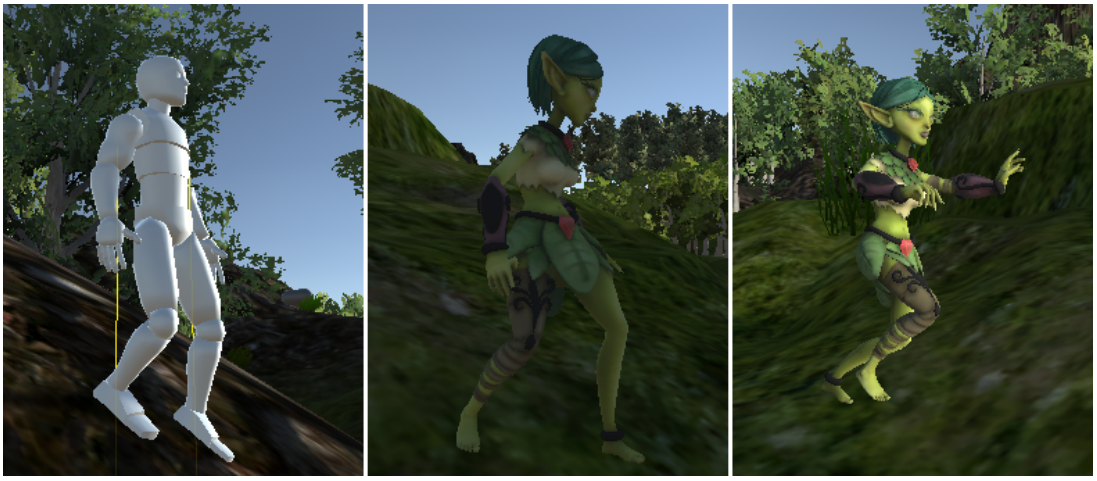
Session 3
Character Animation with IK



In this session, we are including a fully-animated bipedal character in the virtual world that you have been creating during the previous tasks. You will learn how to control it, to define an animation controller with basic movements such as walking or running and to implement an inverse kinematics system for the lower limbs, such that the character can adapt its motion to the irregular terrain.

3.1 Getting started

The project already contains all the necessary elements to work straight-forward and complete the session by fulfilling the code tasks. Nevertheless, you can find below a brief description about each of these new elements, so that you can learn more about how Unity works in case you want to make changes to your project or include your own character.

3.1.1 Humanoid character and animations

In the project, the folder `Assets > Dummy` contains the necessary assets to render and animate our character in the scene. Some of the most important sub-folders are:

- `Dummy > Models`: The `.fbx` file inside contains the general model of our character. It normally consists of: a Skinned Mesh Renderer (appearance), a Transform (location) and an Avatar (interface that performs a mapping between each bone of the model rig and the humanoid skeleton representation from Unity).
- `Dummy > Animations`: Each of these `.fbx` files contains a motion clip that you can use in your character. These clips can be configured in terms of a specific Avatar or by modifying its multiple animation parameters.
- `Dummy > AnimationControllers`. This folder contains the behavior controllers. They take together the animation clips from the previous folder, and provide the character with a certain behaviour (sort of actions), which can be also based on other elements, such as external keyboard inputs.

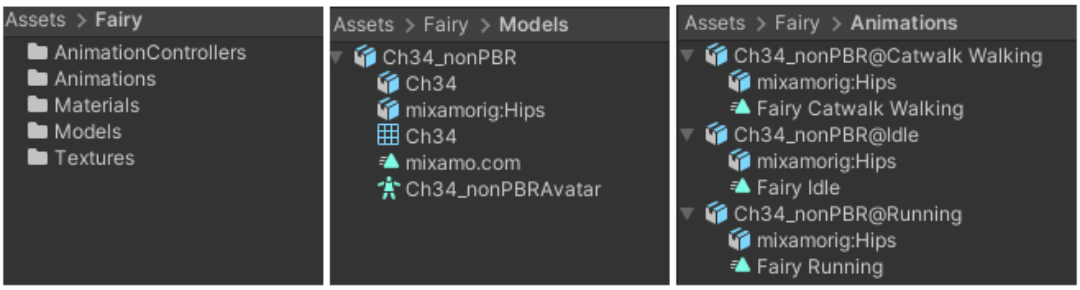


Figure 3.1: Example of folder distribution for a new character

Importing your own character (optional)

This project contains a very basic dummy character for the task. However, you can import your own character and animations if you like. Our recommendation is to use Mixamo to obtain both (you will need an free Adobe account):

