Humanoid Basics

Third Person Controller

Humanoid Basics is an easy to setup Third Person Controller for humanoids with the basics features like:

-Walk
-Run
-Crouch
-Jump
-Climb
-Aim
-Switch aim direction
-Lean while aiming
-Shoot

Including:

-Easy player setup
-Automatic Ragdoll
-Create new weapons

SUMMARY

FIRST THINGS FIRST

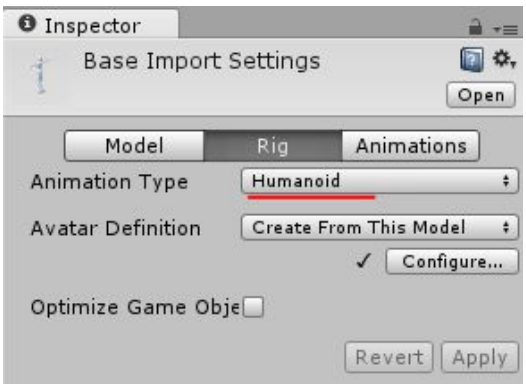
Let’s add some game object **tags** to your project:
Go to **Edit > Project Settings > Tags and Layers** and then add a tag called “**Climbable**” and “**Weapon**”.

NOTE

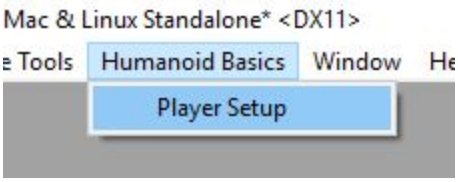
You probably don’t need to do this, as the package import all the Project Settings, but make sure if the mentioned tags exists.

SETTING UP THE PLAYER

Make sure that your FBX model is an humanoid:

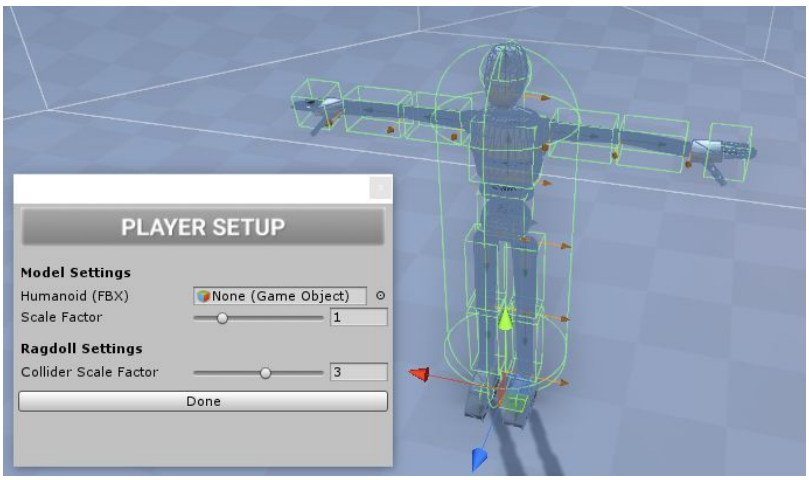


Then go to Humanoid **Basics > Player Setup:**



It will automatically instantiate a Player if there isn't one in the scene

Now drag and drop **your character into** the **Humanoid (FBX) field**.



Adjust the Scale Factor so the Model fits in the Capsule Collider.
The scale of the bones colliders can be adjusted with **Collider Scale Factor**:

NOTE

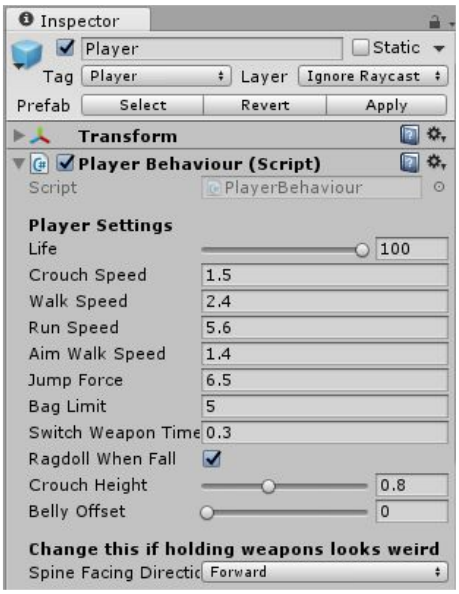
Both the **Scale Factor** and the **Collider Scale Factor** are **only applied when** the **"Done"** button **is pressed**.

IMPORTANT

To make the character climb things, you need to **select the game object** you want to make the character climb and **tag** it to "**Climbable**".

