Allright Rig 2.0

Allright Rig is a plug-in for Unreal Engine 4 that allows you to create character rigs and work with character animation in editor. The main goal is to create a possibility of creation and editing character animation, which will help to make games, previsualization, cinematics and animation movies in Unreal Engine 4.

Key features:

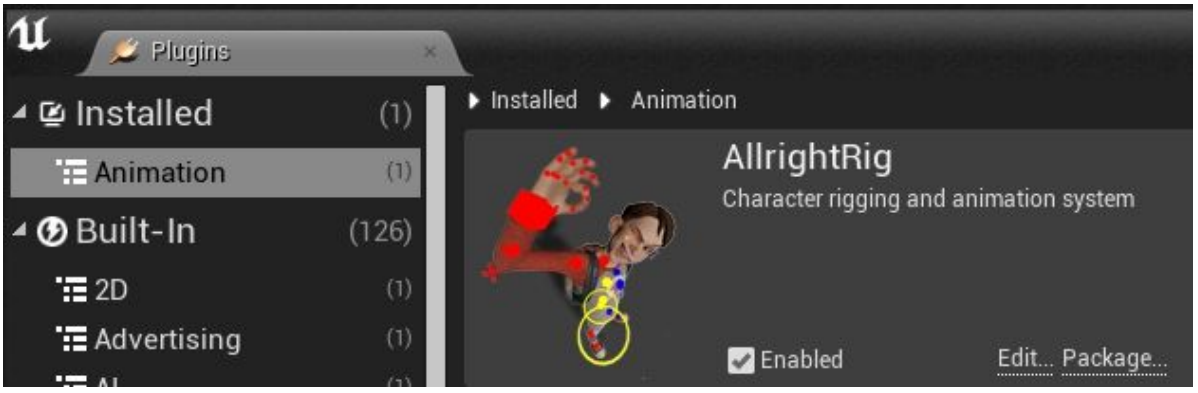
1) Modular rigging system allows you to create any kind of rigs.
2) Facial rigging using bones and blend shapes.
3) Baking animation from an animation sequence to the rig and vise versa.
4) Partial animation editing.
5) Copy/paste, mirror animation using animation tools.
6) Animation constraints such as parent, point, orient, scale and aim.
7) Filters for mocap cleaning.
8) Custom blueprint library with around 70 blueprint nodes and macro library with around 120 blueprint functions.
9) Presets for different character rigs.

Installation Instructions

1) Copy **AllrightRig_2.0_UE_4.1X\Plugins\AllrightRig** folder under your **...\ProjectName\Plugins** folder or also under **...\Epic Games\4.1X\Engine\Plugins\Runtime**

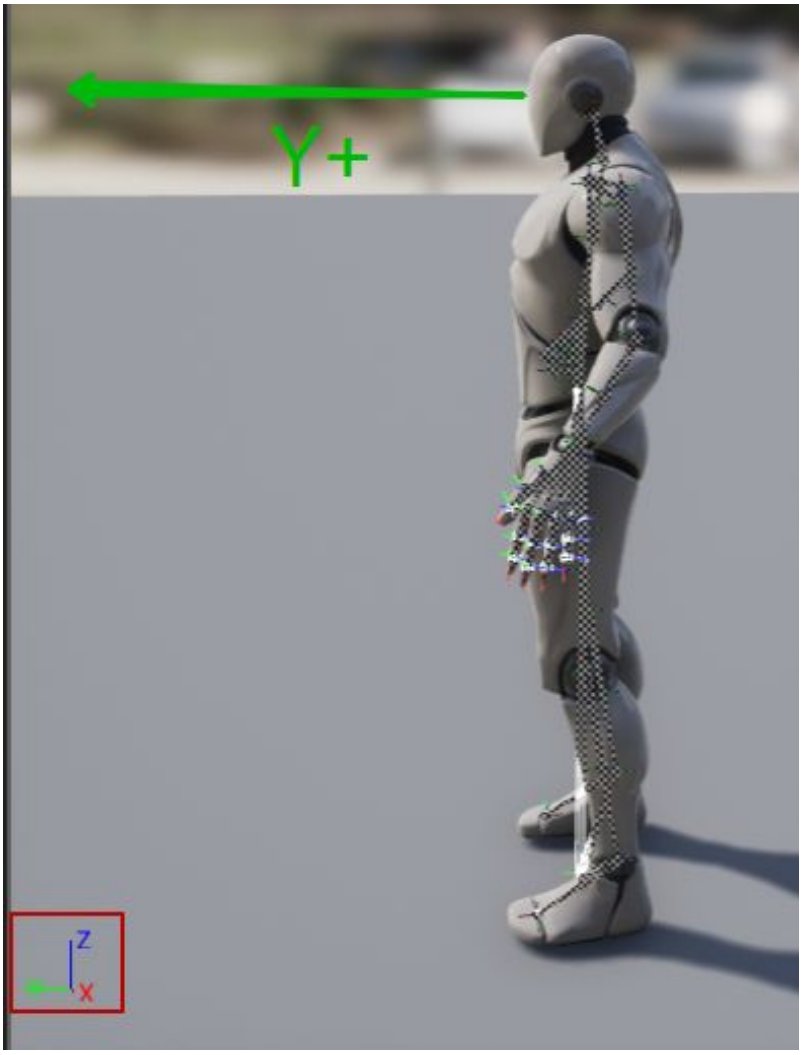
2) Copy all files from **AllrightRig_2.0_UE_4.1X\Saved\SaveGames** folder under **...\ProjectName\Saved\SaveGames** folder to be able to load them in AllrightRig window under **"Presets"** category.

3) Plugin should be enabled by default, however you can find it in **Plugins** window under **Animation** category.

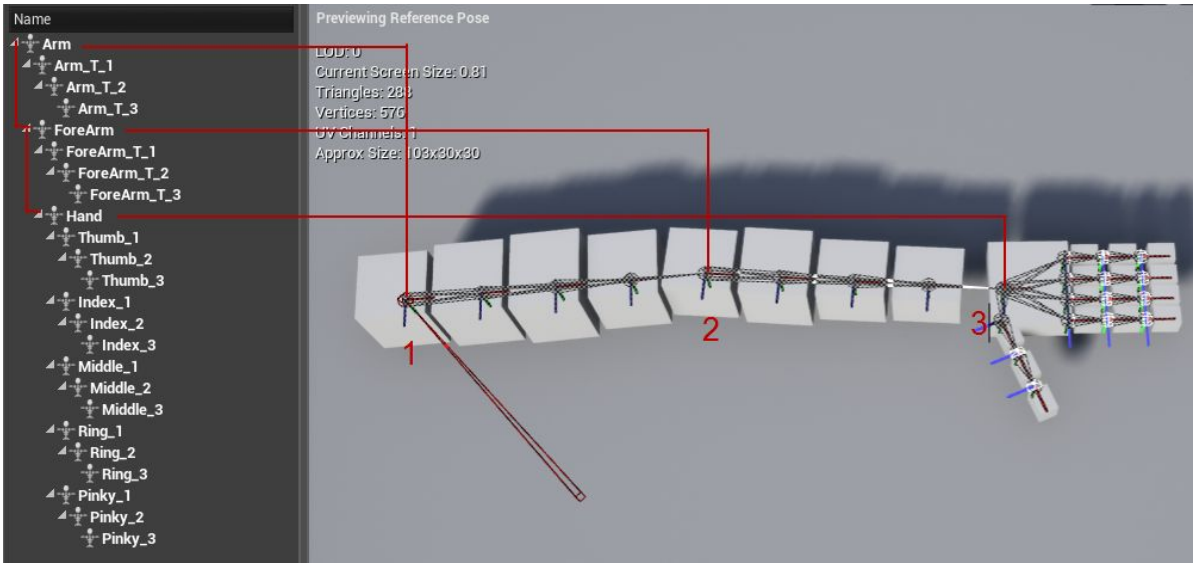


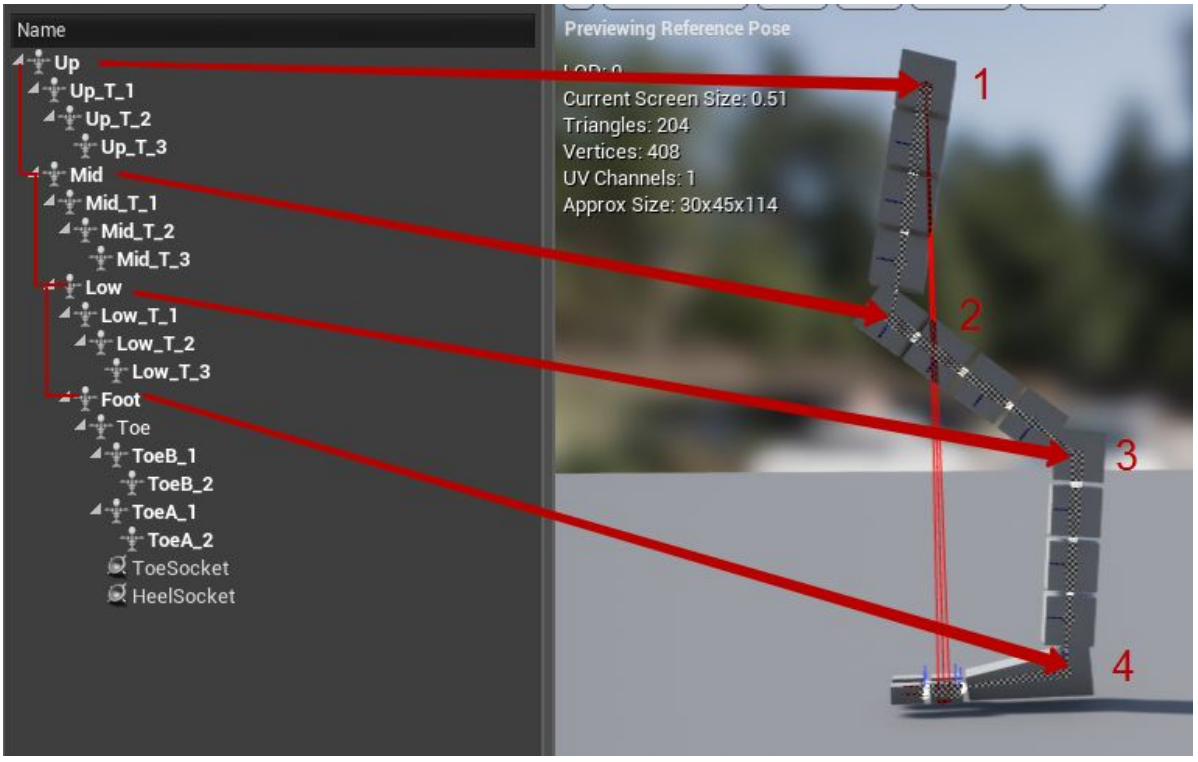
Requirements

1) Character needs to be **faced** to the **Y+** axis in the **reference pose**.



2) In order to use Ik solvers, character limbs hierarchy should consist of **3 (for bipeds) or 4 (for quadrupeds) major bones**.

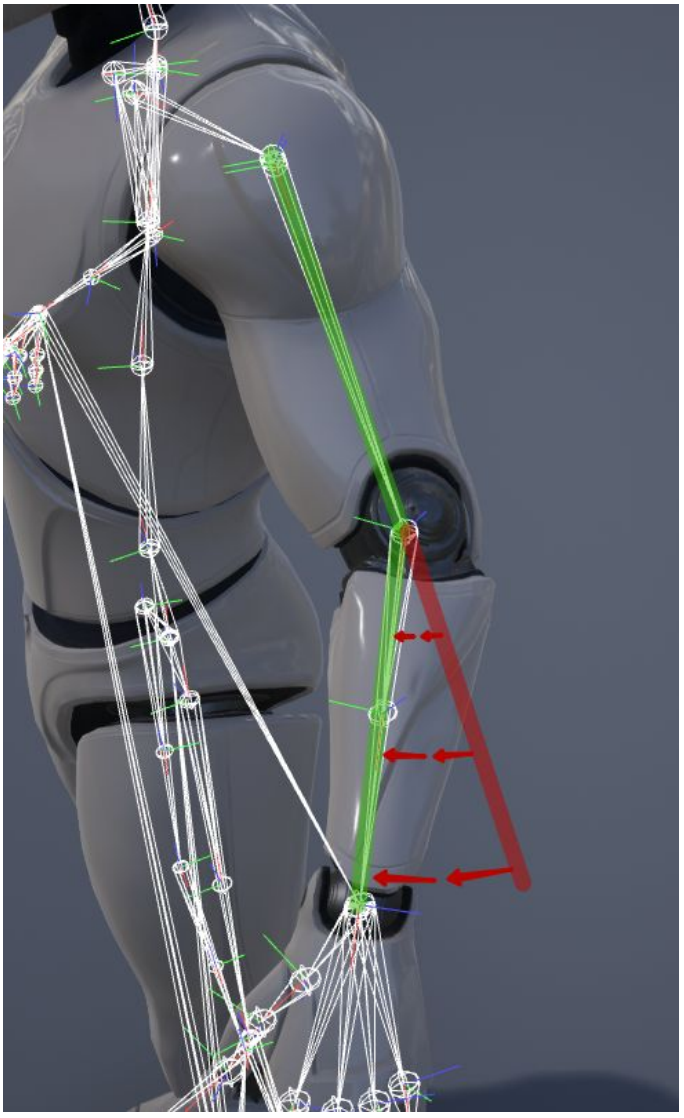




You can attach/parent extra bones to the major bones but **hierarchy order** of the **major bones** should be the same.

For reference you can use **Arm** and **ThreeBoneLeg** assets under **AllrightRigContent/Examples/Characters/** folder.

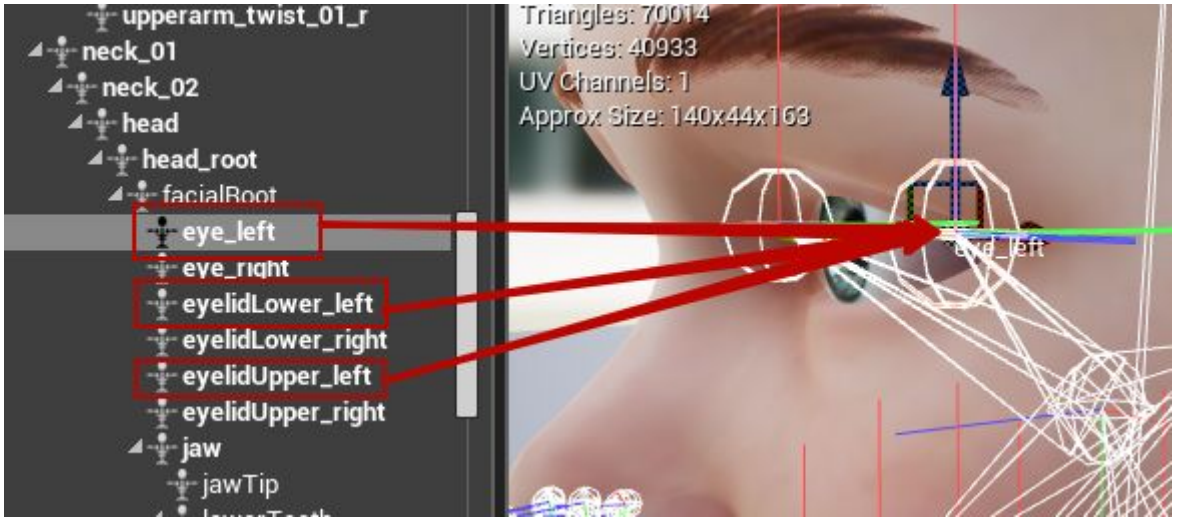
3) If your characters **limbs** are absolutely **straight** in the reference pose, it is impossible to calculate right location for **pole vector** controller during **creation of rig**.



However, in case when limbs are straight, you can use **Pole Vector Simple** mode to find pole vector location.

4) Controllers orientation is defined by orientation of bones. System supports **any bones orientation**. However it is more comfortable to work with rig using at least some basic orientation where most of the bones are oriented to the next bone in hierarchy.

5) **Automatic eyes rig** requires **3 bones** for each eye. For reference you can use **boy_SKELETON** asset from **Allright Rig** plugin **content** under **Examples/Characters/KiteDemo** folder.

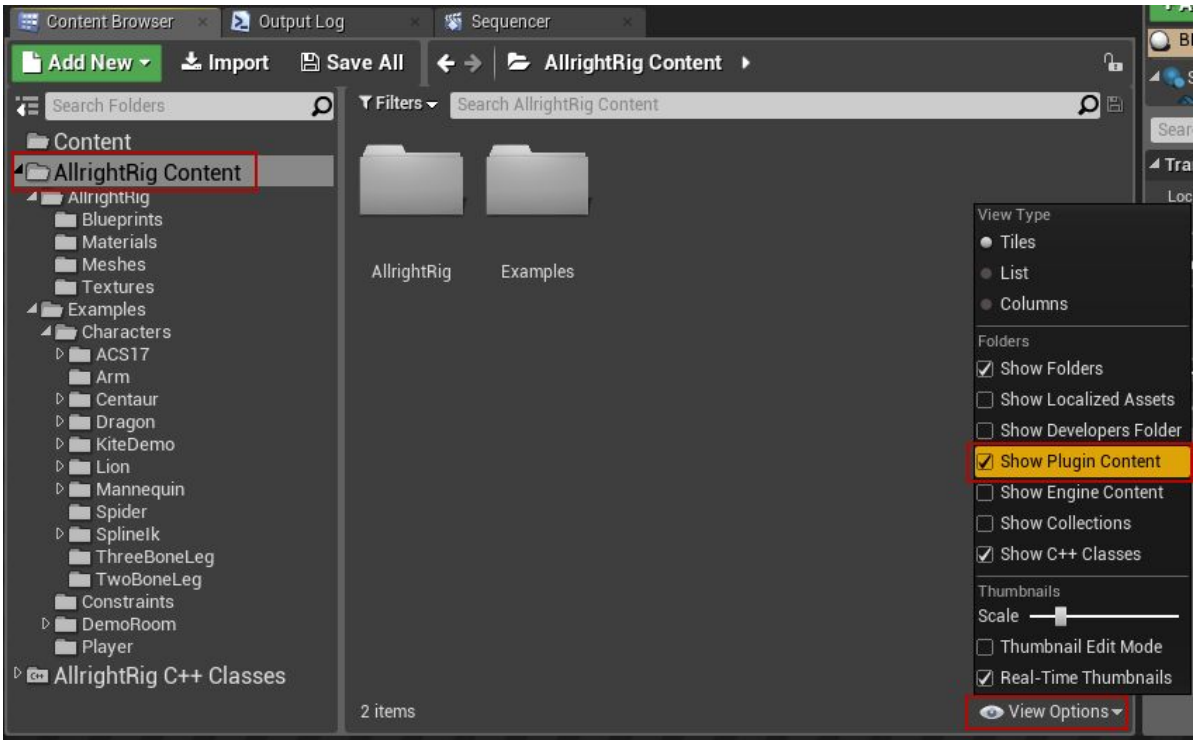


eye_left - influences on left eyeball
eyelidLower_left - influences on left lower eyelid
eyelidUpper_left - influences on left upper eyelid

It is better to use world orientation for this bones.

Allright Rig Content

Enable Show Plugin Content checkbox in **Content Browser View Options**.



Under **AllrightRig Content** folder you can find **AllrightRig** and **Examples** folders.

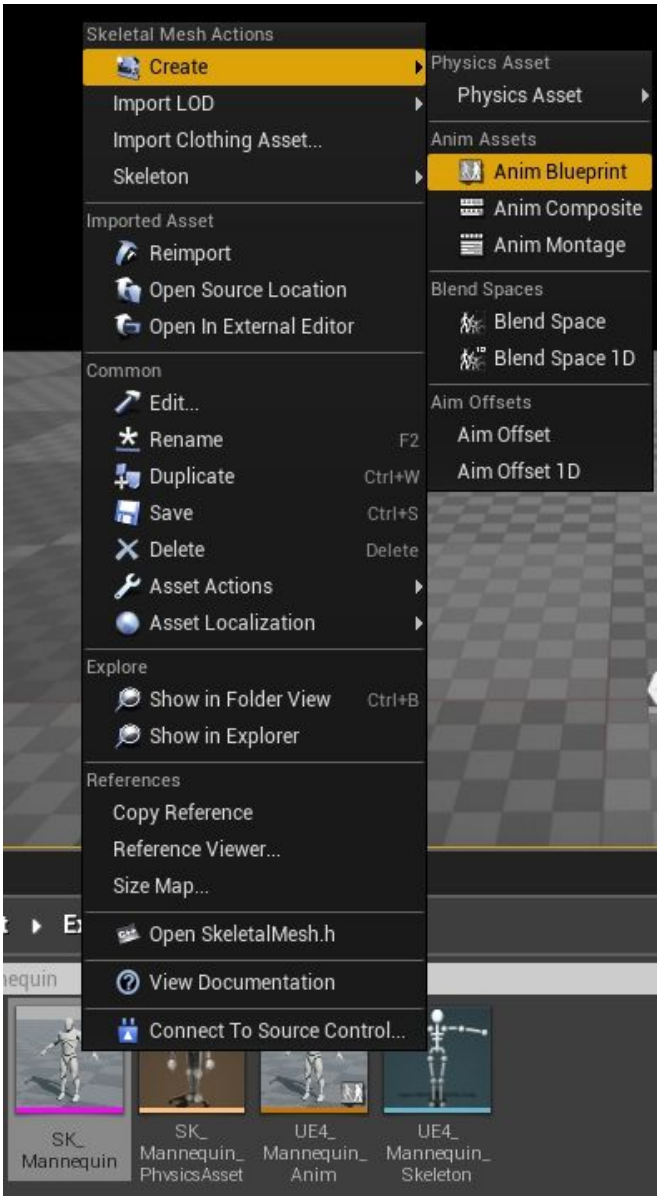
1) **AllrightRig** folder includes all **necessary** assets for plugin such as blueprints, meshes, materials, macros library etc.
2) **Examples** folder includes content for example map that shows different character rigs and main system features.

Creating rigs

1) Create Anim Blueprint

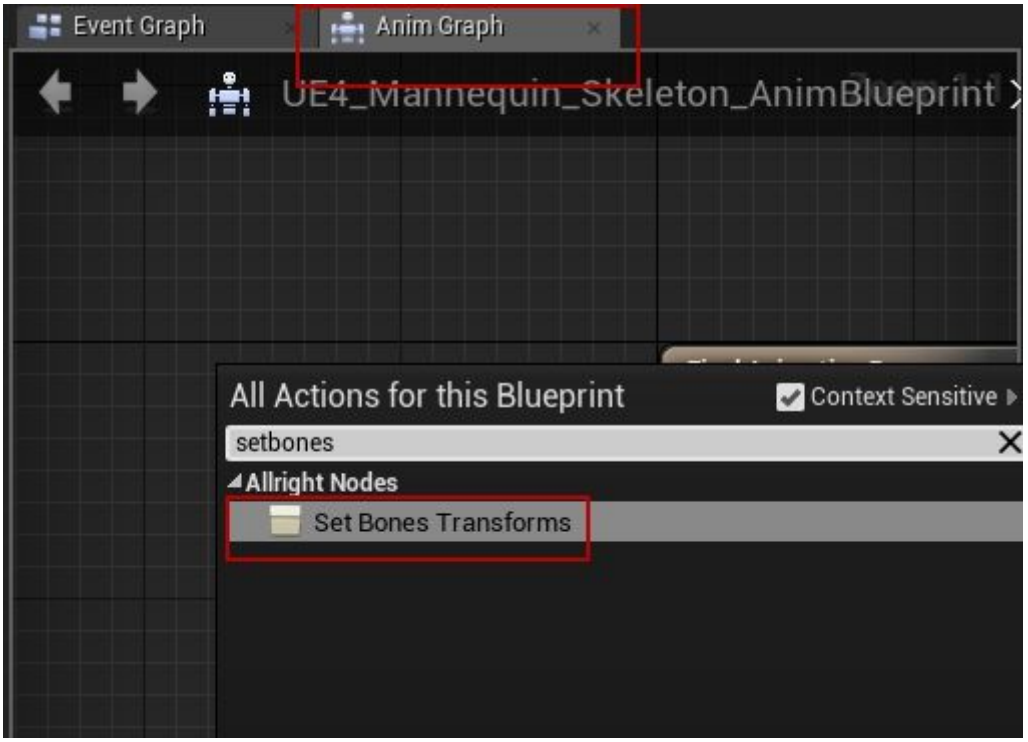
Before creating the rig you need to make an **anim blueprint** for your character.

1.1) Click r**ight mouse button** on your **skeletal mesh asset**, go to **Create** menu and choose **Anim Blueprint**.

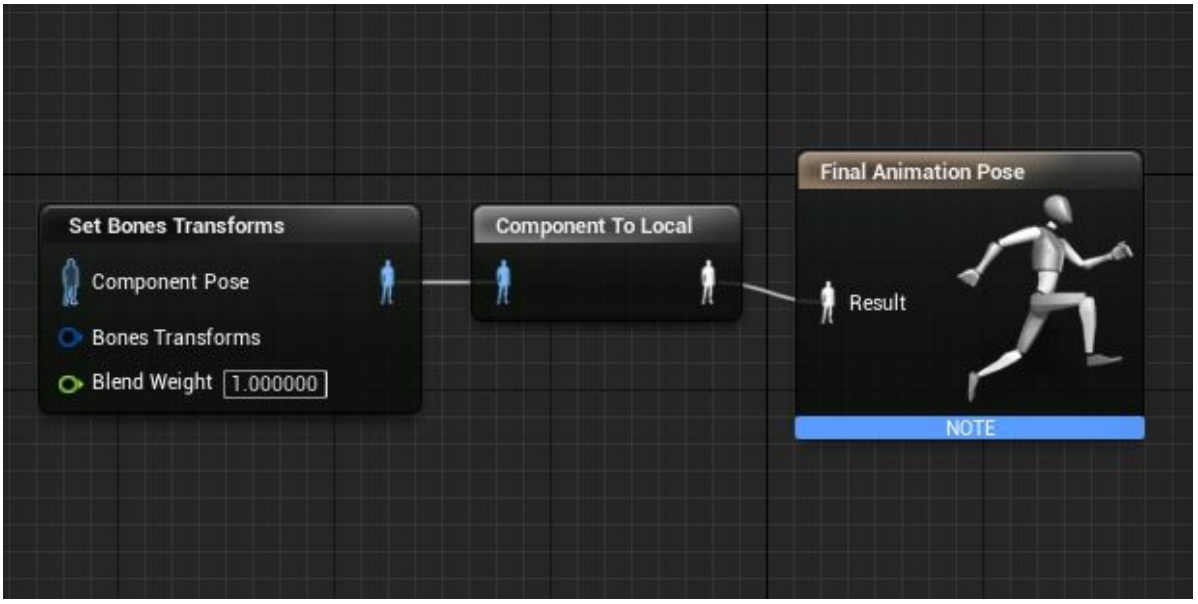


*It is faster to **copy blueprint nodes** from any **anim blueprint** under **Examples** folder in plugin's content, however it is also not hard to setup anim blueprint from scratch. Hope in future I will automate this procedure.*

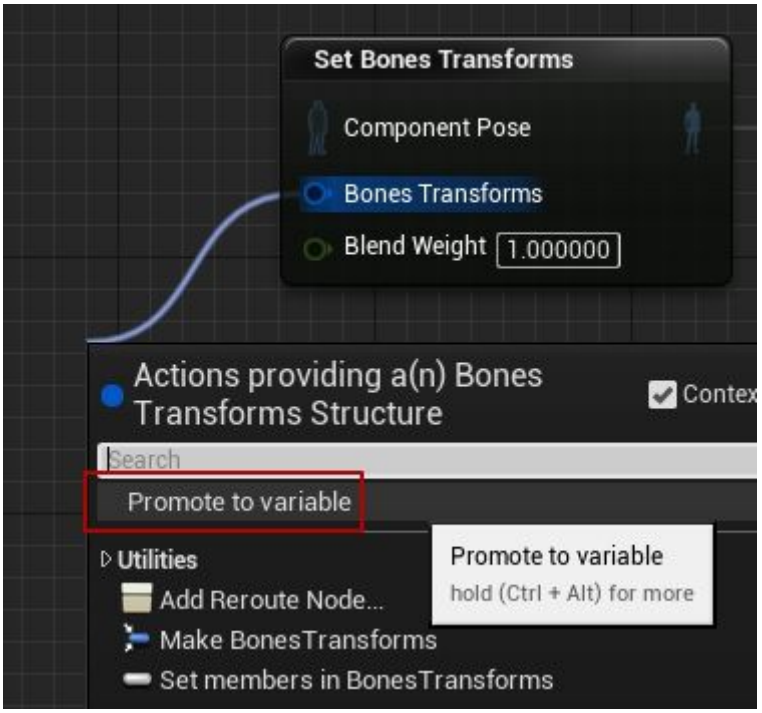
1.2) Open new animation blueprint. Go to **Anim Graph** and create **Set Bones Transforms** node.



1.3) Connect **Set Bones Transforms** to **Final Animation Pose**

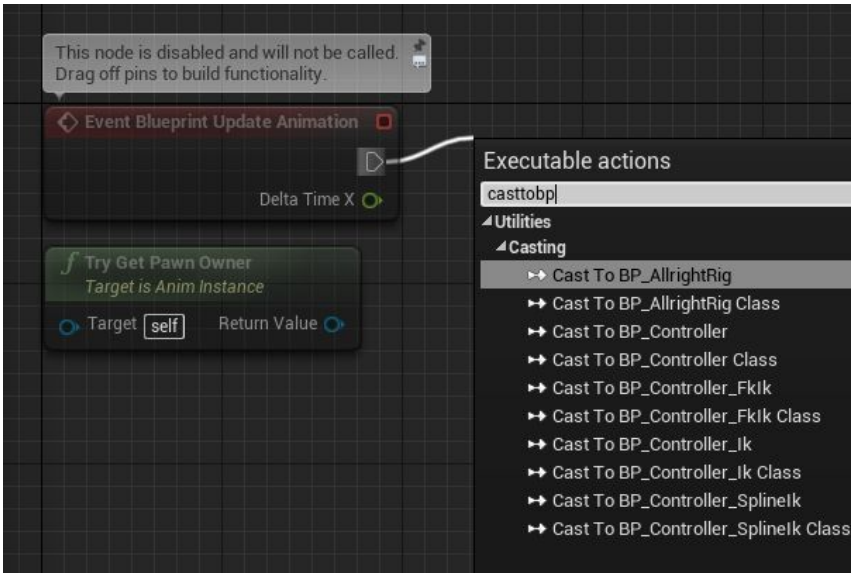


1.4) **Drag** left mouse button from **Bones Transforms** pin and choose **Promote to variable**



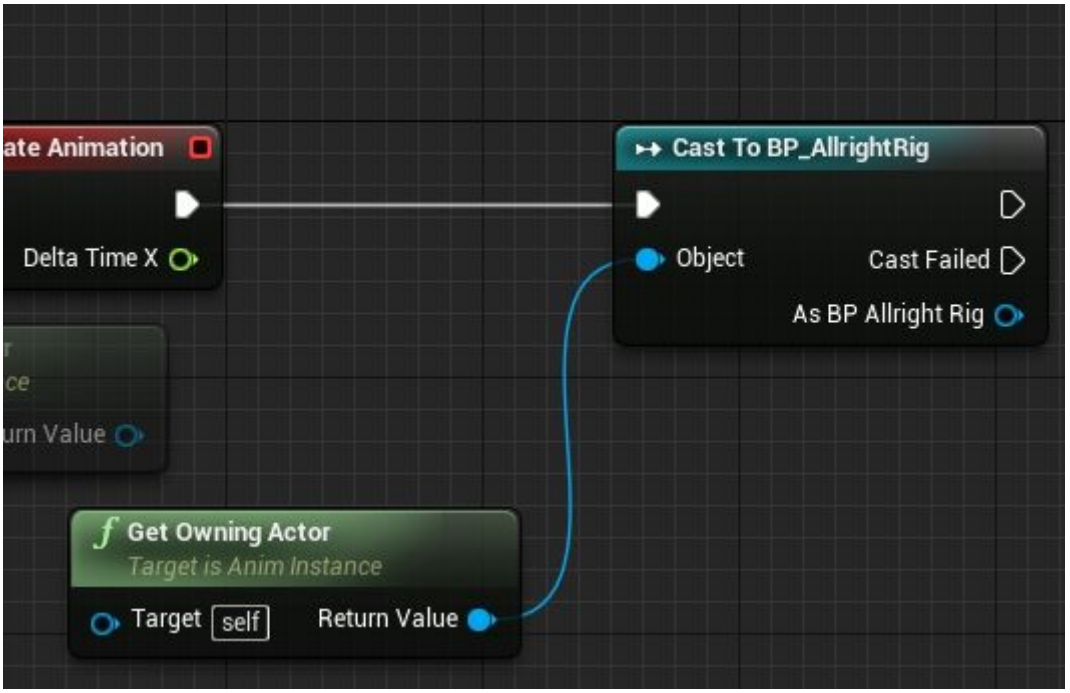
Name of the variable does not matter. Lets leave it **NewVar_0** for this tutorial.

1.5) Go to **Event Graph**, drag execute pin from **Event Blueprint Update Animation** node and type **casttobp**. Choose **Cast To BP_AllrightRig** node.

