NOIR PROJECT

SIMPLE GAUGE MAKER

COMPONENT DOCUMENTATION

NOIR PROJECT
SHADOWNOIR.COM

Table of Contents

Installation

Before we start to install, it's simple component with advanced feature for creating custom gauge or speedometer. Some template examples are available inside Prefabs folder, and some template scene is available for showing the prefabs too, Hope you enjoy. I appreciate any kind of comments and looking forward to make this component much better.

This component is also run in editor too, so you can switch to Game window for preview and setting up the values.

Right now we start to create an ordinary Speedometer with two needle one for Speed and The other for RPM, then create a dial texts and Digital Speedometer on the dial LCD shape. All of the training files are available in assets folder of unity package.

Now create a new scene, then a new empty game object and add the gauge component on the game object from:

Noir Project > Simple Gauge Maker > Gauge

The component should be look like this:

Hide: This will **temporary disable and hide** the Simple Gauge GUI(s).

Gauge Inputs: This can **handle the input** and connect the inputs to the **Needle value** or **Text Layer** with **tag name**.

Dial: This is **a Dial background**, this layer is in back of any other layer and handle the dial background, **all of the position relations are relative** to the dial position.

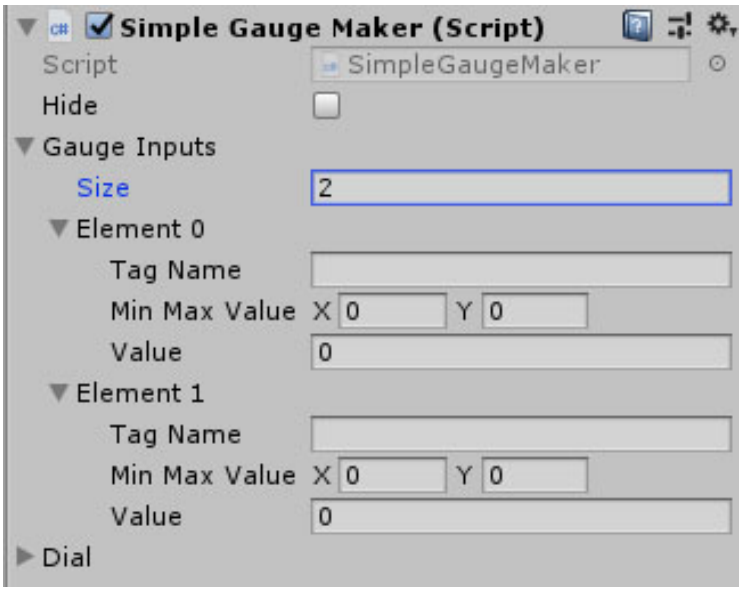
Needles: This is gauge needle, you can have one or more gauge (infinite) on a single dial, such as: **Speedometer** and **RPM Meter** or also the **fuel tank** and **Water Temperatures**. The needle also **support compound gauge** type which have **2** or **more measurement** in dial, the example is available inside prefabs folder, **WaterGauge**, **VoltageGauge** and the **CompundGauge**.

Knots: This is a needle knot and can be aligned automatically or manually with the needle, the knot layer is on top of the others.

Text Layer: With this definitions you can write text on the screen or the dial for example: mph, km/h, the value font type, font style, font size, color, … can be changed the alignment can be customized, the text layer can return the Input values too, so when you send the speed to the speedometer also you can print it out as digital speedometer on the screen.

Gauge Inputs:

can **handle the input** and connect the inputs to the **Needle value** or **Text Layer** with **tag name**. and now we start to work with the custom inputs and handle our inputs for our gauge, we would like to have two inputs: Speed and RPM.



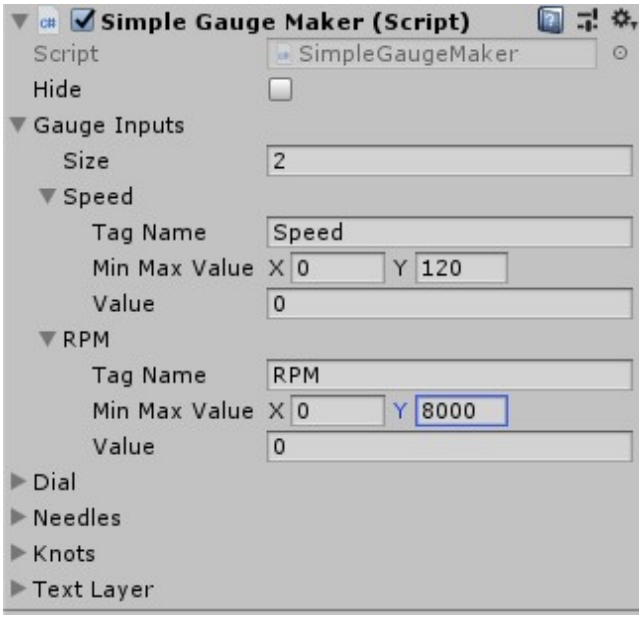
Open up the **Gauge inputs** and then **set** the **Size** to **2.**

Here is 3 Definition for each Gauge Input Element: **Tag Name**, **Min Max Value** and **Value.**

TagName: tag name is the name of the value every where in this component you can link the values with this name to each other. So the tag name is key element for finding the values in other sections, now **Set the TagName of first** element to **Speed** and set **the second tagname** to **RPM**. (tag names are case sensitive so you should always use the Letter cases the same)

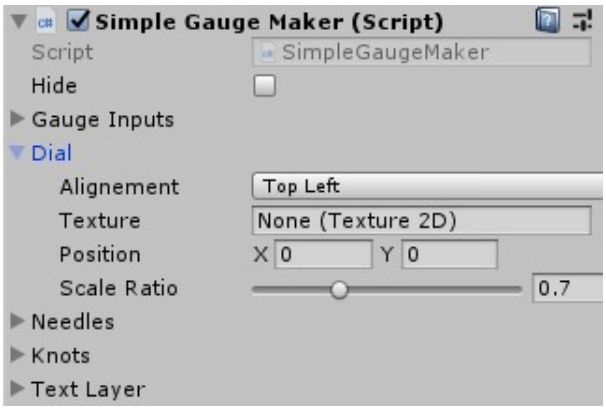
Min Max Value: The X is for minimum and the Y is for the Maximum value, that each of the input elements can handle, so the value can't be exceed from this value. For the **Speed** value set **Min to 0 and Max to 120**. And for the **RPM** value set **Min to 0 and Max to 8.**

Value: this value can be used by the other elements, such as needle, the value reach by the **TagName** you defined earlier.



Dial

Here we can handle how our dial look like. Like the image below it's so simple, define texture, set the alignment, set the position and scale ratio and everything is done. Now we setup our training dial.



