

Realistic Eye Movements

Version 2.1.12

Quickstart for a character with existing presets:

You can also watch the [Quickstart video](#).

REM comes with presets for characters generated with [Autodesk Character Generator](#), [MakeHuman](#), [Reallusion Character Creator](#), [DAZ Studio](#) and [UMA](#). If you have such a character:

Add the scripts LookTargetController and EyeAndHeadAnimator from the Scripts folder to your character. Click on the "Import" button and import the preset for your character. Done!

About Realistic Eye Movements

Thanks for purchasing Realistic Eye Movements! I hope this asset helps you bring your characters to life. If you have any questions or suggestions, please drop me a line at tore.knabe@gmail.com!

For discussions of this asset, visit the [Unity forum thread](#).

The web page for this asset is [Unity Asset: Realistic Eye Movements](#).

Realistic Eye Movements (REM) can control your character's eyes, head and eyelid movements to make them look around, at the player, or at objects, in a lifelike way. You can use it with characters that have eyes rigged to a Mecanim humanoid bone rig or characters that just have two separate eye gameobjects. You can assign points of interest in the character's environment for the character to look at, or let them just look around idly, or let them automatically look at the player when the player comes into view or keeps staring at them.

The animations use data from published research papers. For more information, see [my blog entry](#).

This asset only provides scripts to control animation, no meshes or shaders. For realistic looking eye meshes and shaders, I recommend Scruvystorm Studios' [RealEyes](#) asset or Tanuki Digital's [Eye Advanced](#) asset, there's also Ricardo's [Realistic Eye](#) asset or RRFreelance's [Eye Shader](#) asset. You can compare the three assets in the [webplayer demo](#).

How to Use

There are two main scripts in the folder RealisticEyeMovements/Scripts:

LookTargetController.cs

Chooses what to look at and when to switch look targets.

EyeAndHeadAnimator.cs

Controls the animation of eyes, head and eyelids for a given look target.

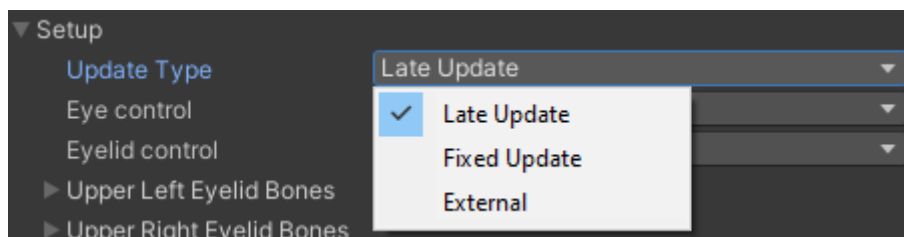
If you want more control over where and when to look you can modify LookTargetController.cs or replace it with your own script that calls functions from EyeAndHeadAnimator.cs. You probably don't need to modify EyeAndHeadAnimator.cs.

Drag the two scripts onto your character game object in the scene hierarchy. Then either import a preset or proceed with the Setup below.

Setup

*If your character was generated with Autodesk's Character Generator, MakeHuman, Reallusion Character Creator, DAZ Studio or UMA, instead of manually configuring the eyes and eyelids as described below, you can just load the correct configuration for it by importing the corresponding preset: see the section **Presets** below.*

If you have a character for which you don't have a preset yet, the first step is to configure it in the **Setup** foldout of the EyeAndHeadAnimator component:



Update Type

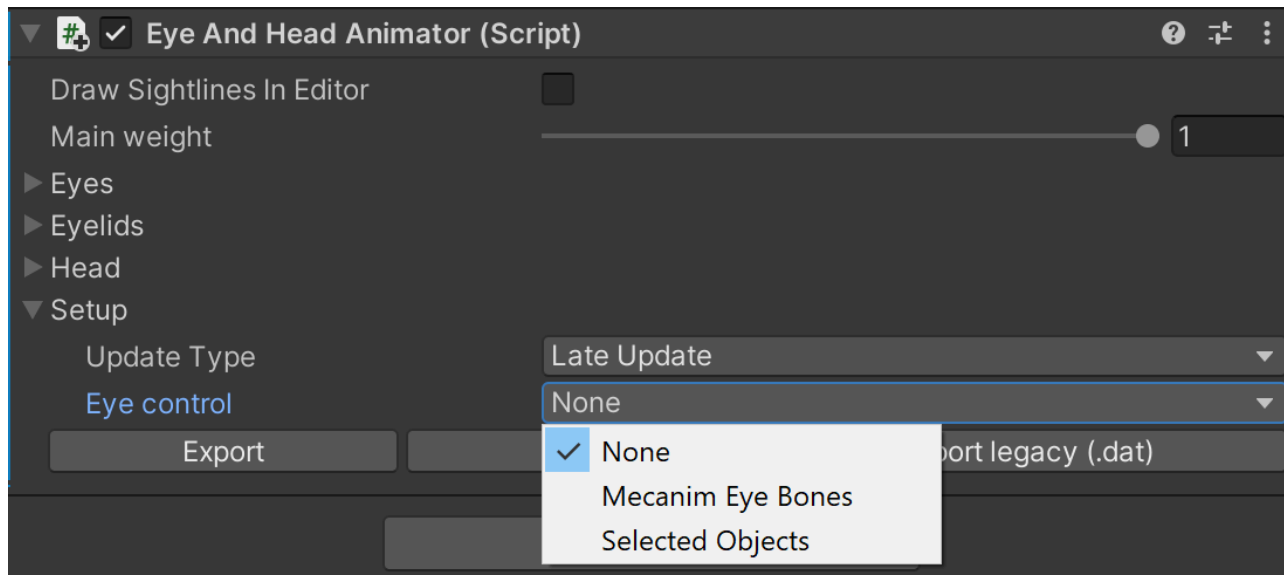
For most scenarios, you can leave **Update Type** to **Late Update**. If your animator is set to *Animate Physics*, you should change this to **Fixed Update** so when REM rotates the head the physics is in line with the rest of the character. This setting will only work when the animator is indeed set to *Animate Physics*.

The setting **External** gives you more fine-grained control over when exactly the head and eye movements are updated: you call `Update1()` and `Update2()` yourself. Normally you don't need that, but in certain special cases like for example you use different FinalIK components such as `LookAtIK`, `FullBodyBipedIK` (fbrik) the order of updates is important: `LookAtIK` normally runs before fbrik, so if fbrik rotates the upper body because the hands grab something to the side, the head will no longer look in the desired direction, so you need to update the FinalIK components yourself. You want fbrik to rotate the upper body so the hand can reach their target, then to let REM figure out where the head should look, then `LookAtIK` should actually turn the head, then REM should rotate the eyes. For such a scenario, check this box and then each frame call the two functions `Update1` and `Update2` on the `EyeAndHeadAnimator` component (with the correct `deltaTime` as parameter). Make sure `Update1` is called before FinalIK orients the head (because `Update1` sets the head target), and call `Update2` after FinalIK is done orienting the head, because `Update2` moves the eyes. For example:

```
eyeAndHeadAnimator.Update1 (Time.deltaTime) ;  
lookAtIK.solver.Update () ;  
eyeAndHeadAnimator.Update2 (Time.deltaTime) ;
```

[Here](#) is a short script that can be added to a character and which does the correct order of updates for the above scenario.

Eye Setup



Set **Eye Control** to either *Mecanim eye bones* or *Selected Objects*.

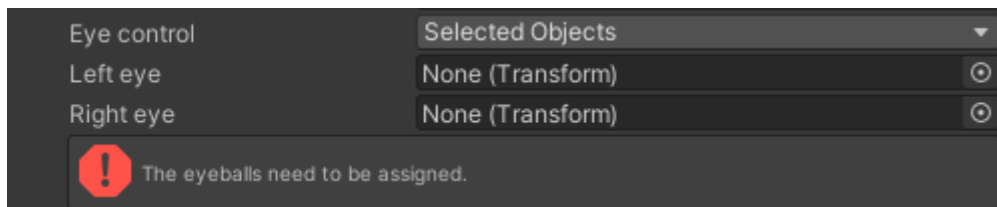
If you choose *Mecanim eye bones*, your character must have a Mecanim humanoid rig and an Animator component. If your Mecanim rig has the eye bones assigned correctly, the script will find and use them to animate the eyes. Choose this option if your character's eye meshes are controlled by Mecanim eye bones.