- First, you will need the character model. In the **Characters** tab, select the character you want and click on **Download**. When asked, select `FBX for Unity(.fbx)` format and `T-pose`.
- Then, we need to obtain the animation clips. We recommend you to start with some basic movements, such as: IDLE (when the character stays still), walking and running. Go to the **Animations** tab and find a clip for each of these (you will see a pre-visualization of the animation in the character you have chosen). Select **Download** and when asked, choose `FBX for Unity(.fbx)` format and `without skin`, since you already downloaded it before. The other options can be left by default.
- Finally, you can create a new folder (`Assets > YourCharacter`) and paste these files, as shown in Figure 3.1. You can follow the same hierarchy as in the `Dummy` folder and create a `Models`, `Animations` and `AnimationControllers` folder (this last one is still empty; we will use it in the next section).

Now, create two new folders inside `Assets > YourCharacter`: `Textures` and `Materials`. Then, go to the `.fbx` model file inside the `Model` folder, and in the Inspector, select the tab **Materials**. There, click on **Extract Textures** and **Extract Materials** to extract them in the respective folders that you have just created.

To finish, go again to the `.fbx` model file inside the `Model` folder, and in the Inspector, select the tab **Rig**. There, choose `Humanoid` for the `Animation Type` and in `Avatar Definition`, select `Create From This Model`. Then, click on **Apply**. A check mark should have appeared next to Configure (Figure 3.2).

3.1.2 Animation controller using blend-trees

The `Animator Controller` takes the animations together and builds a motion behaviour. They can get very complex, for example, by describing a change of animation based on the user-inputs (e.g. while pressing `CTRL`, it runs instead of walking) or by adding blending animations between two different motions, which Unity can build automatically (e.g. how the character moves in the transition between walking and running).

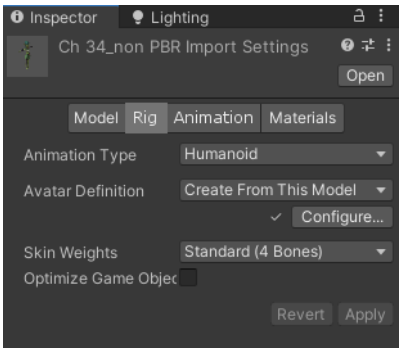


Figure 3.2: Model Rig Configuration

Creating a Character Controller for your character (optional)

For your new character, we recommend you to use the same Character Controller than for the Dummy model, which already contains a basic behavior control. Go to `Assets > Dummy > AnimationControllers > Custom` and select **ImprovedController**. Duplicate it (`CTRL + D`) and paste it on `Assets > YourCharacter > AnimationControllers`. Give it a different name.

When you double-click on the **ImprovedController**, the **Animator** tab will appear (Figure 3.3). Inside, you will see that this controller is using a blend-tree. This system allows to transit between animations smoothly using an external value. The only thing we want to do in this case, is to change the animation clips for each branch of the tree, by the clips that you downloaded in the previous section.

- First, you need to modify each of the animation clips that you downloaded. In the Inspector, go to **Rig** and choose `Humanoid` for the `Animation Type`. Then, in `Avatar Definition` select `Copy From Other Avatar` (unlike before). Then, as `Source` select the Avatar that you created previously from the model itself, as shown in Figure 3.4. Finally, click on **Apply**. Repeat this for each animation clip.

- Second, again for each animation clip, choose the tab **Animation**. You need to check the option **Loop Time** for each of them and apply the changes (this will make it cyclic). Additionally, certain animations can induce undesired position changes due to the nature of their motion. To avoid that, you can bake into pose (freeze) the positions, so they are not affected. For example, an IDLE animation should not move the character in any way. For that specific clip, check the option **Bake into Pose** for `Rotation`, `Position (Y)` and `Position (XZ)`. For other animations (like walking or running), you will only need to bake `Rotation` and `Position (Y)`, since you want the character to move forward or to the sides when executing these (the example for an IDLE animation is shown in Figure Figure 3.4)

- In the **Animator** tab, while being in the **Base Layer**, double-click on **Free Move**. This will open a new layer containing the blend-tree. If you click on it, the Inspector will show the different animation clips and the `InputMagnitude` needed to transit between animations (as shown in Figure 3.5). Now, we just need to replace such clips by the respective animations that we have for our character. Attention: You need to put the animation clip (moving-triangle symbol) that is inside the `.fbx` file. For now, you can remove the last branch use to implement a sprinting animations, since we will not be using it. However, you can add it if you like.