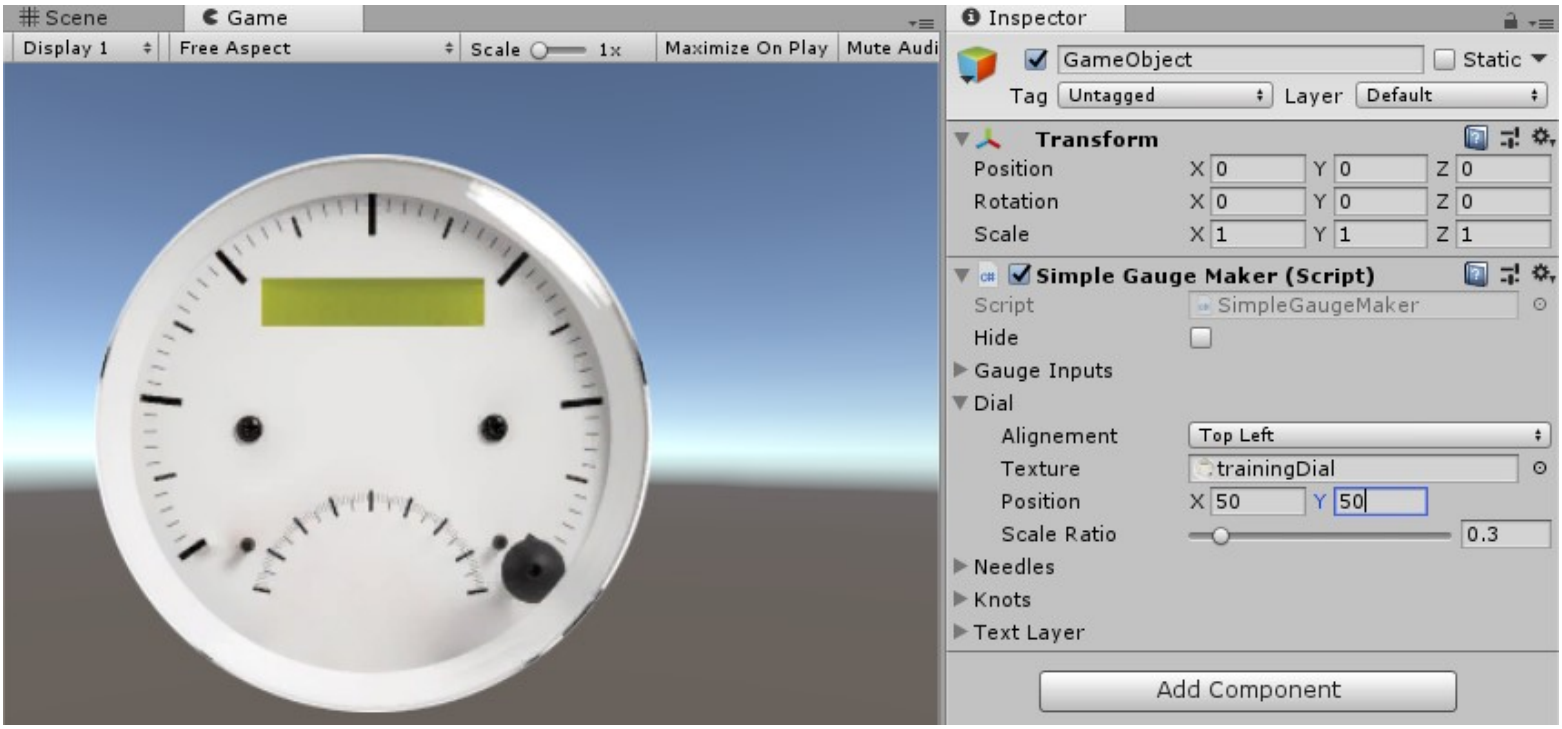
Texture of dial can be set through the texture definition, for now select texture from **Assets** > **UI** folder, the dial texture name is **trainingDial** there in UI folder**.**

Alignments select the kind of alignments. The alignment options are **Top Left** and the other is **Bottom Right**, the alignment values calculated from the screen width and height.

Position set the X and Y to position the ui on screen. The position aligned with alignment type.

Scale Ratio is texture size multiplier, you can zoom in or out in your texture for better sizing.

Set the values like the image below. And the dial setup is completed when you have something similar to the image below.