If you choose Mecanim eye bones and you get the error message „Eye bones not found“, then maybe you have checked „Optimize Game Objects“ in the import settings:



Make sure Optimize Game Objects is unchecked. Even if you expose bones for eyes and eyelids, the transforms for these bones will be read-only so REM cannot control them.

If you choose *Selected Objects*, you can select game objects that are your character's eyes in the character's object hierarchy to be controlled by the script. Drag each into its corresponding slot in the component:

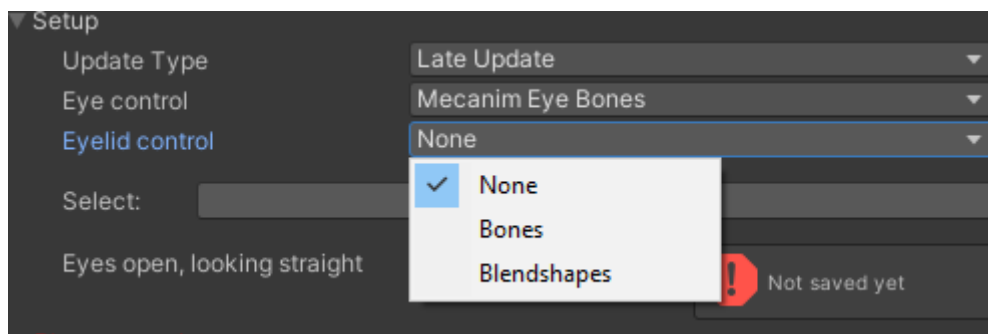


To quickly select a character's head bone in order to get more easily to the eye objects, select the character in the hierarchy and then in the menu choose "Tools/Realistic Eye Movemets/Select Mecanim Head", or press the shortcut Ctrl+Shift+H.

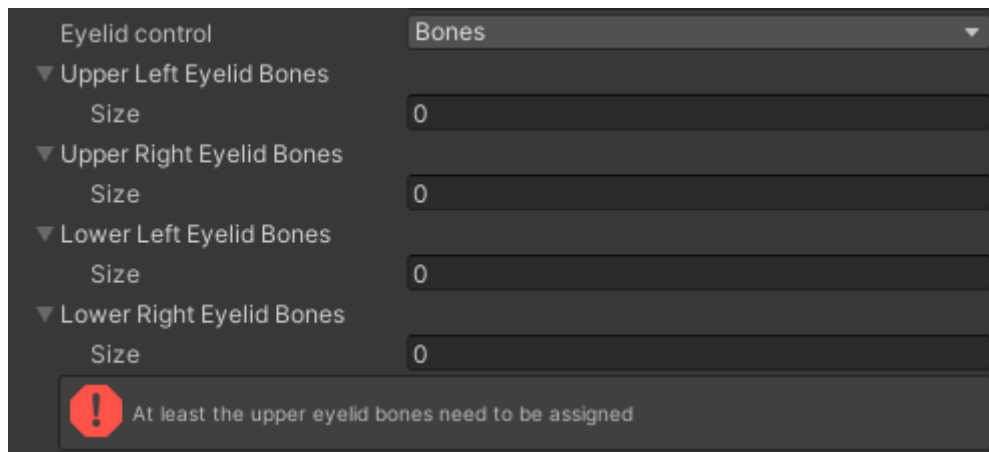
Moving the eye via eye mesh blendshapes is not supported.

Eyelids Setup

Once you've finished Eye Control, the box **Eyelid control** appears. If your character has eyelids that can be controlled by bones or blendshapes, the REM can control them to add realism. If not, leave the box **Eyelid control** set to *None*. If your character does have eyelids, set it to either *Bones* or *Blendshapes*.



If you choose *Bones*, find the eyelid bones in your character's hierarchy and drag them into the corresponding slots. Each eyelid slot can take an array of bones. Most characters have just one bone per eyelid, but you can assign several bones per eyelid if your character has them.



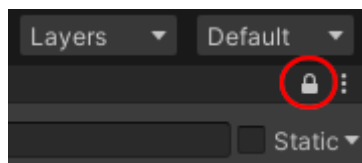
The slots for the lower eyelids can be left empty if your character doesn't have lower eyelids to animate.

If you choose *Blendshapes* for eyelid control, you don't need to assign anything here.

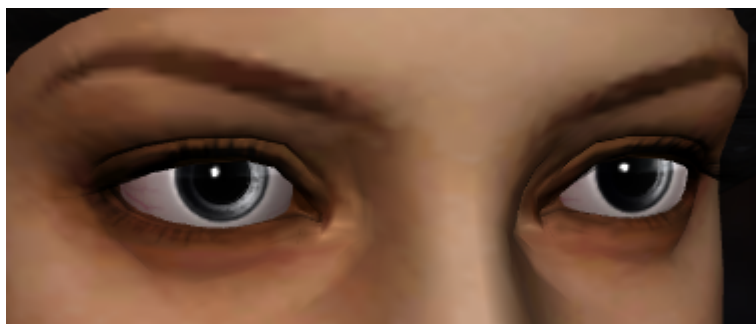
Saving Positions

The next step is to save the positions for looking straight, looking up, looking down, and, if you enabled eyelid control, for closed eyes. This will tell the scripts the limits of how far the eyes can look up and down and how to move the eyelids.

For the following steps I recommend locking the inspector tab of the GameObject that has the EyeAndHeadAnimator component with the little padlock symbol in the inspector pane's upper right:



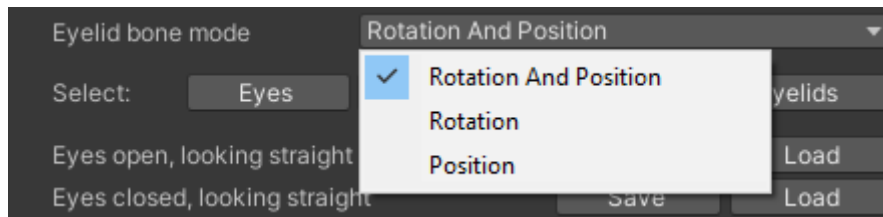
First, make sure the character looks straight (with respect to the character transform) by rotating the head and the eyes if necessary. If you set eye control to *Mecanim bones*, rotate the eye bones. If you set eye control to *Eye gameobjects*, rotate the gameobjects you assigned. The eyes should now look along the character's forward direction: the blue z axis that shows when the character is selected.



If you enabled eyelid control, set the eyelids to how you want them to be when the character looks straight with open eyes.

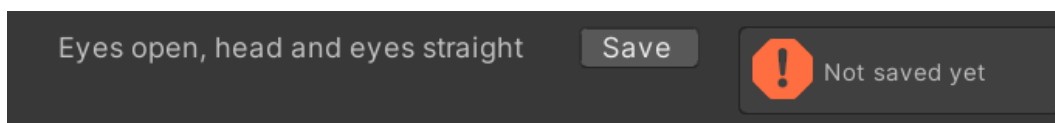
If you set eyelid control by bones, rotate or position the eyelids bones to the desired position. You can set **Eyelid bone mode** to only save the position, rotation, or both of the eyelid bones for the

different poses.

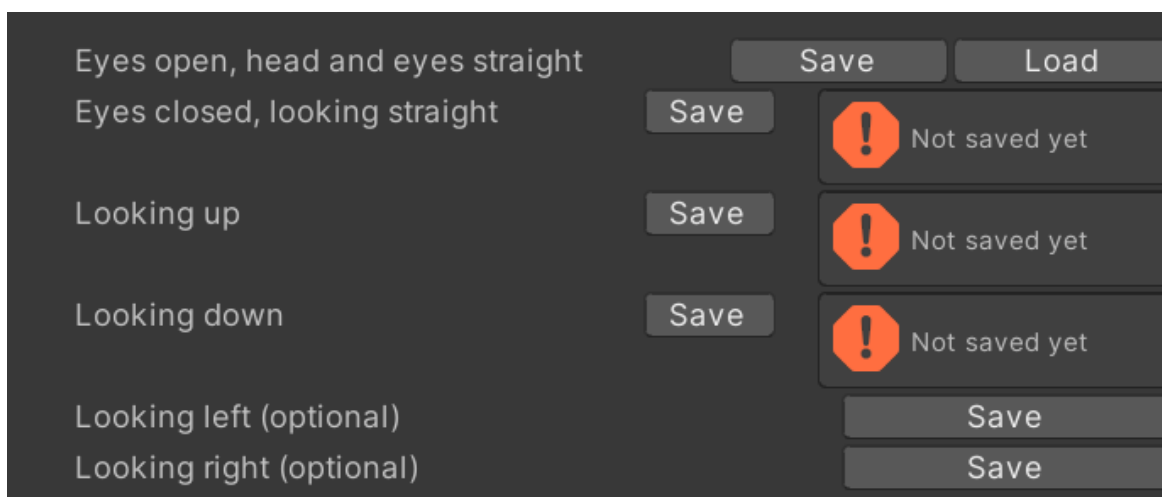


If you set eyelid control to blendshapes, set the blendshape values of the respective SkinnedMeshRenderers such that the eyelids are right.

