Realistic Eye Movements

Version 2.0.2

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 at line at tamulur@yahoo.com!

For discussions of this asset, visit the Unity forum at http://forum.unity3d.com/threads/released-realistic-eye-movements.297610/.

The webpage for this asset is http://tore-knabe.com/unity-asset-realistic-eye-movements

What this Asset Does

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 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 http://tore-knabe.com/experiments-with-head-animation.

This asset only provides scripts to control animation. 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. You can compare the three assets by pressing **1, 2** or **3** in the webplayer demo.

How to Use

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

`LookTargetController.cs`

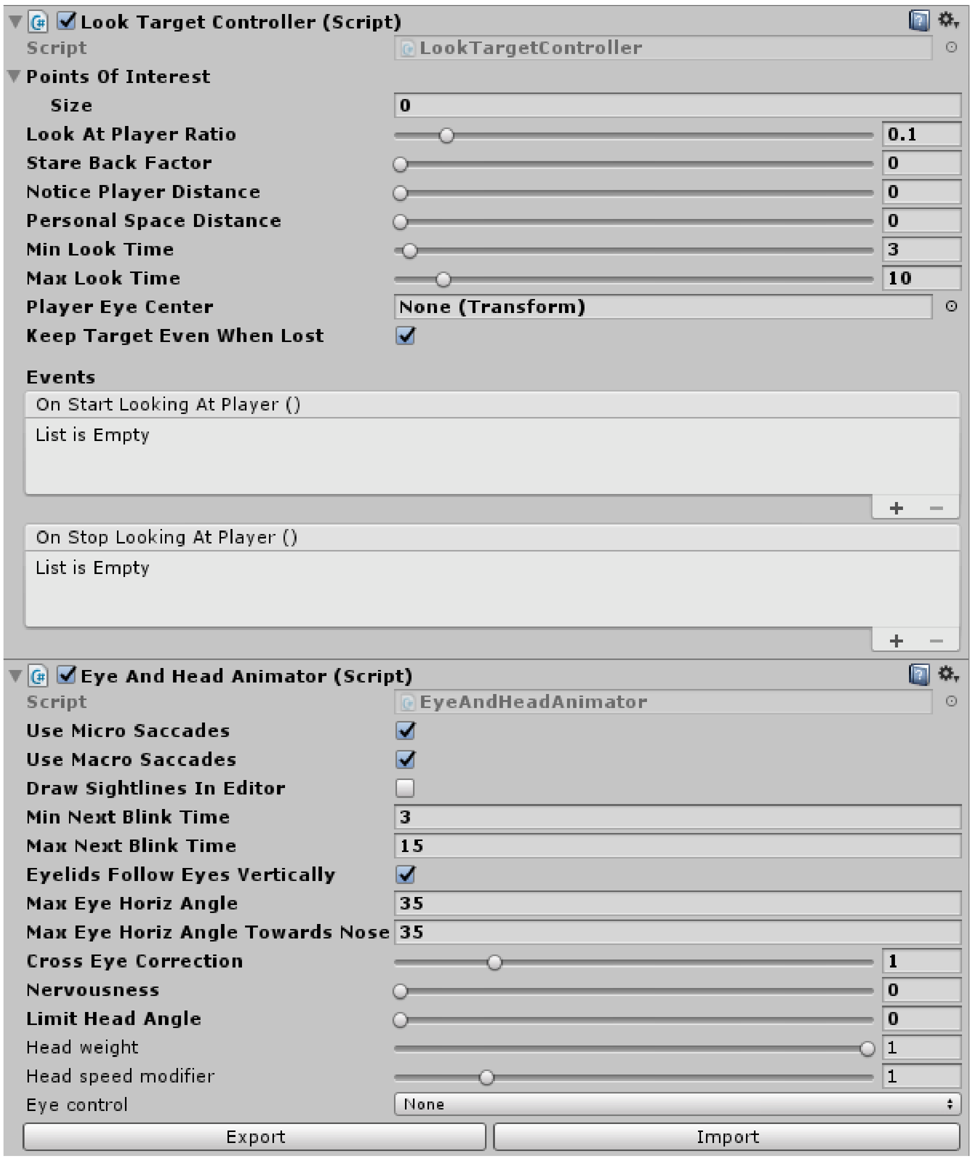
Chooses what to look at and when to switch look targets.