The Player Inspector:

PLAYER SETTINGS:



CONTROLLER SETTINGS:

CAMERA SETTINGS:

NOTE

-In the Player Settings

If the **weapons are to close** to the character's belly, **increasing the "Belly Offset"** should **fix it.**

-In the Controller Settings

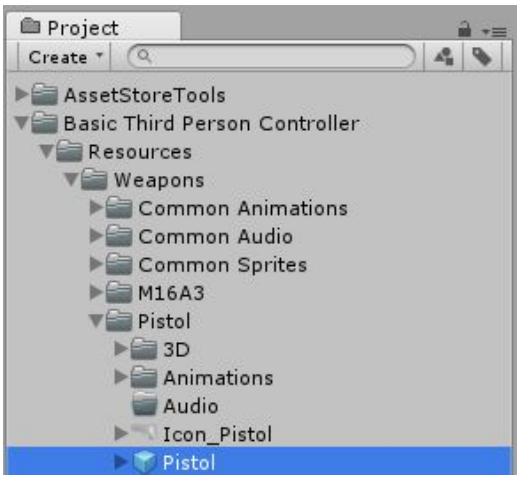
At the moment **only the keyboard and mouse are supported.**

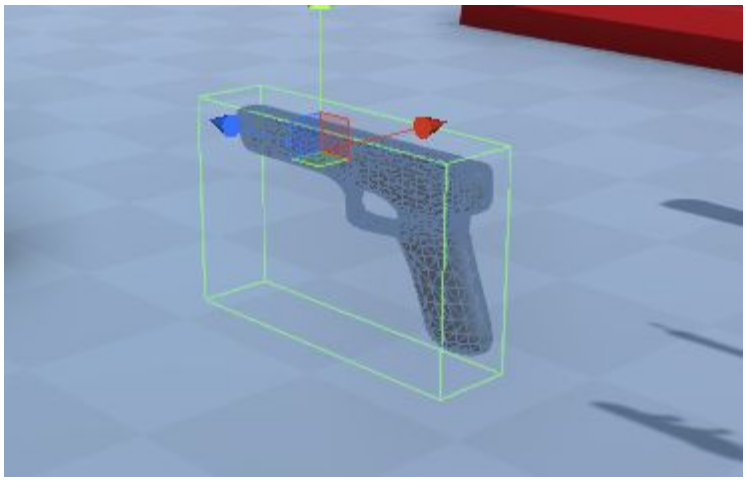
-In the Camera Settings

Decreasing the Recoil influence factor the Camera will feel more the current weapon recoil.

ADDING NEW WEAPONS

Your new weapons will be based on the existing ones. So, if you want to add a new Pistol, go to **Resources > Weapons**, find the Pistol, **drag and drop to the scene** to edit:

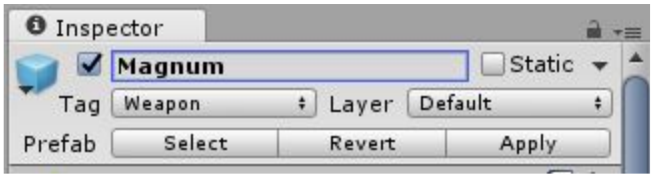




Now let's make some changes:

Creating a new prefab

First, **rename the game object to what you want** to make, in my case, i want a **"Magnum":**



For better organization, lets create a folder for the new weapon and put the game object there:

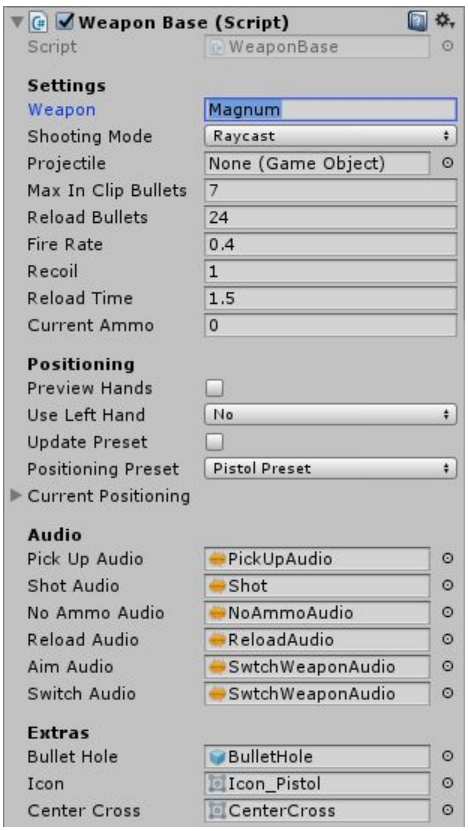
🔫Editing the weapon settings

With your new weapon selected, check the **Weapon Base** in the inspector. As my weapon is based in a pistol, i need to **change the name of it in the Settings:**

