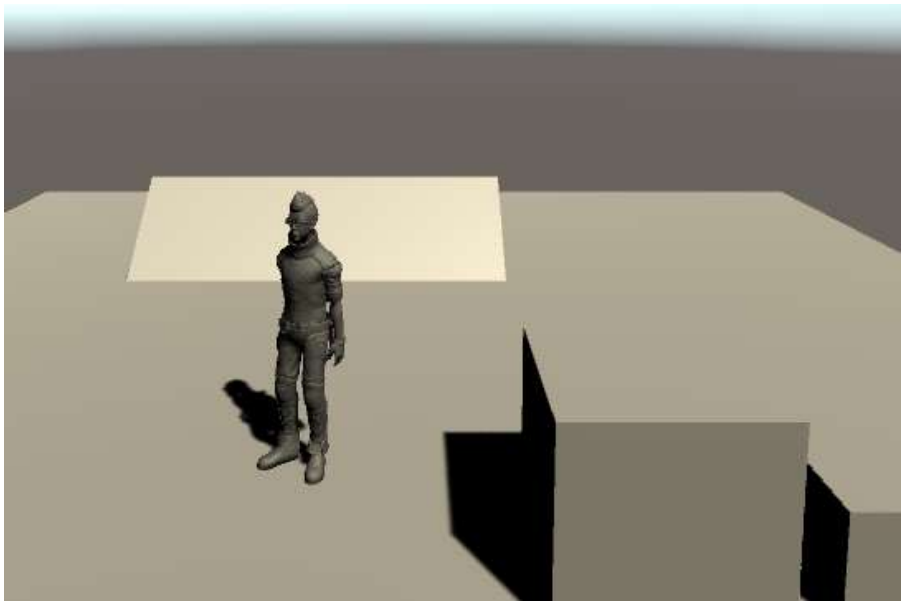
Animation Controller in Unity
Unity Tutorial

Character Controller

- Character controller is important in a game as it allows us to control the player character to interact with the game world
- Involves animation control
- Usually control by scripts together with motion animations
- We will develop a character controller in this tutorial

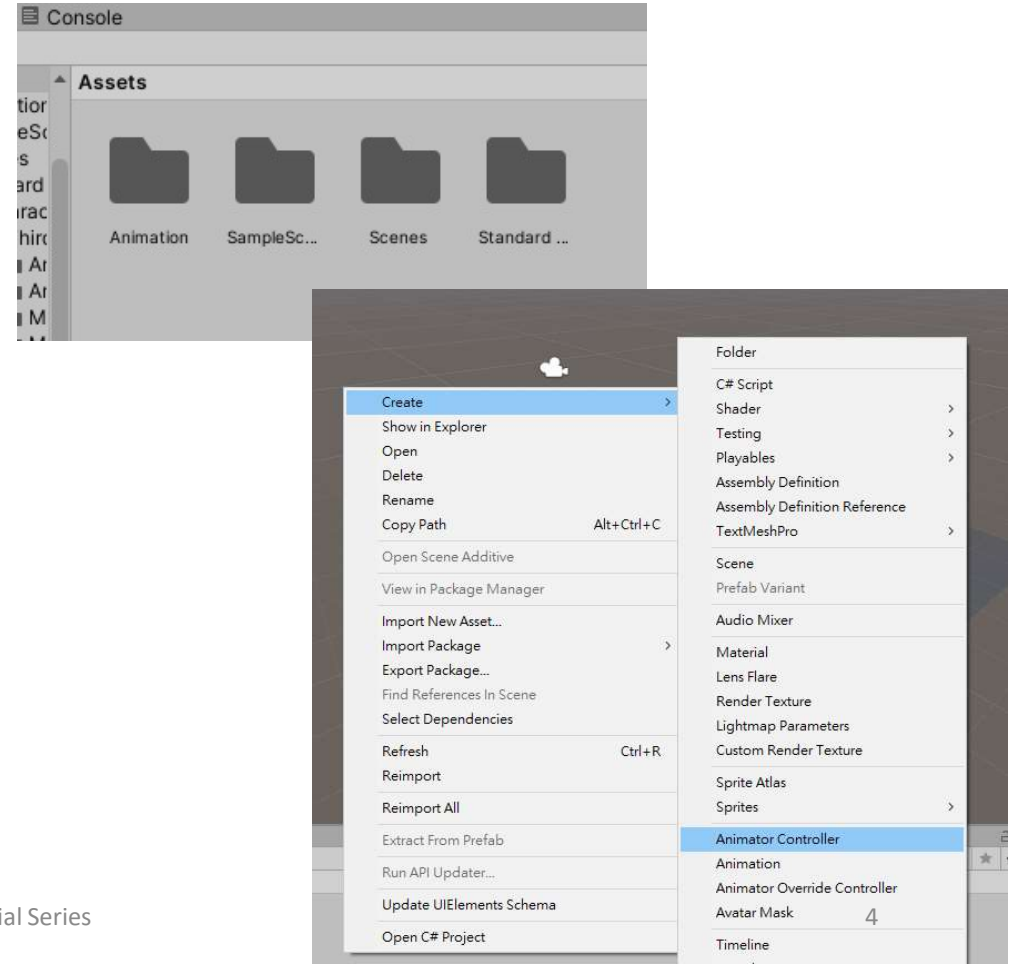
Character Controller

- Create a new project in Unity
- Create a plane and drop some objects so that we can jump on it
- Unzip our “StandardAsset” package into under your project Asset folder
- Go to “StandardAssets/Characters/ThrirdPersonCharacter/Prefabs” folder
- Drag and drop ThirdPersonController to the world and test it



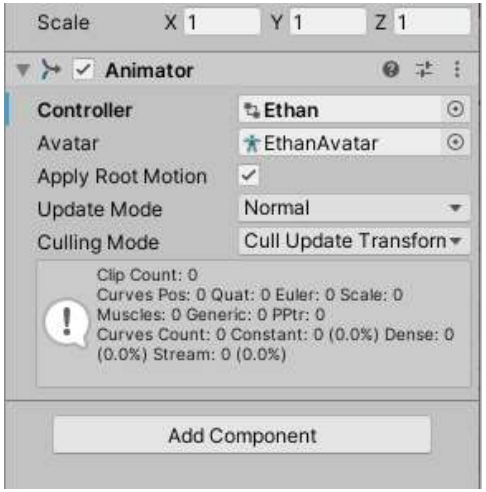
Character Controller

- Now delete the controller
- Go to “Characters/ThirdPersonCharacter/ Models” folder and drag and drop Ethan to the scene
- Create an folder and name it “Animation” under Assets
- Enter the folder and right click and select “Create/Animation Controller”
- Name it “Ethan”



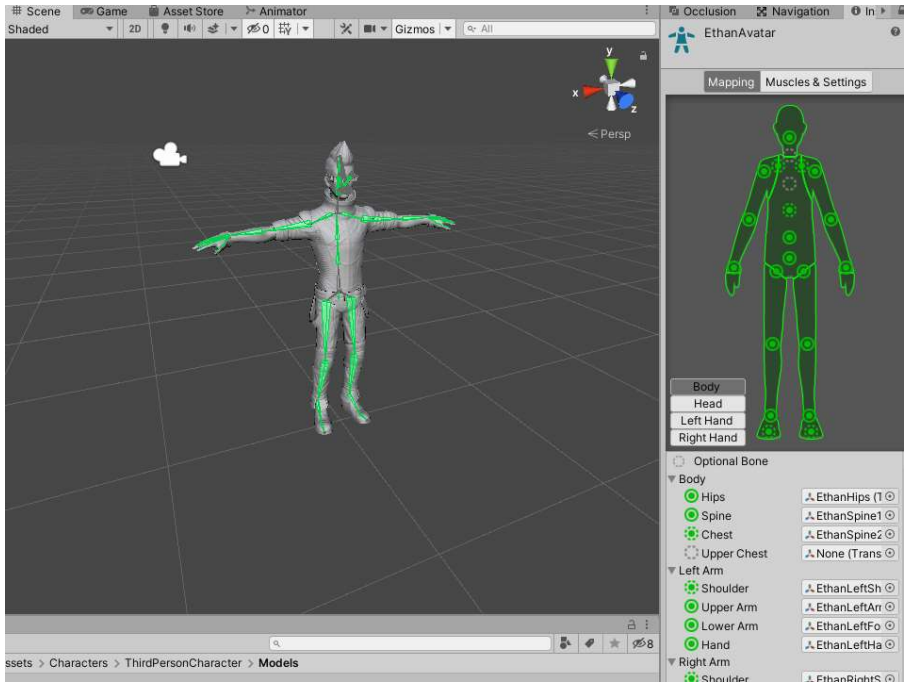
Animator Controller

- With the Ethan model selected, drag and drop our animator controller to the “Controller” field under Animator
- as our character is a human model, it needs something called “Avatar”
- Click on the Avatar filed (should have “EthanAvatar” and note the folder now should reflect as on right
- Click on the EthanAvatar in folder and click “Configure Avatar”