`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 neet to modify EyeAndHeadAnimator.cs.

Drag the two scripts onto your character game object in the scene hierarchy. The new components should look like that:



Look Target Controller (Script)	
Script	LookTargetController
Points Of Interest	
Size	0
Look At Player Ratio	0.1
Stare Back Factor	0
Notice Player Distance	0
Personal Space Distance	0
Min Look Time	3
Max Look Time	10
Player Eye Center	None (Transform)
Keep Target Even When Lost	✓
Events	
On Start Looking At Player ()	List is Empty
On Stop Looking At Player ()	List is Empty

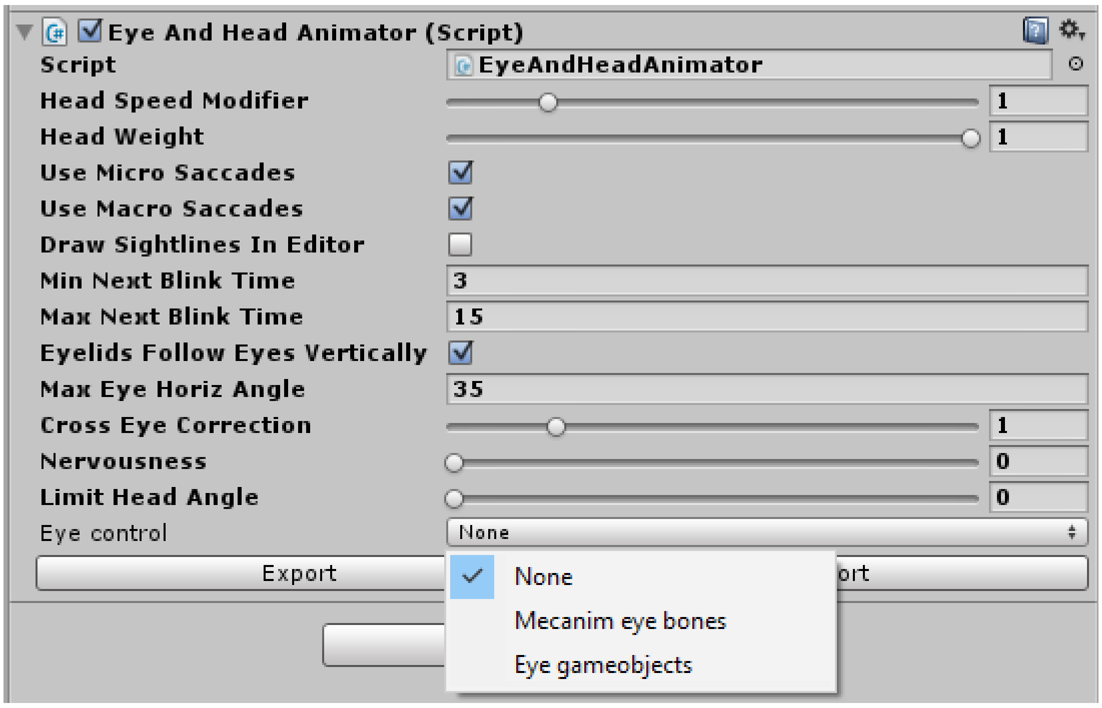
Eye And Head Animator (Script)	
Script	EyeAndHeadAnimator
Use Micro Saccades	✓
Use Macro Saccades	✓
Draw Sightlines In Editor	☐
Min Next Blink Time	3
Max Next Blink Time	15
Eyelids Follow Eyes Vertically	✓
Max Eye Horiz Angle	35
Max Eye Horiz Angle Towards Nose	35
Cross Eye Correction	1
Nervousness	0
Limit Head Angle	0
Head weight	1
Head speed modifier	1
Eye control	None

The asset only works correctly when the head is looking straight ahead with respect to the character transform when the scene starts. If your character's head is not looking straight in the editor (e.g. looking to the character's left), please rotate it to look straight.

