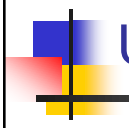


Unity Game Engine

Introduction to Unity – Character & Animation

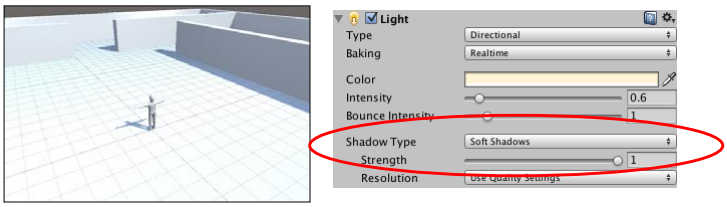
https://docs.unity3d.com/2017.4/Documentation/Manual/AnimationSection.html

Topics

- Import a character model with Animations into the scene.
- Implement character controls such as playing animations on the model according to its movements.

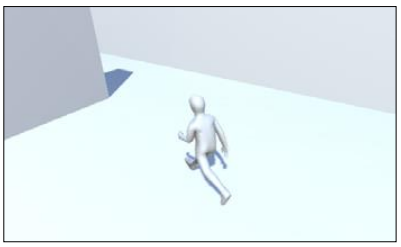
Check Shadow Setting First

- Shadows are turned on for the default light.
- Make sure to turn the shadow for the directional light in your scene.
  - Select the directional light.
  - Inspector > Shadow Type > Soft Shadows



Character with Animations

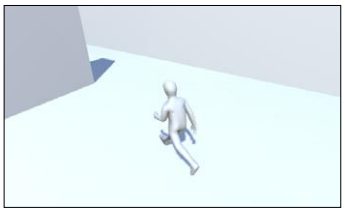
- An animation is a packet of information that defines movement of the associated 3D object.
  - e.g character walking around



FantasyMonster from the Asset Store

Import a Character Model with Animations

- Two files are given:
  - human.fbx
  - human.tga

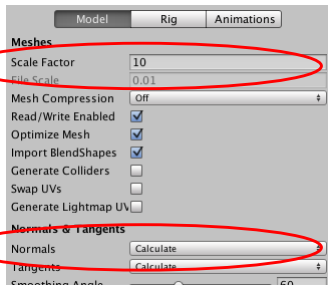


- Find a character model with animations that can be used to replace your cube enemy in the game.

FantasyMonster from the Asset Store

Import “human.fbx”

- Import the FBX file (human.fbx) into the project.
  - In the Inspector, adjust “Import Settings” for the import model
    - Scale as desired.
    - Set “Normals” from “Import” (default) to “Calculate”.
    - Then click “Apply” button.
- Drag the character model from the Project view up into the scene. Position the in the scene.

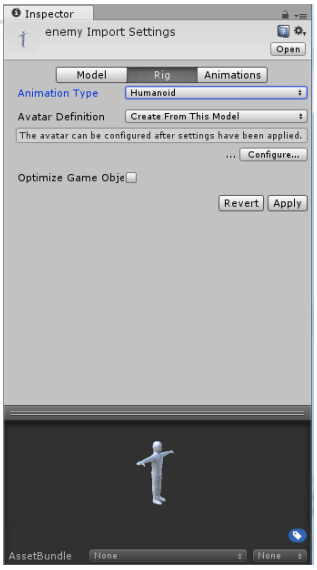


Import a Character Texture

- Import the texture image (human.tga)
- Select the “human” material in the Materials folder and then assign the texture image onto the empty texture slot in the Inspector.
  - “human” material has been generated when you imported the model.
  - You won’t see a dramatic change in the model’s color, but there are shadows painted into the texture that’ll improve the look of the model.

Turn on Animation

- Select the character model
- Inspector > Import > Animations tab > check “Import Animation”
- Rig tab > Animation Type (change from Generic to Humanoid).
- Click the Apply button.

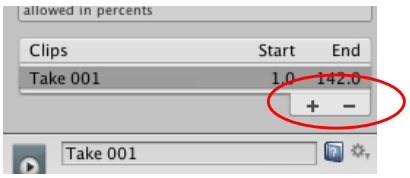


Defining Animation Clip

- A character can have multiple movements (e.g walking, jumping, dancing, etc.)
- Each of these movements is a separate “clip” that can play individually.
- Depending on the way the model was animated, these separate movements might be imported as distinct animation clips or as one single clip where each movement simply follows on from the previous one.