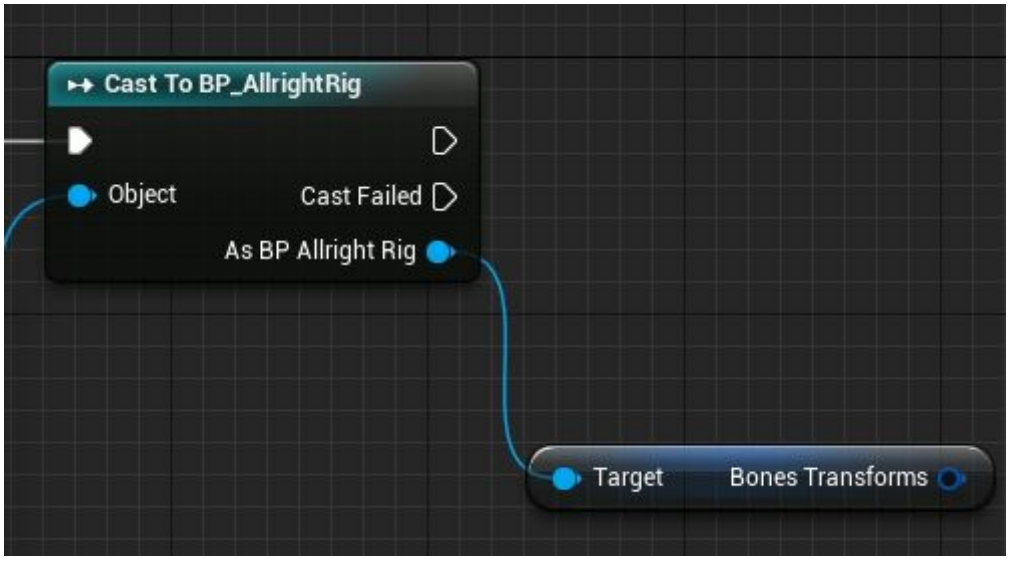


You can cast only to **BP_AllrightRig** in order to get **"BonesTransforms"**!

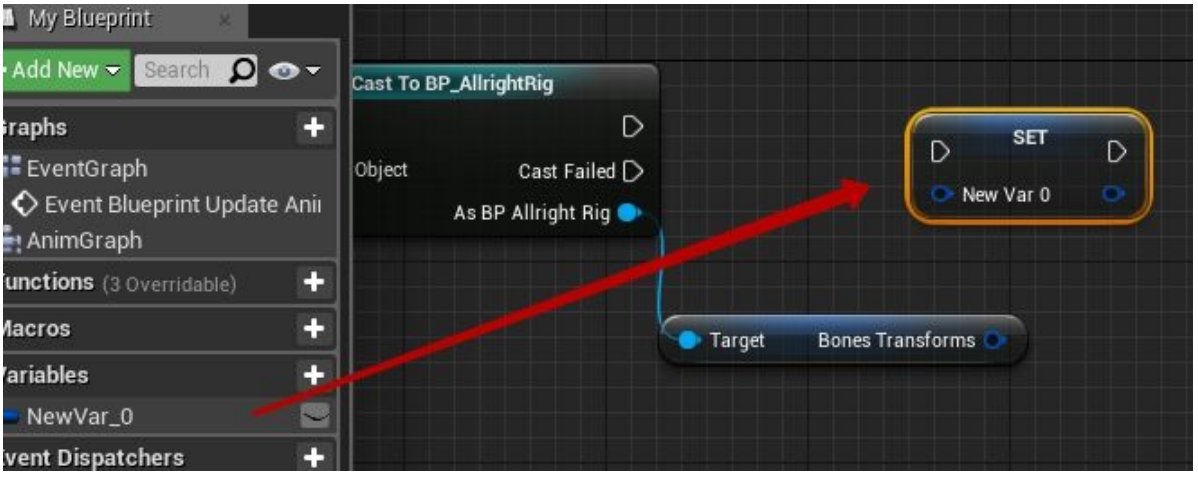
1.6) Create **Get Owning Actor** node and connect it to **Object** pin of **Cast To BP_AllrightRig** node.



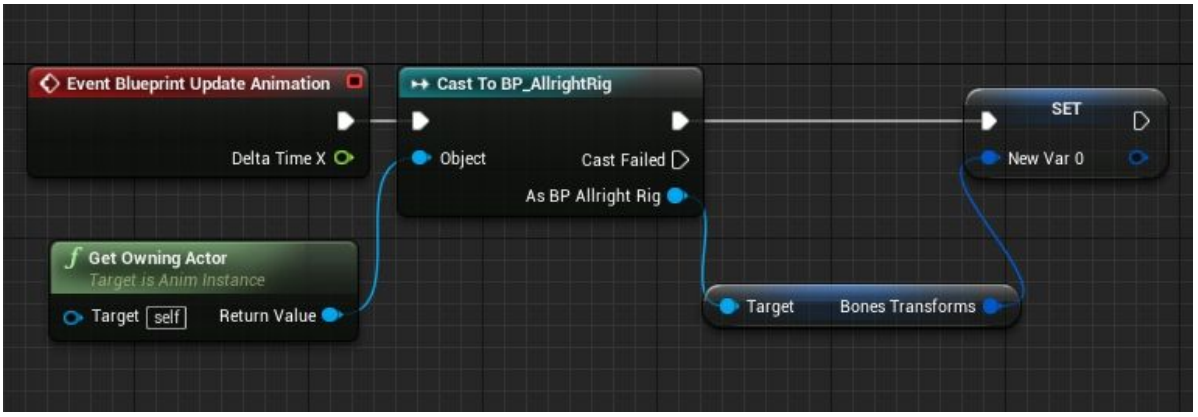
1.7) Now drag left mouse button from **As BP Allright Rig** and chose **Get Bones Transforms** node.



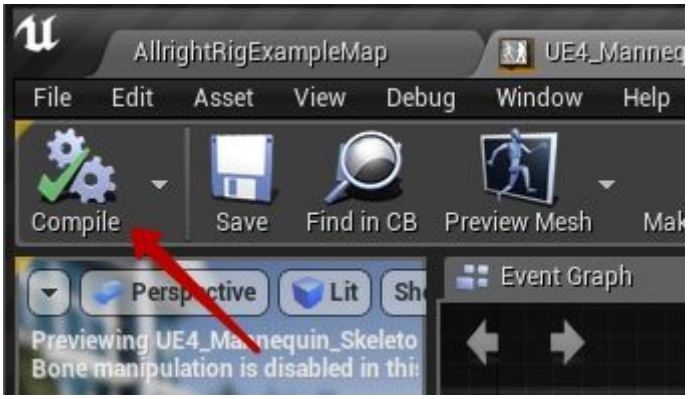
1.8) Drag earlear created variable and choose **Set.**



1.9) Connect execute pin from **Cast To BP_AllrightRig** node and output pin from **Get Bones Transforms** node to this variable.



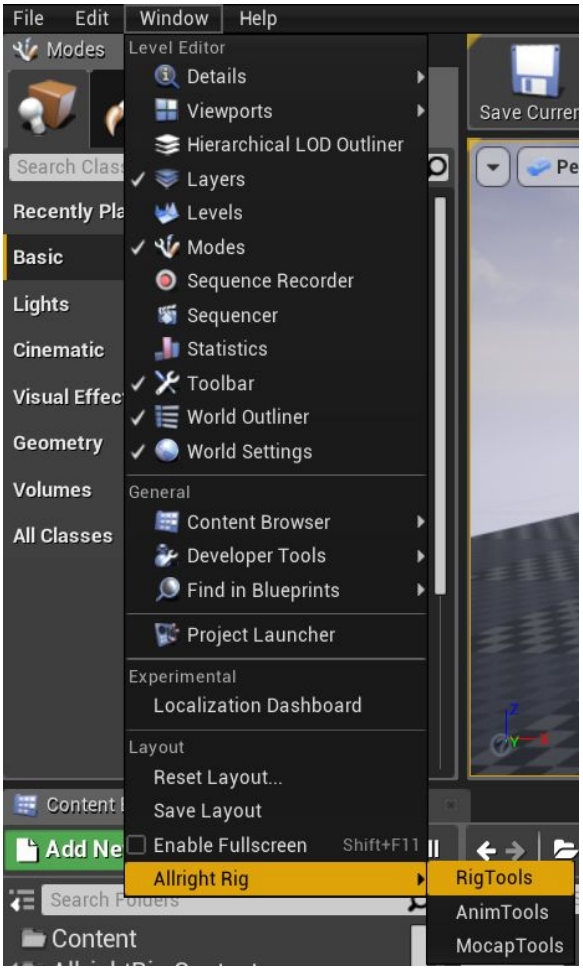
1.10) **Press Compile.**



You can still use other nodes to adjust this anim blueprint.

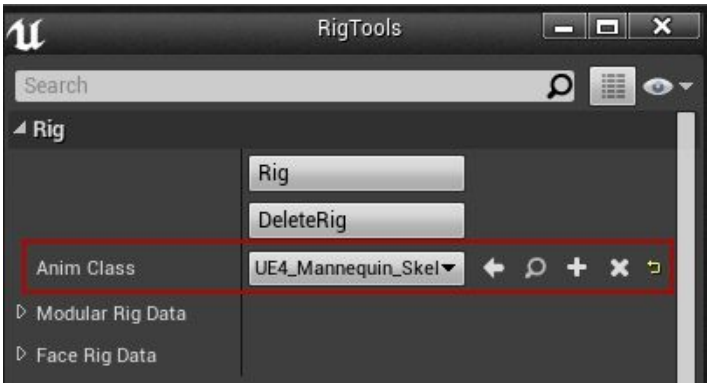
2) Get Ready To Rig

2.1) Open **Rig Tools** window from **Allright Rig** menu under the main **Window** menu.

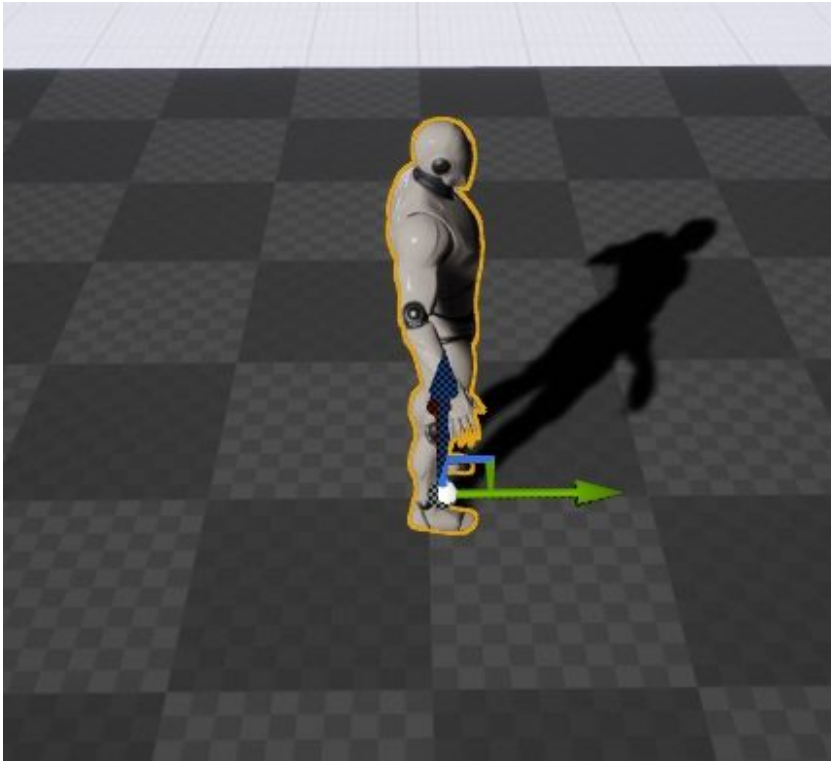


*You can open **many** instances of this window in order to work with **different** characters at the same time.*

2.2) Choose **Anim Blueprint Class** for your character.

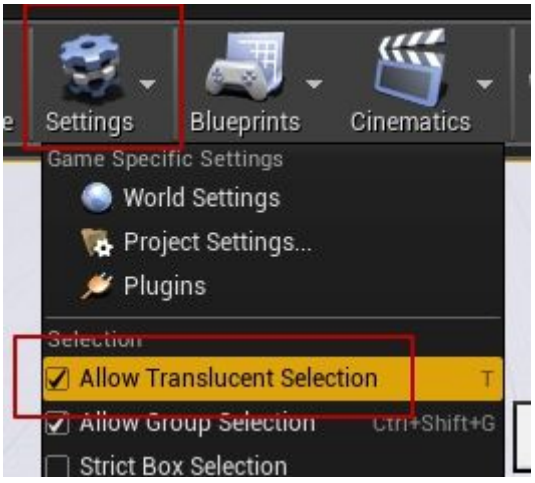


2.3) Drag your skeletal mesh to the scene.



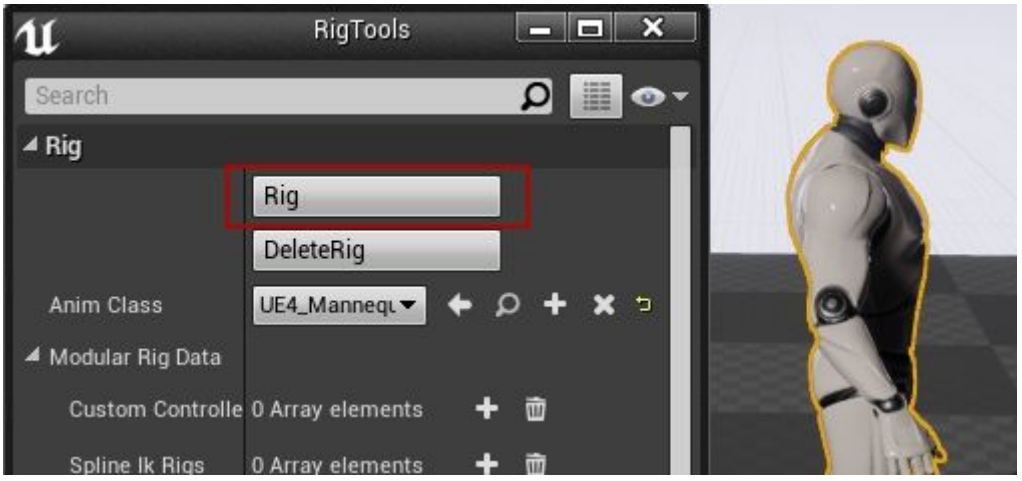
*For **SK_Mannequin** you can use **ManRig** preset for reference.*

2.4) Make sure that **Allow Translucent Selection** is checked under **Scene Settings**.

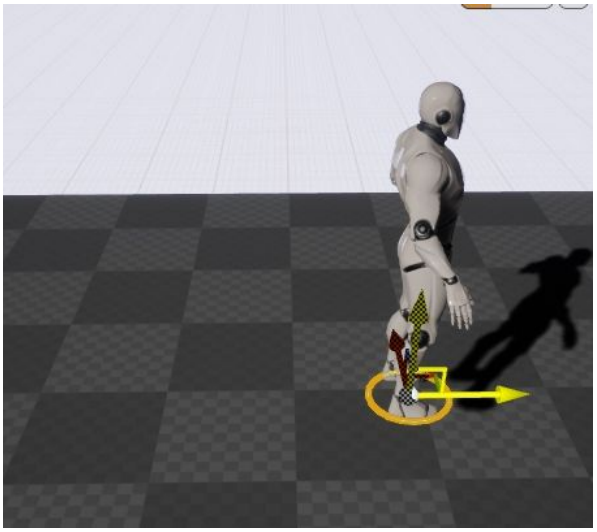


All rig controllers have translucent materials. Translucent material allows to see controllers on the top of the character mesh.

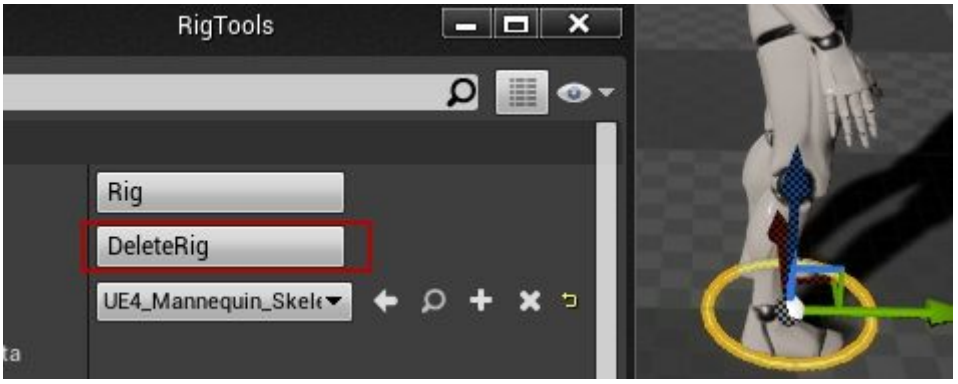
2.5) **Select** character and press **Rig** button.



If everything goes well you will see a yellow circle under the character. This circle is a **Main** controller. All other controllers will be attached to it. Try to **move** this controller to make sure that it **drives the character**.



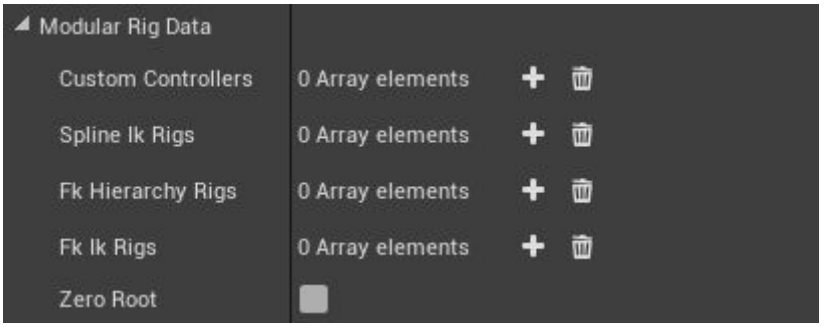
2.6) **Select Main Controller** and press **DeleteRig** button.



In this section I will use default UE4 Mannequin character. Please use corresponding bone and controller names for your character instead.

3) Set Modular Rig Data

Modular Rig Data is a custom structure that handles all modules data.



Custom Controllers is array of CustomControllersData structures that allows you to create controllers that will transform only one bone per controller.
Spline Ik Rigs is array of SplineIkData structures that allows you to create spline Ik rig for bone chains.
Fk Hierarchy Rigs is array of FkHierarchyData structures that allows to drive bone chain using hierarchy of controllers.
Fk Ik Rigs is array of FkIkData structures that allows to drive 3 or 4 bone limbs using TwoBoneIk or ThreeBoneIk solvers and Fk Hierarchy Rig with Fk/Ik blending and Fk/Ik snapping. In addition system has settings and controllers for fingers and toe.
Zero Root if this property is checked, skeletal mesh actor transform will be always in origin. It is important for baking animation.

*You can add **any amount** of this rig modules to **any parts** of your character.*

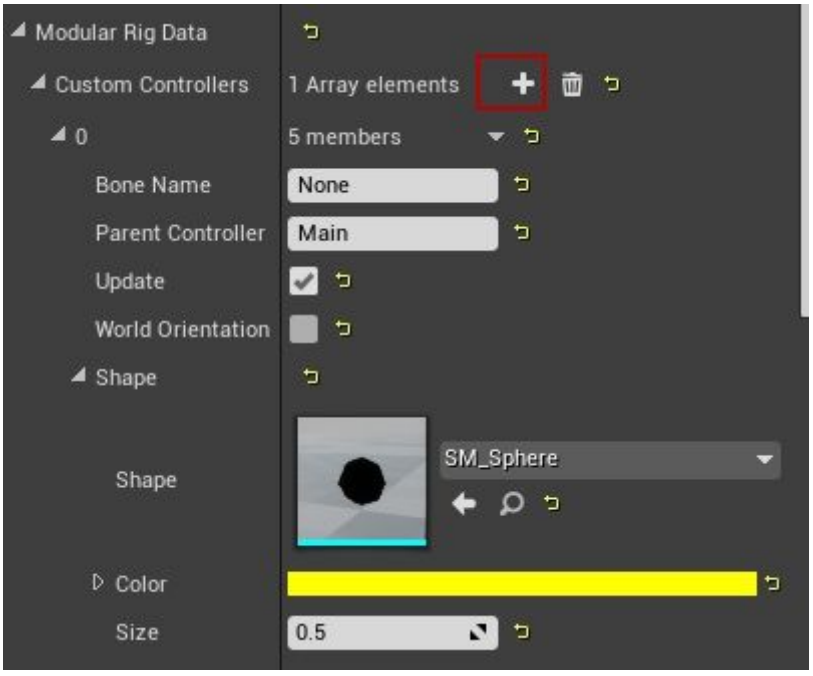
***At first** you may find process of creating modular rig **pretty bulky** but you need to make it **only once** for your character and save it to a **preset**.*

Most of the properties are not case sensitive, however it is better to preserve uppercase or lowercase for all properties.

3.1) Set Custom Controllers

Custom Controllers is array of **Custom Controllers Data** structures that allows you to set properties for controllers that will transform only **one bone per controller**. By default controller orientations will be the same as corresponding bones or you can choose world orientation.

3.1.1) Add new element to **Custom Controllers** array.

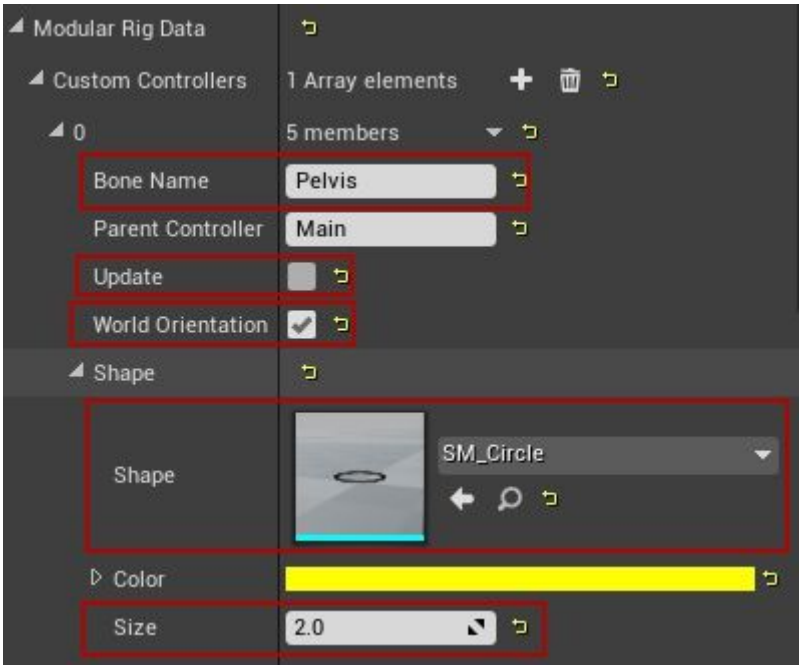


Bone Name is the name of the bone that this controller will drive. Controller will appear on this bone position.
Parent Controller is the name of the controller that this controller will be attached to.
Update property defines if controller should update bone transform.
World Orientation defines if controller will be oriented to world. If this property unchecked controller orientation will be the same as bone orientation.
Shape is a custom structure that defines appearance of controller.

> **Shape** is reference to static mesh. By default it is set to SM_Sphere static mesh from plugin content folder. Defines shape of controller.
> **Color** - controller color.
> **Size** - size of controller.

3.1.2) We do not need this controller to update pelvis bone. We need this controller to attach spine controllers to it.

Set **Custom Controllers Element 0** properties:

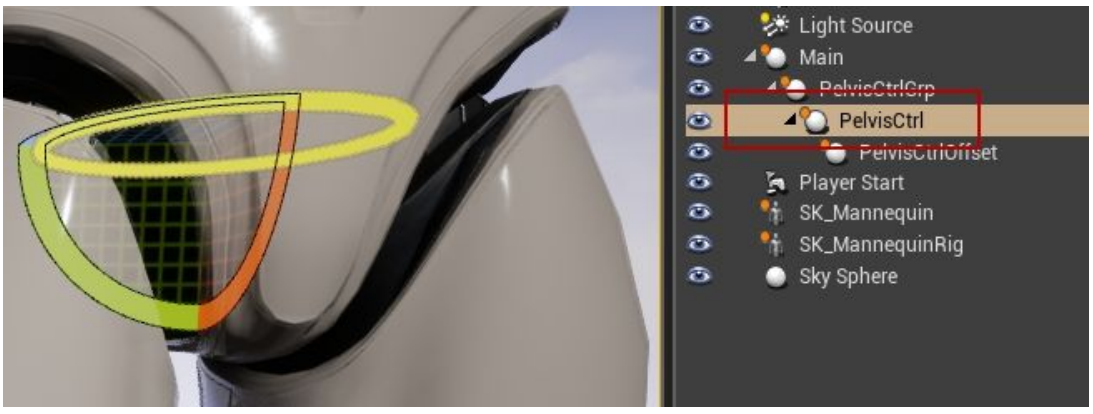


Bone Name = Pelvis
Update = disabled
World Orientation = enabled
Shape = SM_Circle
Size = 2

3.1.3) **Select** character and press **Rig** button.

Now you should see that controller appeared in pelvis location.

3.1.4) **Select pelvis controller** and copy its **name**. It should be **PelvisCtrl**.



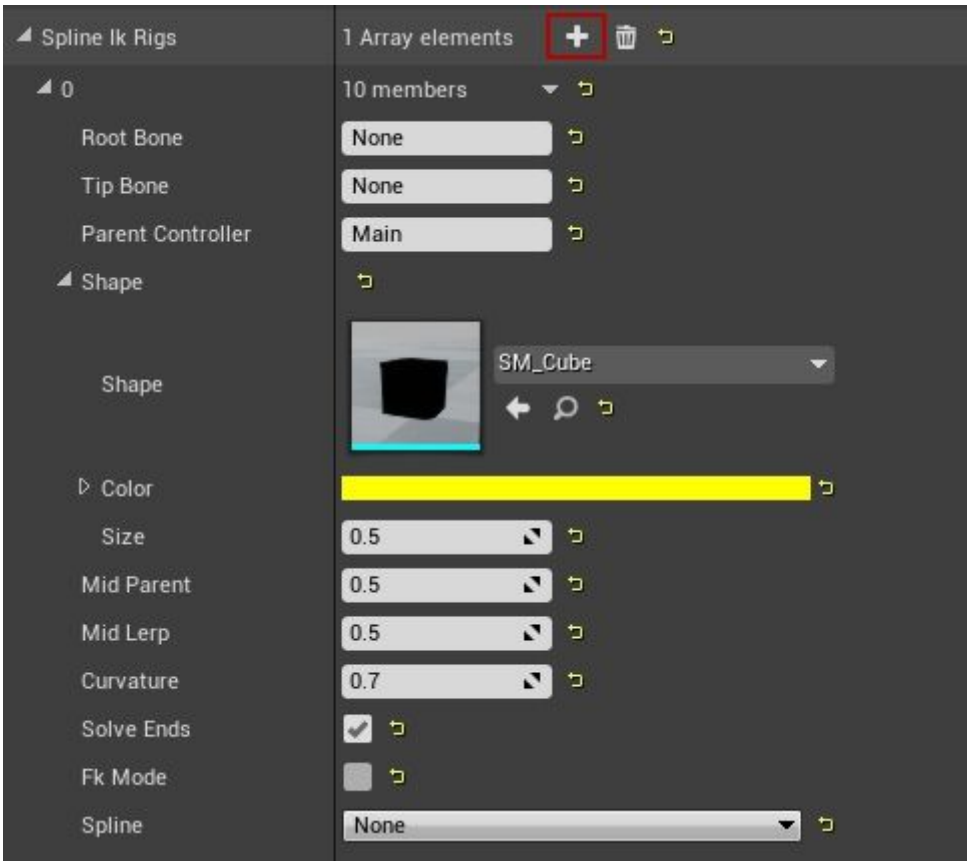
*You can recreate rig at any time to look for controller name for **Parent Controller** property.*

3.1.5) **Select any Controller** and press **DeleteRig** button.

3.2) Set Spline Ik Rigs

Spline Ik Rigs is array of **Spline Ik Rig Data** structures that allows you to create spline Ik rig for bone chains. **Root** and **tip** controllers orientation will be the same as corresponding bones, **middle** controller will be oriented to look at tip controller by default.

3.2.1) Add new element to **Spline Ik Rigs** array.



Root Bone is the name of the first bone in chain.
Tip Bone is the name of the last bone in chain.
Parent Controller is the name of the controllers that this controllers will be attached to.
Shape defines appearance of controller.
Mid Parent - amount of influence of tip and root controllers on middle controller position.
MidLerp - proportion of influence of tip and root controllers on middle controller position.
Curvature - curvature of Ik curve
Solve Ends - solve tangents on curve ends
Fk Mode - switch spline Ik to Fk mode. Controllers will behave like simple Fk hierarchy.

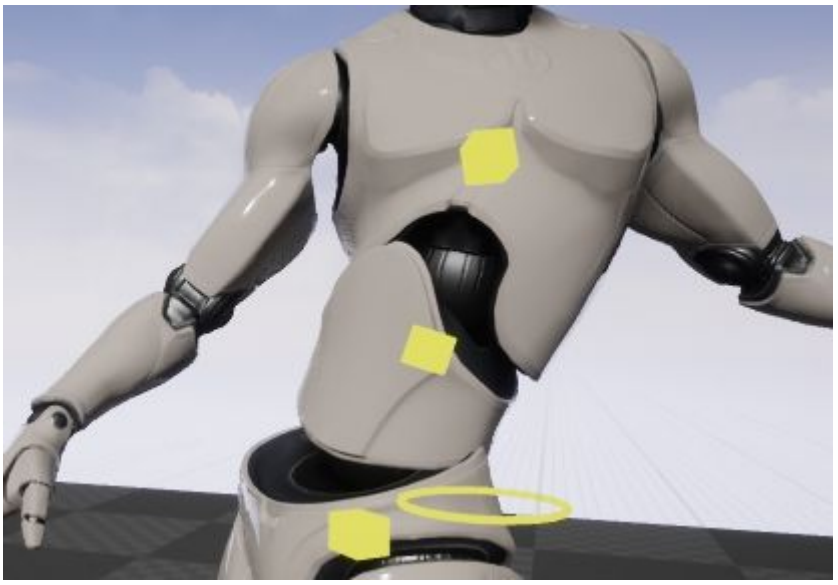
Spline - spline component reference. This property filled automatically, during creation of the rig.

3.2.2) **Set Spline Ik Rigs Element 0** properties:

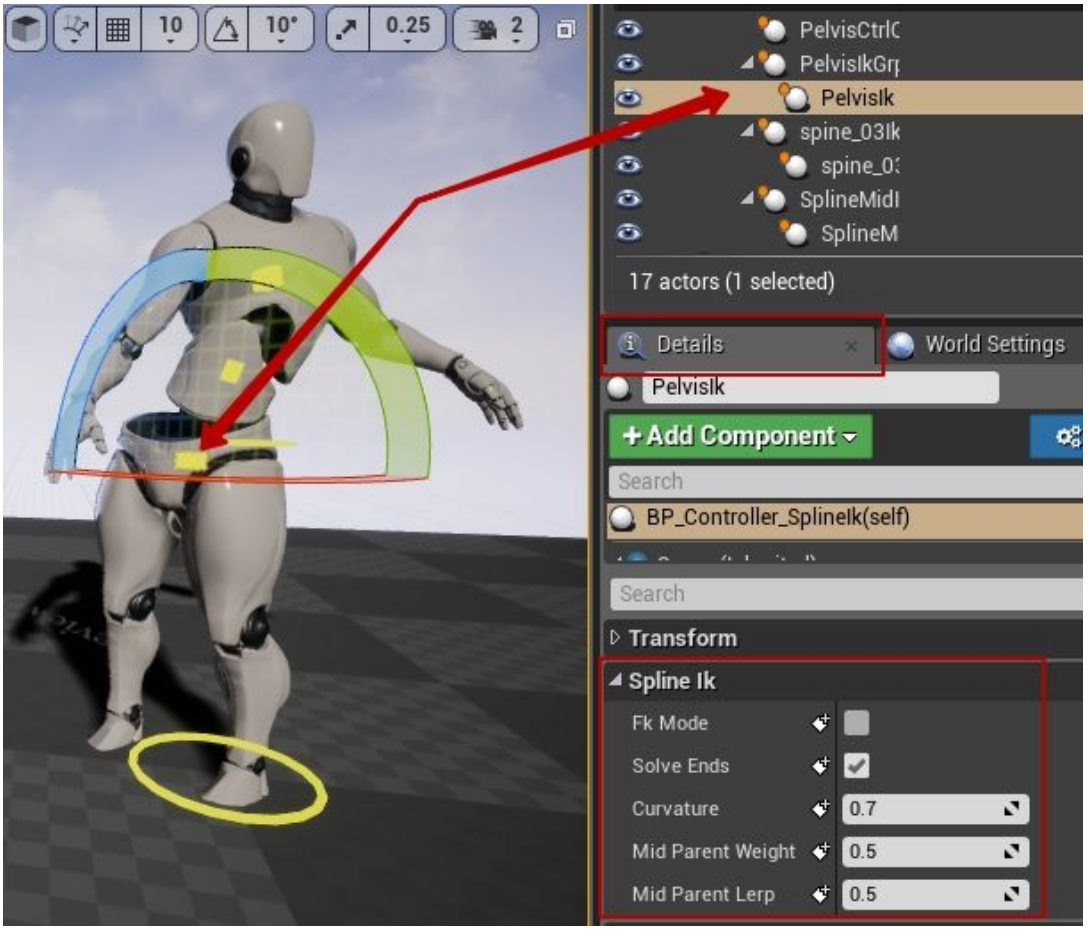
Root Bone = Pelvis
Tip Bone = spine_03
Parent Controller = PelvisCtrl

3.2.3) **Select** character and press **Rig** button.