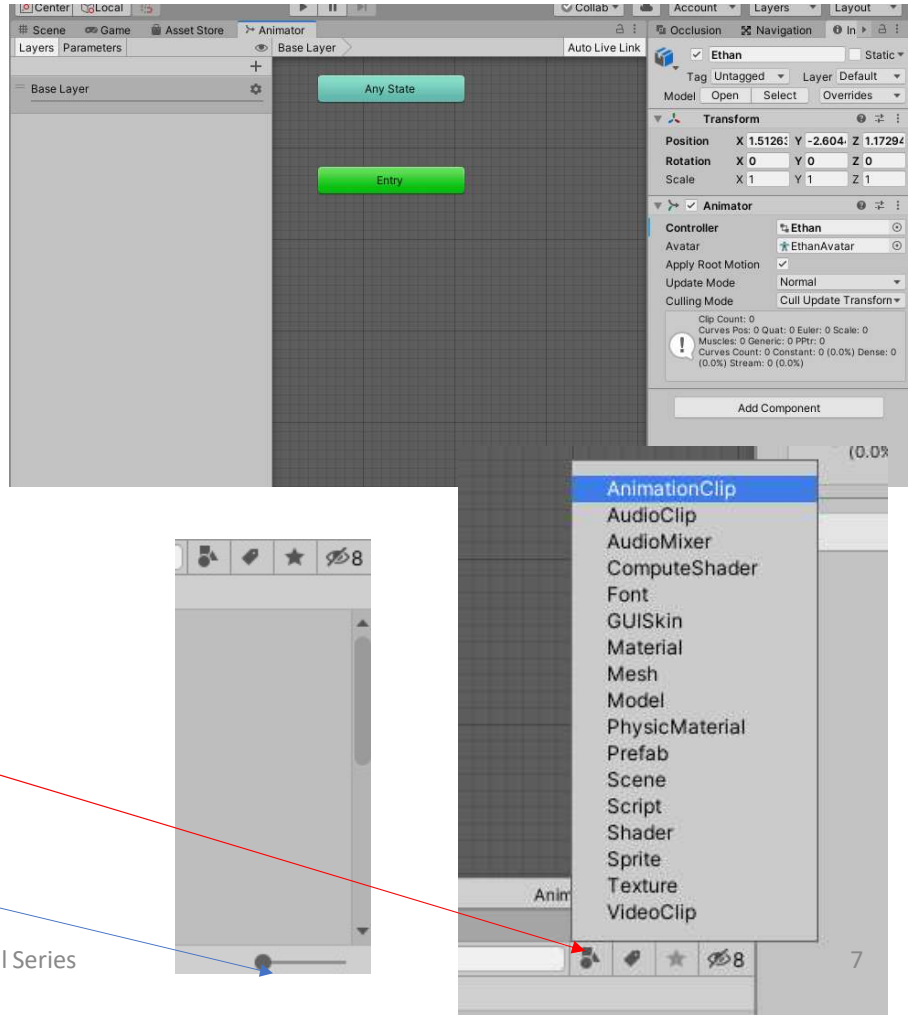
Avatar

- Avatar is to let Unity to configure the bone section of Unity with the skeleton correspondences
- As we already got the configuration done, so scroll down the inspector tab and click “done” to back to scene
- With the Ethan model selected, choose from Menu “Window/Animation/Animator”



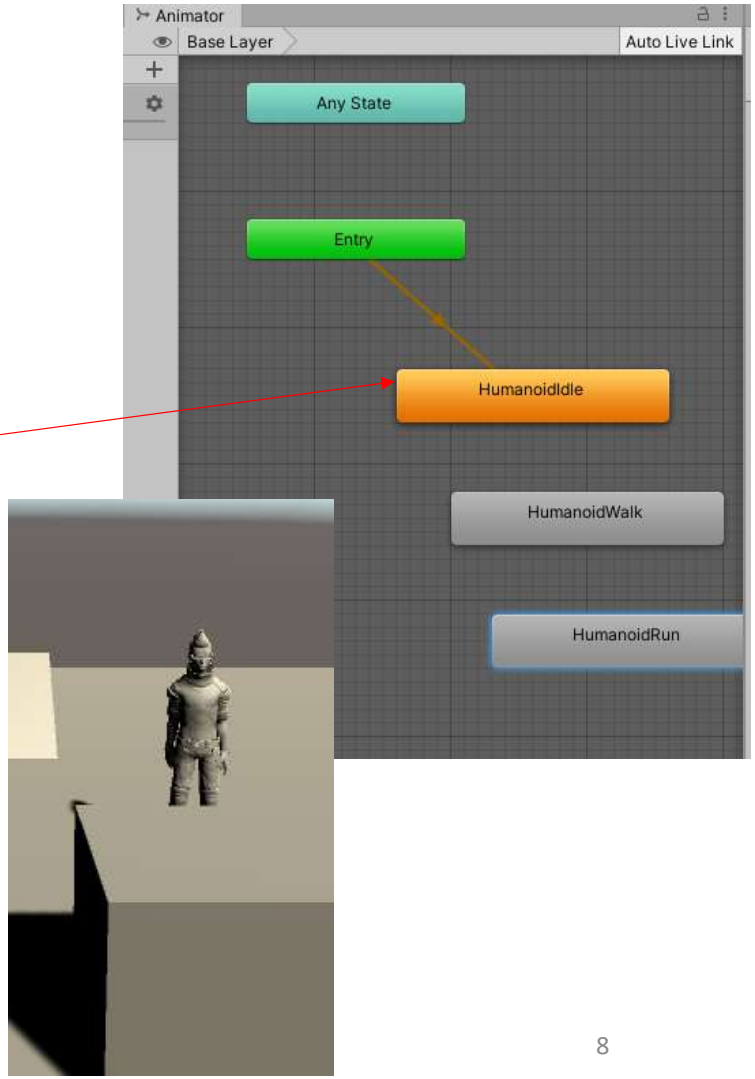
Animator

- Your screen should like that on right
- Here it control how animations are displayed
- We first need to add some animations to the controller
- In the search by type in Project folder, choose AnimationClip
- This filter let you see only animation
- Set the view icon to leftmost to see only text