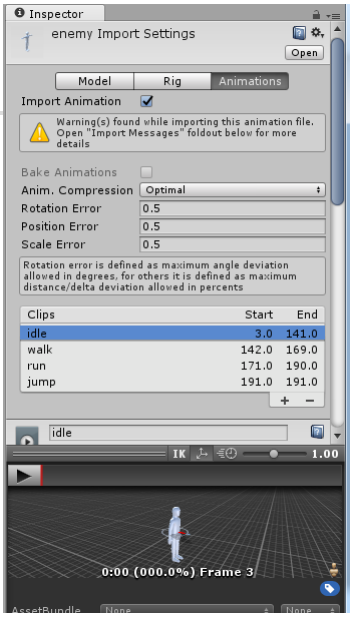
Models that have unsplit animations (human.fbx)

- Often imported animations come as a single long clip that can be cut up into shorter individual animations.
  - To split up the animation clips, first select the Animations tab in the Inspector.
  - Use + and - buttons to add and remove clips on the list.



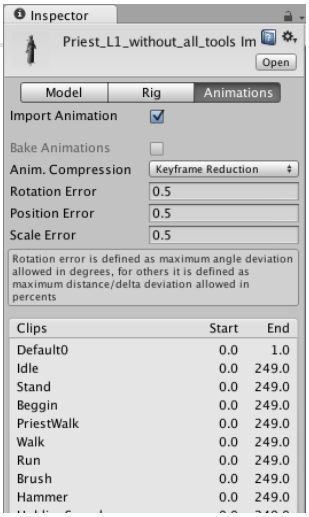
Splitting Animations

- human.fbx – 4 animation clips
  - **Idle** animation (frame 3 – frame 141), **Walk** animation (frame 141 – frame 169), **Run** animation (frame 171 – frame 190)
    - Select “Loop Time” and “Loop Pose”
    - Set Body Orientation/Center Of Mass/Center Of Mass for Root Transformation Rotation/Position/Position in that order.
  - **Jump** (frame 190.5 – frame 191). This is a single-frame pose.
    - Don’t select Loop Time.
  - Click “Apply” after defining all animation clips.



Models that have pre-split animations

- If you have an animation that contain pre-split animations, the **Animations** tab in the **Animation Importer Inspector** will look like the screen shot.
  - You will see a list available clips which you can preview by pressing Play in the **Preview Window** (lower down in the inspector).
  - The frame ranges of the clips can be edited, if needed.

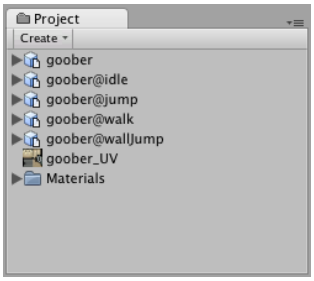


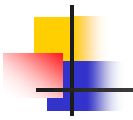
Importing Animations using Multiple Model Files

- When you have multiple model files, follow a naming scheme that Unity allows for the animation files.
- You create separate model files and name them with the convention ‘modelName@animationName.fbx’.
  - For example, for a model called “goober”, you could import separate idle, walk, jump and walljump animations using files named “goober@idle.fbx”, “goober@walk.fbx”, “goober@jump.fbx” and “goober@walljump.fbx”.

Importing Animations using Multiple Model Files

- Unity automatically imports all four files and collects all animations to the file without the @ sign in.
- Only the animation data from these files will be used, even if the original files are exported with mesh data.





Treadmill vs Root Motion

- There are two types of animation, treadmill and root motion.
- **Treadmill** means that the animation stays at the origin and we **use code to move** that asset around.
- **Root Motion** means the motion is built right into the animation and it's the animation that determines how far something moves rather than code.

Mecanim Animation System

- Unity has a rich and sophisticated animation system called **Mecanim**.
- 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.
  - and more …

Animator Controller

- In most situations, it is normal for a character to have **multiple animations** and switch between them when certain game conditions occur.
  - e.g. switching from a walk animation to a jump whenever the spacebar is pressed.
- An Animator Controller allows you to arrange and maintain a set of animations for a character.