Eyes

> *If your character was generated with Mixamo Fuse, Autodesk's Character Generator, MakeHuman or MCS (Morph3D), 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.*

To tell the scripts how to control the eyes, select either *Mecanim eye bones* or *Eye gameobjects* from the box **Eye control**.

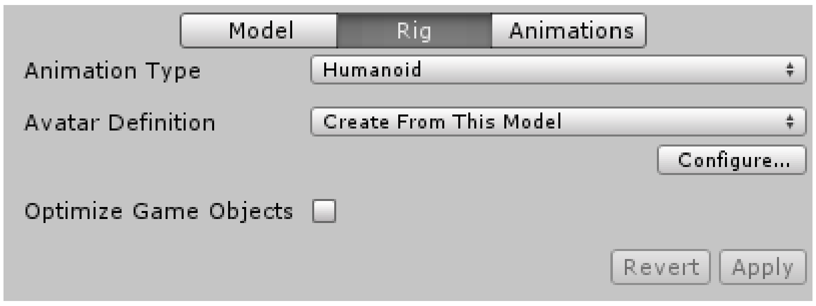


Eye And Head Animator (Script)

Script	EyeAndHeadAnimator
Head Speed Modifier	1
Head Weight	1
Use Micro Saccades	✓
Use Macro Saccades	✓
Draw Sightlines In Editor	
Min Next Blink Time	3
Max Next Blink Time	15
Eyelids Follow Eyes Vertically	✓
Max Eye Horiz Angle	35
Cross Eye Correction	1
Nervousness	0
Limit Head Angle	0
Eye control	None

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 eyes 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:



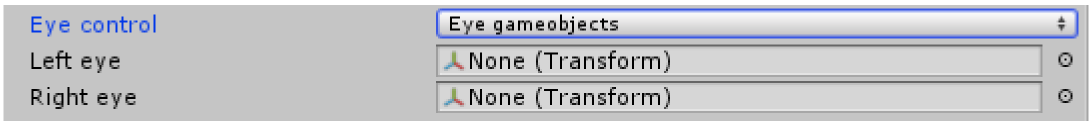
Animation Type: Humanoid

Avatar Definition: Create From This Model

Optimize Game Objects: ☐

Make sure Optimize Game Objects is unchecked or that you expose the eye bones as extra transforms. If you use bones for eyelids, then don't check Optimize Game Objects, because with this option checked Unity will decouple the skinned eyelid mesh from the eyelid bones, even if you expose them, so the eyelids will not move when their eyelid bones are moved.

If you choose *Eye gameobjects*, you can select game objects in your character's object hierarchy to be controlled by the script. Find them in your character's hierarchy and drag each into its corresponding slot in the component:



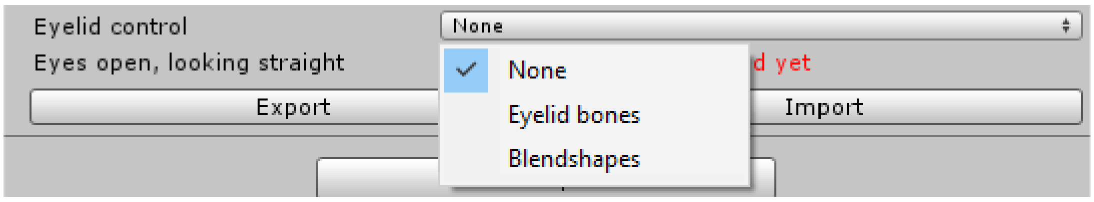
Eye control: Eye gameobjects

Left eye: None (Transform)

Right eye: None (Transform)

Eyelids

If your character has eyelids that can be controlled by bones or blendshapes, the EyeAndHeadAnimator component 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 *Eyelid bones* or *Blendshapes*.

