유니티(Unity)를 활용한

그래픽스 프로그래밍

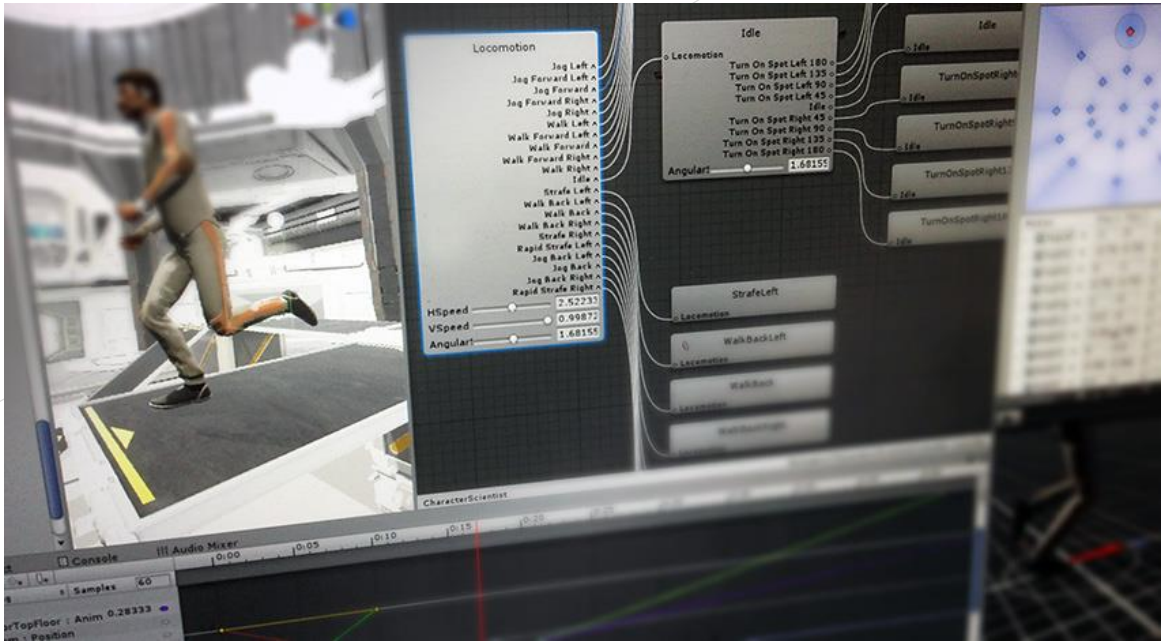
14 Animation

1

Animation

Animation

- Animation features include retargetable animations, full control of animation weights at runtime, event calling from within the animation playback, sophisticated state machine hierarchies and transitions, blend shapes for facial animations, etc.



이미지 출처 :https://docs.unity3d.com/Manual/AnimationSection.html

Animation

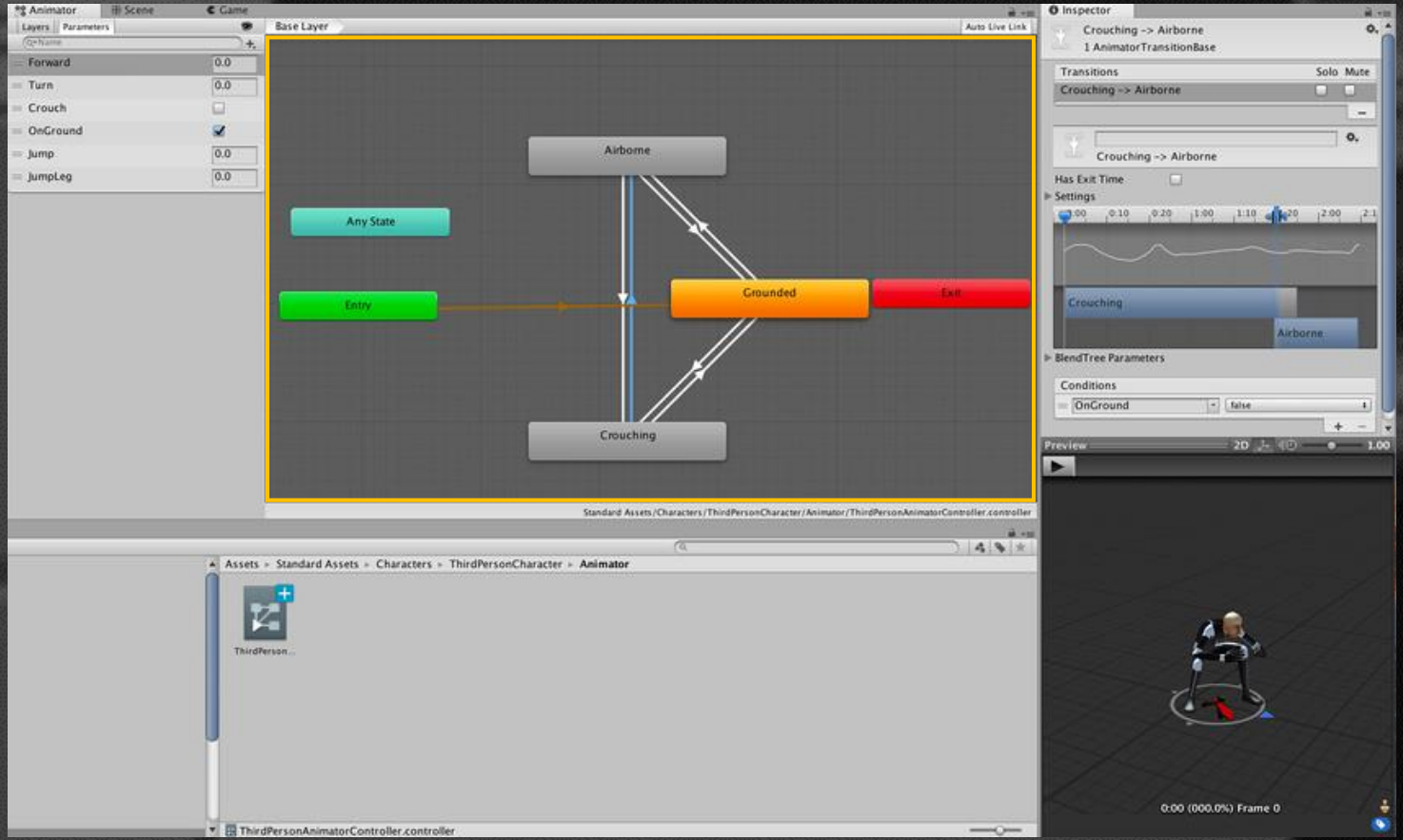
- Unity Animation System (a.k.a ‘Mecanim’) provides
  - Easy workflow and setup of animations for all elements of Unity including objects, characters, and properties.
  - Support for imported animation clips and animation created within Unity
  - Humanoid animation retargeting - the ability to apply animations from one character model onto another.
  - Simplified workflow for aligning animation clips.
  - Convenient preview of animation clips, transitions and interactions between them. This allows animators to work more independently of programmers, prototype and preview their animations before gameplay code is hooked in.

Animation

Animation

- Unity Animation System (a.k.a ‘Mecanim’) provides
  - Management of complex interactions between animations with a visual programming tool.
  - Animating different body parts with different logic.
  - Layering and masking features

Animation



이미지 출처 :https://docs.unity3d.com/Manual/AnimationOverview.html

Animation Workflow

- Unity's animation system is based on the concept of Animation Clips, which contain information about how certain objects should change their position, rotation, or other properties over time.
- Each clip can be thought of as a **single linear recording**.
- Animation clips **from external sources** are created by artists or animators with 3rd party tools such as Autodesk® 3ds Max® or Autodesk® Maya®, or come from motion capture studios or other sources.
- Animation clips are then organized into a structured flowchart-like system called an Animator Controller.

Animation Workflow

Animation Workflow

- The Animator Controller acts as a "State Machine" which keeps track of which clip should currently be playing, and when the animations should change or blend together.
- A very **simple** Animator Controller might only contain 1 or 2 clips, for example to control a powerup spinning and bouncing, or to animate a door opening and closing at the correct time.
- A more **advanced** Animator Controller might contain dozens of humanoid animations for all the main character's actions, and might blend between multiple clips at the same time to provide a fluid motion as the player moves around the scene.

Animation Workflow

Animation Workflow

- Unity's Animation system also has numerous special features for handling **humanoid characters** which give you the ability to **retarget humanoid animation** from any source (for example : motion capture; the Asset Store; or some other third-party animation library) to your own character model, as well as **adjusting muscle definitions**.
- These special features are enabled by Unity's Avatar system, where humanoid characters are mapped to a common internal format.
- Each of these pieces - the Animation Clips, the Animator Controller, and the Avatar, are brought together on a GameObject via the Animator Component.

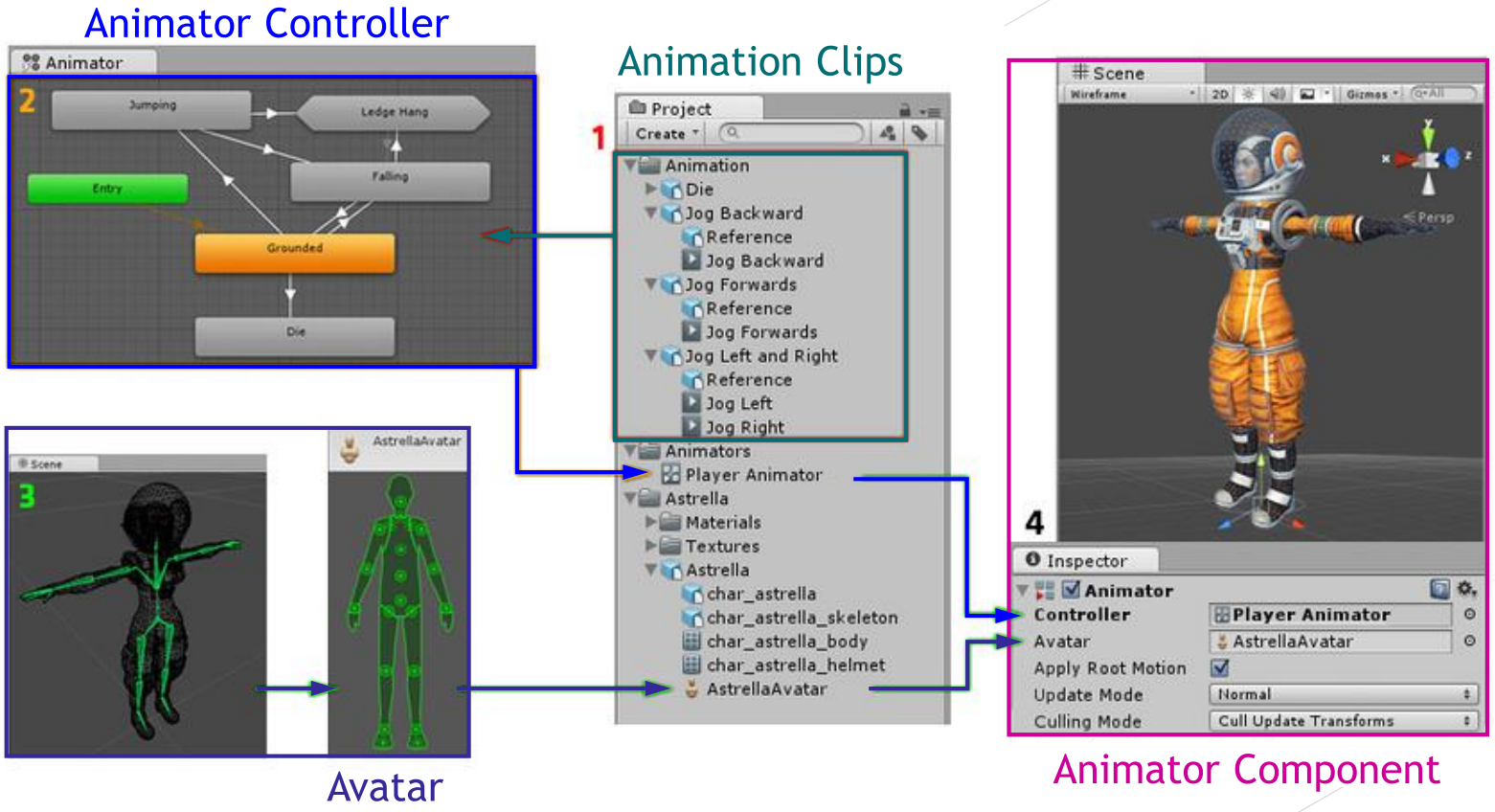
Animation Workflow

Animation Workflow

- The Animator Component has a reference to an Animator Controller, and (if required) the Avatar for this model.
- The Animator Controller, in turn, contains the references to the Animation Clips it uses.

Animation Workflow

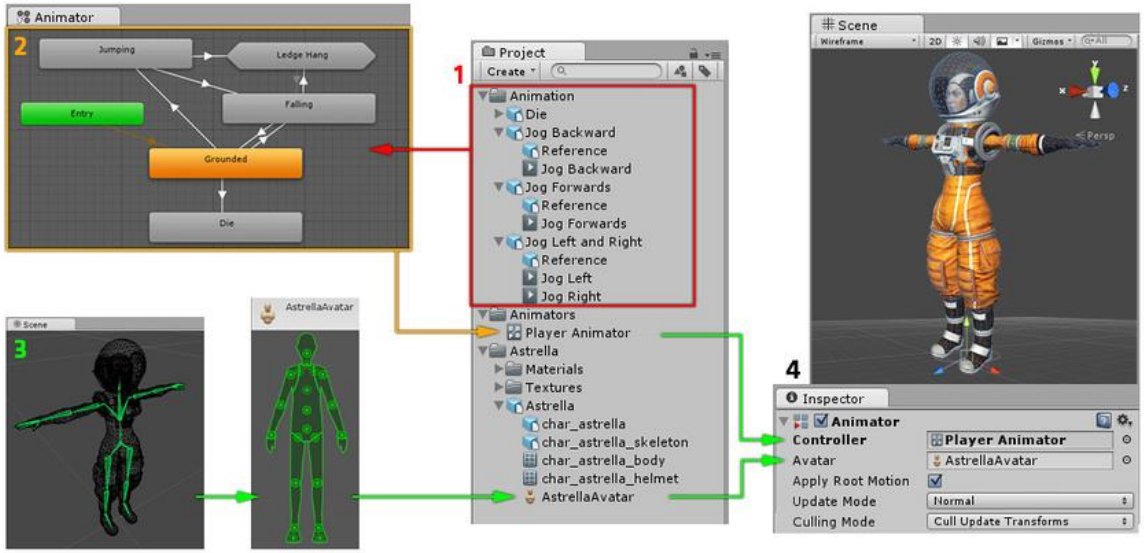
Animation Workflow



이미지 출처 :https://docs.unity3d.com/Manual/AnimationOverview.html

Animation Workflow

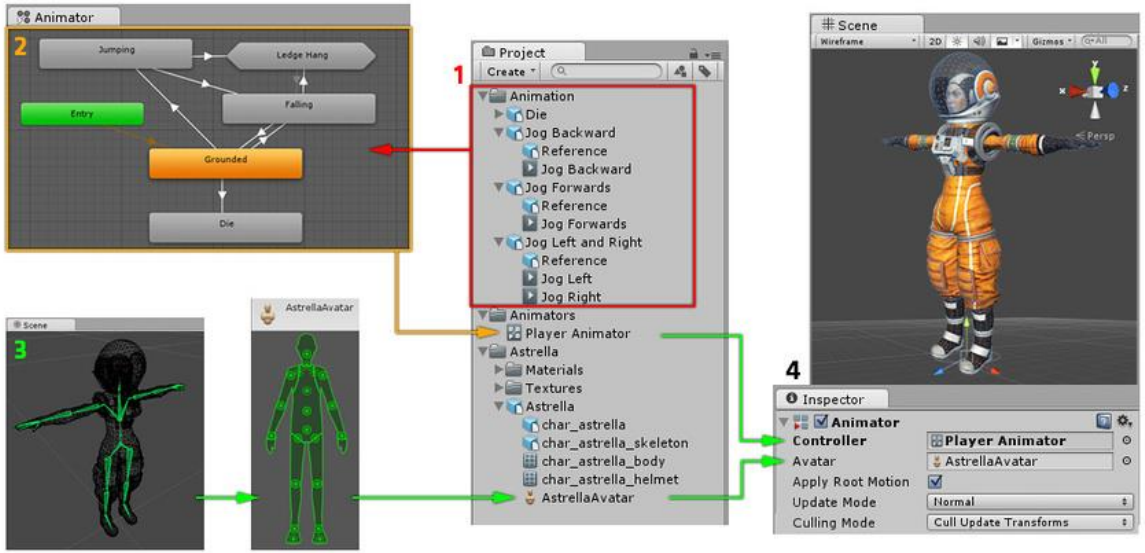
1. Animation clips **are imported** from an external source or created within Unity. In this example, they are imported motion captured humanoid animations.