When the eyes and eyelids are correct for looking straight with open eyes, press the **Save** button next to **Eyes open, head and eyes straight**:

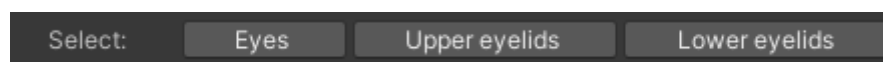


After that, more buttons will appear.



The **Eyes closed, looking straight** line will only appear if you enabled eyelid control.

For each line, set the corresponding eye and eyelid positions and press Save. The easiest way to do that is to use the convenience buttons that select either the eyes or the lower/upper eyelid bones and then adjust the eyes/eyelids for the respective pose:



For **Eyes closed, looking straight**, make sure the eyes are looking straight ahead again and close the eyelids by moving or rotating the upper lids down and the lower up, until they look right for a closed eyes position. This is usually easiest by first rotating both upper eyelids down until only part of the iris is visible on both sides:



Then rotate the lower eyelids up to close the eyes and press **Save** in the line **Eyes closed, looking straight**:



Now press **Load** in the line **Eyes open, looking straight** to open the eyes and make it easier to adjust the next poses.

For **Looking up**, rotate the eyes up as far as still looks good. This position will be saved as the maximum upward angle for the eyes. If you enabled eyelid control, adjust the eyelids for that position. In humans, when we look up, the upper eyelids move up a bit and the lower eyelids move up a bit as well, so the eyelids „follow“ the eyes. This adds a lot to the realism of eye movement. Then press **Save** in the line **Looking up**.



Reference picture:



If your eye bones or eye gameobjects are oriented such that it is difficult to let them look upwards because none of the rotation axes is perpendicular to „up“, just rotate the whole character so his/her head is looking along the global z axis and set the rotation pivot control to Global:



If your character has separate eyebrow objects or such whose blendshape values should follow those of the face renderer, you can either adjust them here when setting poses (so for looking up, adjust the face's eyelid blendshapes to look up, and adjust the eyebrow blendshapes as well). Or you can just adjust the blendshapes for the face and add the BlendshapeCopier component and set it to make the eyebrows follow the face blendshapes at runtime. See the BlendshapeCopier section below.

Now you can easily rotate the eyes around the global horizontal axis. You can rotate your character back after you have saved all positions.

Do the corresponding saving for **Looking down**: rotate the eyes as far down as looks realistic, adjust eyelids (by rotating and/or moving them) if you enabled eyelid control, then press **Save**.



Reference picture:



When using blendshapes for eyelids: If you can, use distinct blendshapes for "Eyes Closed" and for "Looking Down". This will make blinking when looking down look a bit more realistic. For example, one of the existing presets uses the blendshape Eye_Blink for the "Eyes Closed" configuration, and for the "Looking Down" configuration, to lower the upper eyelids, it doesn't use Eye_Blink again but Eye_Blink_L and Eye_Blink_R, which are independent blendshapes from Eye_Blink.

If they are not different but there is an overlap (e.g. you set the blendshape Eye_Blink to 100 for Eyes Closed, and to 20 for Looking Down), REM has to "blink straight", which means it has to reduce how much the eyelids follow the eyes up/down by how when blinking, to make sure that when the blink is "full", the eyelids are indeed fully closed.

If your model doesn't have such blendshapes and you find you have to reuse a blendshape like Eye_Blink for both "Eyes Closed" and "Looking Down", it's no big deal though.

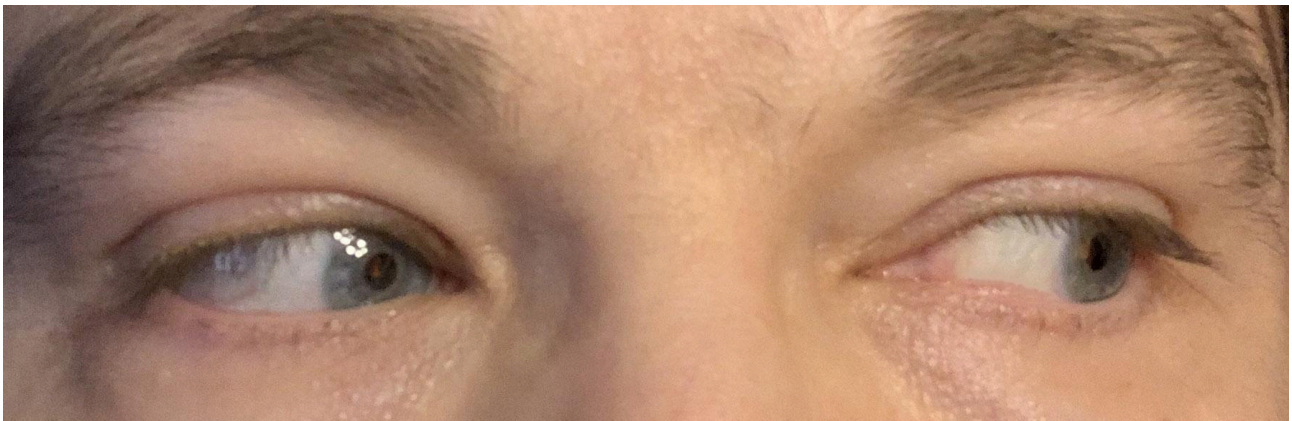
The same applies for "Looking Up": try to keep "Eyes Closed" and "Looking Up" distinct if you can. "Looking Up" doesn't have to be distinct from "Looking Down" though.

Optional: you can similarly save positions for **Looking left** and **Looking right**. You can set these to have finer control over how much the eyes can look to the sides (if you don't set this, the eyes will turn at most 30 degrees to the side). Furthermore, if you use blendshapes for eyelid control and have appropriate blendshapes like "Looking_left" etc., you can use these configurations to make the eyelid follow the eyes horizontally, so the eyelid bulge follows the pupil left or right, not just up and down. (The preset for Reallusion's CC4 characters already supports this. So does the preset for Reallusion's CC3+ characters that use ExPlus blendshapes. There is also a preset for DAZ characters with eyelid side-movement support; read the Readme_Blendshapes.txt file in the DAZ preset folder to find out which blendshapes you have to export from DAZ for it to work.)

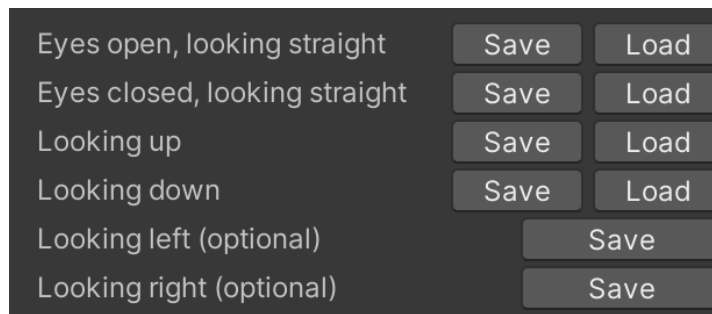
Note that if you use bones for eyelids, their poses are not saved for Looking Left/Right, only blendshape values.



