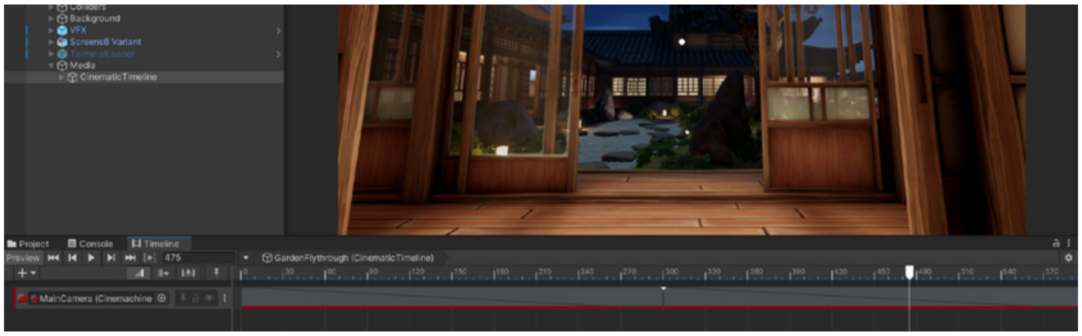
Choosing custom functions from a Signal Asset

Cinemachine tracks

Cinemachine creates complex cinematic camera setups in an efficient way. The Cinemachine track allows you to control the virtual cameras used in Cinemachine to time and adjust your shots.

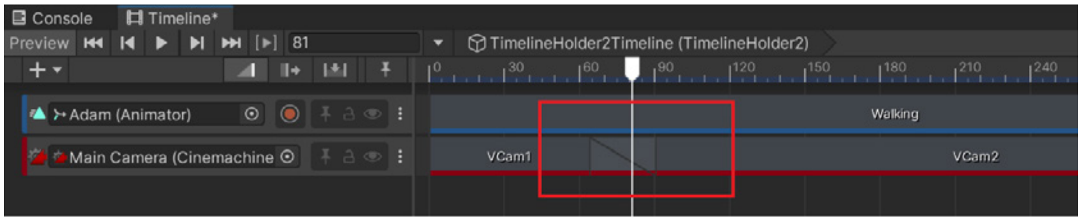
Drag your main camera into the empty slot and it will create a Cinemachine Brain component on that main camera. This is used so the main camera will conform to the virtual camera settings.

After creating a few virtual cameras and framing them they are added to this track. If one camera follows the next you get a straight cut between the two.



Cinemachine straight cut between two camera shots

If one camera block is dragged into another you get a crossover blend so the camera will move between the first shot and the next.



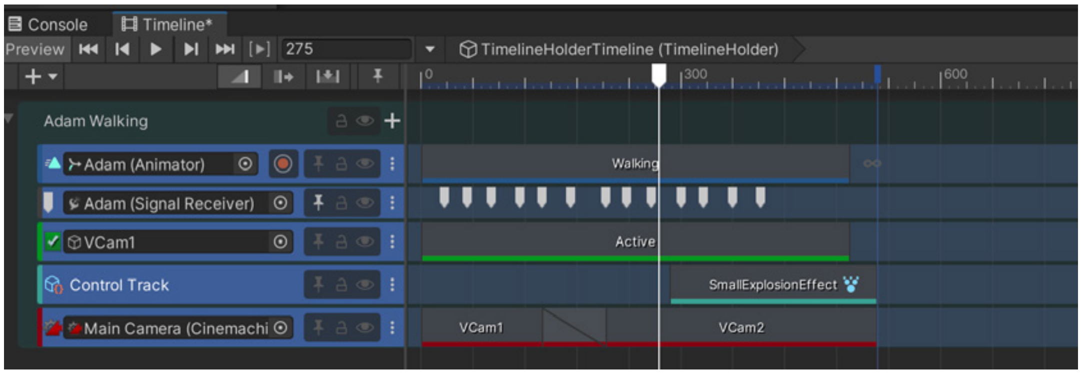
Cinemachine blend between two camera shots

Note that Cinemachine version 2 is used in the screenshots above. For more information on the new features in Cinemachine version 3 see this blog article.

Track group

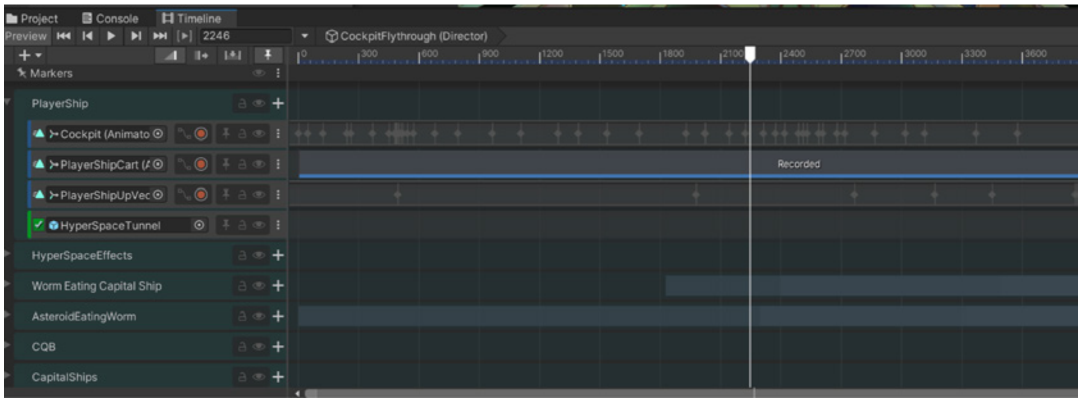
As you add more tracks you could find it difficult to navigate your timeline.

Track groups allow you to order certain tracks together and keep them well organized.



Tracks collected together into a track group

You can then minimize the track groups to create more space on the timeline, making it easier to navigate.



Minimized track groups

Sequences

You can use the Sequences package to make large projects easier to handle when working in Timeline. It allows you to dynamically load scenes at set points during playback, making scenes easier to work with.

See this Unity Learn tutorial for a step by step introduction to working with the Sequences package.

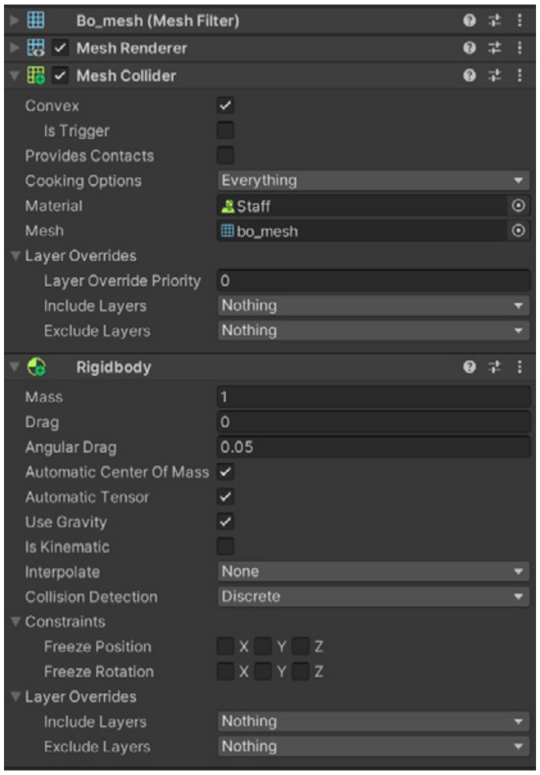
Advanced simulation

Unity's Built-In 3D Physics engine simulates physics events in real-time. Use it to simulate objects reacting to gravity, and realistic movement for objects, like hair, fur, cloth, capes, curtains, and flags.

Rigidbody Physics

To add solid physics, add a Rigidbody component to the object, as well as a Mesh Collider component, so it doesn't fall through the ground.

Solid physics refers to the realistic simulation of physical interactions between objects in a virtual environment. Properties like mass, gravity, friction, and collisions are simulated to show objects behaving in a way that mimics the real world.



Bo_mesh (Mesh Filter)
✓ Mesh Renderer
✓ Mesh Collider
Convex ✓
Is Trigger ☐
Provides Contacts ☐
Cooking Options: Everything
Material: Staff
Mesh: bo_mesh
Layer Overrides
Layer Override Priority: 0
Include Layers: Nothing
Exclude Layers: Nothing
Rigidbody
Mass: 1
Drag: 0
Angular Drag: 0.05
Automatic Center Of Mass ✓
Automatic Tensor ✓
Use Gravity ✓
Is Kinematic ☐
Interpolate: None
Collision Detection: Discrete
Constraints
Freeze Position ☐ X ☐ Y ☐ Z
Freeze Rotation ☐ X ☐ Y ☐ Z
Layer Overrides
Include Layers: Nothing
Exclude Layers: Nothing

A staff GameObject with a Rigidbody component attached to create physics

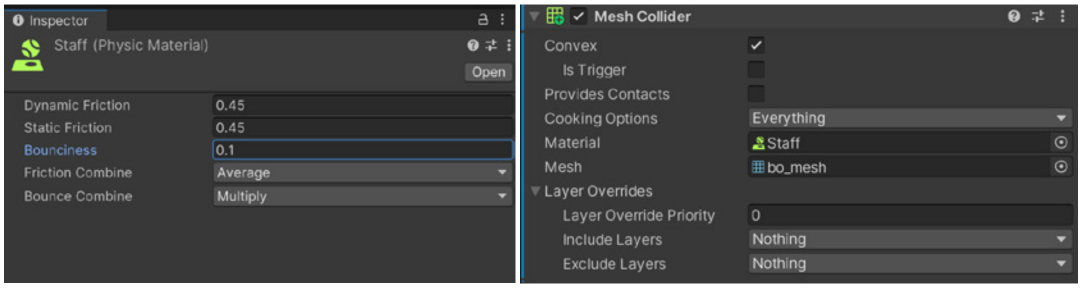
You can configure the Rigidbody component with a set amount of mass and drag to simulate how real-world objects interact with the air as they fall. You can include or exclude certain layers, ensuring the object only collides with other objects on set layers. You can also use the **Freeze Position** and **Freeze Rotation** settings to get more control over the simulation.

Leave the **IsKinematic** property unchecked to ensure the object uses gravity. If IsKinematic is checked then the object will not use gravity but can still process physics-based interactions like collisions.

Colliders are physical colliders by default. This will prevent objects from passing through other objects. However, you can check the **Is Trigger** option to set the collider as a trigger collider which disables the physical aspect, allowing the object to pass through other objects, but still return a trigger when it intersects another object. This can be useful for coding.

You can add a Physic Material asset to a Mesh Collider to ensure the object acts in a way that simulates a real-world object. For a bouncy rubber ball, for example, you need to add a bouncy Physic Material into the **Material** field of the Collider component.

To create a new Physic Material right-click in the Project window, choose **Create > Physic Material**, and adjust the values to get the desired result.


Inspector
Staff (Physic Material)
Open

Dynamic Friction	0.45
Static Friction	0.45
Bounciness	0.1
Friction Combine	Average
Bounce Combine	Multiply

Mesh Collider

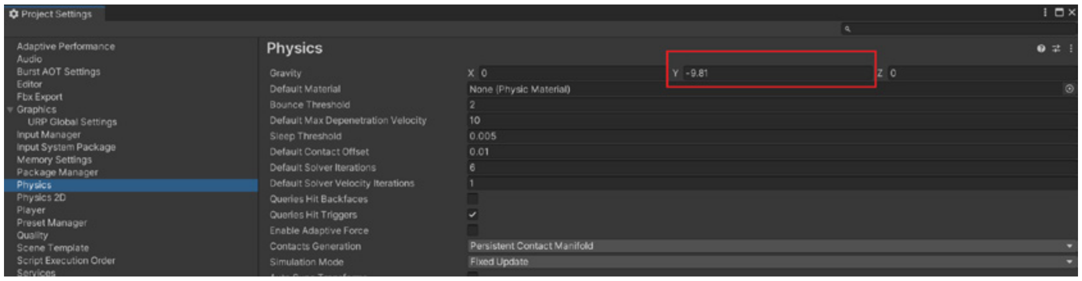
Convex	✓
Is Trigger	
Provides Contacts	
Cooking Options	Everything
Material	Staff
Mesh	bo_mesh
Layer Overrides	
Layer Override Priority	0
Include Layers	Nothing
Exclude Layers	Nothing

Defining a Physic Material for a wooden staff and assigning it to the Mesh Collider

Material	Dinamic friction	Static friction	Bounciness	Friction combine	Bounce combine
Boncy	0.3	0.3	1	Average	Maximum
Ice	0.1	0.1	0	Multiply	Multiply
Max friction	1	1	0	Maximum	Average
Metal	0.15	0.15	0	Minimum	Average
Rubber	1	1	0.5	Maximum	Average
Wood	0.45	0.45	0	Average	Average
Zero friction	0	0	0	Multiply	Average

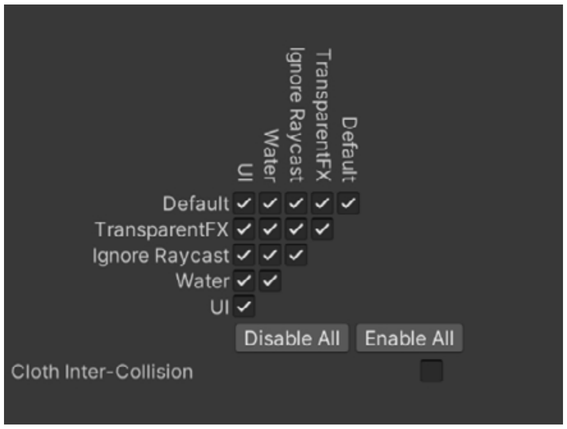
Useful physics presets for different type of objects

Physics settings can be adjusted by going to **Edit > Project Settings**. From here you can change the gravity amount. By default real-world gravity is set at -9.81, but you can alter this, for example, to simulate a character floating in space.



Physics

Gravity	X 0 Y -9.81 Z 0
Default Material	None (Physic Material)
Bounce Threshold	2
Default Max Depenetration Velocity	10
Sleep Threshold	0.005
Default Contact Offset	0.01
Default Solver Iterations	6
Default Solver Velocity Iterations	1
Queries Hit Backfaces	
Queries Hit Triggers	✓
Enable Adaptive Force	
Contacts Generation	Persistent Contact Manifold
Simulation Mode	Fixed Update



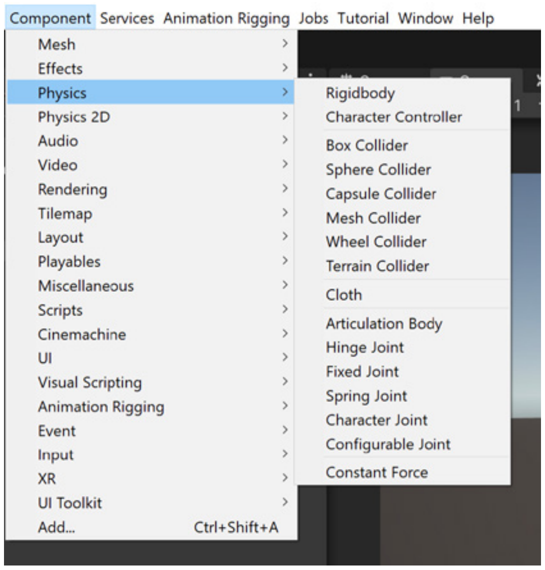
	UI	Water	Ignore Raycast	TransparentFX	Default
Default	✓	✓	✓	✓	✓
TransparentFX	✓	✓	✓	✓	
Ignore Raycast	✓	✓	✓		
Water	✓	✓			
UI	✓				

Cloth Inter-Collision

Physics settings are accessed via the Project Settings window.

Use the layer collision matrix at the bottom to define which layers will be able to interact with each other.

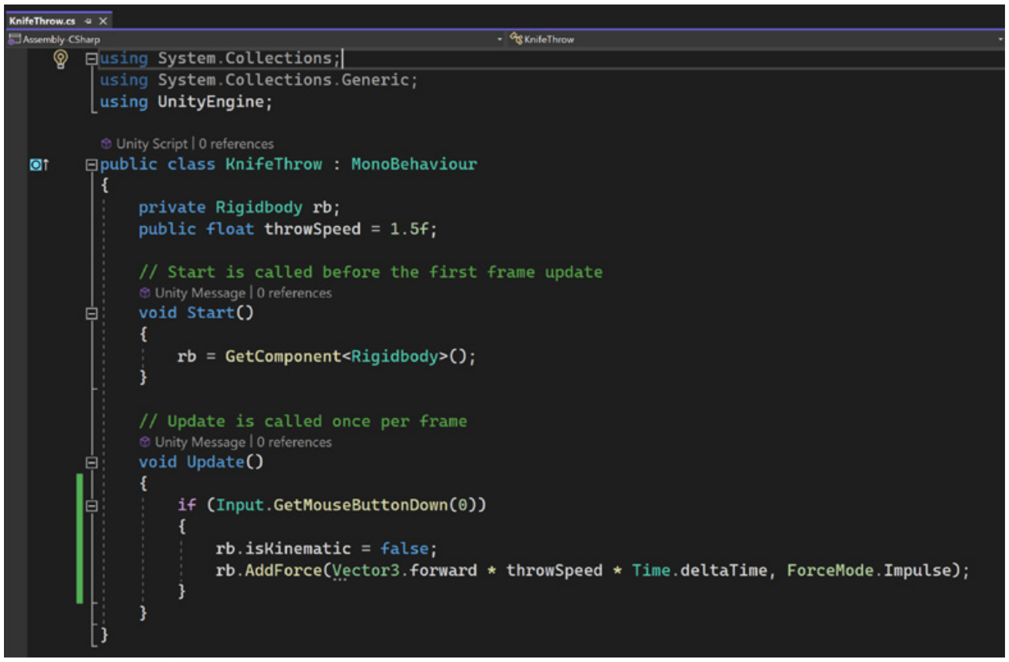
Various physics colliders, joints, and forces can be added from **Component > Physics**.



Physics components

The constant force is useful when you want to apply force in a specific direction to a physical object, such as to simulate wind or objects being thrown.