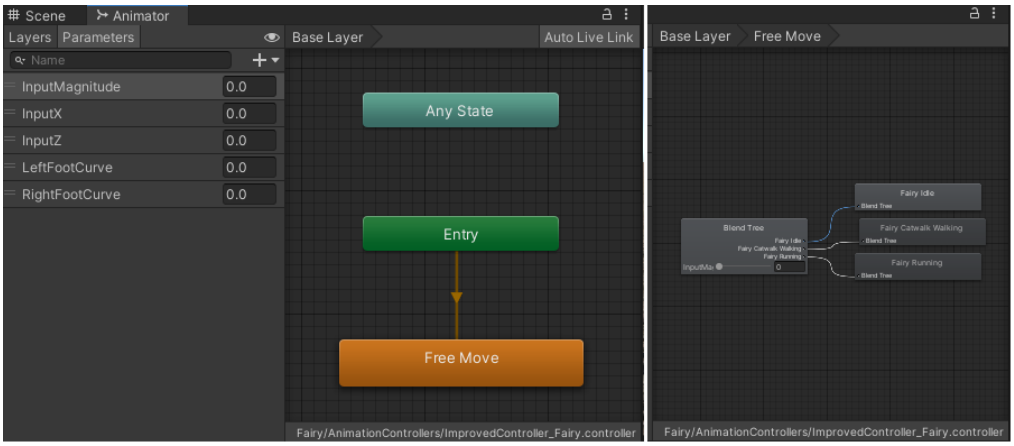


Figure 3.3: Animator Controller and Blend-tree

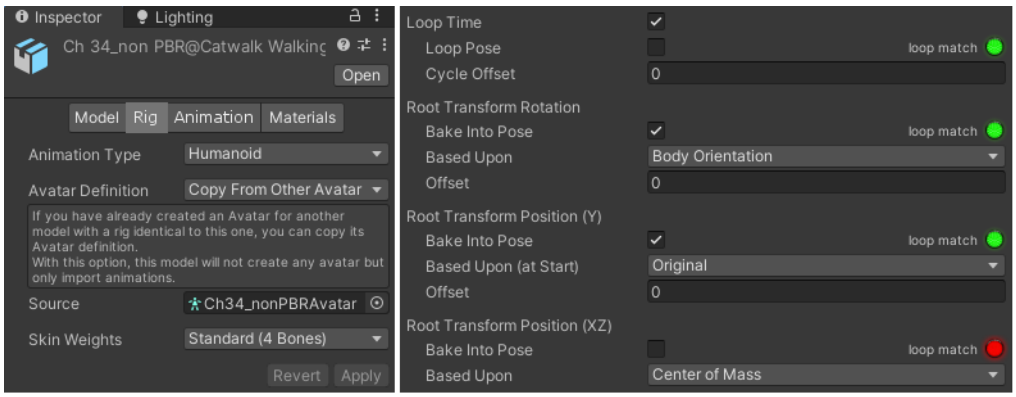


Figure 3.4: Animation Clip Configuration (IDLE animation on the right)

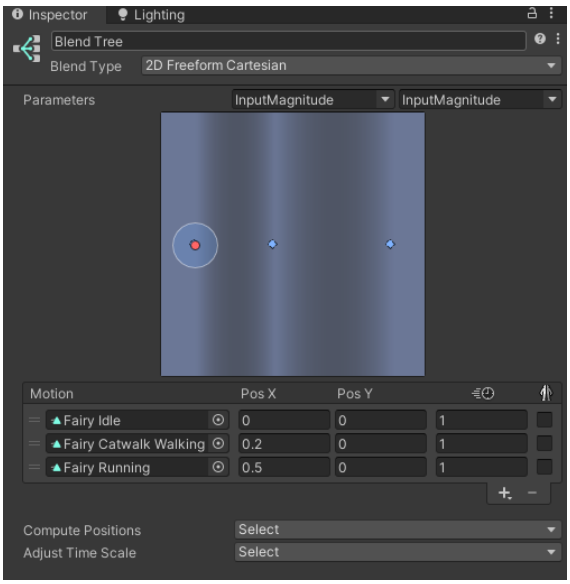


Figure 3.5: Blend-tree Animations

Tip

You can click on the small lock icon on the top-right corner on the Inspector to lock it, so it does not goes away. This is useful when you are navigating on your `Assets` to place elements in the locked Inspector window.

You are ready configuring your character! Go to `Assets > YourCharacter > Models`, and drag and drop the `.fbx` model into the Scene. Place it in the `Characters` game object in the Hierarchy along with the Dummy if you desire.