이미지 출처 :https://docs.unity3d.com/Manual/AnimationOverview.html

Animation Workflow

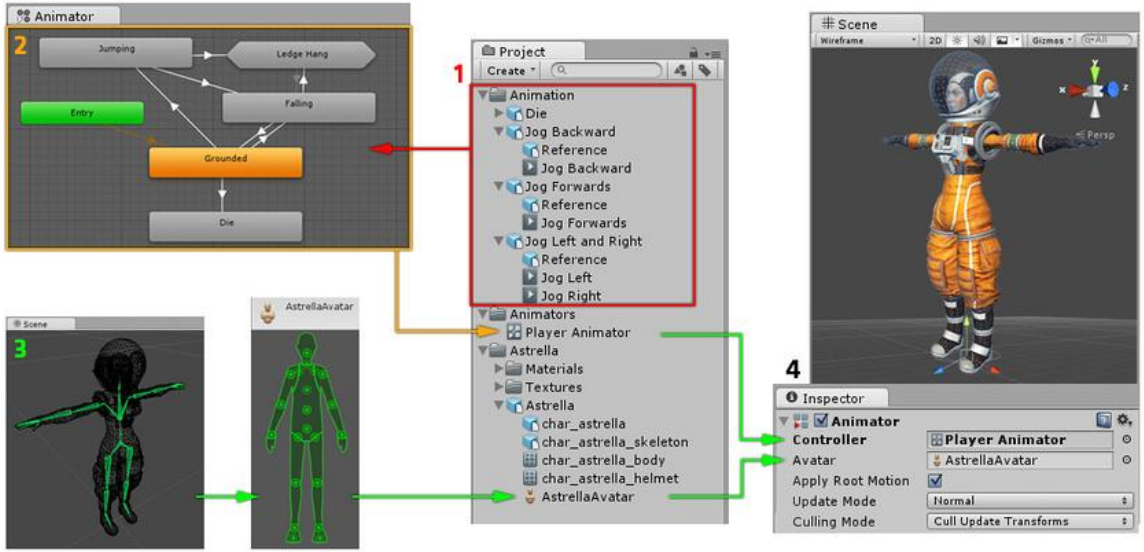
2 The animation clips **are placed and arranged in an** Animator Controller.
This shows a view of an Animator Controller in the Animator window.
The States (which may represent animations or nested sub-state machines) appear as nodes connected by lines. This Animator Controller exists as an asset in the **Project window**.



이미지 출처 :https://docs.unity3d.com/Manual/AnimationOverview.html

Animation Workflow

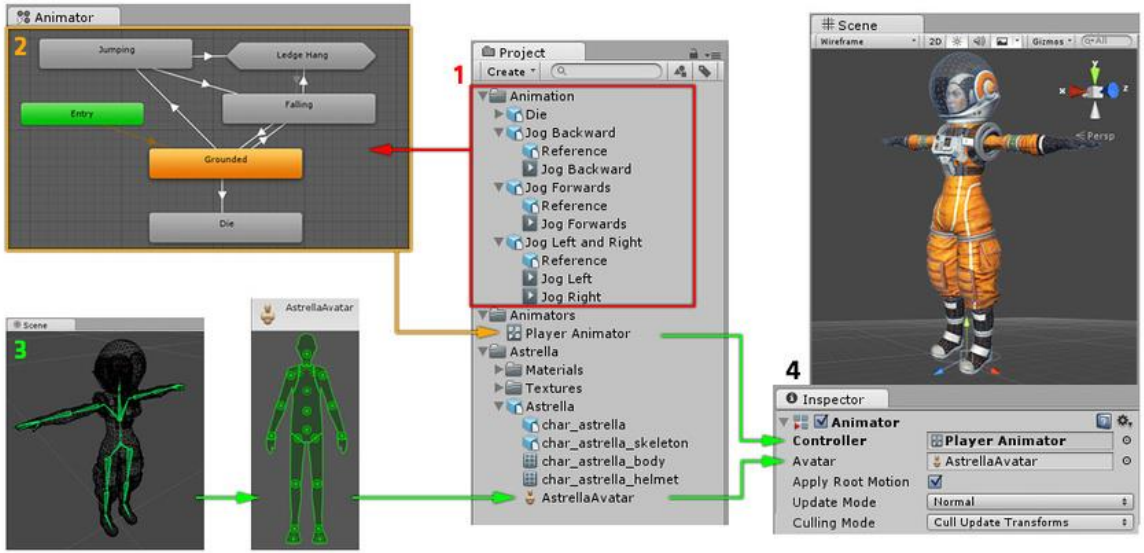
3 **The rigged character model** (in this case, the astronaut “Astrella”) has a specific configuration of **bones** which are mapped to Unity’s common Avatar format. This mapping is stored as an Avatar asset as part of the imported character model, and also appears in the Project window.



이미지 출처 :https://docs.unity3d.com/Manual/AnimationOverview.html

Animation Workflow

4 When animating the character model, it has an Animator Component attached. In the Inspector view, you can see the Animator Component which has both the Animator Controller and the Avatar assigned. The animator uses these together to animate the model. **The Avatar reference is only necessary when animating a humanoid character**. For other types of animation, only an Animator Controller is required.



이미지 출처 :https://docs.unity3d.com/Manual/AnimationOverview.html

Animation Clips

Animation Clip Asset

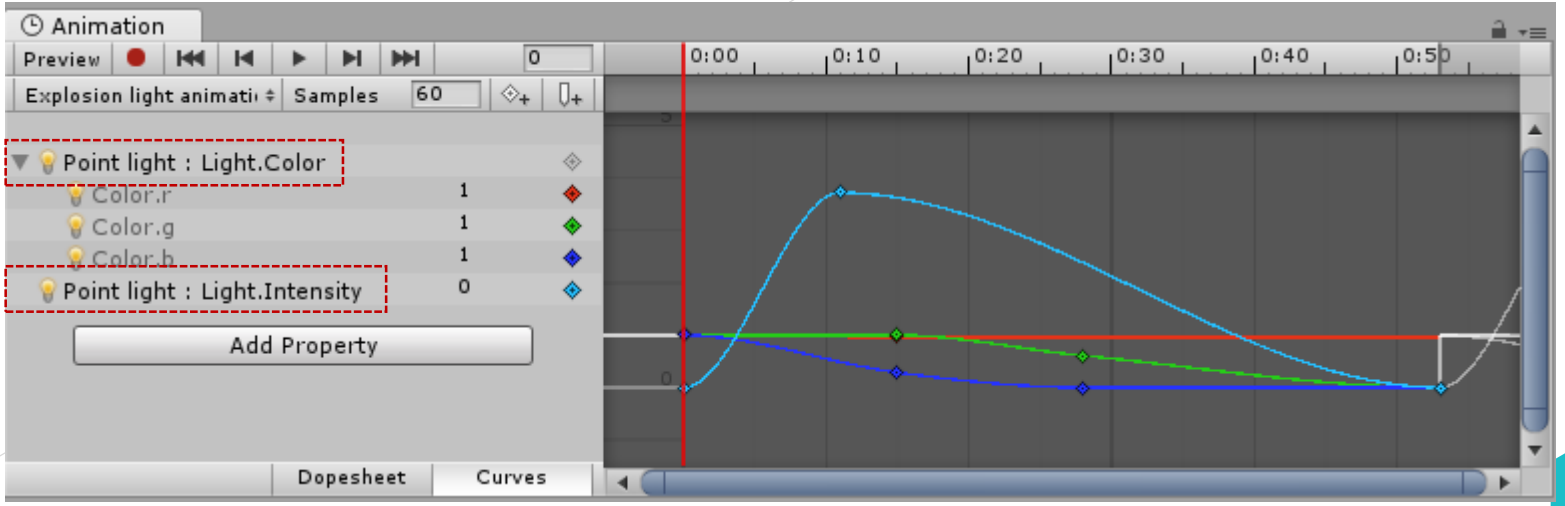
- Animation Clip Assets are one of the core elements to Unity's animation system.
- Animation from External Sources
  - Humanoid animations captured at a motion capture studio
  - Animations created from scratch by an artist in an external 3D application (such as Autodesk® 3ds Max® or Autodesk® Maya®)
  - Animation sets from 3rd-party libraries (e.g., from Unity's asset store)
  - Multiple clips cut and sliced from a single imported timeline.

Animation Clips

Animation Clips

Animation Clips

- Animation Created and Edited Within Unity



이미지 출처 :https://docs.unity3d.com/Manual/AnimationClips.html

Animation Clips

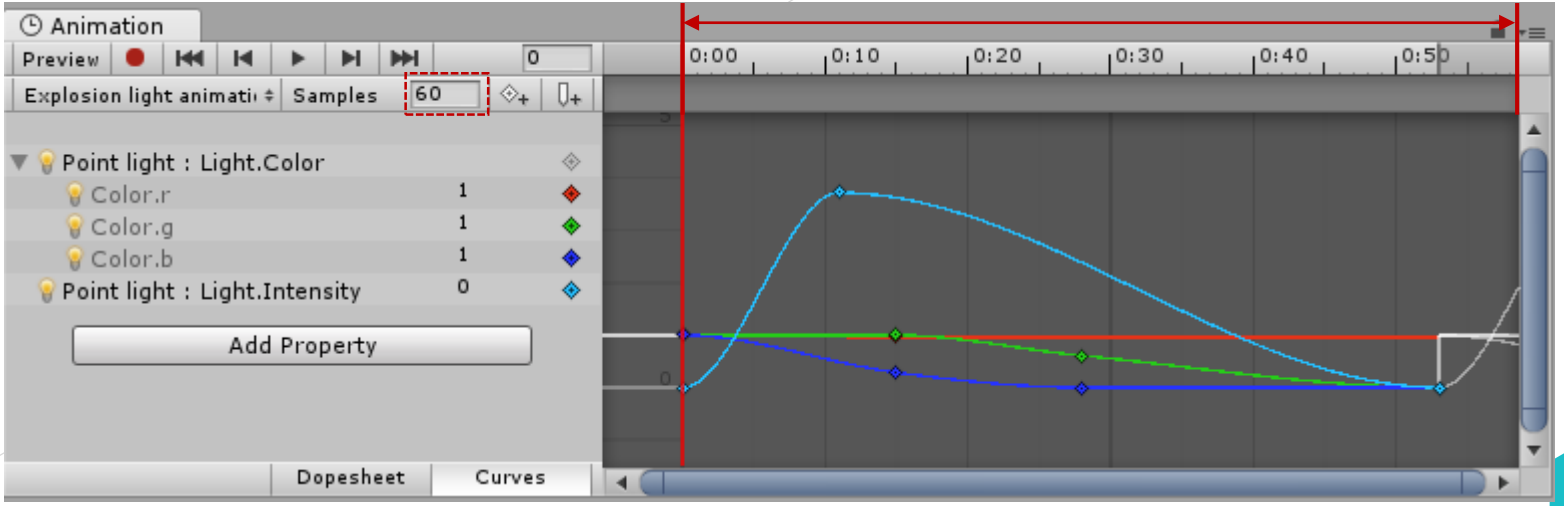
- Animation Created and Edited Within Unity
  - The position, rotation and scale of GameObjects
  - Component properties such as material color, the intensity of a light, the volume of a sound
  - Properties within your own scripts including float, integer, enum, vector and Boolean variables
  - The timing of calling functions within your own scripts

Animation Clips

Animation Clips

Animation Clips

Animation Created and Edited Within Unity

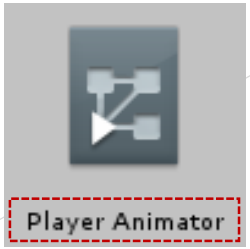


이미지 출처 :https://docs.unity3d.com/Manual/AnimationClips.html

Animator Controller

Animator Controller Asset

- An Animator Controller allows you to arrange and maintain a set of animations for a character or other animated Game Object.
- The controller has references to the animation clips used within it, and manages the various animation states and the transitions between them using a so-called State Machine.



이미지 출처 :https://docs.unity3d.com/Manual/Animator.html

Animator Controller

Animator Controller Asset

- When you have an animation clips ready to use, you need to use an Animator Controller to bring them together.
- An Animator Controller asset is created within Unity and allows you to maintain a set of animations for a character or object.
- Animator Controller assets are created from the Assets menu, or from the Create menu in the Project window.

Animator Controller

Animator Controller

Animator Controller Asset

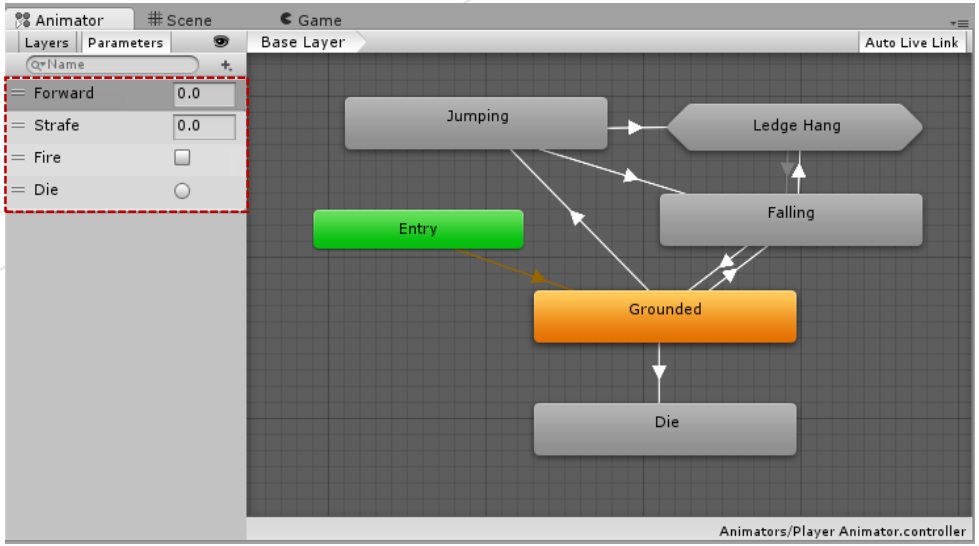
- In most situations, it is normal to have multiple animations and switch between them when certain game conditions occur.
- For example, you could switch from a walk animation to a jump whenever the spacebar is pressed.
- However even if you just have a single animation clip you still need to place it into an animator controller to use it on a Game Object.

Animator Controller

Animator Controller

Animator Controller Asset

- The controller manages the various animation states and the transitions between them using a so-called State Machine.
- The structure of the Animator Controller can be created, viewed and modified in the Animator Window.