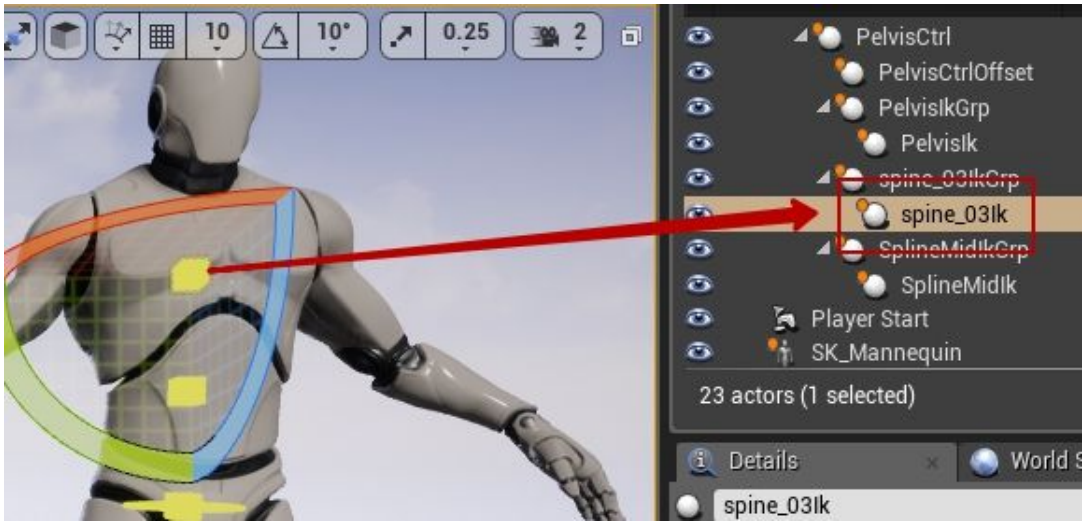
You should see 3 controllers on character spine. Try to move, rotate and scale them.



3.2.4) Select **PelvisIk** and go to the **Details** panel. Under **Spline Ik** category you can find **Spline Ik solver settings** described in paragraph 3.2.1.



3.2.5) **Select upper spline Ik controller** and copy its name. It should be **spine_03Ik**.

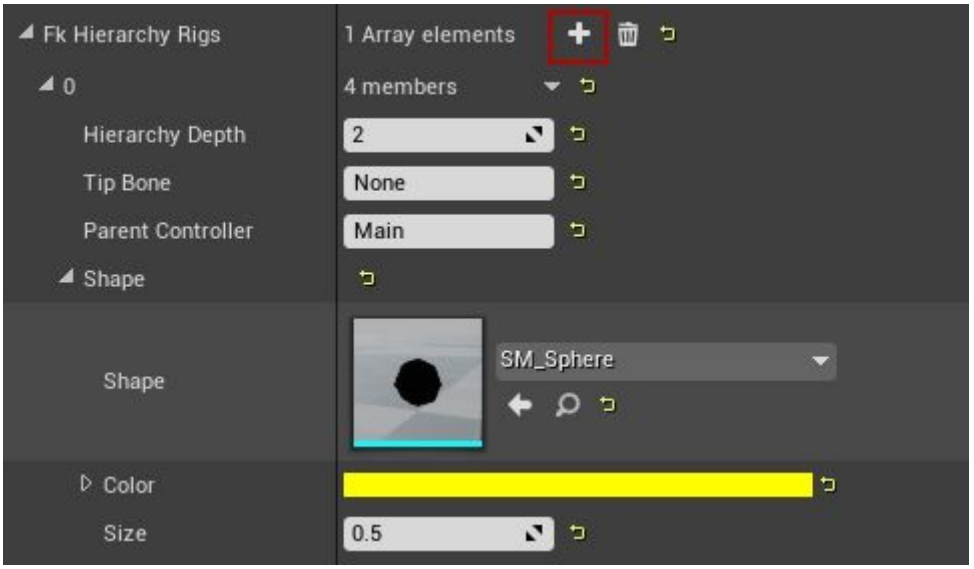


3.2.6) **Select any Controller** and press **DeleteRig** button.

3.3) Set Fk Hierarchy Rigs

Fk Hierarchy Rigs is array of FkHierarchyData structures that allows to drive bone chain using hierarchy of controllers. Controllers orientation will be the same as corresponding bones.

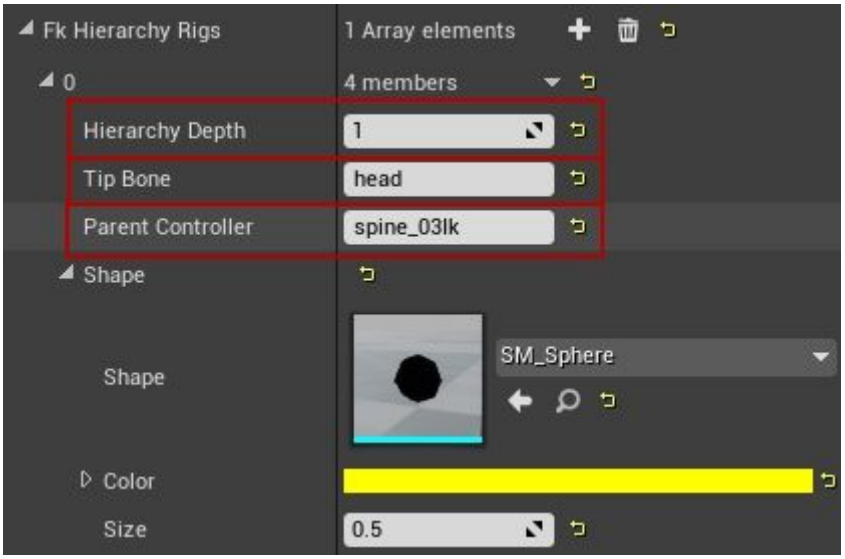
3.3.1) Add new element to **Fk Hierarchy Rigs** array.



Hierarchy Depth - number of bones to rig in chain starting from **Tip Bone** as **0**.
Tip Bone - name of tip bone in chain.
Parent Controller - name of the controllers that this controllers will be attached to.
Shape - appearance of controller.

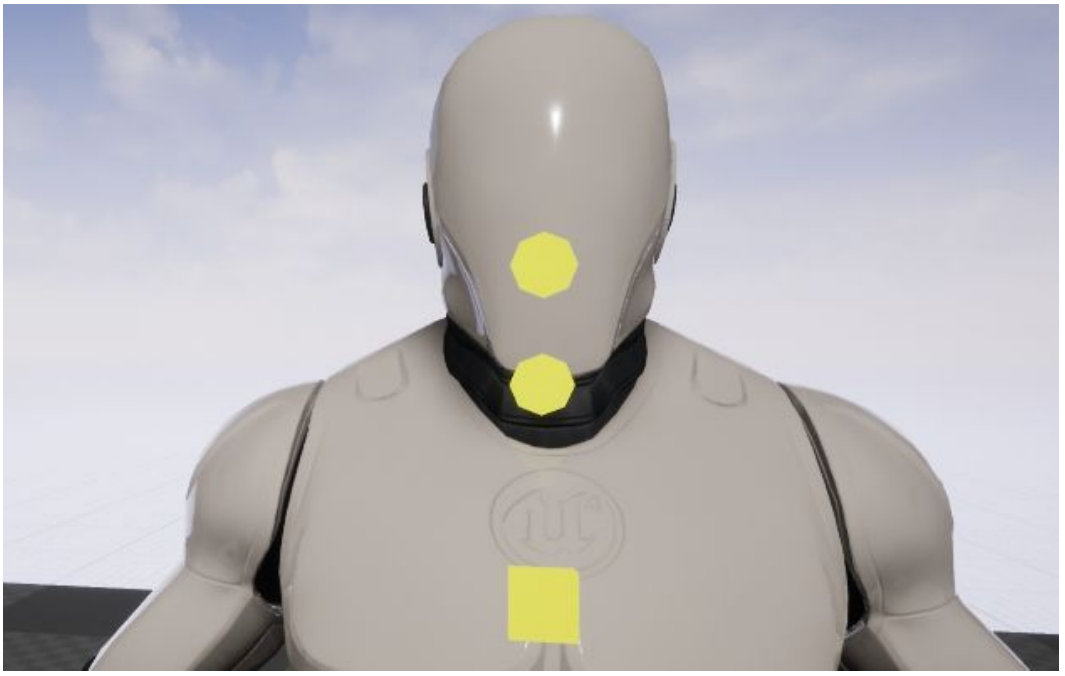
3.3.2) We are going to create hierarchy from **2** controllers for **neck** and **head** bones and parent it to **spine_03Ik** controller.

Set **Fk Hierarchy Rigs Element 0** properties:



Hierarchy Depth = 1
Tip Bone = head
Parent Controller = spine_03Ik

3.3.3) **Select** character and press **Rig** button.

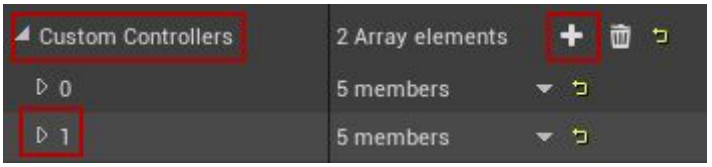


3.3.4) **Select any Controller** and press **DeleteRig** button.

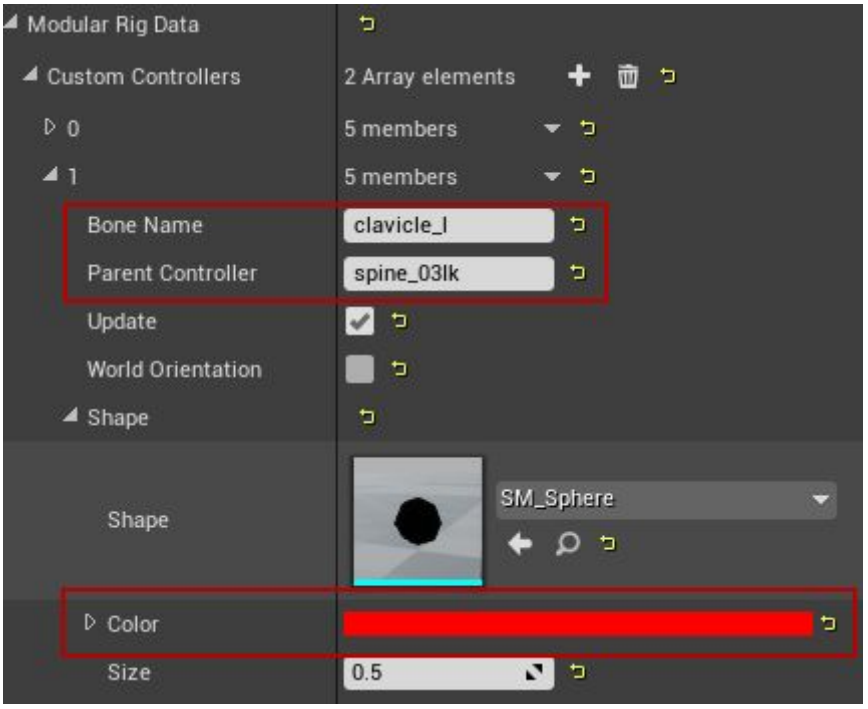
3.4) Shoulders.

Let's add few custom controllers for shoulders.

3.4.1) Add **element 1** to **Custom Controllers** array.

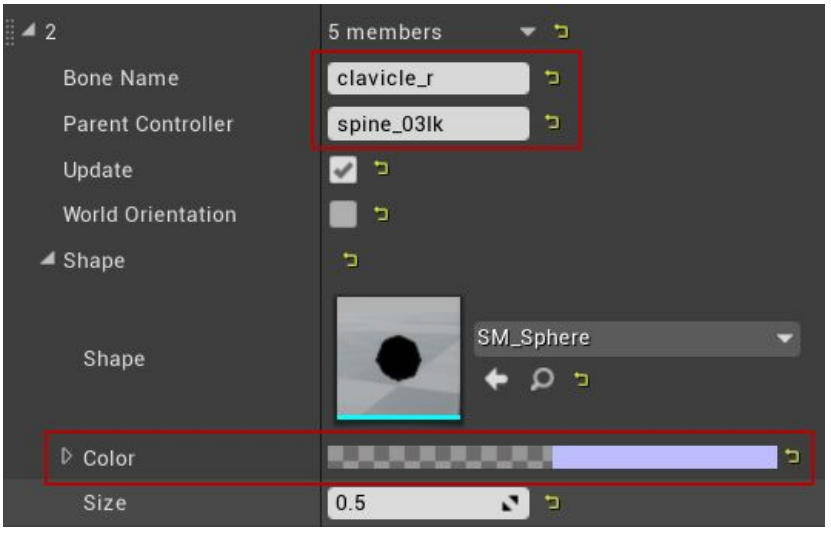


3.4.2) To rig **left shoulder** set **Custom Controllers Element 1** properties:

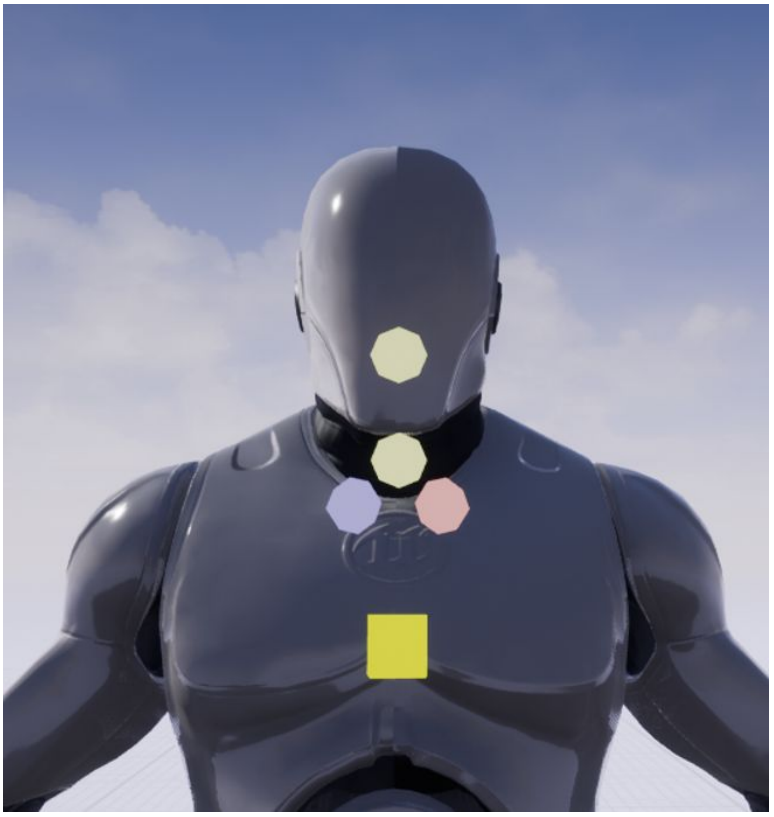


Bone Name = clavicle_l
Parent Controller = spine_03lk
Shape:
Color = Red

3.4.3) Do the same for the **right shoulder.**



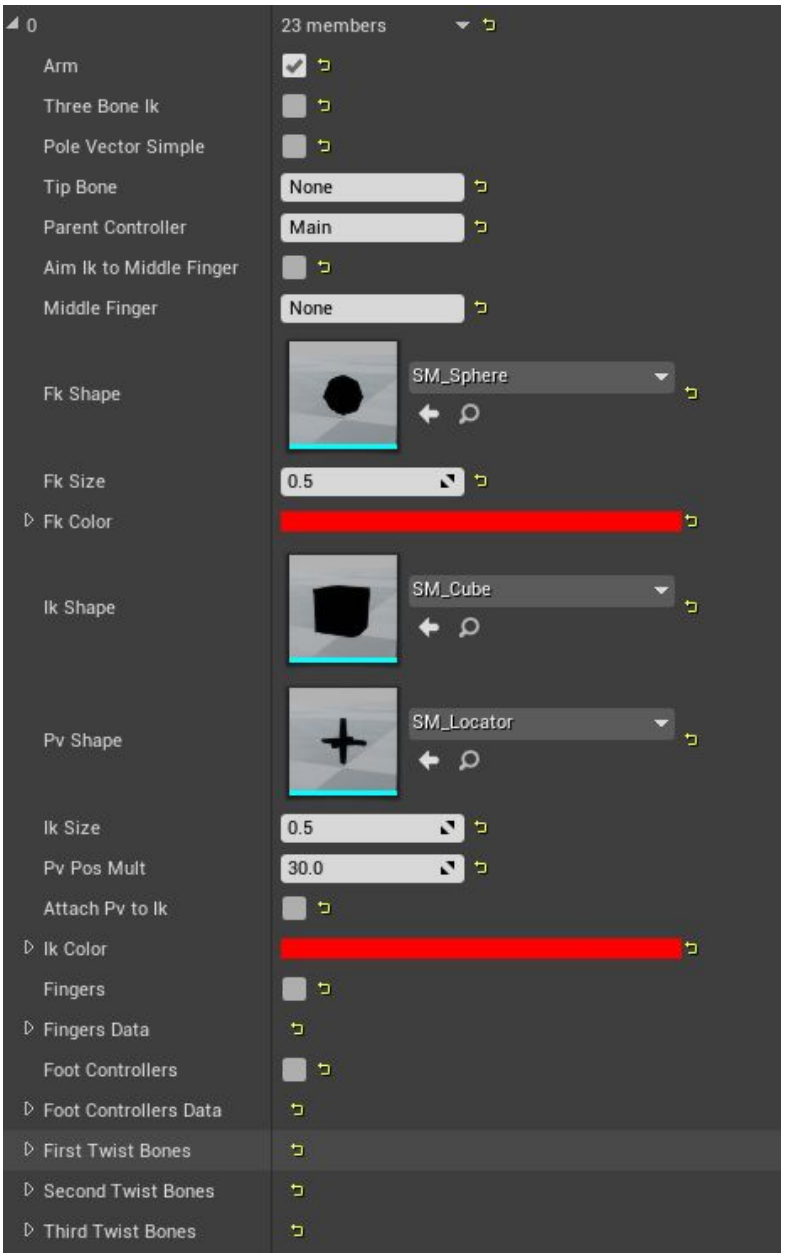
3.4.4) **Recreate Rig**



3.5) Set FkIk Rigs

Fk Ik Rigs is array of **FkIkData** structures that allows to drive **3** or **4** bone limbs using **TwoBoneIk** or **ThreeBoneIk** solvers and **Fk Hierarchy Rig** with **Fk/Ik blending** and **Fk/Ik snapping**. In addition system has settings and controllers for **fingers** and **toe**. Ik controller will be oriented to world (with offset if there is a bone to aim to). Fk controllers orientation will be the same as corresponding bones.

3.5.1) Add new element to **Fk Ik Rigs** array.



Arm - is it arm or leg? It is necessary to know for controllers orientation.
Three Bone Ik - if unchecked system will use TwoBoneIk solver to drive 3-bone limb. If checked system will use ThreeBoneIk solver to drive 4-bone limb (for quadruped back leg).
Pole Vector Simple - use simple algorithm to find pole vector location. If your characters arms or legs are absolutely straight in the reference pose, it is impossible to calculate pole vector location. Simple pole vector mode sets modified middle bone position.
Tip Bone - name of limb tip bone.
Parent Controller - name of the controllers that this controllers will be attached to.
AimIk To Middle Finger - orient Ik controller to look at middle finger if there is one.
Middle Finger - name of the middle finger or toe bone.
Fk Shape - shape of Fk controllers.
Fk Size - size of Fk controllers.
Fk Color - color of Fk controllers.
Ik Shape - shape of Ik controller.
Pv Shape - shape of pole vector controller.
Ik Size - size of Ik controllers.
Pv Pose Mult - multiply distance of pole vector controller.
Attach Pv to Ik - attach pole vector controller to ik controller.
Ik Color - color of Ik controllers.
Fingers - create fingers.
Fingers Data - custom structure that handles settings for fingers controllers.

Fingers Tip Bones - array of all limb fingers tip bone names.
Phalanx Num - number of phalanx in finger.
Size - size of finger controllers (color and shape is the same as for Fk Hierarchy)
Parent to Middle Finger - parent finger controllers to middle finger (for toes)

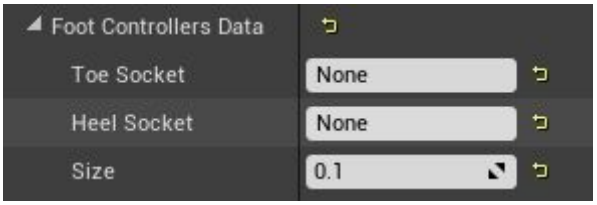
If your character has 5 fingers and 3 phalanx, set tip bone names in this order:

0 = thumb_03_l
1 = index_03_l
2 = middle_03_l
3 = ring_03_l
4 = pinky_03_l

Additional properties will be available to control fingers using FkIk controller.

If your character has fingers with different amount of phalanx you can create some of fingers using Fk Hierarchy Rigs array.

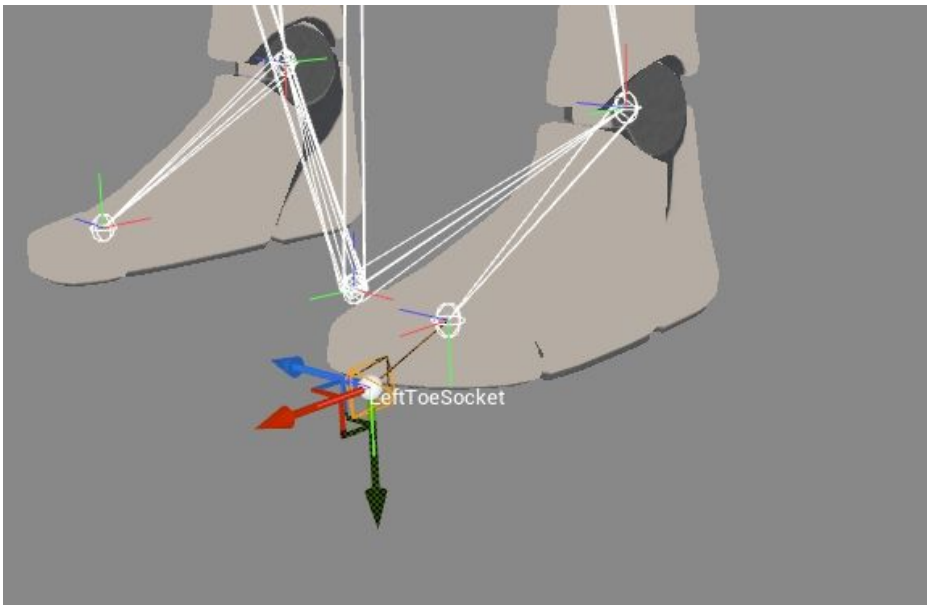
Foot Controllers Data - custom structure that handles settings for additional foot roll properties.



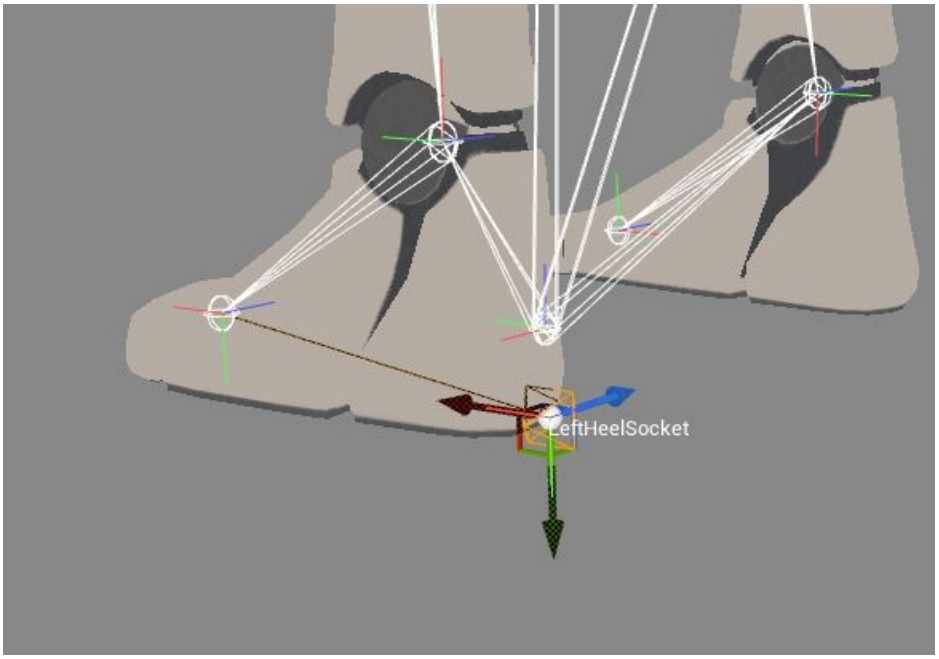
In order to use additional foot roll properties you need to add toe sockets and heel sockets to toe bones.

For more details on how to add sockets please visit Skeletal Mesh Sockets page in unreal engine documentation.

Toe Socket - socket name to get position for toe rotation pivot.

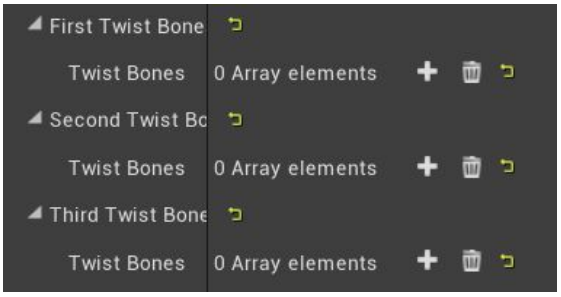


Heel Socket - socket name to get position for heel rotation pivot.



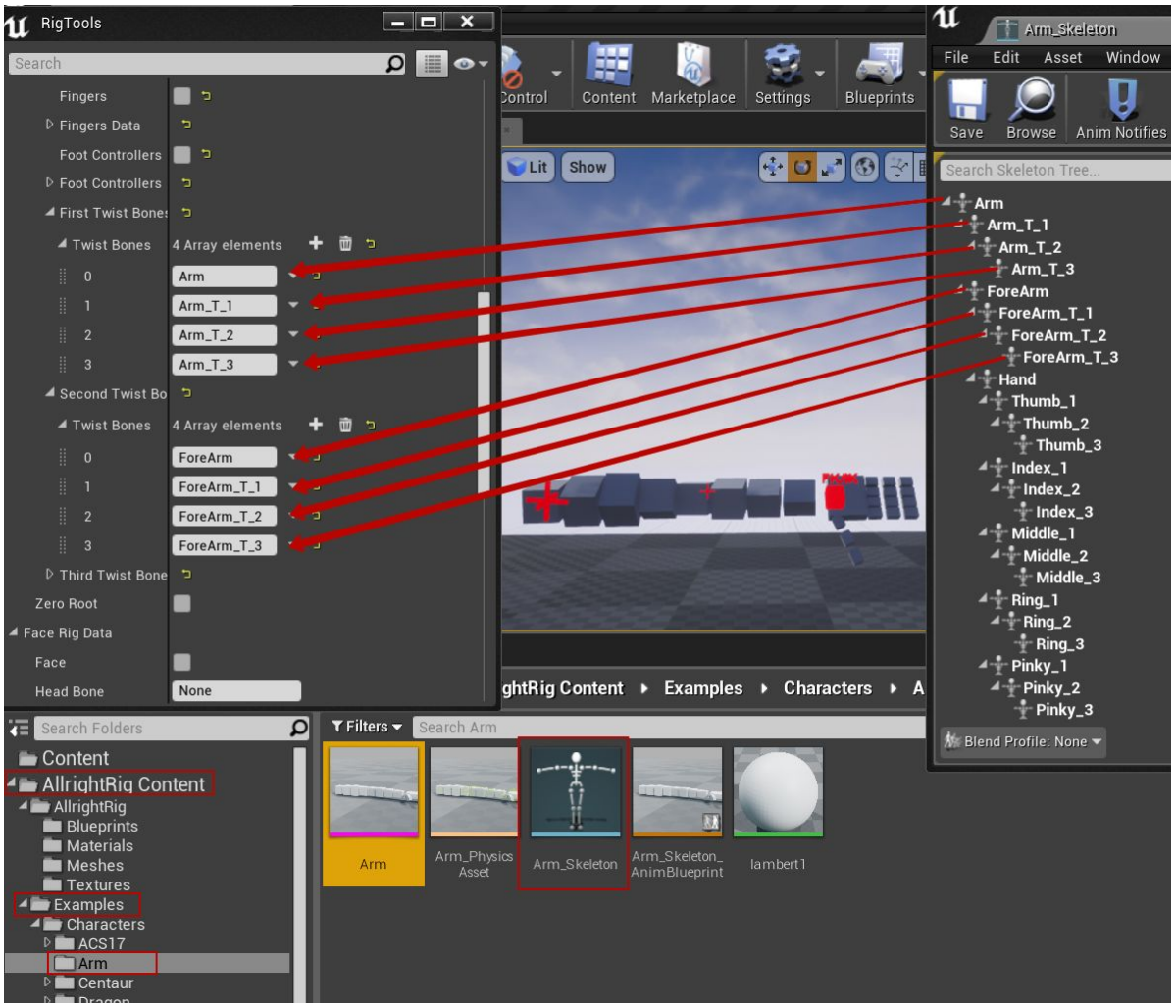
Size - size of toe controller (color and shape is the same as for Fk Hierarchy)

First/Second/Third Twist Bones - arrays of twist bone names for **first/second/third** bone in limb hierarchy.

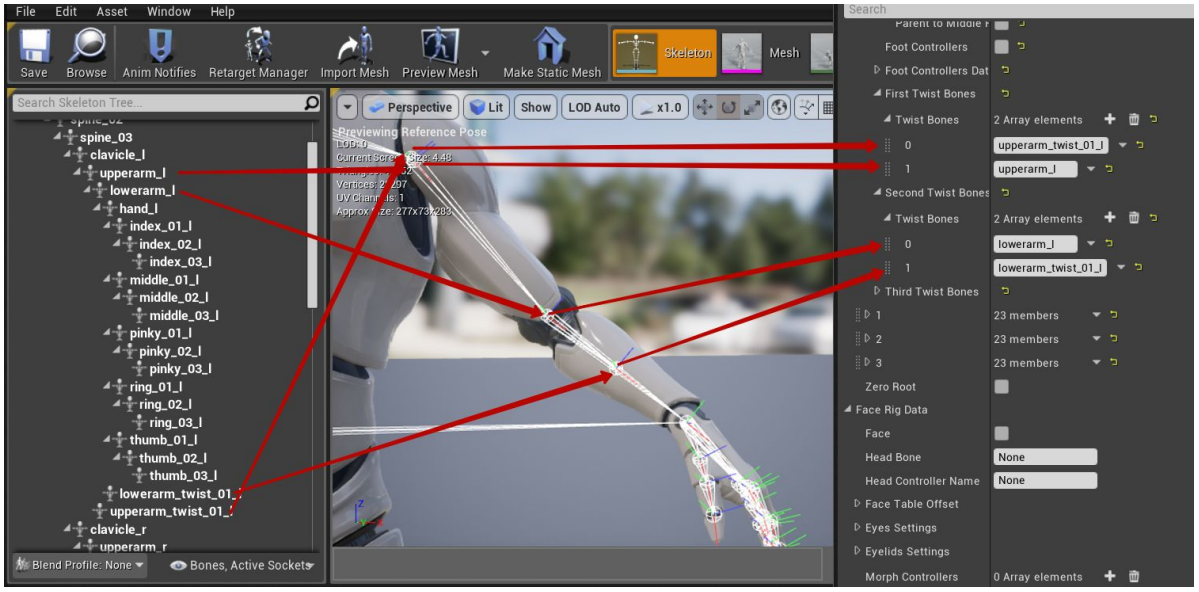


Here you need to set twist bones for every limb in top down order:
0 = upper bone
1 = bone below
2 = bone below …
The order here does not depend on hierarchy structure but it depends on the location of bones and how do they influence on skin.