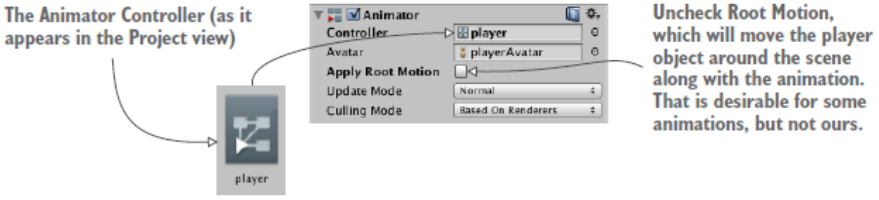
Create an Animator Controller

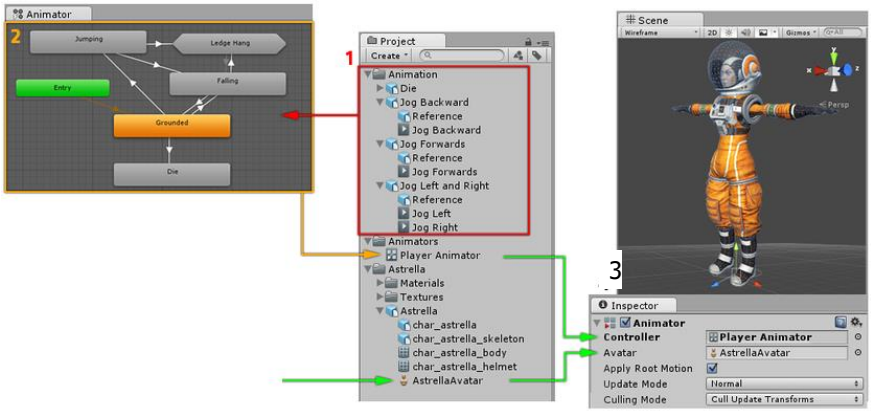
- Create a new **animator controller** asset.
- Assets > Create > Animator Controller.
  - Give it a name that is suitable for your game (e.g EnemyAnimController or Player for 3rd Person View).

A graph icon in Project View

Assign the Animator Controller to a Character(Object)

- Select the character in the scene.
- Assign the controller to the "Controller" slot of Animator component of the character model.
- **Uncheck "Root Motion"**.

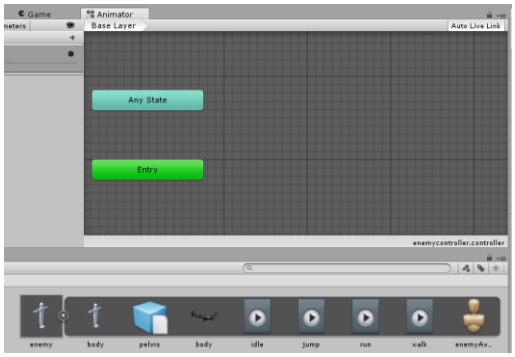




1. Animation clips are imported from an external source or created within Unity.
2. The animation clips are placed and arranged in an Animator Controller.
3. When animating the character model, it has an Animator component attached. In the **Inspector** view, you can see the Animator Component which has the Animator Controller assigned.

Editing Animator Controller

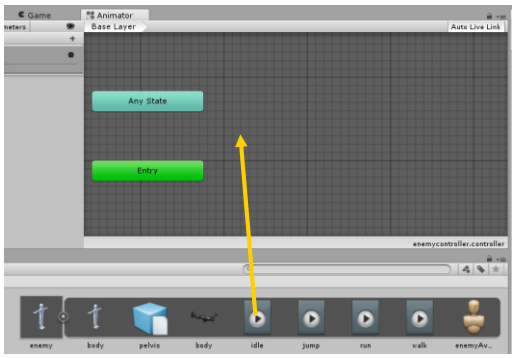
- Open the Animator View (Window > Animator). Or select the animator controller and click “open” in Inspector.
- The animator controller is a graph of connected nodes called **State**.



Initially there are only two default nodes, for **Entry (Green node)** and **Any State**.