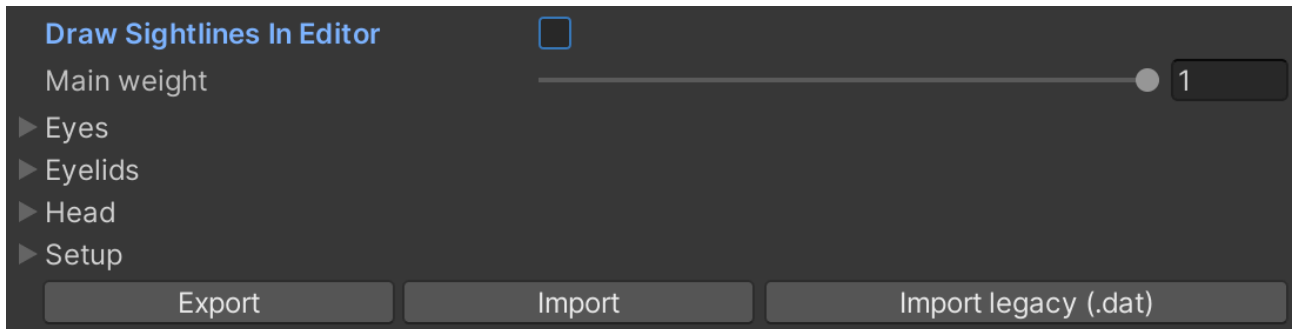
Reference image:



After you saved all required positions, the component knows all it needs about how to animate the lids. Click **Load** next to **Eyes open, looking straight** to let your character look straight ahead.



After completing the setup, you can adjust other parameters in the component.



Checking **Draw Sightlines In Editor** lets you see where the eyes are looking exactly during Play mode in the editor window.

Main weight is the main control for how much REM moves eyes, eyelids and the head: if set to zero, REM doesn't move anything at all. If set to 1, the more specific weights determine how much to move eyes, eyelids, and the head. Concretely, the total influence of REM on the eyes is

$$\text{main weight} * \text{eye weight}$$

The eye weight slider is in the eyes foldout.

The total influence of REM on the eyelids is

$$\text{main weight} * \text{eye weight} * \text{eyelids weight}$$

The total influence of REM on the head is

$$\text{main weight} * \text{head weight}$$

You can use main weight to fade REM in and out when you want to let a temporary animation take control of the character's eyes for example.

The other parameters are grouped into three foldouts: Eyes, Eyelids, and Head.

Eyes Foldout

Setting	Value
Eyes weight	1
Use Micro Saccades	<input checked="" type="checkbox"/>
Use Macro Saccades	<input checked="" type="checkbox"/>
Idle target horiz angle	10
Cross eye correction	1
Saccade speed	0.5
Micro saccades/min looking idle	45
Micro saccades/min looking at POI	80
Macro saccades per min	10

Eyes weight determines how much control REM has over moving the eyes, from 0 to 1. 0 means REM doesn't move the eyes and leaves them as they are (in default pose or controlled by animation etc.). 1 means REM fully controls the eye look direction.

The **Use Micro Saccades** checkbox determines whether the eyes do those little darts from time to time that human eyes usually do and that add to how lifelike the eyes seem.

The **Use Macro Saccades** checkbox determines whether the eyes do larger darts from time to time (similar to micro saccades, but less frequent and larger angles). Macro saccades are not used when the character is looking at the player's face or when you call the **LookAtPoiDirectly** function.

The default is to use both micro and macro saccades for most lifelike animation.

Idle target horiz angle: In Look Idly mode, the next look target is chosen within this number of angles horizontally relative to looking forward.

The **Cross Eye Correction** value determines how much to prevent the eyes from looking cross-eyed when fixating on an object close to the head. The default value should work fine for most setups, but if you find your character still looks cross-eyes for close objects, you can increase it.

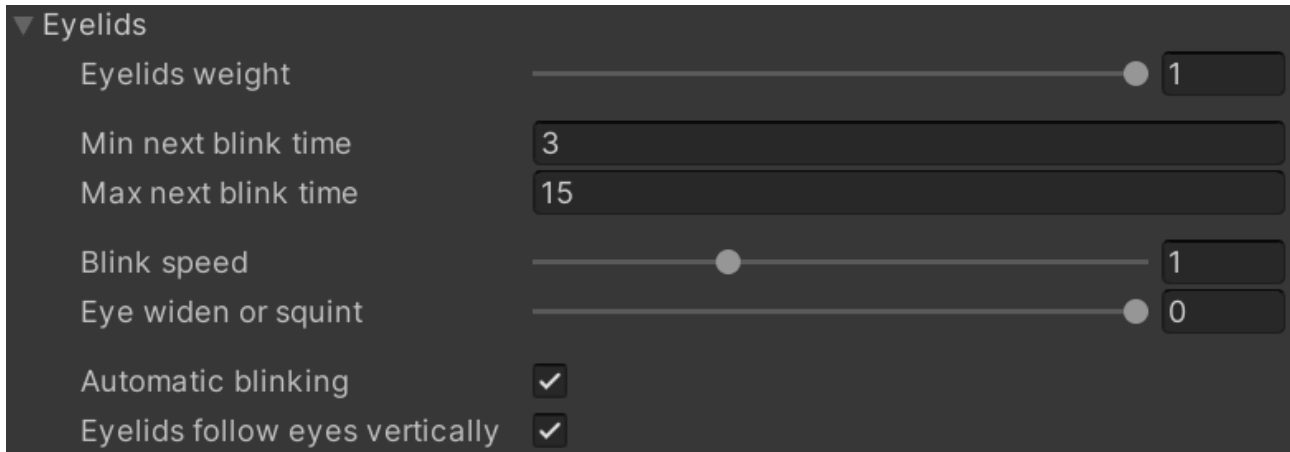
Saccade Speed determines how quick the eyes move during a saccade. 1 is the most realistic, but 0.5 looks better on most characters.

Micro saccades/min looking idle determines the average number of micro saccades per minute when looking around idly.

Micro saccades/min looking at POI determines the average number of micro saccades per minute when looking at a specific thing. This is usually higher than when looking idly.

Macro saccades per minute determines the average number of macro saccades per minute.

Eyelids Foldout



▼ Eyelids

Eyelids weight 1

Min next blink time

Max next blink time

Blink speed 1

Eye widen or squint 0

Automatic blinking ☒

Eyelids follow eyes vertically ☒

Eyelids weight determines how much control REM has over moving the eyelids, from 0 to 1. 0 means REM doesn't move the eyelids and leaves them as they are (in default pose or controlled by animation etc.). 1 means REM fully controls the eyelid movement.

You can control the frequency of automatic blinking by setting **Min Next Blink Time** and **Max Next Blink Time**. After a blink, the time until the next blink is a random number of seconds between min blink time and max blink time.

In general, women tend to blink a bit more than men: in one study men blinked 12 times a minute on average, women 19 times.

Blink Speed allows you to modify the blink speed. Keep it at 1 for the default blink speed.

The **Eye Widen Or Squint** value determines how much the eyes are widened (in surprise) or squinted. A value of 0 means no widening or squinting. A value greater than 0 means widening, a value less than 0 means squinting. Widening only works when eyelids are controlled by bones. Squinting works for both kinds of control of eyelids: bones or blendshapes.

If **Automatic Blinking** is checked, the character blinks automatically from time to time, with one blink separated from the other determined by Min Next Blink Time and Max Next Blink Time. If unchecked, the character doesn't blink unless you call `Blink()` on the `EyeAndHeadAnimator` component.

The **Eyelids Follow Eyes Vertically** checkbox determines whether the eyelids follow the eye's vertical movement a bit by moving up when the eye moves up and down when the eye moves down. This adds to the realism, so by default it is enabled.

Head Foldout

▼ Head

Head control

Transform

Head transform

None (Transform)

Neck transform

None (Transform)

Spine transform

None (Transform)