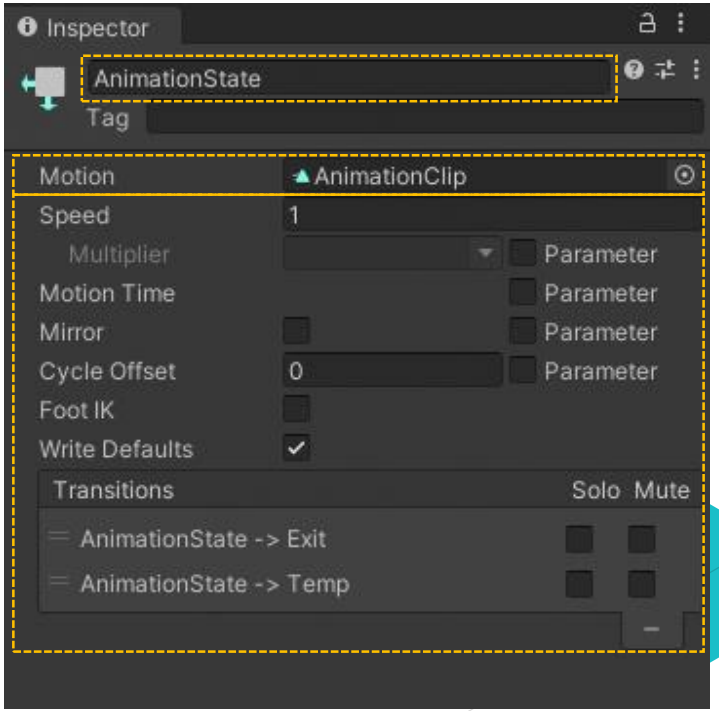


이미지 출처 :https://docs.unity3d.com/Manual/Animator.html

Animator Controller

Animation States

- Each state contains an animation sequence (or blend tree) that plays when the character is in that state.
- Select the state in the Animator Controller, to view the properties for the state in the Inspector window.



이미지 출처 :https://docs.unity3d.com/Manual/Animator.html

Animator Controller

Animation States

Property	Description
Motion	The animation clip or blend tree assigned to this state.
Speed	The default speed of the motion for this state. Enable Parameter to modify the speed with a custom value from a script. For example, you can multiply the speed with a custom value to decelerate or accelerate the play speed.
Motion Time	The time used to play the motion for this state. Enable Parameter to control the motion time with a custom value from a script.
Mirror	This property only applies to states with humanoid animation. Enable to mirror the animation for this state. Enable Parameter to enable or disable mirroring from a script.
Cycle Offset	The offset added to the state time of the motion. This offset does not affect the Motion Time. Enable Parameter to specify the Cycle Offset from a script.

Animator Controller

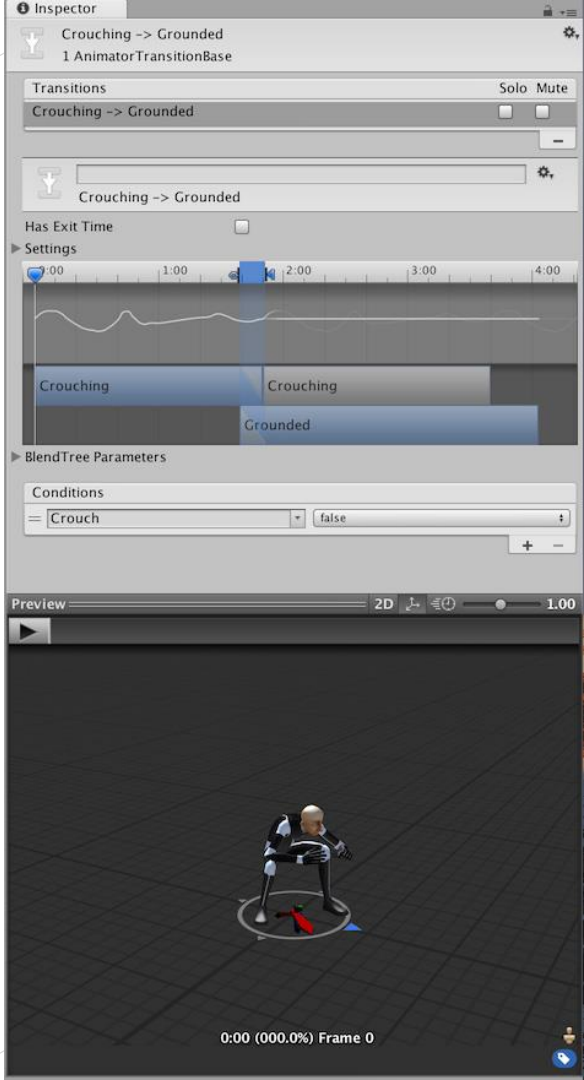
Animation States

Property	Description
Foot IK	This property only applies to states with humanoid animation. Enable to respect Foot IK for this state.
Write Defaults	Whether the AnimatorStates writes the default values for properties that are not animated by its motion.
Transitions	The list of transitions originating from this state.

Animator Controller

Animation Transitions

- Animation transitions allow the state machine to switch or blend from one animation state to another.
- Transitions define not only how long the blend between states should take, but also under what conditions they should activate.
- You can set up a transition to occur only when certain conditions are true.
- To set up these conditions, specify values of parameters in the Animator Controller.



이미지 출처 :https://docs.unity3d.com/Manual/class-Transition.html

Animator Controller

Property	Function
Has Exit Time	Exit Time is a special transition that doesn't rely on a parameter. Instead, it relies on the normalized time of the state. Check to make the transition happen at the specific time specified in Exit Time.
Settings	Fold-out menu containing detailed transition settings as below.
Exit Time	If Has Exit Time is checked, this value represents the exact time at which the transition can take effect. This is represented in normalized time (for example, an exit time of 0.75 means that on the first frame where 75% of the animation has played, the Exit Time condition is true). On the next frame, the condition is false. For looped animations, transitions with exit times smaller than 1 are evaluated every loop, so you can use this to time your transition with the proper timing in the animation every loop. Transitions with an Exit Time greater than 1 are evaluated only once, so they can be used to exit at a specific time after a fixed number of loops. For example, a transition with an exit time of 3.5 are evaluated once, after three and a half loops.

Animator Controller

Property	Function
Fixed Duration	If the Fixed Duration box is checked, the transition time is interpreted in seconds. If the Fixed Duration box is not checked, the transition time is interpreted as a fraction of the normalized time of the source state.
Transition Duration	The duration of the transition, in normalized time or seconds depending on the Fixed Duration mode, relative to the current state's duration. This is visualized in the transition graph as the portion between the two blue markers.
Transition Offset	The offset of the time to begin playing in the destination state which is transitioned to. For example, a value of 0.5 means the target state begins playing at 50% of the way through its own timeline.
Interruption Source	Use this to control the circumstances under which this transition may be interrupted
Ordered Interruption	Determines whether the current transition can be interrupted by other transitions independently of their order (see Transition interruption below).

Animator Controller

Property	Function
Conditions	A transition can have a single condition, multiple conditions, or no conditions at all. If your transition has no conditions, the Unity Editor only considers the Exit Time, and the transition occurs when the exit time is reached. If your transition has one or more conditions, the conditions must all be met before the transition is triggered. A condition consists of - An event parameter (the value considered in the condition). - A conditional predicate (if needed, for example, ‘less than’ or ‘greater than’ for floats). - A parameter value (if needed). If you have Has Exit Time selected for the transition and have one or more conditions, note that the Unity Editor considers whether the conditions are true after the Exit Time. This allows you to ensure that your transition occurs during a certain portion of the animation.

State Machine Basics

State Machine

- The basic idea is that a character is engaged in some particular kind of action at any given time.
- The actions available will depend on the type of gameplay but typical actions include things like idling, walking, running, jumping, etc.
- These actions are referred to as states, in the sense that the character is in a “state” where it is walking, idling or whatever.
- In general, the character will have restrictions on the next state it can go to rather than being able to switch immediately from any state to any other.
- For example, a running jump can only be taken when the character is already running and not when it is at a standstill, so it should never switch straight from the idle state to the running jump state.

State Machine Basics

State Machine Basics

State Machine

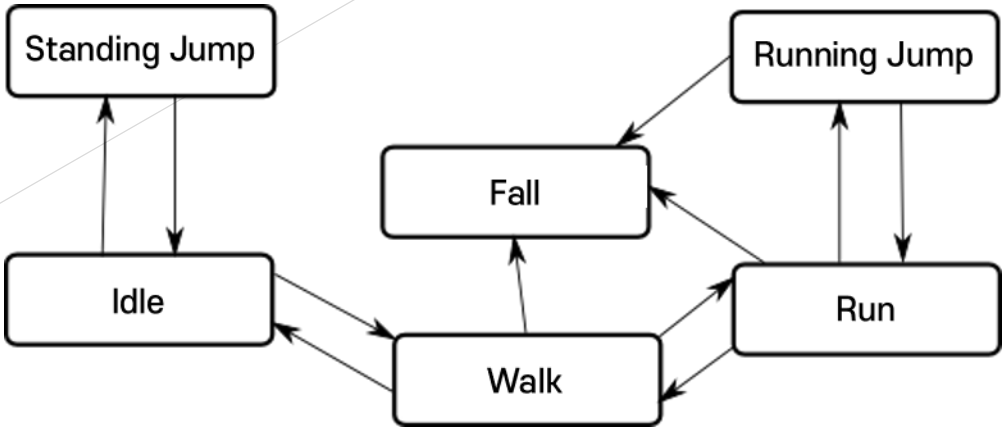
- The options for the next state that a character can enter from its current state are referred to as state transitions.
- Taken together, the set of states, the set of transitions and the variable to remember the current state form a state machine.
- The states and transitions of a state machine can be represented using a graph diagram, where the nodes represent the states and the arcs (arrows between nodes) represent the transitions.
- You can think of the current state as being a marker or highlight that is placed on one of the nodes and can then only jump to another node along one of the arrows.

State Machine Basics

State Machine Basics

State Machine

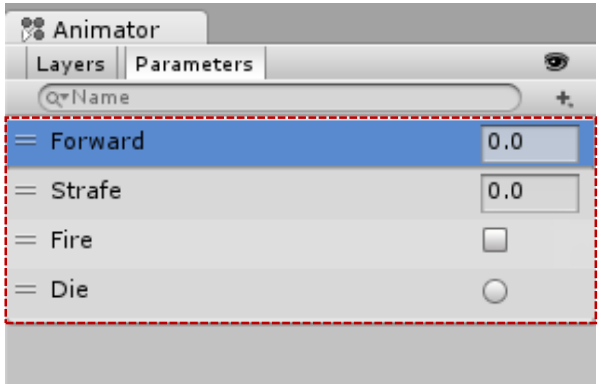
- Unity's Animation State Machines provide a way to overview all of the animation clips related to a particular character and allow various events in the game (e.g., user input) to trigger different animations.
- State Machines consist of States, Transitions and Events and smaller Sub-State Machines can be used as components in larger machines.



이미지 출처 :https://docs.unity3d.com/2021.3/Documentation/Manual/StateMachineBasics.html

Animation Parameters

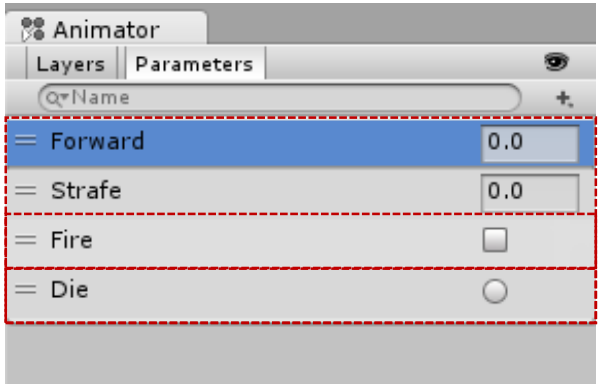
- Animation Parameters are variables that are defined within an Animator Controller that can be accessed and assigned values from scripts.
- This is how a script can control or affect the flow of the state machine.
- For example, the value of a parameter can be updated by an animation curve and then accessed from a script so that, say, the pitch of a sound effect can be varied as if it were a piece of animation.
- Likewise, a script can set parameter values to be picked up by Mecanim.
- For example, a script can set a parameter to control a Blend Tree.



이미지 출처 :Unity

Animation Parameters

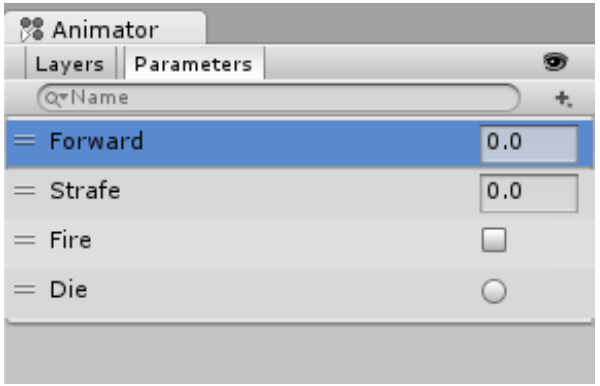
- Default parameter values can be set up using the Parameters section of the Animator window, selectable in the top right corner of the Animator window.
  - Integer : a whole number
  - Float : a number with a fractional part
  - Bool : true or false value
    (represented by a checkbox)
  - Trigger : a boolean parameter that is reset by the controller when consumed by a transition
    (represented by a circle button)



이미지 출처 :Unity

Animation Parameters

- Parameters can be assigned values from a script using functions in the Animator class
  - SetFloat
  - SetInteger
  - SetBool
  - SetTrigger
  - ResetTrigger



이미지 출처 :Unity

State Machine Transitions

- Each view in the animator window has an Entry and Exit node.
- The Entry node is used when transitioning into a state machine.
- The Entry node will be evaluated and will branch to the destination state according to the conditions set.
- In this way the Entry node can control which state the state machine begins in, by evaluating the state of your parameters when the state machine begins.
- Because state machines always have a default state, there will always be a default transition branching from the Entry node to the default state.
- You can then add additional transitions from the Entry node to other states, to control whether the state machine should begin in a different state.
- The Exit node is used to indicate that a state machine should exit.

State Machine Transitions

State Machine Behaviours

- A State Machine Behaviour is a special class of script.
- You can attach a StateMachineBehaviour script to an individual state within a state machine. It will execute when the state machine enters, exits or remains within a particular state.
- A few examples for the use of this feature might be to
  - Play sounds as states are entered or exited
  - Perform certain tests/detections only when in appropriate states
  - Activate and control special effects associated with specific states

State Machine Behaviours

State Machine Behaviours

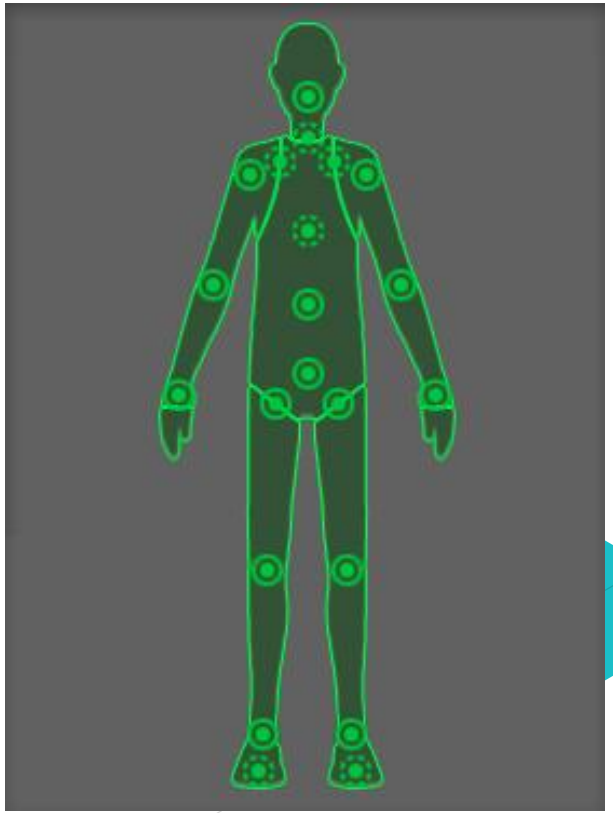
- You can select a state in your state machine, and then in the inspector use the “Add Behaviour” button to select an existing StateMachineBehaviour or create a new one.
- State Machine Behaviour scripts have access to a number of events that are called when the Animator enters, updates and exits different states (or sub-state machines). There are also events which allow you to handle the Root motion and Inverse Kinematics calls.