For example, twist bones arrays for **Arm** asset under **Examples/Arm** folder will be like this:

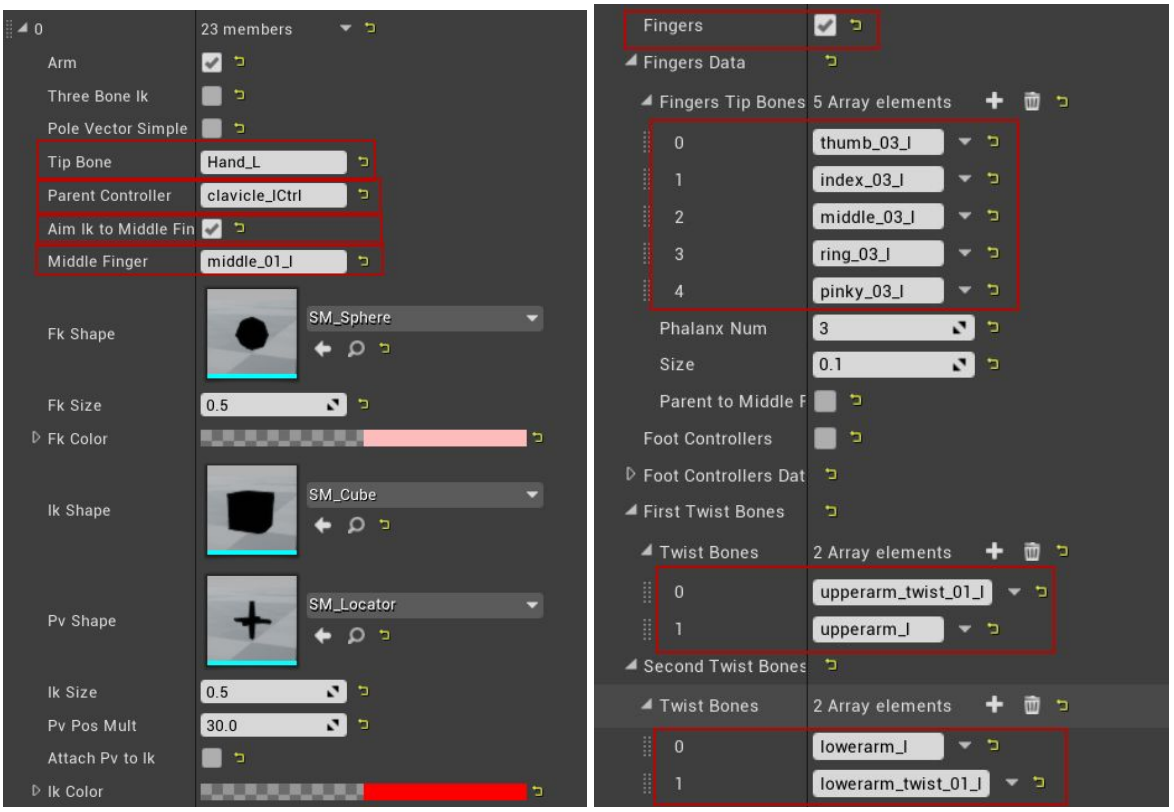


But for **SK_Mannequin** arm it will be like this:



I agree, that skinning logic of mannequin twist bones is pretty weird, but there might be a reason for that.

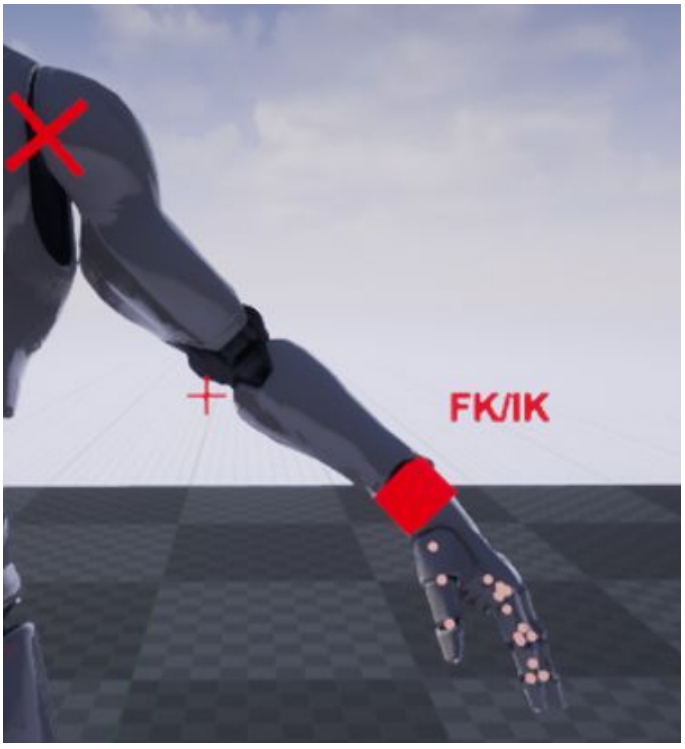
3.5.2) To rig **left arm** set **Fk Ik Rigs Element 0** properties:



Tip Bone = hand_l
Parent Controller = spine_03Ik
Aim Ik To Middle Finger = enabled
Middle Finger = middle_01_l
Fingers = enabled
Fingers Data:
Fingers Tip Bones:
0 = thumb_03_l
1 = index_03_l
2 = middle_03_l
3 = ring_03_l
4 = pinky_03_l
First Twist Bones:
0 = upperarm_twist_01_l
1 = upperarm_l
Second Twist Bones:
0 = lowerarm_l
1 = lowerarm_twist_01_l

3.5.3) **Select** character and press **Rig** button.

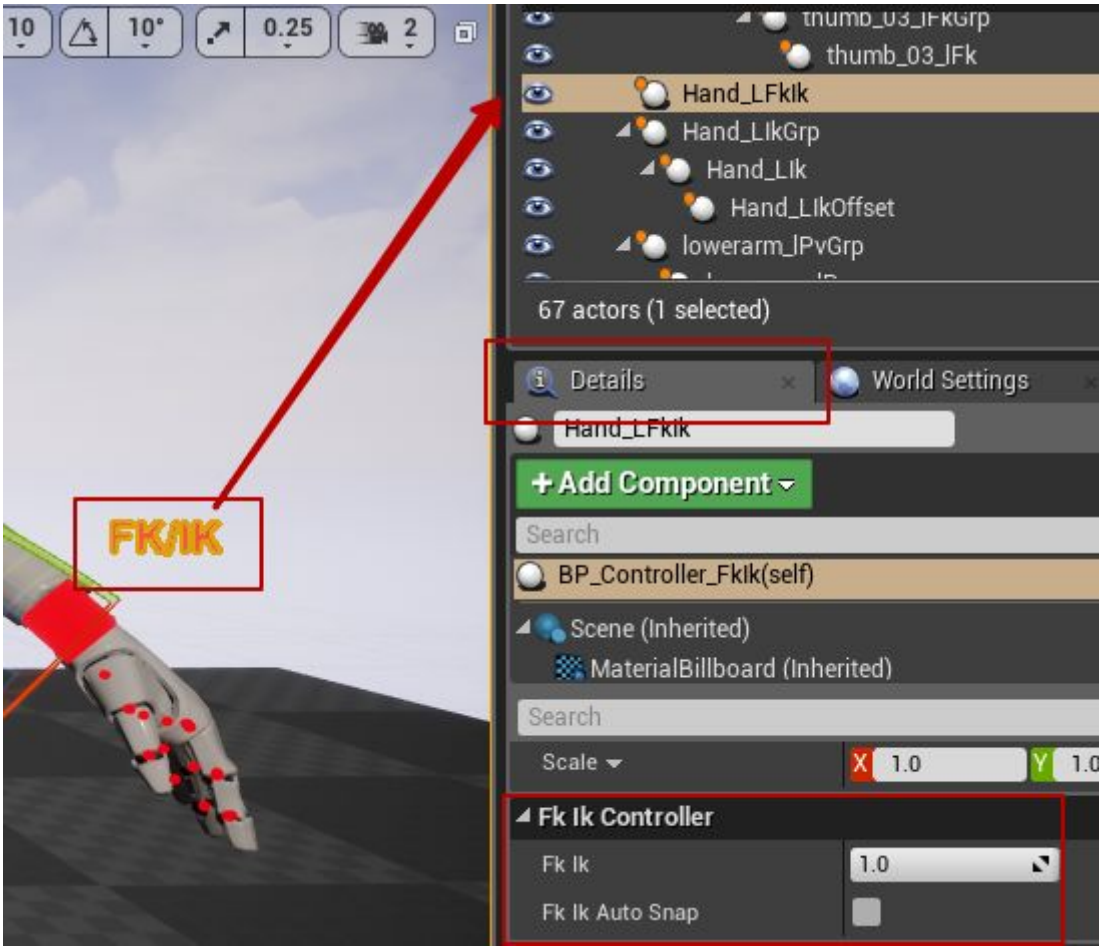
IK controllers:



FK controllers:

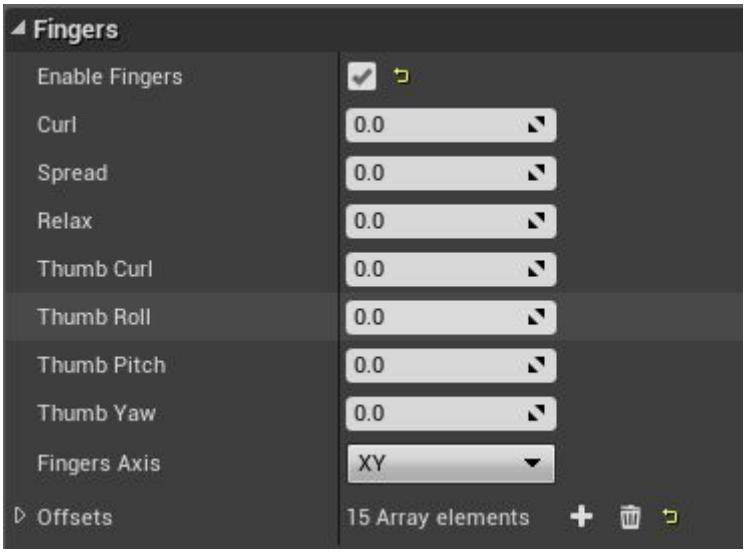
3.5.4) Select **Fk/Ik** controller and go to **Details** panel.

Under **Fk/Ik Controller** category there are following properties:



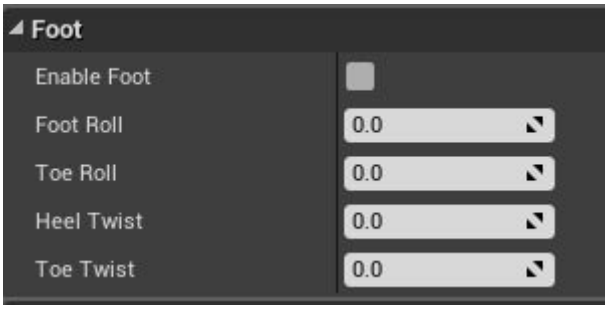
Fk Ik - Fk/Ik blending.
Fk Ik Auto Snap - snap all Fk/Ik controllers to current bones positions in realtime.

Under **Fingers** category there are following properties:



Enable Fingers - enable additional fingers properties.
Curl - bend all fingers except thumb.
Spread - spread all fingers except thumb.
Relax - add relaxed fingers pose
Thumb Curl - bend thumb.
Thumb Roll - thumb roll.
Thumb Pitch - thumb pitch.
Thumb Yaw - thumb yaw.
Fingers Axis - finger controllers orientations. This property filled automatically, during creation of the rig.
Offsets - finger controllers offsets. This property filled automatically, during creation of the rig.

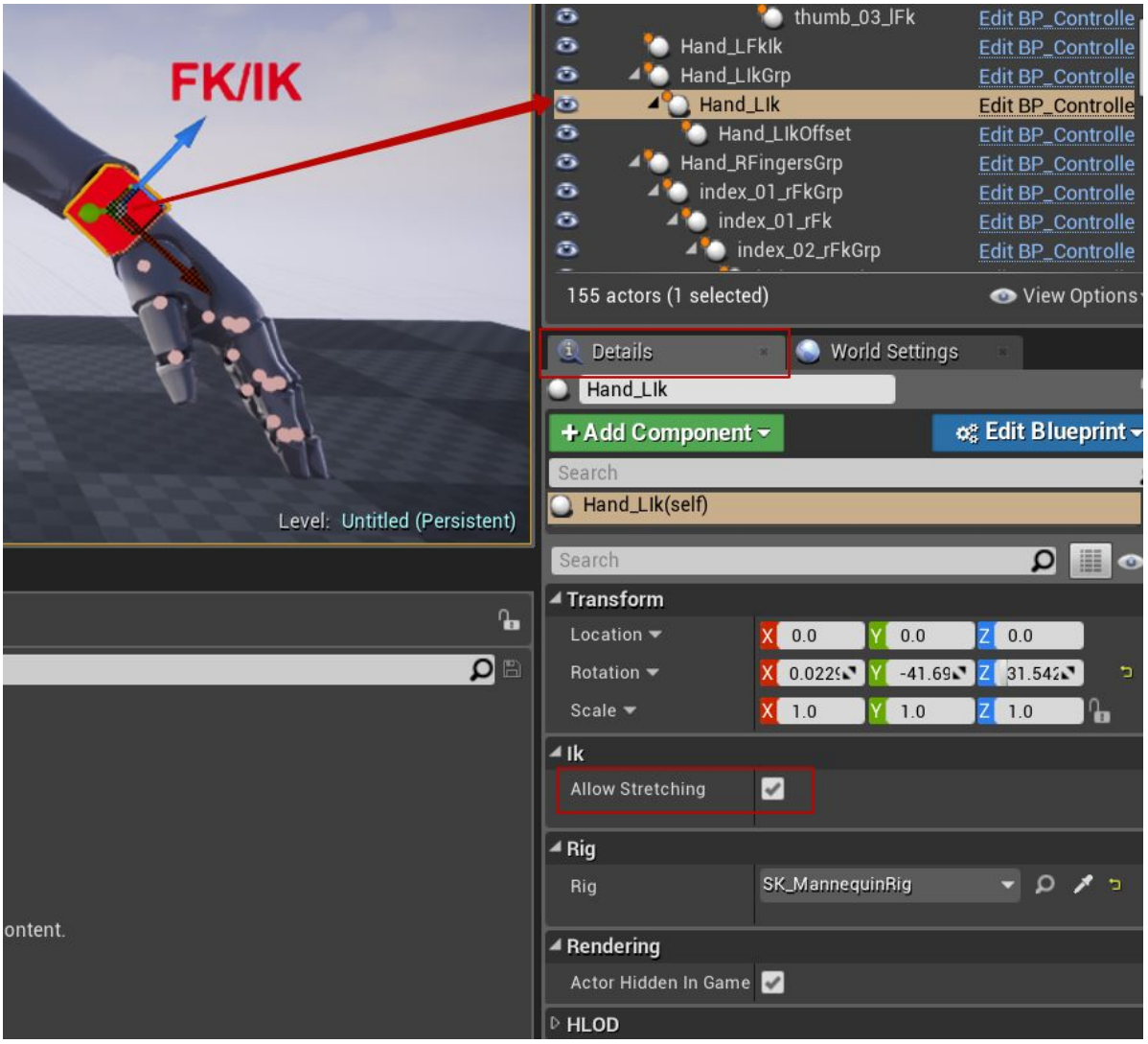
Under **Foot** category there are following properties:



Enable Foot - enable additional Ik foot controllers.
Foot Roll - foot roll.
Toe Roll - toe roll.
Heel Twist - twist leg around heel.
Toe Twist - twist leg around toe.

3.5.5) Select **Ik** controller and go to **Details** panel.

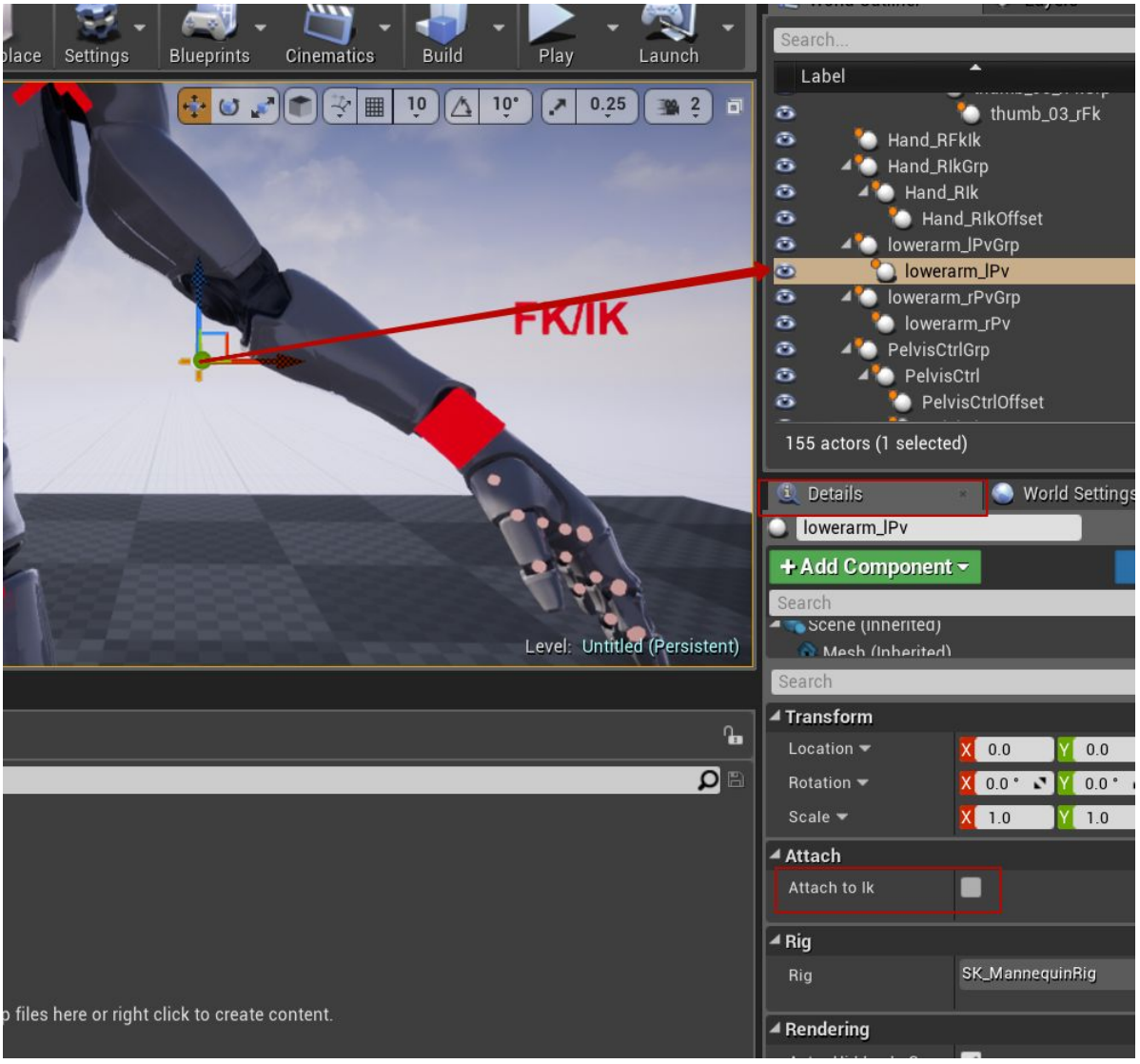
Under **Ik** category there are following properties:



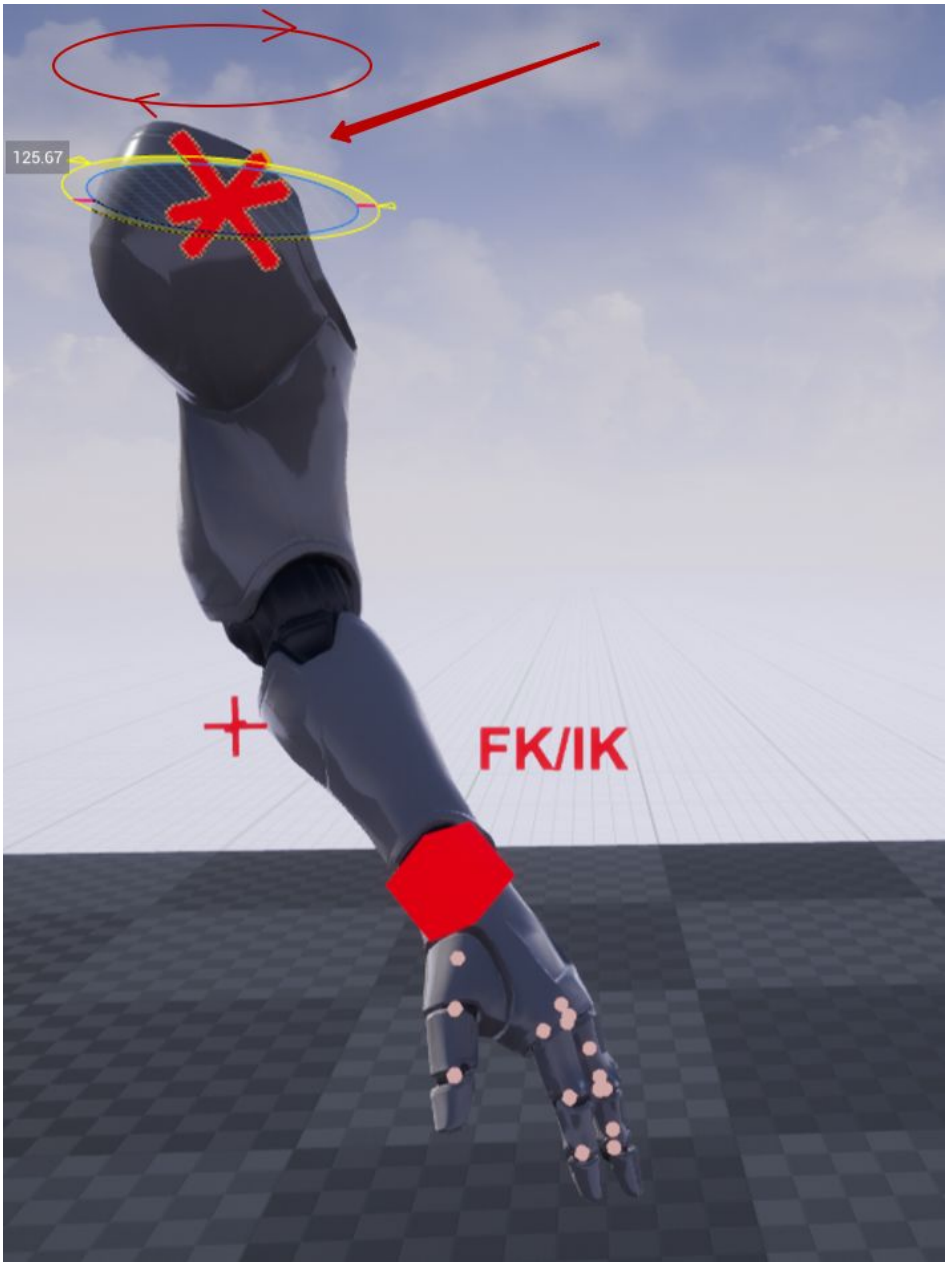
Allow Stretching - allow Ik stretching.

3.5.6) Select **Pole Vector** controller and go to **Details** panel.

Under **Attach** category there are following properties:



Attach to Ik - if enabled, **this** controller will be attached to **Ik** controller, if not - to **Main** controller. Currently this property is not keyable.

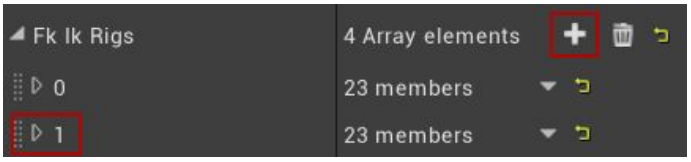


Helper controller has few functions:

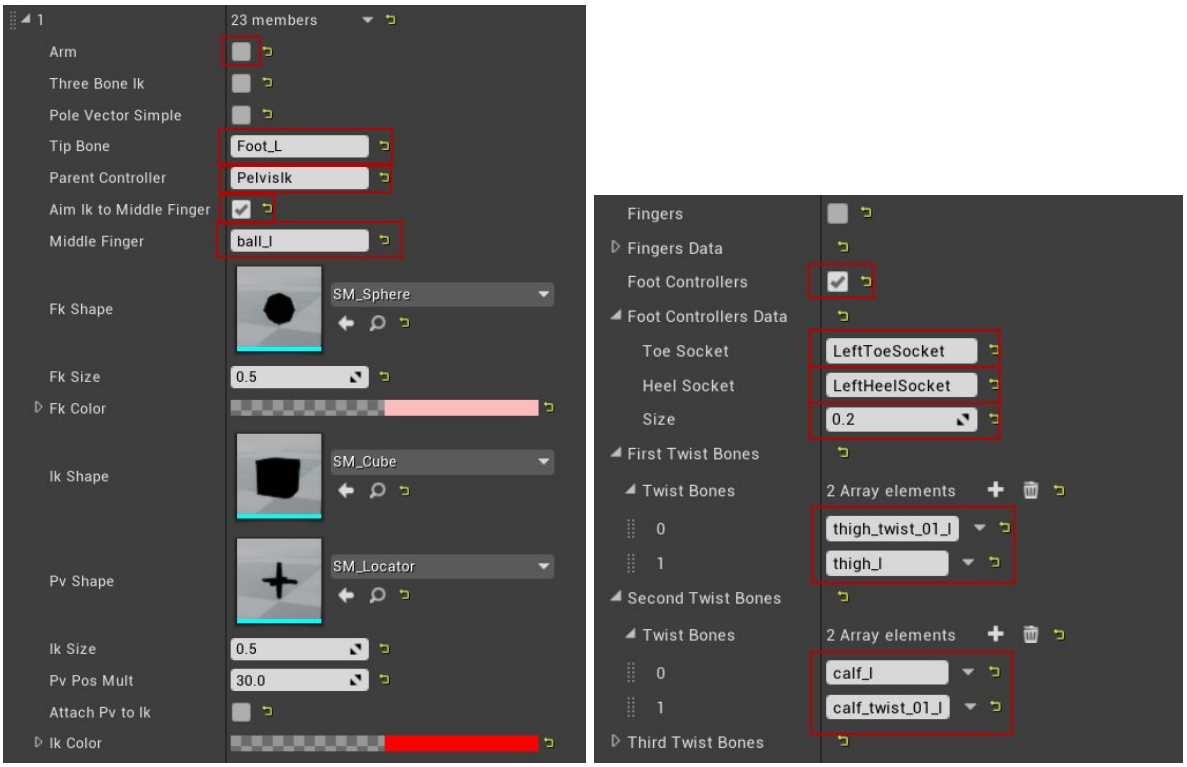
- adjust twist
- adjust rotation for upper bone in ThreeBoneIk (quadruped back leg)
- move ik root

3.5.7) **Select any Controller** and press **DeleteRig** button.

3.5.8) Add **element 1** to **Fk Ik Rigs** array.

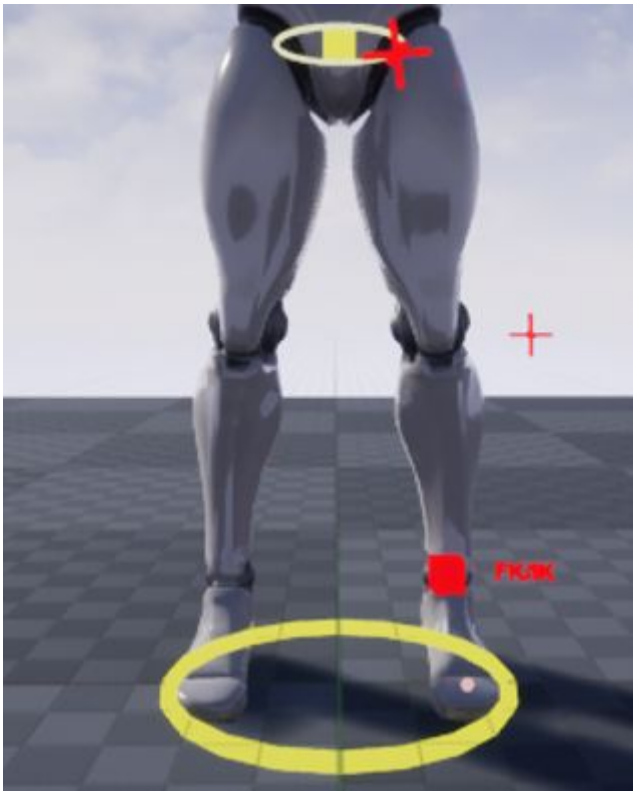


3.5.9) To rig **left leg** set **Fk Ik Rigs Element 1** properties:

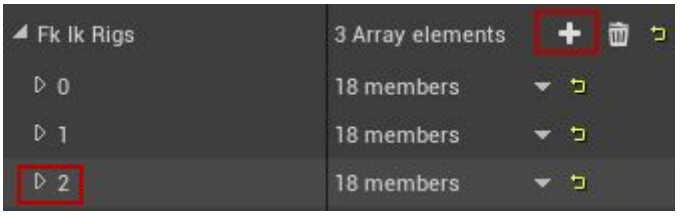


Arm = disabled
Tip Bone = Foot_L
Parent Controller = PelvisIk
Aim Ik To Middle Finger = enabled
Middle Finger = ball_l
Foot Controllers = enabled
Foot Controllers Data (more details in paragraph 3.4.1)) :
Toe Socket = LeftToeSocket
Heel Socket = LeftHeelSocket
First Twist Bones:
0 = thigh_twist_01_l
1 = thigh_l
Second Twist Bones:
0 = calf_l
1 = calf_twist_01_l

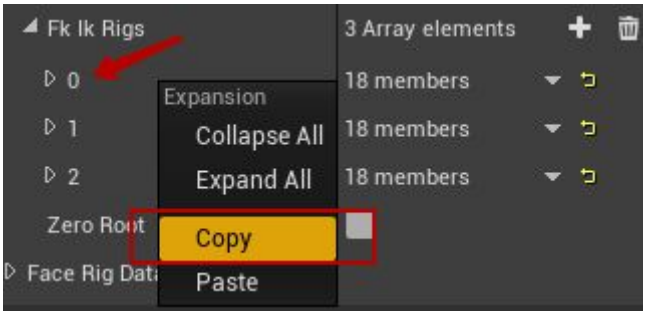
3.5.10) **Select** character and press **Rig** button.



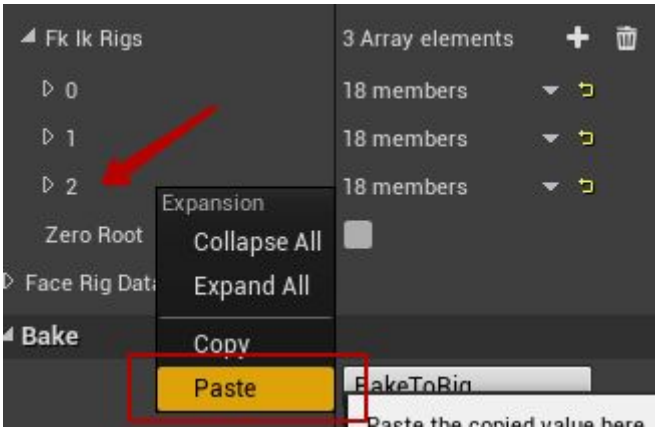
3.5.11) Add **element 2** to **Fk Ik Rigs** array.



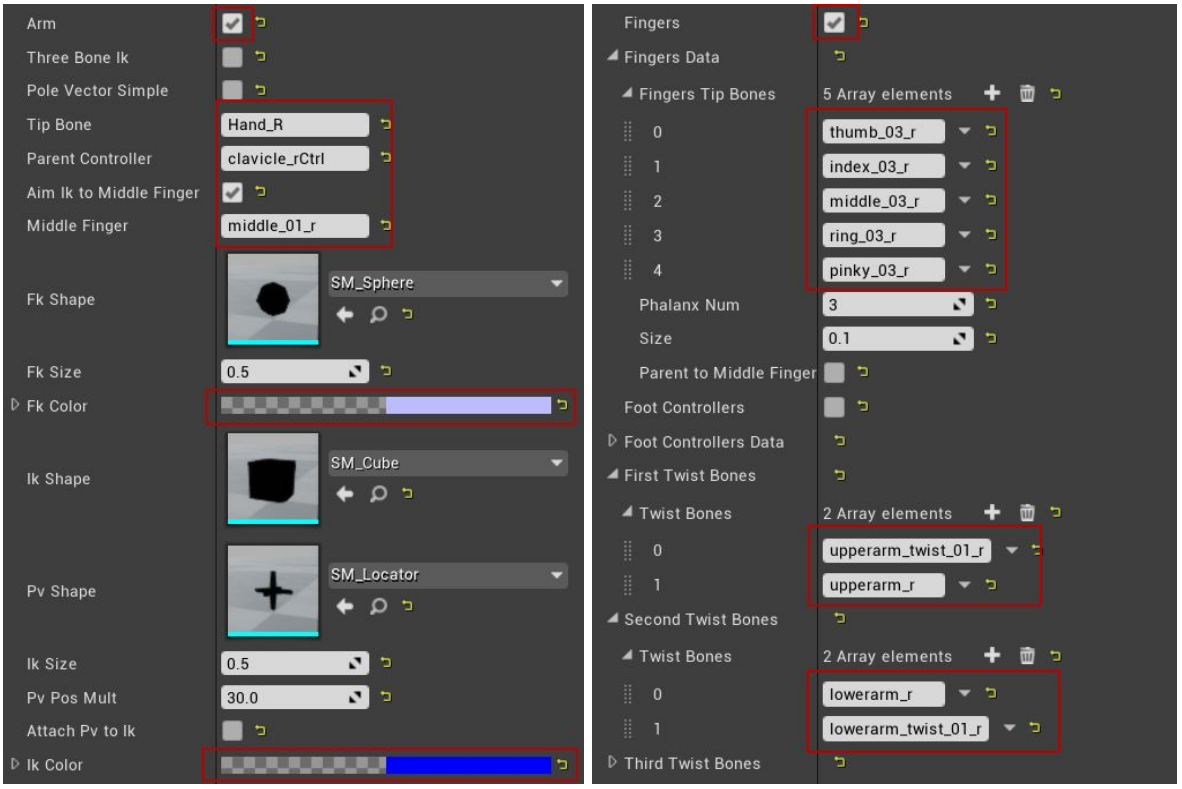
3.5.12) Click **right mouse button** on **element 0** of **Fk Ik Rigs** array and choose **Copy**.



3.5.13) Click **right mouse button** on **element 2** of **Fk Ik Rigs** array and choose **Paste**.