Head animation type

Hill Hybrid

Head weight

1

Neck horizontal weight

0.5

Neck vertical weight

0.5

Additional head tilt

Pitch angle

0

Roll angle

0

Yaw angle

0

Additional neck tilt

Pitch angle

0

Roll angle

0

Yaw angle

0

Reset head at frame start

☐

Head speed

Speed for switching target

1

Speed for following target

1

Head jitter

Use Head Jitter

☒

Jitter frequency

0.2

Jitter amplitude

1

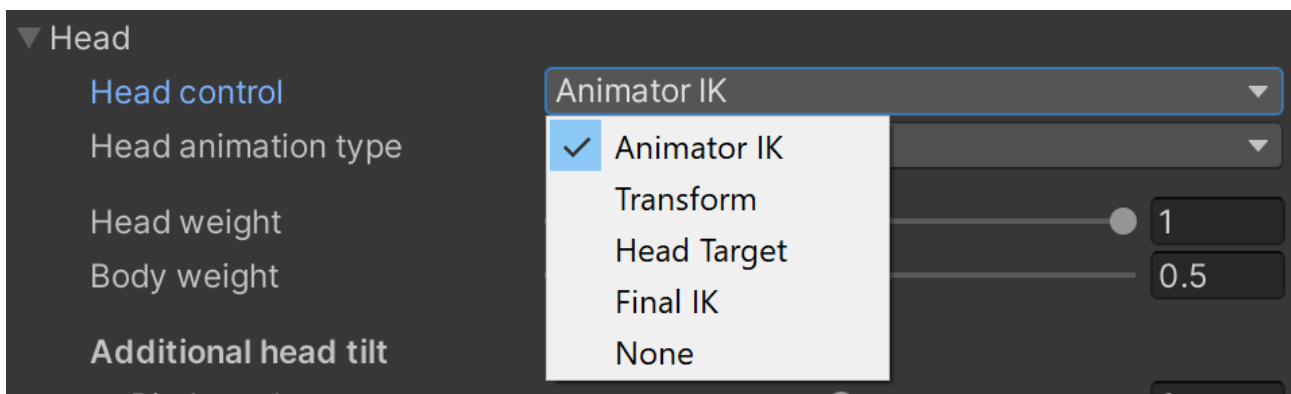
Up instead of spine angle

30

Limit head angle

0

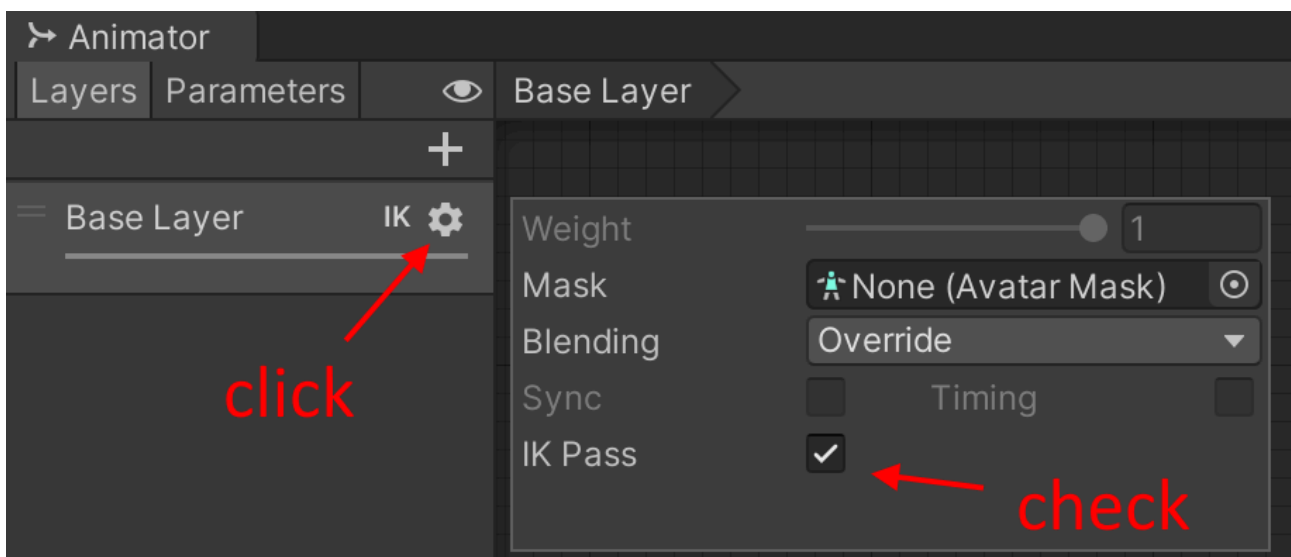
Set **Head Control** to how you want REM to control head movement:



Head Control: AnimatorIK

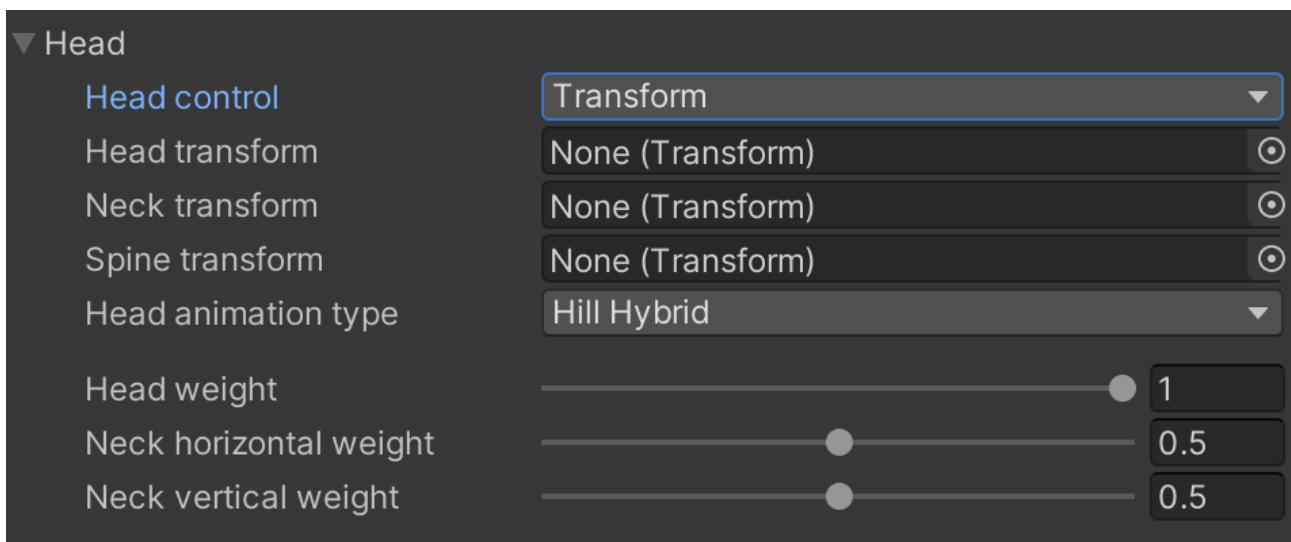
AnimatorIK uses Mecanim's OnAnimatorIK function to orient the head. Adjust **Head weight** and **Body weight** for your character. Mecanim's OnAnimatorIK seems to work best for some characters if **Head weight** is set to a lower value than 1, because otherwise the head “overshoots” when looking to the side.

If you use this option, make sure that your character's animator has the IK pass enabled:



Head Control: Transform

You can also choose **Transform** to control the head, which uses a custom algorithm.



You can assign a head, neck and spine objects (or bones). REM will orient the head (and neck if available, according to the neck weights) and use the spine as reference “forward orientation” for computing head angles. If your character is a Mecanim humanoid, you can leave these slots empty; REM will find appropriate bones on its own. If you don't like how OnAnimatorIK is orienting the head of your Humanoid character, choose **Transform** (and you can leave the slots for Head etc blank); it doesn't have issues like overshooting.

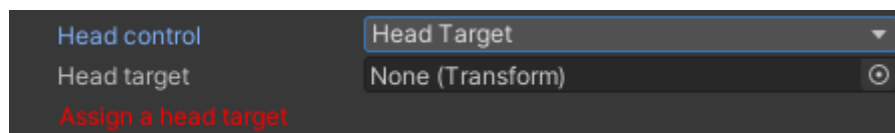
If you assign new object to either the head, neck, or spine slot, make sure to save the "Eyes open, head and eyes straight" pose in the Setup section again, so REM can store the correct straight pose for the new objects.

If your character is not humanoid, you have to assign the correct objects/bones for head and spine in the slots. This head gameobject will then be rotated to make the character's head look in the right direction. If you set this, please make sure the gameobject that has the EyeAndHeadAnimator component's forward direction is along the head and eyes' forward direction, so its forward direction is where the character is „looking“ along. Also make sure the object has a parent gameobject.

If your character's head is not animated by the animator component, the head orientation might drift over time when using Head Control Transform; the head is still looking in the right direction, but its Up direction wanders. In that case, check **Reset head at frame start** to fix the issue.

Head Control: Head Target

Choosing **Head Target** for **Head Control** lets REM keep updating the position of the assigned object in the scene.



Create an empty object in the scene and assign it to the *Head target* slot. You can then assign this object to another system as well that orients the head, for example an Animation Rigging setup on your character. REM will move this target and the other system will rotate the head to look at it.