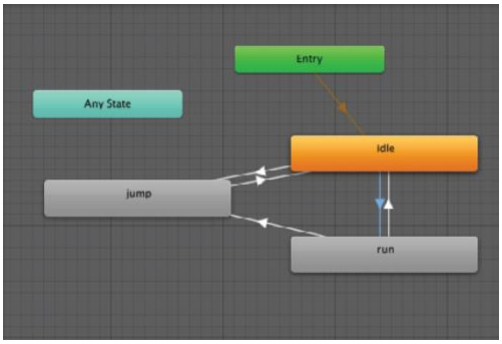
Editing Animator Controller

- Drag animations in the Project view into the Animator view.



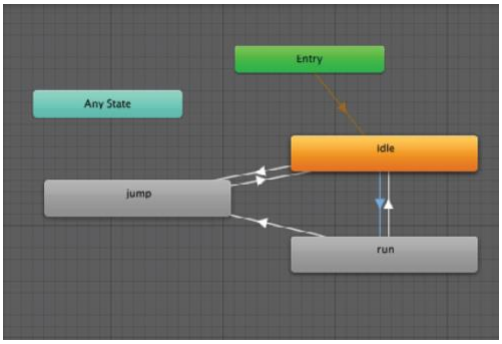
Editing Animator Controller

- State transitions. These transition lines determine how the animation states connect to each other, and control the changes from one state to another during the game.



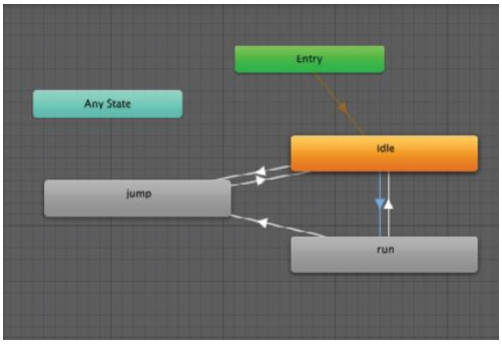
Transitions b/w States

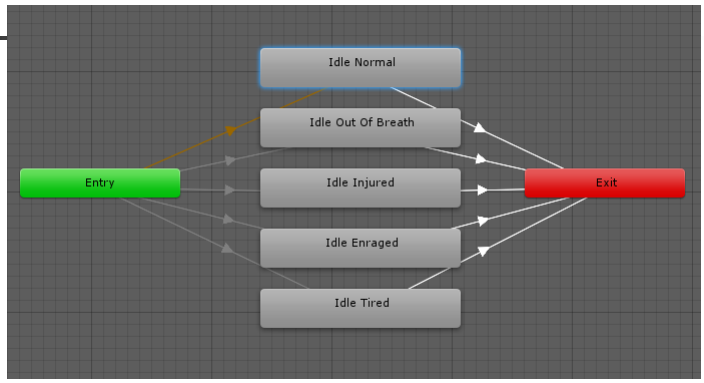
- **Right-click** on a node and select "**Make Transition**" in order to start dragging out an arrow that you can click on another node to connect.
- Connect nodes in the pattern shown in the diagram.



Entry Node

- The entry node (**green**) controls which state the state machine begins in.

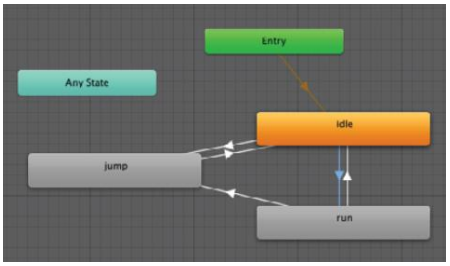




An entry node with a multiple entry transitions.
The entry node will be evaluated and will branch to the destination state according to the conditions set.

Default State

- Right-click on the **Idle** node and select "Set As Layer Default State".
  - Default state node will appear in **orange**.
  - The **default state** is where the network of nodes starts before the game has made any changes.



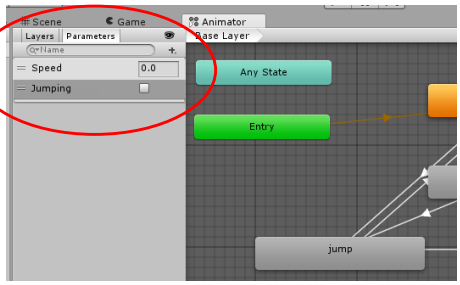
Because state machines always have a default state, there will always be a default transition branching from the entry node to the default state.

