Link to Lab Drive

https://drive.google.com/drive/folders/1jgFTWCJ5JD-JnIY7n_L7s2n2vA1F9nNX?usp=sharing

OR

https://tinyurl.com/ta-comp476-daniel

Movement Using Root Motion

3D Rigging

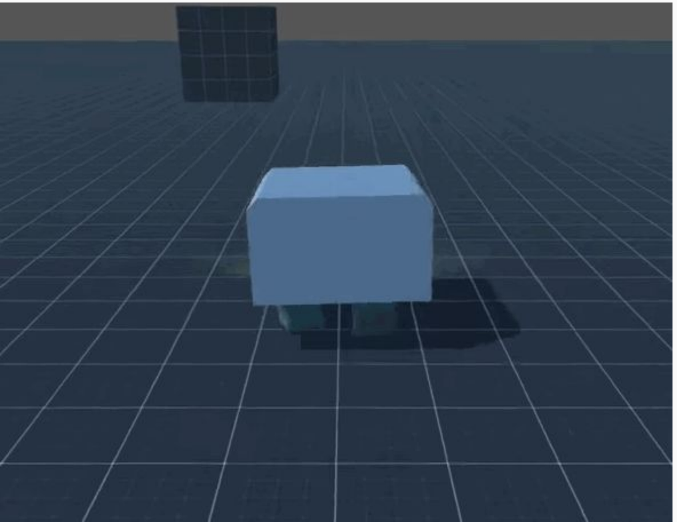
3D rigging is a technique of creating the bone structure (skeleton) of a 3D model. This bone structure simplifies the animation process and is usually created using some 3D modeling software like Blender.

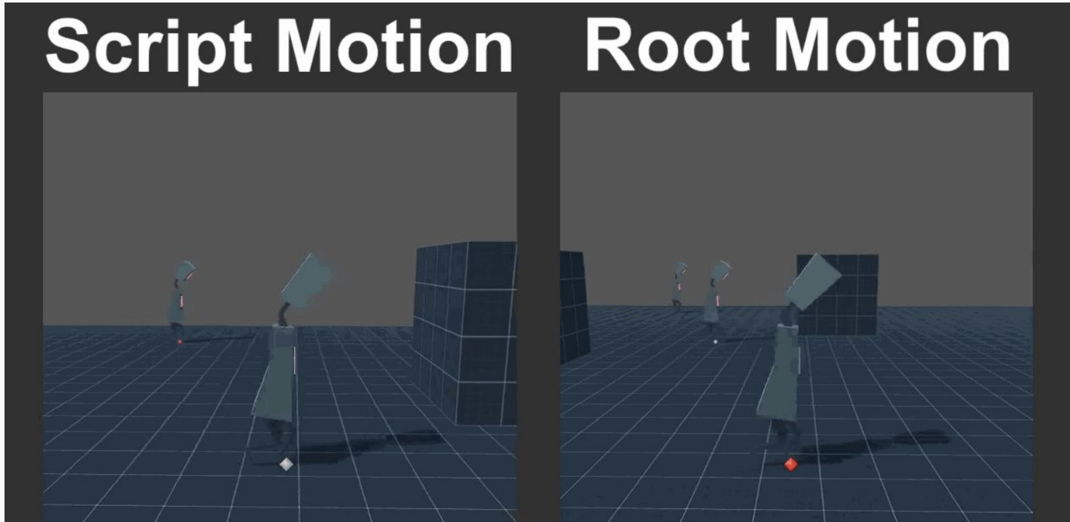
When importing this skeletal rig data into Unity (usually included in the 3D model file), it will be converted into a hierarchy tree of gameobjects with a single root which Unity can understand. This makes it easy to animate as if you want to rotate the arm of a character, then you only need to rotate a single bone (shoulder bone) and the arm will rotate due to the nature of hierarchical transforms. However, while it is possible, you normally do not create animations for a 3D rigged character in Unity as more complex animations would take a very long time to animate (skeletons have many many bones). Usually these animations are done again using some 3D modeling software and the data is included in the 3D model file.