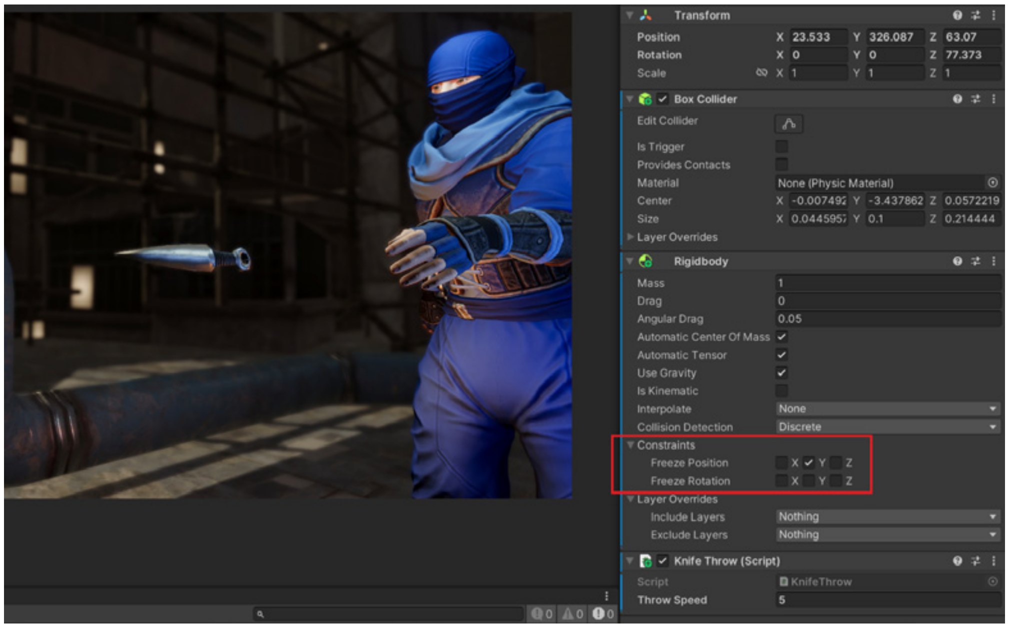
You can also add force to a Rigidbody component using code, for example, to add force on a mouse click to drive a knife forward at a set speed.


KnifeThrow.cs
Assembly-CSharp
KnifeThrow
Unity Script | 0 references
Unity Message | 0 references
Unity Message | 0 references

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class KnifeThrow : MonoBehaviour
{
    private Rigidbody rb;
    public float throwSpeed = 1.5f;

    // Start is called before the first frame update
    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            rb.isKinematic = false;
            rb.AddForce(Vector3.forward * throwSpeed * Time.deltaTime, ForceMode.Impulse);
        }
    }
}
```

Code for adding force to an object when a player left-clicks the mouse

The Rigidbody component is frozen on the Y movement axis ensuring the knife doesn't drop to the ground, but travels forward at the force speed set in the code.



Position	X 23.533	Y 326.087	Z 63.07
Rotation	X 0	Y 0	Z 77.373
Scale	X 1	Y 1	Z 1

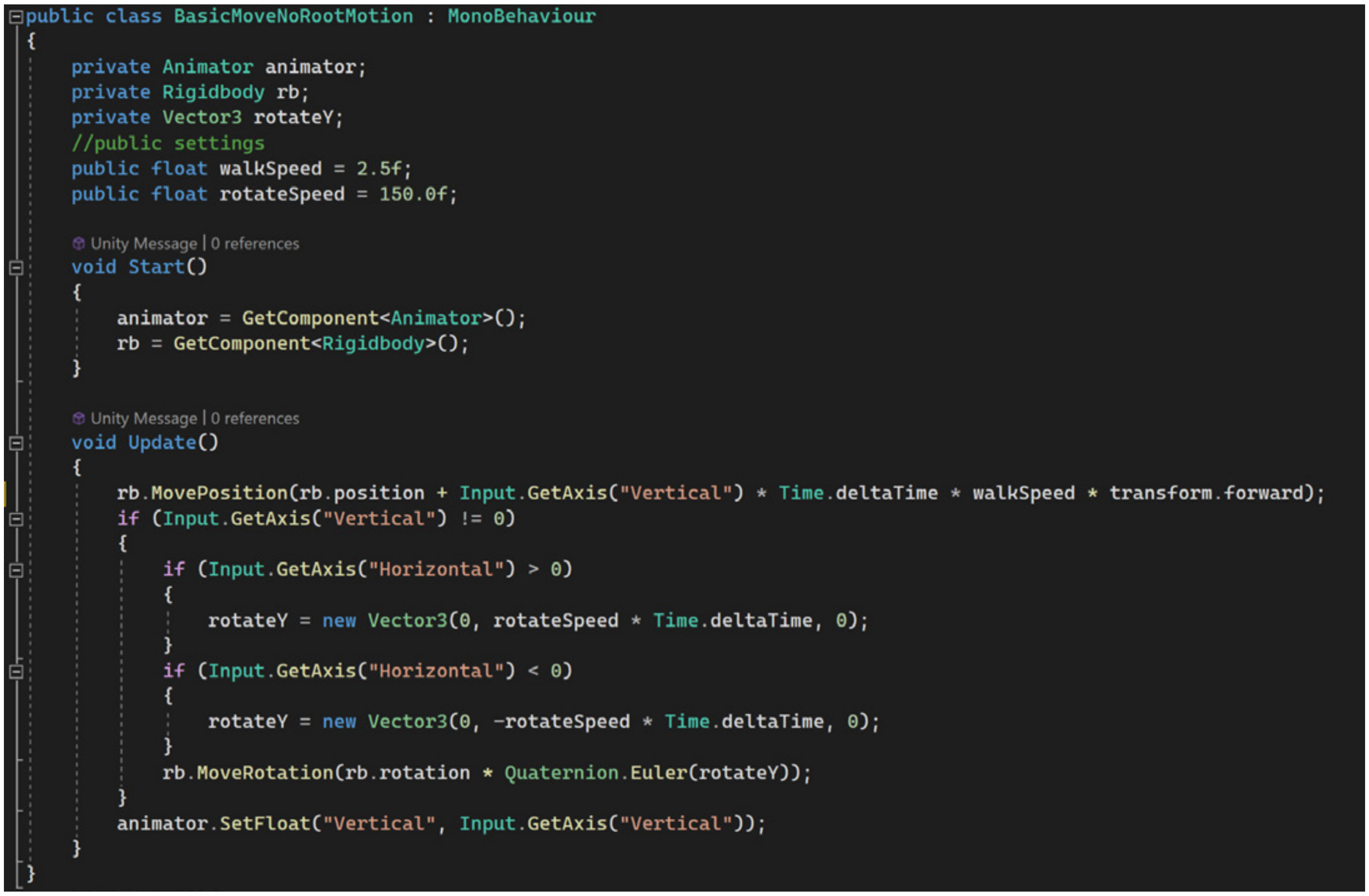
Is Trigger	
Provides Contacts	
Material	None (Physic Material)
Center	X -0.007492 Y -3.437862 Z 0.0572219
Size	X 0.0445957 Y 0.1 Z 0.214444
Layer Overrides	

Mass	1
Drag	0
Angular Drag	0.05
Automatic Center Of Mass	✓
Automatic Tensor	✓
Use Gravity	✓
Is Kinematic	
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	X [✓ Y] Z
Freeze Rotation	X Y Z
Layer Overrides	
Include Layers	Nothing
Exclude Layers	Nothing
Script	KnifeThrow
Throw Speed	5

Throwing a knife and constraining the Y axis position

You can use the Rigidbody component to control the movement of a character. This is useful for creating realistic motion that reacts to physical objects in the game world.

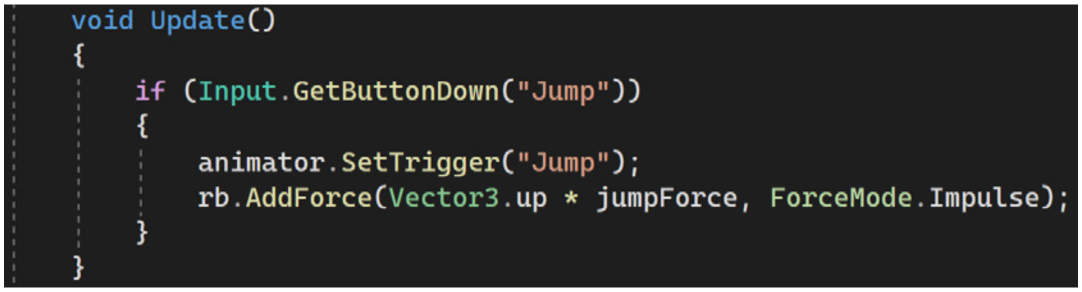
To do this, access the Rigidbody component in code and use the MovePosition or MoveRotation commands for it.



```
public class BasicMoveNoRootMotion : MonoBehaviour
{
    private Animator animator;
    private Rigidbody rb;
    private Vector3 rotateY;
    //public settings
    public float walkSpeed = 2.5f;
    public float rotateSpeed = 150.0f;

    Unity Message | 0 references
    void Start()
    {
        animator = GetComponent<Animator>();
        rb = GetComponent<Rigidbody>();
    }

    Unity Message | 0 references
    void Update()
    {
        rb.MovePosition(rb.position + Input.GetAxis("Vertical") * Time.deltaTime * walkSpeed * transform.forward);
        if (Input.GetAxis("Vertical") != 0)
        {
            if (Input.GetAxis("Horizontal") > 0)
            {
                rotateY = new Vector3(0, rotateSpeed * Time.deltaTime, 0);
            }
            if (Input.GetAxis("Horizontal") < 0)
            {
                rotateY = new Vector3(0, -rotateSpeed * Time.deltaTime, 0);
            }
            rb.MoveRotation(rb.rotation * Quaternion.Euler(rotateY));
        }
        animator.SetFloat("Vertical", Input.GetAxis("Vertical"));
    }
}
```

Controlling character movement using rigidbody movement

To perform a jump, add force to the Rigidbody.



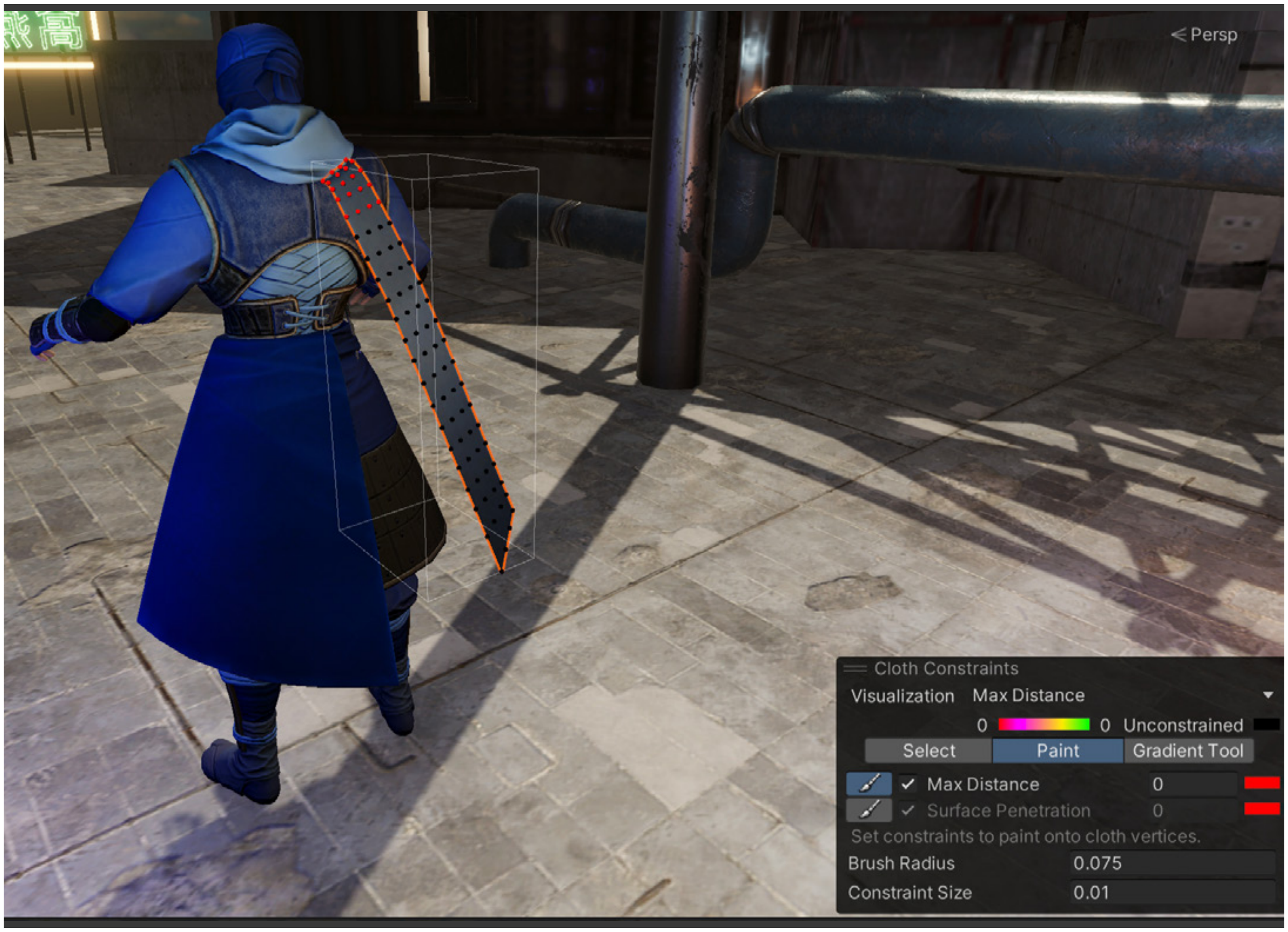
```
void Update()
{
    if (Input.GetButtonDown("Jump"))
    {
        animator.SetTrigger("Jump");
        rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
    }
}
```

Add force to the Rigidbody component to perform a jump

Cloth Physics

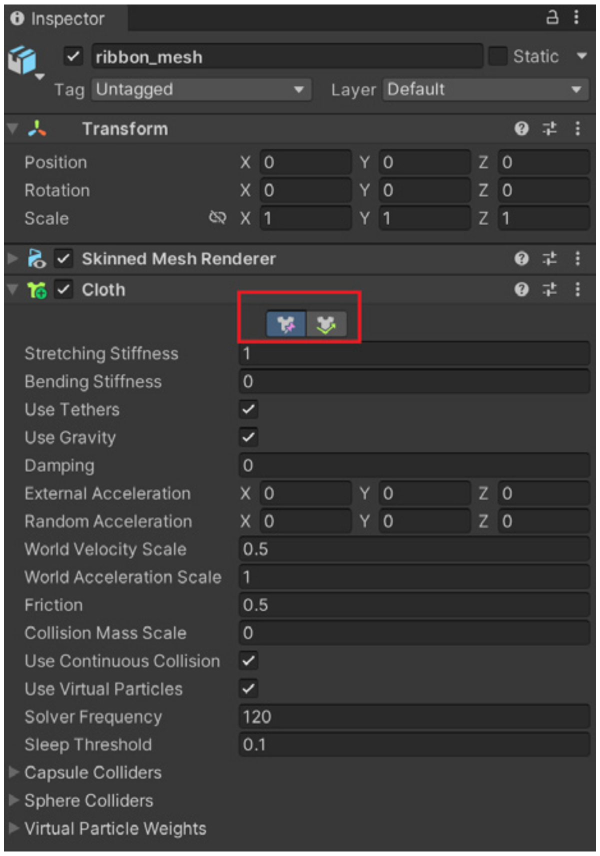
Cloth physics simulate the movement of cloth and can be applied to any mesh that has a good level of subdivisions. A mesh needs to have enough vertices to make the simulated material move in a believable way, but not too many that it increases the performance cost.

Add a Cloth component to an object and then use the settings to define the material type. Adjust properties like **Stretching Stiffness** and **Bending Stiffness** to simulate leather, silk, cotton, etc.



Visualization Max Distance
0 0 Unconstrained
Max Distance 0
Surface Penetration 0
Set constraints to paint onto cloth vertices.
Brush Radius 0.075
Constraint Size 0.01

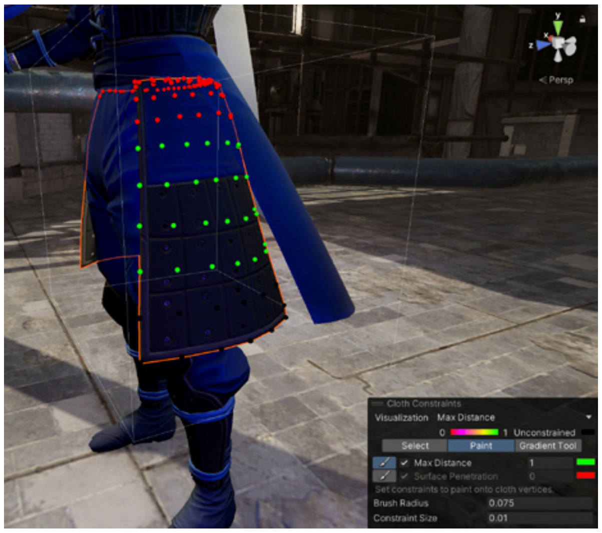
Setting constraints to a cloth object



```
Inspector
ribbon_mesh                                   Static
Tag  Untagged                 Layer  Default
Transform
Position        X 0      Y 0      Z 0
Rotation        X 0      Y 0      Z 0
Scale           X 1      Y 1      Z 1
Skinned Mesh Renderer
Cloth
Stretching Stiffness       1
Bending Stiffness          0
Use Tethers                ✓
Use Gravity                ✓
Damping                    0
External Acceleration      X 0      Y 0      Z 0
Random Acceleration        X 0      Y 0      Z 0
World Velocity Scale       0.5
World Acceleration Scale   1
Friction                   0.5
Collision Mass Scale       0
Use Continuous Collision   ✓
Use Virtual Particles      ✓
Solver Frequency           120
Sleep Threshold            0.1
Capsule Colliders
Sphere Colliders
Virtual Particle Weights
```

Cloth properties and the cloth constraints button

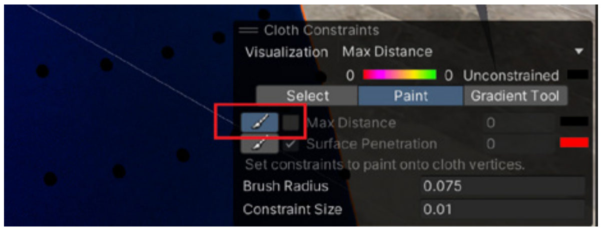
The cloth must be constrained at an origin point otherwise it will fall to the ground. Clicking on the t-shirt icon with the red pin (see the image on the left) will open the Cloth Constraints menu. Set **Max Distance** to 0 and paint red dots on the cloth at the points where you don't want it to move. Black points are unconstrained and will flow about during cloth simulation.