Any State

- **Any State** is a special state which is always present.
- It exists for the situation where you want to go to a specific state regardless of which state you are currently in. (e.g for transitioning to death, which can start from any state).

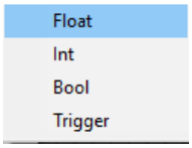
Animation Parameters

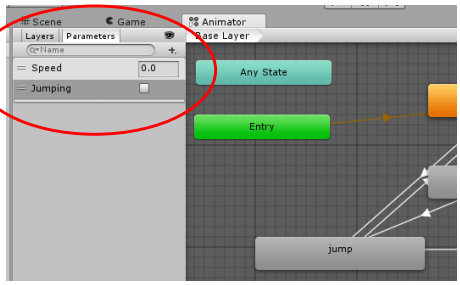
- Animation Parameters are variables that are defined within an Animator Controller.
- They can be accessed and assigned values from scripts. This is how a script can control or affect the flow of the state machine.



Animation Parameters

- Add parameters as you need. For example,
  - Speed (Float).
  - Jumping (Bool).
- Date types





Parameter values changed by using
`SetFloat(key, value)`,
`SetInt(key, value)`,
`SetBool(key, value)`,
`SetTrigger(key)`.

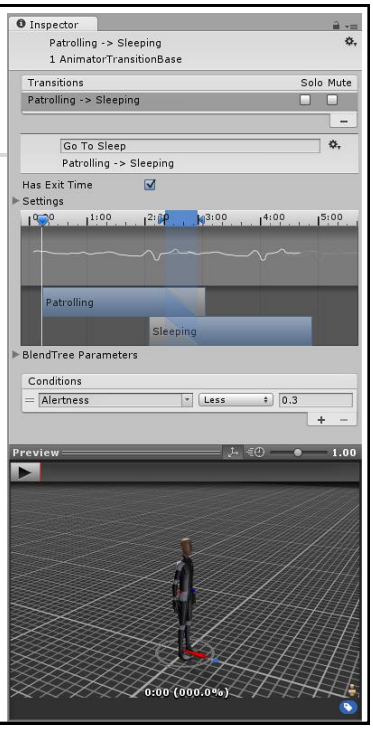
Animation Parameters

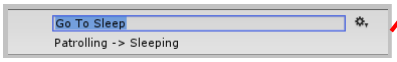
- Trigger type - a boolean parameter that is reset by the controller when consumed by a transition.
  - One scenario showing Bool vs Trigger – Assume to have a state machine with 2 conflicting state change events in one frame. One was float change and the other was bool (from false to true). Bool was true for a very short time, but the float transition was getting priority and until its state finishes the bool was already false and did not trigger his transition. Trigger solves this by "remembering" the event until it is used by some transition.

Animation Transition

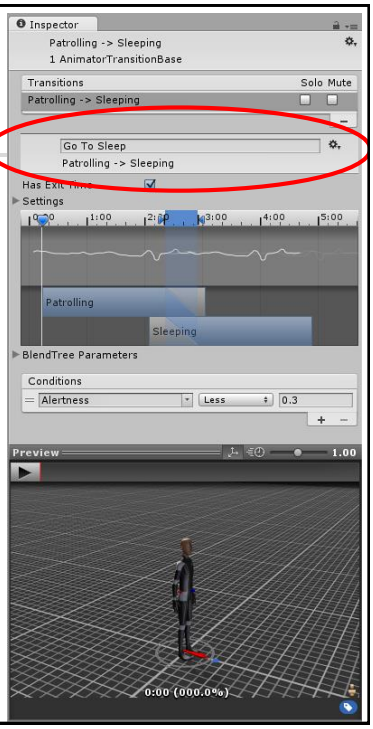
- We'll adjust how the animation states change when the parameters change.
- One example – Assume that your character has a "patrolling" state and a "sleeping" state. You want set the transition between patrolling and sleeping to occur only when an "alertness" parameter value drops below a certain level.

- Click on a **transition line**.
- An example of a transition as viewed in the inspector.