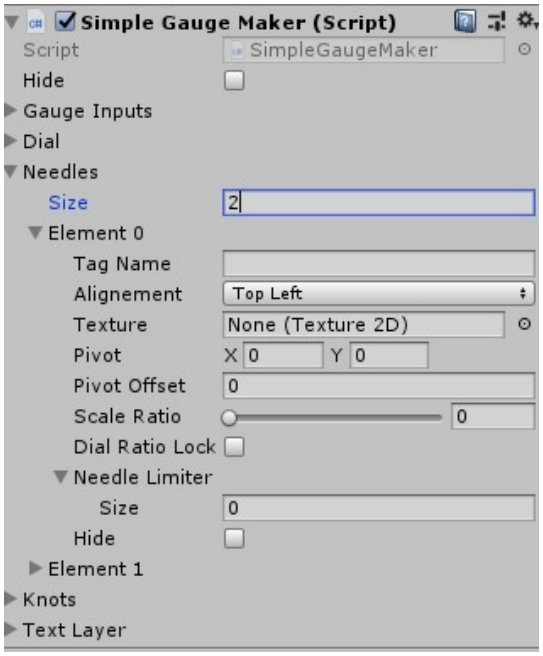


Needles

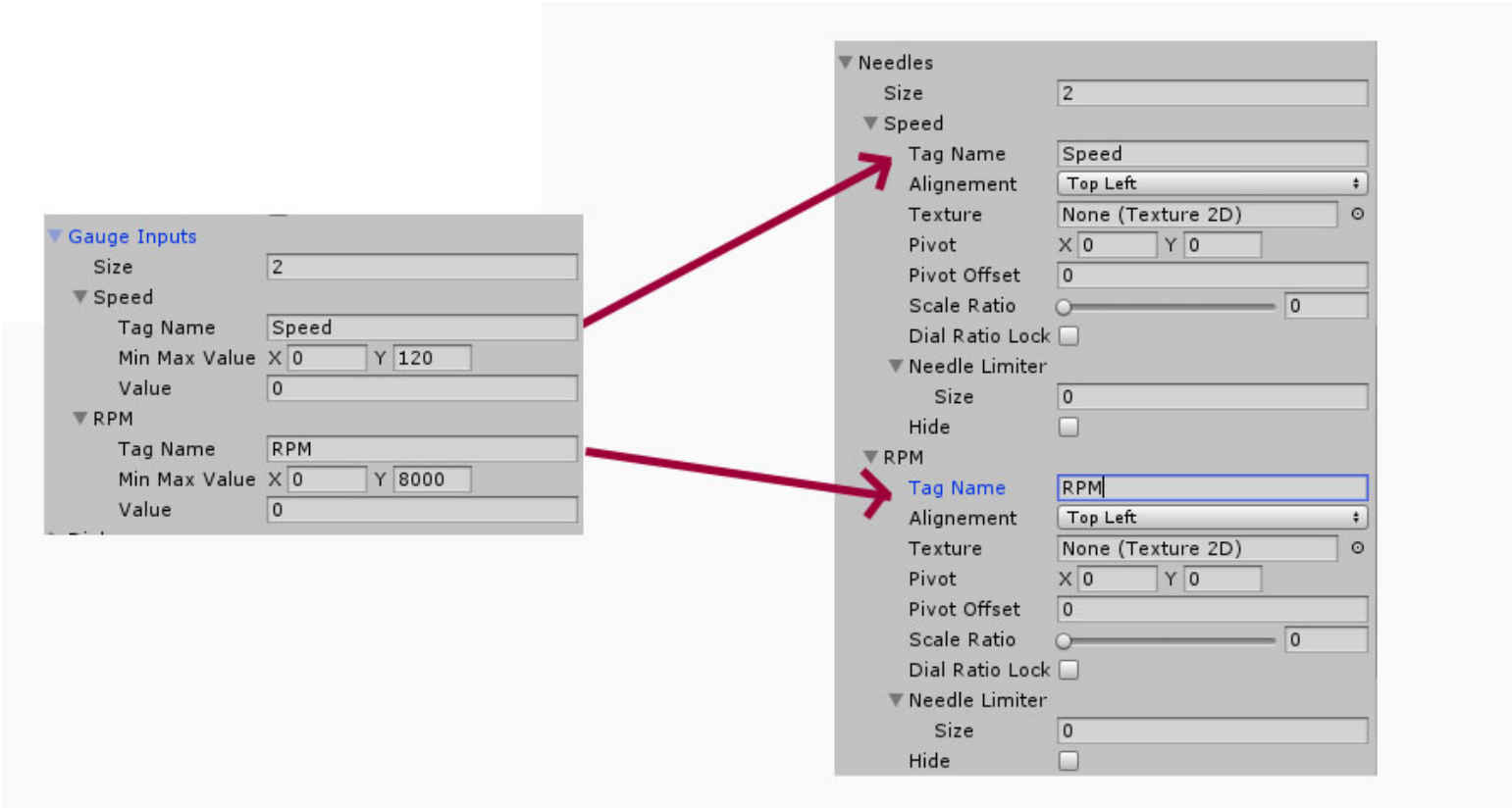
The needles can be infinite in single dial, now in this training we set 2 different dial one for speed and the other for RPM, and the pivot axis of the needle is much with offset. the dial we use in this training not have any knots and is not have any compound value, for this kind of gauges we talk in the later sections.

For start open up the **Needles** arrow and set the **needle size** to **2**, one for **speed needle** and the other for **RPM**.

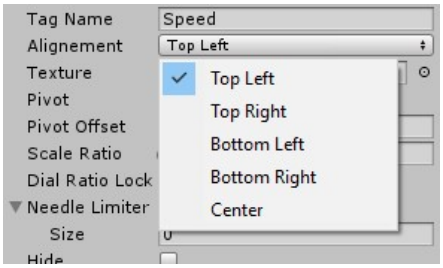
Take a look on needle section.



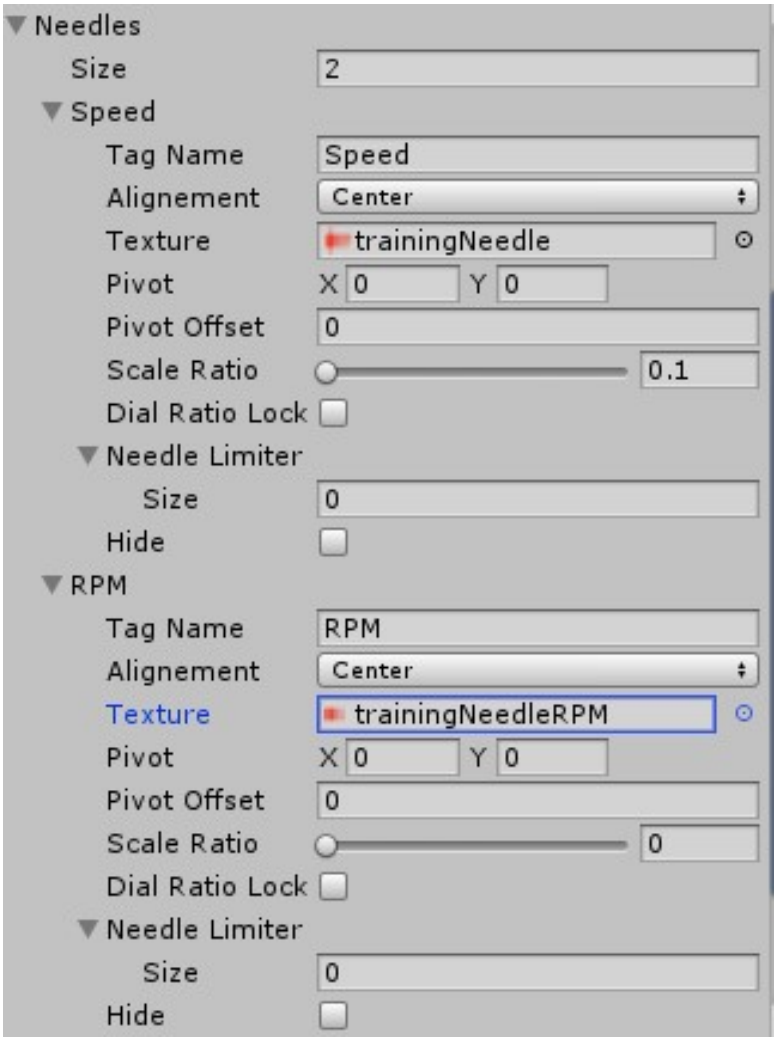
Tagname when this value set as the same as the tagname inside **Gauge Input** section, the gauge input value replicated in here, so for transferring input values to the gauge needle we need to set the tag names the same. For now set the First Element TagName to Speed same as The **Speed** in Gauge Input and set the second Element tag name to **RPM**.