```
Cloth Constraints
Visualization  Max Distance
0  1  Unconstrained
Select  Paint  Gradient Tool
✓ Max Distance          1   🟩
✓ Surface Penetration   0   🟥
Set constraints to paint onto cloth vertices.
Brush Radius       0.075
Constraint Size    0.01
```

Defining a specific distance for cloth movement between points

You can also set a specific number for the maximum distance that the vertex points can move away from each other. This is useful for defining heavy materials like the leather uniform on the ninja character seen in the screenshots. Colors between orange to green define the specific distance for the points moving away from each other, giving the cloth different types of movement, and are painted using the gradient tool.

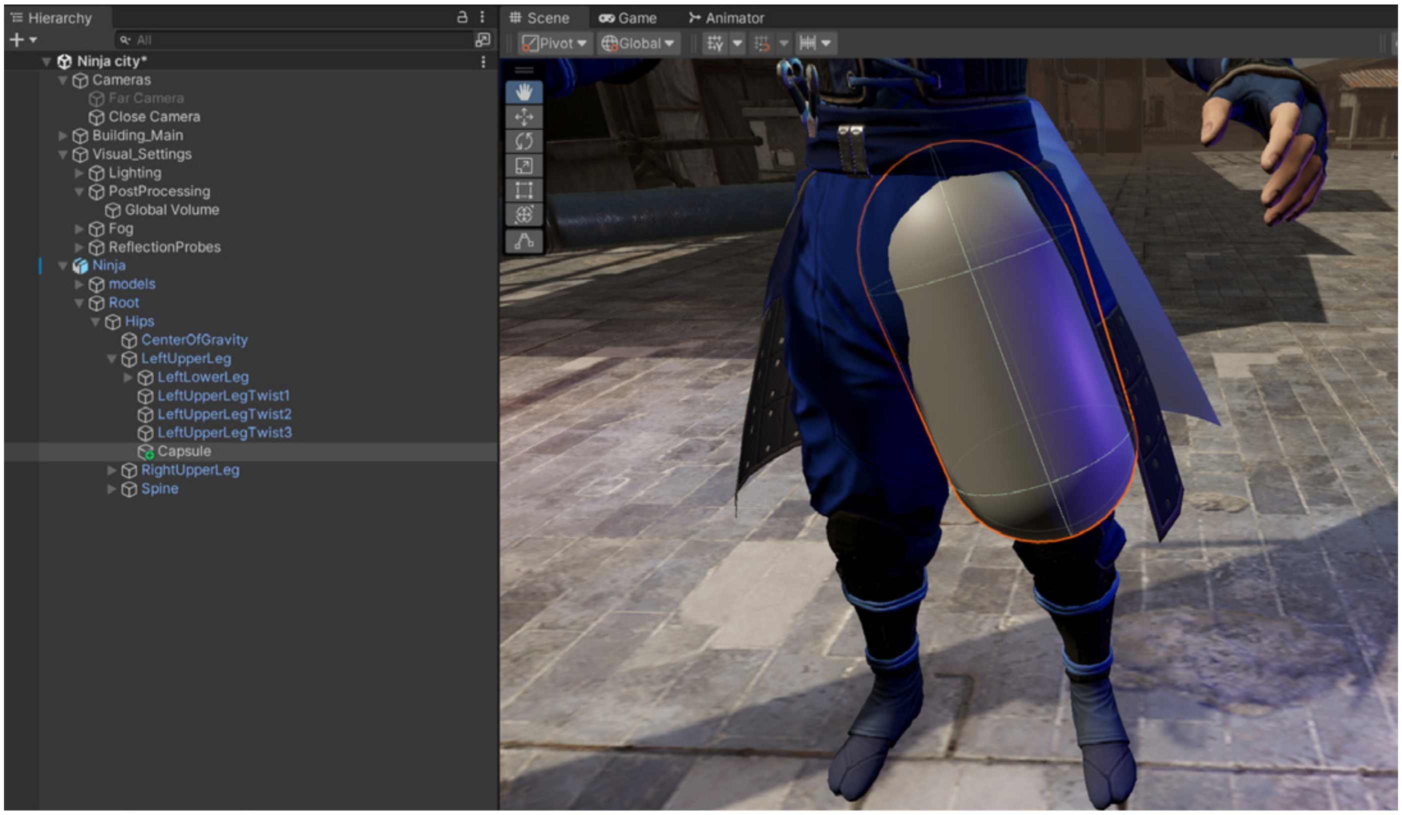


Returning points to unconstrained

If you make a mistake uncheck the Max Distance box to paint over the points, which will return them to black unconstrained points.

The Cloth object will intersect with the rest of the character as it simulates, so you'll need to place colliders to prevent the cloth from moving through the character. Cloth supports Capsule and Sphere colliders only.

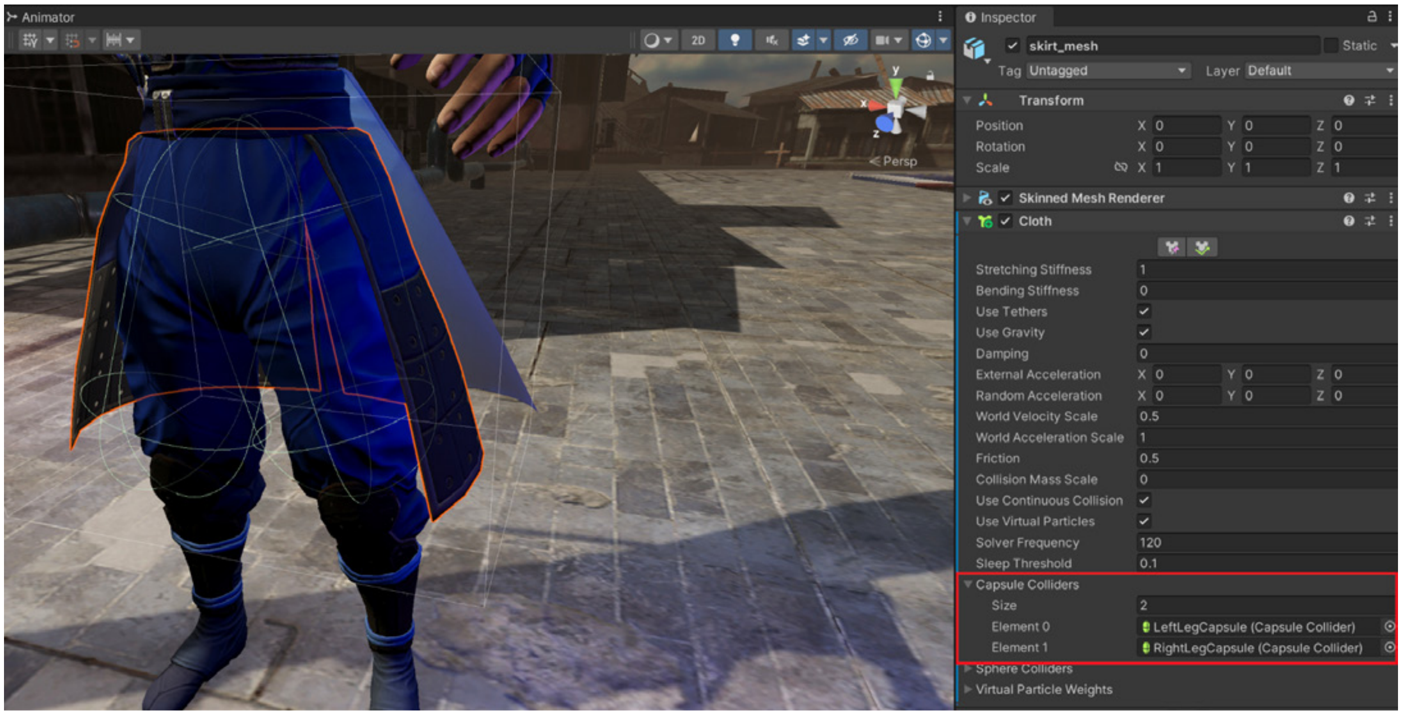
You can build up complex shapes by using multiple capsules and spheres to define an exact shape and then add them to the array of capsule and sphere colliders in the Cloth component.



Adding colliders to the bones

Add either a capsule or a sphere to the correct bone in the Hierarchy and position it. The colliders stay on the bones, helping to provide realistic, correct movement for the character. Once the collider's in place, remove the Renderer component on it by right-clicking and choosing **Remove Component**.

Once both legs have colliders, add them to the Cloth component for each object.



skirt_mesh

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Stretching Stiffness	1
Bending Stiffness	0
Use Tethers	✓
Use Gravity	✓
Damping	0
External Acceleration	X 0 Y 0 Z 0
Random Acceleration	X 0 Y 0 Z 0
World Velocity Scale	0.5
World Acceleration Scale	1
Friction	0.5
Collision Mass Scale	0
Use Continuous Collision	✓
Use Virtual Particles	✓
Solver Frequency	120
Sleep Threshold	0.1
Capsule Colliders	
Size	2
Element 0	LeftLegCapsule (Capsule Collider)
Element 1	RightLegCapsule (Capsule Collider)
Sphere Colliders	
Virtual Particle Weights	

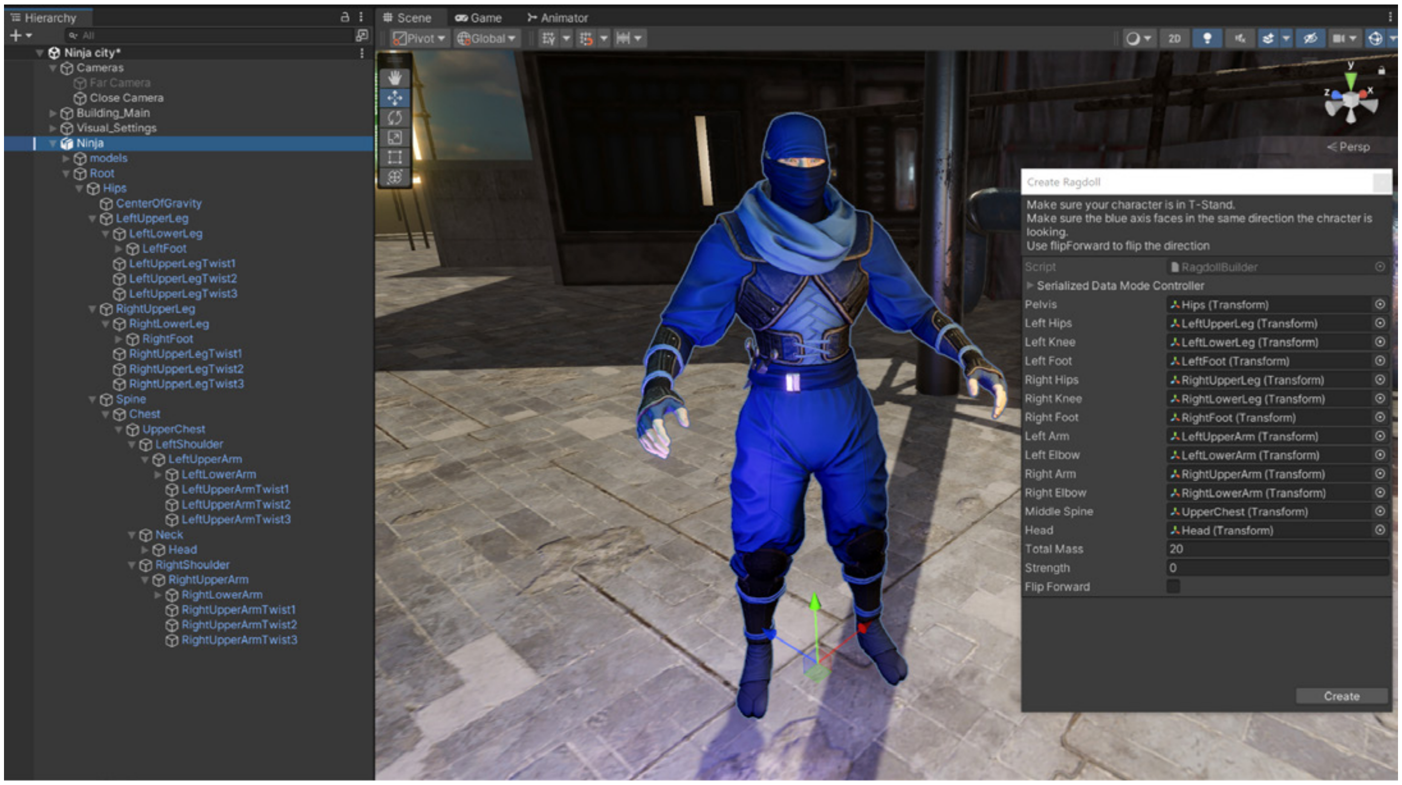
Add the capsules to the Skinned Mesh Renderer component

Realistic simulation of cloth physics

Ragdoll physics

Ragdoll physics simulates the flailing of a character’s legs and arms when it falls or collapses to the ground. Unity has a quick setup system to add the physics components to a character’s skeleton.

Select the character and go to **GameObject > 3D Object > Ragdoll**. This opens the setup window. The character should be in either a T or A pose. Drag the bones into the correct slots and click **Create**.

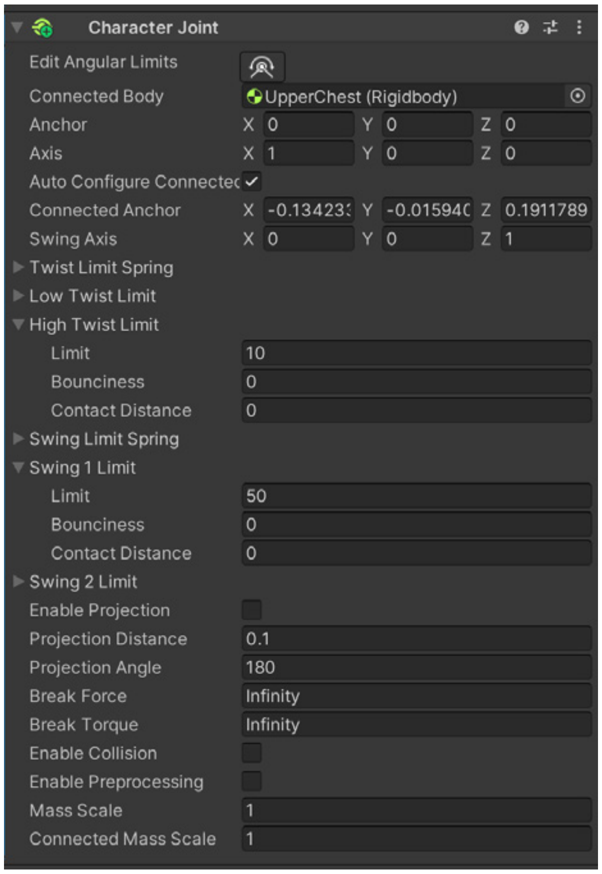


Make sure your character is in T-Stand.
Make sure the blue axis faces in the same direction the chracter is looking.
Use flipForward to flip the direction

Script	RagdollBuilder
▸ Serialized Data Mode Controller	
Pelvis	Hips (Transform)
Left Hips	LeftUpperLeg (Transform)
Left Knee	LeftLowerLeg (Transform)
Left Foot	LeftFoot (Transform)
Right Hips	RightUpperLeg (Transform)
Right Knee	RightLowerLeg (Transform)
Right Foot	RightFoot (Transform)
Left Arm	LeftUpperArm (Transform)
Left Elbow	LeftLowerArm (Transform)
Right Arm	RightUpperArm (Transform)
Right Elbow	RightLowerArm (Transform)
Middle Spine	UpperChest (Transform)
Head	Head (Transform)
Total Mass	20
Strength	0
Flip Forward	☐

Ragdoll setup window

This now adds a Rigidbody component to each bone in the list and a Character Joint component that simulates the ragdoll-style physics.



Character Joint

Edit Angular Limits	
Connected Body	UpperChest (Rigidbody)
Anchor	X 0 Y 0 Z 0
Axis	X 1 Y 0 Z 0
Auto Configure Connectec	✓
Connected Anchor	X -0.13423: Y -0.01594(Z 0.1911789
Swing Axis	X 0 Y 0 Z 1
Twist Limit Spring	
Low Twist Limit	
High Twist Limit	
Limit	10
Bounciness	0
Contact Distance	0
Swing Limit Spring	
Swing 1 Limit	
Limit	50
Bounciness	0
Contact Distance	0
Swing 2 Limit	
Enable Projection	
Projection Distance	0.1
Projection Angle	180
Break Force	Infinity
Break Torque	Infinity
Enable Collision	
Enable Preprocessing	
Mass Scale	1
Connected Mass Scale	1

The Character Joint component that handles the ragdoll physics

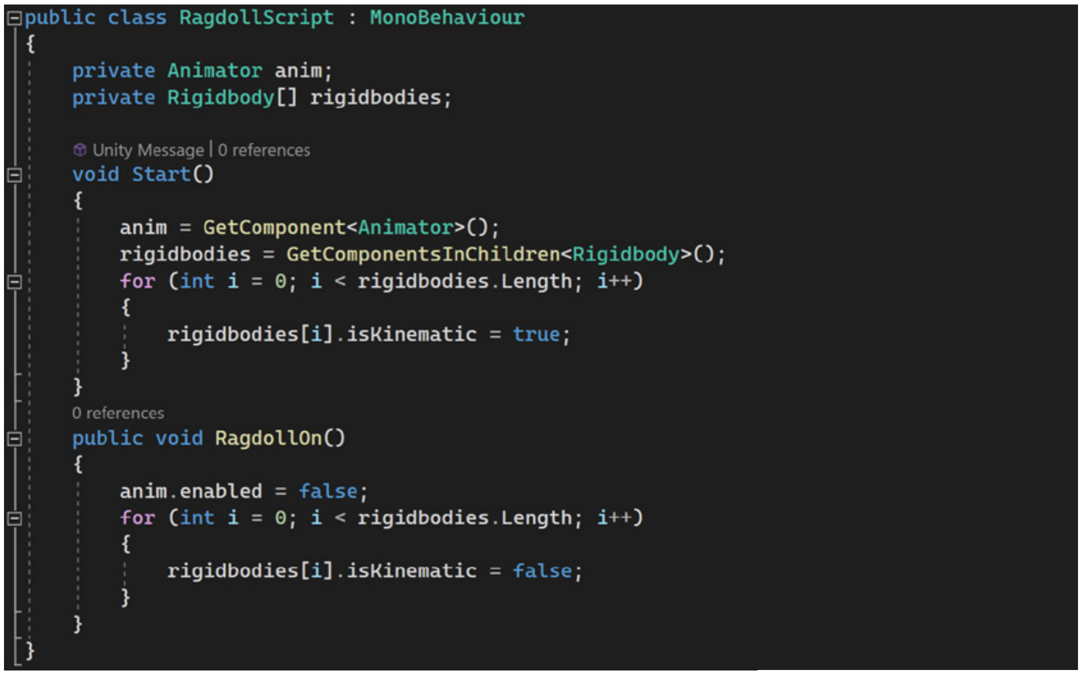
You can modify the Character Joint component to get your desired ragdoll effects. The value you set for the **Mass** property on the Rigidbody component will also have an effect on the simulation. You can copy and paste the components to other bones in the skeleton that weren't in the list, helping you to create your own custom ragdoll setup.

Any animations that are playing will override the ragdoll simulation, so make sure to deactivate the Animator component to see the ragdoll physics in Play mode.

Ragdoll simulation on the ninja character

You can choose to deactivate the Animator component with a script, for example, to play the ragdoll effect when a character gets shot. However, you will notice weird results when going directly from an animation to ragdoll – the character physics can act in an unpredictable manner.

To fix this issue, set the Rigidbody components on each bone to **isKinematic** to prevent them being affected by gravity until the Animator is switched off. Then, set isKinematic back to false to see the ragdoll effect play without any issues.