Along with rotations of all the various bones, sometimes the animation data also includes translations and rotations of the root bone (usually the hips for humanoid rigs). By default, unity will ignore these transformations on the root because if it did not then the character would move on its own which is usually unwanted (ex: we want an **in-place** running animation and the movement will be driven by code). Although, there are times when we might want the movement to be driven by the animation. This is called **root motion**, as you are allowing the root bone to be moved using animation rather than through code.

Root Motion

Root motion solves a few issues with regards to the **accuracy** of animations. For example, using root motion avoids the problem known as **foot sliding** (shown in the gif at the bottom right). While subtle, the result is consistently having the feet stay in the right place instead of sliding on the ground. You can also notice that the root motion robot in the bottom right has its speed start and stop as it takes its steps rather than going at a constant speed all the time resulting in a more accurate looking sneaking animation. Another advantage is more accurate collisions. Take the robot in the bottom left gif for example, it sways to the left and right a lot while walking. Now imagine moving this robot through some tight corridors with some porcelain pots on some shelves. If you simply had a regular box collider on the root gameobject, it would not sway with the motions resulting in the player being able to move close to the breakable porcelain pots and have the model clip through them while walking without breaking them because the box collider is simply not moving to match the motions of the root gameobject.





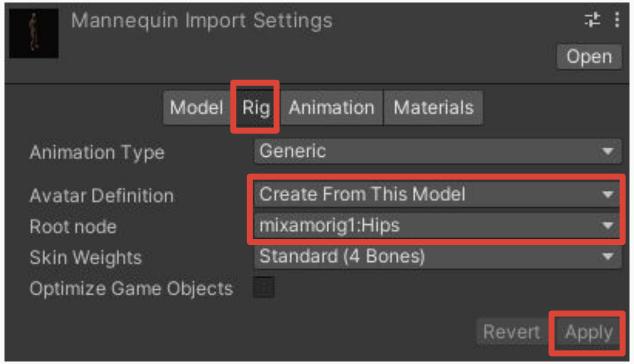
Root Motion

Root Motion is very nice and can be very useful although using root motion is also makes the workflow more complex. There are different levels at which you can use root motion and all these take time to animate, more than a script driven player, and will require your team or self to have a good handle between programming, Unity's mechanism system (used for the animator), and the animation process itself. You especially must have a programmer and animator that are willing to communicate and work a lot between each other too get your player moving the way your designer and game needs it to. Root motion requires all these things to work well together before you can even begin moving the character.

Is it worth it? While polished root motion is more accurate and can be tempting, it is worth considering if you really need it. In the examples shown in the above slide, there are ways to solve those problems via script that while not 100% accurate, produce visually pleasing results without using the complexity root motion introduces into your workflow. In the end, I only recommend using root motion if your game has a heavy emphasis on realistically animated movements and if your team can manage it.

Root Motion in Unity

In order to setup root motion in Unity given a fully animated 3D model with root motion baked into it, start by first selecting the 3D model asset in the project window which will make the import settings for this model appear in the inspector view. Make sure that the model has an avatar setup, if it does not then you can try to automatically generate one using the options provided in the inspector view under the **Rig tab**:



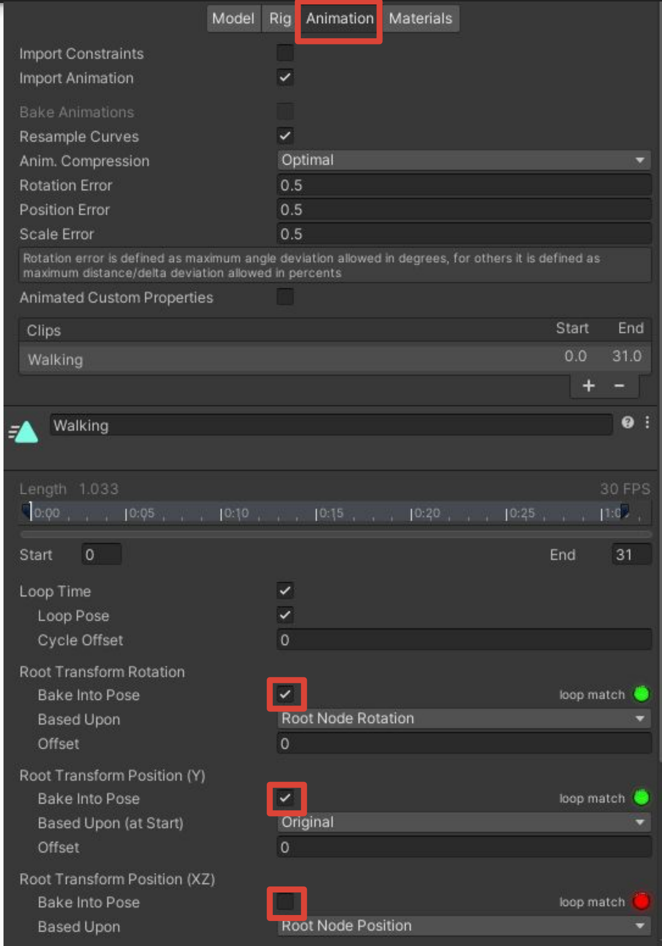
Mannequin Import Settings

Animation Type	Generic
Avatar Definition	Create From This Model
Root node	mixamorig1:Hips
Skin Weights	Standard (4 Bones)
Optimize Game Objects	

Next, switch over to the **Animation tab**, for each animation clip that you want to use root motion you need to check/uncheck the appropriate checkboxes and configure the options correctly (see image on the right).

Once all options are configured, don't forget to hit **apply** located in the bottom right.

See the next slide for more information on these options.



Import Constraints	
Import Animation	✓
Bake Animations	
Resample Curves	✓
Anim. Compression	Optimal
Rotation Error	0.5
Position Error	0.5
Scale Error	0.5

Rotation error is defined as maximum angle deviation allowed in degrees, for others it is defined as maximum distance/delta deviation allowed in percents

Animated Custom Properties	

Clips	Start	End
Walking	0.0	31.0

Walking

Length 1.033 30 FPS

Start	0	
End	31	
Loop Time	✓	
Loop Pose	✓	
Cycle Offset	0	
Root Transform Rotation		
Bake Into Pose	✓	loop match 🟢
Based Upon	Root Node Rotation	
Offset	0	
Root Transform Position (Y)		
Bake Into Pose	✓	loop match 🟢
Based Upon (at Start)	Original	
Offset	0	
Root Transform Position (XZ)		
Bake Into Pose		loop match 🔴
Based Upon	Root Node Position	

Root Motion in Unity

Root Transform Rotation

Bake into Pose: The orientation will stay on the body transform (or Pose). The Root Orientation will be constant and delta Orientation will be identity. This means that the root transform will not be rotated at all by that AnimationClip.

- Only animation clips that have similar start and stop Root Orientation should use this option. You will have a Green Light in the UI telling you that an animation clip is a good candidate.
- An example suitable candidate would be a straight walk or run.

Based Upon: This lets you set what the root orientation of the clip is based upon. You have two options:

- **Root Node Orientation**: the clip will be oriented to follow the forward vector of the root node. This default setting works well for most Motion Capture (Mocap) data like walks, runs, and jumps, but it will fail with motion like strafing where the motion is perpendicular to the body's forward vector. In those cases you can manually adjust the orientation using the Offset setting.
- **Original**: this will automatically add the authored offset found in the imported clip. It is usually used with keyframed data to respect orientation that was set by the artist.

Offset: used to enter the offset (in degrees) for the option that is chosen for Based Upon.

Root Motion in Unity

Root Transform Position (Y)

This uses the same concepts described in Root Transform Rotation.