3.1.3 Character controller and capsule collider

Now we need to control our character. In Unity exist diverse ways to establish our character in the world and to define how it reacts back based on the existing physical forces or controller inputs. Below you can find a brief description of the most important physical components in Unity:

- `RigidBody`: When a `GameObject` has a `RigidBody` attached to it, it will react to/against the forces built-in the physics system from Unity, such as gravity.
- `Collider`: A `Collider` attached to a `GameObject` will make the element to react against other colliders in the scene (e.g. by hitting them physically or by triggering other actions).
- `CharacterController`: It provides an already-built interface to control your character. It already includes a `Collider`, therefore you do not need to add one. It also allows to program every external force manually (like gravity) if you do not want to use a `RigidBody`. A `GameObject` with only a `CharacterController` will not react to the Unity physics system.

There are normally two different approaches to implement the control of your character using the previous components. You can either use a combination of `RigidBody + Collider` or a `CharacterController` directly. While both approaches react against other colliders, the first approach will be normally less permissive and more sensitive if it hits other obstacles, no matter how small they are. Also, the `RigidBody` API implements many more functions to interact with physics, while a `CharacterController` is more suitable for basic functionalities.

Implement the motion for your character (optional)

Next, we will add the necessary components in the Inspector to implement the movement of our character, in the same way as the Dummy model does.

- **Animator**: This component should be already included. In the `Controller` option, select the animator controller that you created for your character.
- **Character Controller**: You can find it on `Add Component > Physics > Character Controller`. You will need to modify the `Capsule Collider` based on the size of your character. This capsule will define its physical volume and is represented by a green capsule on the Scene. Change the Center (`Y`), Radius and Height to adapt it to your character.
- **Scripts**: You can use the already-created motion scripts that the Dummy model uses. Go to `Assets > Scripts` and drag/drop the following `.cs` files to the Inspector of your model:

  `PlayerControllerInput.cs`: Used to transform the keyboard user-inputs into motion to the character. Inside, change the `Desired Rotation Speed` parameter to be **larger than 0** (e.g. 0.2).

  `04_IK > PlayerControllerIKFeetPlacementTask.cs`: **Your IK task!** Continue to the next section to know more about.

Camera Control

The project uses a built-in camera system called **Cinemachine**, to create a 3rd-person view of your character. By default, if you click on **Run** and show **Display 2**, the camera will follow the Dummy model. To change that, to into the Hierarchy and select `CM FreeLook1`. In the Inspector, drag and drop your character model directly to the `Follow` and `Look At` properties, such that it selects automatically its Transform component.

Congratulations! You already have your fully-operational character ready-to-use.

3.2 Inverse Kinematics (IK)

When we implement a normal (direct) kinematics system, the animation controller moves the character and the capsule collider defines the volume that reacts with the environment. Although the approach is simple to implement, some collisions can look unrealistic in specific moments, because the shape of the collider already hits the floor, and not the feet (e.g. foot goes through the ground when climbing up a hill).

A way to solve this problem is using inverse kinematics. While with direct kinematics, you need to set for each keyframe a particular `Vector3` position for each limb and joint, inverse kinematics allows you to define one fixed location and adapt the entire morphology with respect to it (e.g. if you are forcing the foot to be always parallel touching to the ground, you can induce a particular shape to the respective leg).

In our task, we will implement an inverse kinematic system to induce a reaction to the feet and adapt the leg to the irregular ground, providing more realism to the shape of the character's body.

3.2.1 Main Scripts

In the hierarchy, you can find an empty GameObject called `Characters` that contains the prefab for our `dummy` (or your custom model if you decided to create it in the previous section). Either way, it should contain a `CharacterController` component (`CapsuleCollider` included) and two scripts:

```
- PlayerControllerInput.cs
- PlayerControllerIKFeetPlacementTask.cs (TODO)
```

The first one implements the necessary code to control our character with the keyboard and adapt the animations accordingly. The second should implement the IK system for the feet adaptation, and belong to the task that you need to complete.

3.2.2 Implement a basic IK system (`OnAnimatorIK`)

The most important part of the IK system belongs to the message `OnAnimatorIK`, which is called immediately before it updates its internal IK system. It will perform this in two steps:

- During the animation process, each part of the body receives a weight describing how much the inverse kinematics system influences its position. If it is **0**, the body part will not adapt and will follow a fully direct kinematics movement. On the other side, **1** means that the body limb will redirect its position totally to the goal or direction that the IK system defines.
- After setting the IK weights, we need to move the limb to the position where we want it to be.