```
public class RagdollScript : MonoBehaviour
{
    private Animator anim;
    private Rigidbody[] rigidbodies;

    Unity Message | 0 references
    void Start()
    {
        anim = GetComponent<Animator>();
        rigidbodies = GetComponentsInChildren<Rigidbody>();
        for (int i = 0; i < rigidbodies.Length; i++)
        {
            rigidbodies[i].isKinematic = true;
        }
    }
    0 references
    public void RagdollOn()
    {
        anim.enabled = false;
        for (int i = 0; i < rigidbodies.Length; i++)
        {
            rigidbodies[i].isKinematic = false;
        }
    }
}
```

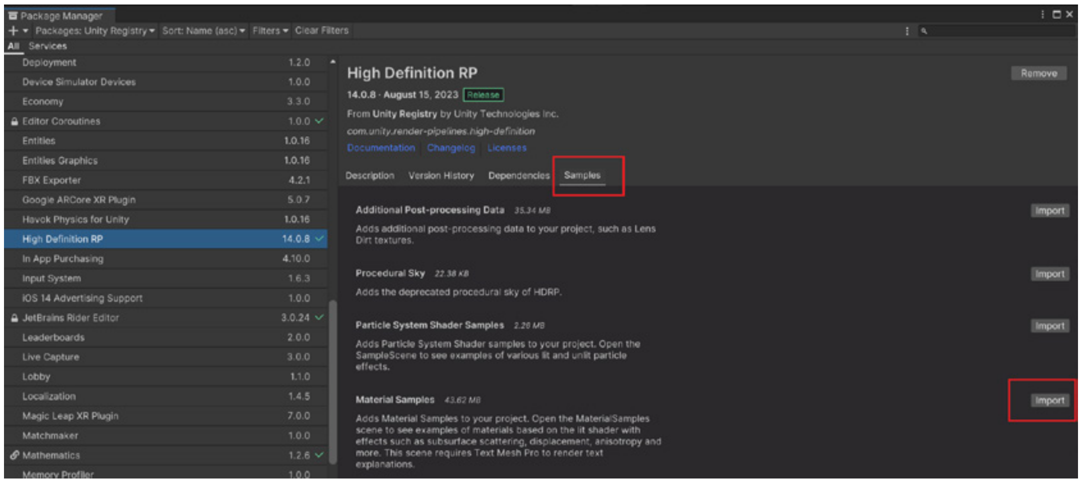
Switching Rigidbodies to isKinematic when switching off the Animator component; this is placed on the parent GameObject

Fur and hair

Unity's High Definition Render Pipeline (HDRP) features a sophisticated Shader Graph-based hair shader for creating realistic hair and fur. You can also create your own hair effects that use line-based renderers; see this in action in Unity's Lion demo.

This Lion demo shows real-time content created with Wētā Digital, SpeedTree, Ziva, SyncSketch, and artist tools in the Unity Editor

You can take a look at the sample scenes for the hair shader by installing the Materials Samples folder from the Package Manager under the High Definition RP section.



Deployment	1.2.0
Device Simulator Devices	1.0.0
Economy	3.3.0
Editor Coroutines	1.0.0 ✓
Entities	1.0.16
Entities Graphics	1.0.16
FBX Exporter	4.2.1
Google ARCore XR Plugin	5.0.7
Havok Physics for Unity	1.0.16
High Definition RP	14.0.8 ✓
In App Purchasing	4.10.0
Input System	1.6.3
iOS 14 Advertising Support	1.0.0
JetBrains Rider Editor	3.0.24 ✓
Leaderboards	2.0.0
Live Capture	3.0.0
Lobby	1.1.0
Localization	1.4.5
Magic Leap XR Plugin	7.0.0
Matchmaker	1.0.0
Mathematics	1.2.6 ✓
Memory Profiler	1.0.0

High Definition RP

14.0.8 · August 15, 2023

From Unity Registry by Unity Technologies Inc.

com.unity.render-pipelines.high-definition

Additional Post-processing Data *35.34 MB*
Adds additional post-processing data to your project, such as Lens Dirt textures.

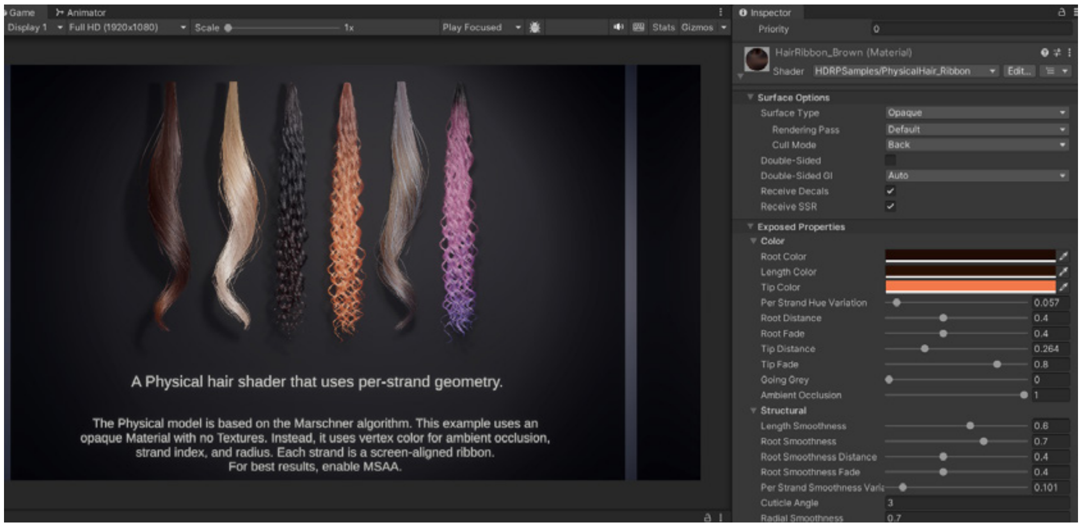
Procedural Sky *22.38 KB*
Adds the deprecated procedural sky of HDRP.

Particle System Shader Samples *2.26 MB*
Adds Particle System Shader samples to your project. Open the SampleScene to see examples of various lit and unlit particle effects.

Material Samples *43.62 MB*
Adds Material Samples to your project. Open the MaterialSamples scene to see examples of materials based on the lit shader with effects such as subsurface scattering, displacement, anisotropy and more. This scene requires Text Mesh Pro to render text explanations.

Material samples contains the hair samples scene

You'll find the hair scene in the Scenes folder. It features strand- and card-based hair rendering, with the material options in the Inspector.



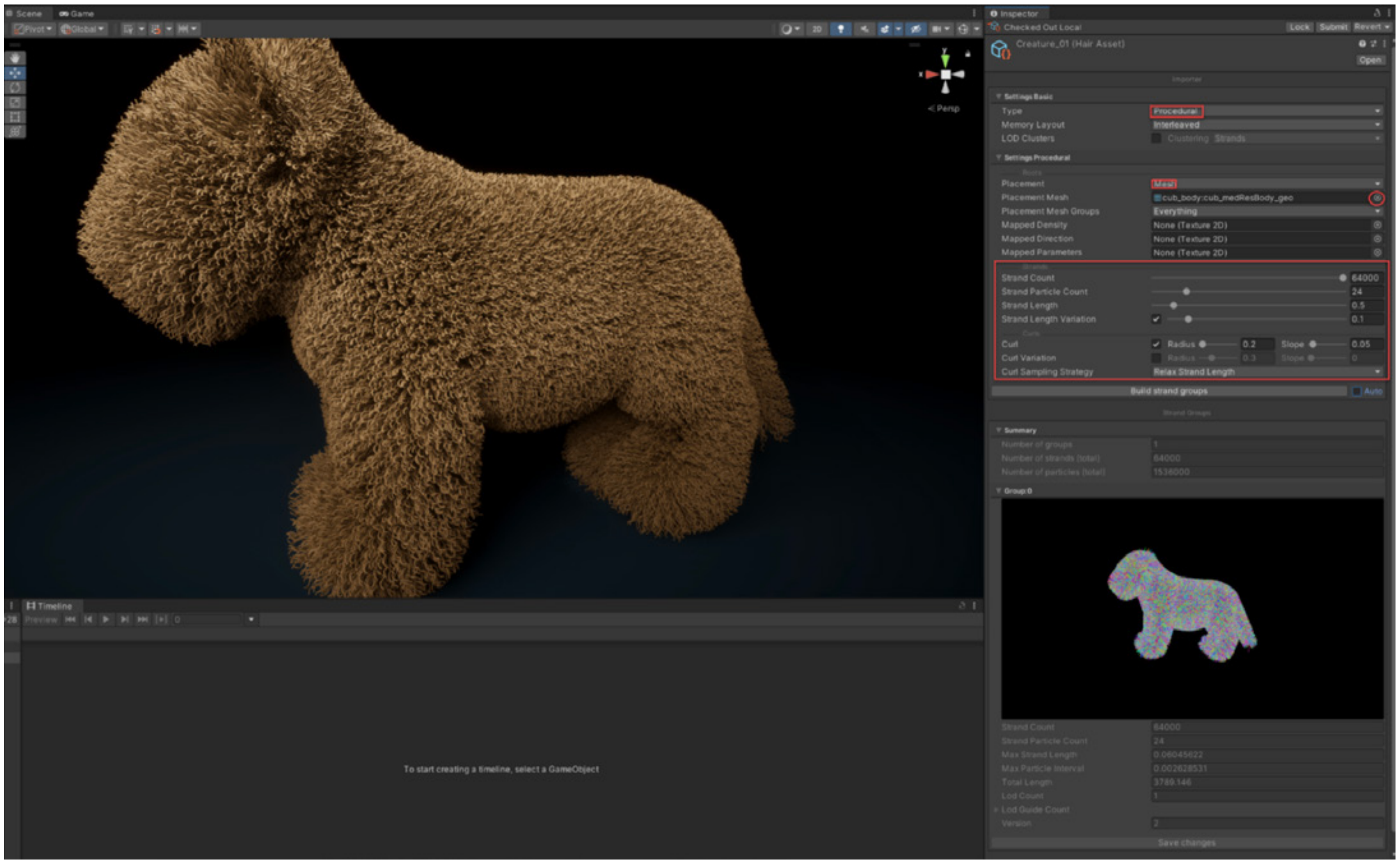
A Physical hair shader that uses per-strand geometry.

The Physical model is based on the Marschner algorithm. This example uses an opaque Material with no Textures. Instead, it uses vertex color for ambient occlusion, strand index, and radius. Each strand is a screen-aligned ribbon.
For best results, enable MSAA.

Priority	0
HairRibbon_Brown (Material)	
Shader	HDRPSamples/PhysicalHair_Ribbon
Surface Options	
Surface Type	Opaque
Rendering Pass	Default
Cull Mode	Back
Double-Sided	
Double-Sided GI	Auto
Receive Decals	✓
Receive SSR	✓
Exposed Properties	
Color	
Root Color	
Length Color	
Tip Color	
Per Strand Hue Variation	0.057
Root Distance	0.4
Root Fade	0.4
Tip Distance	0.264
Tip Fade	0.8
Going Grey	0
Ambient Occlusion	1
Structural	
Length Smoothness	0.6
Root Smoothness	0.7
Root Smoothness Distance	0.4
Root Smoothness Fade	0.4
Per Strand Smoothness Varia	0.101
Cuticle Angle	3
Radial Smoothness	0.7

Realistic hair shader options in HDRP Materials Samples folder

You can also create your own dynamic hair systems that work with gravity and forces by following this tutorial on Unity Learn.



To start creating a timeline, select a GameObject

Checked Out Local

Creature_01 (Hair Asset)