State Machine Behaviours

Avatar

Avatar

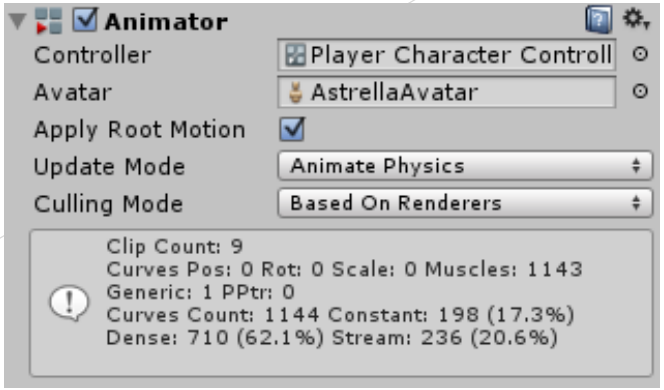
- Unity's Animation System has special features for working with **humanoid characters**.
- The Avatar system is how Unity identifies that a particular animated model is **humanoid** in layout, and which parts of the model correspond to the legs, arms, head and body.
- It is possible to map animations from one humanoid character to another, allowing **retargeting** and **inverse kinematics (IK)**.



이미지 출처 :https://docs.unity3d.com/Manual/AvatarCreationandSetup.html

Animator Component

- The Animator component assigns **animation** to a GameObject.
- The Animator component requires a reference to an Animator Controller which defines which animation clips to use, and controls when and how to blend and transition between them.
- If the GameObject is a **humanoid character** with an Avatar definition, the Avatar should also be assigned in this component.



이미지 출처 :https://docs.unity3d.com/2021.3/Documentation/Manual/class-Animator.html

Animator Component

Property	Function
Controller	The animator controller attached to this character.
Avatar	The Avatar for this character. (If the Animator is being used to animate a humanoid character)
Apply Root Motion	Select whether to control the character's position and rotation from the animation itself or from script.

Animator Component

Property		Function
Update Mode		This allows you to select when the Animator updates, and which timescale it should use.
	Normal	The Animator is updated in-sync with the Update call, and the animator's speed matches the current timescale. If the timescale is slowed, animations will slow down to match.
	Animate Physics	The animator is updated in-sync with the FixedUpdate call (i.e. in lock-step with the physics system). You should use this mode if you are animating the motion of objects with physics interactions, such as characters which can push rigidbody objects around.
	Unscaled Time	The animator is updated in-sync with the Update call, but the animator's speed ignores the current timescale and animates at 100% speed regardless. This is useful for animating a GUI system at normal speed while using modified timescales for special effects or to pause gameplay.

Animator Component

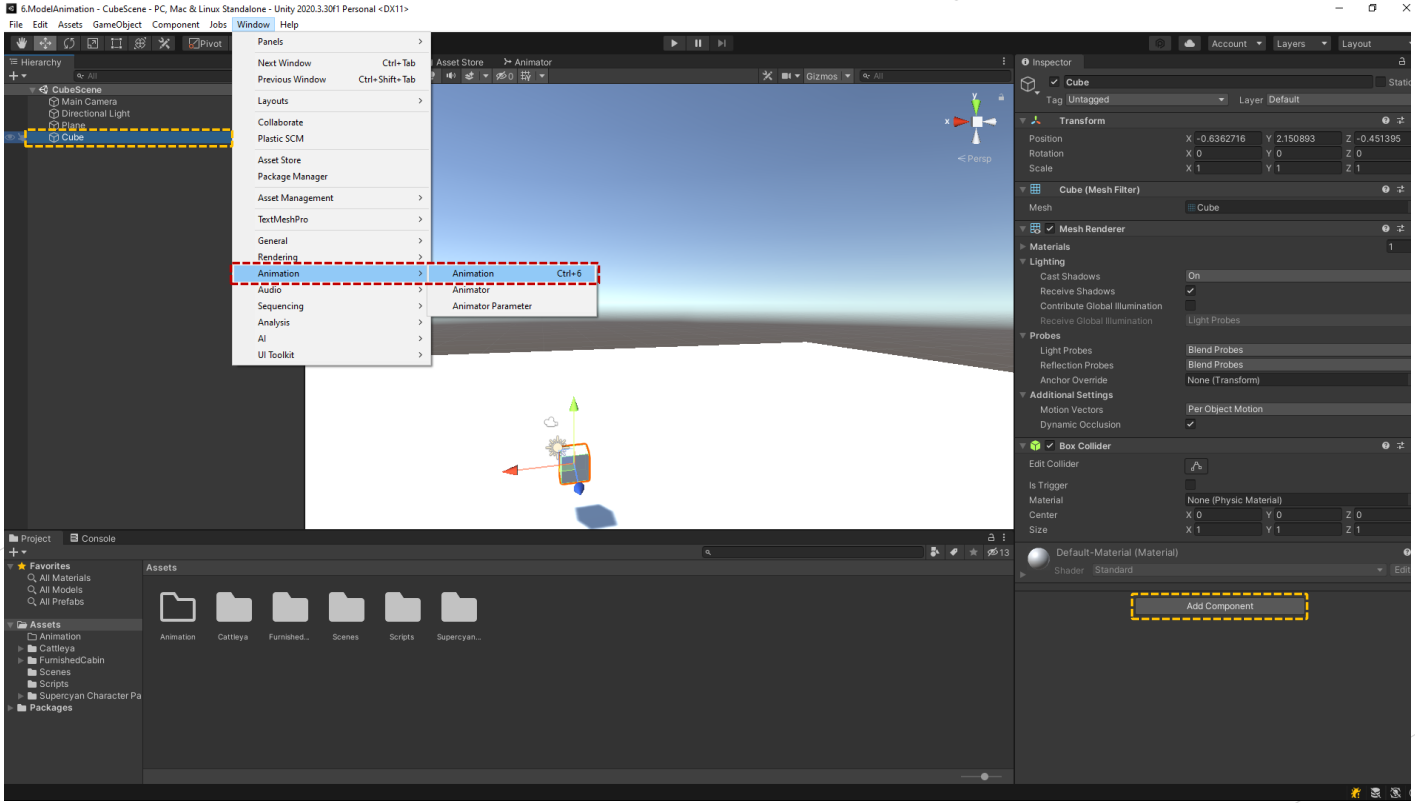
Property		Function
Culling Mode		Culling mode you can choose for animations.
	Always Animate	Always animate, don’t do culling even when offscreen.
	Cull Update Transforms	Retarget, IK and write of Transforms are disabled when renderers are not visible.
	Cull Completely	Animation is completely disabled when renderers are not visible.

2

Cube Animation

Cube Animation

Cube → add Component Animation

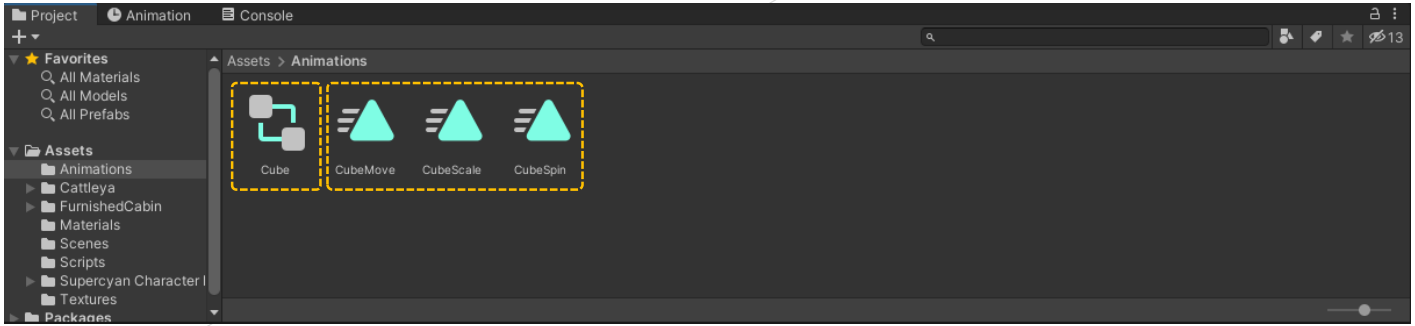


이미지 출처 :Unity

Cube Animation

Animation Clips in Animation Window

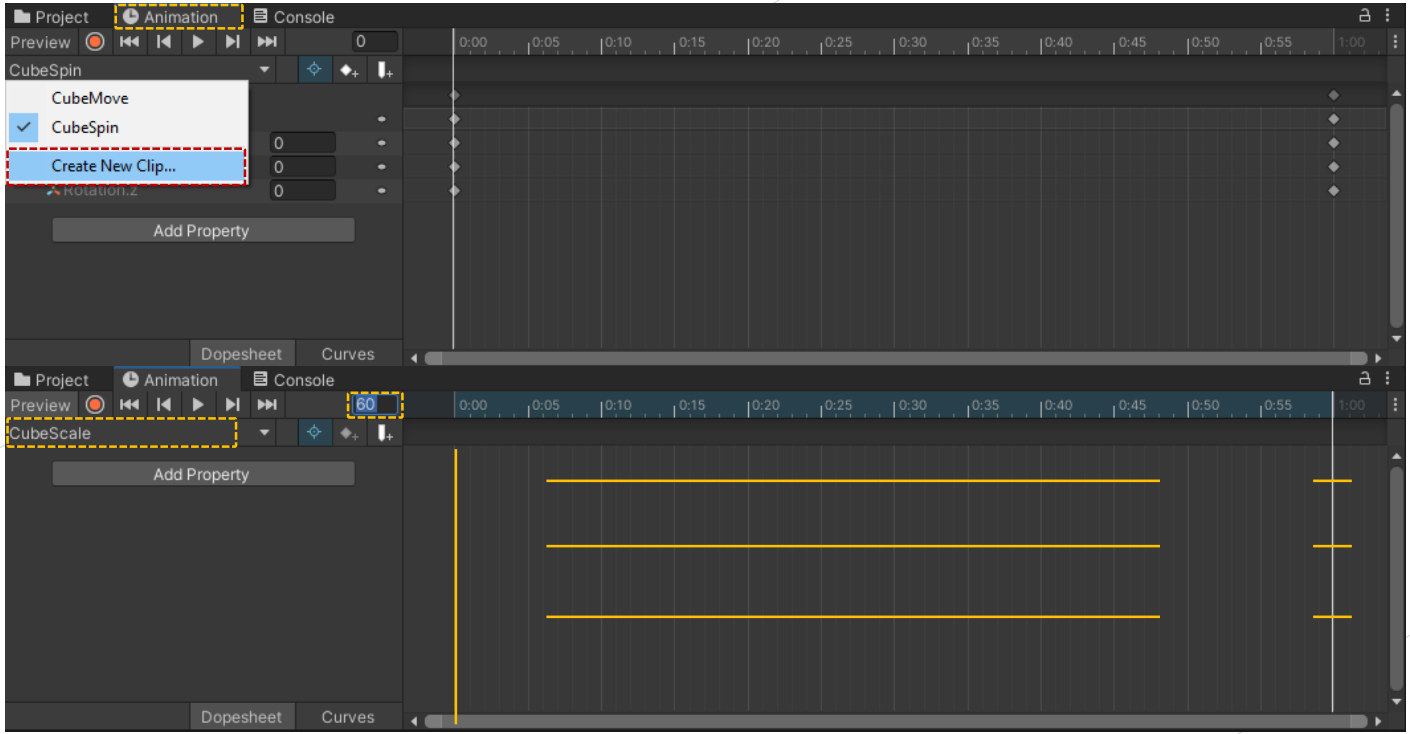
- Create CubeSpin.anim, CubeMove.anim, CubeScale.anim



이미지 출처 :Unity

Cube Animation

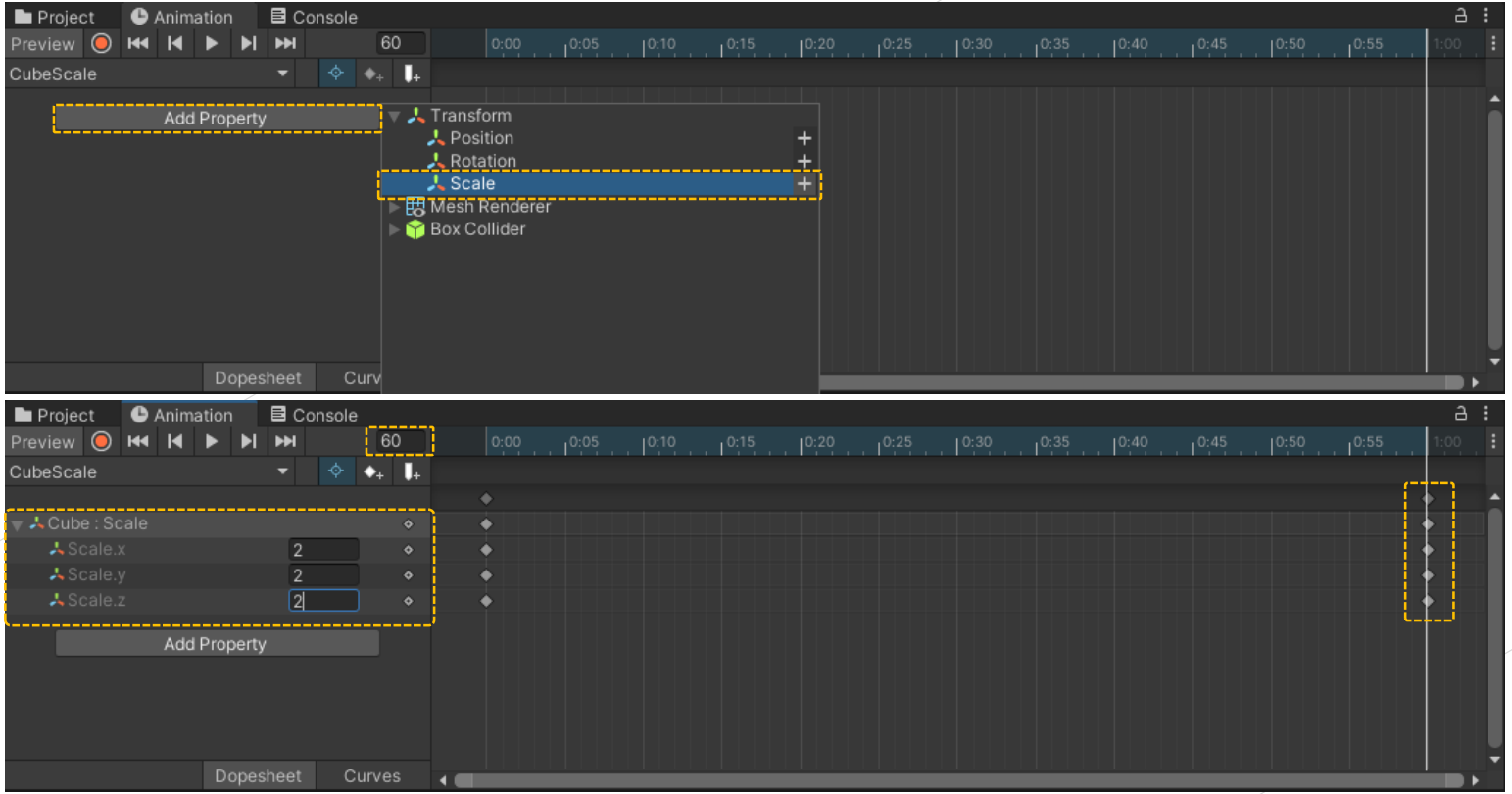
- Create Animation Clips in Animation Window
  - CubeSpin.anim, CubeMove.anim, CubeScale.anim



이미지 출처 :Unity

Cube Animation

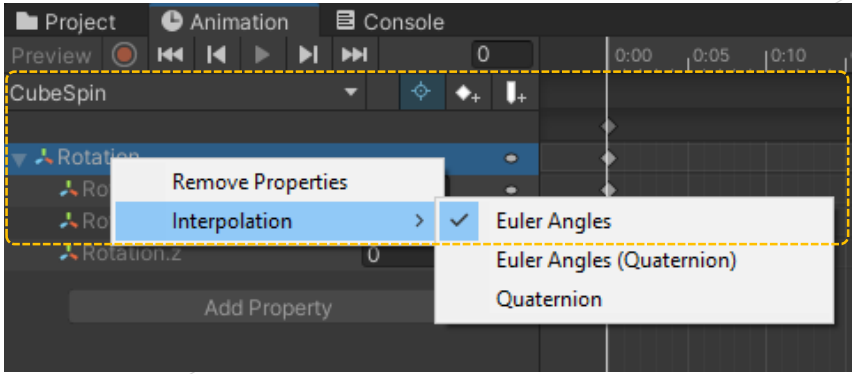
Create Animation Clips in Animation Window



이미지 출처 : Unity

Cube Animation

- You can use the Animation window to choose how Unity applies rotation to your GameObject.