Alignement is **calculated from dial position** and **alignment** and applied to this needle, the pivot position changed on the different Alignment type and it contain the alignments type such as shown in image below, for now I will use **Center** from the list for each of two needle.

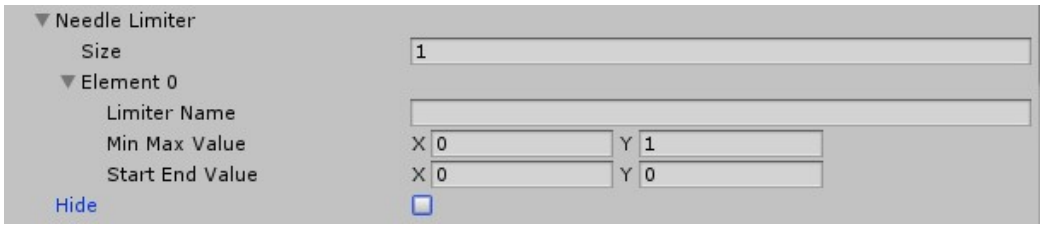


Texture this is a needle texture, the texture should be from left to right and the size should be power of two. Such as: 128x32, now I will use the **trainingNeedle** for the Speed needle and the **trainingNeedleRPM** for the **RPM** needle, the textures are available in **UI** folder, you can use same texture for two needle but because the RPM needle is smaller in our dial so you need to reduce the **scale ratio** of the second needle, with the option below. For now I will use two different texture, **trainingNeedle** and **trainingNeedleRPM** from UI folder.



NOTE: BEFORE START CONFIG

Before we start to configure this values, you should do something for preview the needle, on top of the hide we have something called Needle Limiter, set the size if this value to 1 if it's 0, we talk about this later. Then inside the Needle limiter we have min max value, set the min to 0 and the max to 1 for now, It's mandatory for preview the needle on game window.



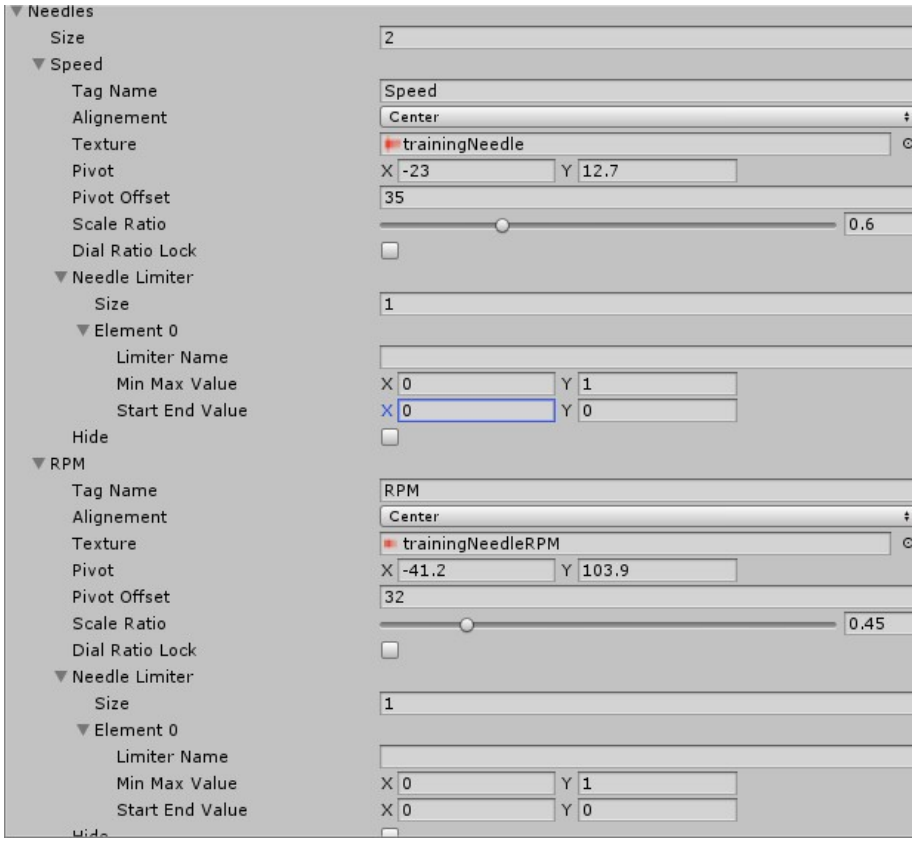
Scale Ratio this value will set the texture scale of the Needle, the ratio is multiply to the width and height and the new size will calculated, for now set **the scale ratio** to **0.6** for **speed**, and the **RPM** to **0.45**.

Pivot the pivot is the point of rotation, the needle rotate around this pivot position and the value calculated based on alignment. In other word the pivot is the point the needle attached with his nail to the gauge in the real dial.

For now I will set the pivot **X to -23 and the Y to 12.7** for the **speed ,** and the X for the **RPM** to **-41.2** and the **Y** to **103.9**.

Pivot Offset the pivot offset is the offset between the left side of the texture and the needle nail, good to know, the pivot offset will changes when scale ratio is changed. Now I will set the Pivot offset for **speed** to **35** and for the **RPM** set the value of pivot offset to **32.**

Dial Ratio Lock when this value set to checked or true, the needle scale ratio locked on the dial scale ratio, and set as the same, if works when the dial image and needle is in same scale ratio in texture width and height, for now leave the Dial Ratio Lock unchecked or false.



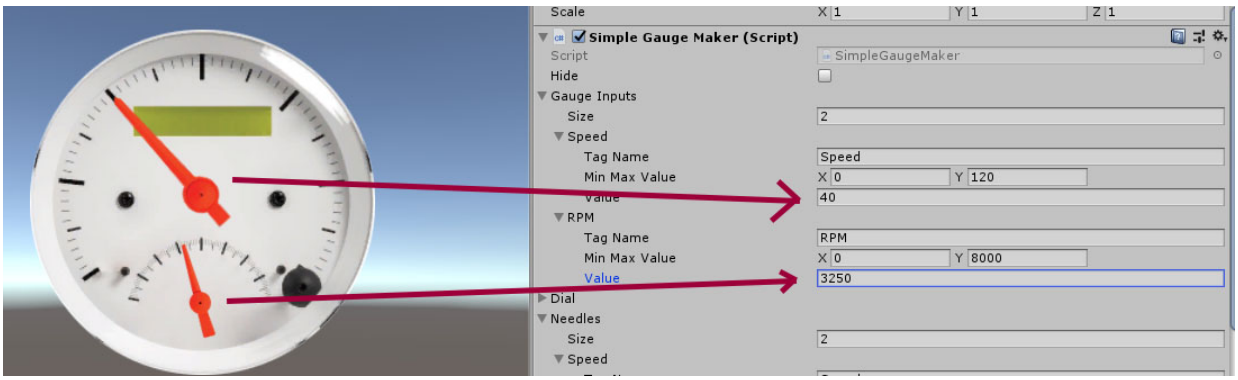
Needle Limiter in the past I will tell you to set the size of this definition to 1 and then set the min max to 0 1, so what is it and why this value is array. For now we just need single value and set the size to 1, the more value is used in compound gauge we talk about that after this section. Now set the size to 1 now we have this values for each limiter element.