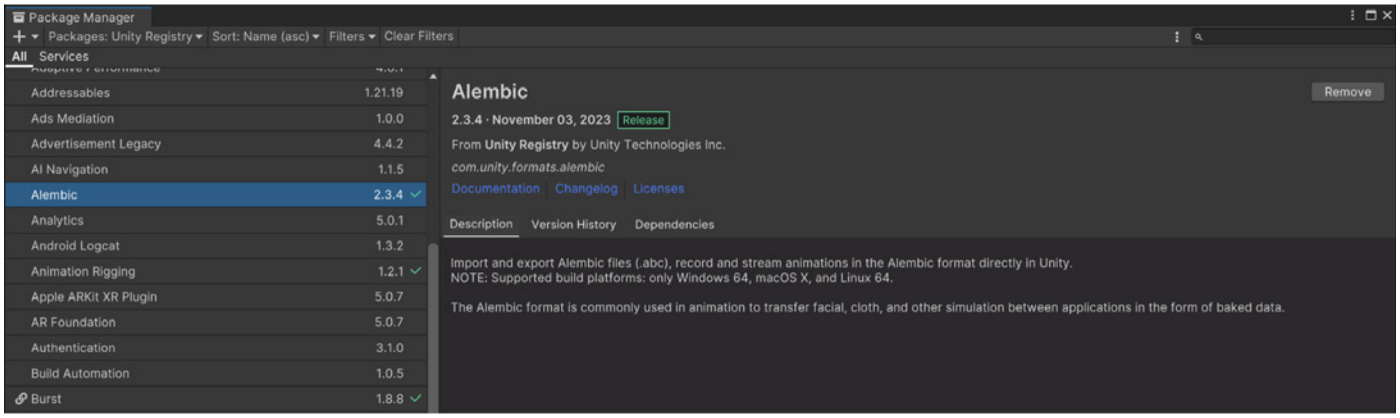
Settings Basic	
Type	Procedural
Memory Layout	Interleaved
LOD Clusters	Clustering Strands
Settings Procedural	
Placement	Mesh
Placement Mesh	cub_body:cub_medResBody_geo
Placement Mesh Groups	Everything
Mapped Density	None (Texture 2D)
Mapped Direction	None (Texture 2D)
Mapped Parameters	None (Texture 2D)
Strand Count	64000
Strand Particle Count	24
Strand Length	0.5
Strand Length Variation	✓ 0.1
Curl	✓ Radius 0.2 Slope 0.05
Curl Variation	Radius 0.3 Slope 0
Curl Sampling Strategy	Relax Strand Length
Summary	
Number of groups	1
Number of strands (total)	64000
Number of particles (total)	1536000
Group:0	
Strand Count	64000
Strand Particle Count	24
Max Strand Length	0.06045622
Max Particle Interval	0.002628531
Total Length	3789.146
Lod Count	1
Lod Guide Count	
Version	2

Learn how to add your own hair simulations to characters and objects with the Unity Learn tutorial "Get started with hair simulation."

Alembic

The Alembic package makes it possible to use alembic files in Unity. Use this package to create complex animations for water and other fluids, as well as hair and cloth movement, without the performance cost of calculating such advanced realistic simulations. Alembic bakes the vertex movement into a file that's played back during Play mode.

Install the Alembic package in the Package Manager to import alembic.abc files into Unity.


Package Manager

Packages: Unity Registry | Sort: Name (asc) | Filters | Clear Filters

All | Services

Addressables	1.21.19
Ads Mediation	1.0.0
Advertisement Legacy	4.4.2
AI Navigation	1.1.5
Alembic	2.3.4 ✓
Analytics	5.0.1
Android Logcat	1.3.2
Animation Rigging	1.2.1 ✓
Apple ARKit XR Plugin	5.0.7
AR Foundation	5.0.7
Authentication	3.1.0
Build Automation	1.0.5
Burst	1.8.8 ✓

Alembic

2.3.4 · November 03, 2023 Release

From **Unity Registry** by Unity Technologies Inc.

com.unity.formats.alembic

Documentation | Changelog | Licenses

Remove

Description | Version History | Dependencies

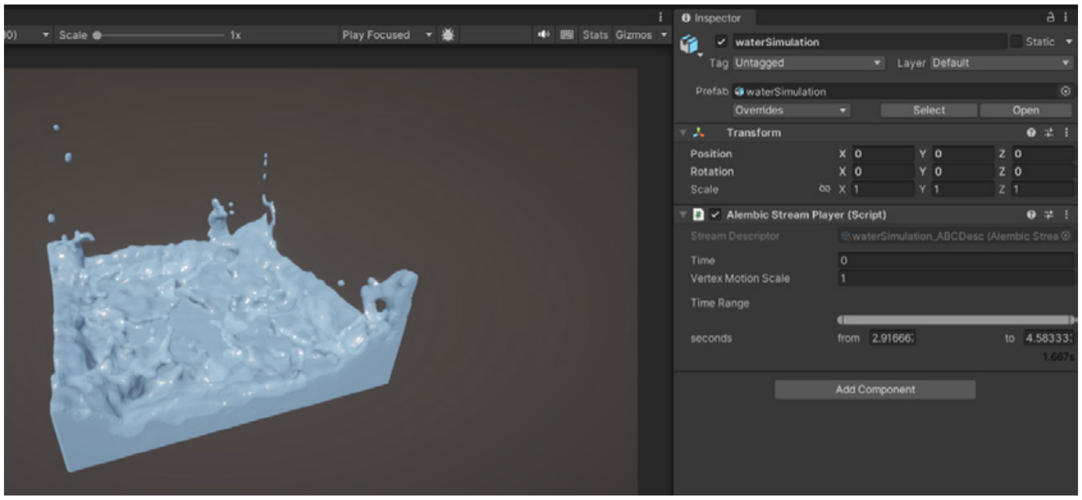
Import and export Alembic files (.abc), record and stream animations in the Alembic format directly in Unity.
NOTE: Supported build platforms: only Windows 64, macOS X, and Linux 64.

The Alembic format is commonly used in animation to transfer facial, cloth, and other simulation between applications in the form of baked data.

The Alembic importer and playback package in the Unity Package Manager

The simulations are usually made in other software, like Maya, Blender, or Marvelous Designer and then exported using the .abc alembic file extension. Alembic file sizes can be very large as they store vertex positions per frame – expect file sizes to be over one gigabyte or above, depending on the complexity of the simulation. Therefore, they're best-suited to animated sequences rather than real-time gameplay.

Import the alembic file by dragging it into the Project window, and then add it to your scene.



Inspector

waterSimulation | Static

Tag: Untagged | Layer: Default

Prefab: waterSimulation

Overrides | Select | Open

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Alembic Stream Player (Script)

Stream Descriptor: waterSimulation_ABCDesc (Alembic Strea

Time: 0

Vertex Motion Scale: 1

Time Range

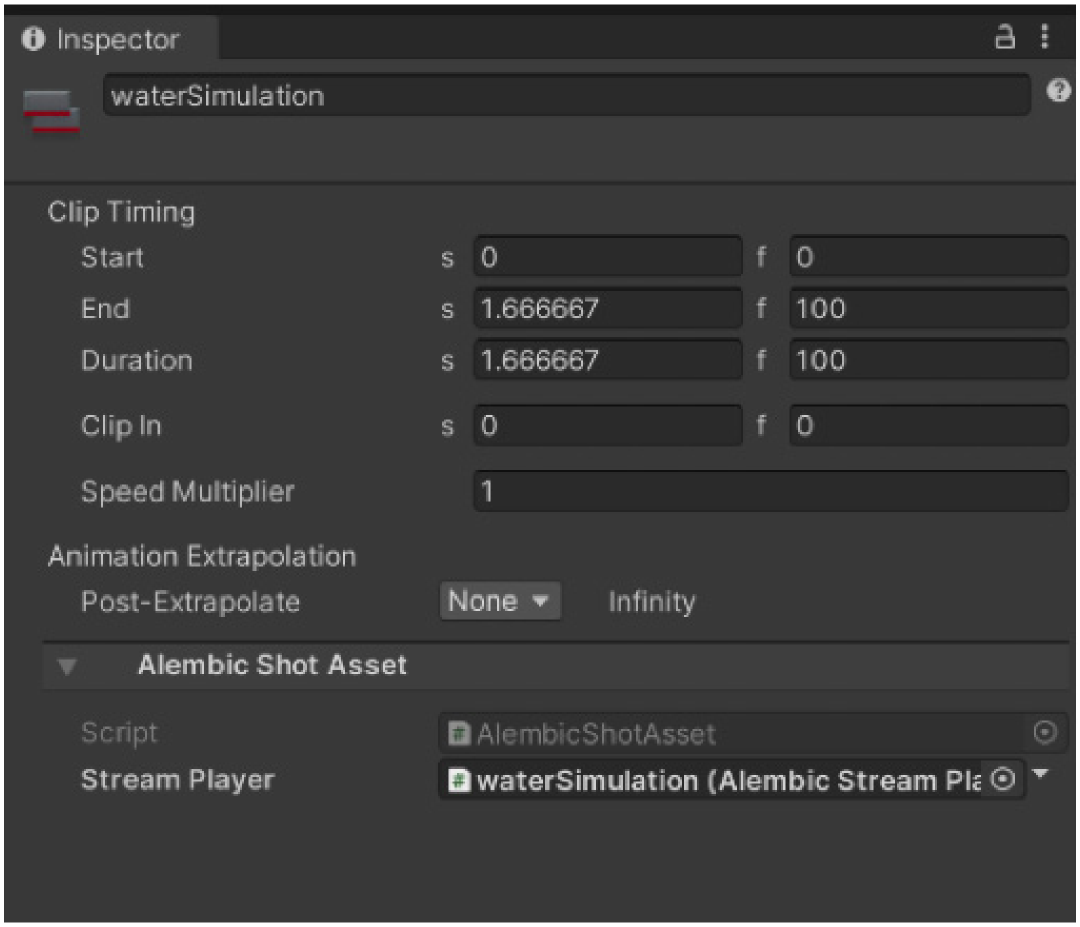
seconds: from 2.91666 to 4.58333

1.667s

Add Component

An alembic fluid simulation created in Blender and imported into Unity

To play back the alembic file, add the object to a Timeline sequence using an Alembic track.



waterSimulation

Clip Timing

Start	s	0	f	0
End	s	1.666667	f	100
Duration	s	1.666667	f	100
Clip In	s	0	f	0
Speed Multiplier		1		

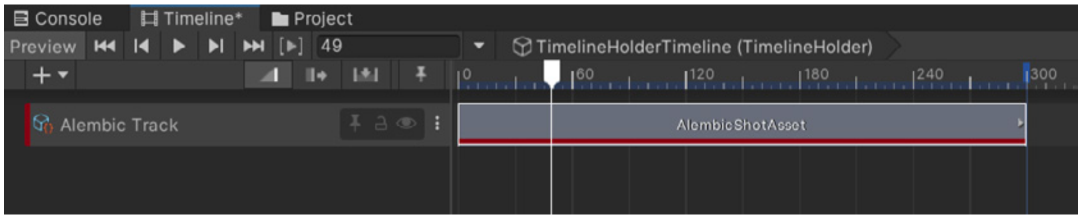
Animation Extrapolation

Post-Extrapolate None Infinity

Alembic Shot Asset

Script AlembicShotAsset

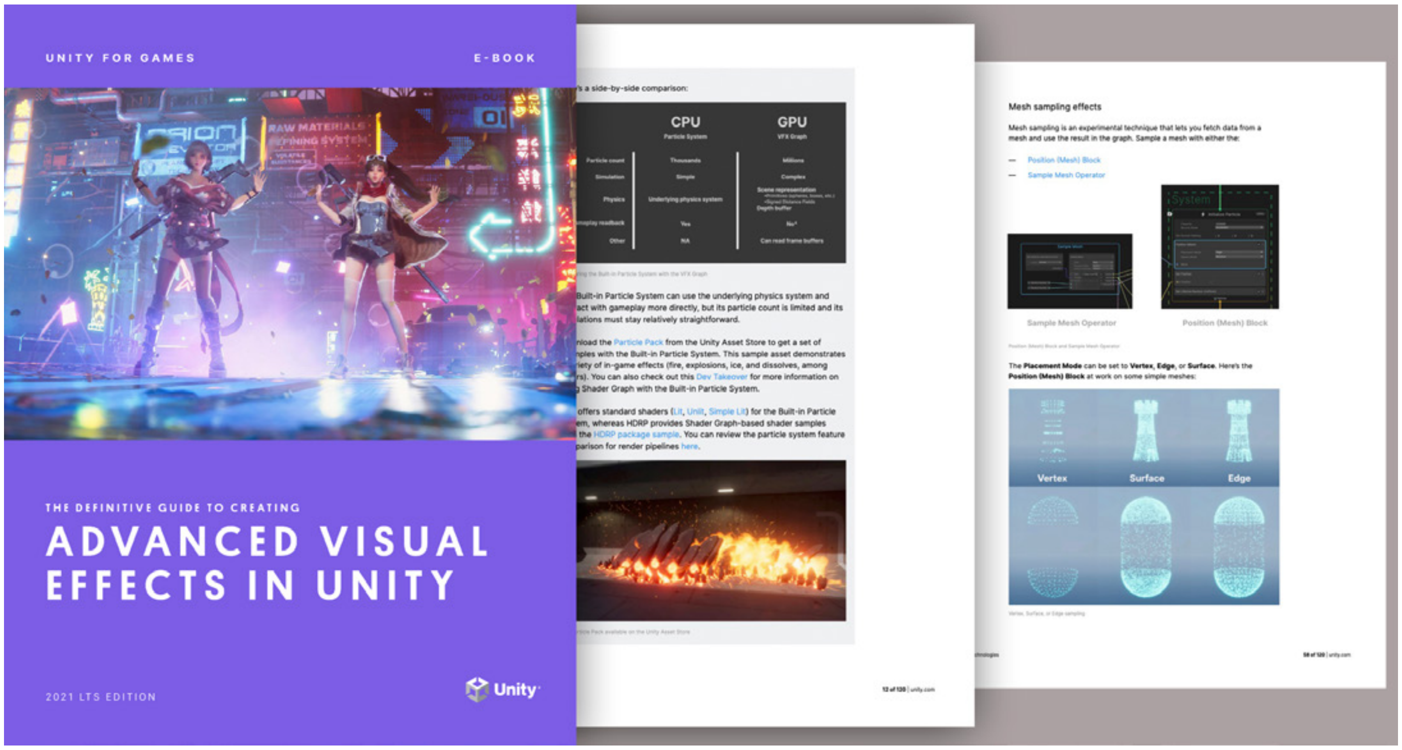
Stream Player waterSimulation (Alembic Stream Pla



Playback of alembic files using the Alembic track in Timeline

Particles