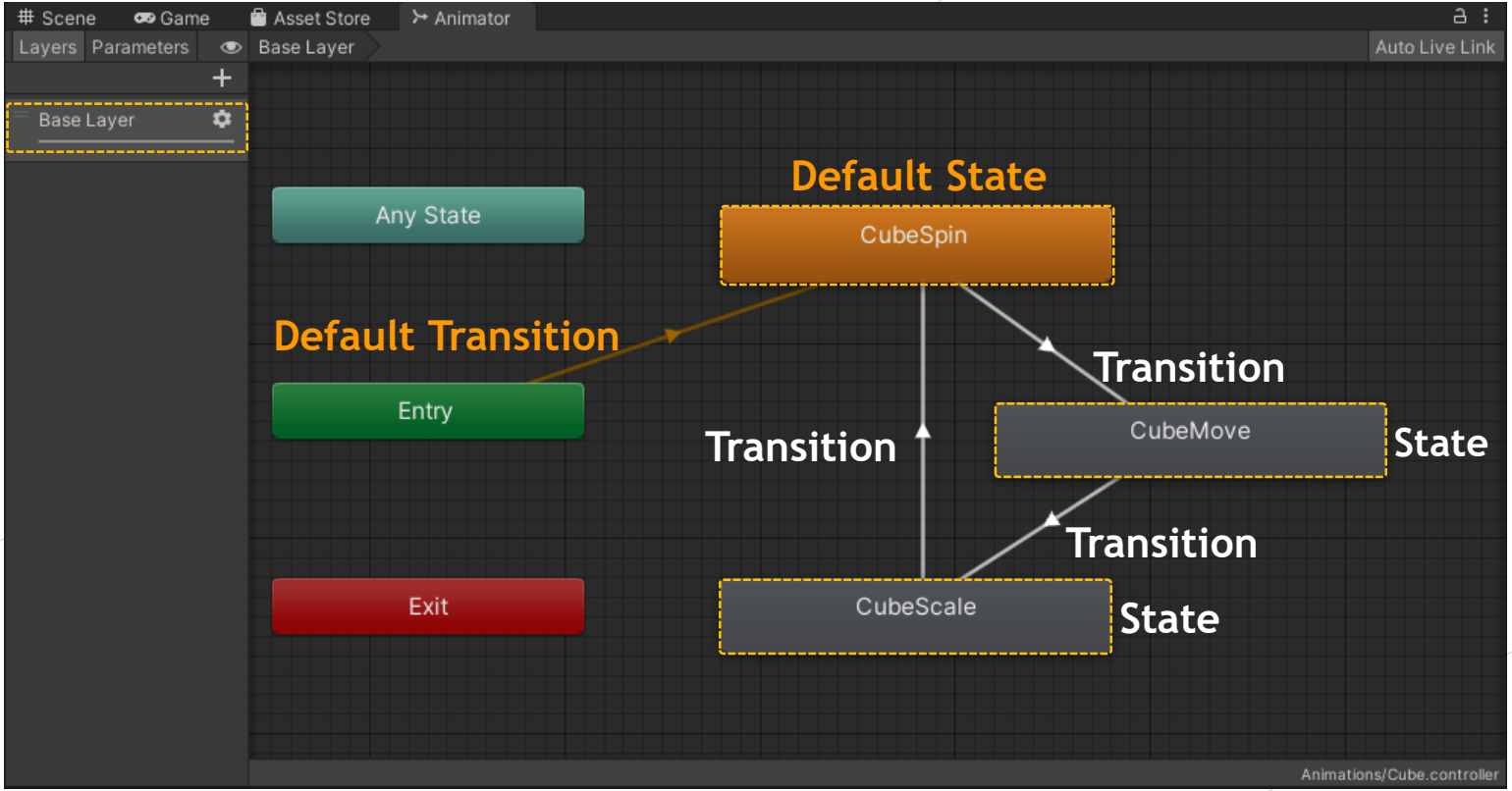
이미지 출처 :Unity

Cube Animation

- You can use the Animation window to choose how Unity applies rotation to your GameObject.
  - Unity offers three types of interpolation for your animations
    - Euler Angles interpolation applies the full range of motion to the GameObject specified by the angles you enter; if the rotation is greater than 360 degrees, the GameObject rotates fully before it stops at the correct orientation.
    - Euler Angles (Quaternion) interpolation uses the above interpolation method but bakes the information into a quaternion curve. This method uses more memory but results in a slightly faster runtime.
    - Quaternion interpolation rotates the GameObject across the shortest distance to a particular orientation. For example, regardless of whether the rotation value is 5 degrees or 365 degrees, the GameObject rotates 5 degrees.

Cube Animation

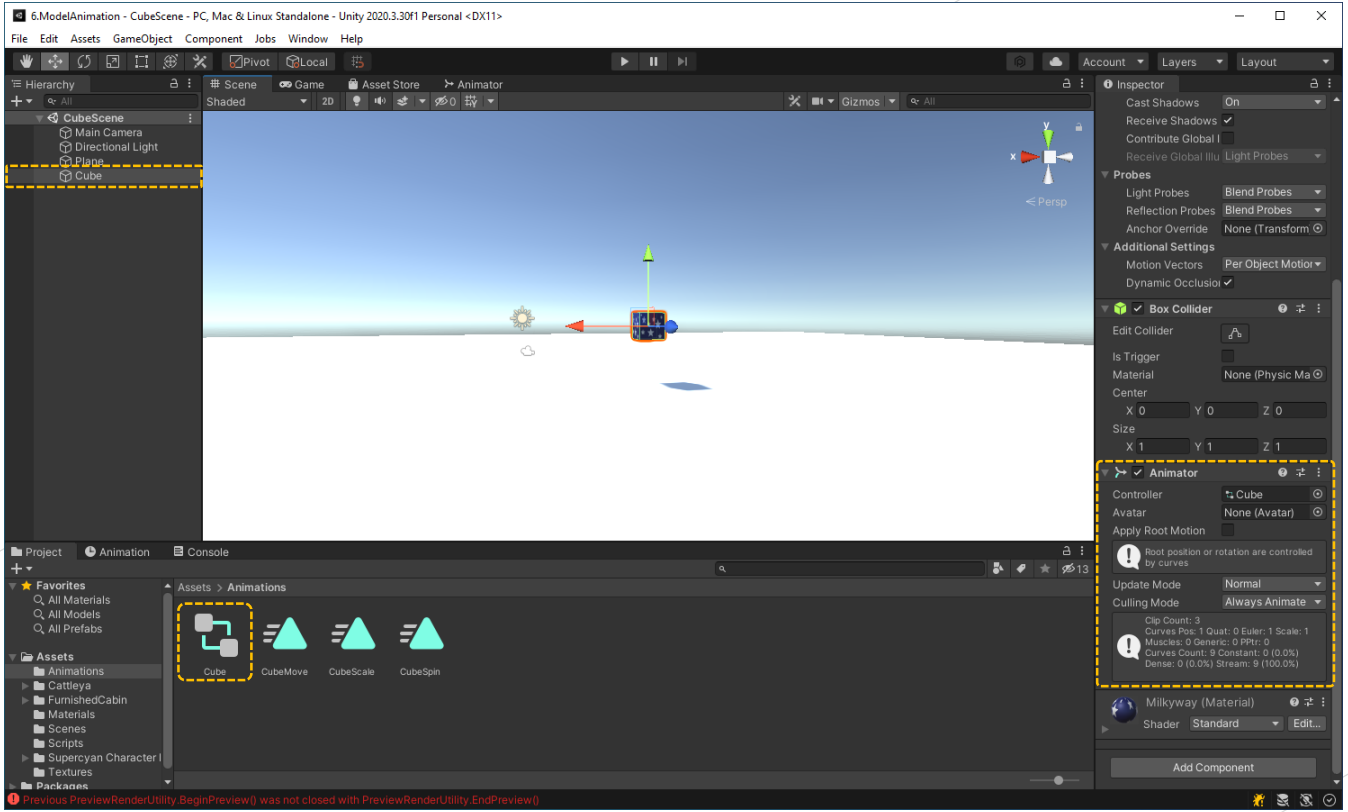
Animator Controller



이미지 출처 : Unity

Cube Animation

Animator Component

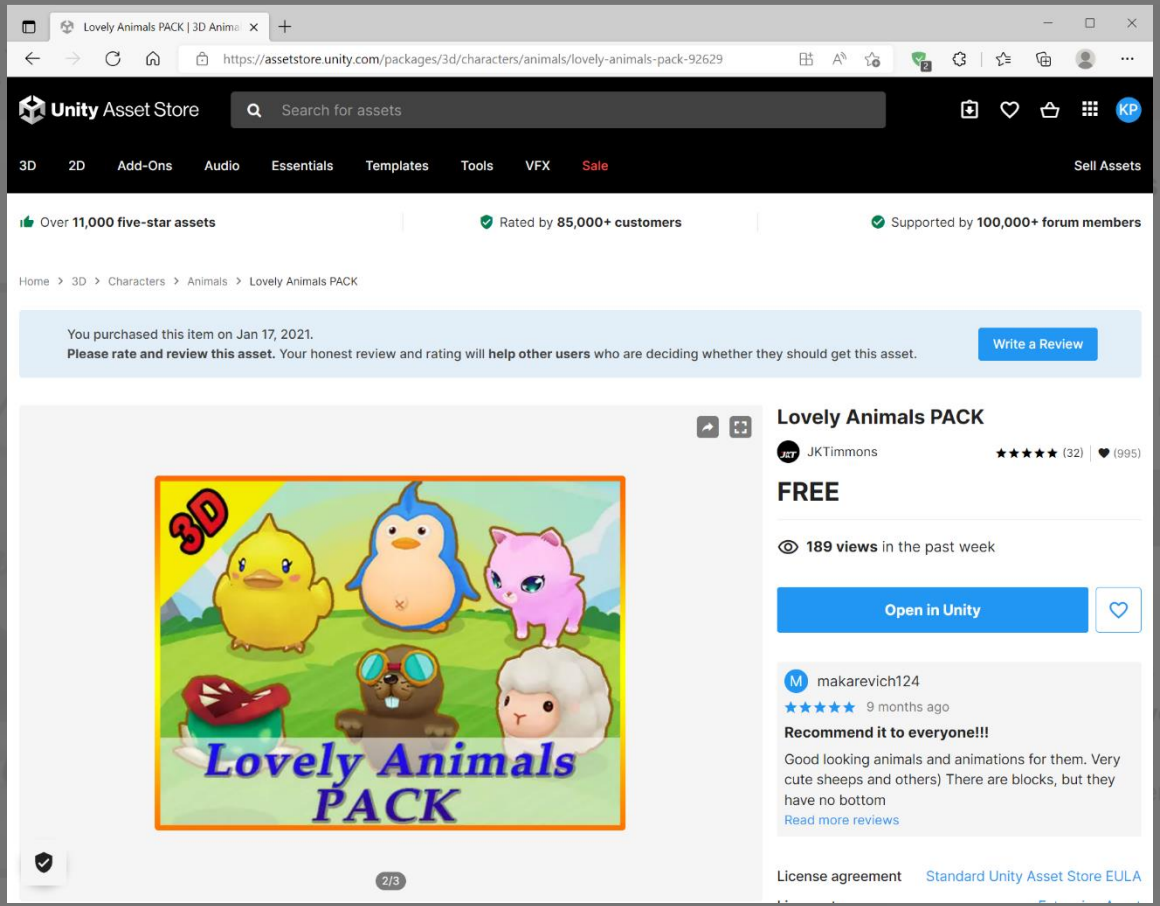


이미지 출처 :Unity

3

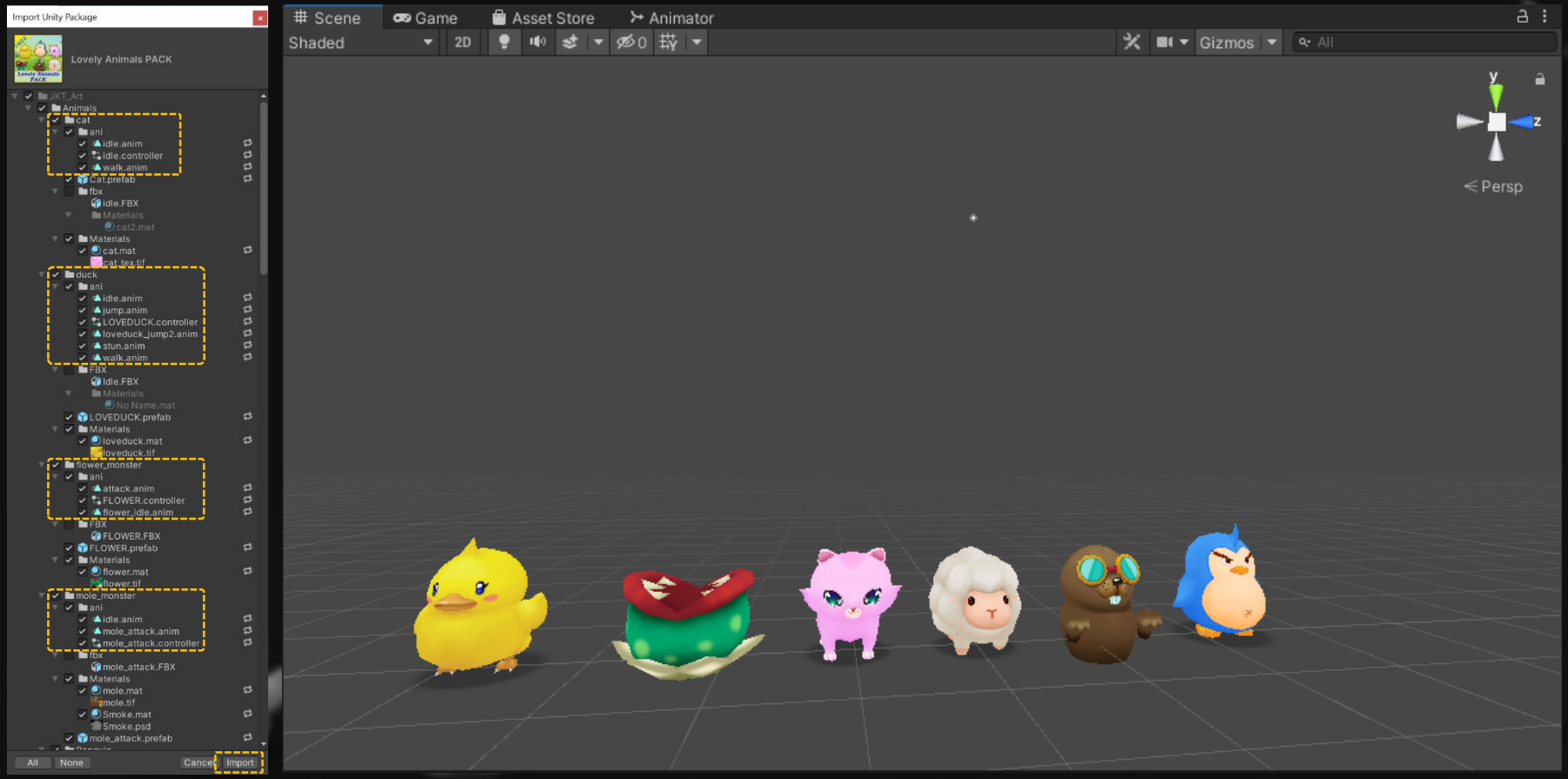
Character Model Animation

Asset Store



이미지 출처 :https://assetstore.unity.com/packages/3d/characters/animals/lovely-animals-pack-92629