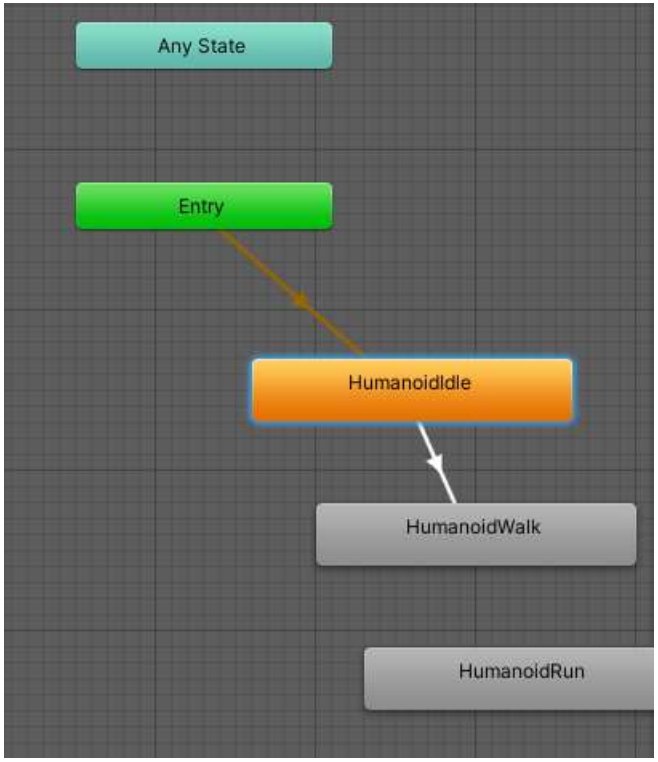
Animator

- Find the “HumanoidIdle”, “HumanoidWalk” and “HumanoidRun” clips
- Drag and drop them into the animator grid
- Depends on your selection order, the box with orange color is the default state
- You can change the default state by right click on a state and choose “set as layer default state”
- Set “HumanoidIdle” as default state
- Now play to see



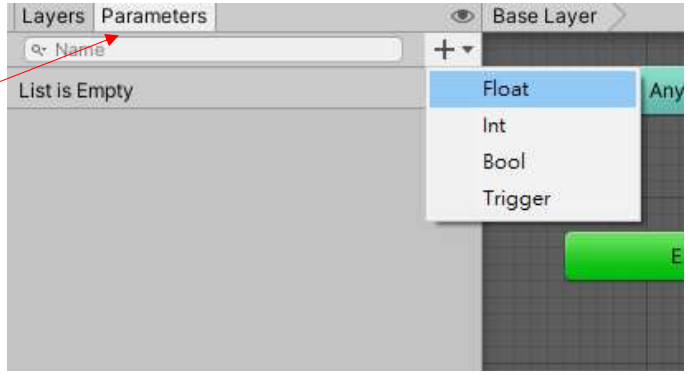
Animator

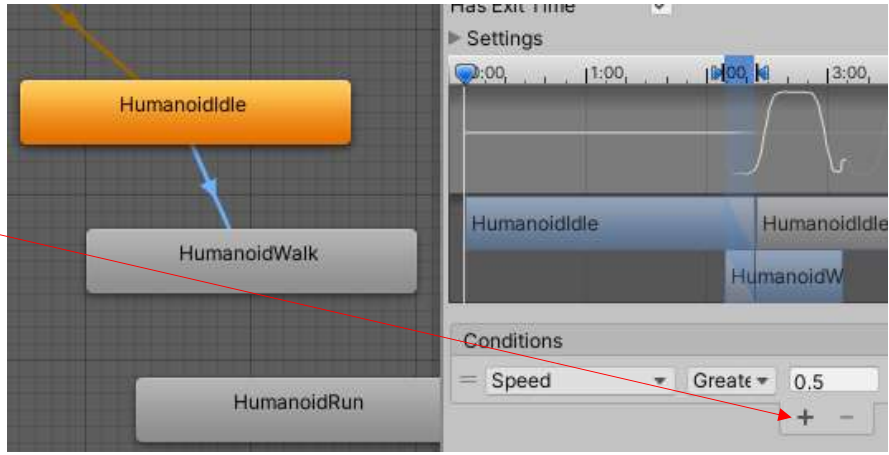
- Go to Animator screen and select HumanoidIdle
- Right click on the state and choose “Make transition”, then drag the line or click on HumanoidWalk
- Play to see the character idle and walk
- You can also do further transition to HumanoidRun to see the effect



Animator

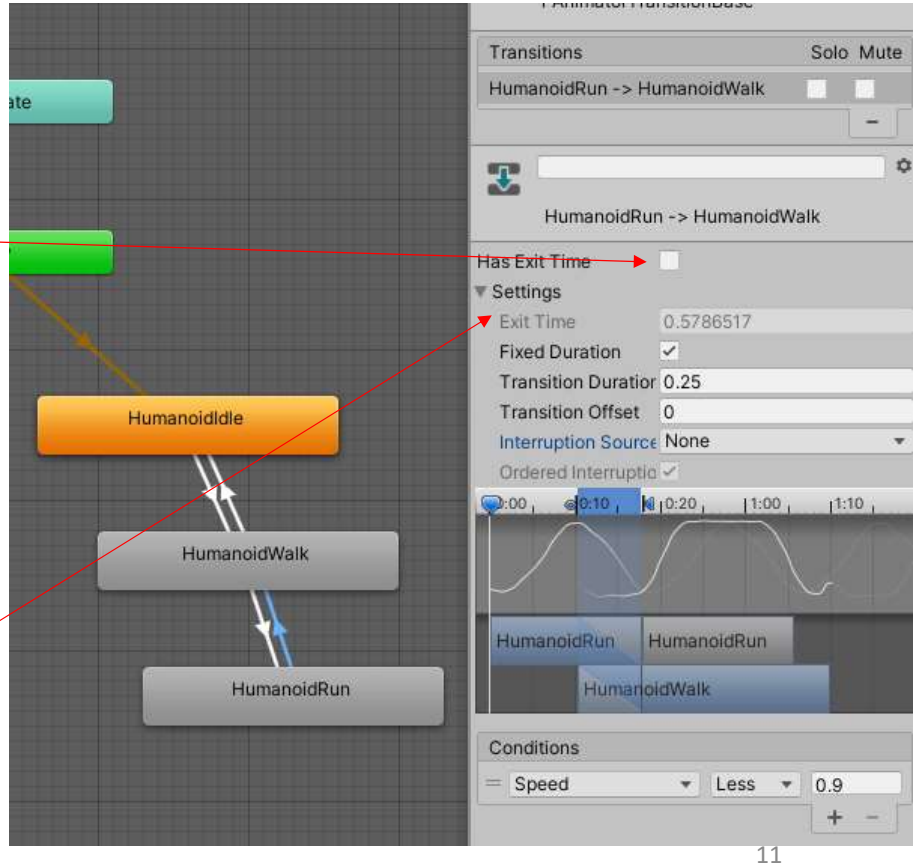
- Now we can animate the character, but more control is needed
- Click the parameter tab and add a “Float” parameter and name it “Speed”
- Click the transition line and add a condition for the transition
- Set the condition to greater than 0.5 for Speed
- Play and change the value of Speed to test (you can dock the Animator tab to Inspector)





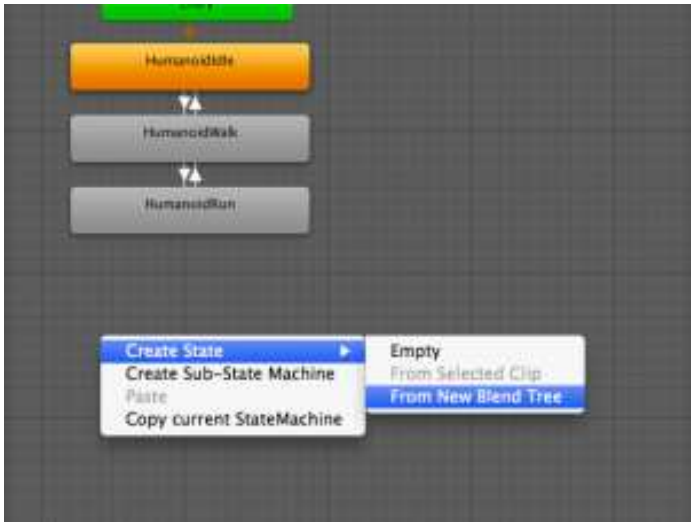
Animator

- Create all transitions between three states as on right
- Uncheck the “Has Exit Time” for all transition – it will allow the transition animation immediately displayed once the control is on
- Make the transition between Idle to/From Walk for Speed 0.5 (Greater / Less) and Walk to/From Run for Speed 0.9 (Greater / Less)
- You can also expand the Settings to experiment with fine control on animation