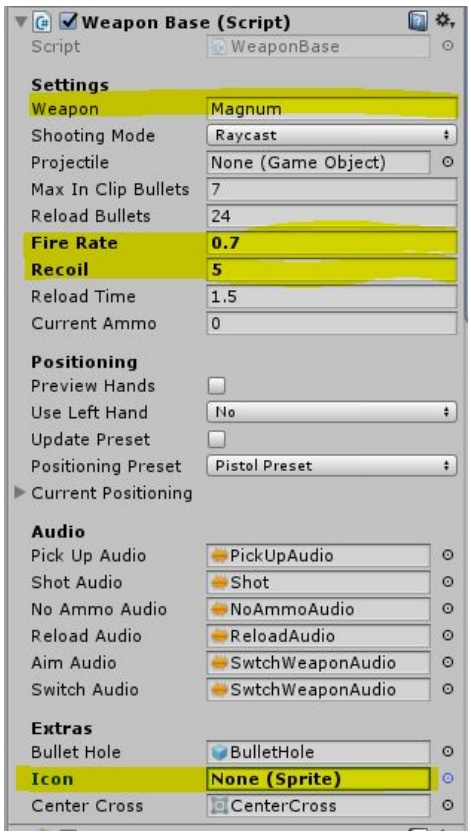
There are various of variables you can change, in my case i want to:

-Increase the Fire Rate (It acctually makes slower)
-Increase the Recoil
-Remove the Icon in the Extras (The current icon is of the pistol)

-BEFORE



-NOW



NOTE
It's fine to leave it without an icon

IMPORTANT
The "Weapon" variable needs to be different of the others

UPDATE

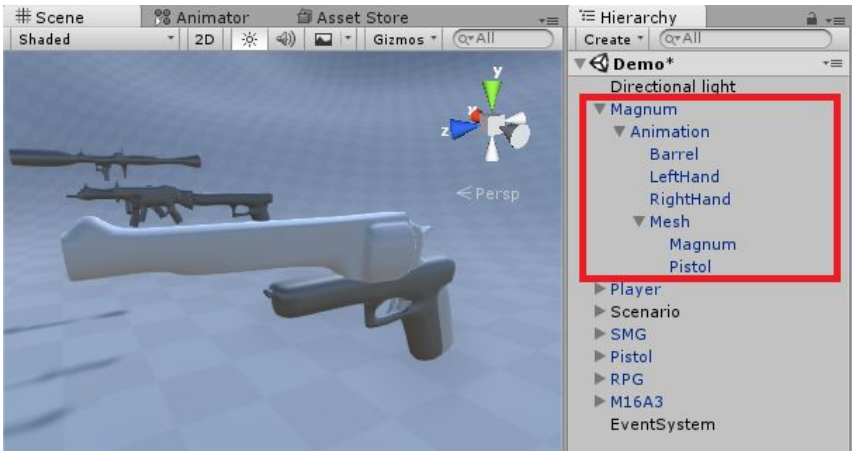
A new option was added, that allows you to make shotguns

Changing the 3D Model

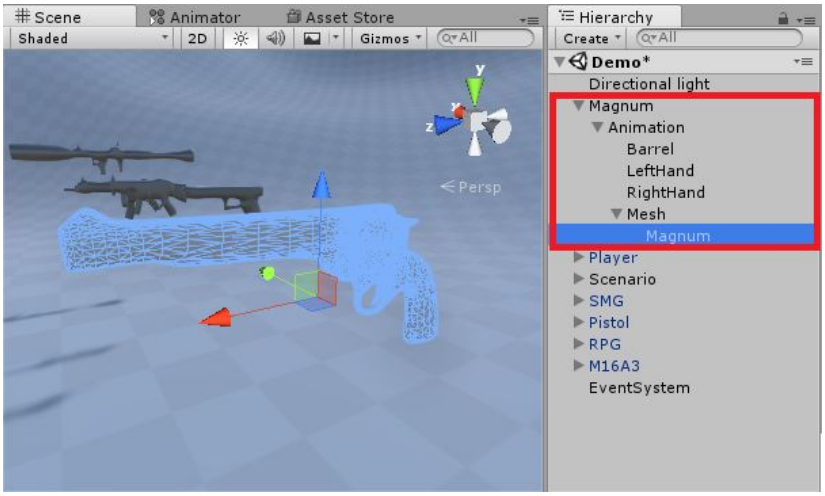
For better organization **create a folder** called **"3D"** just for the new model:

Go to your **scene** and open the **hierarchy** of your weapon like this **(*weapon_name*/Animation/Mesh)**:

Drag and drop the new weapon **FBX into** the **Mesh** and **align** them:

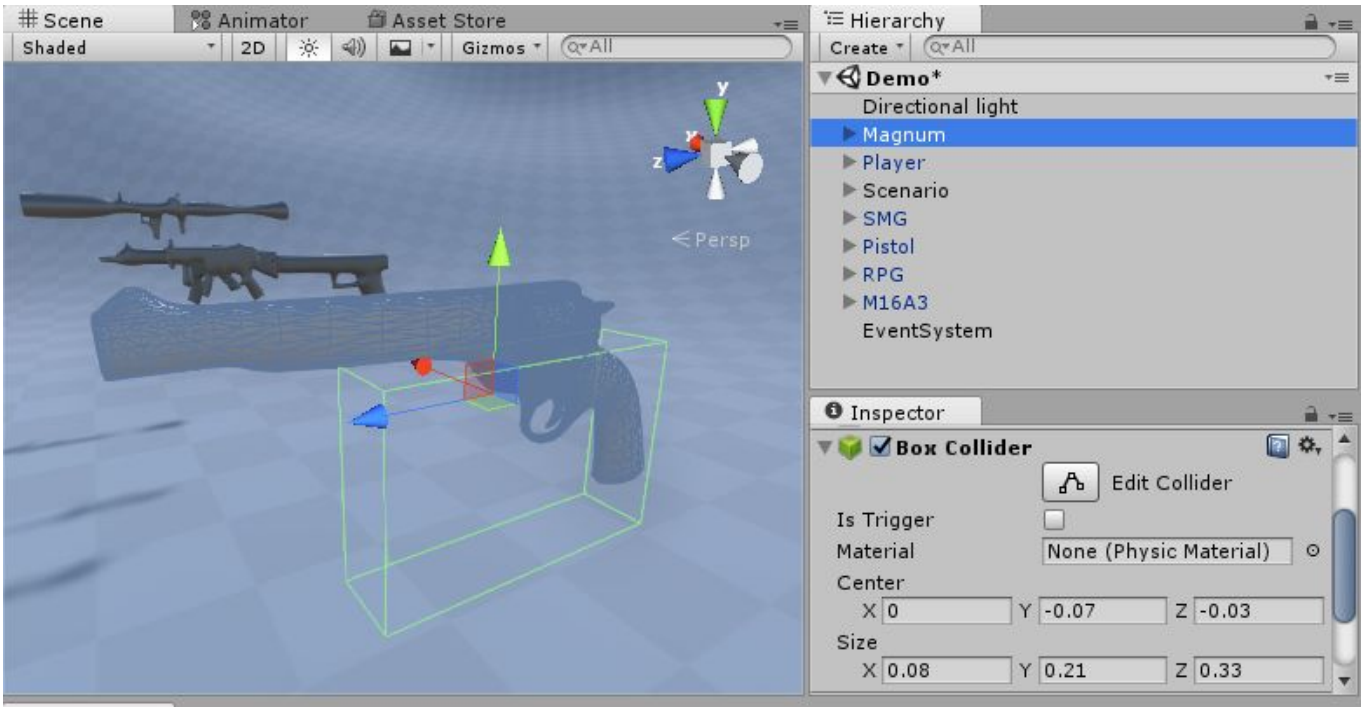


Now you can delete the old model:

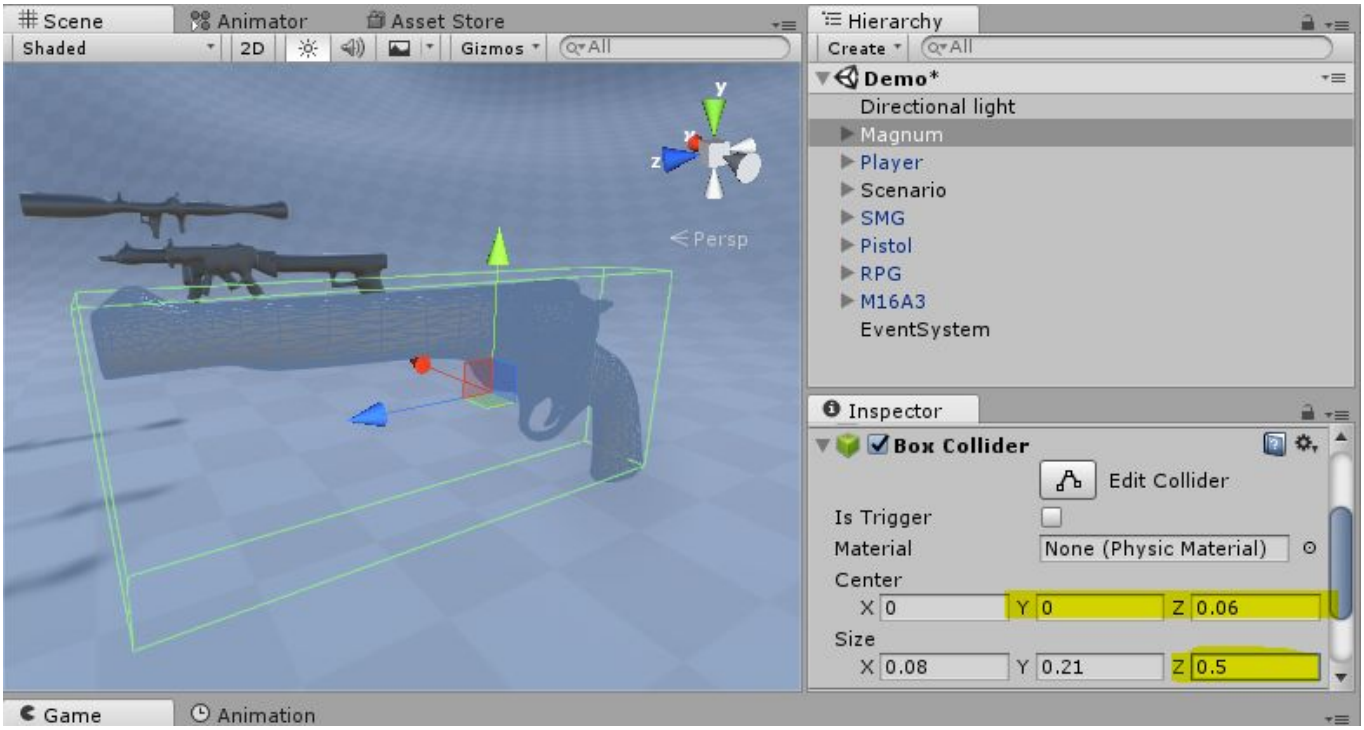


Adjusting the collider

Every weapon needs a proportional collider for them, so, as i based my magnum on a base pistol, i need to adjust the box collider:

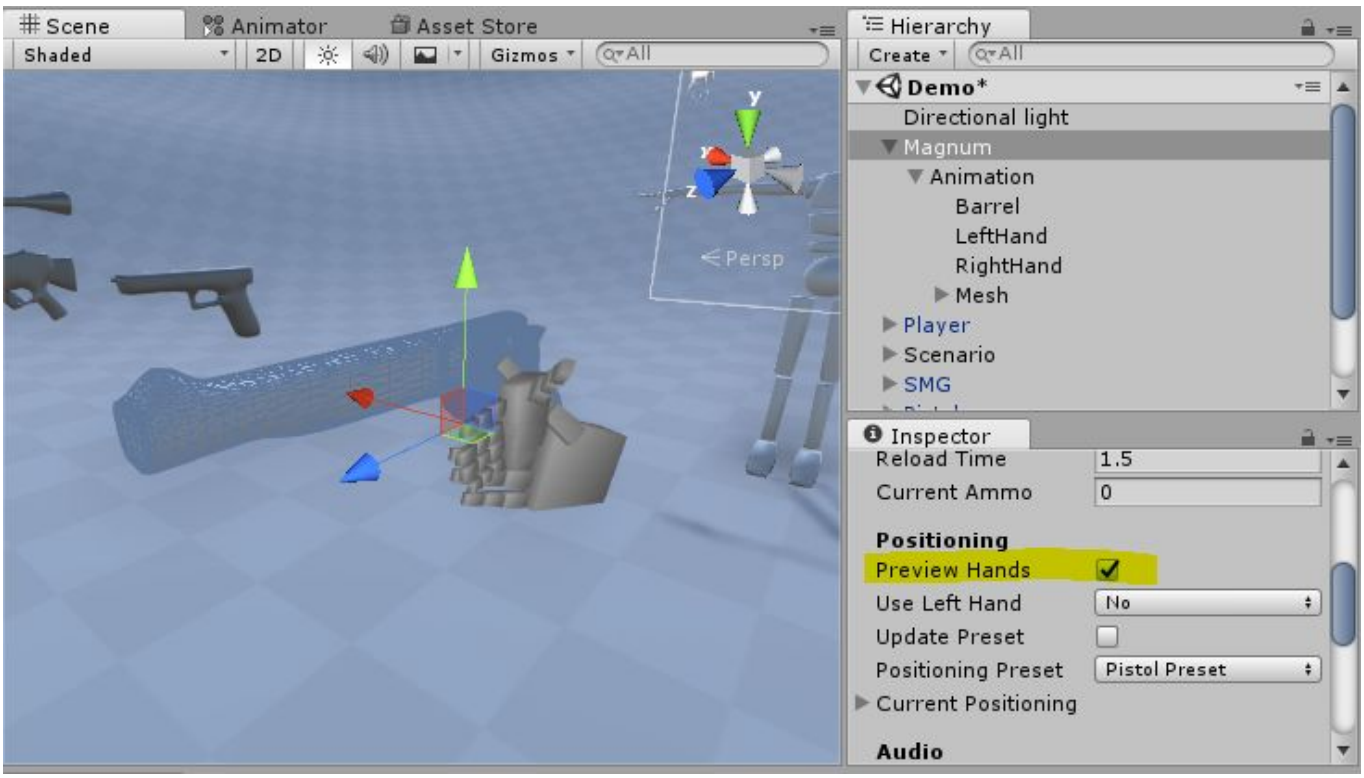


With the values adjusted:



Adjusting the hands position

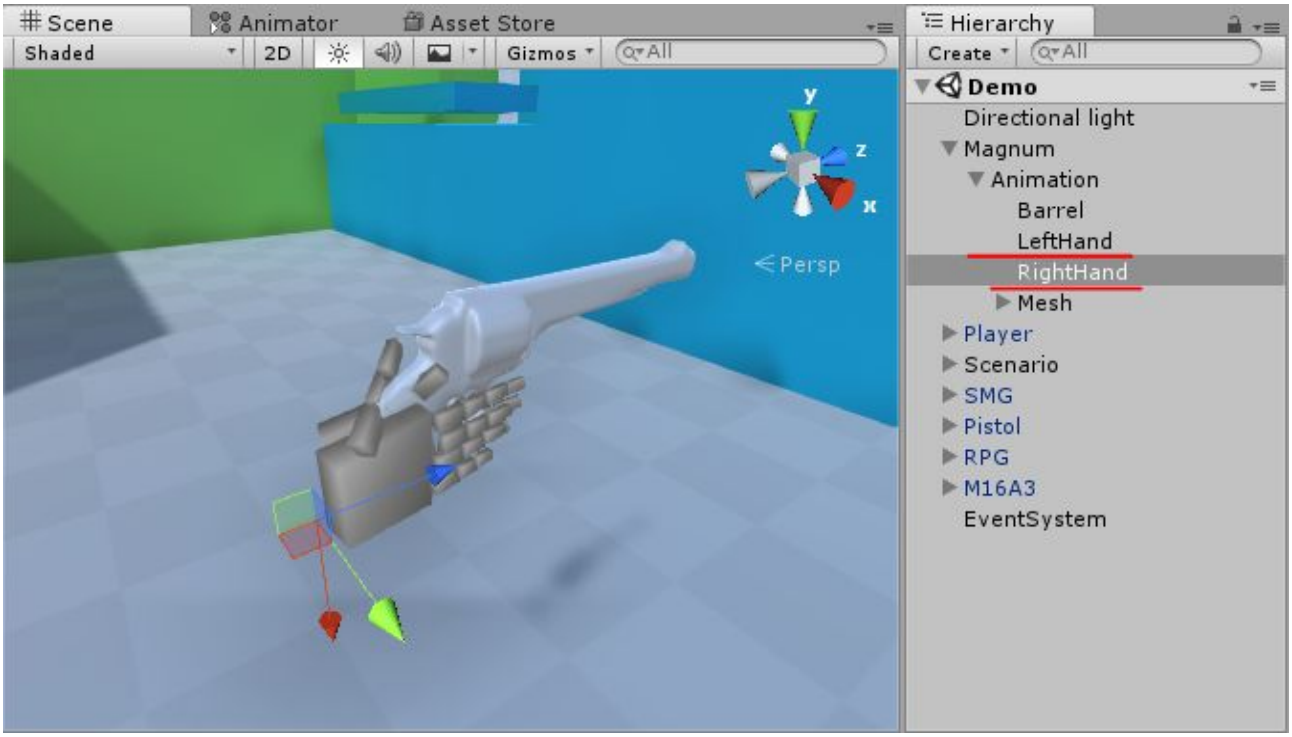
With the weapon selected, go to the "**Weapon Base**" in the **inspector** and check the box "**Preview Hands**", it will show how the hands will be holding the weapon:



NOTE

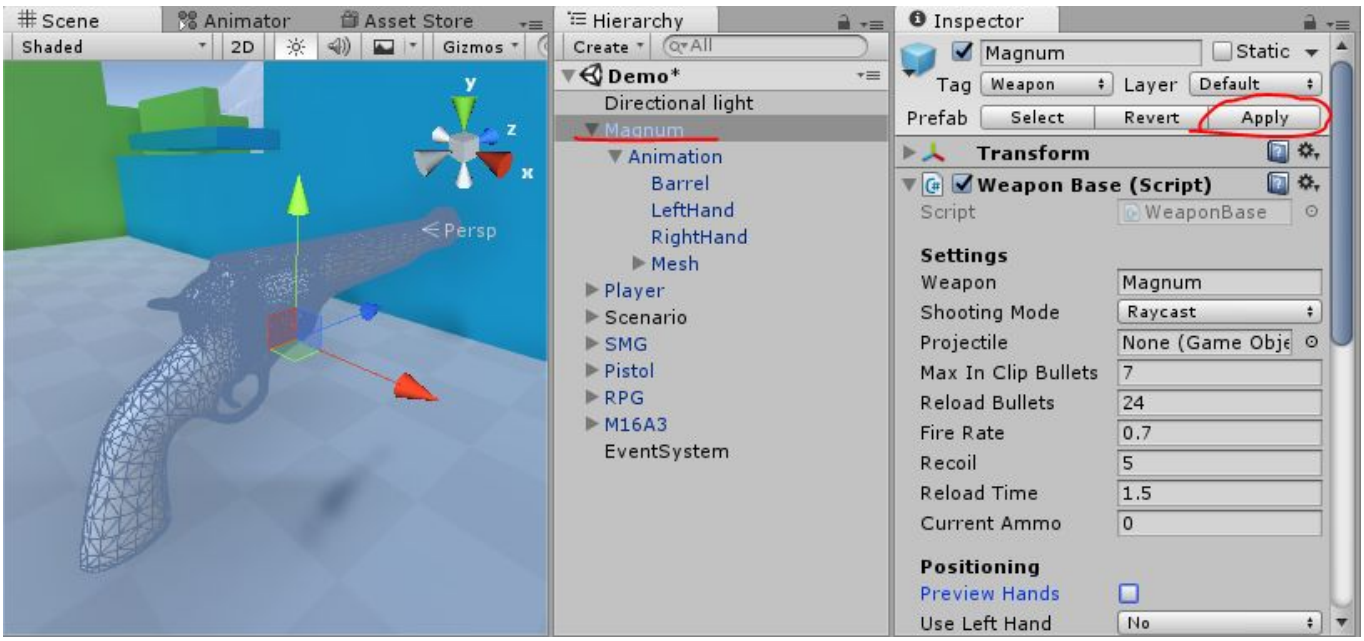
It may be a little confusing, but it's good to have a reference.