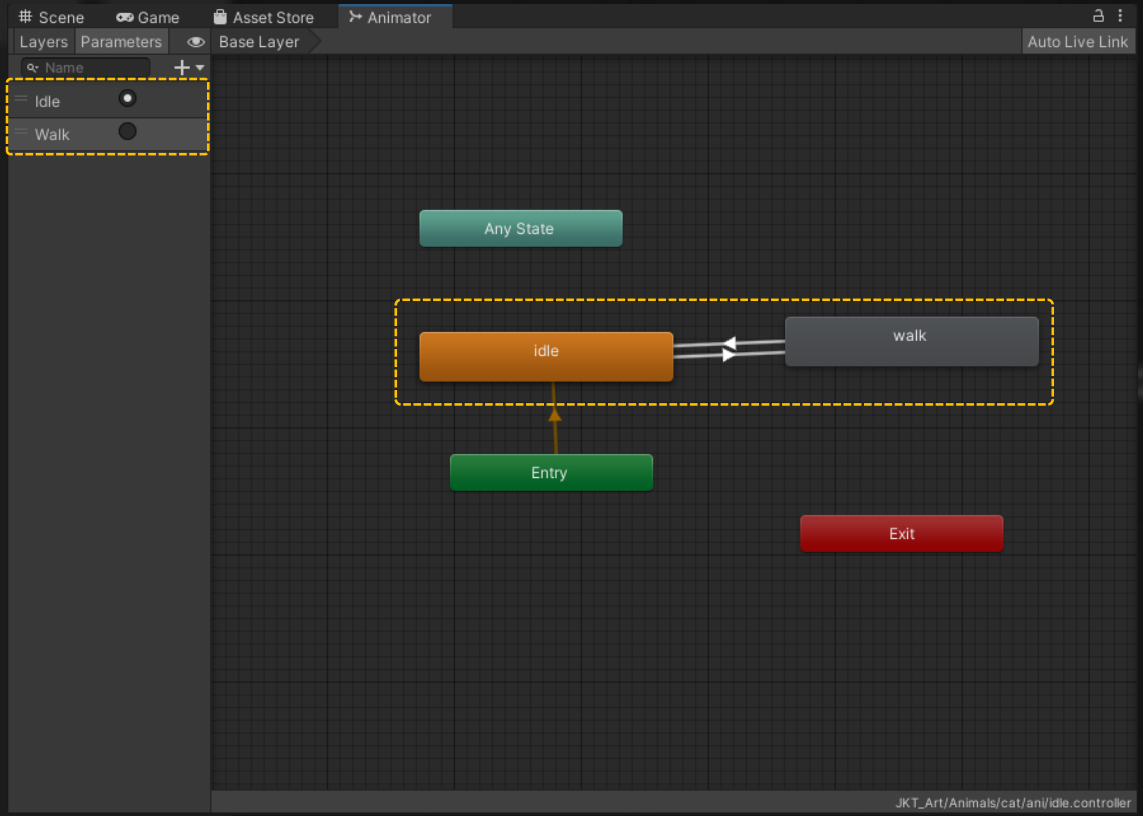
Import Unity Package

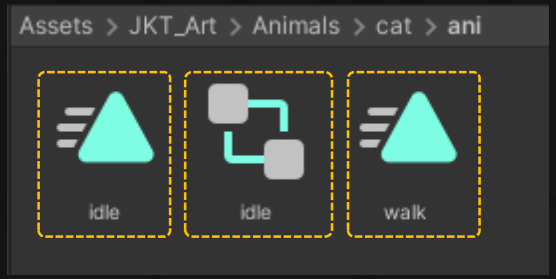


이미지 출처 :Unity

Animator Controller

Cat States : Idle, Walk

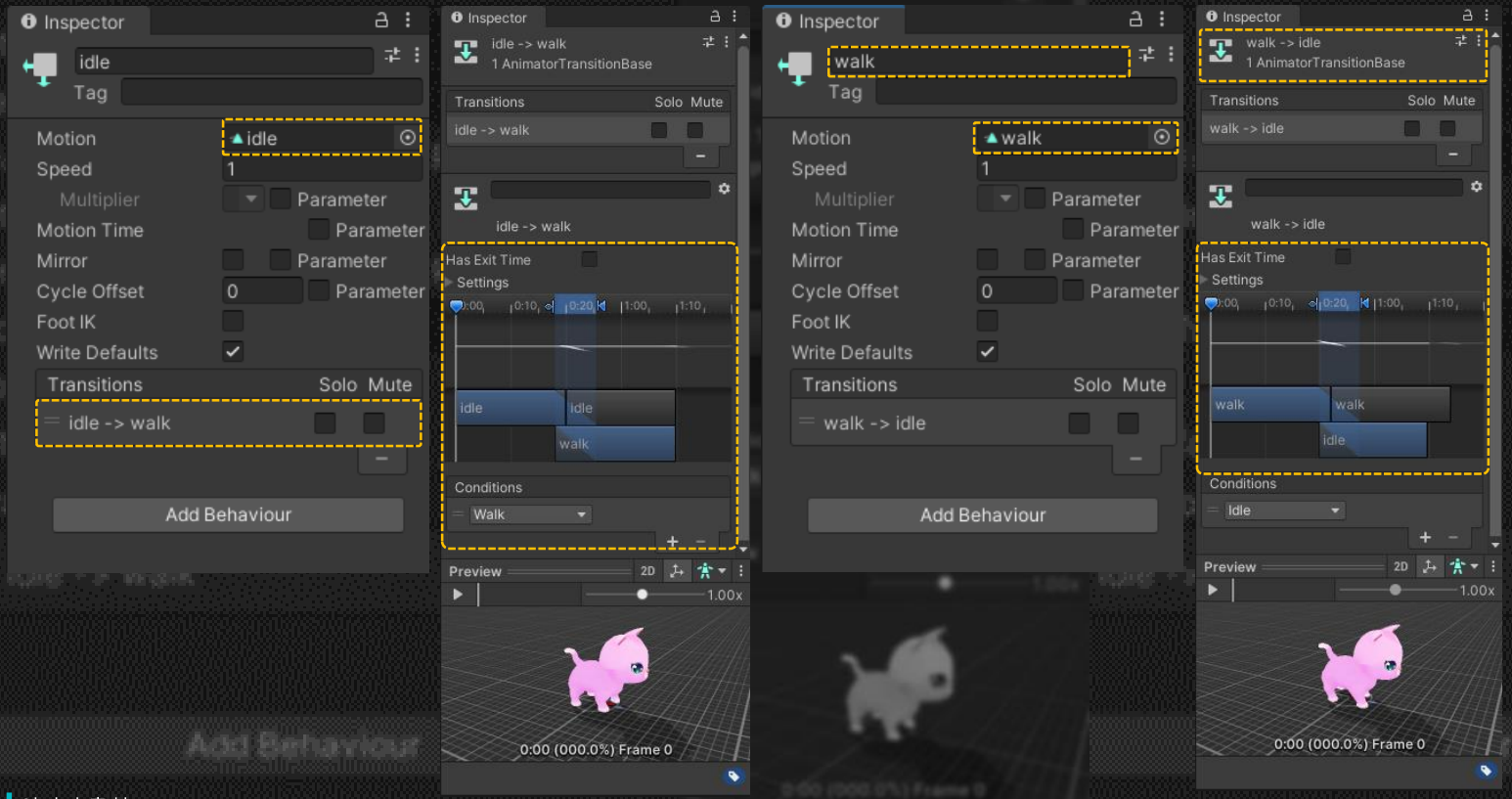




이미지 출처 :Unity

Animator Controller

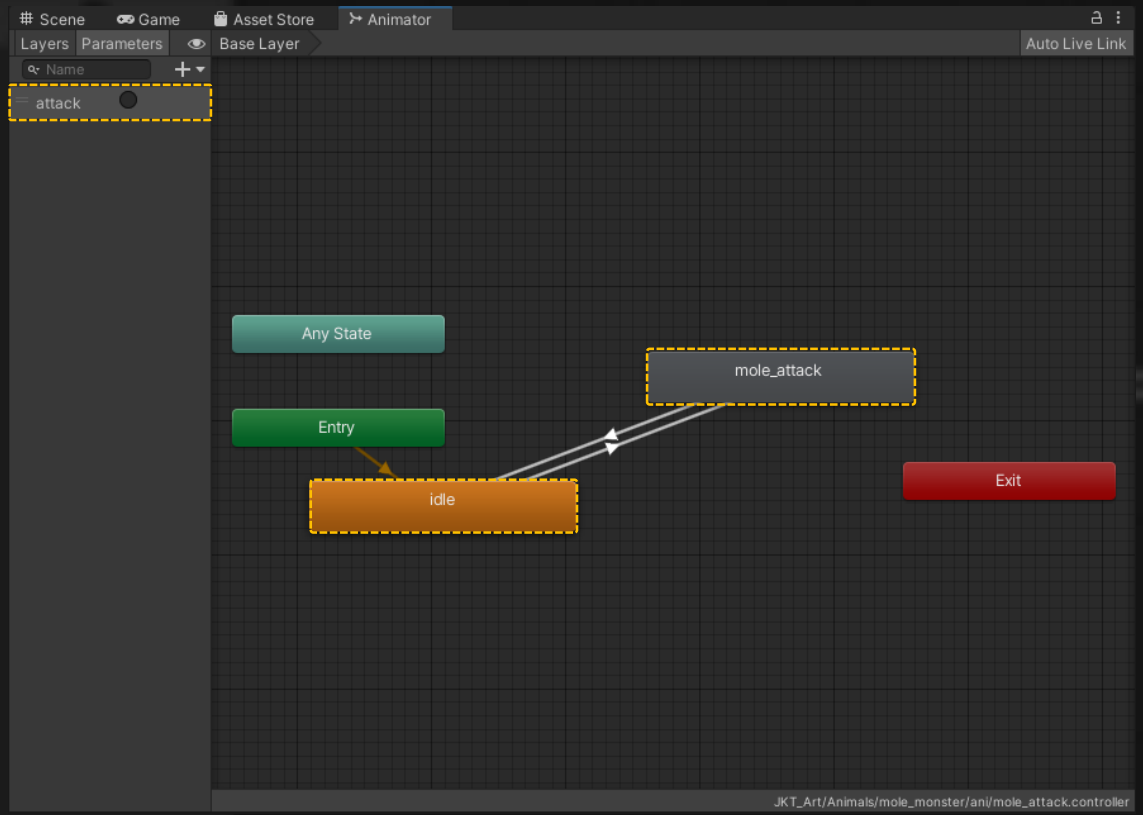
Cat Animation States and Transitions

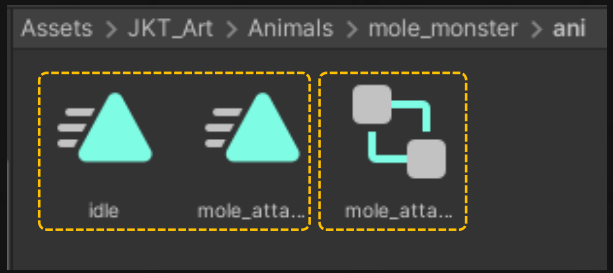


이미지 출처 :Unity

Animator Controller

Mole States : Idle, Attack

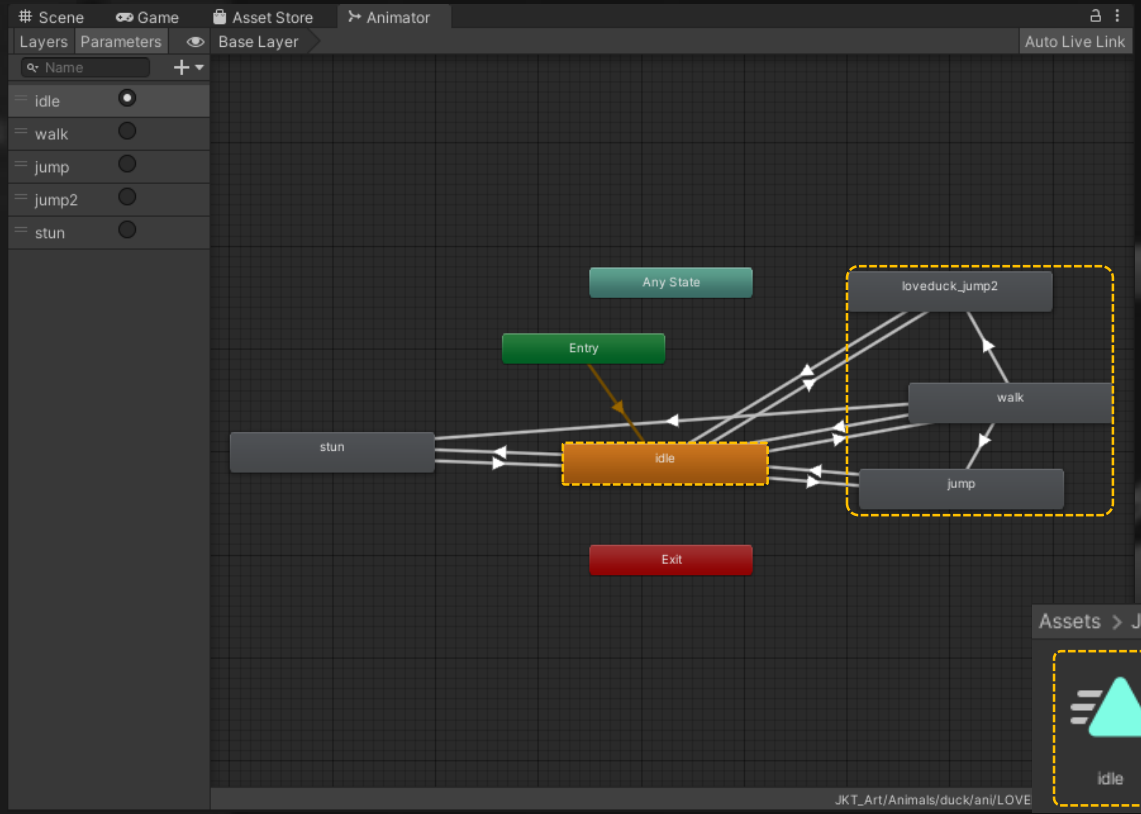


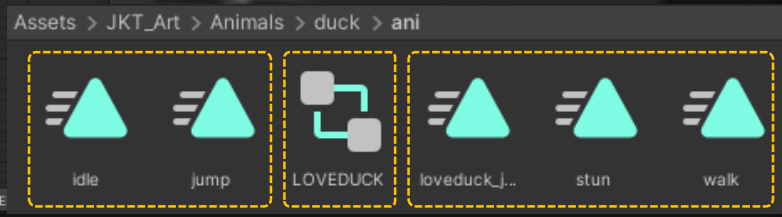


이미지 출처 :Unity

Animator Controller

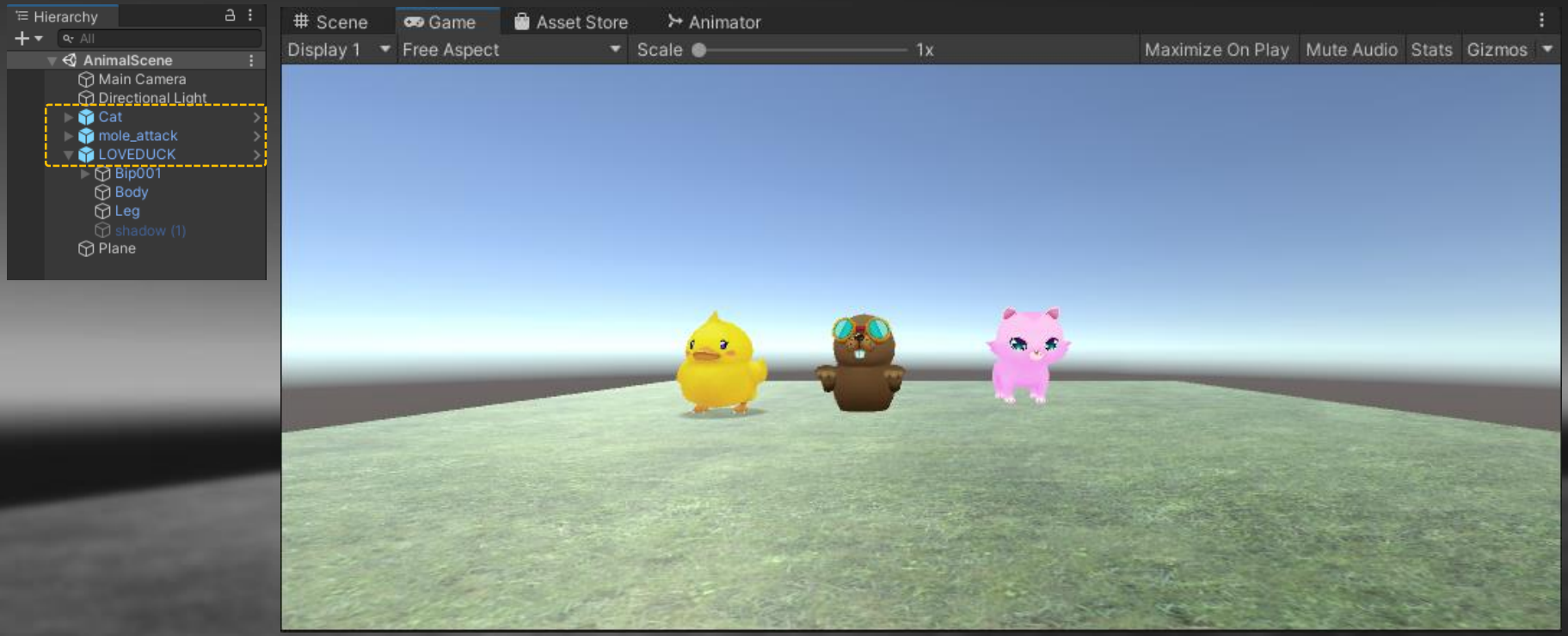
Duck States : Idle, Walk, Jump, Jump2, Stun