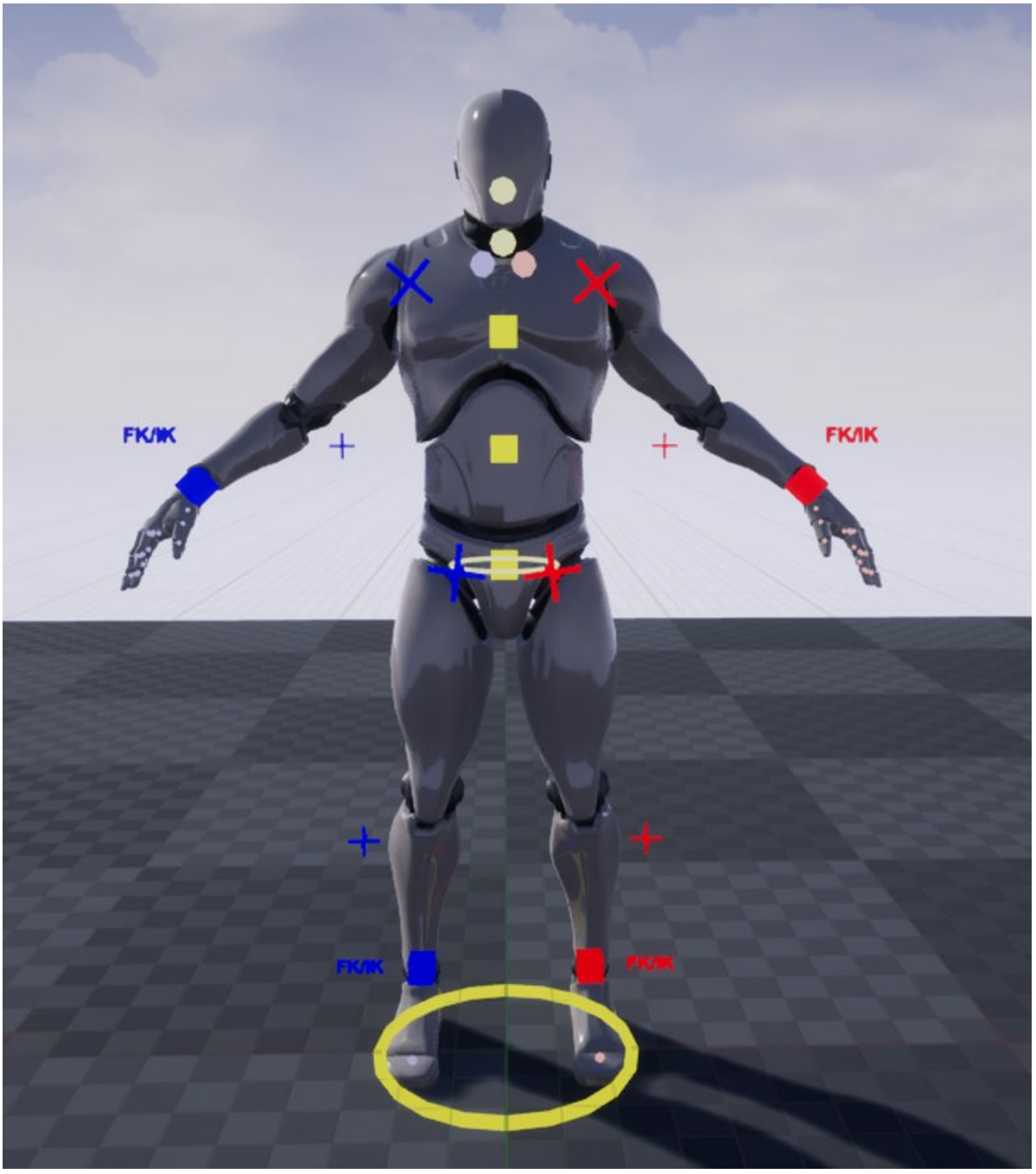


3.5.14) Replace all **“l” to “r”** and set color to **blue**.



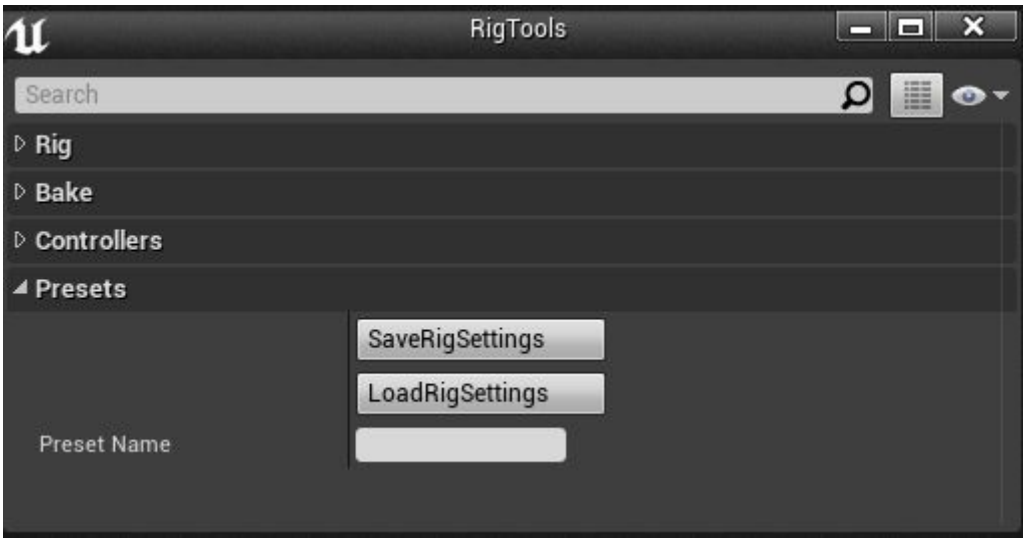
3.5.15) Repeat same iterations for the **right leg.**

3.5.16) **Recreate rig**.



4) Presets

It is a good time to **save** all **Rig Tools window** settings into a **preset** in order to **load** them at any moment. You can find **Presets** category on the **bottom** of the **Rig Tools** window.



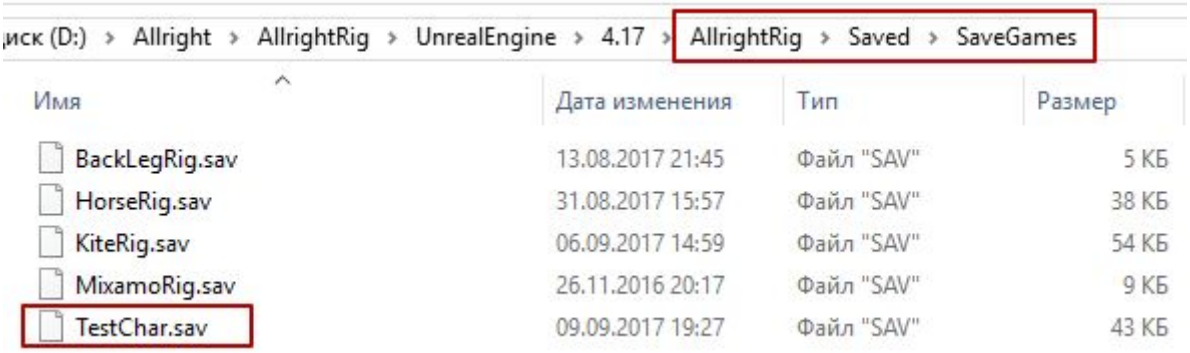
System is implemented by using **SaveGameObject**. You can **find** or **add** your preset files under **YourProjectName\Saved\SaveGames** folder.

*Preset file **does not depend** on **engine version**.*

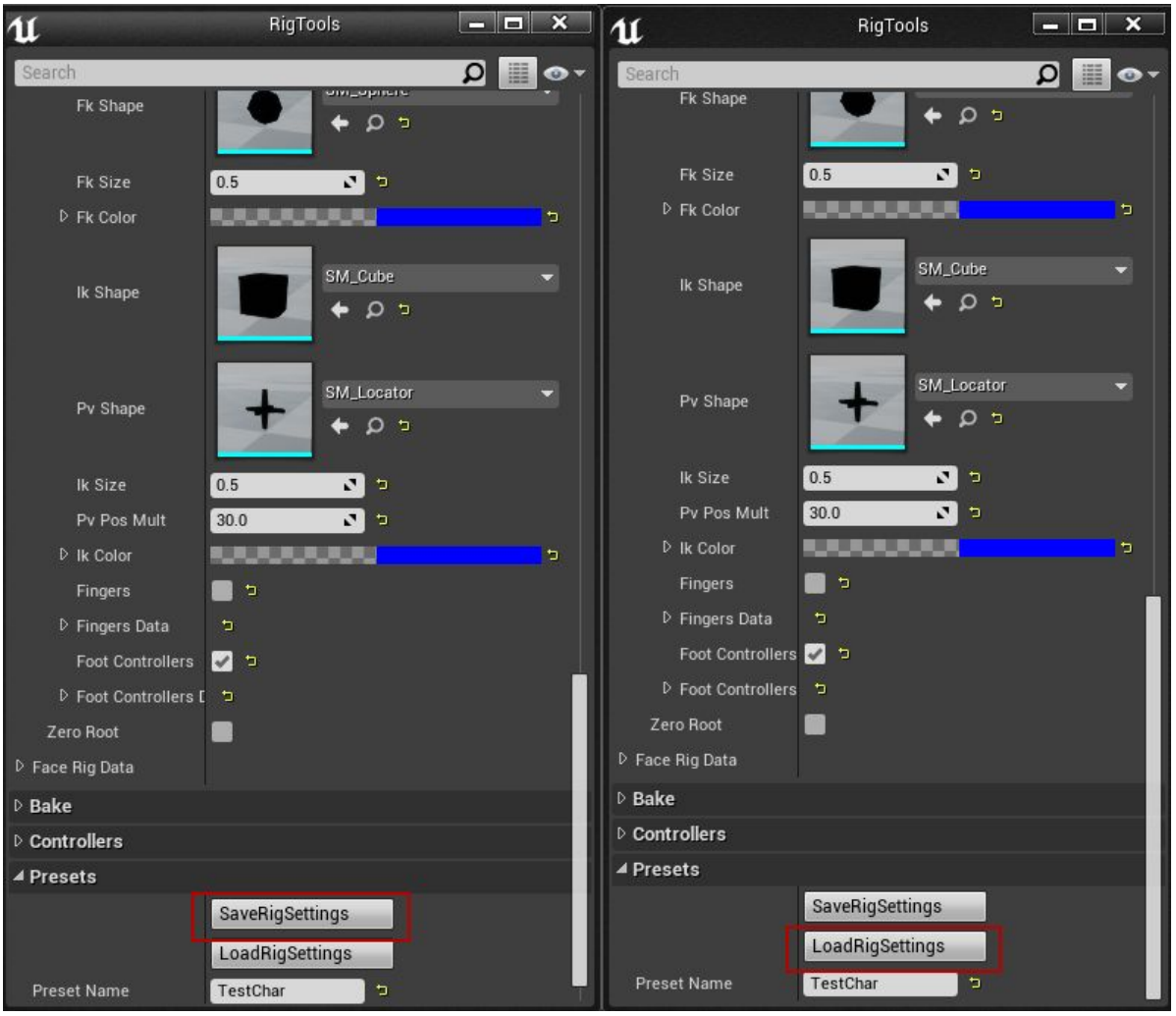
SaveRigSettings - save Rig Tools window settings to Project\Saved\SaveGames.
LoadRigSettings - load Rig Tools window settings from Project\Saved\SaveGames
Preset Name - name for preset file.

4.1) Go to the **Presets** category, type the **name** for a preset file and press **SaveRigSettings.**

If you go to **YourProjectName\Saved\SaveGames** folder, you can find your preset file with extansion **.sav**.



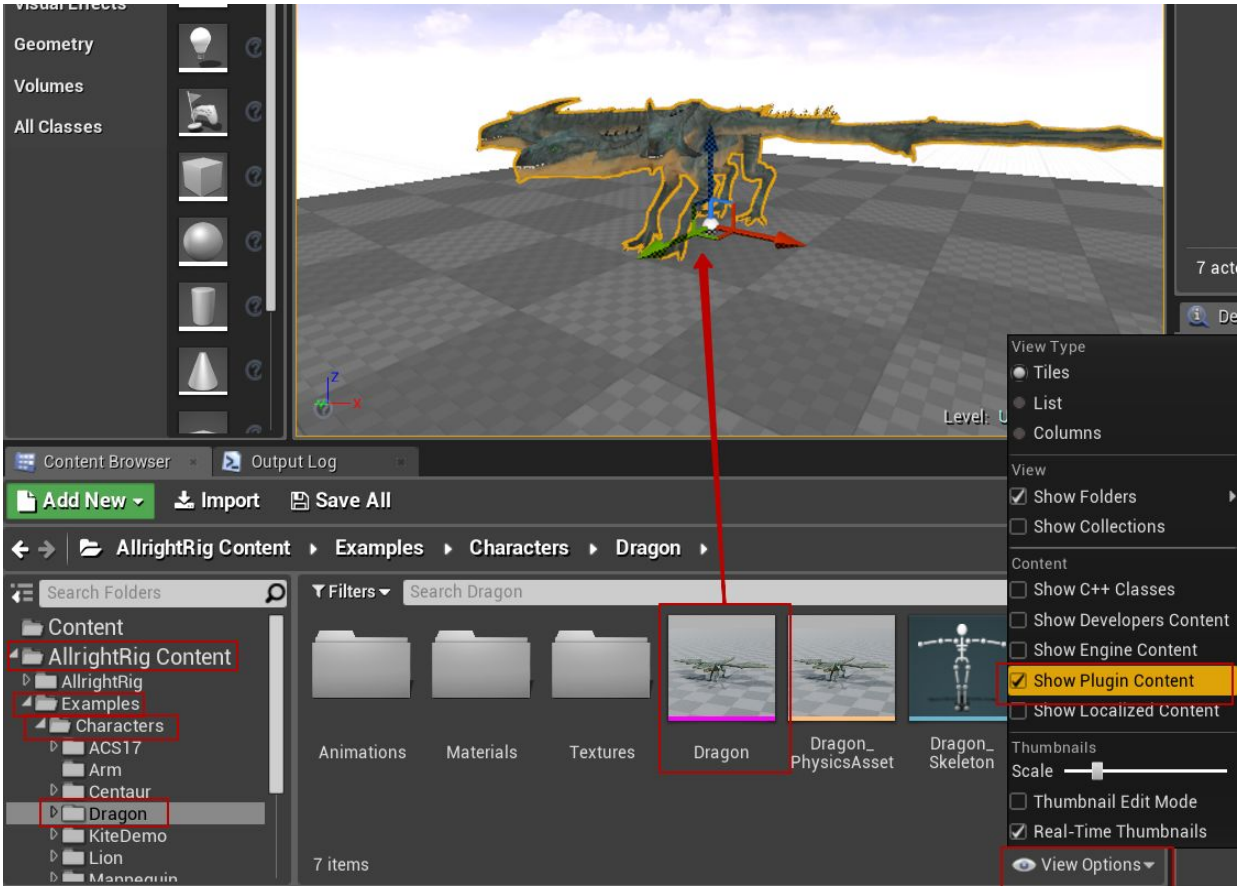
4.2) Open a **new** instance of **Rig Tools** window, go to **Presets** category, type **Preset Name** and push **LoadRigSettings** button. All settings should be equal.



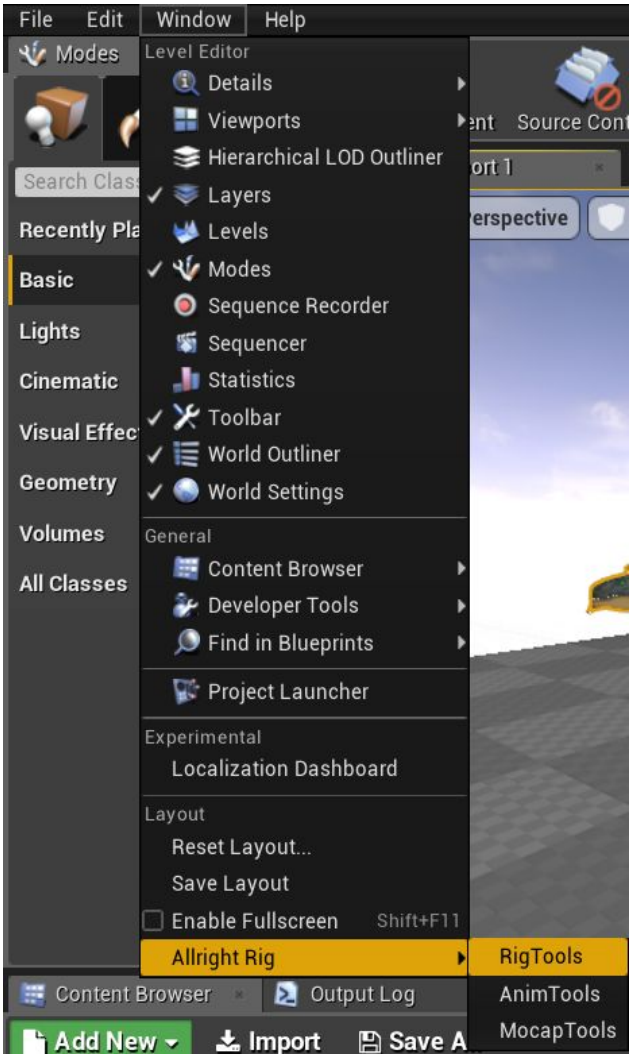
4.3) **Recreate rig** from **new** Rig Tools window instance to check if everything is fine. Now you can close both windows and load settings at any time.

5) 4-bone limb.

5.1) In **Allright Rig Content** content under folder go to **Examples/Characters/Dragon** folder and add **Dragon** skeletal mesh to the scene.

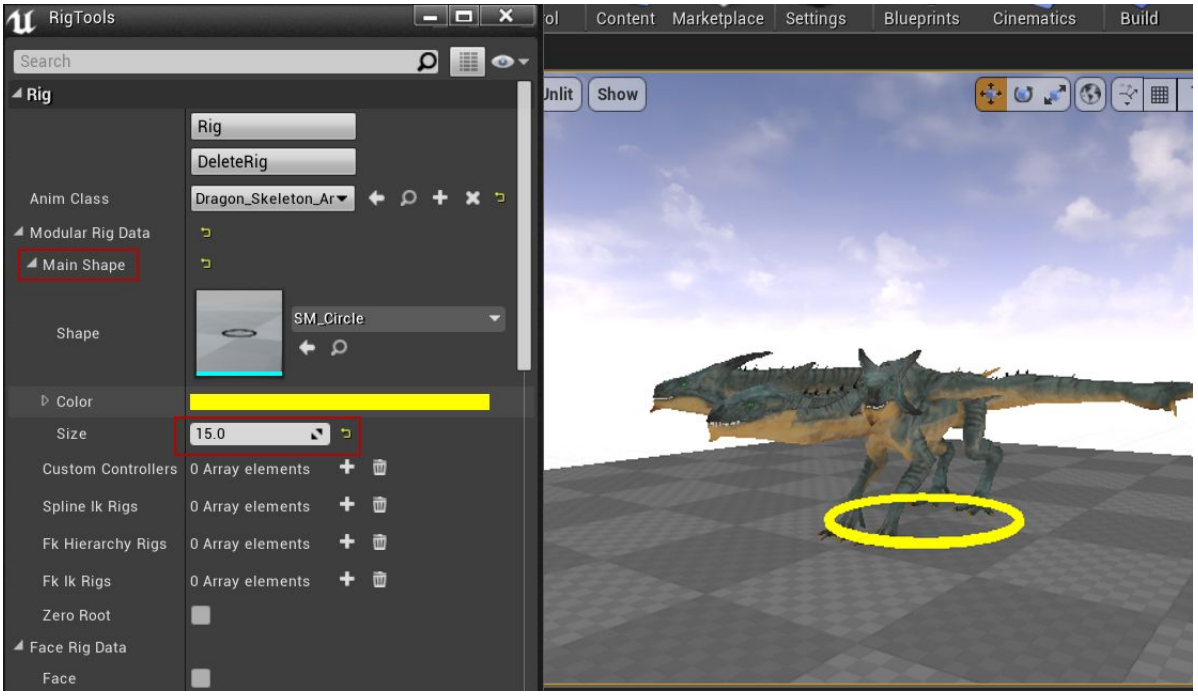


5.2) Open **new** instance of **Rig Tools** window.

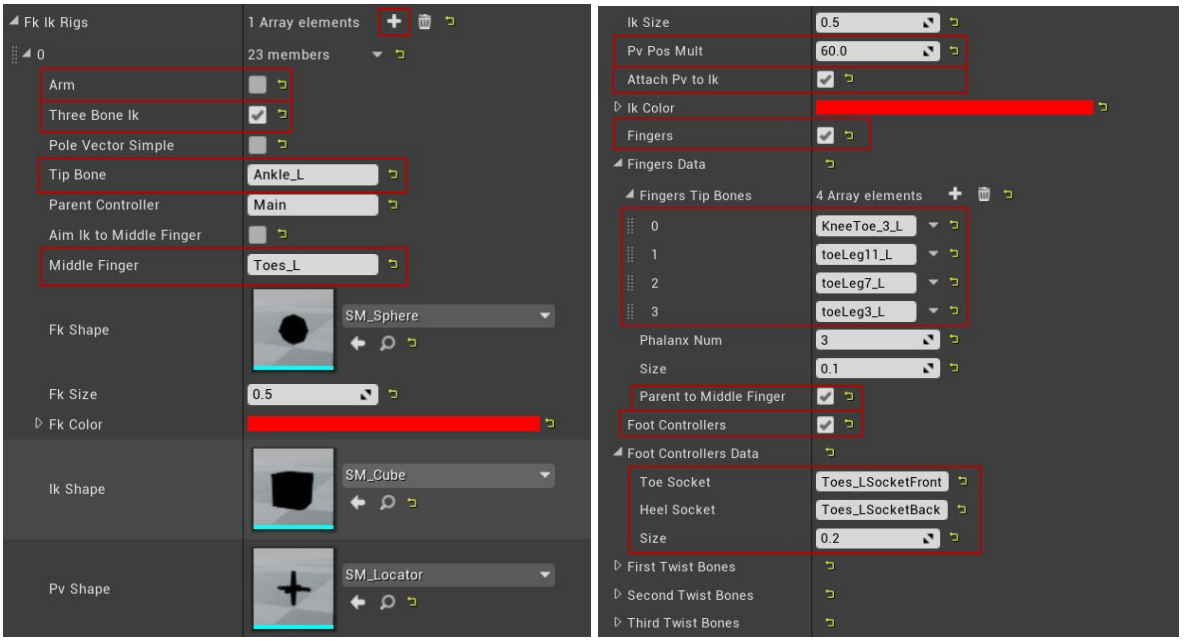


5.3) Set **Anim Class** property to **Dragon_Skeleton_AnimBlueprint**.

5.4) Set **Size** of **Main Shape** to **15** and **create rig**.

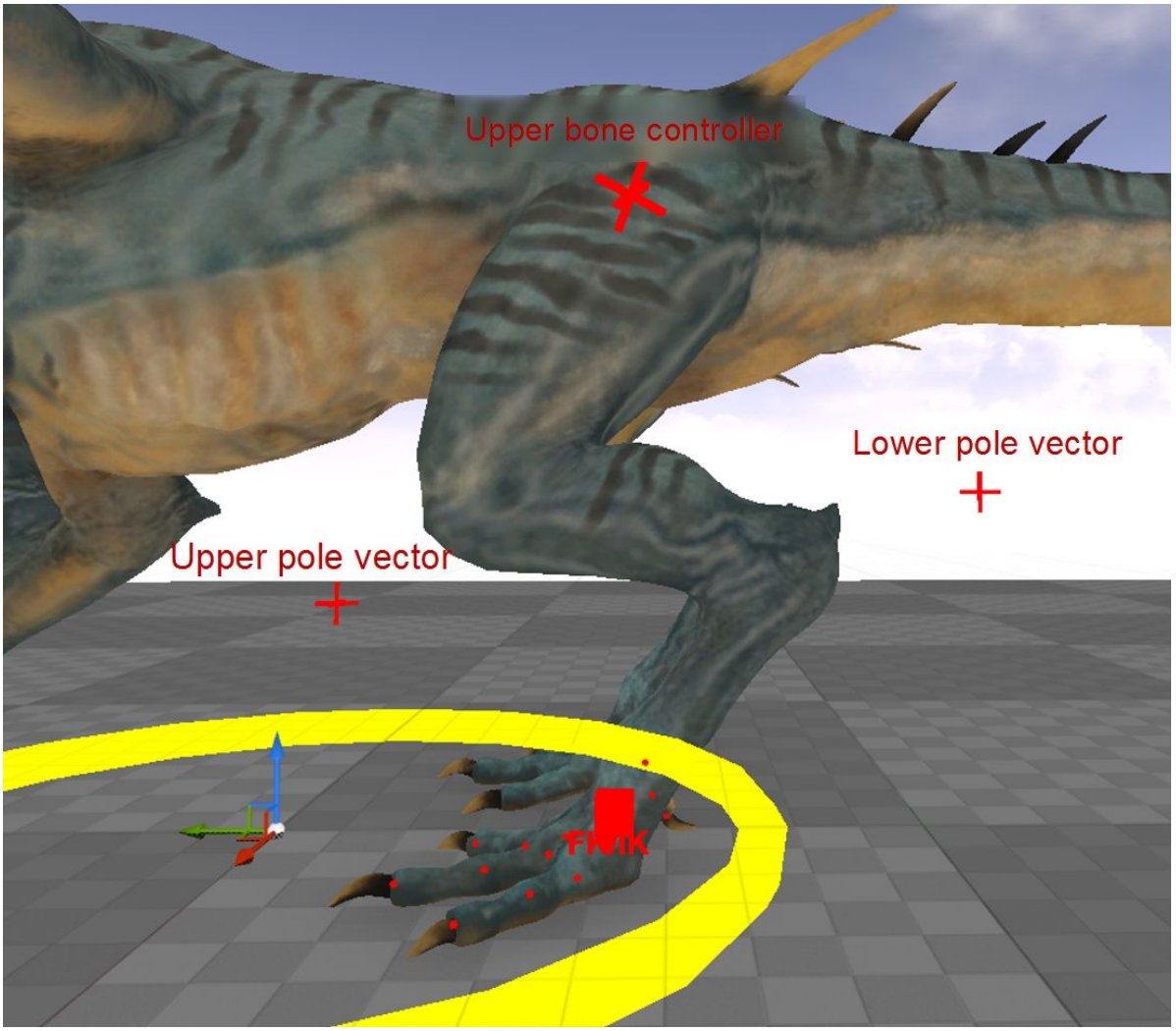


5.5) Add new element to **Fk Ik Rigs** array and **set** following **properties**:

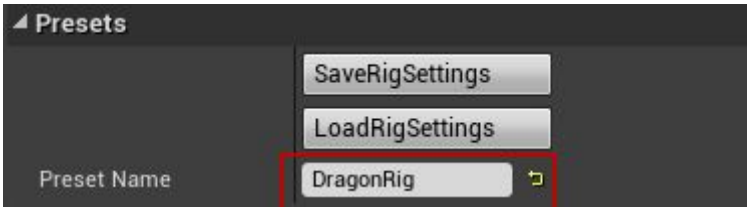


Arm = disabled
Three Bone Ik = enabled
Tip Bone = Ankle_L
Middle Finger = Toes_L
Pv Pos Mult = 60
Attach Pv to Ik = enabled
Fingers = Enabled
Fingers Data:
    Fingers Tip Bones:
        0 = KneeToe_3_L
        1 = toeLeg11_L
        2 = toeLeg7_L
        3 = toeLeg3_L
    Parent to Middle Finger = enabled
Foot Controllers = enabled
Foot Controllers Data:
    Toe Socket = Toes_LSocketFront
    Heel Socket = Toes_LSocketBack
    Size = 0.2

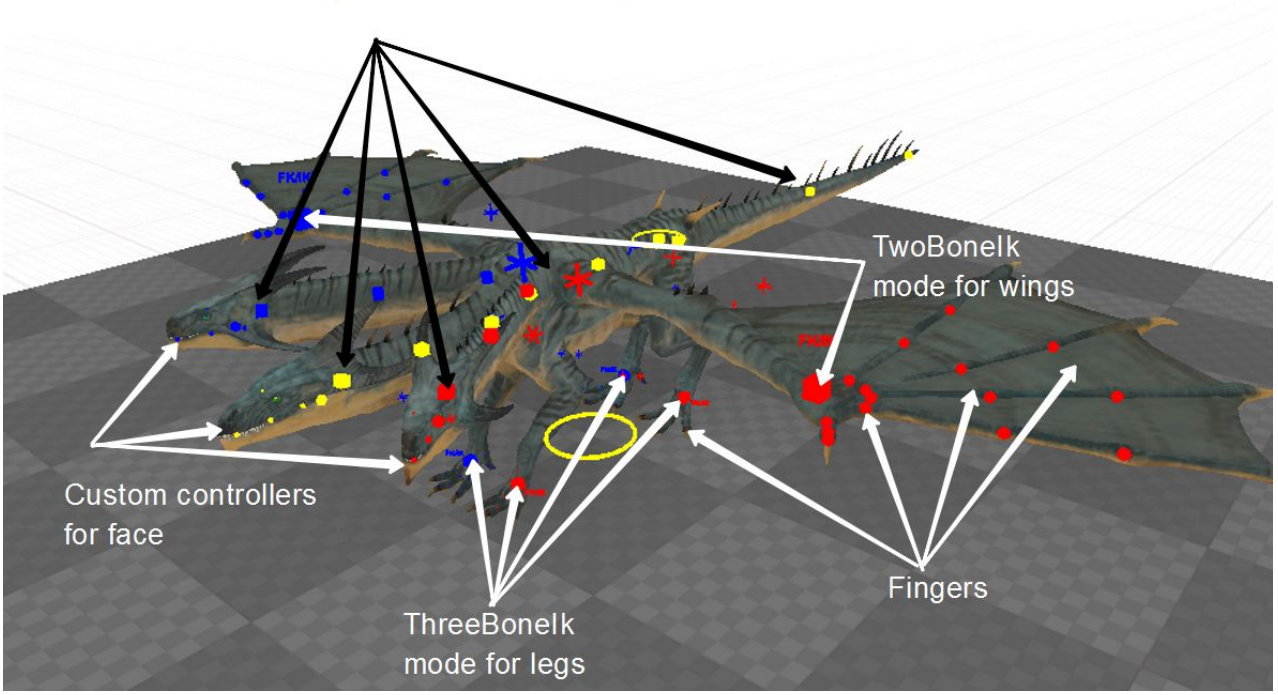
5.6) **Recreate rig**.



5.7) Load **DragonRig** preset and **recreate rig**.



SplineIk for 3 heads, spine and tail

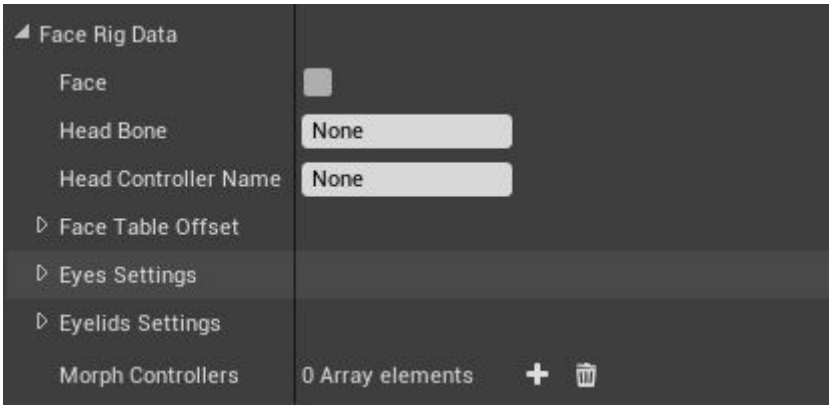


You can use different combinations of modules for different parts of your character.

6) Face rig

Facial system allows you to create controllers using eye **bones** and **morph** targets. You can easily setup joints for eyes and eyelids using **Eyes Rig Script For Maya**. Or you can create same joints hierarchy in any other software.