In my case, i don't need to make any changes in the hands, i think it fits nice, but if you need to, just **select the hand you want in the hierarchy**, and **change** the **position/rotation** as needed:



Finishing

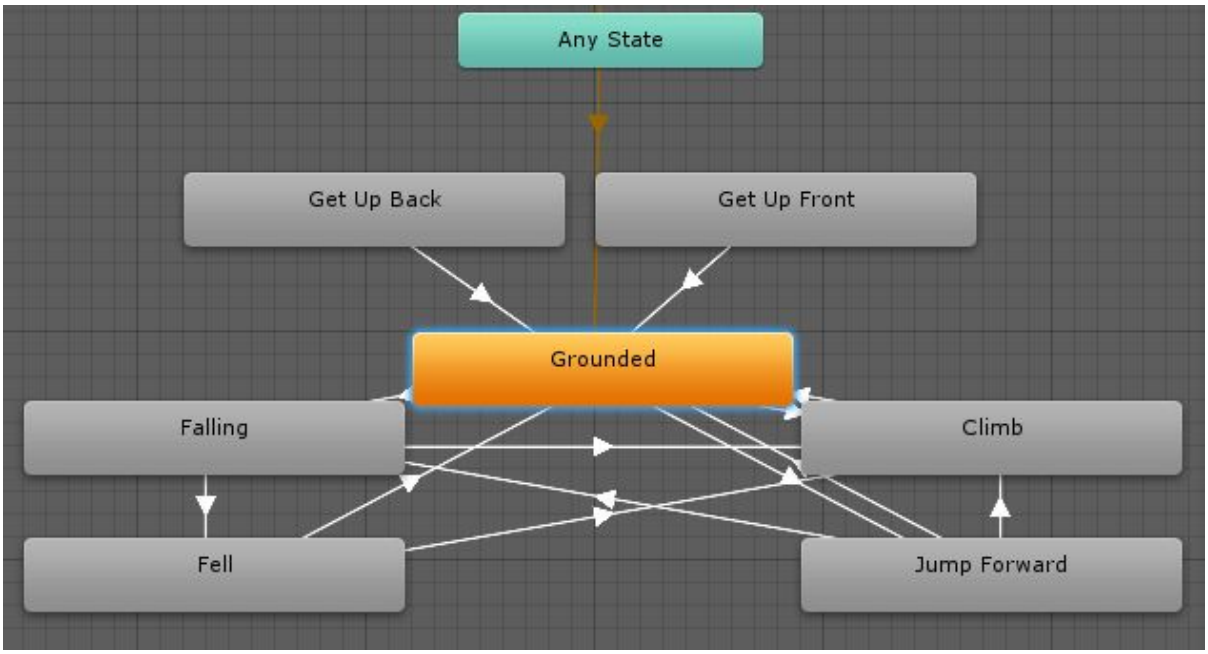
Almost finishing! You tested it and think **it's good?** Just **select the game object and hit Apply!**



Want to change the shooting animation? Follow this tutorial:
https://youtu.be/fabB6vuTnqI?t=2m46s

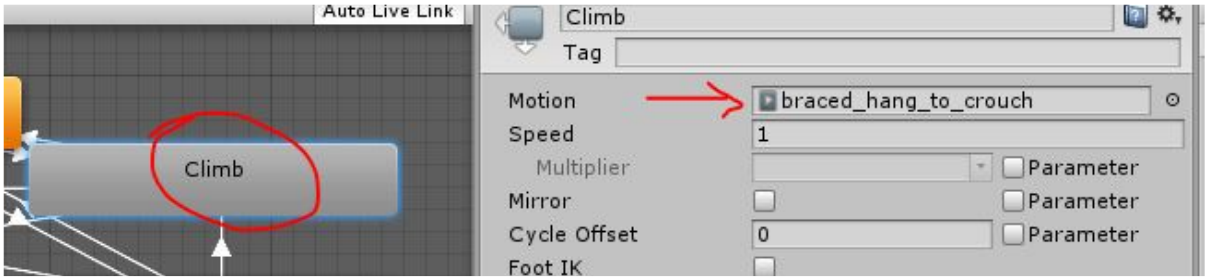
CHANGING CHARACTER ANIMATIONS

Our Animator looks like this. There is no mistery here.