이미지 출처 :Unity

Animal Animation



이미지 출처 :Unity

Animal Animation

```
[RequireComponent(typeof(Animator))]
[RequireComponent(typeof(Rigidbody))]
Unity Script (1 asset reference) | 0 references
public class CatControl : MonoBehaviour
{
    [SerializeField] private string horizontalInputName = "Horizontal";
    [SerializeField] private string verticalInputName = "Vertical";
    [SerializeField] private float movementSpeed = 100f;

    private Animator animator;
    private Rigidbody rb;
    Vector3 eulerAngleVelocity;

    Unity Message | 0 references
    void Start()
    {
        animator = GetComponent<Animator>();
        rb = GetComponent<Rigidbody>();
        eulerAngleVelocity = new Vector3(0, 1, 0);
    }

    Unity Message | 0 references
    void FixedUpdate()
    {
        float vertInput = Input.GetAxis(verticalInputName) * movementSpeed * Time.deltaTime;
        float horizInput = Input.GetAxis(horizontalInputName) * movementSpeed * Time.deltaTime;

        if (vertInput > 0.01 || horizInput > 0.01) // walk
        {
            animator.SetTrigger("Walk");
        }
        else if (vertInput <= 0.01 || horizInput <= 0.01) // idle
        {
            animator.SetTrigger("Idle");
        }

        rb.velocity = transform.forward * vertInput;
        Quaternion deltaRotation = Quaternion.Euler(eulerAngleVelocity * horizInput);
        rb.MoveRotation(rb.rotation * deltaRotation);
    }
}
```

이미지 출처 :Unity

Animal Animation

```
[RequireComponent(typeof(Animator))]
Unity Script (1 asset reference) | 0 references
public class MoleControl : MonoBehaviour
{
    private Animator animator;

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

    }

    Unity Message | 0 references
    void Update()
    {
        if (Input.GetKey(KeyCode.A)) // attack
        {
            animator.SetTrigger("attack");
        }
    }
}
```

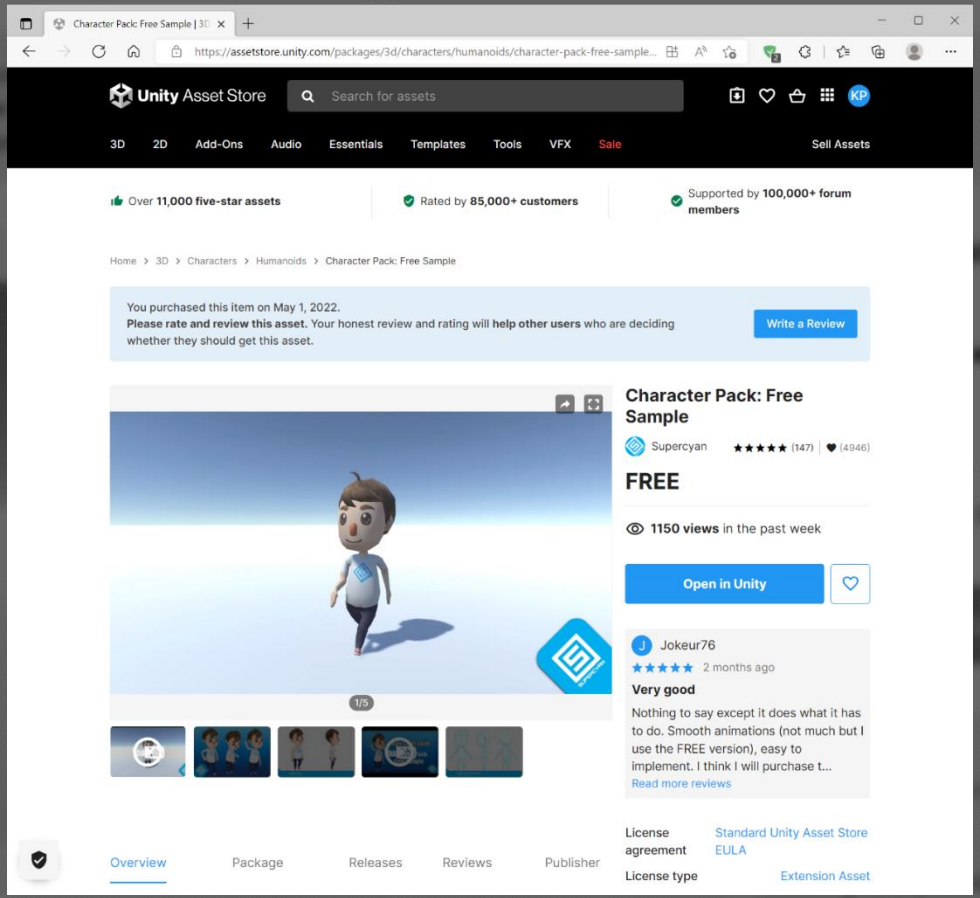
이미지 출처 :Unity

Animal Animation

```
[RequireComponent(typeof(Animator))]
[RequireComponent(typeof(Rigidbody))]
Unity Script (1 asset reference) | 0 references
public class DuckControl : MonoBehaviour
{
    [SerializeField] private string horizontalInputName = "Horizontal";
    [SerializeField] private string verticalInputName = "Vertical";
    [SerializeField] private float movementSpeed = 100f;
    private Animator animator;
    private Rigidbody rb;

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

    Unity Message | 0 references
    void FixedUpdate()
    {
        float vertInput = Input.GetAxis(verticalInputName) * movementSpeed * Time.deltaTime;
        float horizInput = Input.GetAxis(horizontalInputName) * movementSpeed * Time.deltaTime;

        if (Math.Abs(vertInput) > 0.01 || Math.Abs(horizInput) > 0.01) // walk
        {
            animator.SetTrigger("walk");
        }
        else if (Math.Abs(vertInput) <= 0.01 || Math.Abs(horizInput) <= 0.01) // idle
        {
            animator.SetTrigger("idle");
        }
        if (Input.GetKey(KeyCode.Space))
        {
            animator.SetTrigger("jump");
        }
        if (Input.GetKey(KeyCode.J))
        {
            animator.SetTrigger("jump2");
        }
        if (Input.GetKey(KeyCode.S))
        {
            animator.SetTrigger("stun");
        }

        rb.velocity = transform.forward * vertInput;
        Quaternion deltaRotation = Quaternion.Euler(Vector3.up * horizInput);
        rb.MoveRotation(rb.rotation * deltaRotation);
    }
}
```

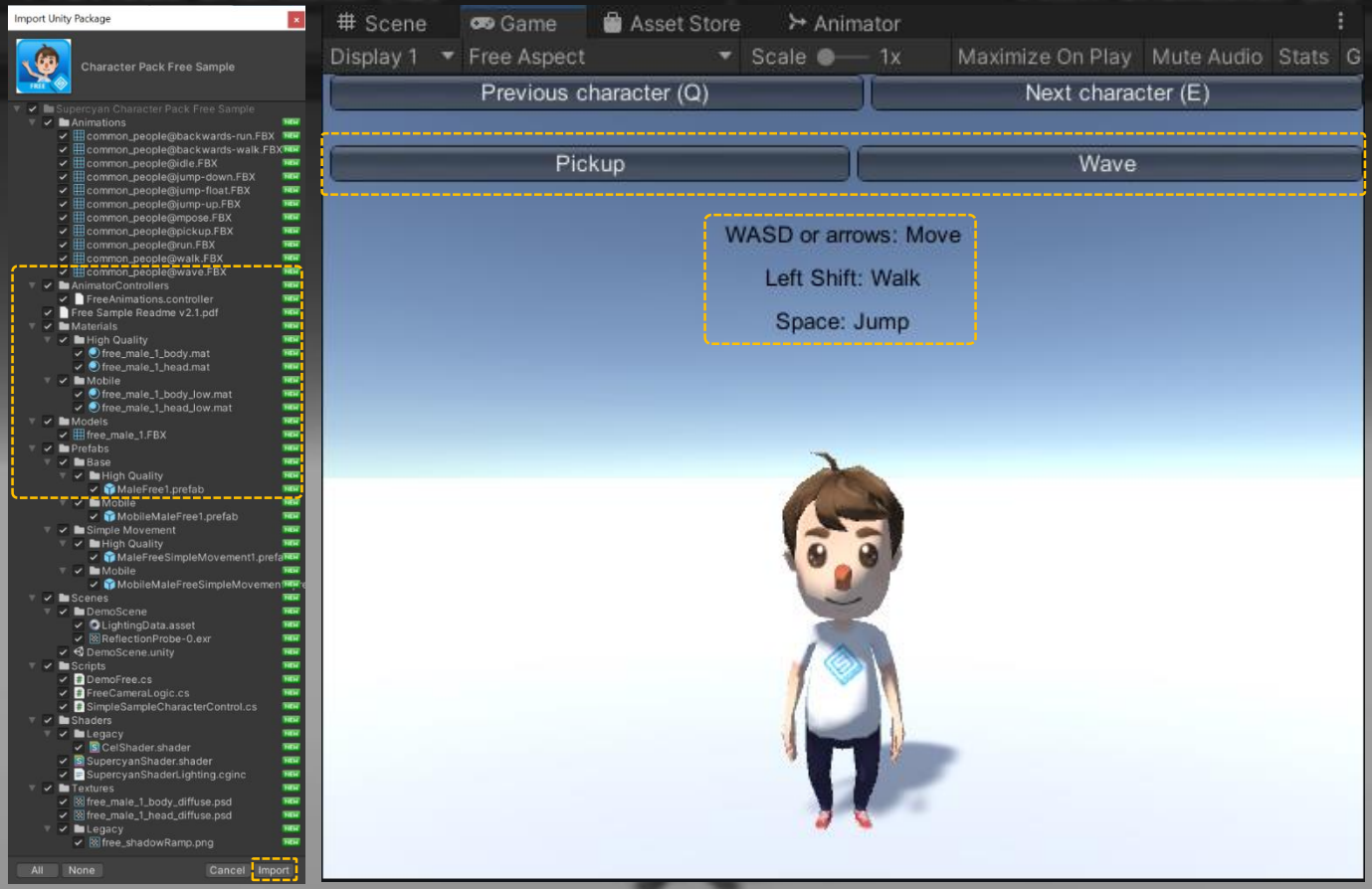
이미지 출처 :Unity

Asset Store



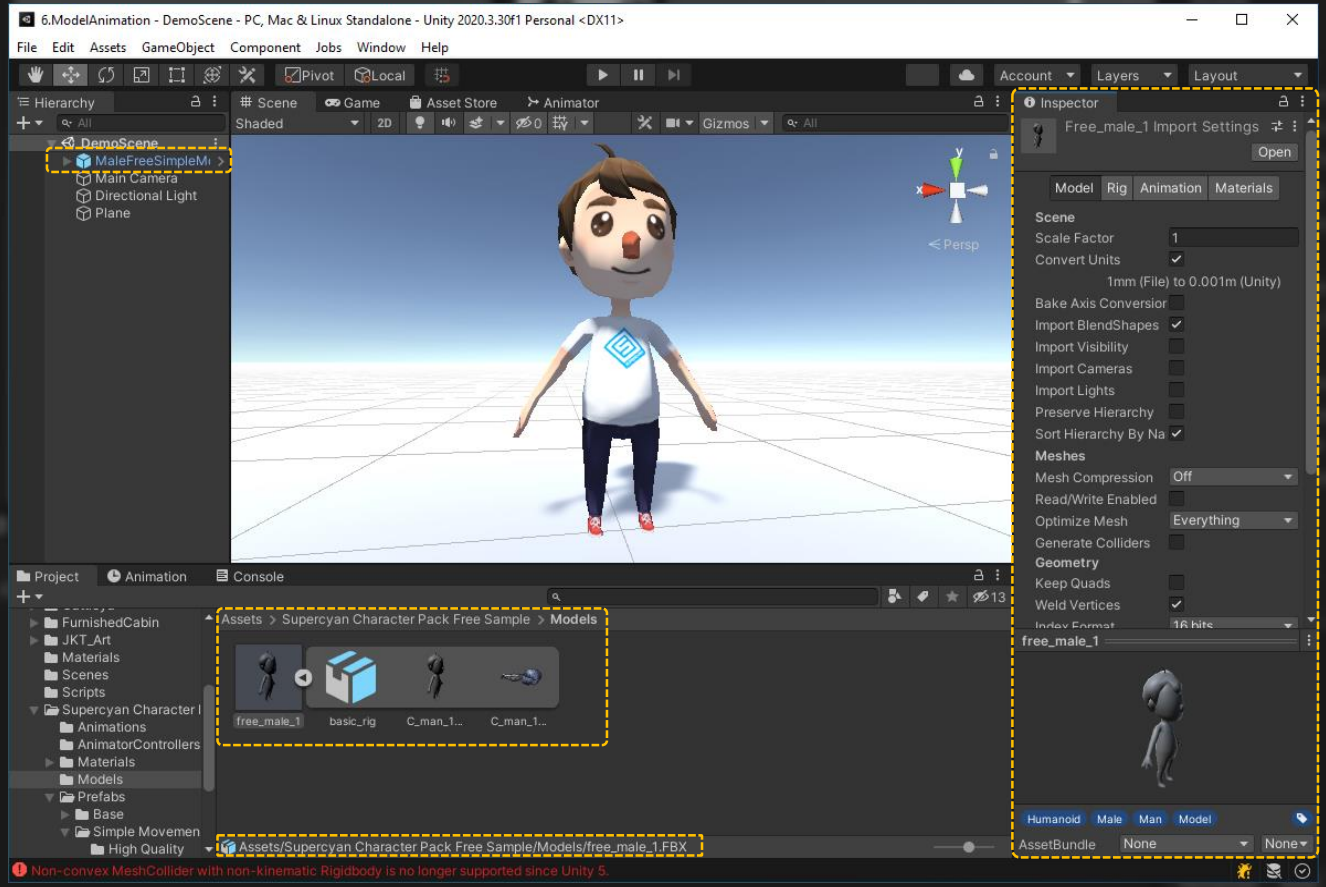
이미지 출처 :https://assetstore.unity.com/lists/supercyan-character-packs-99828