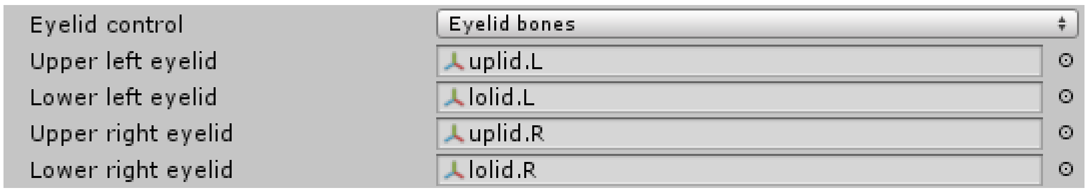


Eyelid control: None

Eyes open, looking straight

d yet

If you choose *Eyelid bones*, find the eyelid bones in your character's hierarchy and drag them into the corresponding slots:



Eyelid control	Eyelid bones
Upper left eyelid	uplid.L
Lower left eyelid	lolid.L
Upper right eyelid	uplid.R
Lower right eyelid	lolid.R

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

If you choose Blendshapes, you don't need to assign anything.

If your character has more than one bone per eyelid (e.g. Daz3D characters):

The asset currently supports only one bone per eyelid. If your character has several bones per eyelid, you need to choose one of them as the main bone and parent the other bones of that lid to that bone, then assign the main parent bone to the REM asset as the single bone for this lid. When you move or rotate that bone, the other bones of the lid will follow. You do that by dragging some of the bones in the Hierarchy panel of the editor to an new place in their hierarchy. Let's assume your bone hierarchy for the head has three bones for each eyelid. For the upper left eyelid, the hierarchy looks like this:

```
- head bone
   - LEyelidUpperInner
   - LEyelidUpperOuter
   - LEyelidUpper
```

Drag LEyelidUpperInner to be a child of LEyelidUpper. Also do that to LEyelidUpperOuter. The new hierarchy should look like this:

```
-head bone
   - LEyelidUpper
      - LEyelidUpperInner
      - LEyelidUpperOuter
```

Do the same to the lower and the right eyelids. Then you can assign LEyelidUpper as the single bone for the left upper eyelid, and respectively for the others.

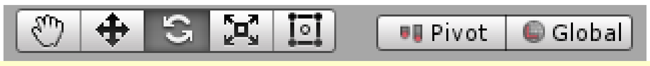
If the eyelid bones' pivots are not in the center of the eyeball but it would be easier it they were (so you can rotate the parent eyelid bone instead of moving it up or down), you can create a new eyelid parents, one for the upper and one for the lower eyelid bones, and make sure this paren't pivot is in the center of the eyeball, by selecting the respective eyeball first, then choose Create New GameObject. The new GameObject will have its pivot where the eyeball's pivot is. You can then drag the new GameObject out from under the eyeball and to where the other eyelid bones are, then make the respective eyelid bones children of this new GameObject and use this new parent as the eyelid bone for REM.

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. The following steps might be easier if you lock the inspector tab to 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 by rotating 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.

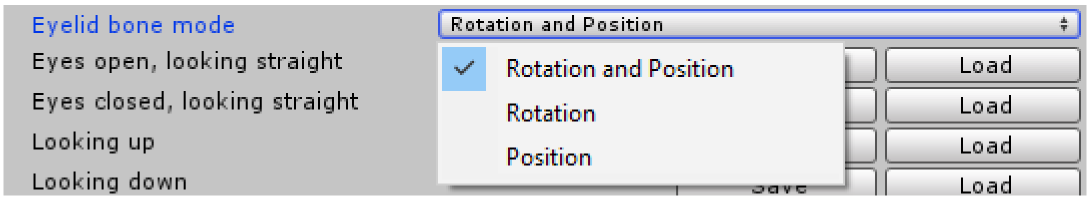
Hint: If your eye bones or eye gameobjects are oriented such that it is difficult to let them look upwards because none of the rotation axes is perpendicular to „up“, just rotate the whole character so his/her head is looking along the global z axis and set the rotation pivot control to *Global*:



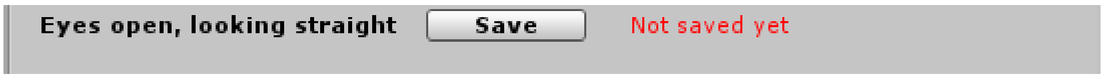
You can rotate your character back after you have saved all positions.

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

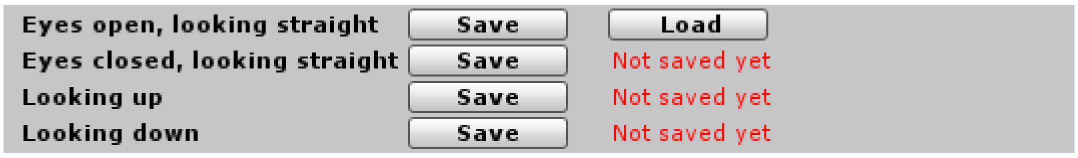
If you set eyelid control by bones, rotate or position the eyelids bones to the desired position. You can set **Eyelid bone mode** to only save the position, rotation, or both of the eyelid bones for the different poses.


Eyelid bone mode: Rotation and Position

If you set eyelid control to blendshapes, set the blendshape values 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, looking straight**:


Eyes open, looking straight Save Not saved yet

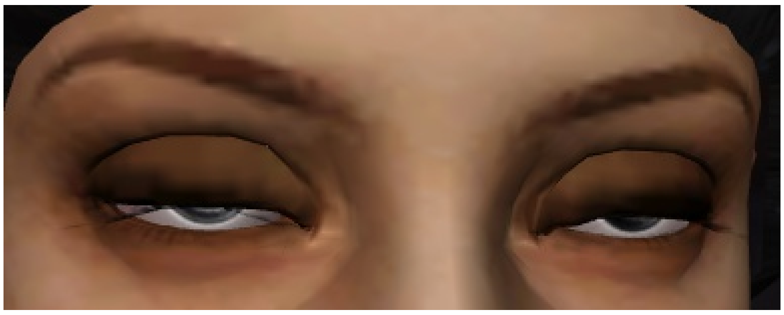
After that, more buttons will appear.



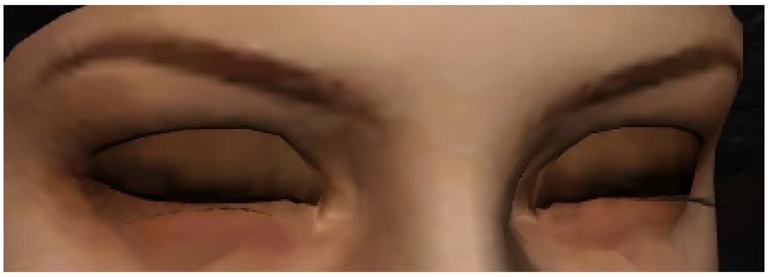
Eyes open, looking straight	Save	Load
Eyes closed, looking straight	Save	Not saved yet
Looking up	Save	Not saved yet
Looking down	Save	Not saved yet

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. You can reset the positions to the default looking straight, eyes open position (to make it easier to set the next position from there) by pressing **Load** in the line **Eyes open, looking straight**.