Blend Tree

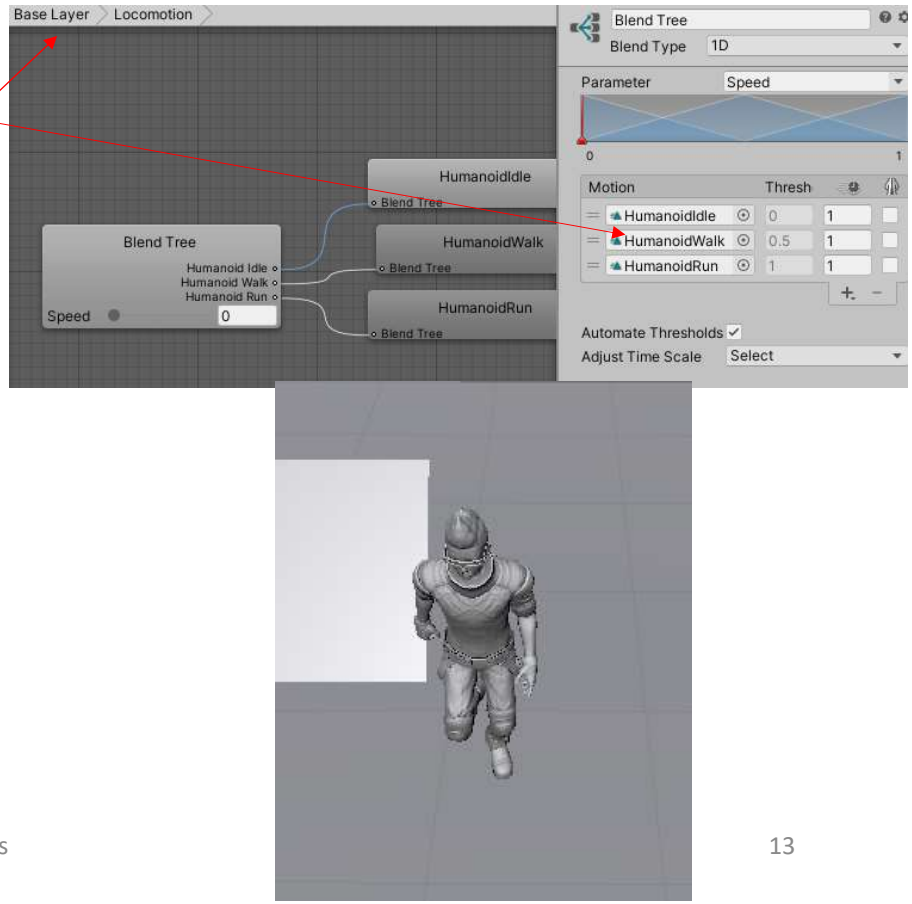
- Now we have a way to control the animation by using a parameter, however it can only be in one direction
- Blend Tree is a more elegant way to handle this
- Right click in animator and choose “Create State/From New Blend Tree”
- Name it “Locomotion”



Blend Tree

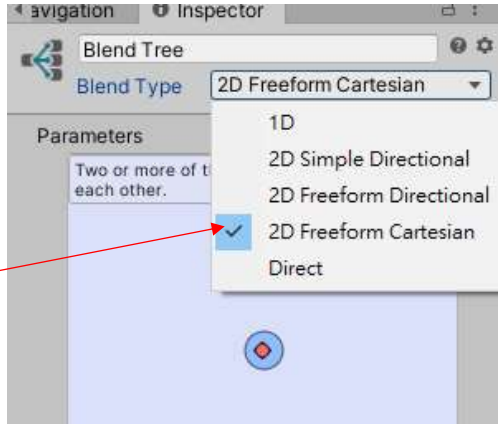
- Double click it and add 3 motion fields
- Drag and drop our three animations same as before to these field entries
- You will see the conditions for transition are automatically set up as 0, 0.5 and 1

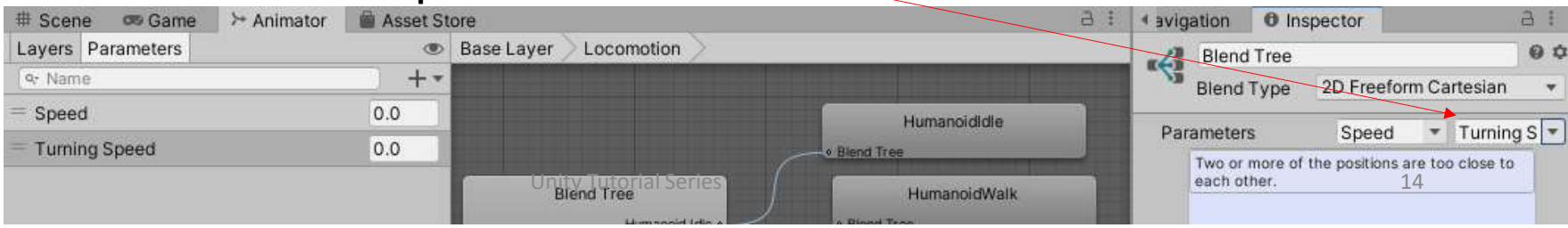
- Go back to the base layer
- Delete all the three states originally
- Make the default transition to our new blend tree and test
- Check the blending effect when Speed is between 0.5 and 1



Blend Tree

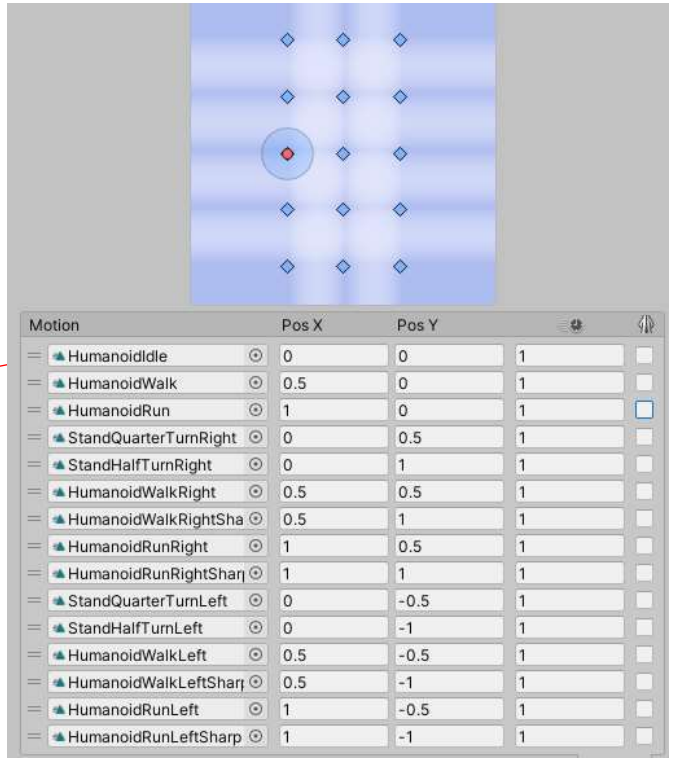
- Still we have only 1-direction movement
- To implement 2D motion (turn left/right) , double click the Locomotion and select Blend Tree
- Change the type of Blend Tree in inspector to “2D Freeform Cartesian”
- It would allow turning in addition to walk and run
- Add a new Float parameter called “Turning Speed” and add it to other parameter slot