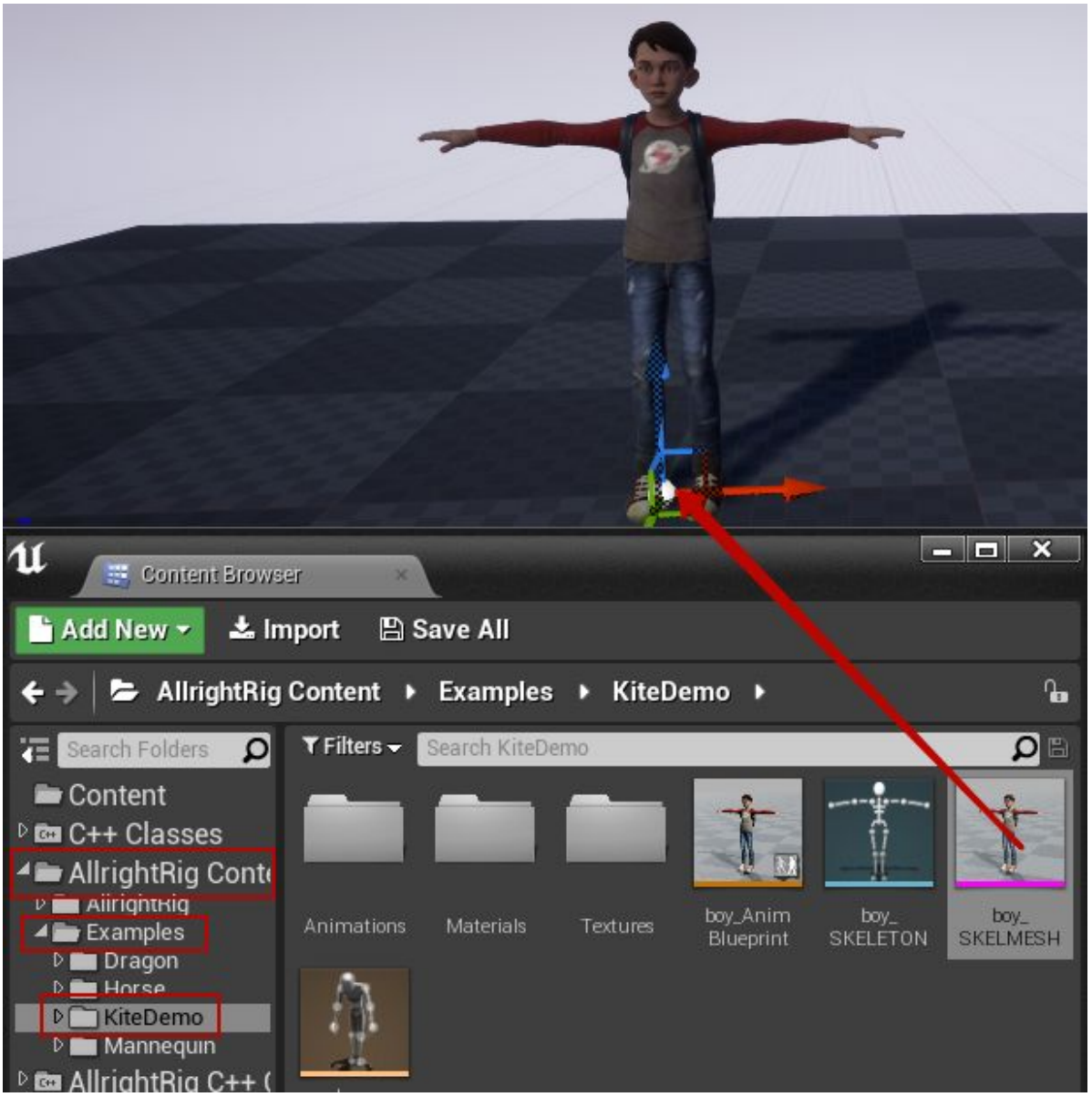
Face Rig Data is a custom structure that handles all structures for facial setup.



Face - create and update face rig
Head Bone - head bone name
Head Controller Name - head controller name (attach face controllers to)
Face Table Offset - offset face controllers from head controller
Eyes Settings - settings for eyes rig
Eyelids Settings - settings for eyelids rig
Morph Controllers - array of settings for morph controllers

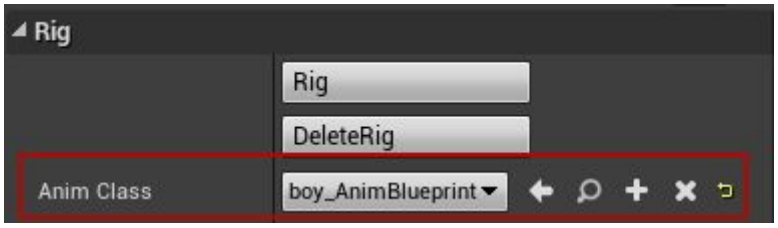
6.1) **Get Ready To Rig Face**

6.1.1) Create **new level** and add **boy_SKELMESH** to the scene.

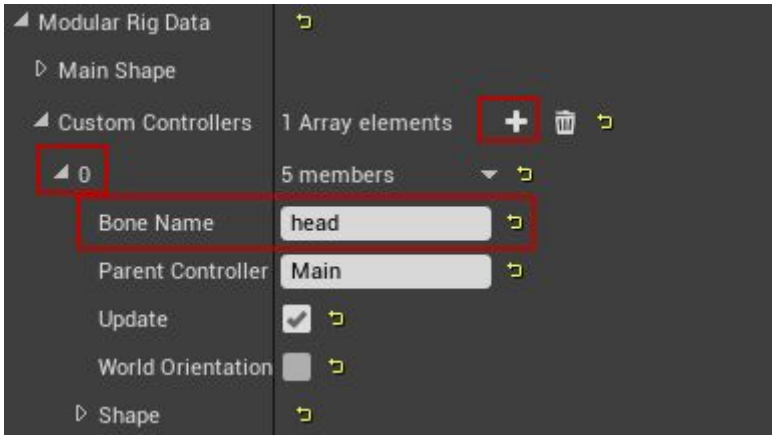


You can find **boy_SKELMESH** asset under **Examples/KiteDemo** folder in Allright Rig content.

6.1.2) Open **new** instance of **Rig Tools** window and set **Anim Class** to **boy_AnimBlueprint**.



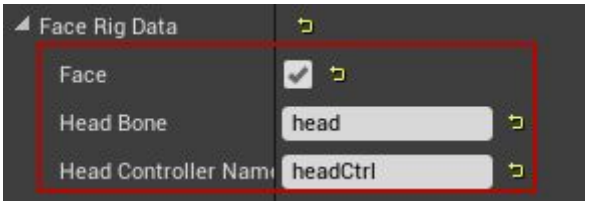
6.1.3) Add new element to **Custom Controllers** array and set **Bone Name** to **head**.



6.1.4) Select **boy_SKELMESH** actor and create rig.

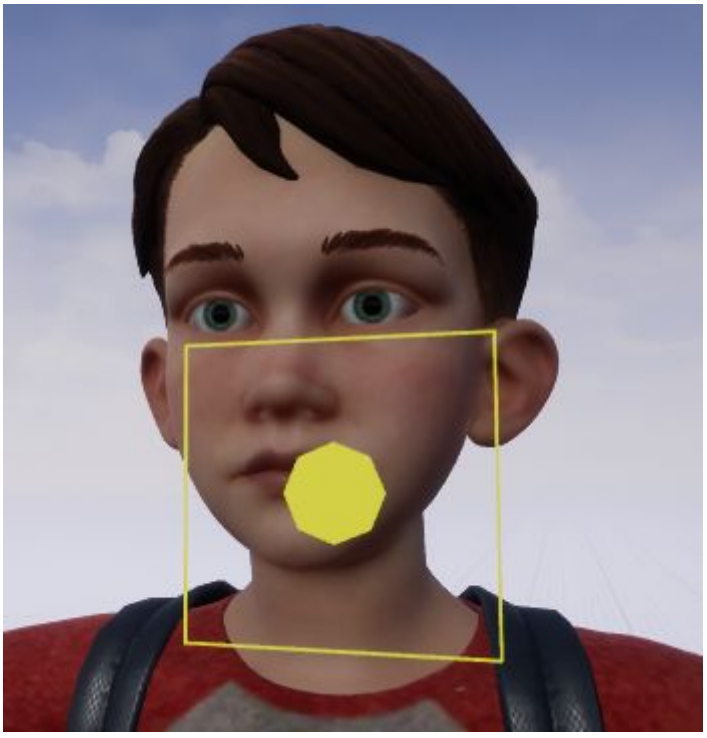
We will use this controller to attach face rig to it.

6.1.5) Under **Face Rig Data** set following properties:



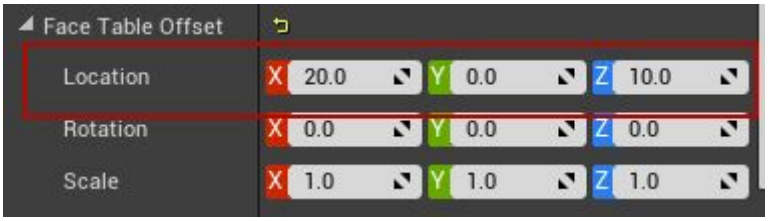
Face - enabled
Head Bone - head
Head Controller - headCtrl

6.1.6) **Recreate rig**.

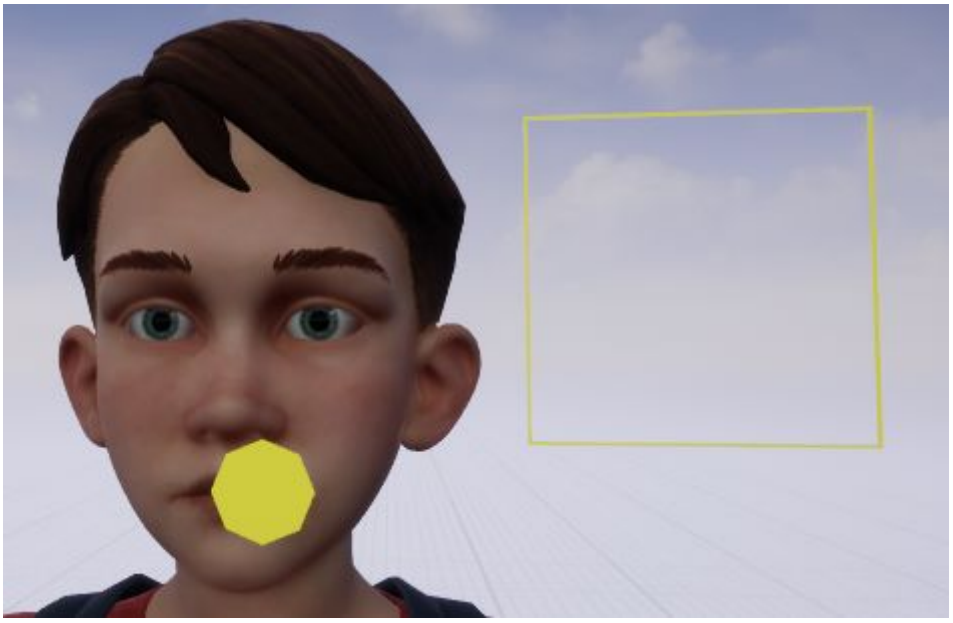


This square represents area for facial controllers. Facial controllers will be attached to it.

6.1.7) Set **Face Table Offset** transform to something like this:



6.1.8) **Recreate rig**.



Yellow square should appear on the right side of the head.

6.2) Eyes rig

Eyes Settings represent settings for eyes rig:

Left Eye Bone - name of left eye bone
Right Eye Bone - name of right eye bone
Mid Color - middle controllers color
Left Color - left controllers color
Right Color - right controllers color

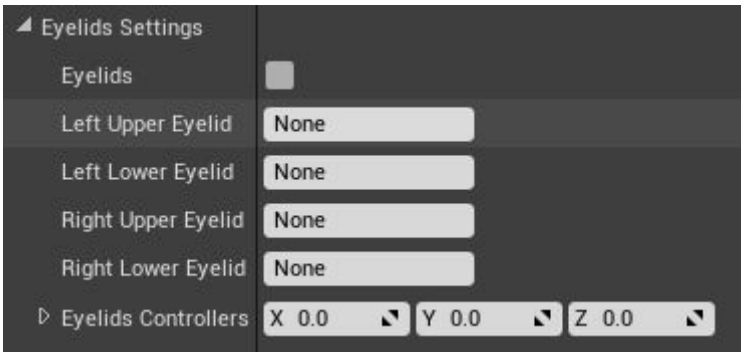
6.2.1) Set **Left Eye Bone** to **eye_left** and **Right Eye Bone** to **eye_right**.

6.2.2) **Recreate rig.**

Use eyes controllers to aim eyes.

6.3) Eyelids rig

Eyelids Settings represent settings for eyes rig:

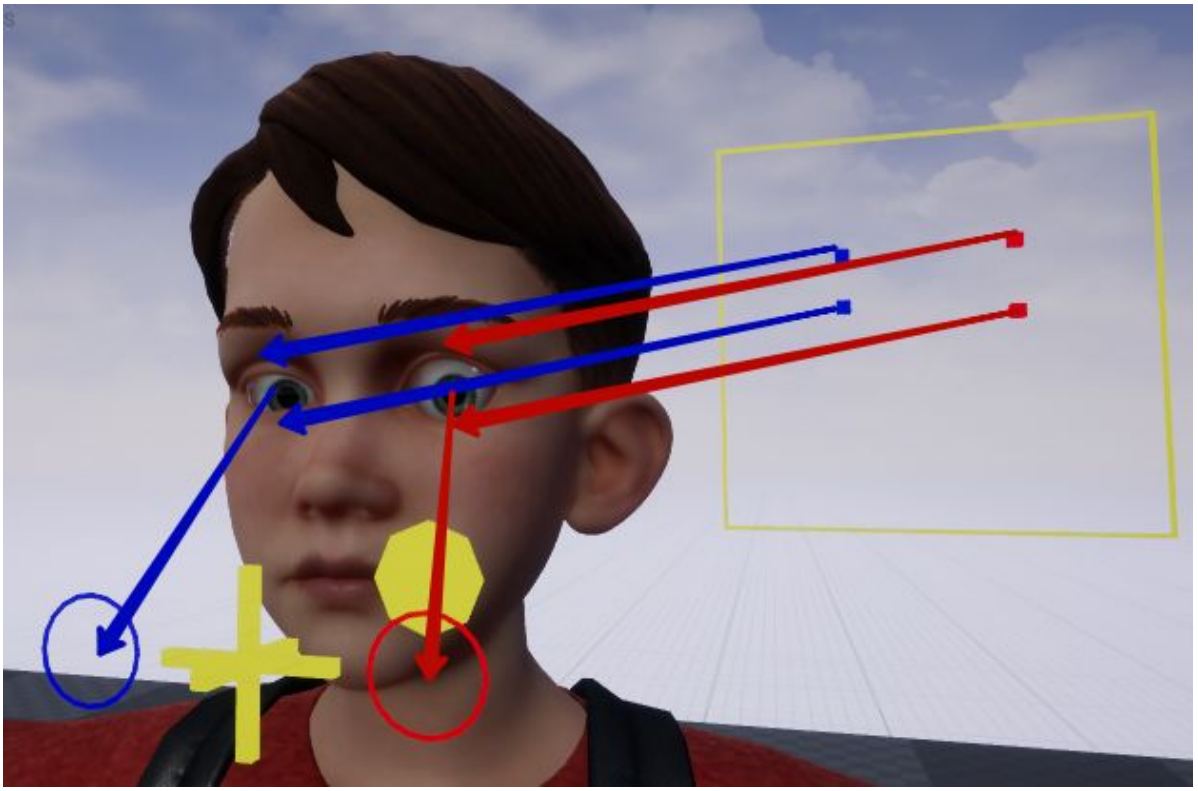


Eyelids - create eyelids controllers.
Left Upper Eyelid - name of left upper eyelid bone.
Left Lower Eyelid - name of left lower eyelid bone.
Right Upper Eyelid - name of right upper eyelid bone.
Right Lower Eyelid - name of right lower eyelid bone.
Eyelids Controllers Position - position for left eyelids controllers. Right controllers position will be mirrored.

6.3.1) Set following **eyelids properties**:

Eyelids = enabled
Left Upper Eyelid = eyelidUpper_left
Left Lower Eyelid = eyelidLower_left
Right Upper Eyelid = eyelidUpper_right
Right Lower Eyelid = eyelidLower_right
Eyelids Controllers Position = 3, 0, 2

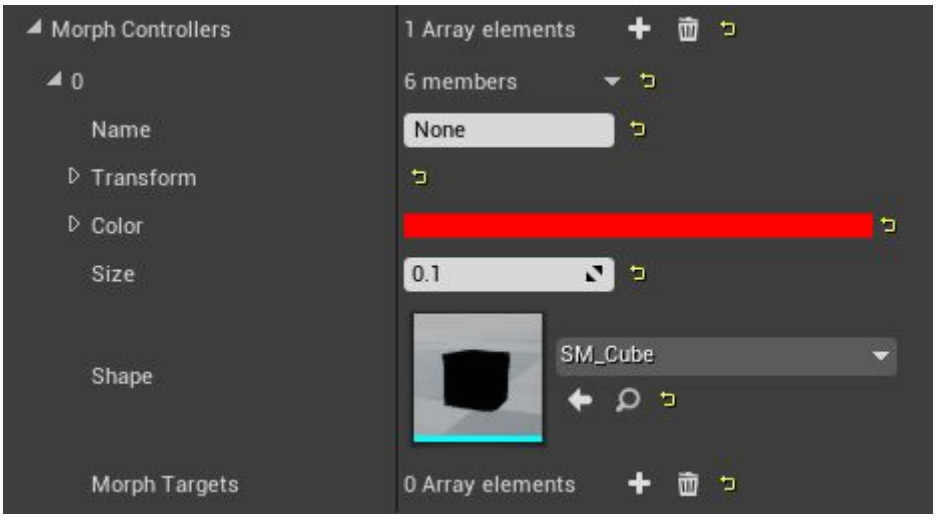
6.3.2) **Recreate rig.**



Eyelids are **half driven** by rotation of **eyes** and **full driven** by eyelids **controllers**.

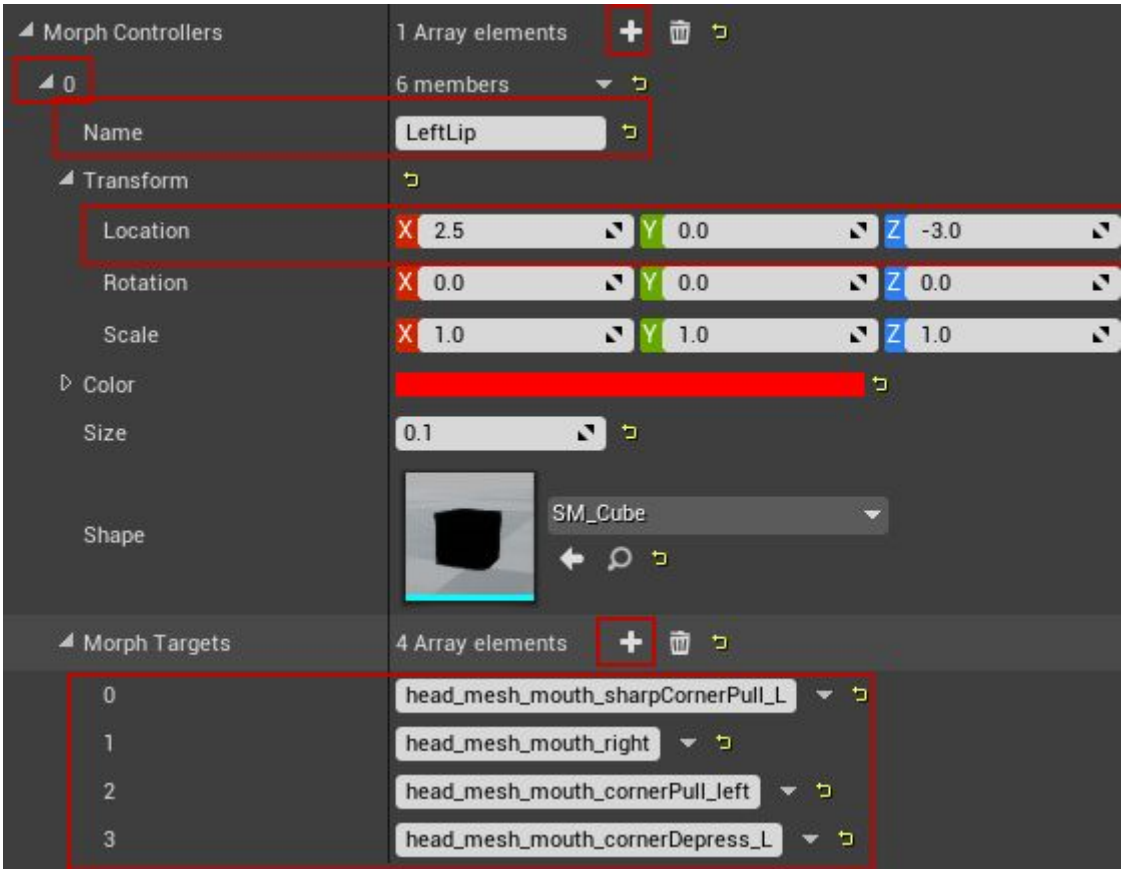
6.4) Morph controllers

Morph Controllers is array of settings for morph controllers.



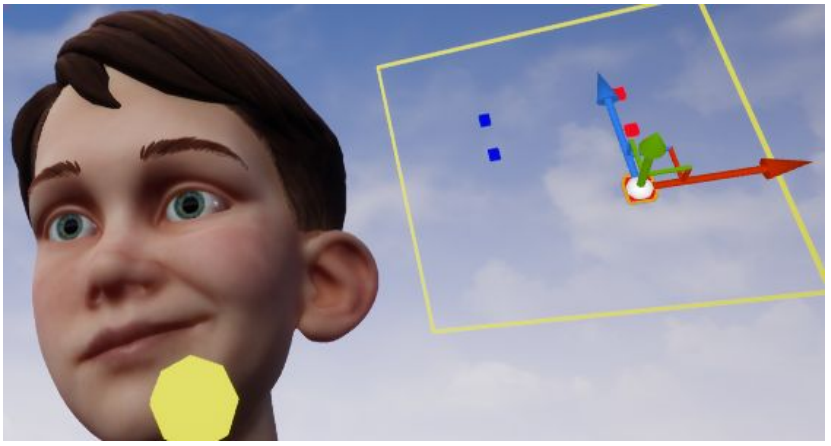
Name - controller name for comfort. Does not effect on anything.
Transform - local controller position in face table.
Color - controller color.
Size - controller size.
Shape - controller shape.
MorphTargets - array of morph targets to use. You can move morph controller on X and Z axis. So here you need to specify what morph targets (0, 1, 2, 3) will your controller activate when you move it on the +X, -X, +Z or -Z axis. You can also add 4, 5, 6, 7 morph names if there few geometry shapes that share one morph deformation. For example face geometry and eyebrows geometry.

6.4.1) Add **new element** to **Morph Controllers** array and set following parameters:

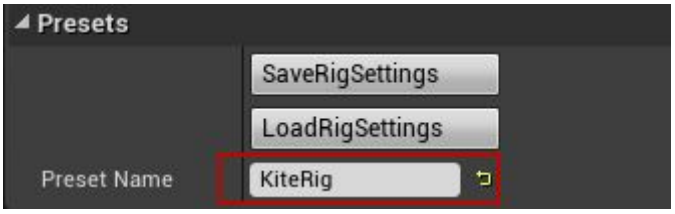


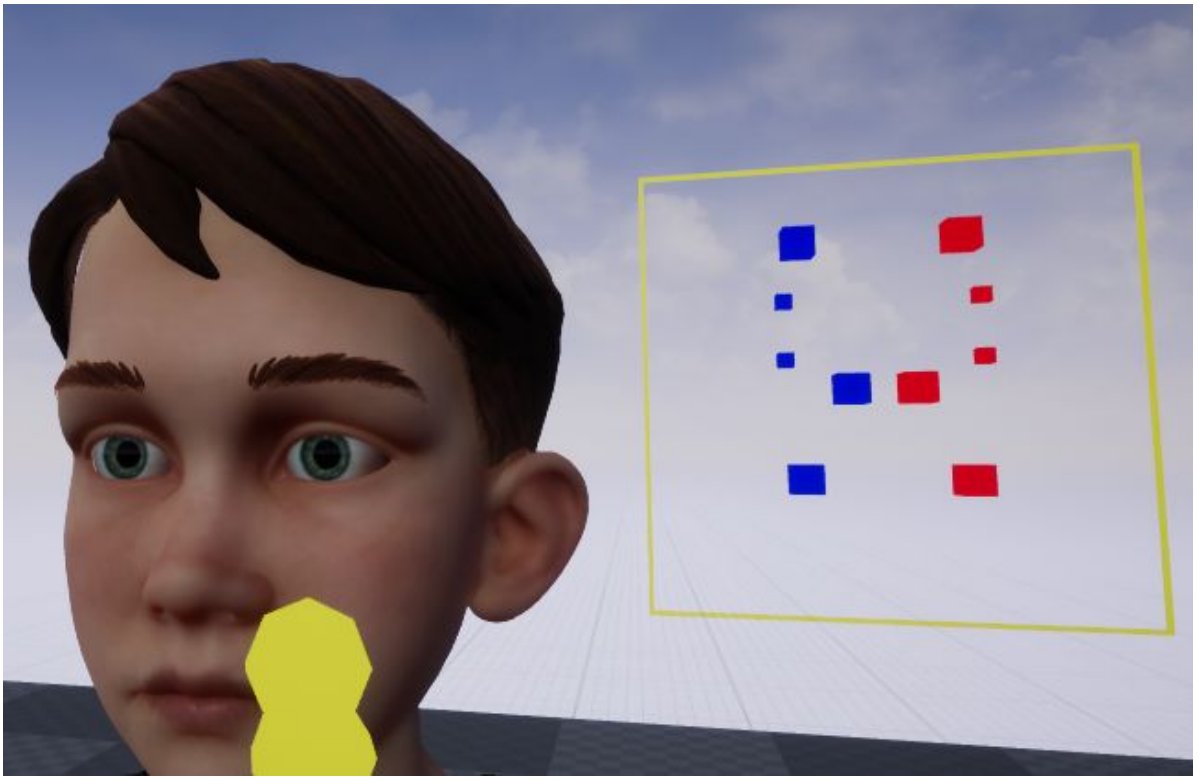
Name = LeftLip
Transform.Location = 2.5, 0, -3
Morph Targets:
0 = head_mesh_mouth_sharpCornerPull_L
1 = head_mesh_mouth_right
2 = head_mesh_mouth_cornerPull_left
3 = head_mesh_mouth_cornerDepress_L

6.4.2) **Recreate rig.**



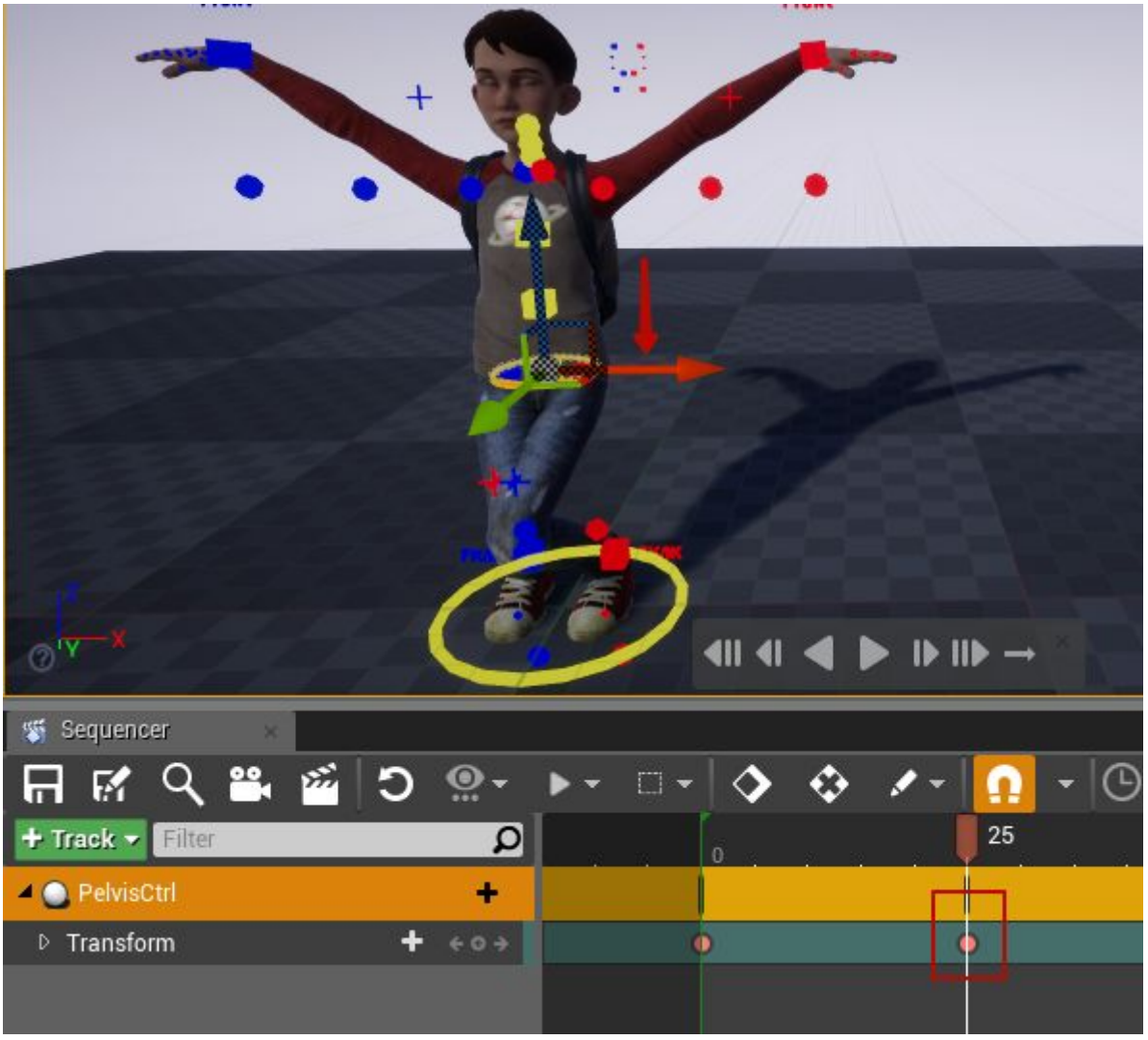
6.5) Load full **KiteRig** preset for reference.





Working with animation

To create and play animation you can use **Level Sequencer.**

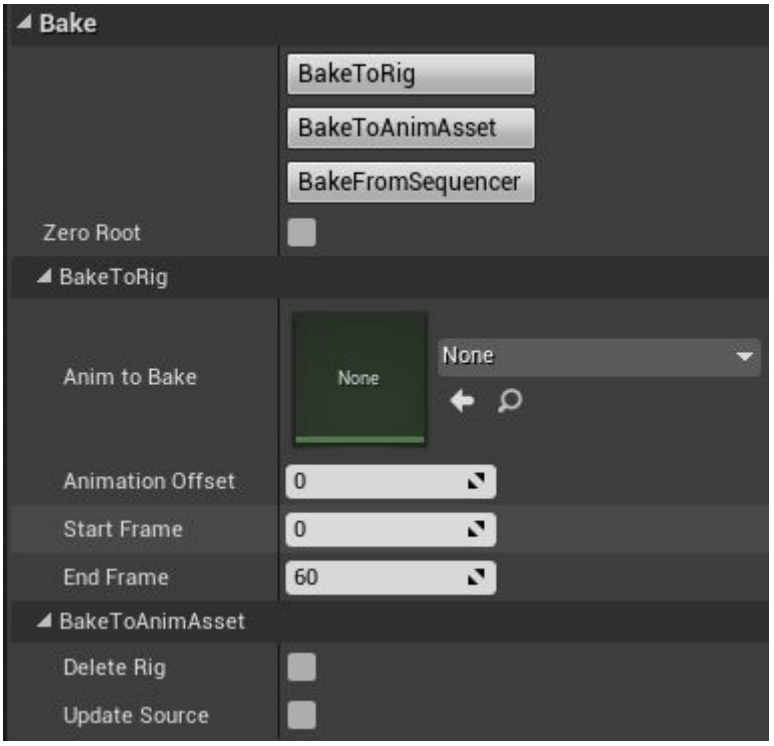


On how to work with sequencer please visit Sequencer Editor page in Unreal Engine documentation.

During deletion of the rig sequencer animation will also be deleted.

Baking animation

Allright rig allows you to **bake** animation from an **animation sequence** to the **rig** and **vise versa**. You can find baking settings under the **Bake** category in **Rig Tools** window.



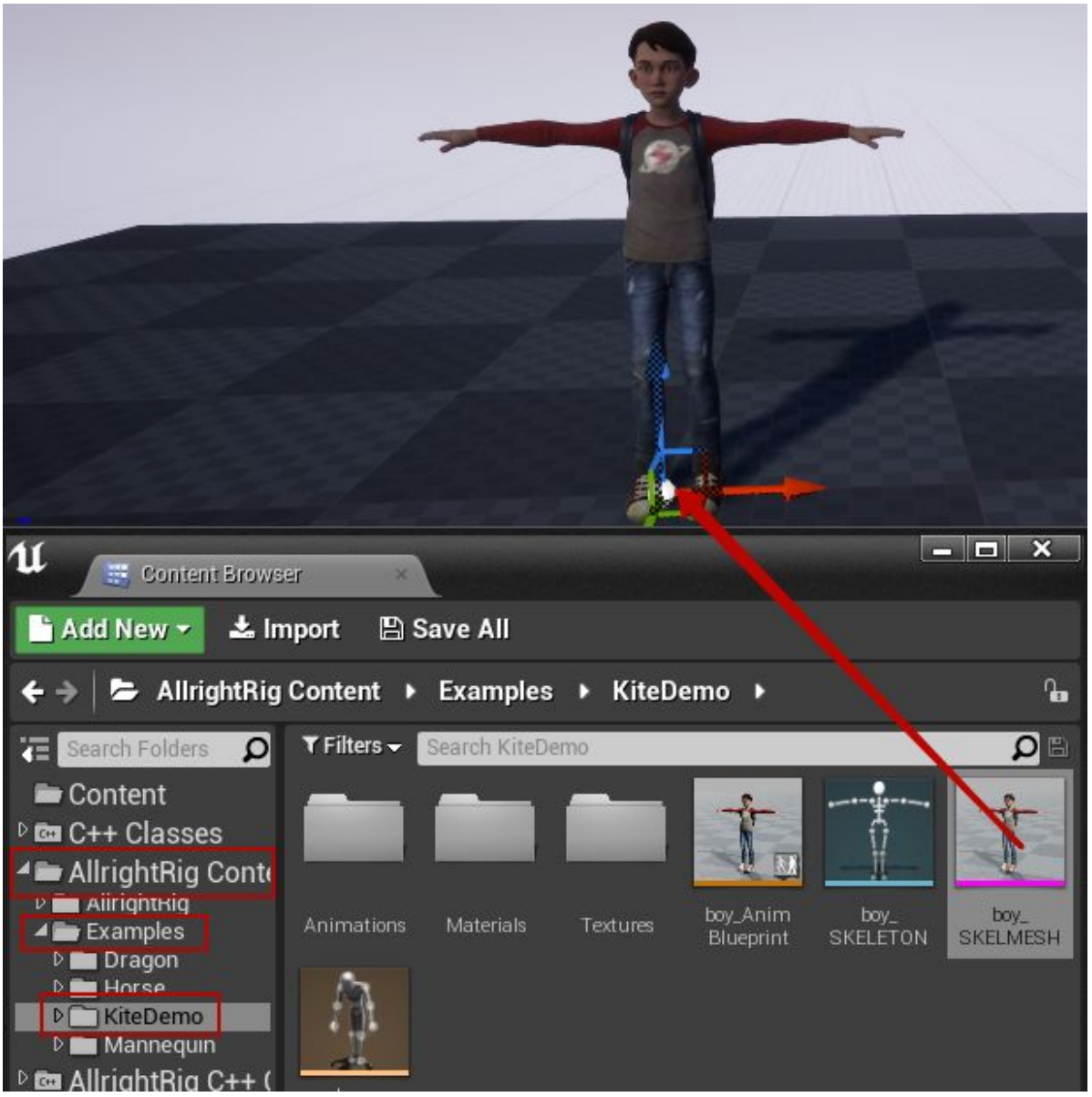
In order to work with bake settings please make sure that you are able to create rig for your character.