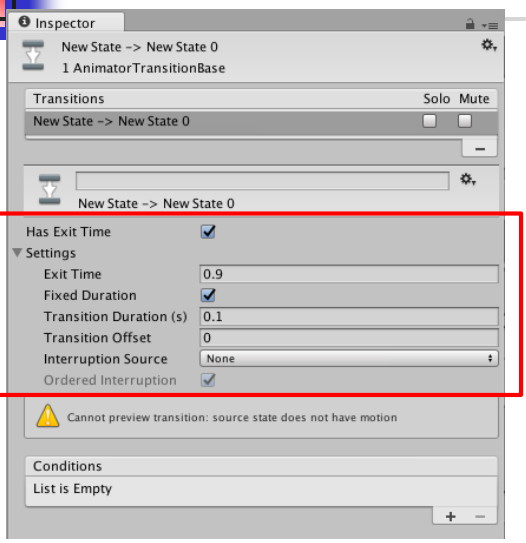




To give transitions a name, type it into the field.

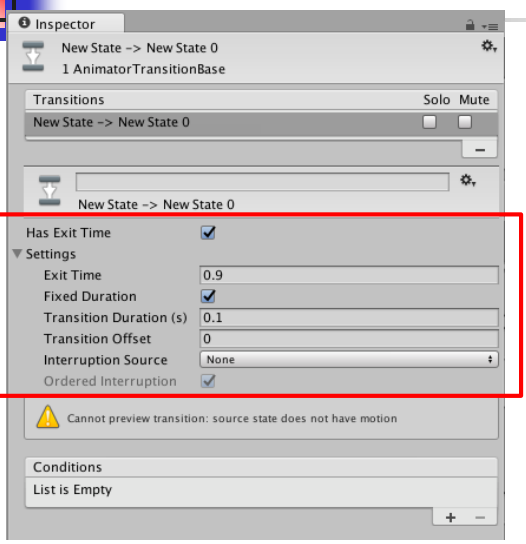


Transition Properties



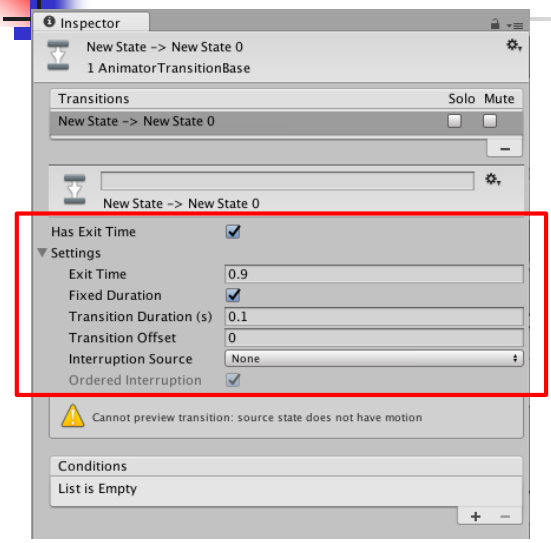
- **Exit Time**
  - If **Has Exit Time** is checked, this value represents the exact time at which the transition can take effect. This is represented in normalized time (for example, an exit time of 0.75 means that on the first frame where 75% of the animation has played, the **Exit Time** condition is true).

Transition Properties



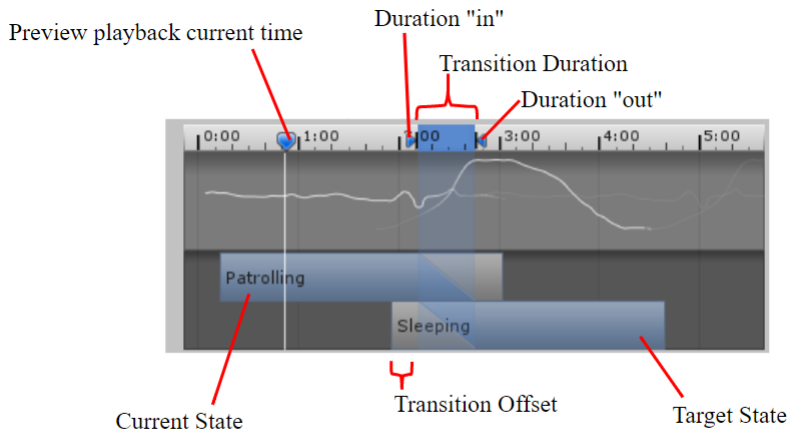
- **Fixed Duration**
  - If the **Fixed Duration** box is checked, the transition time is interpreted in seconds.
  - If the **Fixed Duration** box is not checked, the transition time is interpreted as a fraction of the normalized time of the source state.