Limiter Name this value is just name for knowing the item. No link and replication you can also leave it blank.

Min Max Value the x value is the same as the minimum and maximum value we set earlier on the gauge input section, this is the minimum and maximum input of the gauge needle, this value will normalized with the input value . for now set the values like in input values for **speed X: 0 Y: 120** and the **RPM to X:0 and Y: 8000**

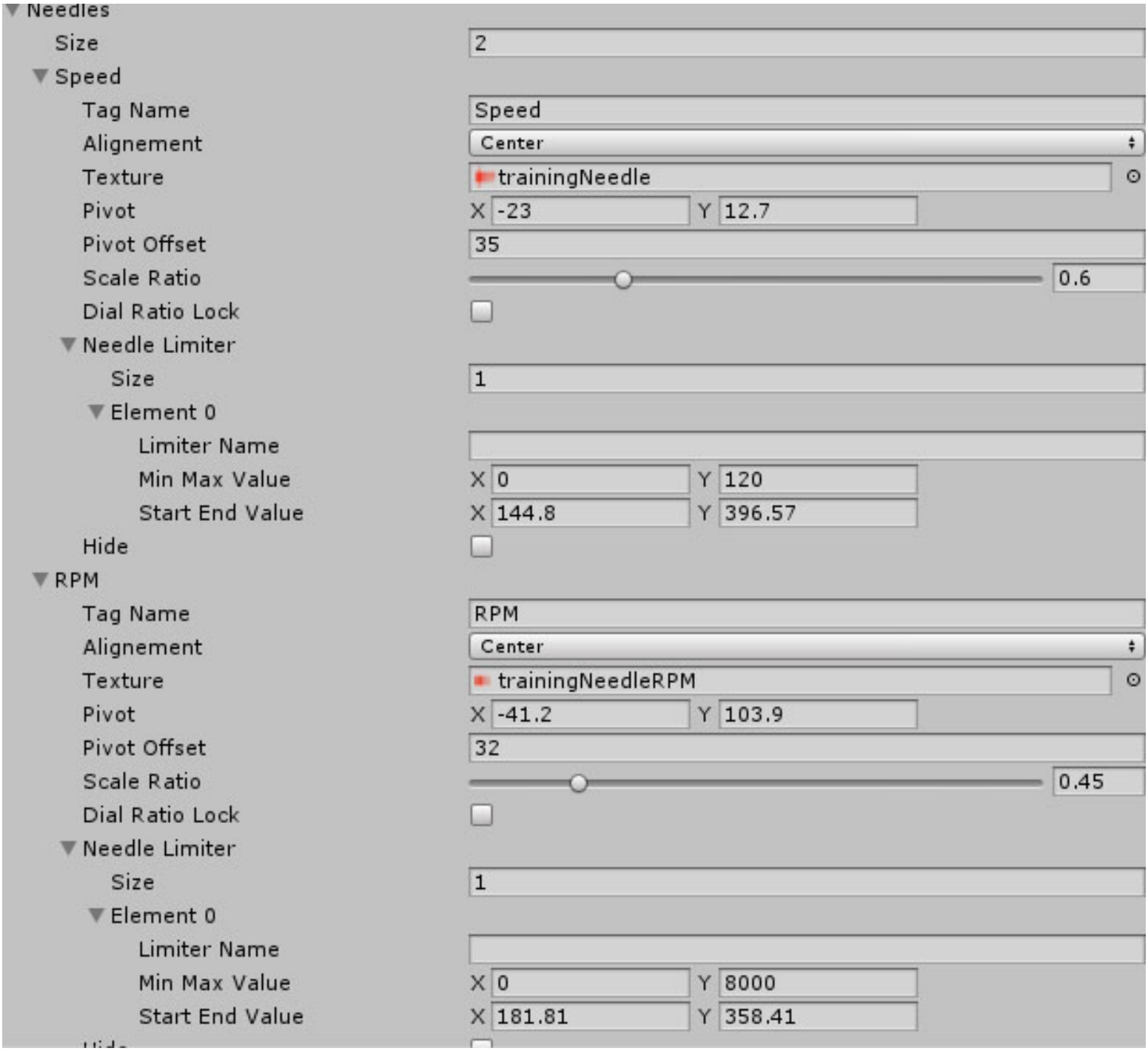
Start End Value this **X an Y values** are for the **degree limitation of the needle**, X value is the degree of Minimum input value and the Y is Maximum value from input. **Now for the Speed** set the **X** to : 144.8 and for the **Y**: 396.57, and then for the **RPM** set the **X** to: 181.81 and the **Y** to: 358.41

Now you should able to change the gauge input values and then the analog gauge should be synced and rotate around in correct way. All of the inputs should be applied through the gauge input section.



Hide this can hidden the needle when checked and vise versa.

Full view of needles:



Later we talk about how we can use needle limiter for compound mesurement dials.