Head Control: Final IK

If you have **Final IK** in your project and want to use that to rotate the head, add FinalIK's

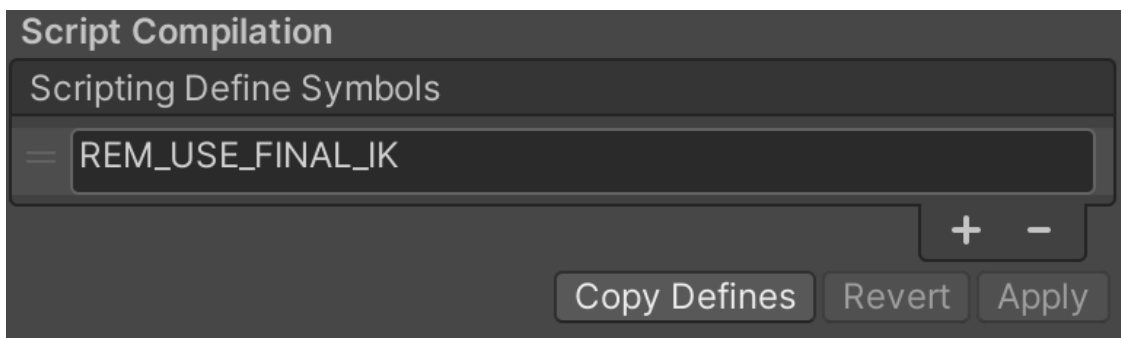
LookAtIK component to the character (and optionally a FullBodyBipedIK component if you need it) and set Head Control to **Final IK**. In addition, the symbol REM_USE_FINAL_IK needs to be defined in the Project Settings for REM to be able to use FinalIK.

Open *Project Settings* and in the *Player* tab under *Other Settings*, add REM_USE_FINAL_IK to *Scripting Define Symbols*.

For Unity 2019:

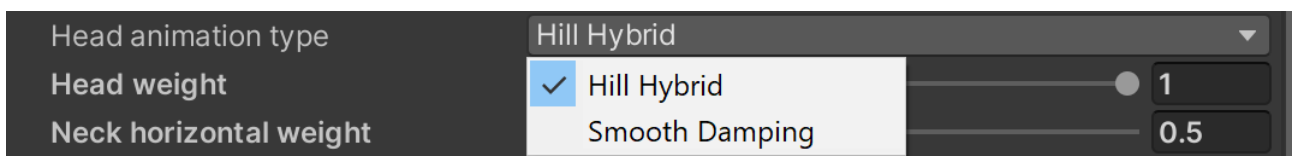


For Unity 2020 or newer:



You might want to set LookAtIK's Head Weight slider to 1, because REM already takes head limits into account and with a slider value of 1 the character will look directly at their targets.

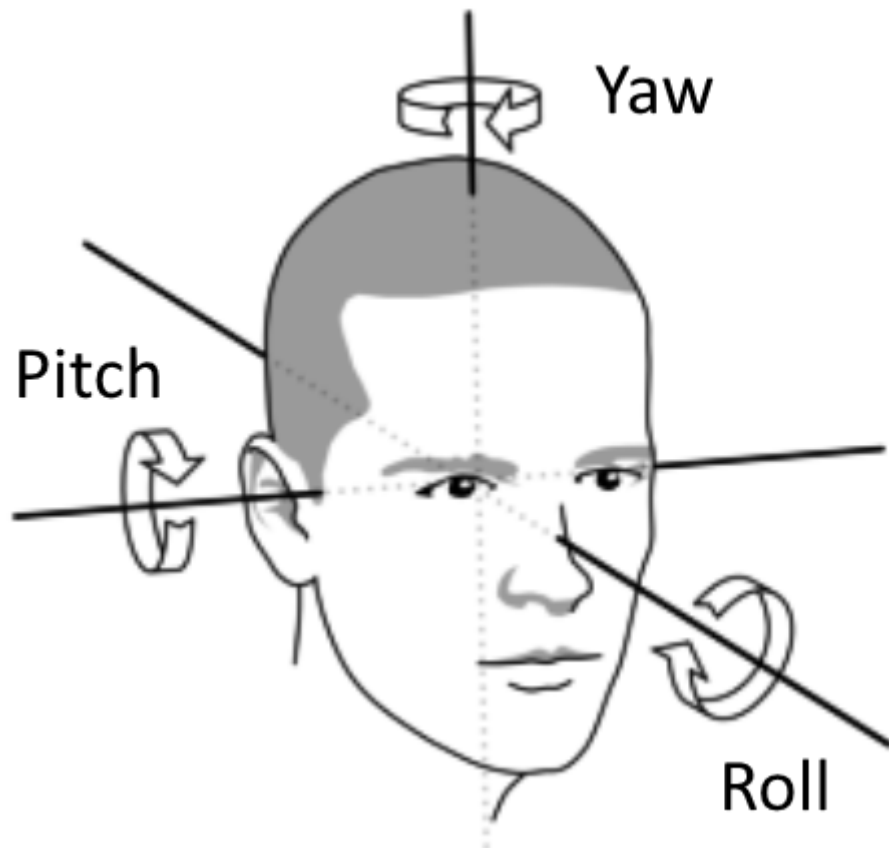
Head Animation Type



The default head animation type is HillHybrid, an experimental method to make the head rotation look more realistic. If you want, you can switch to the old SmoothDamping method instead.

The **Head weight** slider value determines how much the script controls the character's head movement when looking at targets. A value of 0 means the head is controlled by animation only.

Additional head tilt and **Additional neck tilt** angles let you adjust the head and neck after the head has been orientated to look at the target.



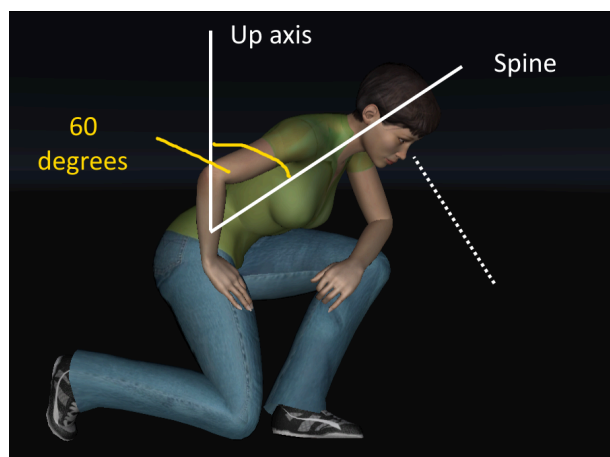
You can animate this/set it through code in case you want your character to roll their head to express curiosity or so.

The **Head Speed for Switching Target** value allows you to increase or decrease the head turn speed when switching the look target from one to another.

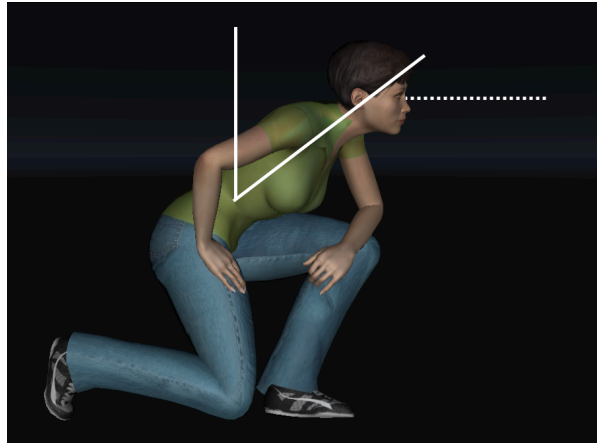
The **Head Speed for Following Target** value allows you to increase or decrease the speed with which the head tracks its current look target.

If **Use Head Jitter** is checked, the head will keep moving slightly when looking at a point, to make it look more natural and less like a stiff robot. You can control the parameters of the head jitter with **Head Jitter Frequency** and **Head Jitter Amplitude**.

The **Up Instead of Spine Angle** is used when choosing where to look when *looking idly*. Normally the character chooses a point in the plane perpendicular to the spine. So if the character is bending down with their spine at an angle of 60 degrees to the world Up axis, they will look around the area perpendicular to the spine, in this case at the ground:



But if `UpInsteadOfSpineAngle` is set to e.g. 70, then since the current spine angle of 60 is within this threshold of 90, the world Up axis will be used, and the character will look horizontally in front of them when looking idly:



Adjust `UpInsteadOfSpineAngle` if you find that your character whose animation keeps them bending the spine looks too much at the ground or ceiling and you would rather they looked horizontally in front of them.