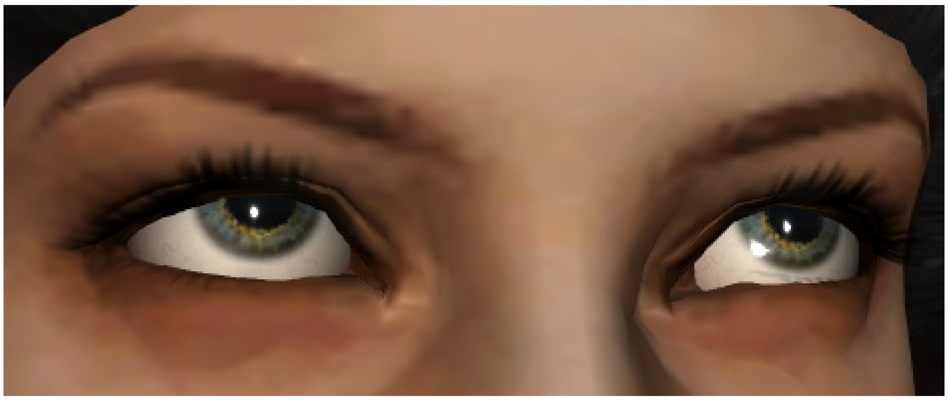
For **Eyes closed, looking straight**, make sure the eyes are looking straight ahead and close the eyelids by moving or rotating the upper down and the lower up, until they look right for a closed eyes position. This is usually easiest by first rotating both upper eyelids down until only part of the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



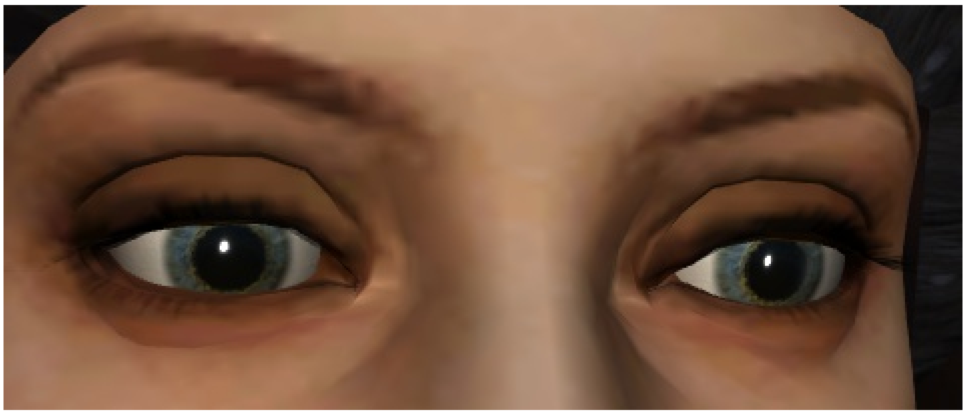
Then rotate the lower eyelids up to close the eyes and press **Save**:



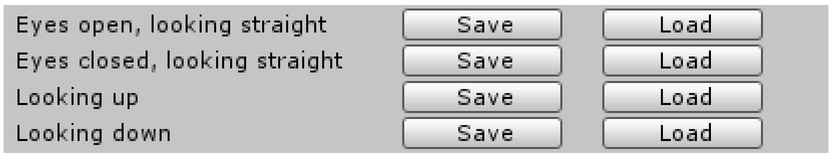
For **Looking up**, rotate the eyes up as far as still looks good. This position will be saved as the maximum upward angle for the eyes. If you enabled eyelid control, adjust the eyelids for that position. In humans, when we look up, the upper eyelids move up a bit and the lower eyelids move up a bit 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**.



Do the corresponding saving of position 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**.



After you saved all required positions, the component has all it needs to know about how to animate the lids. If you want you can click **Load** next to **Eyes open, looking straight** to let your character look straight ahead.