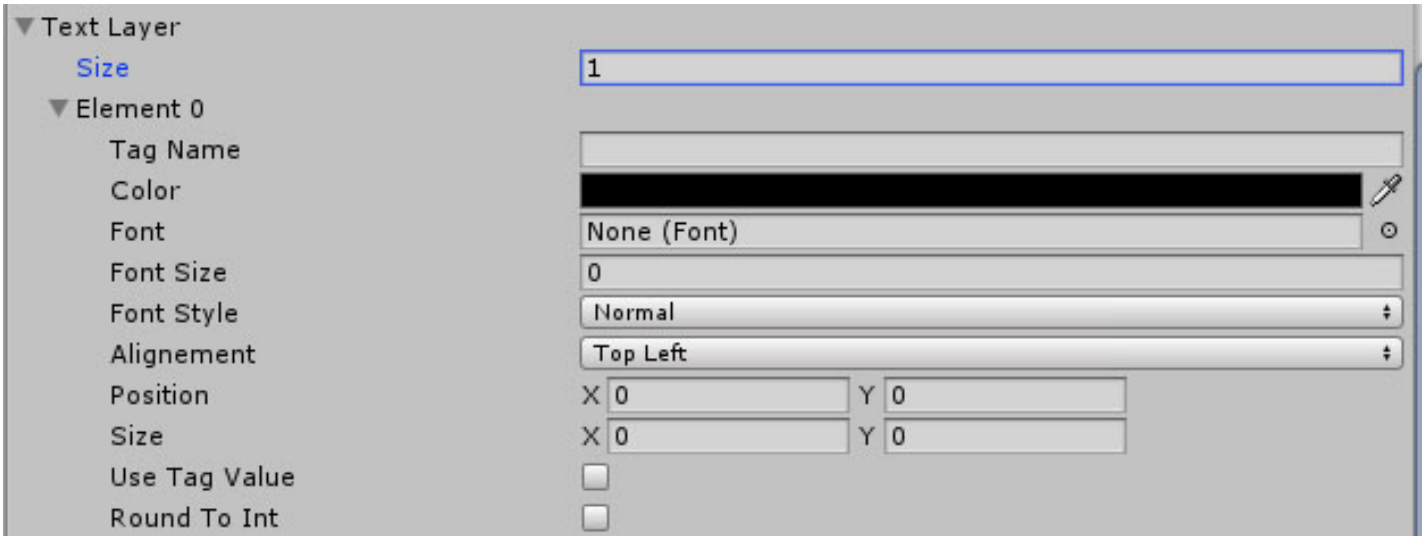
Knots

We talk about knots in compound dials section later. Here in this training we do not need ant knots for the needles. In the next training we use another dial require knot.

Text Layer this is a text layer, you can print any text on the screen and or you can print out the input values too, in this training I will make the all of texts on the dial. Let see the Text Layer definitions.

Here I need almost 20 text just for training, but you can make all of the text via photo editing tool such as Photoshop too. Here I just show to make some of the text layer but all of the training text available on the **training01** scene **inside scene folder**. The file is containing all of this training changes.

Ok lets see the Text Layer definitions:



Size is for the how many text we need on the ui, you can also handle the other texts you need on the screen with this text layer and input values, Each element containing few definitions:

Tag Name when you want to print out just a text or number you can put anything in this value and the value showed on the screen. Sometime you would like dynamic values such as speed or gear, water flow, voltage, etc. you need to set the TagName to the same Tagname from the gauge inputs. For now one one my labels tagname called Speed, as the same of the speed previously we ake in guage input values.

Color this is color of the text. Take care of color alpha channel and set it to 1 if your text isn't appeared.

Font the fontface type. For now select Arial.

Font Size the fontsize of text layer. For speed text I will set 20.

Font Style the font style, **bold**, **italic**, **bold italic** and **normal**. Now Set It 14 for speed text layer. Select Bold For Speed Text layer.

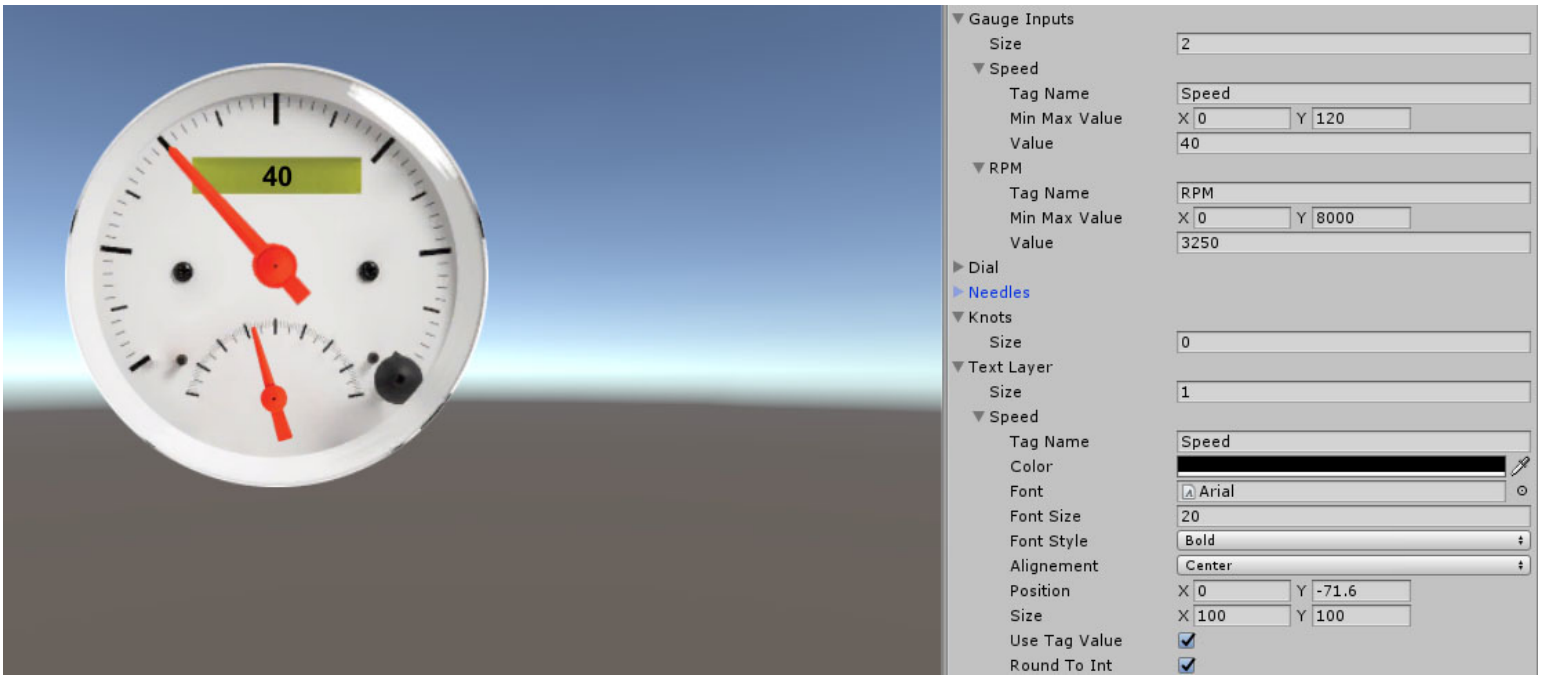
Alignment is the textlayer alignment, this contains, **Center**, **Bottom Left**, **Bottom Right**, **Top Left**, **Top Right**.

Position this is XY position of the text on the screen, based on dial alignment and text alignment. For speed I will set x: 0 and y: -71.6, to position the speed on the dial lcd shape.