The **Limit Head Angle** value determines how much to limit the head's left/right rotation. At zero, the character turns his or her head towards the point of interest, at larger values the head is kept straight ahead and only the eyes look at the target point.

Presets

The **Export** and **Import** buttons let you save and load presets. Once you set up the component, you can export the settings to a file. Similar characters can then be quickly set up by just dragging the `LookTargetController` and `EyeAndHeadAnimator` components onto them and importing the saved file. For a preset to be applicable to a character, the character needs to have a compatible structure with respect to the eye bones/blendshapes etc. For example, if your setup uses eyelid bones, then the names of the eyelid bones must be the same in the character you saved the preset from and the one you want to apply it to.

There are already presets in the folder Presets for these type of characters: MakeHuman, Autodesk Character Generator, Reallusion Character Creator, DAZ Studio and UMA. If your character is one of these types, just drag a `LookTargetController` and a `EyeAndHeadAnimator` component on him, import the appropriate file from `RealisticEyeAnimations/Presets` and it should work. If you don't use the default eye shapes but random or custom ones, you might have to adjust the settings.

Adding the Scripts at Runtime

You can add the scripts at runtime (for example if you generate your characters at runtime) and load a preset to have their eyes animated correctly. Here is an example of how you could add the scripts

at runtime. Copy the preset to a folder in your project called StreamingAssets (a folder with that name will have all its files included unmodified in the final build). Then, after you generated a character in your code (let's say you saved it in the `GameObject` variable `newCharacterGameObject`), use this code (except if you use Android or WebGL, see below):

```
EyeAndHeadAnimator eyeAndHeadAnimator =
newCharacterGameObject.AddComponent<EyeAndHeadAnimator>();

eyeAndHeadAnimator.ImportFromFile(Application.streamingAssetsPath +
"/mypreset.json");

LookTargetController lookTargetController =
newGO.AddComponent<LookTargetController>();

lookTargetController.Initialize();
```

If you are using **Android** or **WebGL**, the way the contents of the StreamingAssets folder is accessed is different. Instead of the above code, use this:

```
#if UNITY_EDITOR || UNITY_STANDALONE

    EyeAndHeadAnimator eyeAndHeadAnimator =
newCharacterGameObject.AddComponent<EyeAndHeadAnimator>();

    eyeAndHeadAnimator.ImportFromFile(Application.streamingAssetsPath +
"/mypreset.json");

    LookTargetController lookTargetController =
newCharacterGameObject.AddComponent<LookTargetController>();

    lookTargetController.Initialize();

#else

    StartCoroutine(LoadPreset("mypreset.json"));

#endif
```

And in the same class, add this code:

For Android:

```
using System.Collections;
using System.IO;

IEnumerator LoadPreset(string presetFilename)
{
    string path = "jar:file://" + Application.dataPath + "!/assets/" +
presetFilename;

    WWW loadPreset = new WWW(path);

    yield return loadPreset;

    string newPath = Application.persistentDataPath + "/" + presetFilename;
    File.WriteAllBytes(newPath, loadPreset.bytes);

    EyeAndHeadAnimator eyeAndHeadAnimator =
newCharacterGameObject.AddComponent<EyeAndHeadAnimator>();

    eyeAndHeadAnimator.ImportFromFile(newPath);
```

```
LookTargetController lookTargetController =  
newCharacterGameObject.AddComponent<LookTargetController>();  
lookTargetController.Initialize();  
}
```

For WebGL:

```
IEnumerator LoadPreset(GameObject character, string presetFilename)  
{  
    string uri = Application.streamingAssetsPath + "/" + presetFilename;  
    UnityWebRequest www = UnityWebRequest.Get(uri);  
    yield return www.SendWebRequest();  
    EyeAndHeadAnimator eyeAndHeadAnimator =  
    character.AddComponent<EyeAndHeadAnimator>();  
    eyeAndHeadAnimator.ImportFromJson(www.downloadHandler.text);  
    LookTargetController lookTargetController =  
    character.AddComponent<LookTargetController>();  
    lookTargetController.Initialize();  
}
```

Using UMA (Unity Multipurpose Avatar)

If you use UMA, you need to add the components at runtime because the characters are generated at runtime. The previous section described how to do this. Just use the existing preset UMA.json from the preset folder. In the case of UMA, you might need to wait a frame after the character is generated for all the bones to be added before applying the scripts. Here's how you can do this: after generating the character, use this code:

```
StartCoroutine (AddREM (newCharacterGameObject));
```

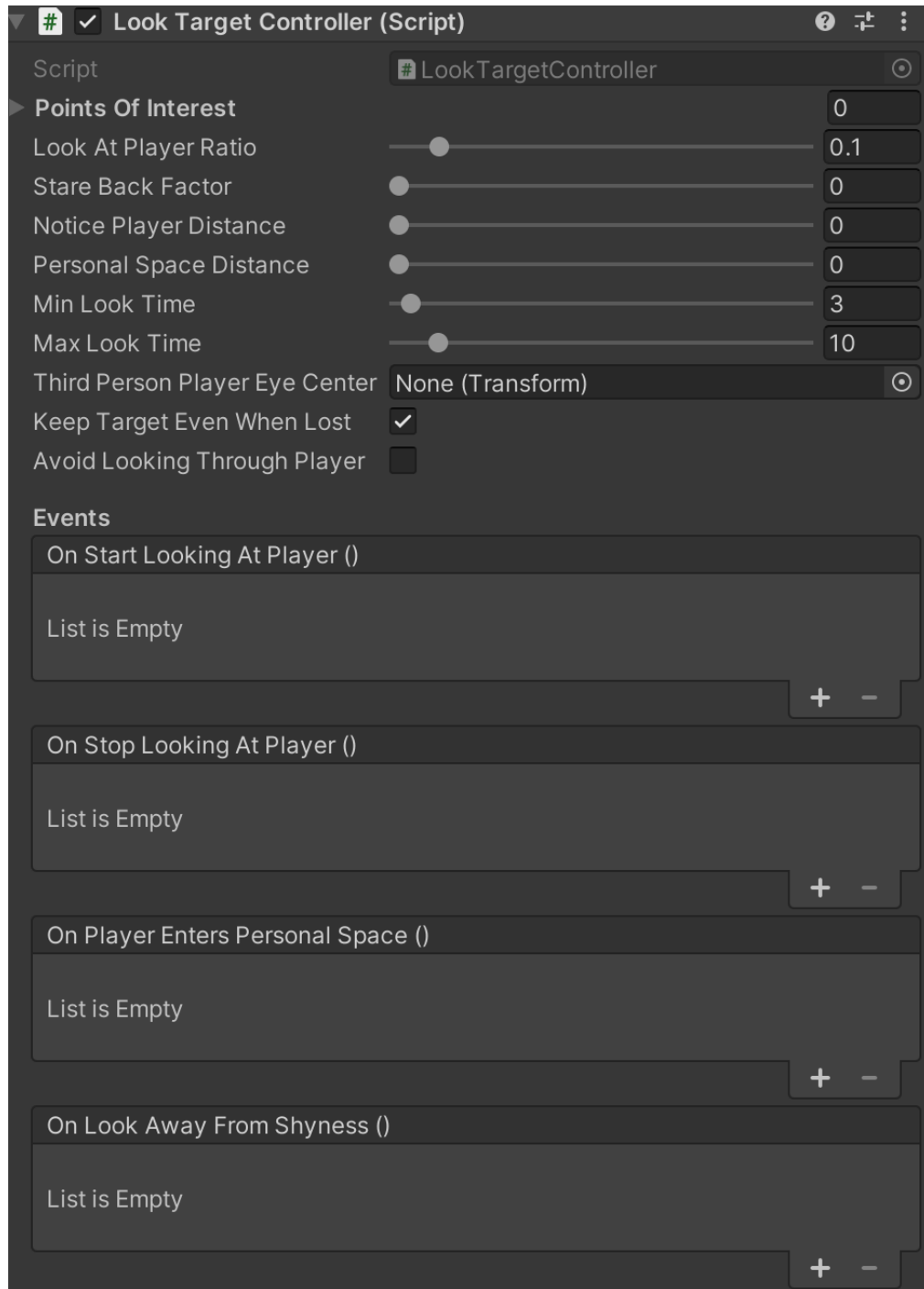
Here is the function AddREM (it adds the components after one frame to make sure all bones have been added by UMA):

```
IEnumerator AddREM(GameObject newGO)
{
    yield return null;
    yield return null;

    EyeAndHeadAnimator eyeAndHeadAnimator =
newGO.AddComponent<EyeAndHeadAnimator>();
    eyeAndHeadAnimator.ImportFromFile (Application.streamingAssetsPath +
"/UMA.json");

    LookTargetController lookTargetController =
newGO.AddComponent<LookTargetController>();
    lookTargetController.Initialize();
}
```

LookTargetController



Look Targets

The **LookTargetController** component lets you control where your character is looking. By default, the character looks around idly in random directions (close to straight ahead), and sometimes at the player if the player is in view. If you have specific objects in the character's environment that you want the character to choose as look targets, drag them into the array **Points of Interest**. The character will look at a random object from that list for some time, then choose another object from the list to look at.