Baking system always uses sequencer frame fate.

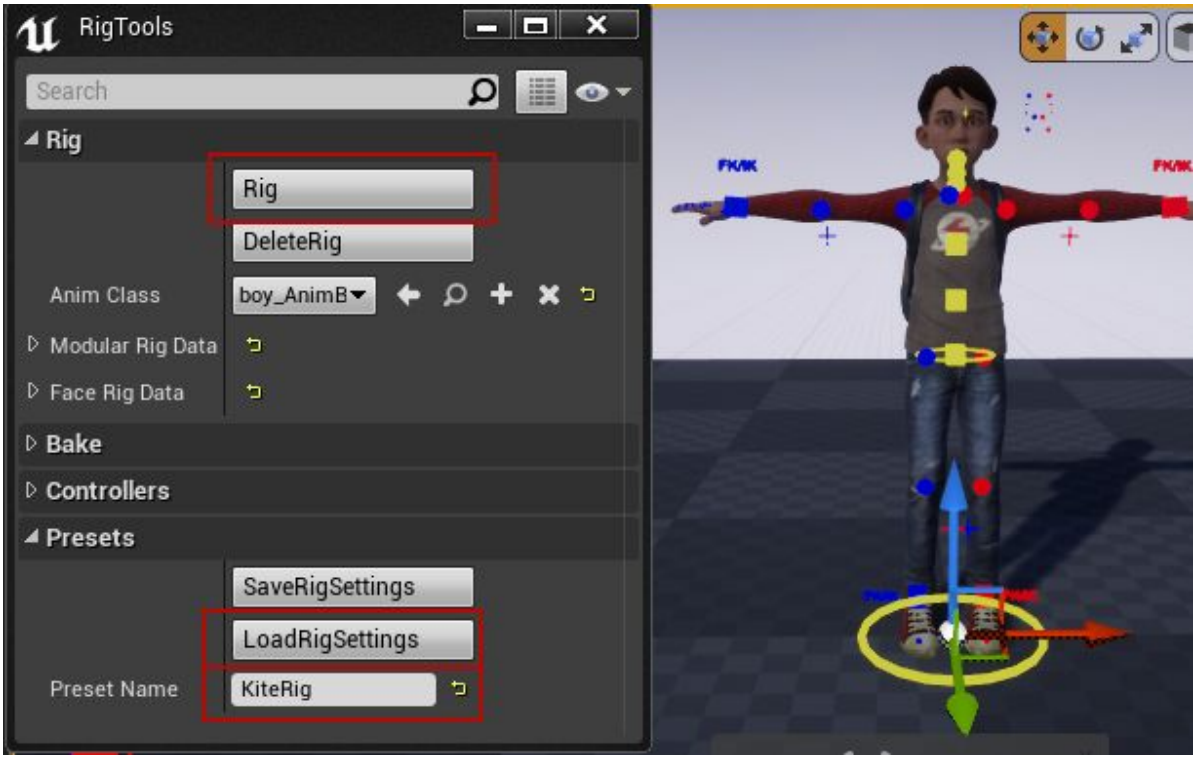
1) Baking animation from rig to an animation sequence

You can **play** final animation using the rig, but of course it is better to save it to an **animation sequence**.

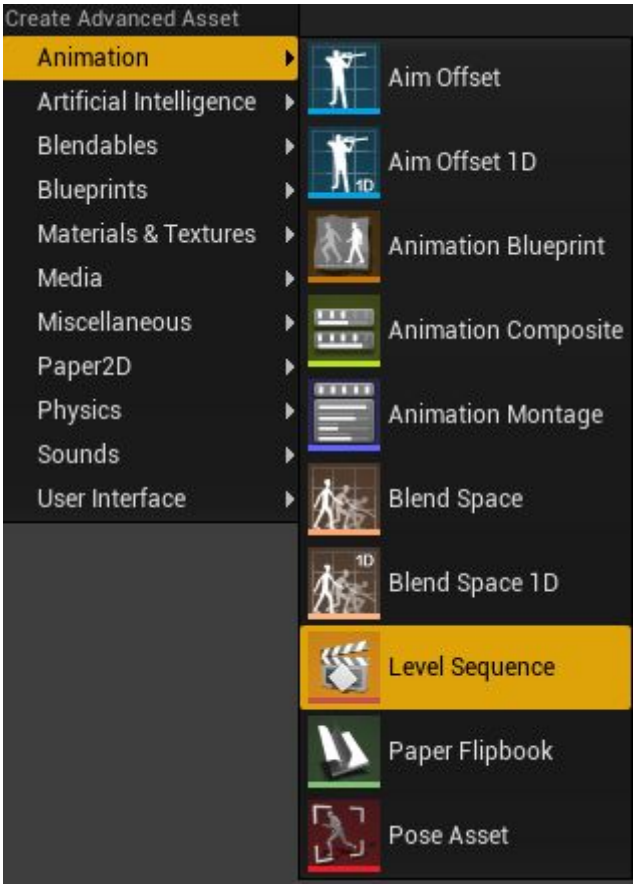
1.1) Create **new level** and add **boy_SKELMESH** to the scene.



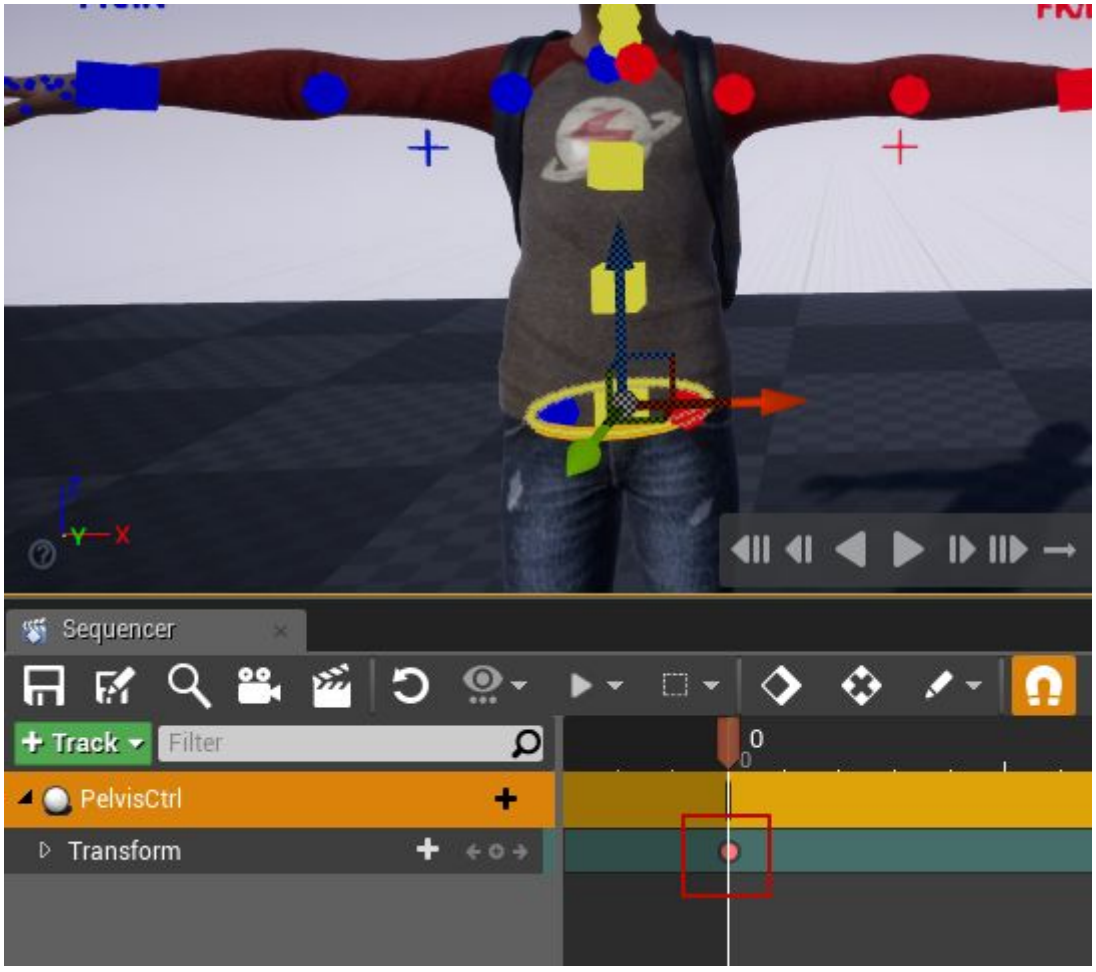
1.2) **Create rig** using **KiteRig** preset.



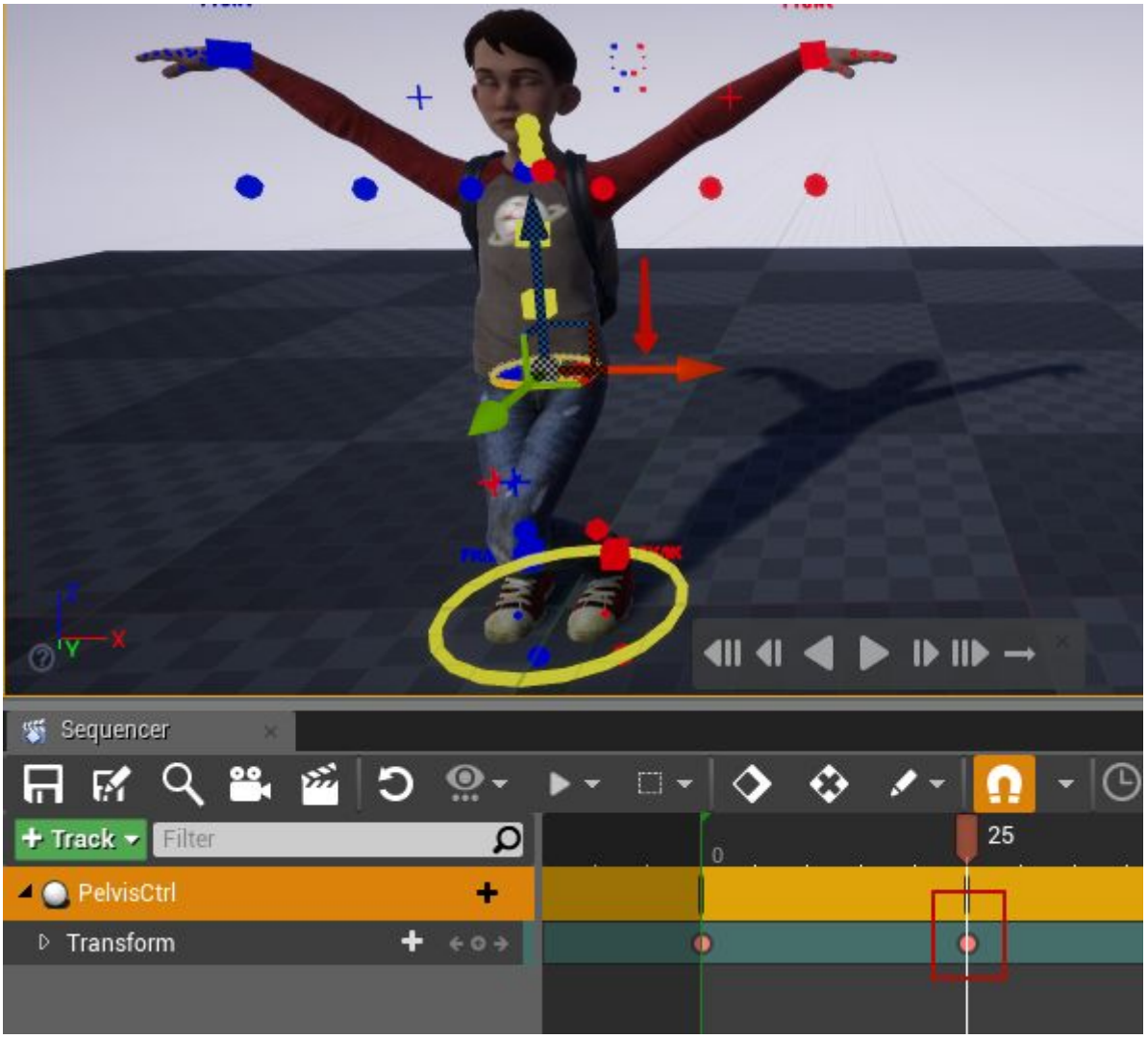
1.3) Create **new level sequence**



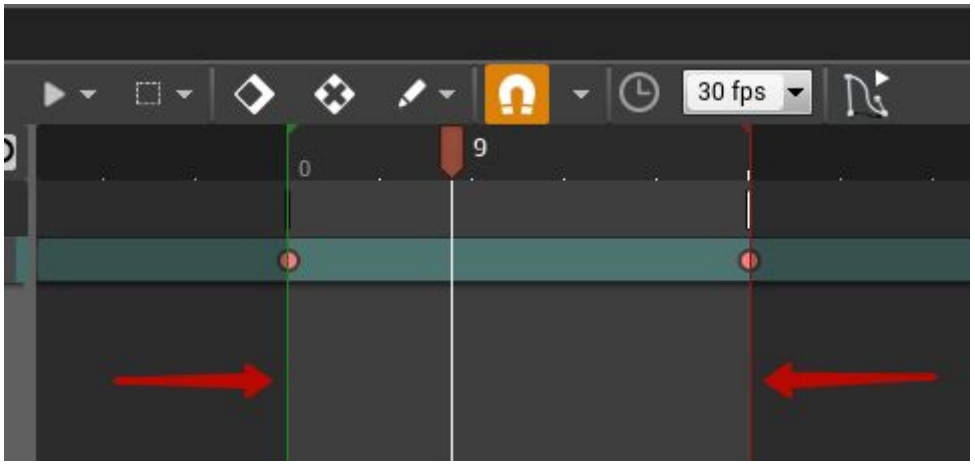
1.4) Open **level sequence**, **select any controller** and press **“s”** button.



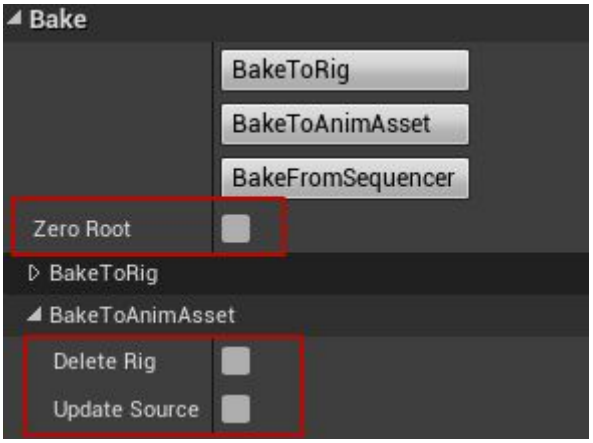
1.5) In **sequencer** go to any other **frame**, **move** controller and press **“s”** button again.



1.4) Set sequencer **playback range** to current animation **length**.

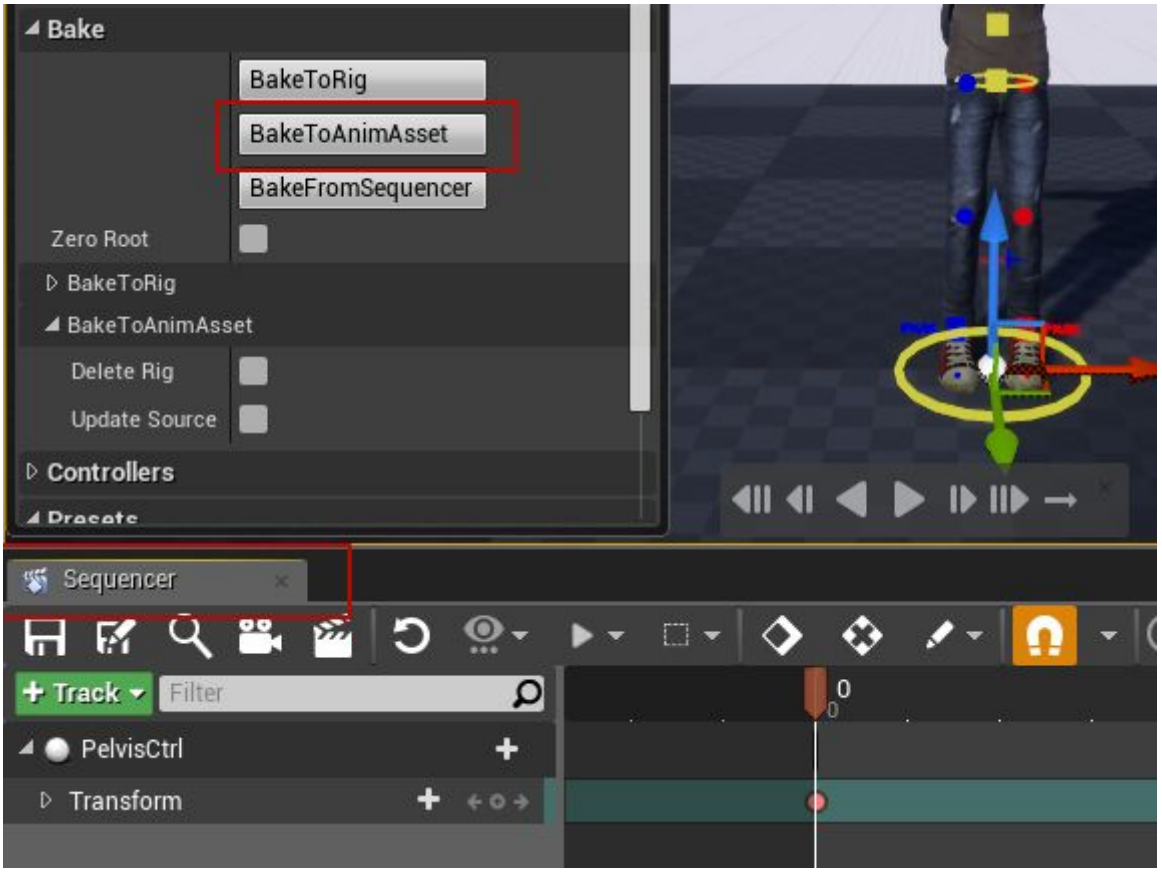


1.5) Under **Bake** category choose:

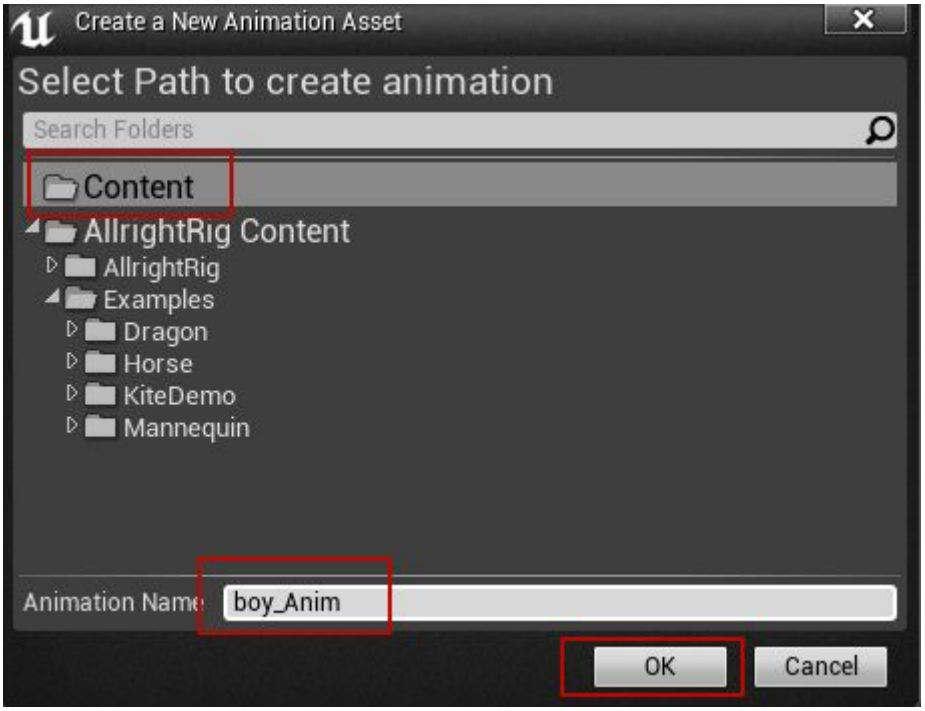


Zero Root - if you want actor pivot to be at origin during the baking process.
Delete Rig - if you want to delete current rig after baking.
Update Source - if you want to **add** or **update** source skeletal mesh animation in level sequence.

1.6) **Select any rig controller,** make sure that current **sequencer window is opened** and press **BakeToAnimAsset** button.



1.7) Select **path** for new animation sequence, type **animation name** and press **OK** button.



New animation sequence should appear under selected folder.

2) Baking animation from animation sequence to the rig.

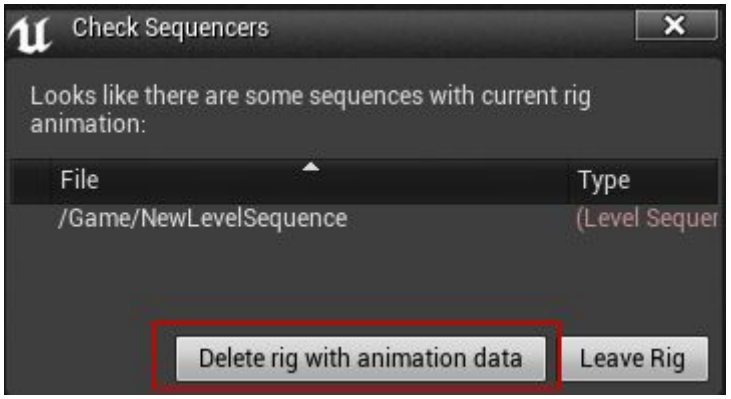
Also, you can bake animation from animation sequence to the rig.

2.1) Bake from sequencer

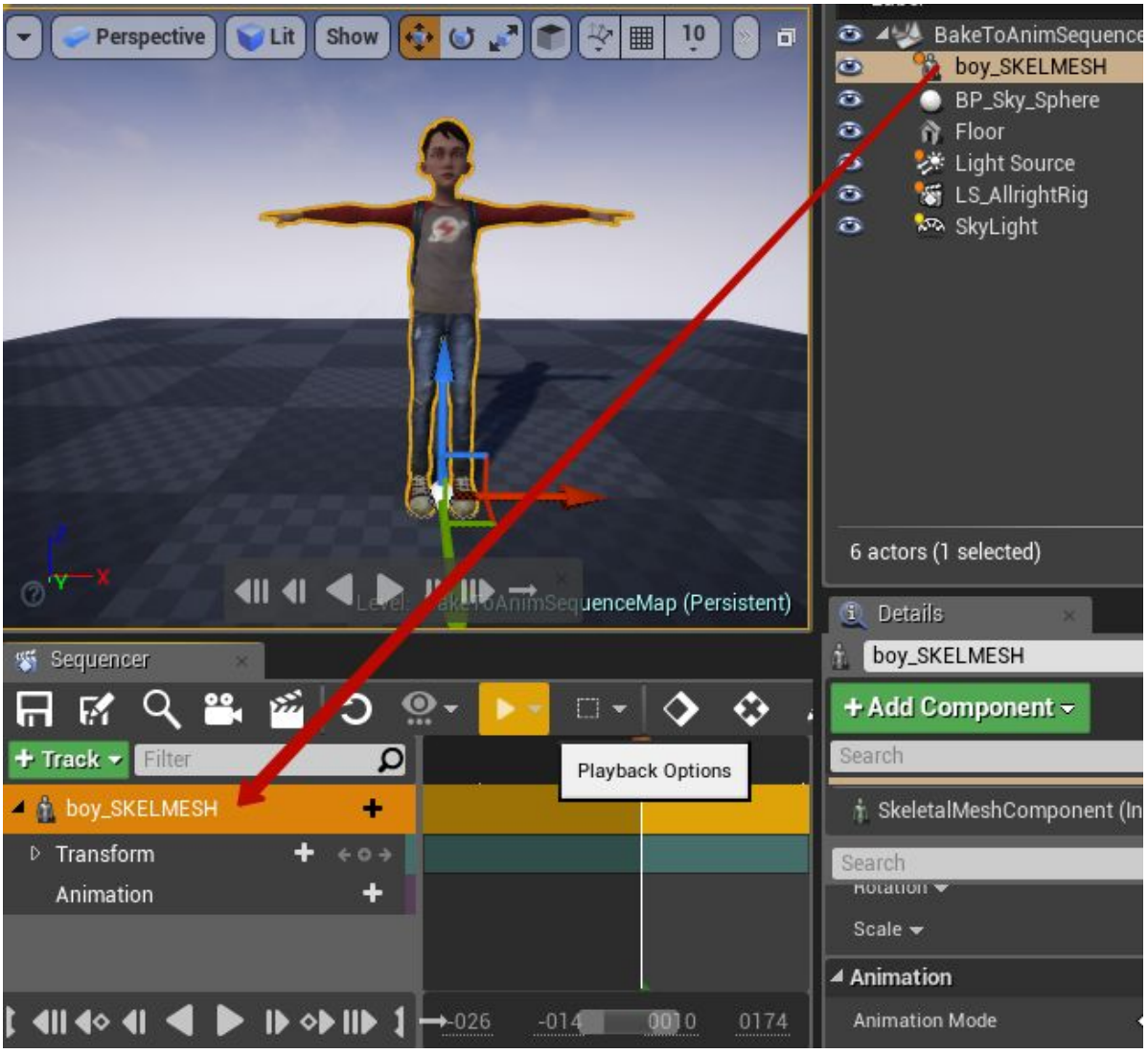
Bake blended skeletal animation sections from sequencer right into the rig

2.1.1) **Select** any controller and press **DeleteRig** button.

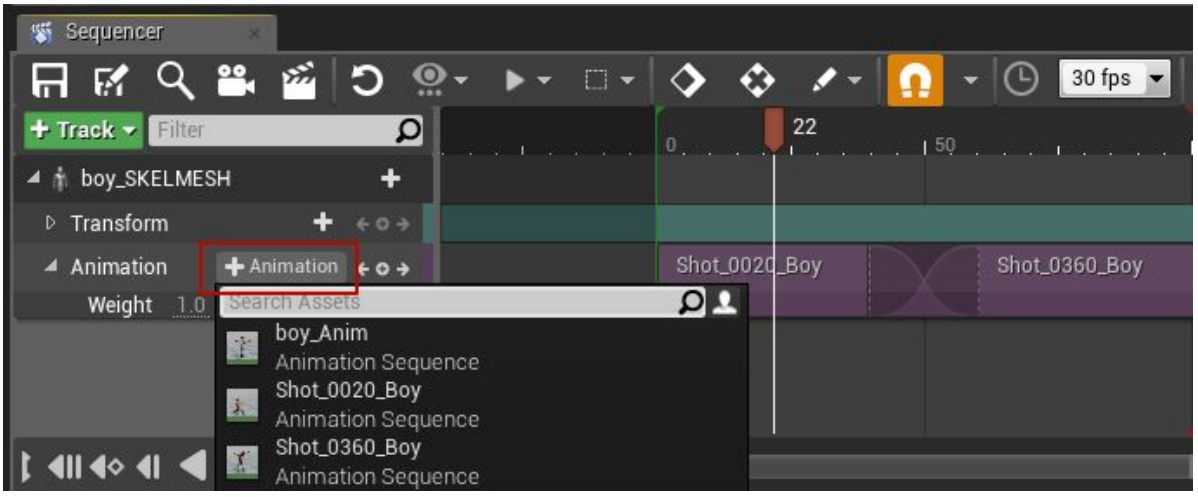
2.1.2) If **current rig** has **any animation** in any **level sequence**, “**Check Sequencers”** window should appear. Choose **Delete rig with animation data**.



2.1.3) Add **boy_SKELMESH actor** to the **level sequence**.

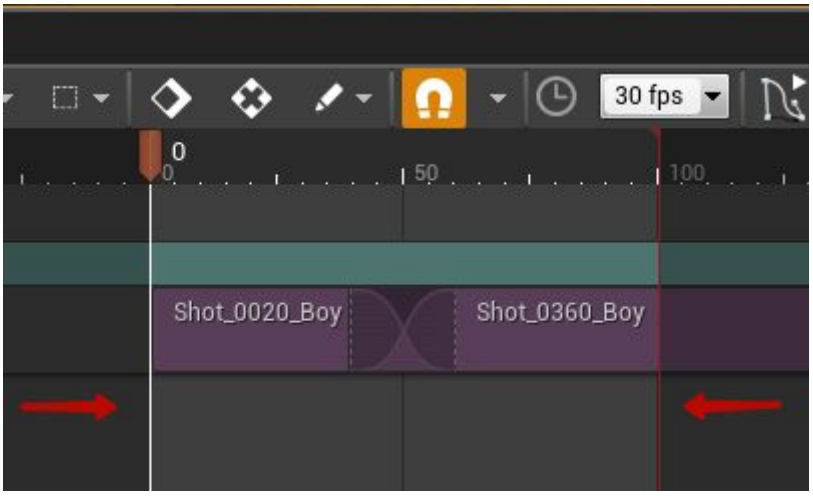


2.1.4) Add **one** or **few** animation sequences to boy_SKELMESH **animation track**.

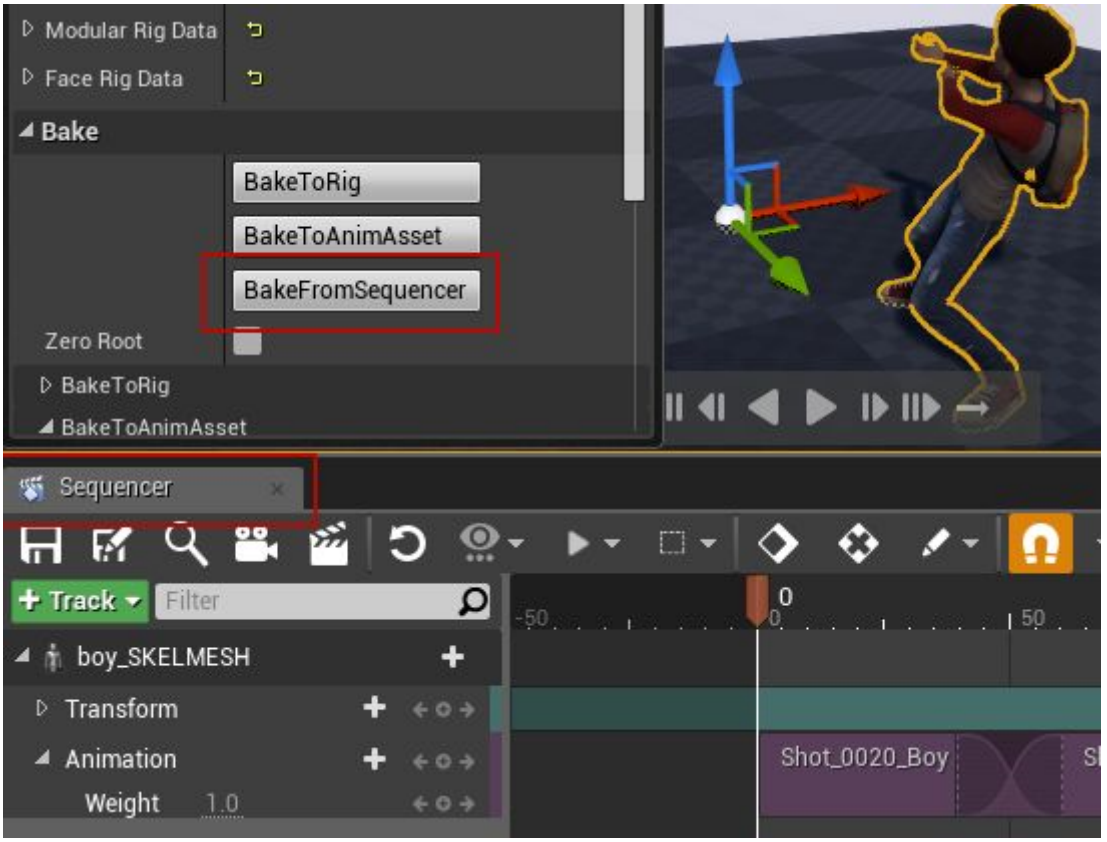


On how to work with animation tracks please visit unreal engine Animation & Animation Blending documentation page.