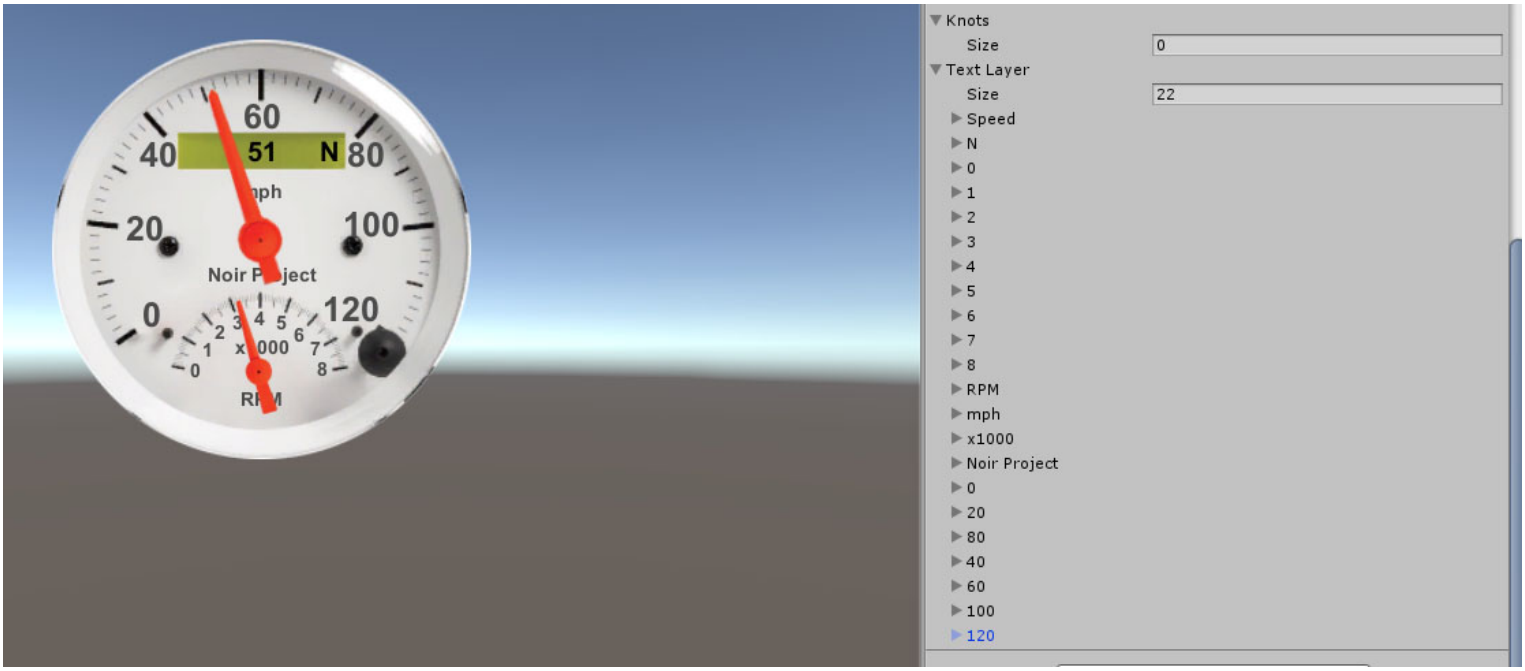
Size the size of text gui width and height. I will set 100 100 for all of the texts.

Use Tag Value when this value is checked and the tag name is the same as the gauge input tagname, the value of input text will shown on the text layer. When the input value isnt available on the Gauge Inputs, it will return 0.

Round to Int the value returned is float in type, with this checked you can parse it as a Integer. For speed we need this checked.



Now you should have something like the image above. Just for speed. Now add the other texts such as rpm, mph, gear name, dial numbers if you would like to add here instead of photoshop or something else, the result would be something like this:



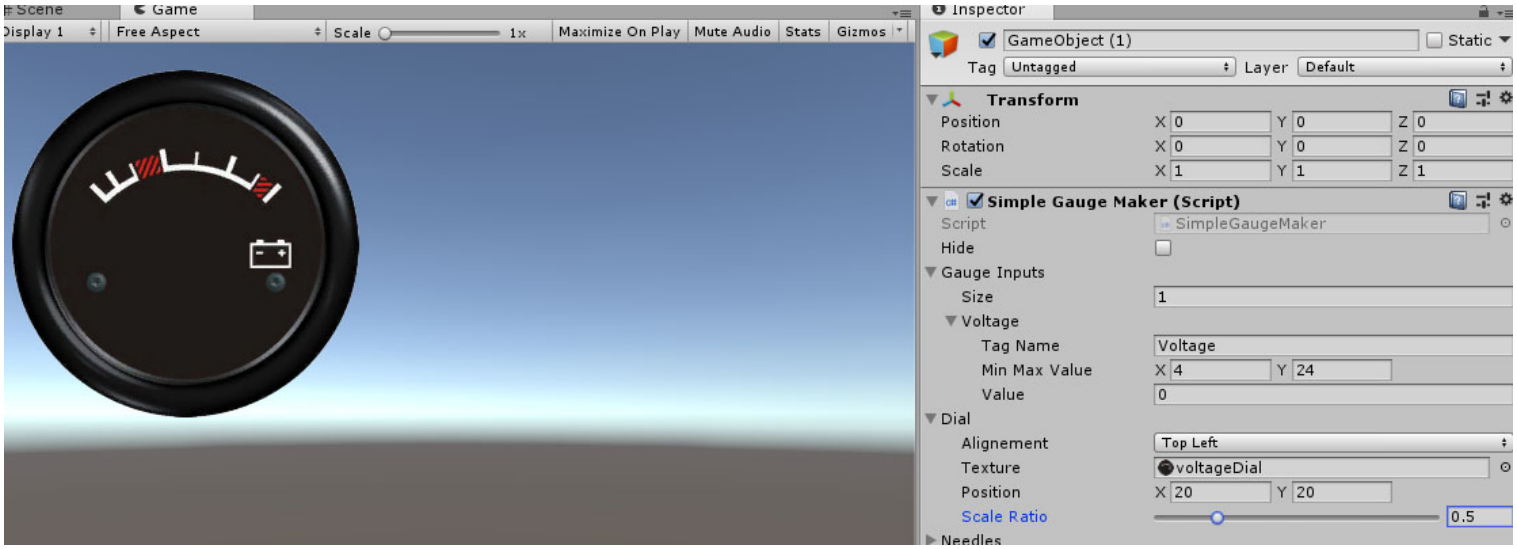
as you see all of the text layer now made by Guage maker and ready to use.

Now you learnt how to make a normal speedometer, for how to use scripting to access this component we talk later in Scripting section of this documentation.

Compound Dial

Right now you know, how to make a dial, but sometime we need to have a dial with different measurement like voltage dial, the ranges are different and the analog dial should show the same as the input value, so the normal dial cannot carry this. In other hand sometime we made a picture of real dial but because of perspective of the image the measurement isn't correct, so the compound dial can help us to solve the issue.