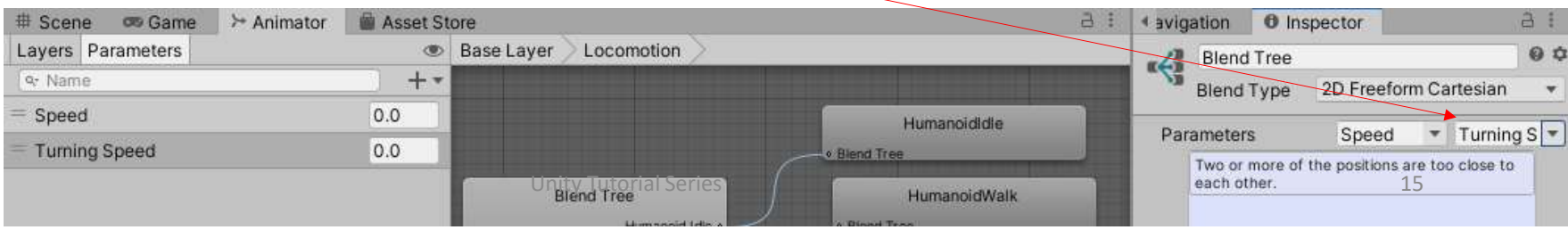




Blend Tree

- Add 12 more motion slots
- Add the following animations:
- “StandQuarterTurnRight”,”StandHalfTurnRight”, HumanoidWalkRight”,”HumanoidWalkRightSharp”,”HumanoidRunRight”,”HumanoidRunRightSharp”,”StandQuarterTurnLeft”,”StandHalfTurnLeft”,” HumaoidWalkLeft”,”HumanoidWalkLeftSharp”,HumaoidRunLeft”and “HumanoidRunLeftSharp”
- Play now to see





Scripting

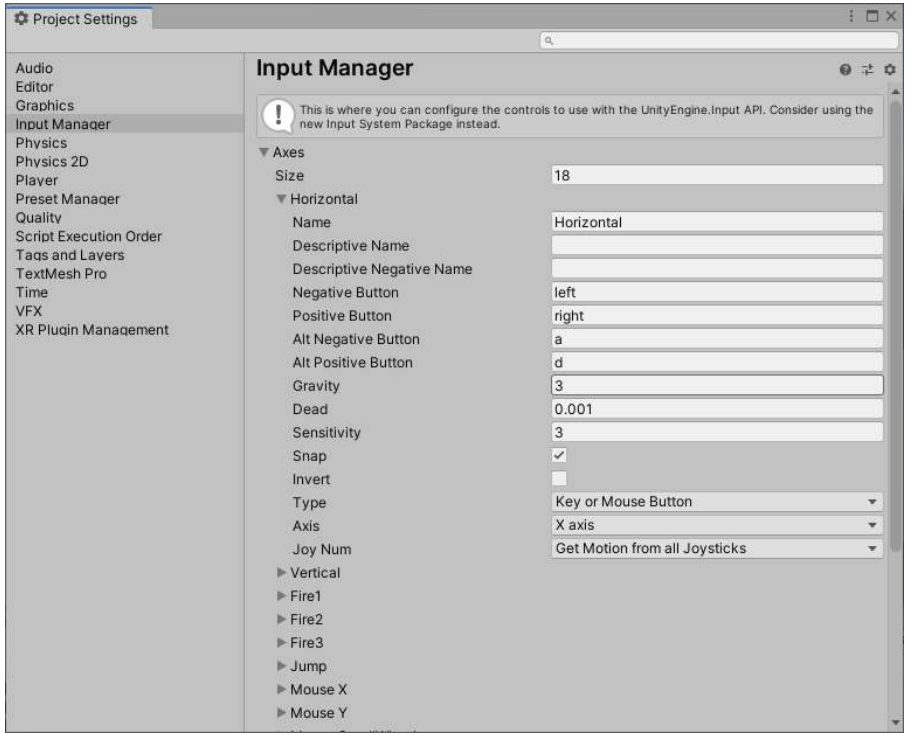
- Add a script called “PlayerController” by choosing Add Component
- Change the Start() and Update method and add a global variable as follow

```
private Animator thisAnim;

 void Start (){
    thisAnim = GetComponent<Animator>();
}
void Update () {
  float h = Input.GetAxis ("Horizontal");
  float v = Input.GetAxis ("Vertical");
  thisAnim.SetFloat ("Speed", v);
  thisAnim.SetFloat ("TurningSpeed", h);
}
```

Scripting

- The keys can be chosen under “Edit/Project Settings/Input Manager”
- Now you can test your character controller



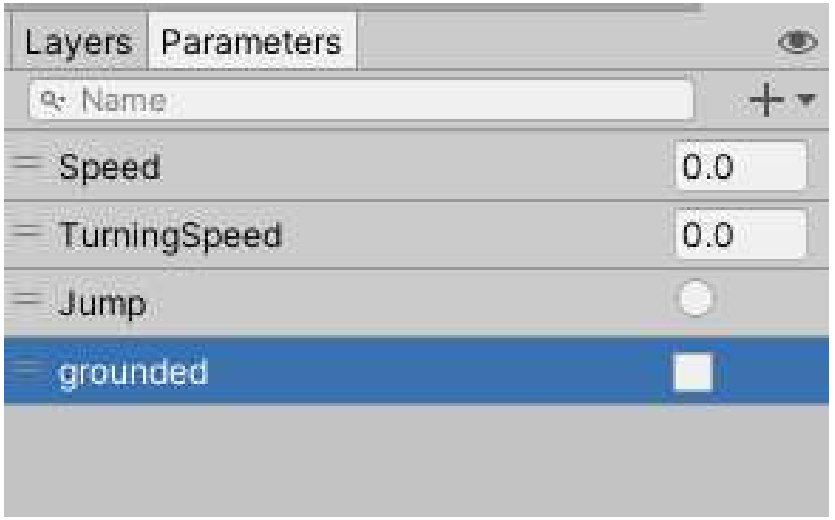
Jump!

- Now our character can walk or run in the game world
- Sometimes we may want a jump action
- To do this, we need a number of things
- For animation, we need a jump up pose, a fall pose, and landed pose, and a way to check if we hit the ground

- Finally we need to create a sub-state machine
- It puts a number of animations together and acts as one

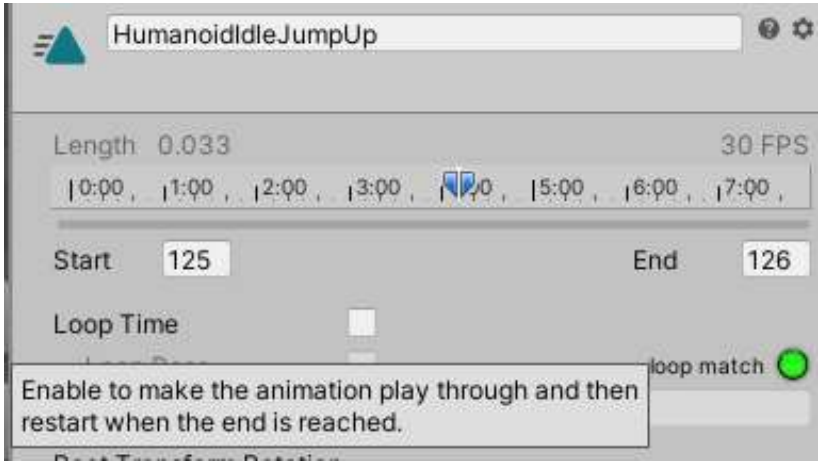
Jump!