Particles are used to create effects like fire, smoke, water, magic attacks, and more. You can render thousands of particles in a scene with little impact on the frame rate, making them very useful. Find a wide range of particle effects on the Unity Asset Store, and discover how to craft particle effects in Unity with the e-book *The definitive guide to creating advanced visual effects in Unity*.

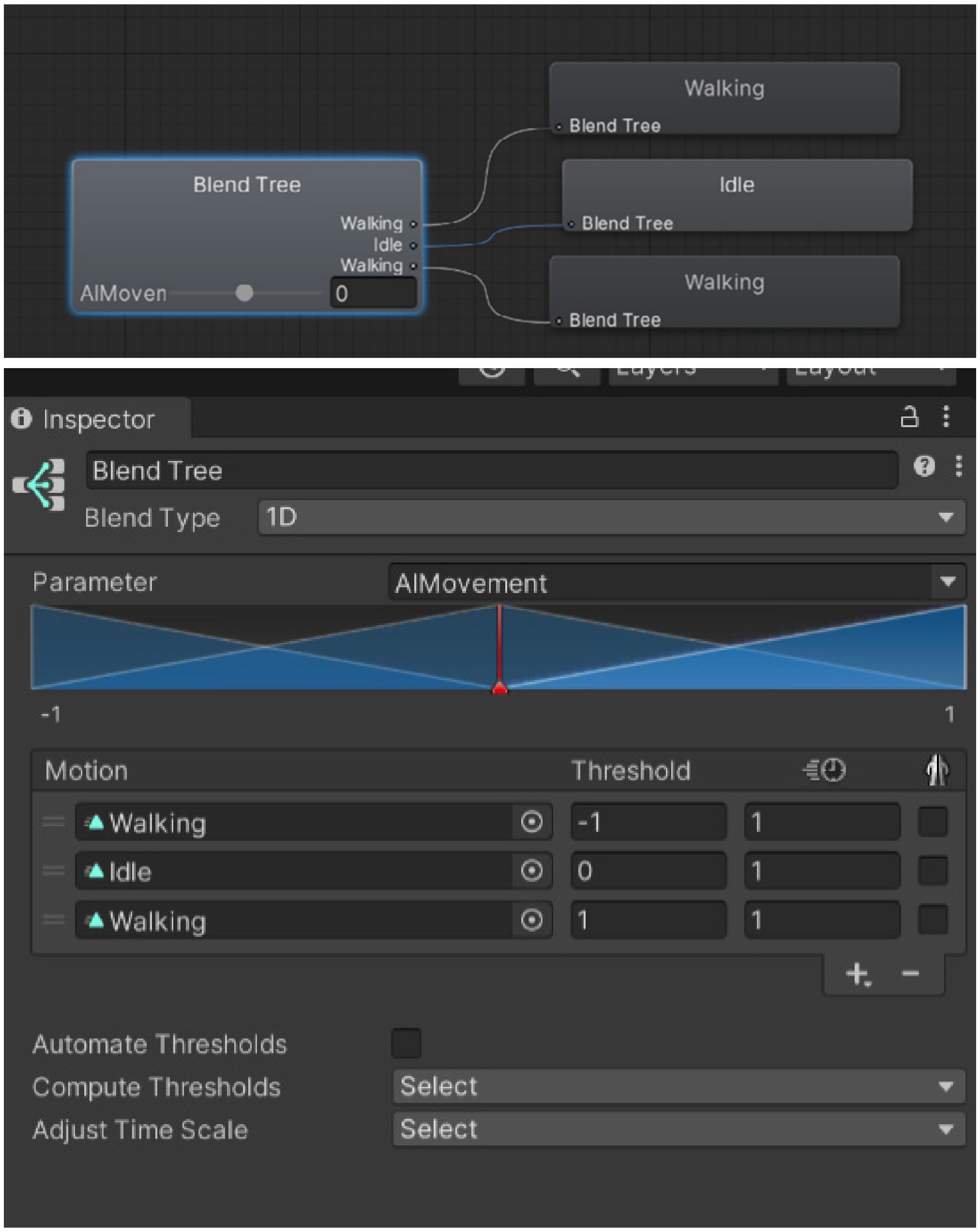

UNITY FOR GAMES
E-BOOK
THE DEFINITIVE GUIDE TO CREATING
ADVANCED VISUAL EFFECTS IN UNITY
2021 LTS EDITION
Unity

Download the Unity e-book

AI characters

When you have to animate a non-playable character (NPC) or have a scene with a large crowd, you can use Unity's AI Navigation system, which will help you avoid spending hours animating each character. This system allows you to control hundreds of characters in the scene and ensure they avoid walking through buildings or objects, as well as bumping into each other.

To start with the navigation system, create a blend tree for a character controlled by a variable that goes from -1 to 1 to transition between idle and walking.



Blend Tree

Blend Type: 1D

Parameter: AIMovement

-1 … 1

Motion	Threshold		
Walking	-1	1	
Idle	0	1	
Walking	1	1	

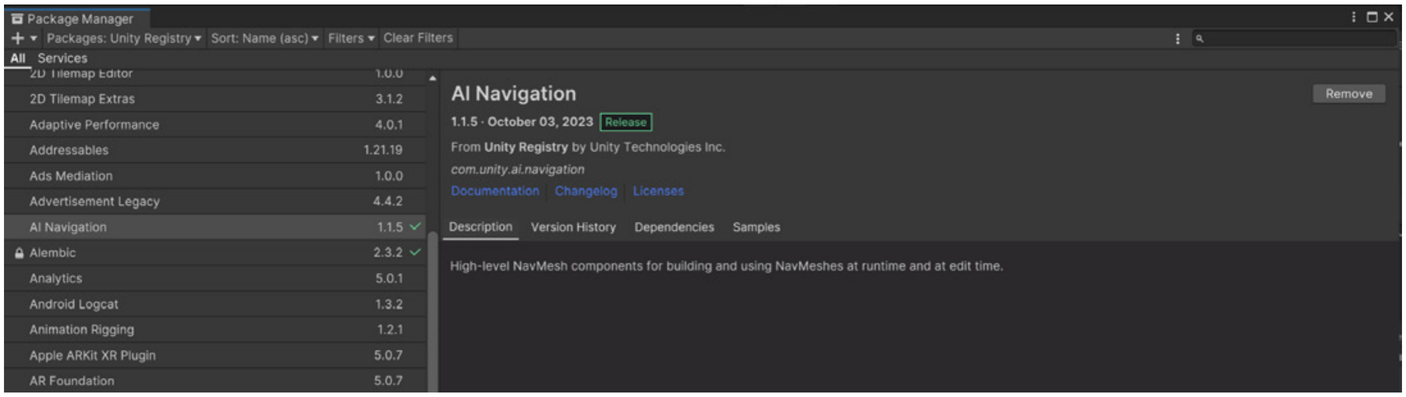
Automate Thresholds: ☐

Compute Thresholds: Select

Adjust Time Scale: Select

Blend tree animator for an NPC character

Install the AI Navigation package from the Package Manager.

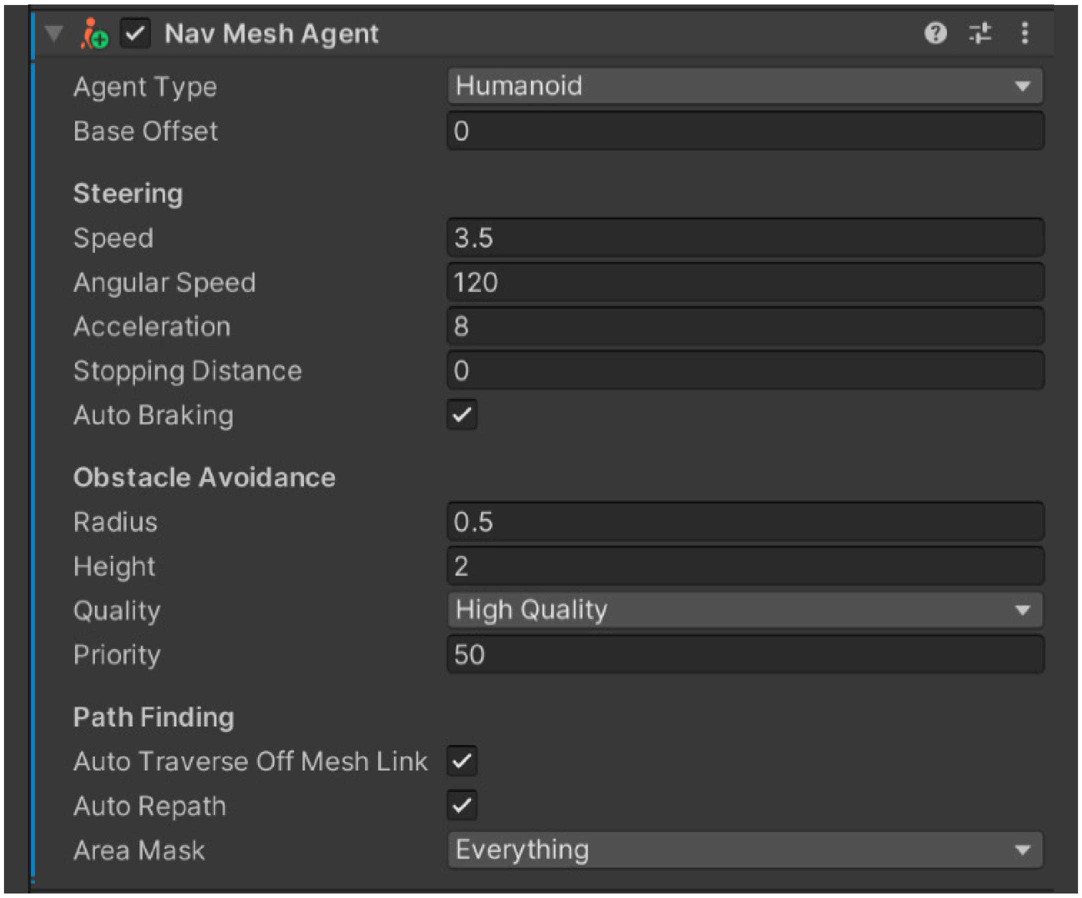

Package Manager
Packages: Unity Registry ▾ Sort: Name (asc) ▾ Filters ▾ Clear Filters
All Services

2D Tilemap Editor	1.0.0
2D Tilemap Extras	3.1.2
Adaptive Performance	4.0.1
Addressables	1.21.19
Ads Mediation	1.0.0
Advertisement Legacy	4.4.2
AI Navigation	1.1.5 ✓
Alembic	2.3.2 ✓
Analytics	5.0.1
Android Logcat	1.3.2
Animation Rigging	1.2.1
Apple ARKit XR Plugin	5.0.7
AR Foundation	5.0.7

AI Navigation
1.1.5 · October 03, 2023 Release
From **Unity Registry** by Unity Technologies Inc.
com.unity.ai.navigation
Documentation Changelog Licenses
Remove
Description Version History Dependencies Samples
High-level NavMesh components for building and using NavMeshes at runtime and at edit time.

The AI Navigation package in the Package Manager

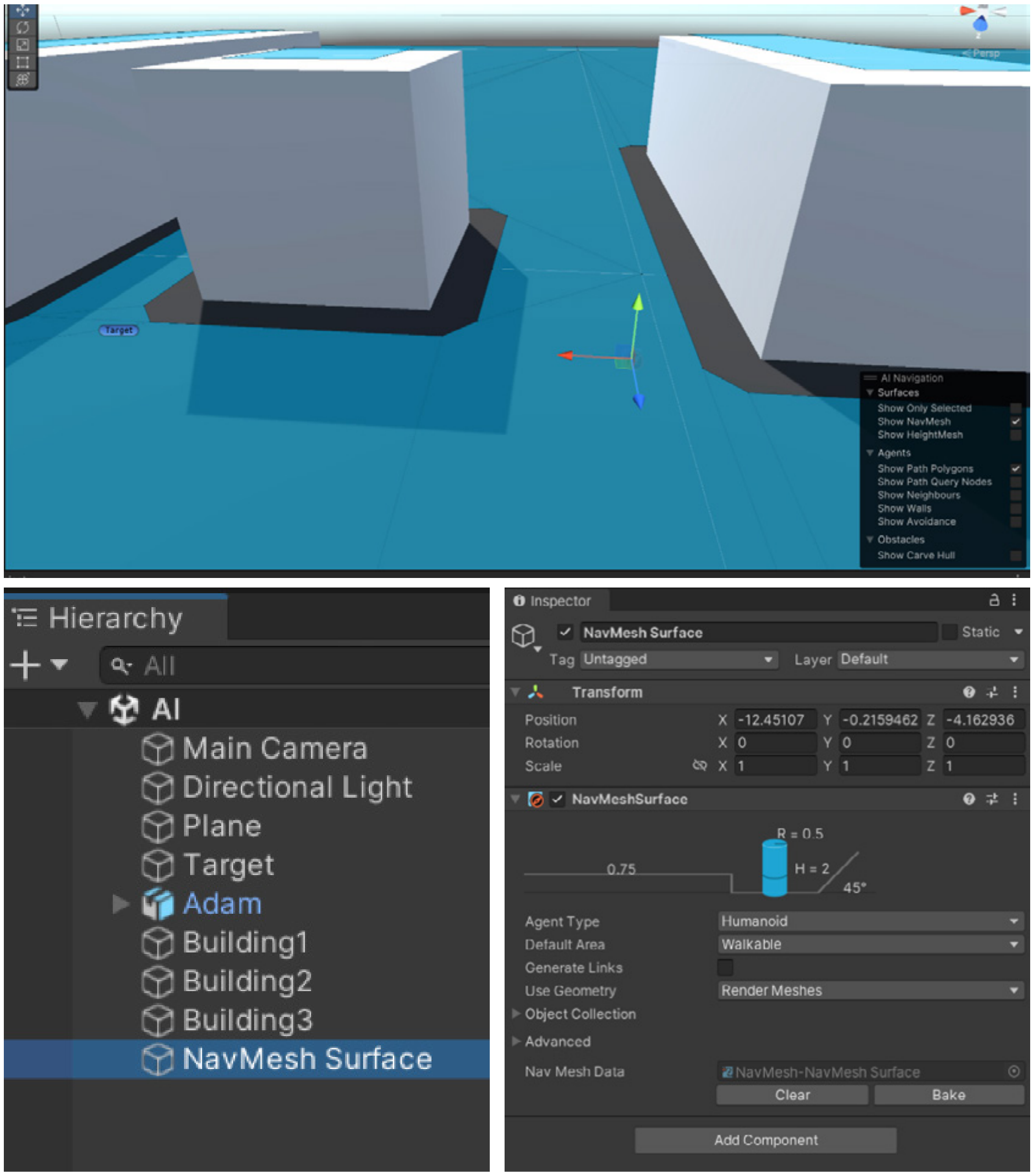
Now you can add a NavMesh agent to your NPC. You can change the settings to control how fast the character walks and rotates, as well as the radius of the agent, which determines how close other characters can get to them.


Nav Mesh Agent

Agent Type	Humanoid
Base Offset	0
Steering	
Speed	3.5
Angular Speed	120
Acceleration	8
Stopping Distance	0
Auto Braking	✓
Obstacle Avoidance	
Radius	0.5
Height	2
Quality	High Quality
Priority	50
Path Finding	
Auto Traverse Off Mesh Link	✓
Auto Repath	✓
Area Mask	Everything

NavMesh agent settings for NPC control

The NavMesh agent will need a NavMesh Surface component to direct their movement. Go to **GameObject > AI > Navmesh surface**. Switch off any dynamic characters before baking the NavMesh. This will show up as a blue surface when you have the **Show NavMesh** option checked. Add a target object for the NPC to walk toward.



AI Navigation
Surfaces
Show Only Selected
Show NavMesh ✓
Show HeightMesh
Agents
Show Path Polygons ✓
Show Path Query Nodes
Show Neighbours
Show Walls
Show Avoidance
Obstacles
Show Carve Hull

Hierarchy
- AI
  - Main Camera
  - Directional Light
  - Plane
  - Target
  - Adam
  - Building1
  - Building2
  - Building3
  - NavMesh Surface

Inspector
NavMesh Surface ✓ Static
Tag Untagged Layer Default

Transform			
Position	X -12.45107	Y -0.2159462	Z -4.162936
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

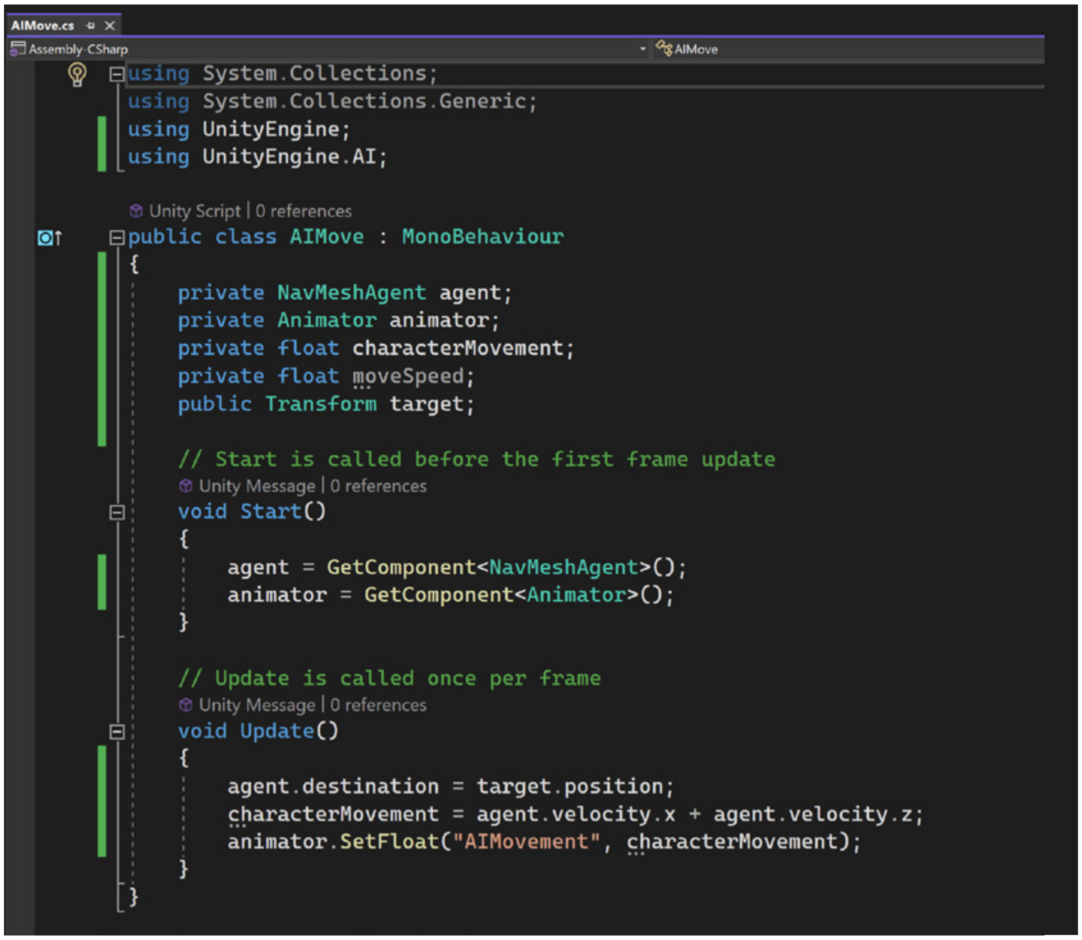
NavMeshSurface ✓
R = 0.5, 0.75, H = 2, 45°

Agent Type	Humanoid
Default Area	Walkable
Generate Links	
Use Geometry	Render Meshes
Object Collection	
Advanced	
Nav Mesh Data	NavMesh-NavMesh Surface

Clear Bake
Add Component

Baking and showing a NavMesh in the scene; the NPC can only walk in the blue areas

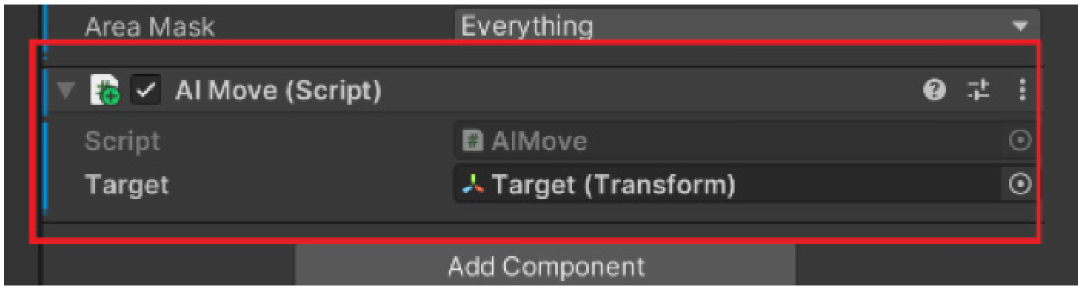
You will need a script to control the NavMesh Agent. This uses the velocity of the agent to tell the animator whether it should play the walk when in motion or idle when stopped.