If the player is in the character's view, the slider value of **Loot at Player Ratio** determines how often the character chooses the player as next target to look at: for example, a slider value of 0.1 means the character chooses a random direction or an object from the Points of Interest list 90% of the time, and the player as look target 10% of the time. To not have the character look at the player, set this to 0.

The slider **Stare Back Factor** determines how quickly the character looks back at the player if the character sees the player keep staring at him or her (so how sensitive the character is to being stared at).

The slider **Notice Player Distance** lets the character notice and look at the player when the character is looking idly and the player comes closer than this distance for the first time. A value of 0 means the player coming closer has no effect. A value of 2 means if the character hasn't noticed the player before and the player comes into view and is closer than 2 units, the character starts looking at the player.

If you set the slider **Personal Space Distance** to a value greater than zero and the player comes closer than this distance, the character looks away (avoiding the player in a „shy“ way).

The time the character looks at a target before choosing another target is a random number of seconds between **Min Look Time** and **Max Look Time**.

Third Person Player Eye Center: Realistic Eye Movements uses the main camera as Player by default (for settings like Look at Player Ratio), so if you are developing a 3rd person game, where the main camera does not show the player's perspective, you need to tell EyeAndHeadAnimator where the player's eyes are. For this, you assign a transform that is always located between the player character's eyes to the **Third Person Player Eye Center** slot of the non-player character's EyeAndHeadAnimator component (the one who is supposed to look at the player). You can create an empty gameobject if necessary, position it between the player character's eyes and parent it to the player's head bone. Also make sure the transform's forward direction is the same as the player head's forward direction. *Note: Player Eye Center is the center of the player character eyes, not the eye center of the character which has the EyeAndHeadAnimator component that you are editing!*

The checkbox **Keep Target Even When Lost** determines whether the character keeps trying to look at the target (by turning the head and eyes as much as possible) even when the target gets behind the character, so out of sight. If the checkbox is off, the character will stop tracking the target (and return to looking around idly or at another point of interest).

If you check the checkbox **Avoid Looking Through Player**, the character will try to avoid choosing look targets that happens to be on the other side of the player. You can use this to avoid the character seemingly looking at the player when they are actually looking at the target behind the player.

There are four events in LookTargetController that you can subscribe to:

OnStartLookingAtPlayer, **OnStopLookingAtPlayer**, **OnPlayerEntersPersonalSpace**, and **OnLookAwayFromShyness**. For example, you might make your character smile every time he or she looks at the player:

```
lookTargetController.OnStartLookingAtPlayer.AddListener(OnStartLookingAtMe);

void OnStartLookingAtMe() {
    if ( lookTargetController.IsPlayerLookingAtMe() )
        StartSmiling();
}
```


BlendshapeCopier

If your character has SkinnedMeshRenderer with blendshapes that should follow the blendshapes of the character's face (for example, the eyebrow render's blendshape value for the eyeWide blendshape should always be the same as the value for the face's eyeWide blendshape so the eyebrow moves correctly with the face skin), you can either adjust both objects' blendshape values when storing positions for looking up/down etc., or just do that for the main face renderer and add the BlendshapeCopier component. This component can be set to make some objects' blendshape values follow the values of a source object at runtime. To do that, add the BlendshapeCopier component, set its SourceRenderer parameter to the SkinnedMeshRenderer whose blendshapes you used to make the eyelids move in the Setup section, and add all objects as TargetRenderers whose blendshape values you want to follow those of the source renderer. The BlendshapeCopier will compare the names of the blendshapes between the source and each target object and for all blendshapes in the target render that have a blendshape of the same name in the source renderer, it will copy the blendshape's value in the source renderer and apply it to the target renderer at runtime each frame. This makes it easy to setup REM eyelid movement just by adjusting the blendshapes in the face renderer during setup and let other objects like eyebrows or beards move accordingly without having to adjust them during setup as well.

VR: Virtual Reality Headsets

By default, the script uses the camera tagged MainCamera to determine the player's position. If you use Unity's VR support, the script uses that to find out where the player's left and right eyes are. When characters look at the player, they will cycle randomly among the left and right eye and the mouth as look target (the so called „social triangle“ people use when looking at someone's face). You don't need to change anything for the script to work in a VR scene.

Script API

You can call these functions on the LookTargetController component for more control (if possible, call them from your LateUpdate function, not your Update function):

LookTargetController.cs

Delegates

LookTargetController has optional delegates you can set for more control:

- **IsPlayerInViewDelegate**
 - Used to determine whether the player is in view, and can be looked at. Register your own function here if you want some more custom logic, for example based on distance, raycasting to see whether there are occluding things in between etc.
- **IsPlayerLookingAtMeDelegate**
 - Used to determine whether that player is looking at the character (for example to decide whether to stare back). Register your own function here if you want some more custom logic, for example based on distance, raycasting to see whether there are occluding things in between etc.
- **GetPOIsDelegate**
 - Called every time the character is ready to look at a new thing. Returns the things that the character can choose from to look at. So instead of assigning a fixed list at start, you can update the points of interest whenever needed via this delegate. This is useful if there are changing/appearing/disappearing things in the environment that the character can look at. For example, in VR House Disco, every time the character is ready to look at a new target, this delegate returns the other dancers in front of the character as potential look targets to choose from.

You can set a delegate like this:

```
lookTargetController.GetPOIsDelegate = UpdatePOIs;
```

And here is an example delegate function:

```
List<Transform> UpdatePOIs()
{
    pois.Clear();

    foreach ( EyeAndHeadAnimator otherREM in eyeAndHeadAnimators )
        if ( IsInView(otherREM.GetOwnEyeCenter()) )
            pois.Add(otherREM.GetOwnEyeCenterXform());

    return pois;
}
```

The whole code for this example is [here](#), it's from [VR House Disco](#). It lets the character look at other characters in view.

Methods

void **Blink()**

Makes the character blink.

void **ClearLookTarget()**

Clears the current look target and makes the character look straight ahead, until a new order is issued (so the character doesn't notice the player even when he should according to settings like Look At Player Ratio, for example).

bool **IsLookingAtPlayer()**

Returns whether the character is currently looking at the player.

bool **IsLookingIdly()**

Returns whether the character is currently looking around idly.

bool **IsPlayerInView()**

Returns whether the character can see the player (only checks viewing angles, doesn't check

range or for visual obstacles in between character and player)

bool IsPlayerLookingAtMe()

Returns whether the character is seeing the player looking at the character.

void LookAtPlayer(float duration=-1, float headLatency=0.075f)

Looks at the player for duration seconds. To keep looking until a new command is given, set duration to -1. HeadLatency determines how much later the head starts moving than the eyes.

void LookAroundIdly()

Starts looking around in random directions (close to straight ahead) or at points of interest if the list Points of Interest has objects. This uses the component's settings like Look At Player Ratio etc., so under certain conditions the player will be noticed or avoided.

void LookAtPoiDirectly(Transform targetTransform, float duration=-1, float headLatency=0.075f

)

Looks at a specific transform for duration seconds. Keeps following the transform with the eyes if the transform moves. After the duration has passed, the character continues looking according to the component settings (like Look At Player Ratio etc.). To keep looking until a new command is given, set duration to -1. HeadLatency determines how much later the head starts moving than the eyes.

void LookAtPoiDirectly(Vector3 targetPoint, float duration=-1, float headLatency=0.075f)

Looks at a specific point for duration seconds. To keep looking until a new command is given, set duration to -1. HeadLatency determines how much later the head starts moving than the eyes.

If you need more control over the animation than the components can provide in their current state, please let me know at tore.knabe@gmail.com.

Good luck with your projects!

Tore Knabe