- Create a trigger in animator called “Jump”
- Then create a Boolean called “grounded”
- Using filter search, found
  - “HumaoidCrounchIdle”
  - “HumanoidFall”
  - “HumanoidJumpUp”



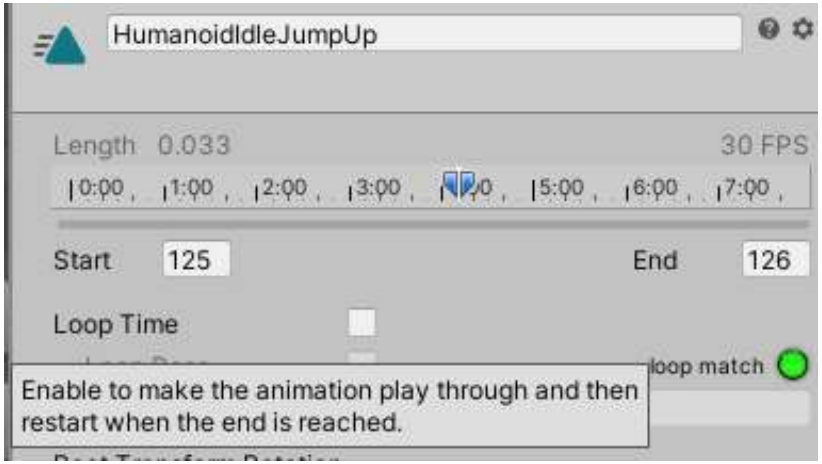
Jump!

- For “HumanoidJumpUp”, the time is too long
- Select the animation clip, click “Edit”
- Change it to only 1 frame long (125-126 for JumpUp)
- Click “Apply”
- Do the same for the others



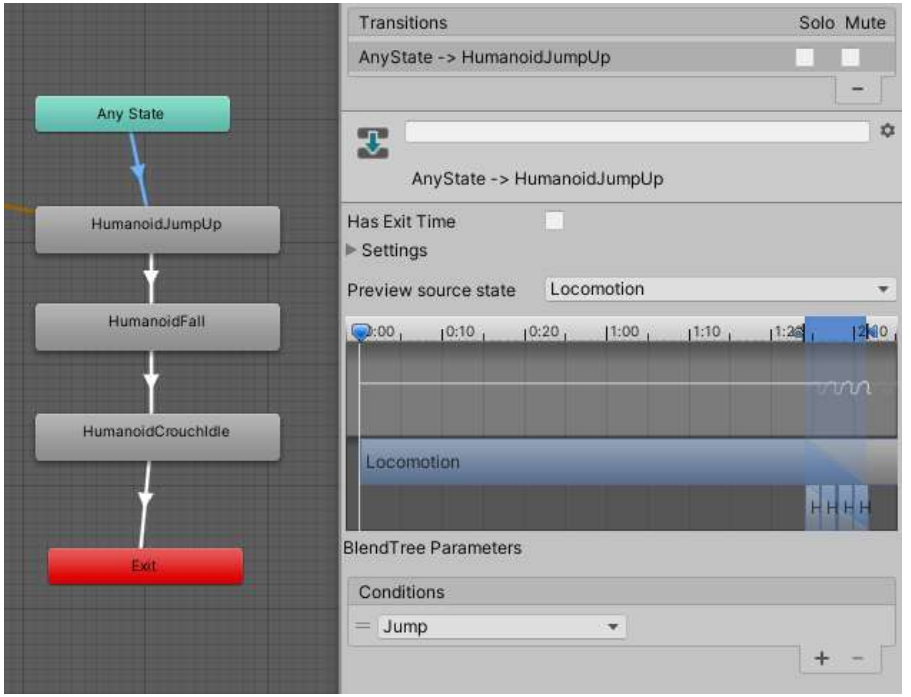
Jump!

- Go to Animator and choose Base Layer
- Right click in animator and choose “Create Sub-State Machine”
- Name it “Jump”
- Double click the “Jump”
- Drag in the three animations



Jump!

- From “Any State” make a transition to “HumaoidJumpUp”
- Set the condition to “Jump” trigger
- Set up further transition from “JumpUp” to “Fall”
- Then transition from “Fall” to “CrounchIdle”
- Set the condition of Fall to CorunchIdel be “grounded” true
- Because we don’t know how far the body will fall

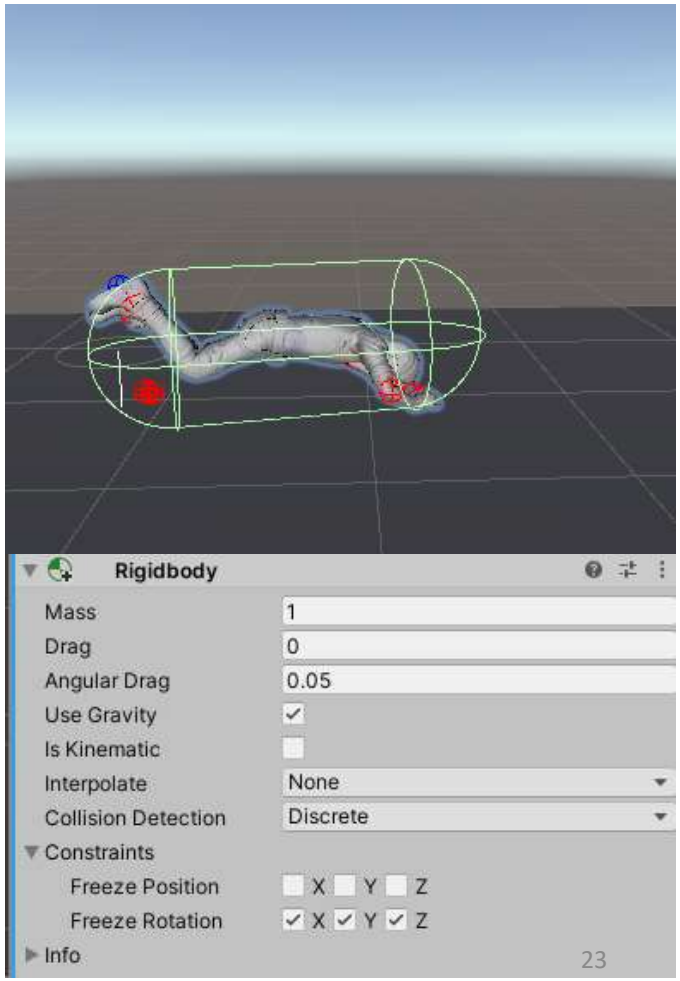


Jump!

- We have the animator set up done
- Go to script editor
- Add the following line to Start() at end

```
thisAnim.SetTrigger ("Jump");
```

- Add “Rigidbody” and “Capsule Collider” components to the “Ethan”
- Now test play
- To prevent this, set constraint of rigid body to freeze rotations at x,y, and z



Jump!

- Add the follow variables to global area of PlayerController

```
private Rigidbody rigid;
public float JumpForce = 500;
```

- Replace the SetTrigger statement in Start with the following

```
rigid = GetComponent<Rigidbody>();
```

- Add the following lines to the end of Update()

```
if (Input.GetButtonDown ("Jump")) {
        rigid.AddForce (Vector3.up * JumpForce);
        thisAnim.SetTrigger ("Jump");
    }
```

Jump!

- Within Update(), we need to check whether we are at ground all the times
- Add the following statements to Update()

```
if (Physics.Raycast (transform.position + (Vector3.up * 0.1f),
        Vector3.down, groundDistance, whatIsGround)) {
  thisAnim.SetBool ("grounded", true);
  thisAnim.applyRootMotion = true;
} else {
   thisAnim.SetBool ("grounded", false);
}
```

Jump!