```
AIMove.cs
Assembly-CSharp                                   AIMove
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

Unity Script | 0 references
public class AIMove : MonoBehaviour
{
    private NavMeshAgent agent;
    private Animator animator;
    private float characterMovement;
    private float moveSpeed;
    public Transform target;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
        animator = GetComponent<Animator>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        agent.destination = target.position;
        characterMovement = agent.velocity.x + agent.velocity.z;
        animator.SetFloat("AIMovement", characterMovement);
    }
}
```

Script to control the NavMesh Agent on the NPC

Drag the target into the slot of this script. You can have as many targets and characters as you like. If possible, keep things simple by having one target per character.



Area Mask: Everything
AI Move (Script)
Script: AIMove
Target: Target (Transform)

The NPC will always walk toward the target and stop once it reaches it.

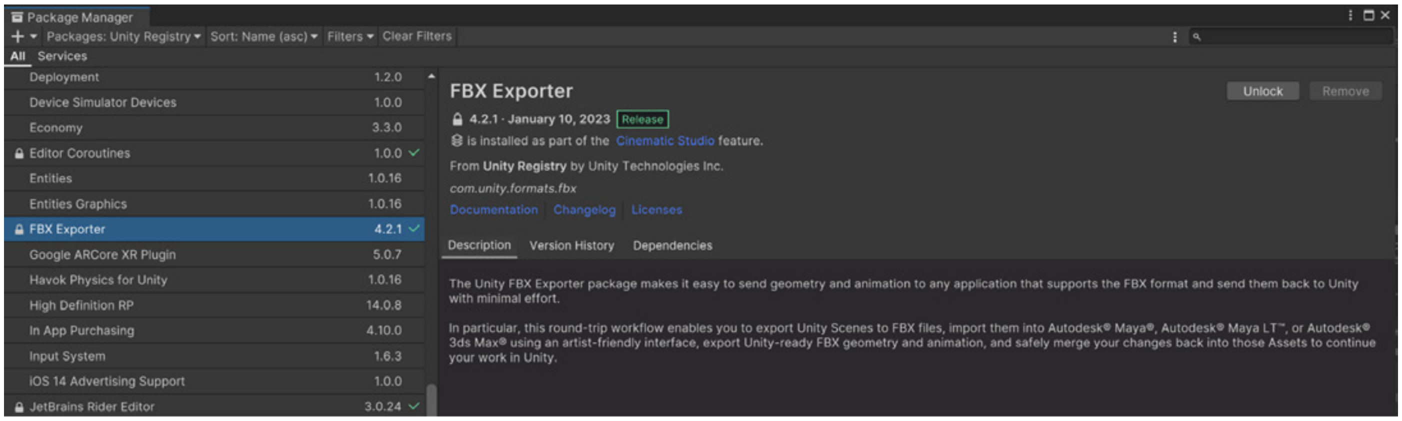
Now the character will walk toward the target during Play mode. You can animate the moving target to get the NPC to follow it. Notice how easy it can be to create entire crowd simulations using AI navigation.

Exporting animations

If you work in a team or want to refine animations in a DCC tool there might be numerous times when you will need to export animated characters from Unity to other software to make adjustments.

Export FBX models

In the Package Manager ensure that the FBX Exporter has been installed from the Registry.



Deployment	1.2.0
Device Simulator Devices	1.0.0
Economy	3.3.0
Editor Coroutines	1.0.0 ✓
Entities	1.0.16
Entities Graphics	1.0.16
FBX Exporter	4.2.1 ✓
Google ARCore XR Plugin	5.0.7
Havok Physics for Unity	1.0.16
High Definition RP	14.0.8
In App Purchasing	4.10.0
Input System	1.6.3
iOS 14 Advertising Support	1.0.0
JetBrains Rider Editor	3.0.24 ✓

FBX Exporter

4.2.1 · January 10, 2023 Release

is installed as part of the Cinematic Studio feature.

From **Unity Registry** by Unity Technologies Inc.

com.unity.formats.fbx

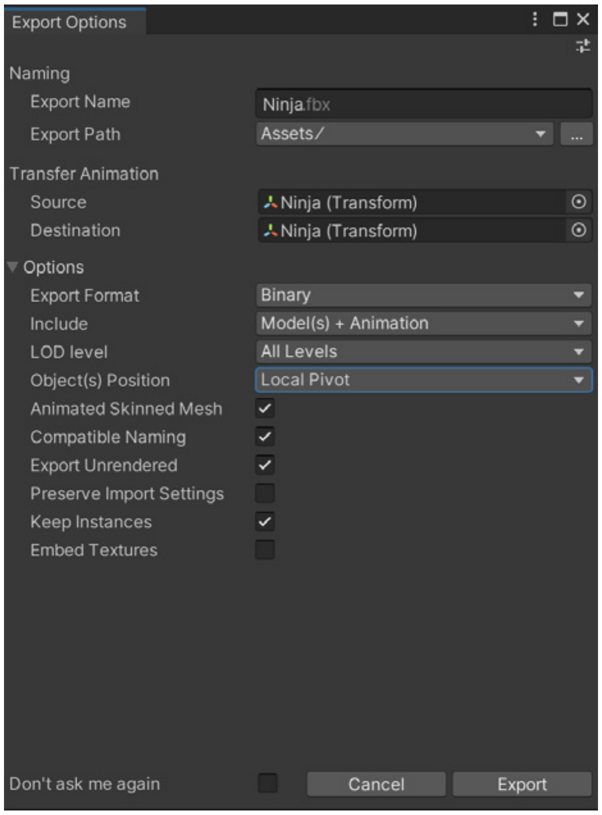
Documentation | Changelog | Licenses

The Unity FBX Exporter package makes it easy to send geometry and animation to any application that supports the FBX format and send them back to Unity with minimal effort.

In particular, this round-trip workflow enables you to export Unity Scenes to FBX files, import them into Autodesk® Maya®, Autodesk® Maya LT™, or Autodesk® 3ds Max® using an artist-friendly interface, export Unity-ready FBX geometry and animation, and safely merge your changes back into those Assets to continue your work in Unity.

You'll need to install the FBX Exporter to export FBX files.

The character should have the animations set up in the Animator window. Right-click the character and choose **Export to FBX**.



Export Options

Naming

Export Name	Ninja.fbx
Export Path	Assets/

Transfer Animation

Source	Ninja (Transform)
Destination	Ninja (Transform)

Options

Export Format	Binary
Include	Model(s) + Animation
LOD level	All Levels
Object(s) Position	Local Pivot
Animated Skinned Mesh	✓
Compatible Naming	✓
Export Unrendered	✓
Preserve Import Settings	
Keep Instances	✓
Embed Textures	

Don't ask me again

FBX Export options

The export box will open, allowing you to choose the destination folder. By default this is in the Project Assets folder.

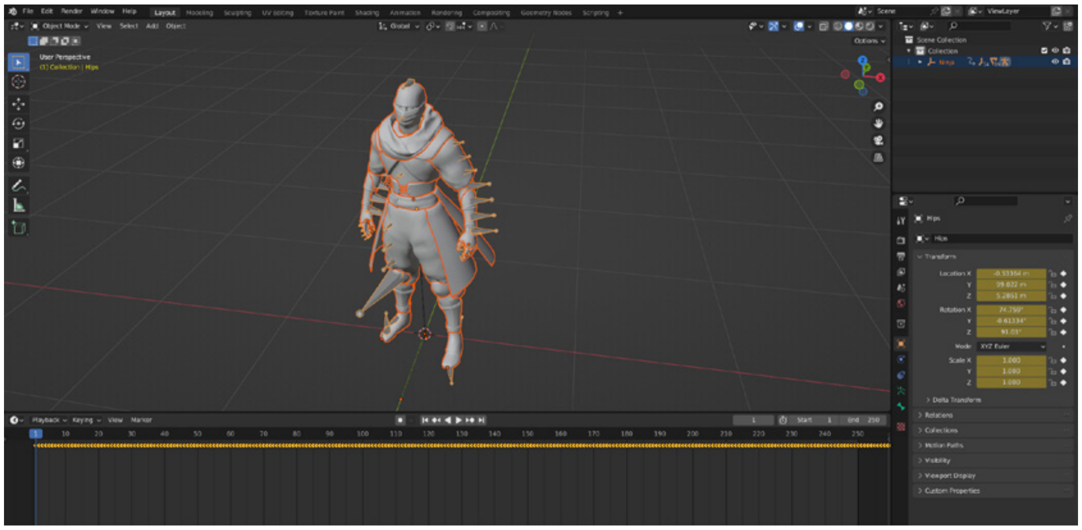
You can export the file in ASCII or binary format depending on which software you will open the FBX file in. Choose binary for Blender.

The file can be just the 3D mesh or the animation files, or both.

Choose to export all LOD levels if they exist, or either the highest or lowest level of detail.

Check the animated skinned mesh to ensure that skinned geometry shows up in the software. If you want the textures to also appear in other software, check the **Embed Textures** option.

Once exported you can now open the file in other 3D software to make adjustments to either the 3D mesh or to the animation files.



The exported FBX file in Blender with embedded animations

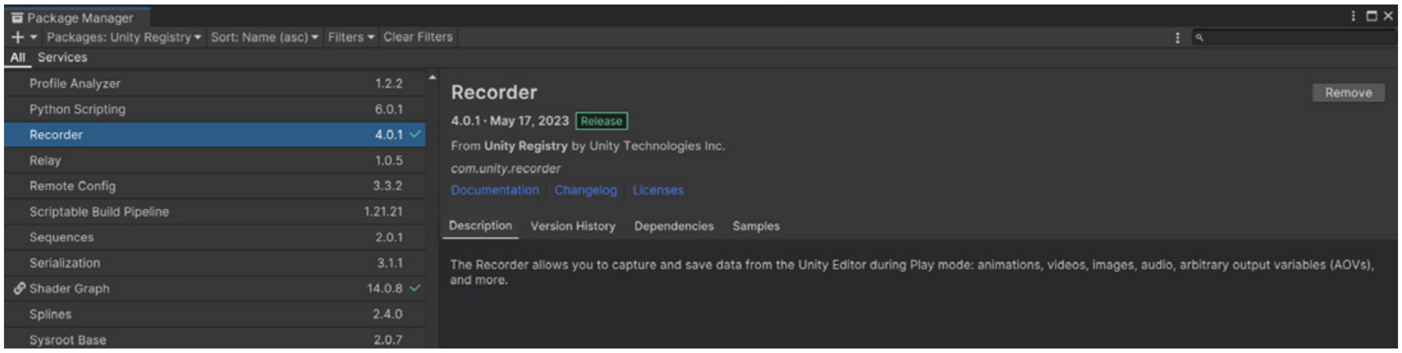
The embedded animations are available in the non linear animation window in Blender. Once you adjust the file you can export it as an FBX file and open it in Unity.

Unity Recorder

If you are creating an animated feature in Unity or need high-quality footage of your game then export out the final animation as a video file.

Capturing raw footage while playing the game in the Editor or a Build simultaneously to running video capturing software can be tasking on the resources of your computer making the game not run as it should for marketing or video content purposes. To ensure a stable frame rate, not stuttering due to frame skipping, and achieve the best possible recording quality Unity Recorder lets you create video or image sequences in real-time from inside the Editor.

Install the Recorder from the Package Manager.



Package Manager

Packages: Unity Registry | Sort: Name (asc) | Filters | Clear Filters

All | Services

Profile Analyzer	1.2.2
Python Scripting	6.0.1
Recorder	4.0.1 ✓
Relay	1.0.5
Remote Config	3.3.2
Scriptable Build Pipeline	1.21.21
Sequences	2.0.1
Serialization	3.1.1
Shader Graph	14.0.8 ✓
Splines	2.4.0
Sysroot Base	2.0.7

Recorder

4.0.1 · May 17, 2023 Release

From Unity Registry by Unity Technologies Inc.

com.unity.recorder

Documentation Changelog Licenses

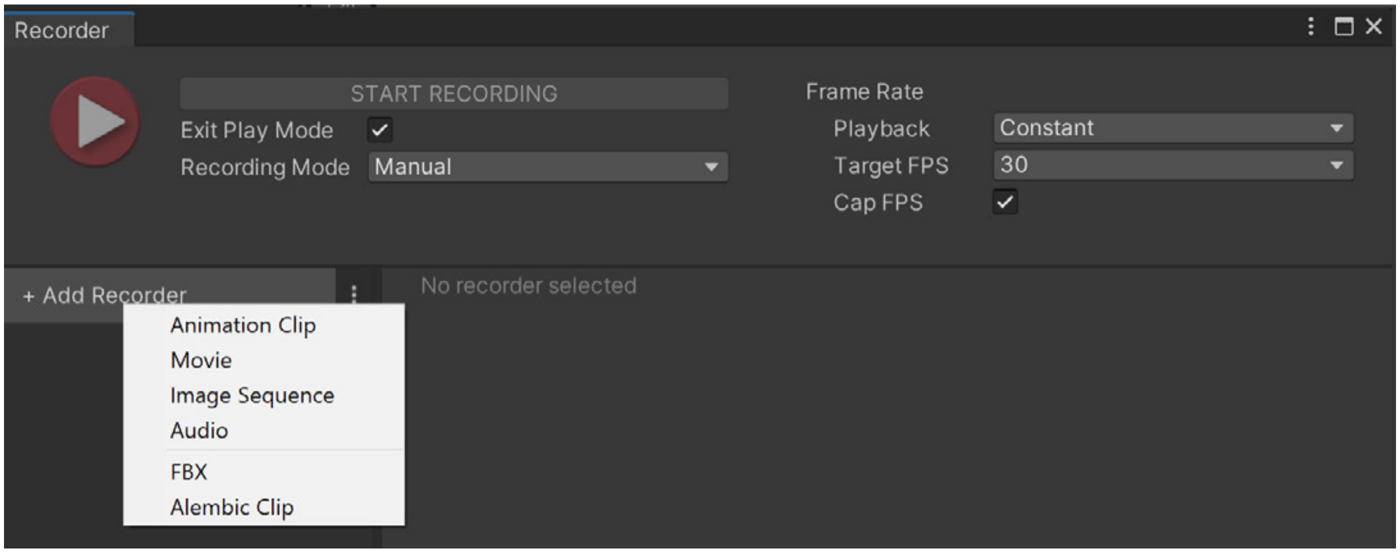
Description | Version History | Dependencies | Samples

The Recorder allows you to capture and save data from the Unity Editor during Play mode: animations, videos, images, audio, arbitrary output variables (AOVs), and more.

The Recorder package for exporting videos and image sequences

With your scene setup and ready to be recorded go to **Window > General > Recorder > Recorder window**. You can set a frame rate from a set list or define your own. Keep the bit rate constant for a more stable video file.

To create a new recording click the **Add Recorder** tab and choose from the available options. You can create a movie, an image sequence, or export audio.



Recorder

START RECORDING

Exit Play Mode ✓

Recording Mode: Manual

Frame Rate

Playback: Constant

Target FPS: 30

Cap FPS ✓

+ Add Recorder

No recorder selected

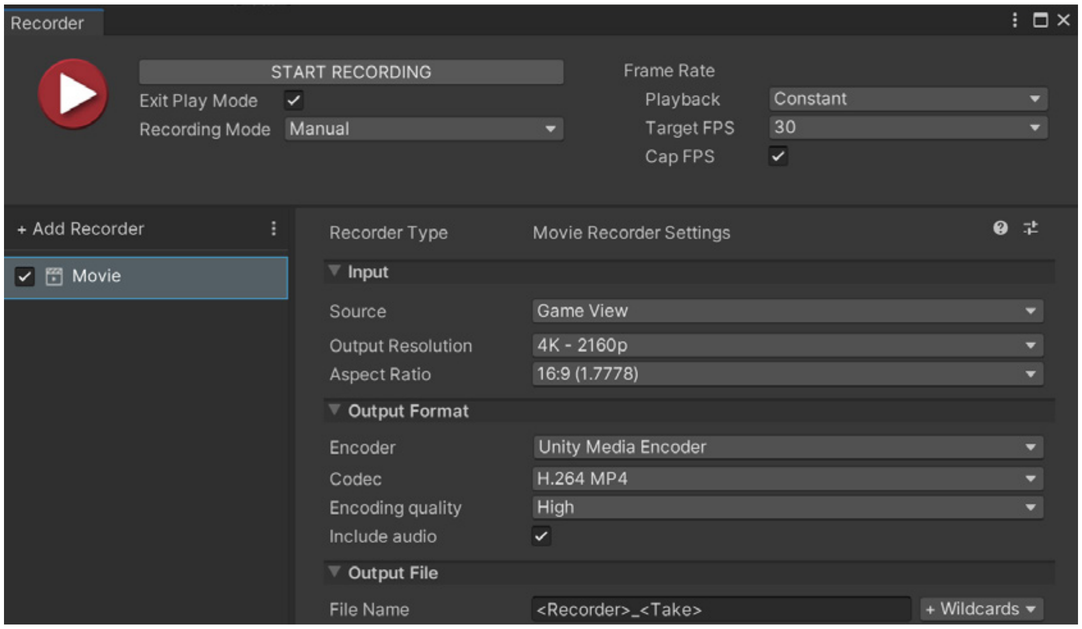
Creating a new recorder

Select the **Movie** option and customize the settings to fit your export requirements. The video will be recorded from the Scene view, so anything displayed there will be recorded. You can also choose to record video only from a specific camera, or a 360° view for VR.

You can also set an output resolution from a set list or define your own and the Scene view will then conform to that resolution when recording. Be aware that if your scene has been set up in a different ratio, for example 4:3, and you select 16:9 as an output, the Scene view will change to 16:9 as it records, resulting in your video looking different to the scene in the Editor.

In the output format you can select which encoder you prefer to use. The Unity Media Encoder supports the H.264 MP4 and WebM codecs. The Apple ProRes encoder supports Apple ProRes 4444 and 422 codecs. The Gif encoder is used for exporting animated gifs.

You can also export video with or without audio and the files will be saved in your project in a folder called recordings.



Exit Play Mode ✓
Recording Mode Manual
Frame Rate
Playback Constant
Target FPS 30
Cap FPS ✓
Recorder Type Movie Recorder Settings
Input
Source Game View
Output Resolution 4K - 2160p
Aspect Ratio 16:9 (1.7778)
Output Format
Encoder Unity Media Encoder
Codec H.264 MP4
Encoding quality High
Include audio ✓
Output File