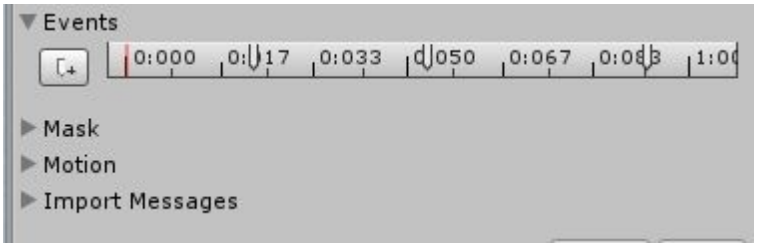


-CHANGING THE CLIMB ANIMATION (NOT RECOMENDED):

With your climb animation in hands, go to the “**Player Animator**”, click on “**Climb**” and change the “**Motion**” field to your desired animation.



Our Climb system takes advantage of the **Animation Events.** So you need to set some events in your animation. To do it, click on your animations and go to **Events:**

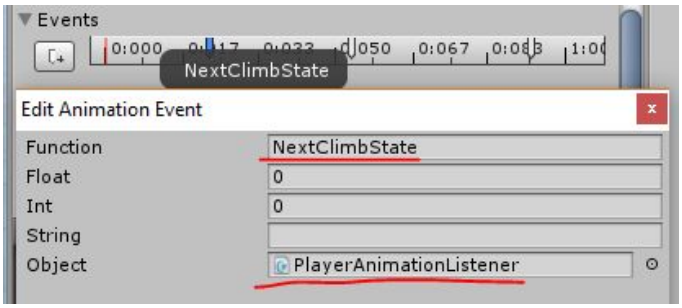


IMPORTANT
Your animation MUST HAVE 3 EVENTS like the image above:

Follow the Steps to make it right:

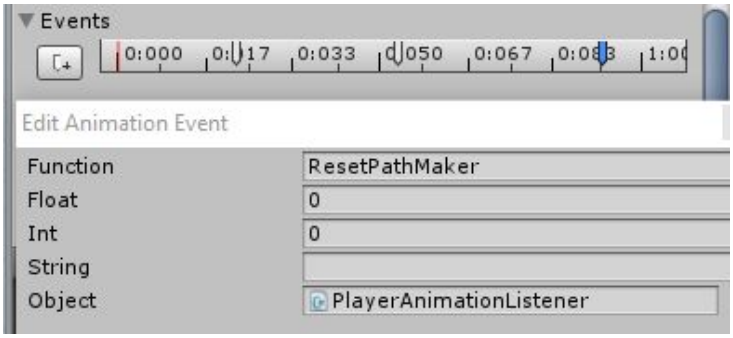
1- The 1° and 2° Event must have this configuration, the 1° on the begginning and the 2° in the middle:

The **Function** it will call: **NextClimbState**;
The **Object**: **PlayerAnimationListener**.



2- To finish the process, at the end of the animation you need to add the last event with the following configuration:

The **Function** it will call: **ResetPathMaker**;
The **Object**: **PlayerAnimationListener**.



DEFAULT CONTROLLS

MOVE	W A S D
RUN	LEFT SHIFT
CROUCH	C
JUMP	SPACE BAR
CLIMB	SPACE BAR
SHOOT	LEFT CLICK
AIM	RIGHT CLICK
SWITCH AIM SIDE	T
RELOAD WEAPON	R
PICK UP WEAPON	E

HANDLING SHOT HITS

Basically, when a weapon that uses **Raycast (Pistol, Rifle, SMG)** hits some collider, this function in the "**WeaponBase**" **(located at Resources/Scripts/Weapon)** called, you can do your stuff in it:

```
void HandleHit(RaycastHit h)
{
    //your code
}
```

If the weapon uses a **Projectile (like the RPG)** hits some collider, a function in "**ProjectileBase**" **(located at Resources/Scripts/Weapon)** is called, you can code there:

```
void OnCollisionEnter()
{
    //get all the colliders inside the radius
    Collider[] collider = Physics.OverlapSphere(transform.position, explosionRadius);

    for(int i = 0; i < collider.Length; i++)
    {
        //

        //your code


        //
        Rigidbody r = collider[i].GetComponent<Rigidbody>();
        if (r)
        {
            PlayerBehaviour pB = r.GetComponent<PlayerBehaviour>();

            if (pB)
            {
                pB.Damage(damage / Vector3.Distance(transform.position, pB.transform.position));
                if (!pB.ragdollh.ragdolled)
                {
                    pB.ToggleRagdoll();

                }
            }
            r.AddExplosionForce(explosionForce, transform.position, explosionRadius);
        }
    }
```