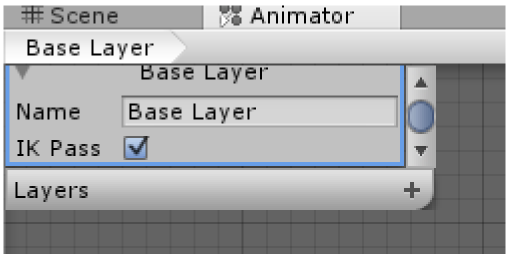
Eyes open, looking straight	Save	Load
Eyes closed, looking straight	Save	Load
Looking up	Save	Load
Looking down	Save	Load

There are a few more controls in the EyeAndHeadAnimator component.

The **Head Transform** is a gameobject that you can assign to control head movement if you are not using a Mecanim humanoid rig . This 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 you are using a Mecanim humanoid setup or Final-IK to control head movement, the Head Transform is ignored.

The **Head Speed Modifier** allows you to increase or decrease the head turn speed.

The **Head weight** slider value determines how much the script controls the character's head movement when looking at targets. For head animation to work, your character needs to have a Mecanim human rig and the Animator Controller must have „IK Pass“ checked:



Name: Base Layer
IK Pass: ✓

Note: In Unity 4, Mecanim head IK only works in Unity Pro.

Final IK: If you have [Final IK]() in your project and want to use that instead of Mecanim's IK to move the head, add FinalIK's LookAtIK component to the character (and optionally a FullBodyBipedIK component if you need it), and in the Scripts/EyeAndHeadAnimator.cs, at the top, change the line

```
//#define USE_FINAL_IK
```

to

```
#define USE_FINAL_IK
```

If you are using LookAtIK, 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.

The checkbox **Are Updates Controlled Externally** gives you more fine-grained control over when exactly the head and eye movements are updated. Normally you don't need that, but in certain special cases like for example you use different FinalIK components like LookAtIK, BoxingIK, and the order of updates is important, check this box and then each frame call the two functions Update1 and Update2 on the EyeAndHeadAnimator component. 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();
lookAtIK.solver.Update();
```

```
eyeAndHeadAnimator.Update2();
```

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.

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

You can control the frequency of 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. You can also change the speed of blinking via **Blink Speed**.

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.

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

The **Max Eye Horiz Angle** value determines how much the eyes can rotate horizontally away from the nose (so for the left eye, the angle to the left relative to the head).

The **Max Eye Horiz Angle Towards Nose** value determines how much the eyes can rotate horizontally towards the nose (so for the left eye, the angle to the right). For some models, it looks better if the angle *Max Eye Horiz Angle Towards Nose* is smaller than *Max Eye Horiz Angle*.

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.

The **Nervousness** value determines how often the character's eyes do micro- and macro saccades. At zero, the eyes are relatively calm. At larger values, the eyes move around more often, giving the character a nervous look.

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.

The **Eye Widen Or Squint** value determines how much the eyes are widened (in suprise) 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.

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 EyeAndHeadAnimator component 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 and the names of their parent transforms (all the names in the parent chain from the object having the EyeAndHeadAnimator component down to the lids 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, Mixamo, MCS (Morph3D) and UMA.

> If your character is a MCS (Morph3D) character, you need to make changes to two REM files:
>
> In the files Scripts/ControlData.cs and Editor/EyeAndHeadAnimatorEditor.cs, at the top, change the line
>
> ```
> //#define USE_MCS
> ```
>
> to
>
> ```
> #define USE_MCS
> ```
>
> If you find your MCS character moves his/her head too far to the side when looking at something at the side, this is an issue with MCS rig. To solve it, in the character's Mecanim humanoid rig import settings, assign the bone neck_upper instead of the by default assigned neck_lower to the neck slot.
>
> If your MCS character has a modified height, you have to move the EyeAndHeadAnimator.cs script to execute after the Default script execution order, otherwise REM will work as if the character was still at unmodified height. To do this, open Edit → Project Settings → Script Execution Order, then drag the EyeAndHeadAnimator.cs script from RealisticEyeAnimations/Scripts below the “Default Time” panel in the window that opens, then click Apply.

If you created your character with either MakeHuman, Autodesk Character Generator, Mixamo or MCS, 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.

> If you are importing a Mixamo preset for a Mixamo character and get an error message, make sure the eyelids mesh object in the character is called „Eyelids“--some Mixamo characters name it differently and then the preset will not work.

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 an character in your code (let's say you saved it in the GameObject variable newCharacterGameObject), use this code (except if you use Android, see below):

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

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

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.dat");
	LookTargetController lookTargetController =
newCharacterGameObject.AddComponent<LookTargetController>();
	lookTargetController.Initialize();
#else
	StartCoroutine(LoadPreset("mypreset.dat"));
#endif
```