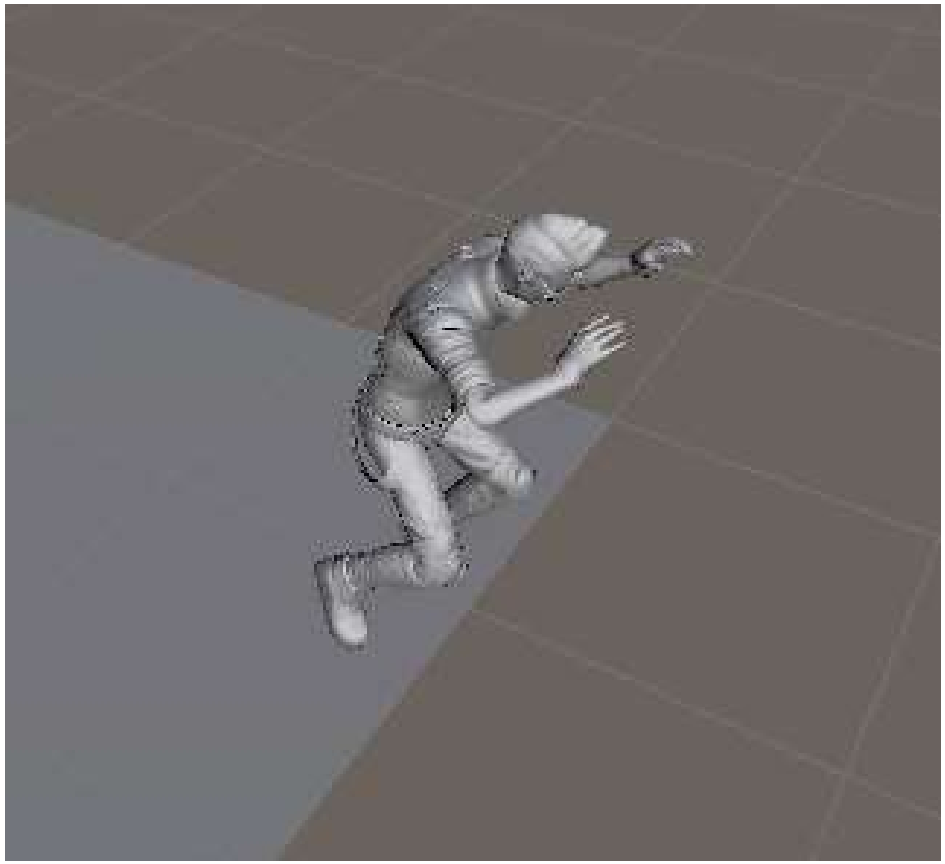
- A float parameter needed to check how close we are to ground
- Unity allow you to filter out those game objects that you don't want to check with by a LayerMask
- Here we define Default layer is thelayer where we want to check with i.e. ground.
- So add the following lines to the global section of PlayerController

```
public float groundDistance = 0.3f;
public LayerMask whatIsGround;
```

```
public class PlayerController : MonoBehaviour {
    private Animator thisAnim;
    private Rigidbody rigid;
    public float groundDistance = 0.3f;
    public float JumpForce = 500;
    public LayerMask whatIsGround;
    void Start()    {
        thisAnim = GetComponent<Animator>();
        rigid = GetComponent<Rigidbody>();      }
     void Update()   {
        float h = Input.GetAxis("Horizontal");
        float v = Input.GetAxis("Vertical");
        thisAnim.SetFloat("Speed", v);
        thisAnim.SetFloat("TurningSpeed", h);
        if (Input.GetButtonDown("Jump"))        {
            rigid.AddForce(Vector3.up * JumpForce);
            thisAnim.SetTrigger("Jump");        }
        if (Physics.Raycast(transform.position + (Vector3.up * 0.1f), Vector3.down, groundDistance, whatIsGround))       {
            thisAnim.SetBool("grounded", true);
            thisAnim.applyRootMotion = true;        }
        else  {
            thisAnim.SetBool("grounded", false);        }
    }
}
```

Jump

- Go to scene editor and set the LayerMask “WhatIsGround” of PlayerController to “Default”
- Now test to see our character jumping performance!

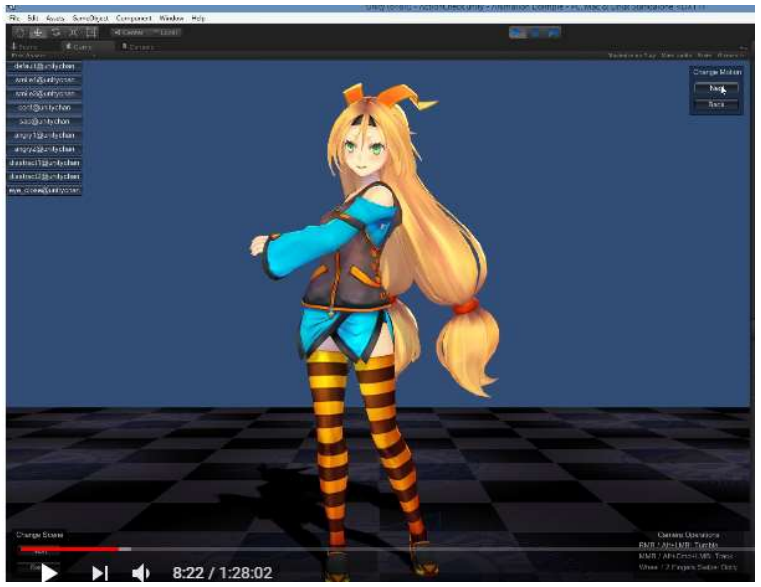


- Our tutorial is based on this excellent tutorial:

https://gamedevacademy.org/unity-animator-tutorial/

Motion Retarget

- First create a new project in Unity
- Import Unity-Chan into your project
- Open the scene “Action Check” and run
- Click the “Next”button to see various animations provided

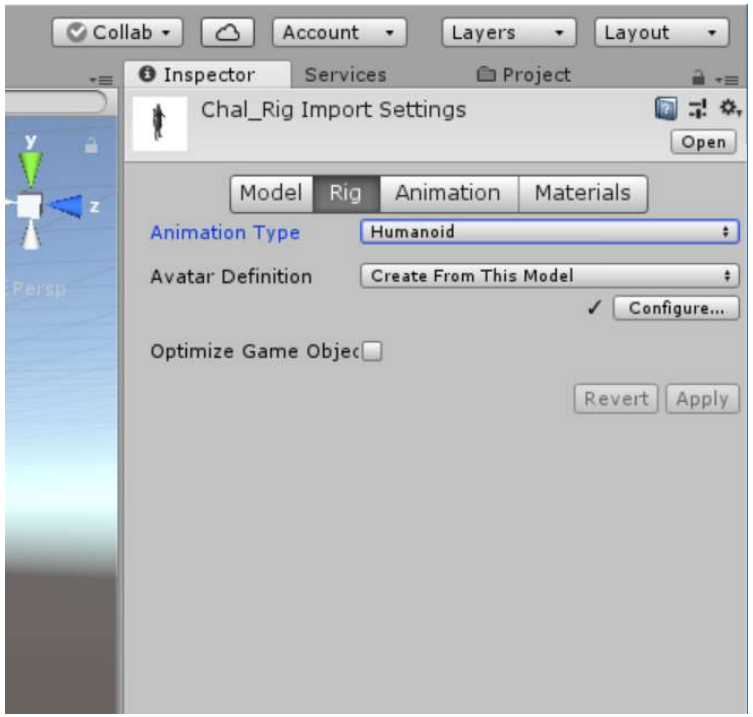


Challenger

- Now create a new scene
- Import the Blacksmith package
- Go to “Art/Charactes” folder and drag the “Challenger_prefab” into the scene
- Adjust the camera and check the challenger scene