Tip

Inside the `OnAnimatorIK` message, you will just need to call some API-based methods and functions that we have already implemented for you. The code is commented, therefore should be feasible to do and understand. An IK system can be really complex, but for the sake of simplicity we will keep the task adapted to be fast to code.

```
// Sets the translative weight of an IK goal (0 = at the original animation
  before IK, 1 = at the goal).
public void Animator.SetIKPositionWeight(AvatarIKGoal goal, float value);

// Sets the rotational weight of an IK goal (0 = rotation before IK, 1 =
  rotation at the IK goal).
public void Animator.SetIKRotationWeight(AvatarIKGoal goal, float value);

// Move feet to the IK Goal (custom built function)
void PlayerControllerIKFeetPlacementTask.MoveFeetToIKPoint(AvatarIKGoal foot,
  Vector3 positionIKHolder, Quaternion rotationIKHolder, ref float
  lastFootPositionY)
```

Useful functions

3.2.3 Moving the feet to the IK Goals (`MoveFeetToIKPoint`)

After setting the weights for the IK animations, you will need to call `MoveFeetToIKPoint()` at the end of `OnAnimatorIK` in order to move the limbs to the goal positions.

In this part of the exercise, you have to retrieve first the current IK positions for each foot. Then, you want to define which is the translation that you need to apply to each foot, in order to move it to its goal position. This translation will be only perform on the height (`Y` component). Finally, you will add this desired translation to the current IK position (saved at the beginning).

```
// Returns a Vector3 with the current position of this IK goal in world space.
public Vector3 Animator.GetIKPosition(AvatarIKGoal goal);

// Linearly interpolates between a and b by t.
public static float Math.Lerp(float a, float b, float t);
```

Useful functions

Warning

On the Inspector, for the `PlayerControllerIKFeetPlacementTask.cs` script in your character model, you will need to do some changes to make the IK system run properly. First, change the Ground and Environment Layer to **Default** (or whatever layer which is selected for your terrain). Also, check the option **Use Pro IK Feature**, such the IK system affects the rotation of the feet too. Finally, drag your animator controller and character controller component and drop them to their respective parameters, `Anim` and `Controller`.

Once you have completed the task, you can test the IK system. Click on **Run** and select the 3rd-person view of your character by selecting **Display 2**. You can control your character with the keyboard arrows or the W-A-S-D combination to experiment with the IK system, while observing the world that you have created around you (Figure 3.6).

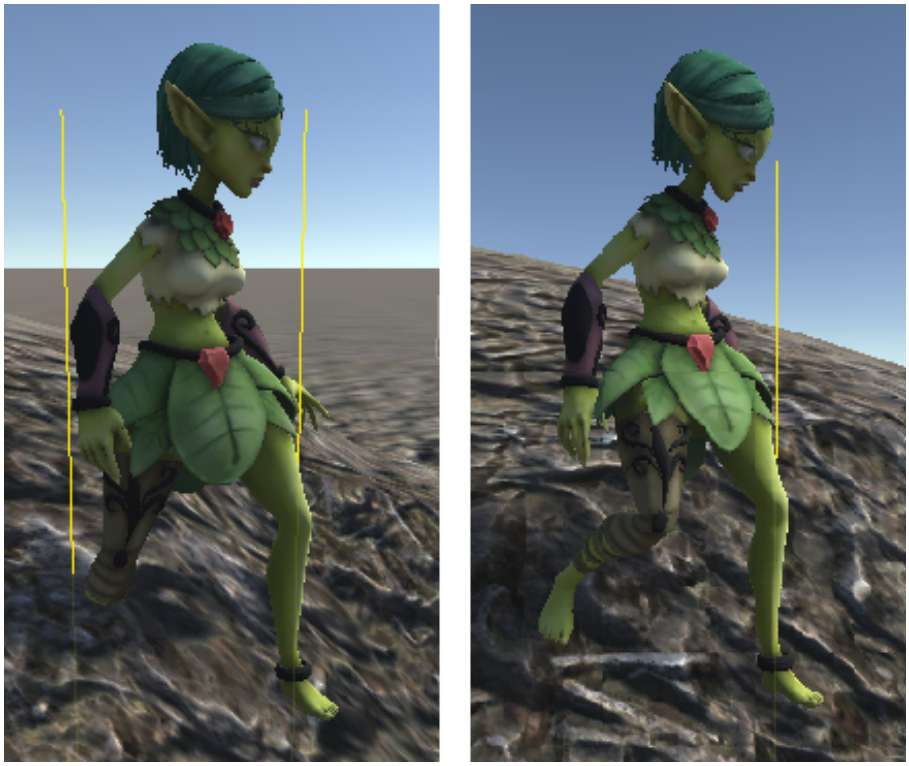


Figure 3.6: IK disabled (left) - IK enabled (right)