And in the same class, add this code:

```
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();
}
```

If you are using WebGL, use something like this:

```
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.ImportFromBytes(www.downloadHandler.data);

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

	lookTargetController.Initialize();

}
```

Using UMA (Unity Multipurpose Avatar)

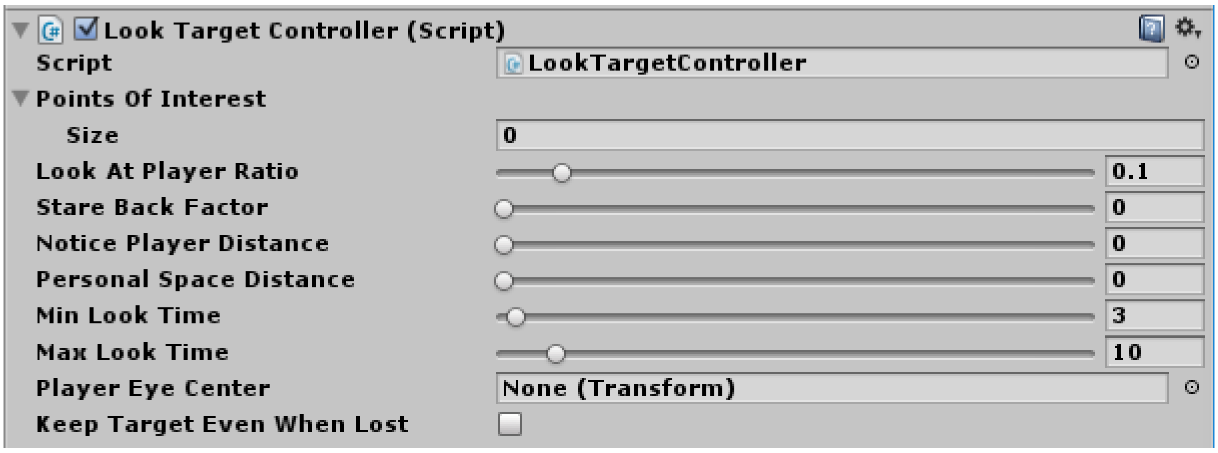
If you use UMA, you need to add the components at runtime because the characters are generated at runtime. The previous section described how to do this. Just use the existing preset UMA.dat from the preset folder. In the case of UMA, you might need to wait a frame after the character is generated for 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.dat");


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

	lookTargetController.Initialize();

}
```

Look Targets



Look Target Controller (Script)	
Script	LookTargetController
Points Of Interest	
Size	0
Look At Player Ratio	0.1
Stare Back Factor	0
Notice Player Distance	0
Personal Space Distance	0
Min Look Time	3
Max Look Time	10
Player Eye Center	None (Transform)
Keep Target Even When Lost	☐

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 characte 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 characters sees the player keep staring at him or her (so how sensitive the character is to being stared at).

The slider **Notice Player Distance** determines what distance the player has to come nearer than for the character to start looking at him. 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 that 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**.

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 **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).

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.

Virtual Reality Headsets

By default, the script uses the camera tagged MainCamera to determine the player's position. If you use Unity's native VR support or an Oculus camera rig prefab, 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

void **Blink**(bool isShortBlink)

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 **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)

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 tamulur@yahoo.com.

Good luck with your projects!

Tore Knabe