Transition Properties



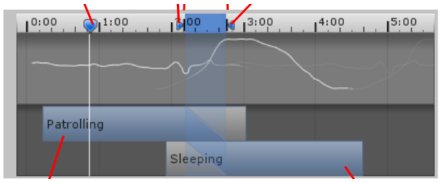
- **Transition interruption**
  - Control how your transition can be interrupted.
  - **None (**usually**)**
  - **Current State** - Queue the transitions from the current state. Other transitions from the current state can interrupt the current transition.

Transition Graph



The Transition settings and graph as shown in the Inspector

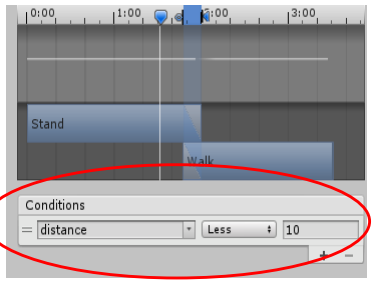
Transition Graph



- Drag the **Duration "out"** marker to change the **Duration of the transition**.
- Drag the **Duration "in"** marker to change the duration of the transition and the **Exit Time**.
- Drag the target transition to adjust the **Transition Offset**.
- Drag the preview playback marker to scrub through the animation blend in the preview window at the bottom of the Inspector.

Transition 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.
- A condition consists of:
  - A parameter
  - A conditional predicate (if needed, for example, 'less than' or 'greater than' for floats)
  - A parameter value



The transition from "Stand" to "Walk" will occur when "distance" is less than 10.

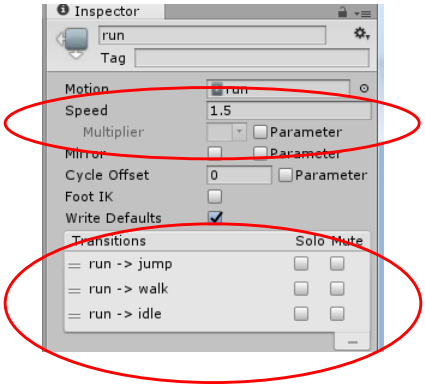
Editing Animator Controller

- Set all the transitions based on the following table.

Transition	Condition	Interruption
Idle-to-Run	Speed greater than .1	Current State
Run-to-Idle	Speed less than .1	None
Idle-to-Jump	Jumping is true	None
Run-to-Jump	Jumping is true	None
Jump-to-Idle	Jumping is false	None

Animation State Node

- Select an **animation state/node** (e.g. Idle node). As you desire
  - change the **playback speed** if the animation looks too slow.
  - adjust the **ordering of transitions**



Script Operating the Animator

- Create a Movement script and attach it to the character.
- Get Animator Component.

```
public class SimplePlayer : MonoBehaviour {

  private Animator _animator;
  // Use this for initialization
  void Start () {
    _animator = GetComponent<Animator>();
    ...
  }
...
```


Script Operating the Animator

- Access Animation Parameters

```
public class SimplePlayer : MonoBehaviour {
  private Animator _animator;
  private float _speed=0.01f;
  // Use this for initialization
  void Start () {
    _animator = GetComponent<Animator>();
    _animator.SetFloat("Speed", _speed);
    ...
  }
  ...
}
```

Movement.cs

```
...
private Animator _animator;
private float _speed=0.01f;

// Use this for initialization
void Start() {
        _animator = GetComponent<Animator>();
        _animator.SetFloat("Speed", _speed);
        _animator.SetBool("Jumping", false);
}

// Update is called once per frame
void Update() {
        if (Input.GetKey ("1")) {
                _animator.SetBool ("Jumping", true);
                _speed = 0.01f;
                _animator.SetFloat ("Speed", _speed);
        }
        else if (Input.GetKey ("2")) {
                _animator.SetBool ("Jumping", false);
                _speed = 0.5f;
                _animator.SetFloat ("Speed", _speed);
        }
        else if (Input.GetKey("3")) {
                _animator.SetBool ("Jumping", false);
                _speed = 0.01f;
                _animator.SetFloat ("Speed", _speed);
        }
}
...
```

Practice

- Find a character model with animations and replace the cube enemy in your game.