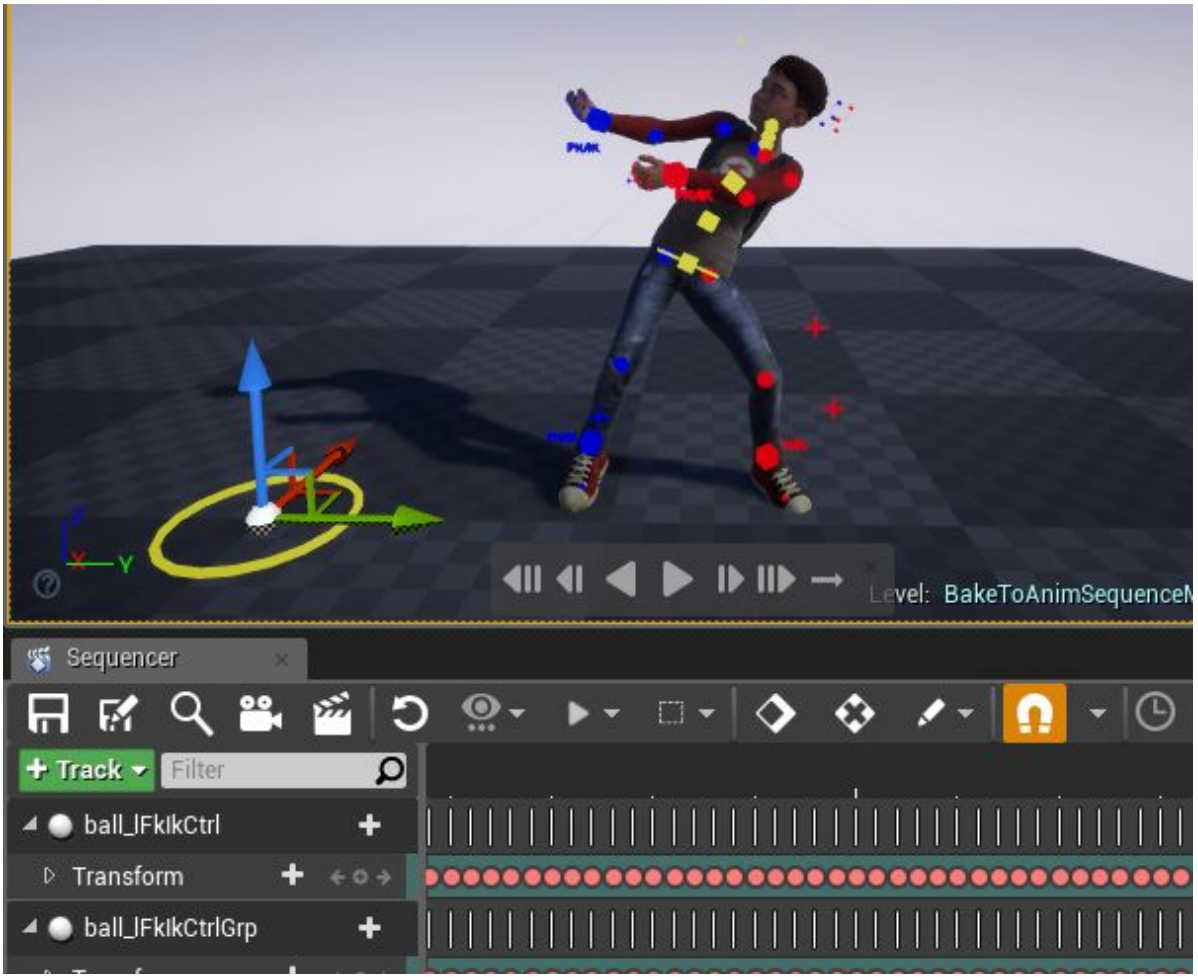
2.1.5) Set sequencer **playback range** to work with.



2.1.5) Select **boy_SKELMESH** actor**,** make sure that current **sequencer window is opened** and press **BakeFromSequencer** button.



In a few seconds skeletal mesh animation should be baked to the rig.

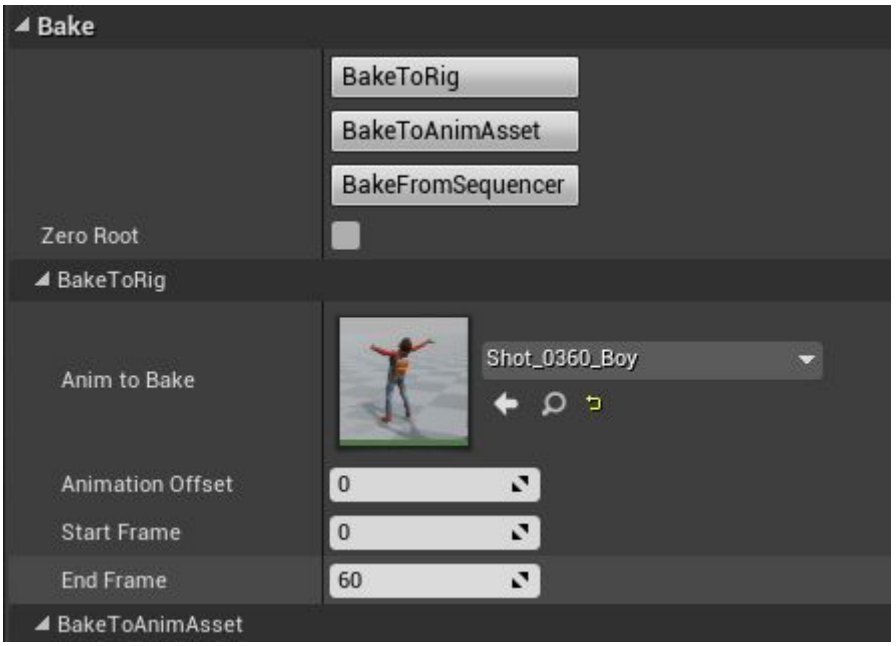


Now you can edit animation, bake it to an animation sequence and vise versa.

***Absolute, additive** and **relative** sections are **not** available for baking yet. However this feature will be added in future versions.*

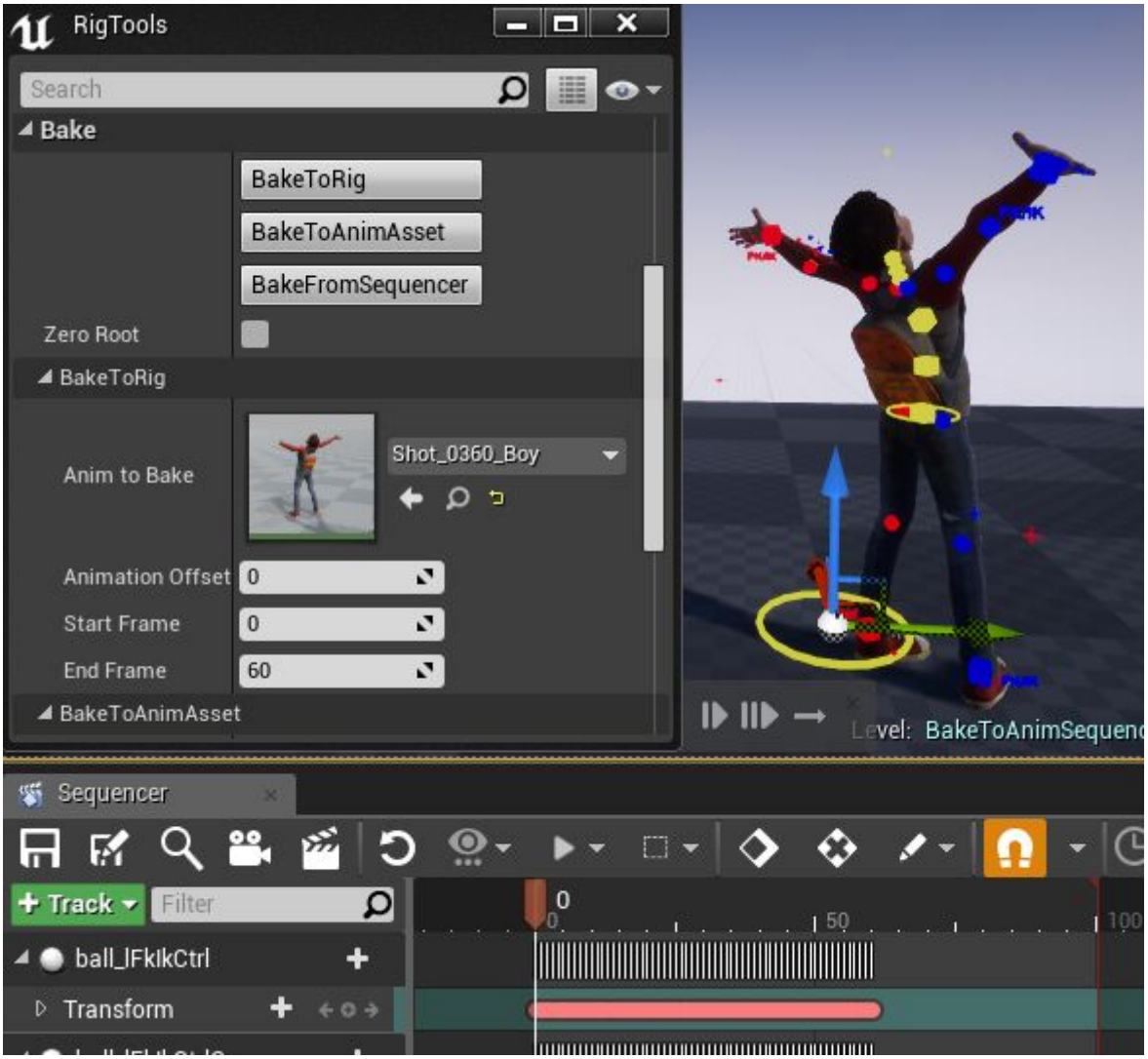
2.2) Bake to rig

Usually it is more comfortable to work with **BakeFromSequencer** command, however you can also bake animation to the rig using properties under **BakeToRig** subcategory.



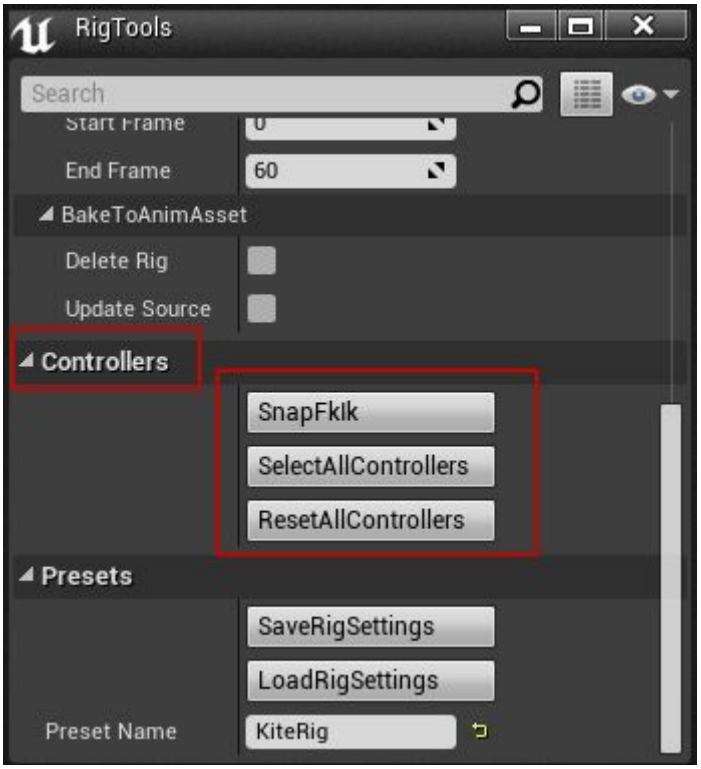
Anim to Bake - anim asset to bake from. If **AnimToBake = nullptr** - animation will be taken from skeletal mesh actor **Anim To Play** property.
Animation Offset - anim sequence start frame offset.
Start Frame - start frame.
End Frame - end frame.

2.2.1) Set **Anim To Bake, Animation Offset, Start Frame, End Frame** properties. **Open level sequence** editor and **select rig controller** or **skeletal mesh actor** to work with.



Controllers category

Under **Controllers** category in **RigTools** window you can find following commands:

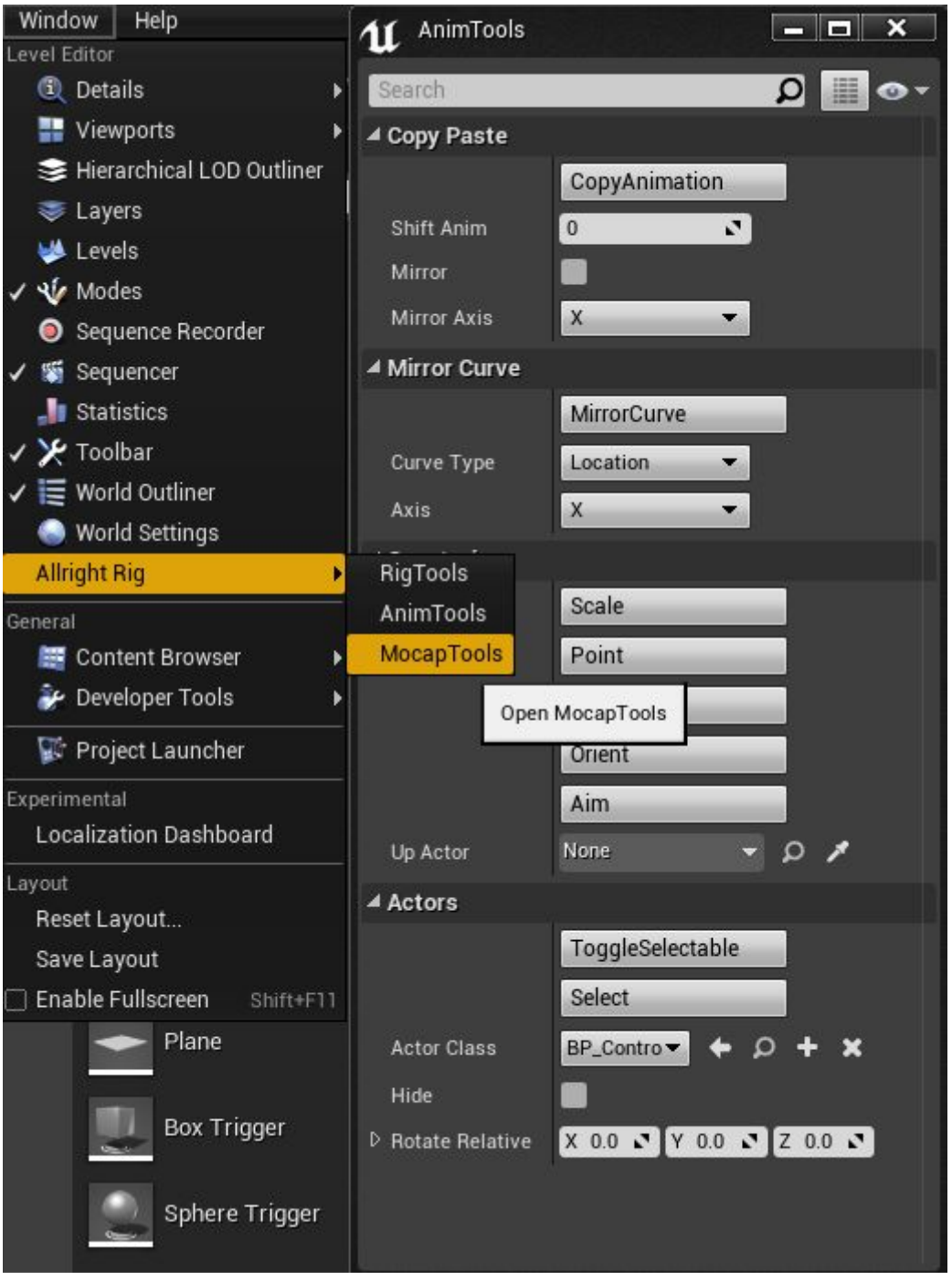


SnapFkIk - snaps all fk and ik controllers to current bone positions.
SelectAllControllers - selects all rig controllers.
ResetAllControllers - reset all controllers positions.

In order to work with this commands please **select any rig controller**.

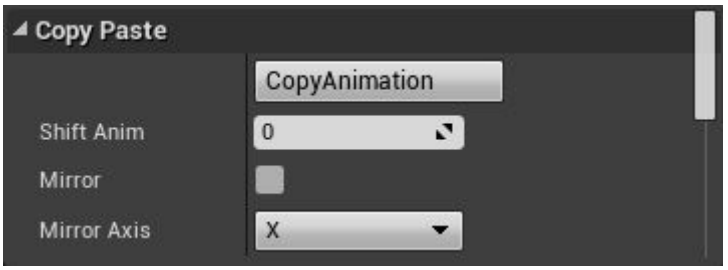
Animation tools

Under **Allright Rig menu** you can find **AnimTools** window.



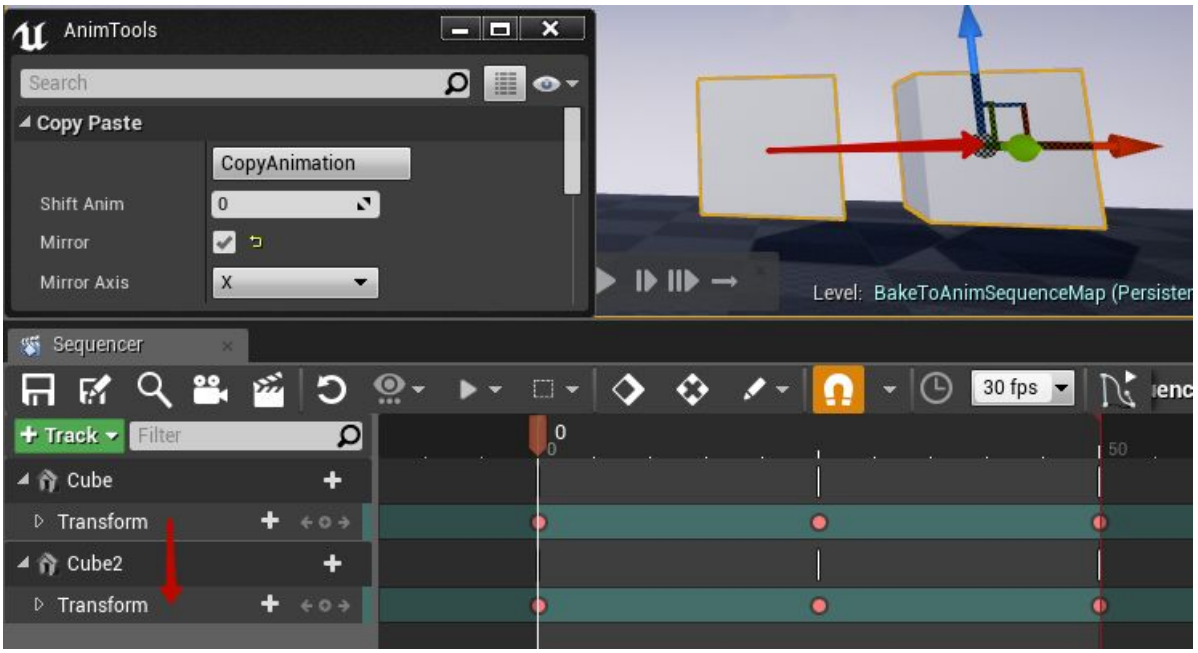
Various animation tools were collected in this window to make animation creation process esear. This tools **does not depend on rigging system**. Animation tools were created to work with **actors** and **level sequencer**.

1) Copy Paste

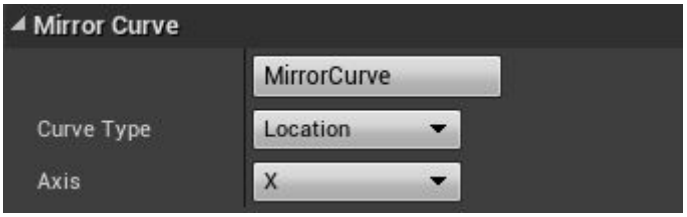


CopyAnimation command allows to copy sequencer transform track animation from one actor to another.
Shift Anim - amount of frames to shift pasted animation.
Mirror - mirror pasted animation.
Mirror Axis - mirror axis.

1.1) Open **level sequence editor**, select **source** actor, **target** actor and press **CopyAnimation** button.

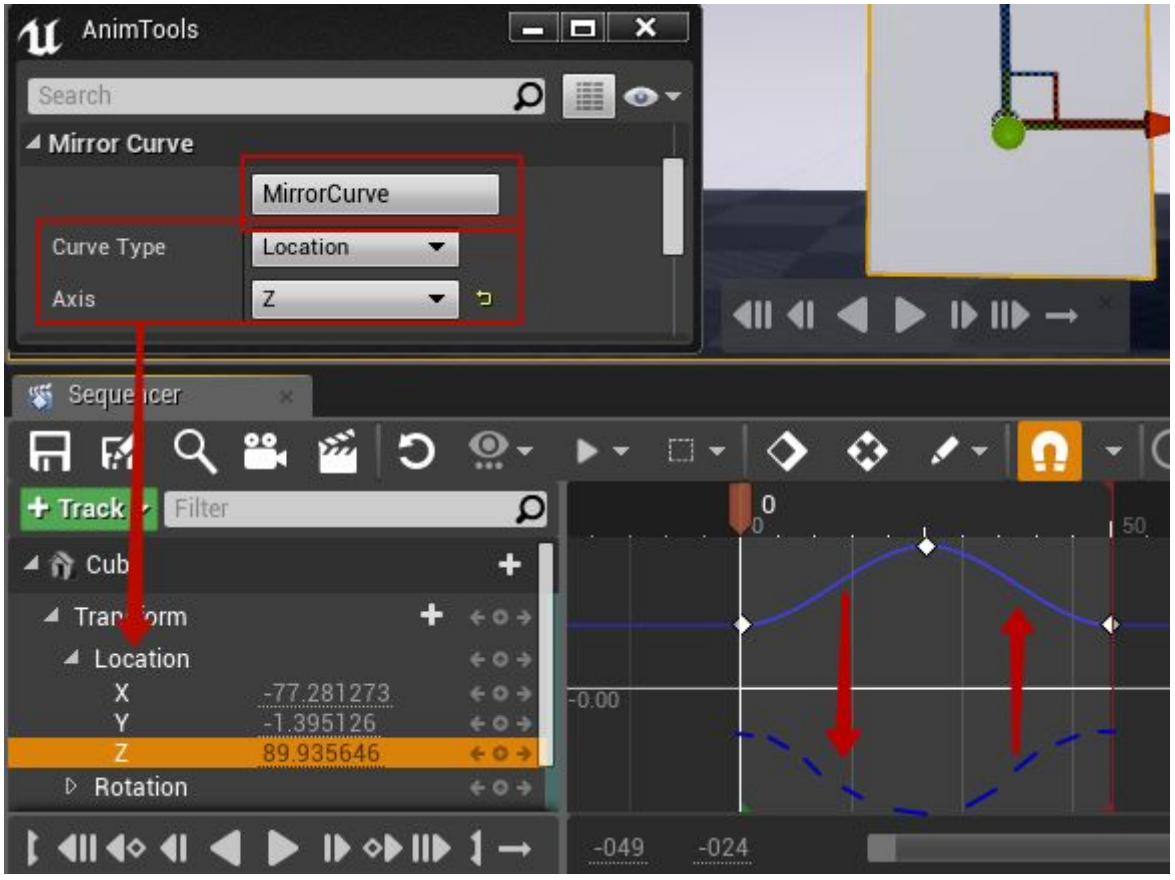


2) Mirror Curve



MirrorCurve command allows to **mirror transform section animation curve**.
Curve Type - animation curve type.
Axis - curve axis.

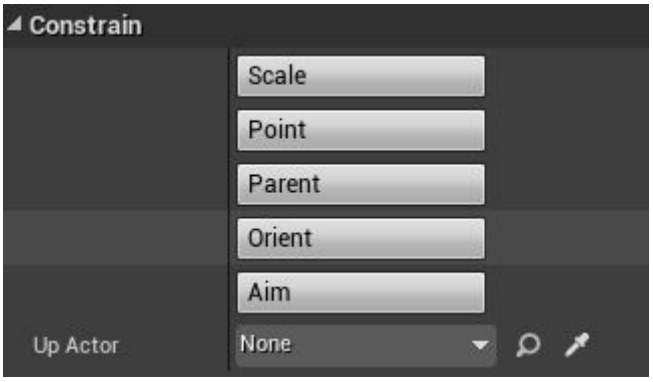
2.1) Open **level sequence editor**, **select actor**, set **Curve Type, Axis** and press **MirrorCurve** button.



In future there will be ability to mirror any type of animation curves.

3) Constrain

Constraints let you constrain the position, orientation, or scale of one actor to other actors.



Point constraint causes an actor to move to and follow the position of an actor, or the average position of several actors.
Orient constraint matches the orientation of one actor to one or more other actors.
Scale constraint matches the scaling of one actor to one or more other actors.
Parent constraint lets you to relate the position and rotation of one actor to another, so that they behave as if part of a parent-child relationship that has multiple target parents.
Aim constraint constrains an actor's orientation so that the actor aims at other actors.

Main concept of working with animation constraints:

1) **Select targets** and **last select** actor to apply constraint to.
2) Use **W1/2..** properties to specify weight of each target. Right now you can not use more than 5 targets, but there will be no limitations in future.
3) **Offset** vector sets offset from resulting position.
4) **Rest Position** property is used to specify position of actor when **all weights are set to 0.**

4) Actors

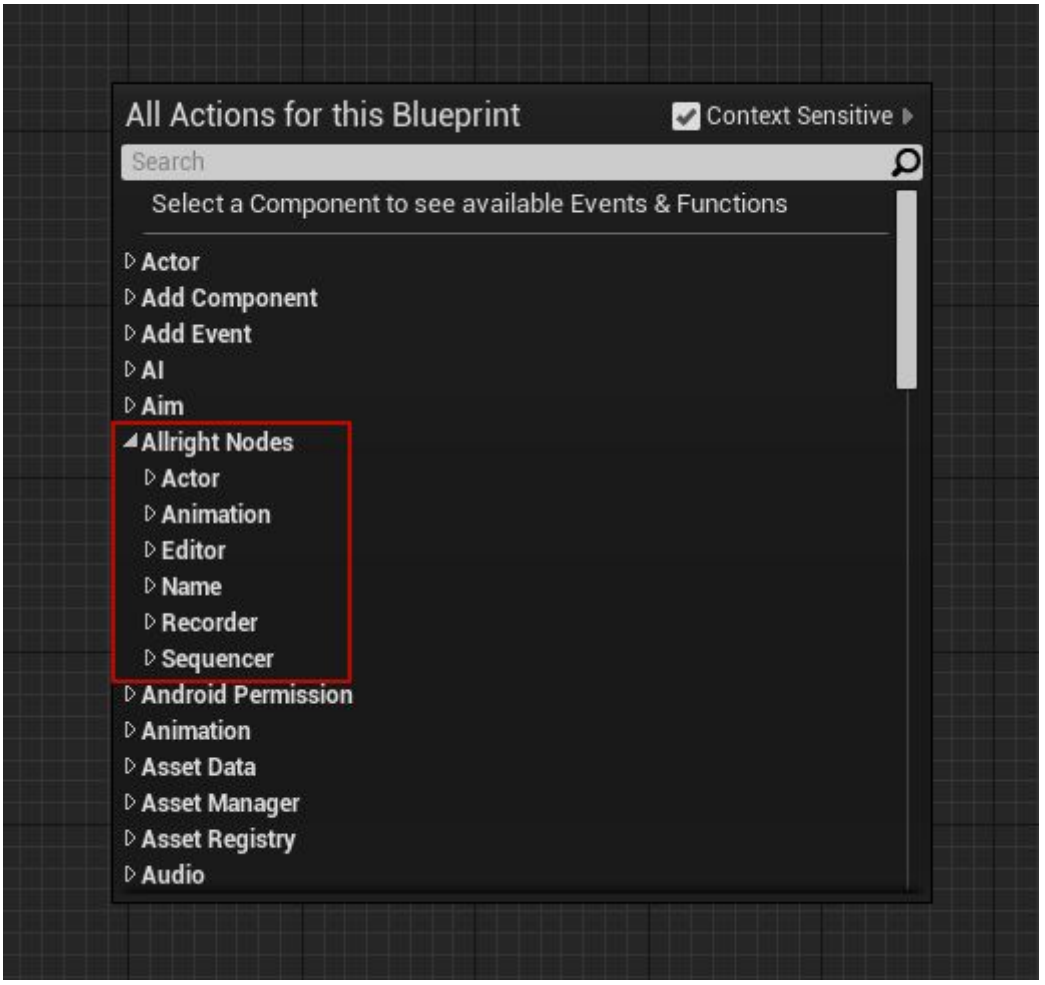
Mocap tools

1) Butterworth

2) Peak remover

Blueprint library

Allright Rig plugin comes with a library of blueprint nodes, which allows to create rigs and work with character animation in editor. You can find them under **Allright Nodes** category in blueprint actions list.



All this nodes can be used with **any blueprint classes**, so they might be helpful in some other cases.

Actor

Parent - is a stripped-down version of AttachActorToActor node which attaches one actor to another
RenameActor - renames actor
SelectActorInEditor - selects actor in editor
PostEditChange - evaluates construction script
SetSelectable - sets component selectable

Name

AppendName - appends to name
MatchesWildcard - checks if a name matches other name
StringsToNames - convert array of strings to array of names
AddToArrayOfNames - add name to array of names and return array

Sequencer

IsSequencerHasBindingForActor - checks if a sequencer has binding for an actor
GetTransformSectionFromBinding - gets transform section from actor binding
GetFloatSectionFromBinding - gets float section from actor binding
SetTransformSectionKey - sets or updates a key on this transform section
SetTransformKey - sets or updates a transform section key for actor
SetSectionStartTime - sets a new start time for this section
SetSectionEndTime - sets a new end time for this section
GetSequencerComponents - gets an array of sequencer components and bindings
UpdateSequncerComponent - updates the sequencer components location in a specific time
UpdateSequncer - updates sequencer
UpdateSequencerActorTransform - updates actor transform in a specific time
AddBindingToSequencer - adds actors binding to the sequencer
AddTransformTrack - adds a transform track to the actors binding
AddAnimTrack - adds animation track to the actors binding
RemoveTrackFromBinding - removes track from the actors binding
RemoveBindingFromSequencer - deletes binding from sequencer
GetTransformSectionFromActor - gets sequencer transform section from actor
GetAnimSectionsFromActor - gets sequencer skeletal animation sections from actor
CopyActorAnimation - copy sequencer animation for actor
MirrorTransformSectionAnim - mirrors transform section animation
MirrorTransformSectionCurve - mirrors transform section curve animtion
ShiftTransformSectionAnim - shifts sequencer animation for an actor
CopyTransformSectionCurve - copyes transform section curve
GetSequencerFrameRate - gets sequencer frame rate
UpdateFloatSection - updates sequencer float section
UpdateSkeletalMeshSection - updates sequencer skeletal mesh section
PreviewSetAnimPosition - updates skeletal mesh anim instance
GetSequencerPlaybackRange - gets sequencer playback range

Recorder

AddRawKeyFromPose - adds a raw transform key to an animation sequence
CreateAnimSequence - creates a new animation sequence for skeletal mesh component
PostProcessAnimSequence - applies PostProcessSequence to AnimSequence
GetSpaceBases - gets an array of bone transformations from skinned mesh component
AddRawMorphTargetKey - adds raw morph target key into anim sequence

Editor

NotificationBox - shows message in the corner of screen
OpenEditorForAsset - opens editor for asset
CloseAllEditorsForAsset - closes all editors for asset
PromtDialogue - shows message with list and possibility to select
GetAssetsByClass - gets an array of all objects by specified class
GetAssetWithOpenedEditor - gets asset with opened editor
ConvertTransformToRelativeReverse - returns the given transform, converted to be relative to the given ParentTransform (old style)
ModifyObject - modify object

Animation

TwoBoneIk - two bone ik with aim correction and stretch
GetPoleVectorLocation - finds location where should be the pole vector for ik
AimConstraint - aim constraint solver
RemapVectorFromAim - remaps vector using aim settings
GetRigMapping - gets rig mapping settings from retarget window
UpdateSkeletalMeshPose - updates SkeletalMeshComponent pose in editor
SetMorphTarget - sets morph target weight in editor (anim event graph node)
GetWeightedAverageRotator - gets weighted average rotator from array of rotators and weights
GetWeightedAverageVector - gets weighted average vector from array of vectors and weights
GetWeightedAverageTransform - gets weighted average transform from array of transforms, weights and offsets
GetClosestAxis - gets closest axis to the target vector
GetAimAxis - gets aim axis for aim constraint
GetAimSettings - gets aim settings for aim constraint
GetBonesHierarchy - gets hierarchy of bones between root and tip bones
SetBonesTransforms - alters translation and rotation of bones array (anim blueprint graph node)
SetBoneTransform - alters translation and rotation of the bone (anim blueprint graph node)