Bake Into Pose: The Y component of the motion will stay on the Body Transform (Pose). The Y component of the Root Transform will be constant and Delta Root Position Y will be 0. This means that this clip won't change the y position of the root transform.

- Again you have a Green Light telling you that a clip is a good candidate for baking Y motion into pose.
- Most animation clips will enable this setting. Only clips that will change the GameObject height should have this turned off, like jumping up or down.

Based Upon: Works in the same way as Based Upon for Root Transform Rotation except for y position.

- **Root Node Position**: the clip will be positioned to follow the y position of the root node.
- **Original**: this will automatically add the authored offset found in the imported clip. It is usually used with keyframed data to respect the y positioning that was set by the artist.

Offset: In a similar way to Root Transform Rotation, you can manually adjust the animation clip y position using this setting.

Root Motion in Unity

Root Transform Position (XZ)

Again, this uses same concepts described in Root Transform Rotation and Root Motion Position (Y).

Bake Into Pose: Works in the same way as Based Upon for Root Motion Position (Y) except for xz position.

- Again you have a Green Light telling you that a clip is a good candidate for baking XZ motion into pose.
- This is usually used for “idle” animations where you want to force the delta Position (XZ) to be 0. It will stop the accumulation of small deltas drifting the character after many evaluations.

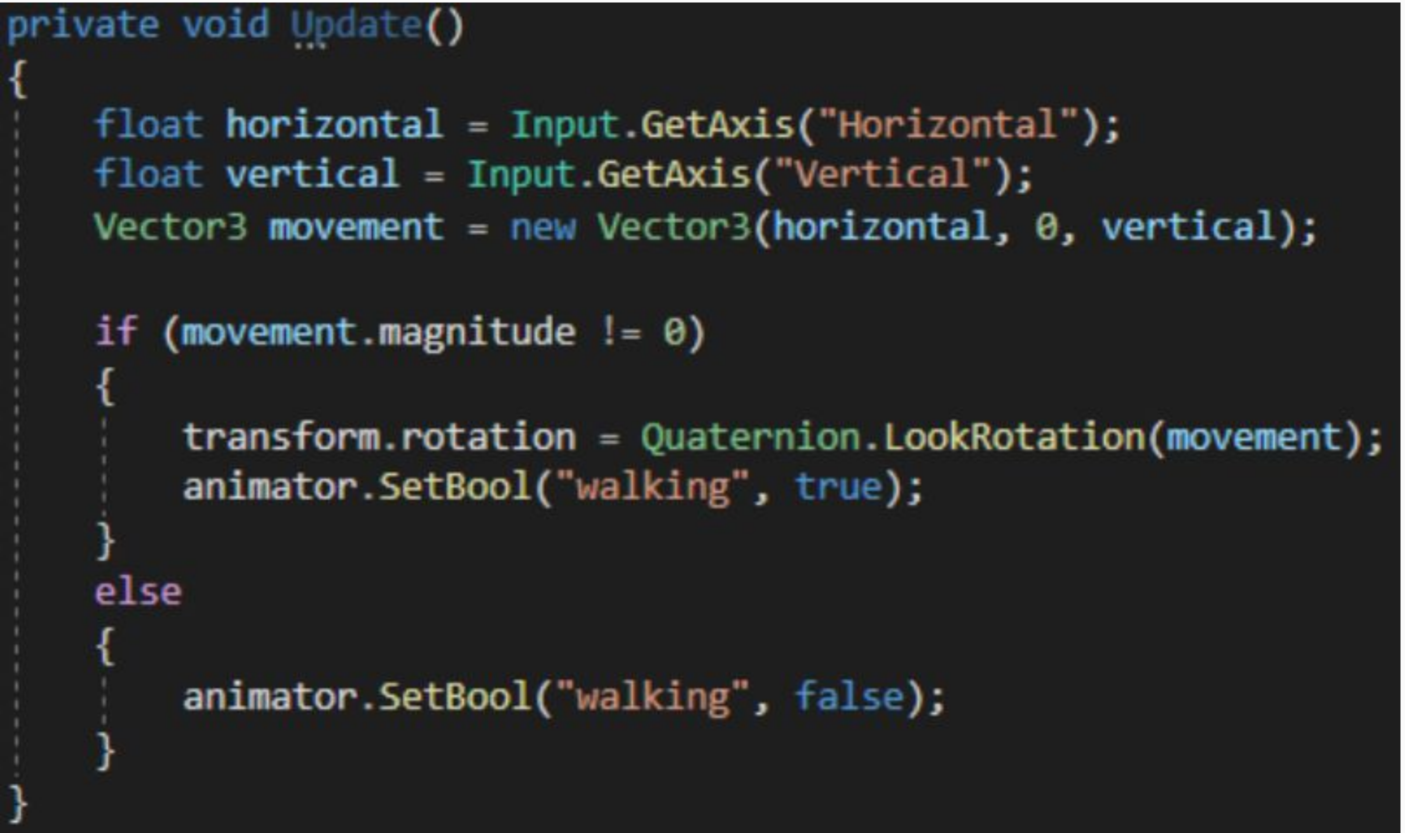
Based Upon: Works in the same way as Based Upon for Root Motion Position (Y) except for xz position.