Import Unity Package



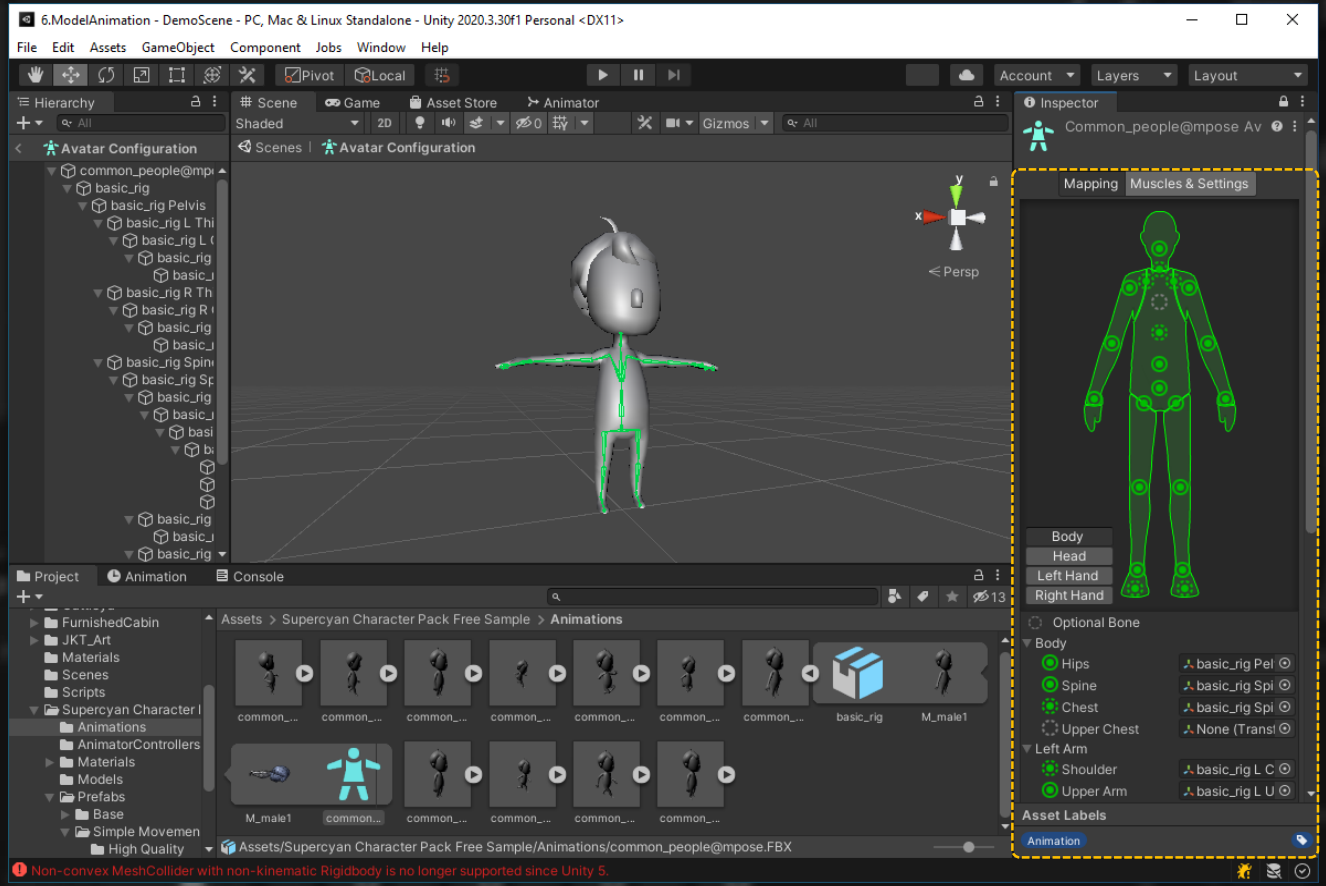
이미지 출처 : Unity

Model



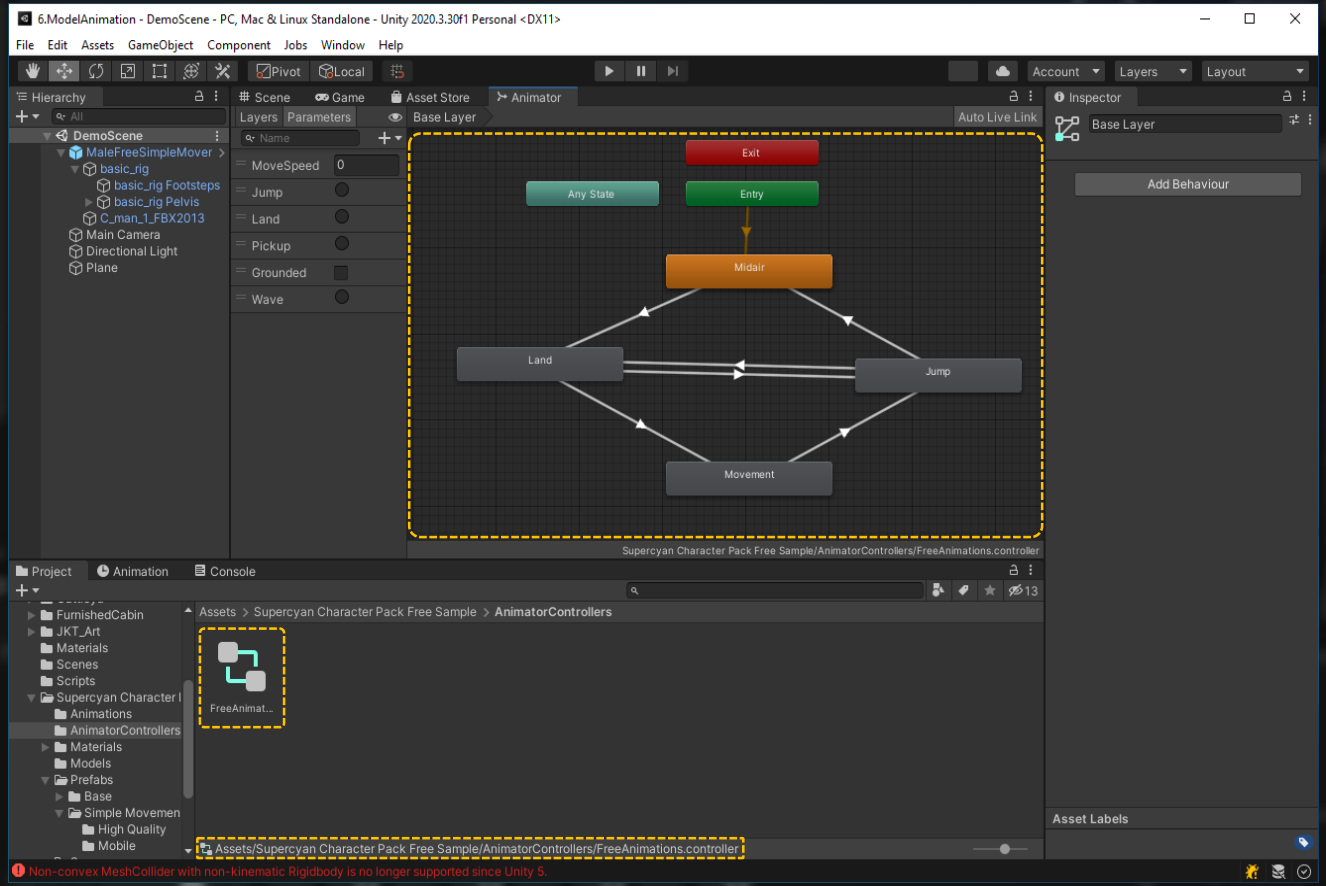
이미지 출처 :Unity

Avatar



이미지 출처 :Unity

Animator Controller



이미지 출처 :Unity

Animator Controller

```
public class DemoFree : MonoBehaviour
{

    private readonly string[] m_animations = { "Pickup", "Wave" };
    private Animator[] m_animators = null;
    [SerializeField] private FreeCameraLogic m_cameraLogic = null;

    Unity Message | 0 references
    private void Start()
    {
        m_animators = FindObjectsOfType<Animator>();
    }
```

이미지 출처 :Unity

Animator Controller

```
Unity Message | 0 references
private void OnGUI()
{
    GUILayout.BeginVertical(GUILayout.Width(Screen.width));

    GUILayout.BeginHorizontal();

    if (GUILayout.Button("Previous character (Q)"))
    {
        m_cameraLogic.PreviousTarget();
    }

    if (GUILayout.Button("Next character (E)"))
    {
        m_cameraLogic.NextTarget();
    }

    GUILayout.EndHorizontal();

    GUILayout.Space(16);

    for (int i = 0; i < m_animations.Length; i++)
    {
        if (i == 0) { GUILayout.BeginHorizontal(); }

        if (GUILayout.Button(m_animations[i]))
        {
            for (int j = 0; j < m_animators.Length; j++)
            {
                m_animators[j].SetTrigger(m_animations[i]);
            }
        }

        if (i == m_animations.Length - 1) { GUILayout.EndHorizontal(); }
        else if (i == (m_animations.Length / 2)) { GUILayout.EndHorizontal(); GUILayout.BeginHorizontal(); }
    }
```

이미지 출처 :Unity

Animator Controller

```
public class SimpleSampleCharacterControl : MonoBehaviour
{
    4 references
    private enum ControlMode
    {
        /// <summary>
        /// Up moves the character forward, left and right turn the character gradu
        /// </summary>
        Tank,
        /// <summary>
        /// Character freely moves in the chosen direction from the perspective of
        /// </summary>
        Direct
    }

    [SerializeField] private float m_moveSpeed = 2;
    [SerializeField] private float m_turnSpeed = 200;
    [SerializeField] private float m_jumpForce = 4;

    [SerializeField] private Animator m_animator = null;
    [SerializeField] private Rigidbody m_rigidBody = null;

    [SerializeField] private ControlMode m_controlMode = ControlMode.Direct;
```

이미지 출처 :Unity

Animator Controller

```
private void DirectUpdate()
{
    float v = Input.GetAxis("Vertical");
    float h = Input.GetAxis("Horizontal");

    Transform camera = Camera.main.transform;

    if (Input.GetKey(KeyCode.LeftShift))
    {
        v *= m_walkScale;
        h *= m_walkScale;
    }

    m_currentV = Mathf.Lerp(m_currentV, v, Time.deltaTime * m_interpolation);
    m_currentH = Mathf.Lerp(m_currentH, h, Time.deltaTime * m_interpolation);

    Vector3 direction = camera.forward * m_currentV + camera.right * m_currentH;

    float directionLength = direction.magnitude;
    direction.y = 0;
    direction = direction.normalized * directionLength;

    if (direction != Vector3.zero)
    {
        m_currentDirection = Vector3.Slerp(m_currentDirection, direction, Time.deltaTime * m_interpolation);

        transform.rotation = Quaternion.LookRotation(m_currentDirection);
        transform.position += m_currentDirection * m_moveSpeed * Time.deltaTime;

        m_animator.SetFloat("MoveSpeed", direction.magnitude);
    }

    JumpingAndLanding();
}
```

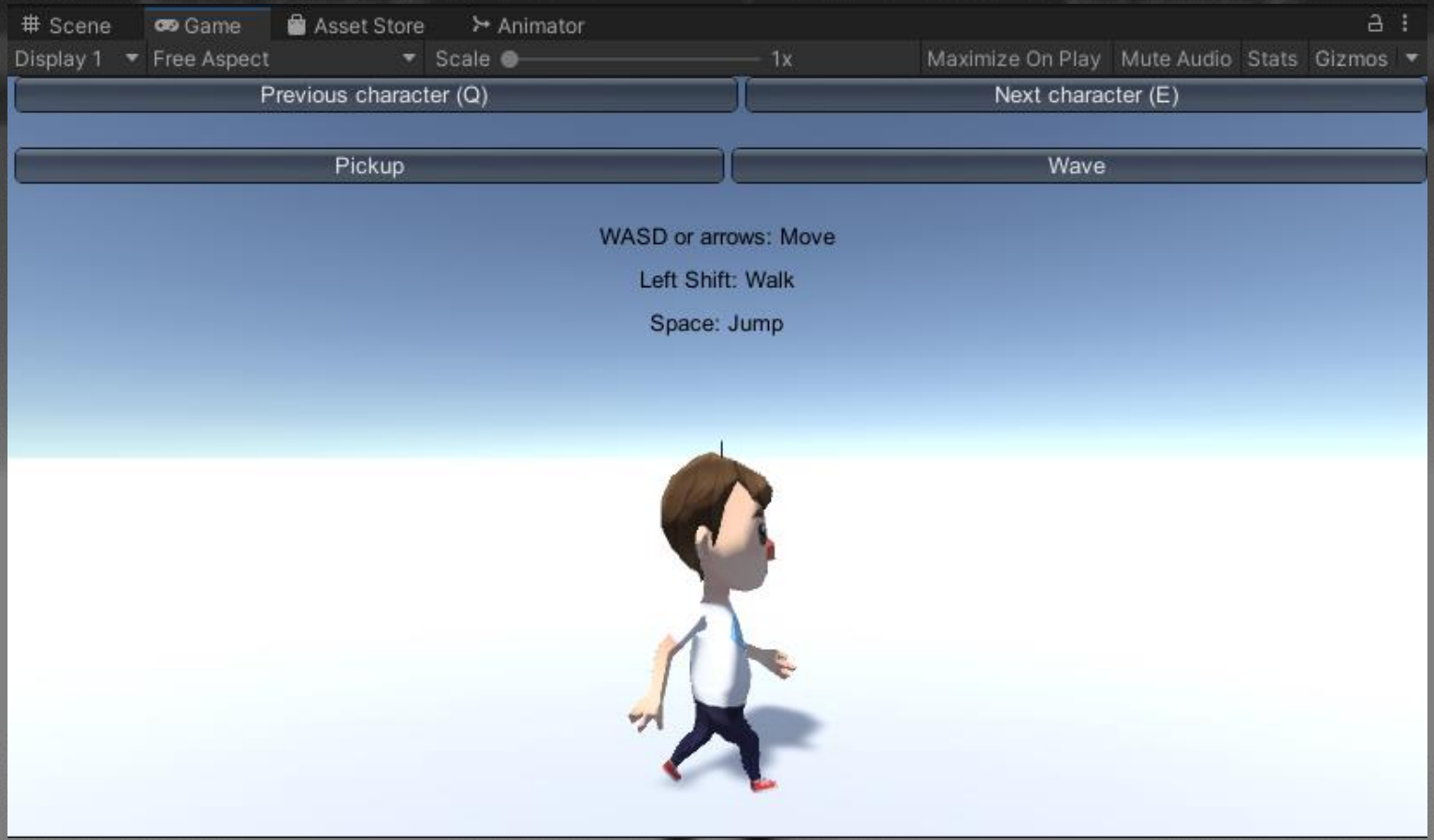
이미지 출처 :Unity

Animator Controller

```
private void JumpingAndLanding()
{
    bool jumpCooldownOver = (Time.time - m_jumpTimeStamp) >= m_minJumpInterval;

    if (jumpCooldownOver && m_isGrounded && m_jumpInput)
    {
        m_jumpTimeStamp = Time.time;
        m_rigidBody.AddForce(Vector3.up * m_jumpForce, ForceMode.Impulse);
    }

    if (!m_wasGrounded && m_isGrounded)
    {
        m_animator.SetTrigger("Land");
    }

    if (!m_isGrounded && m_wasGrounded)
    {
        m_animator.SetTrigger("Jump");
    }
}
```

이미지 출처 :Unity

Character Animation



이미지 출처 :https://assetstore.unity.com/lists/supercyan-character-packs-99828

Reference

- https://learn.unity.com/course/introduction-to-3d-animation-systems
- https://docs.unity3d.com/Manual/AnimationSection.html
- https://www.youtube.com/watch?v=2bwH9gudwCk