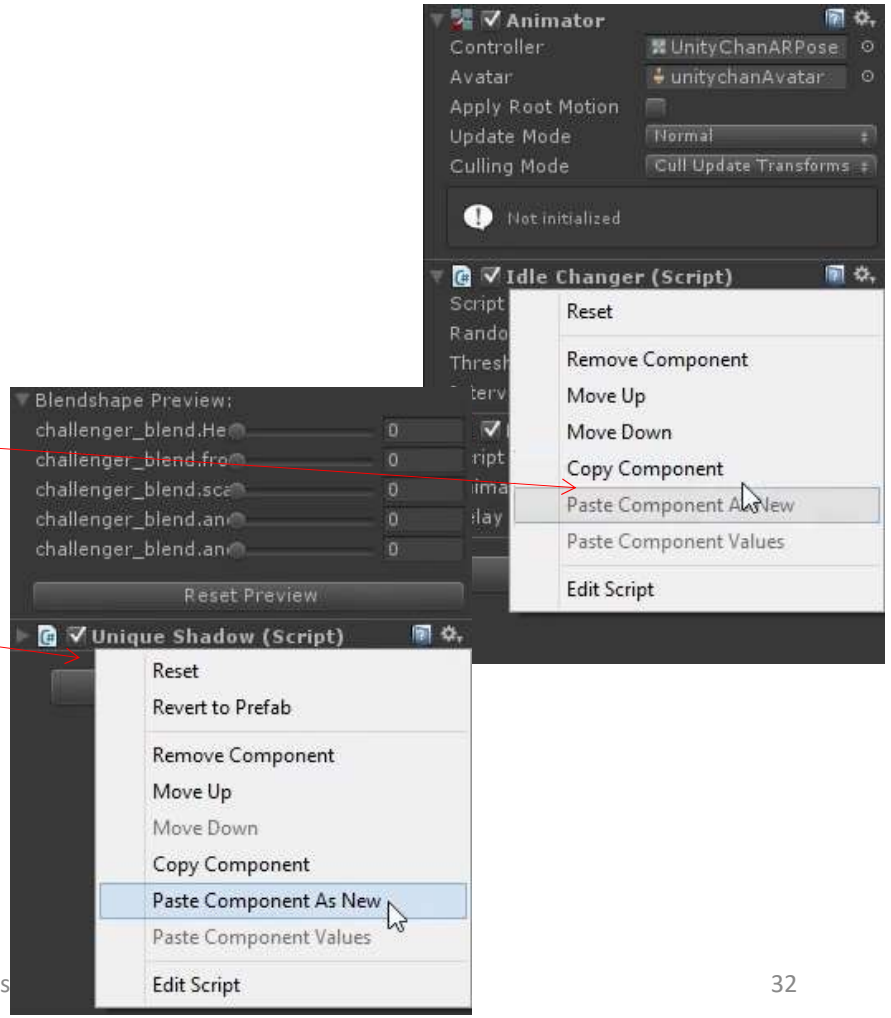
Configure Rig

- Go to project explorer under “Assets/Art/Character/ Challenger”, select the chal_Rig
- In Inspector click “Rig”, choose the “Animation Type” and change it from Generic to Humaniod
- Click “Apply”
- Now the bone system is compatible with that of Unity-Chan



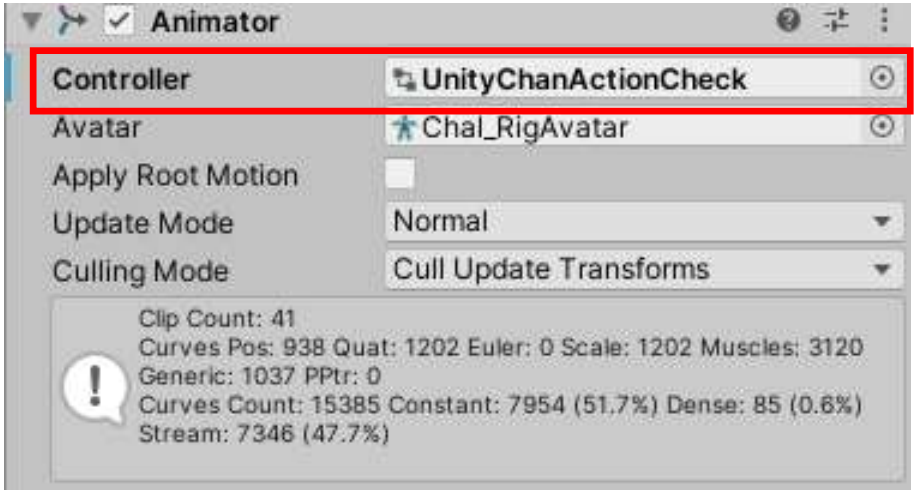
Motion Retargeting

- Go to “Unity-Chan/Model/Prefabs” folder
- Click UnityChan prefab
- In Inspector choose the “Idle-Change” script and select the “Copy Component”
- Go back to select Challenger prefab in scene
- Right click and paste the component to the character



Motion Retargeting

- With the challenger still selected, go to Unity-Chan folder in project explorer
- locate the “UnityChanActionCheck” controller (under Art/Animations/Animators)
- Drag the controller to replace that in Challenger

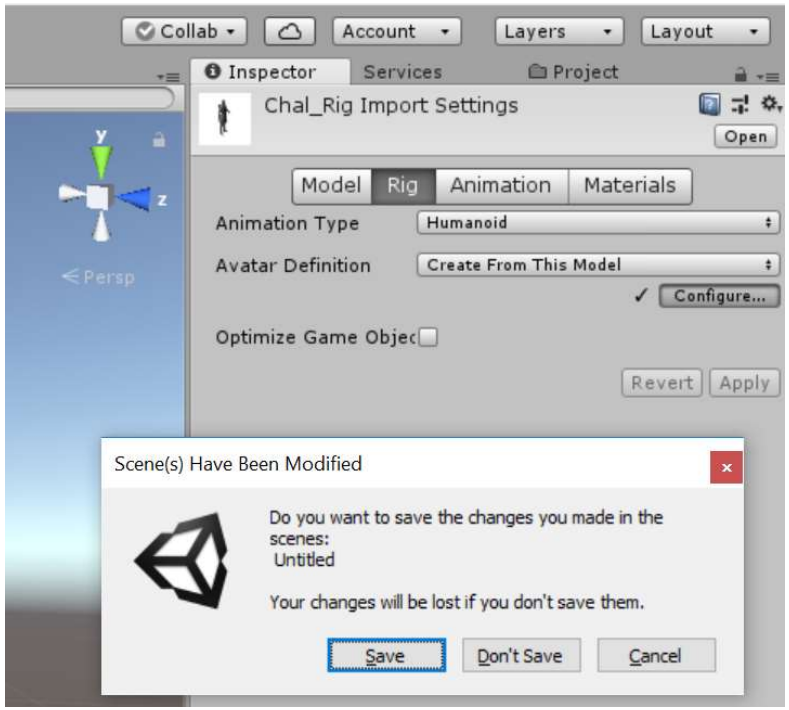


Motion Retargeting

- Now run the scene to see the new animations controlling the Challenger character
- This is “Motion retargeting” - reuse animations on possibly different rigs

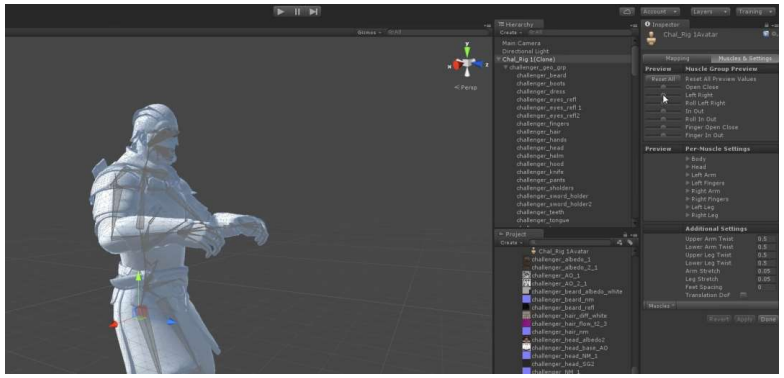
Muscles Based Animation

- Now closed the scene and create a new scene
- Drag the challenger_prefab to it
- Click on the project tab and select Challenger_rig, under the rig menu, choose “Configure”
- It will ask you to save the current scene, choose save



Muscles Based Animation

- Click the Muscles and settings tab
- You can twitch the muscle settings to see the effect
- Now change the muscle setting to as right
- We reduce the movement of left arm to compare with right
- Click Apply at bottom



Muscles Based Animation

- Now go back to original scene and run the scene with the new setting of the rig
- Compare the left and right arm motion
- Left arm now move much smaller extent than the right one

Reference

- This tutorial is derived from this live training: https://www.youtube.com/watch?v=wf6vtCgLk6w