- **Root Node Position**: the clip will be positioned to follow the xz position of the root node.
- **Original**: this will automatically add the authored offset found in the imported clip. It is usually used with keyframed data to respect the xz positioning that was set by the artist.

Offset: In a similar way to Root Motion Position (Y), you can manually adjust the animation clip xz position using this setting.

Root Motion in Unity

Once everything is configured in the Editor, the code must also be configured to accommodate the movement that is now being driven by the animation. The movement code is now simplified:



```
private void Update()
{
    float horizontal = Input.GetAxis("Horizontal");
    float vertical = Input.GetAxis("Vertical");
    Vector3 movement = new Vector3(horizontal, 0, vertical);

    if (movement.magnitude != 0)
    {
        transform.rotation = Quaternion.LookRotation(movement);
        animator.SetBool("walking", true);
    }
    else
    {
        animator.SetBool("walking", false);
    }
}
```

Tasks

Character Animation Setup

Begin by obtaining all the required assets for this lab. These are a character model, and animations that are appropriate for applying root motion.

- Use the Mixamo platform to select a humanoid character. You may download any model you prefer.
- Next, download idle, walking, and running animation as fbx files. These animations will be applied on your model.
- If applicable, extract the materials and textures from the model. You may want to lower the materials' smoothness value, to remove unwanted "glossiness".
- Create a prefab based on the downloaded model (The model should be the root of the prefab). Call it "Player".
- Add a Collider component to the root GameObject of the prefab (this should be the root gameobject of your model). Adjust the collider to roughly envelope the character's core. It is expected to not cover the arms in a T-pose.
  - Tip: Using the isometric camera mode in the scene view can make editing the collider easier.

Character Controls

- The character must move around the scene by responding to keyboard inputs. WASD to move and Shift while moving to run.
- The character must be able to pick up (Destroy) coins by touching them.

Coin Spawner

- Spawn a new coin every 5 seconds by adding the CoinSpawner prefab to the scene and modifying it. The location of the new coin must be randomly sampled within a volume.
- Coins should automatically be destroyed after 10 seconds. Coins should also be destroyed when colliding with the player.

Thank You

Link to Lab Drive

https://drive.google.com/drive/folders/1jgFTWCJ5JD-JnIY7n_L7s2n2vA1F9nNX?usp=sharing

OR

https://tinyurl.com/ta-comp476-daniel

Other Links

https://docs.unity3d.com/2020.3/Documentation/Manual/index.html

https://docs.unity3d.com/2020.3/Documentation/Manual/ExecutionOrder.html

https://docs.unity3d.com/Manual/RootMotion.html