File Name <Recorder>_<Take> + Wildcards

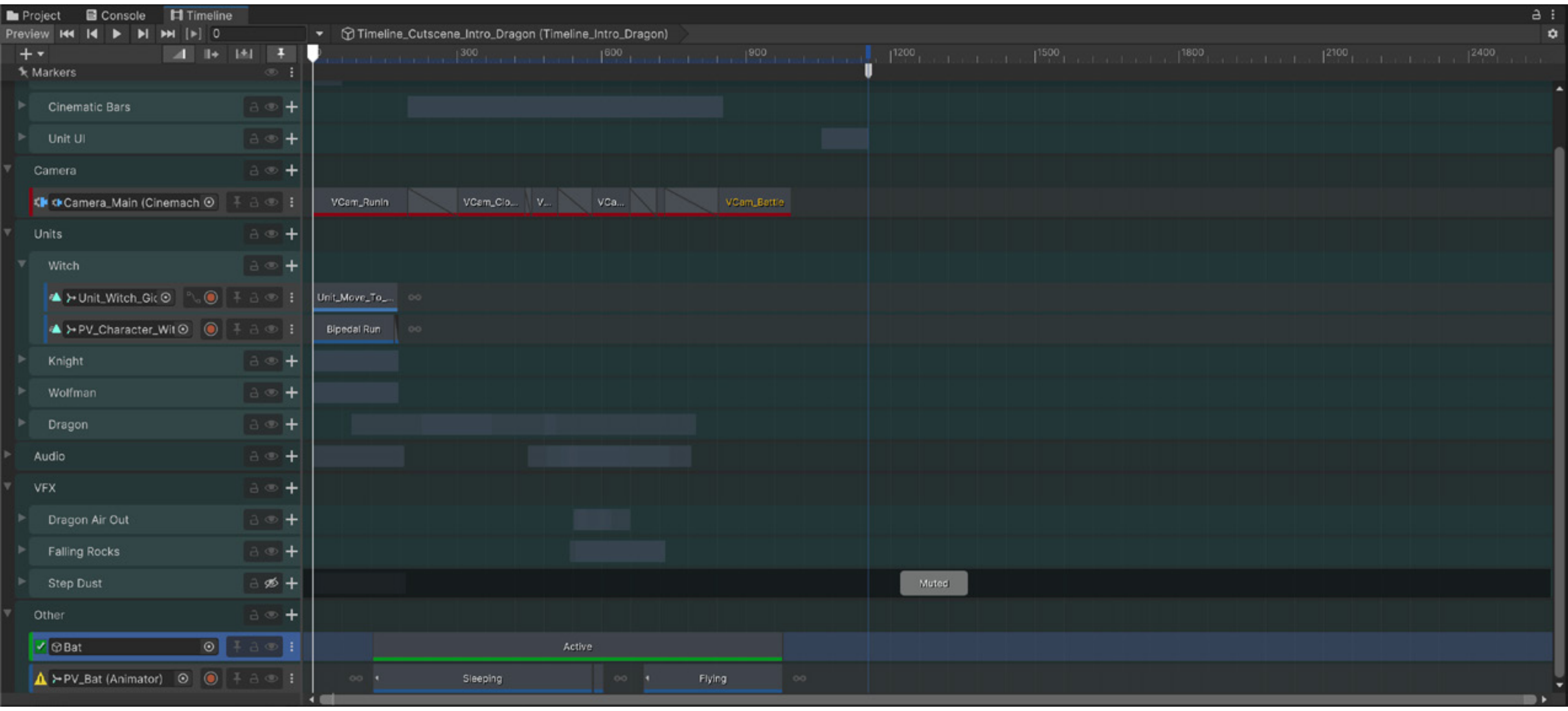
Movie settings for exporting video files

You could also export out your project as an image sequence in JPG, PNG, or EXR files. These may be useful if you want to do the final encoding in video editing software, perhaps after color grading and adding video effects.

Press **Start Recording** to go into record mode, and the sequence will begin to play as the video is recorded. You may notice that it will cap at the target frame rate and even play slower than the target frame rate. This is a result of the encoding process; the video however, will play back smoothly at the target frame rate.

If the recording mode is set to manual you will have to press stop at the set time. You can type in the start and end frames for the recording under the recording mode setting.

You can also use Timeline to add a recorder track. Right-click the track to add a RecorderClip. This acts like an active block. It will record for the duration of this block.



Setting the recorder start and stop points using Timeline

2D Animation

2D projects in Unity share similar tools and workflows. The equivalent in a 2D project to an FBX 3D model is a 2D sprite. There are also 2D physics and lighting toolsets in Unity, in addition to a 2D Sprite Editor, the 2D Tilemap system and 2D Sprite Shape. The Animator Controller and Timeline tools also work for 2D projects.

Traditional 2D animation can be the most time-consuming and challenging part of developing art for a 2D game. 2D skeletal animation can help you save time, create smooth and adaptable animation, and enable you to create more animations with fewer assets.

The PSD workflow

The 2D Animation package enables you to import your character artwork directly from Photoshop into Unity. The 2D PSD Importer package handles layered PSD and PSB files for this purpose. You can import all of the character's layers as sprites and place them exactly as they were painted in the app.

When you select an imported PSD or PSB file, it shows options similar to the sprite import settings, but with additional ones for 2D animation and rigging. The key options you'll need to select are:

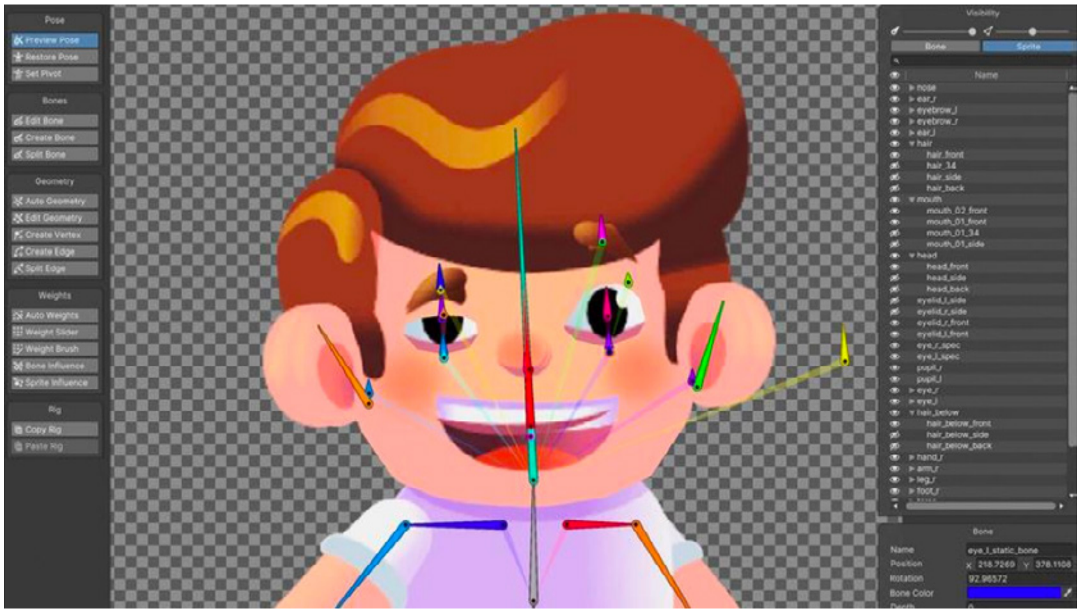
— **Import Hidden Layers:** This option imports all layers from a PSD or PSB file, including hidden layers.

— **Individual sprites (Mosaic):** This setting is only available if the Texture Type is set to Multiple. It makes sprites from imported layers and puts them onto a texture atlas. Leave this option enabled if you want to rig a character.

— **Use as Rig:** This option generates a character prefab with the same layer hierarchy and position as the PSD or PSB file.

— **Use Layer Group:** This adds layer grouping from the PSD or PSB file. Turn it on to group parts of a character, like sprite-swapped parts.

In the Layer Management section you can see the hierarchy of layers, and manually add or remove layers to be imported.



The main character from the sample project *Happy Harvest*

Rigging in the Sprite Editor

You can create a character's skeleton in the Skinning Editor inside the Sprite Editor. Start by clicking the Create Bone button and then left-click in the main window area. The first click creates a bone center, and the second marks the bone tip location. This tool chains and nests bones inside one another.

Use the Edit Bone button to refine bones. The Split Bone button enables you to split bones in two. This is a good option for making limbs. If you make one leg bone and click where the knee should be, the bone will split into the thigh and calf sections.

To verify that the hierarchy is correct, select the Preview Pose button and test some poses. To reset the position of bones, press the Reset Pose button on the toolbar.

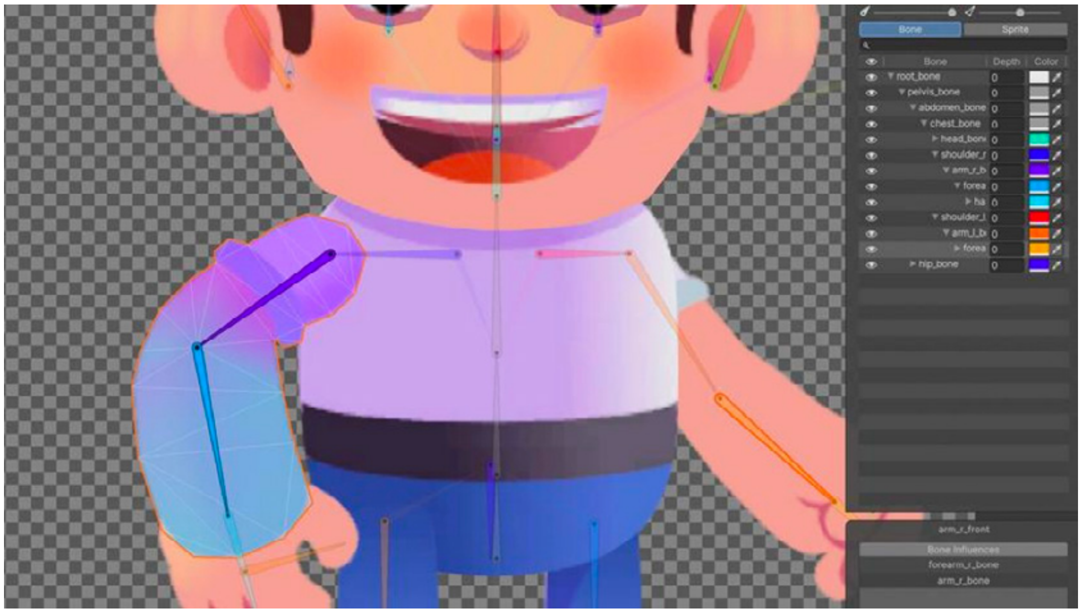
Connecting sprites to bones, geometry, and weights

In order to assign sprites to bones, you'll need to create geometry. Start by pressing the Auto Geometry button. A small pop-up window will open, and you can define how the geometry will be created.

It’s a good idea to set all sliders to zero to keep the geometry as simple as possible. Enable the Weights option to bind bones to the sprites automatically. Clicking the Generate For All Visible button creates and sets bone weights for all sprites. To do this for an individual sprite, double-click the sprite. This is useful for tweaking the geometry of certain sprites.

To fully control a number of vertices and how the geometry bends, you’ll need to edit the mesh manually with the tools in the Geometry section.

Use as few vertices as possible. Fewer vertices help you save on performance, since each vertex position needs to be calculated based on the bones’ rotation. Fewer vertices also make for better-looking mesh bending since it’s easier to set weights for fewer points. Additionally, each target platform has an ideal game vertex count, so it’s always a good idea to optimize geometry.



Bone	Depth	Color
▼ root_bone	0	⬜
▼ pelvis_bone	0	⬜
▼ abdomen_bone	0	⬜
▼ chest_bone	0	⬜
▶ head_bon	0	🟩
▼ shoulder_r	0	🟦
▼ arm_r_b	0	🟪
▼ forea	0	🟦
▶ ha	0	🟦
▼ shoulder_l	0	🟥
▼ arm_l_b	0	🟧
▶ forea	0	🟨
▶ hip_bone	0	🟦

Clean geometry is important for optimized animations, especially in areas that experience high levels of deformation, like the articulations.

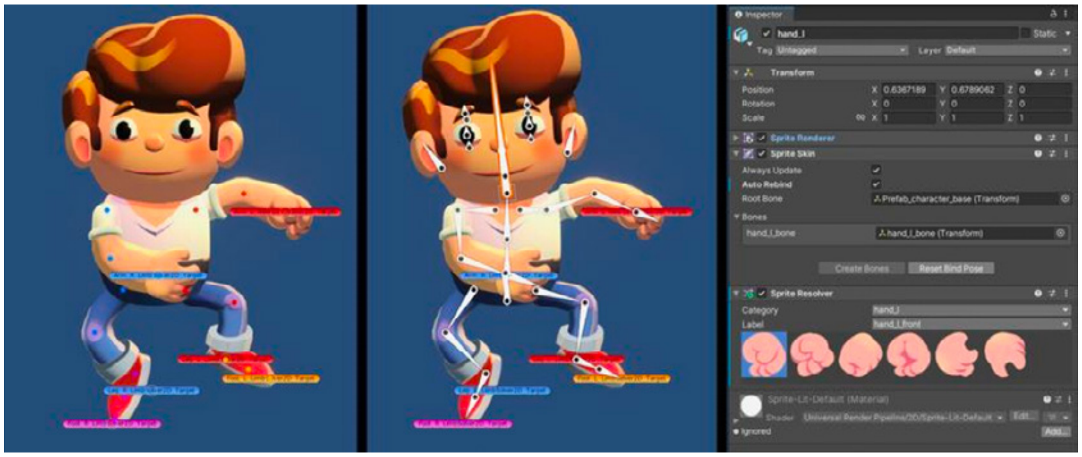
Weights

Once the geometry of your sprites is clean, it’s time to set up weights. Weights define bone influence on each vertex from zero to one. Zero means that a bone has no influence over the vertex, and one means that the vertex will move like it’s glued to the bone. Good weights on the mesh can result in realistic-looking bending. Setting weights incorrectly can break the game’s illusion and distort your sprites.

To start setting up weights, you need to define which bones will affect a certain sprite. Click the Bone Influence button, and select a sprite. In the small pop-up window, you can set which bones influence the selected sprite. Set only the relevant bones that need to affect the sprite, and remove the rest.

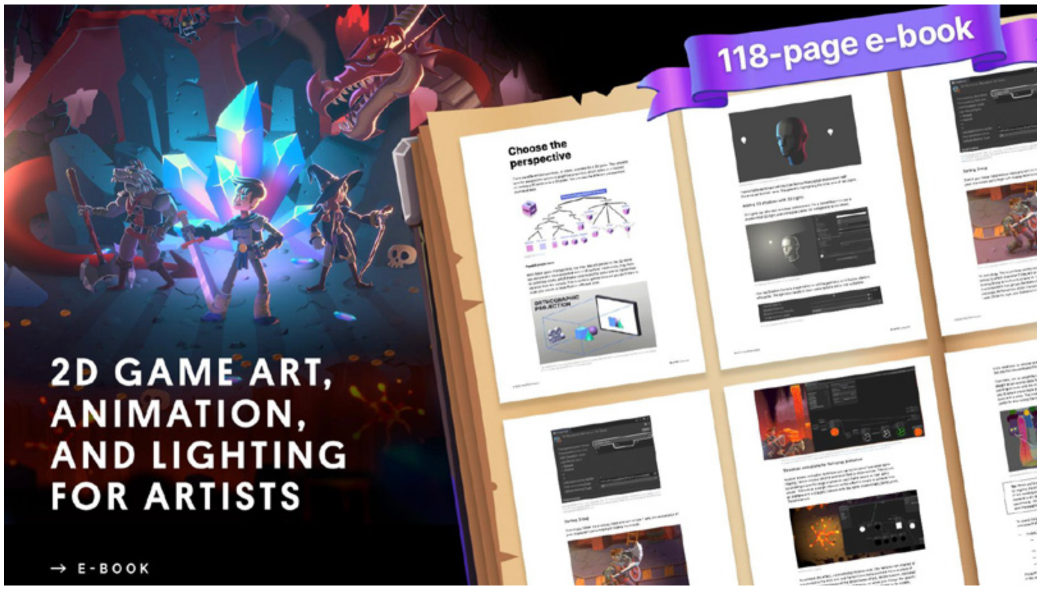
Sprite resolvers and 2D IK

In the Unity sample project, *Happy Harvest*, you can see how several sprites are influenced by the same bone. For example, the hand sprite has variations, including side, front, and back to swap during the animation process or gameplay. Additionally, to make animations react to the game world, 2D Inverse Kinematics calculates the rotations and allows a chain of bones to move them to target positions.

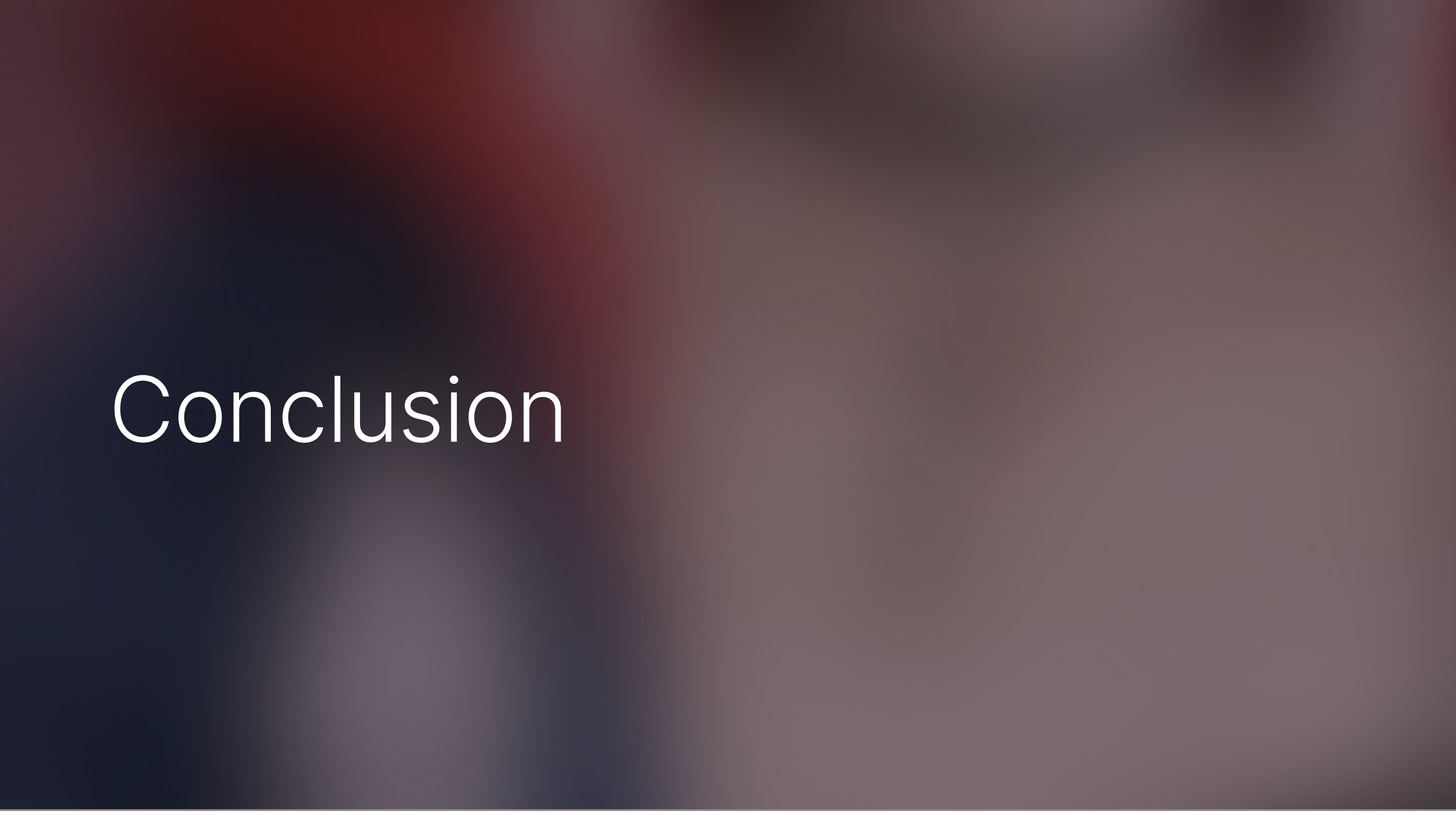


hand_l
Tag: Untagged
Layer: Default
Position: X 0.6367189, Y 0.6789062, Z 0
Rotation: X 0, Y 0, Z 0
Scale: X 1, Y 1, Z 1
Always Update: ✓
Auto Rebind: ✓
Root Bone: Prefab_character_base (Transform)
Bones: hand_l_bone — hand_l_bone (Transform)
Category: hand_l
Label: hand_l_front
Sprite-Lit-Default (Material)
Shader: Universal Render Pipeline/2D/Sprite-Lit-Default
Ignored

Animating the main character in *Happy Harvest* with 2D IK solvers. The left image shows modifying the position of the bones and rotation. The right image shows changing the sprite for the hand with sprite resolver



Download the Unity e-book about 2D graphics, tools, and animation



Conclusion

For artists looking to create animations in Unity, be sure to check out the full Unity Documentation, as well as Unity Learn, the Unity Blog, and the Animation Forum.

You can find all of Unity's in-depth developer and artist e-books in the Unity best practices hub or the best practices section in the Unity manual.

The Unity Product Board provides an overview of current animation features being developed, in addition to what's coming up next. You can even add your own feature requests.

Good luck with your game development.

Unity®

unity.com