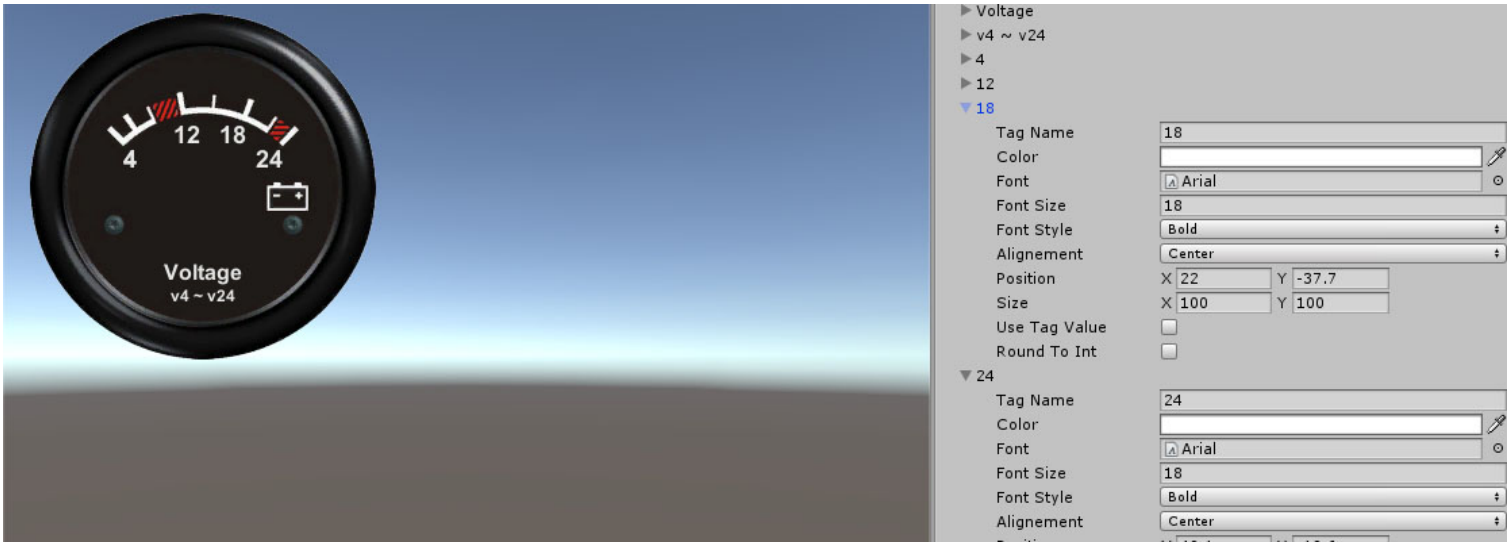
Now create a new game object add the gauge component on the game object, then **set the dial from UI** folder, the dial called: **voltageDial,** set the **position** I set **20**, **20**, and then set the **scale ratio** to something you like, I will use **0.5**. And also **add new input** called **Voltage**, in gauge input manager and set the **min max** to **4** and **24**, You should now have something like this.



Till now everything was the same as a normal dial.

We need some text layer for texts and a **Knot**, and also a needle for showing different range of voltage.

Start to add some text layer, because we talk about the text layer earlier, I just show the result, the text aren't replicate any value from inputs in this gauge. Add some normal text like the image below.



The dial and text are now ready now we start to add a compound needle.

Add one needle by set the needles size to 1. Then pick **needleVoltage** texture from **UI** folder for needle. And set the tag name **Voltage** as same as the input value. Set the **alignment** to **center**, leave position offset to 0, set the **scale Ratio** to **0.6,** set the pivot to (X -26, Y 39.42), leave dial lock unchecked.

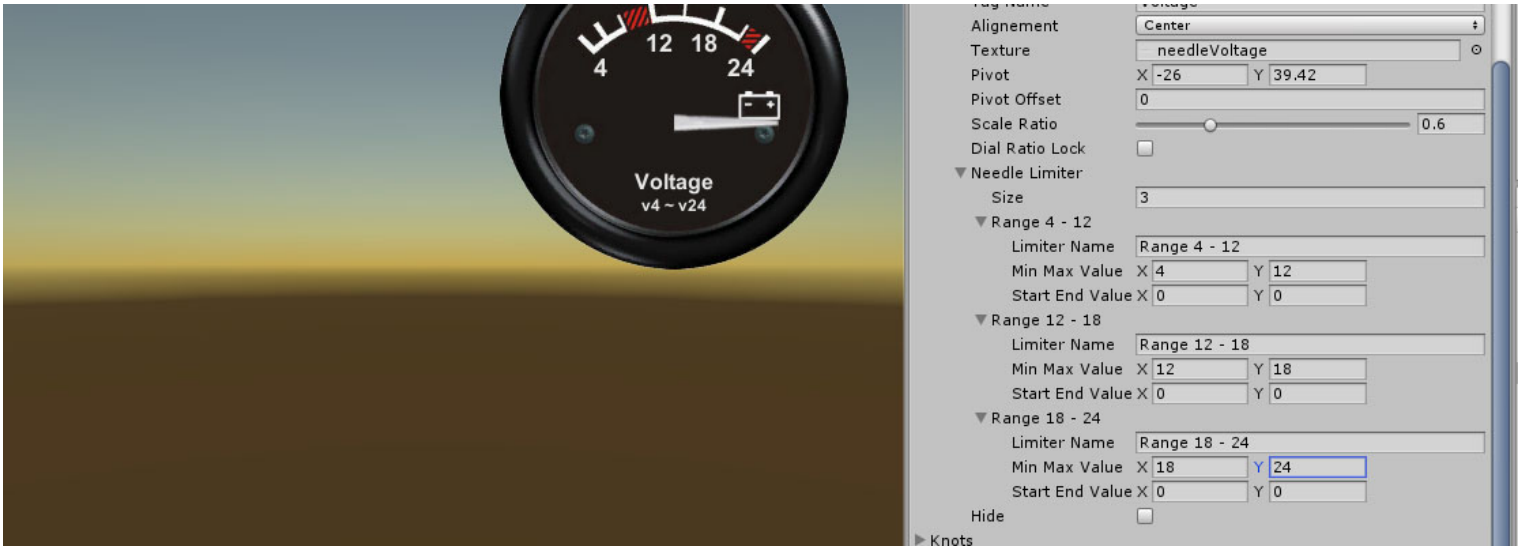
The compound dial is works with how we use needle limiter option.

For now we have something similar to this without any needle showing.

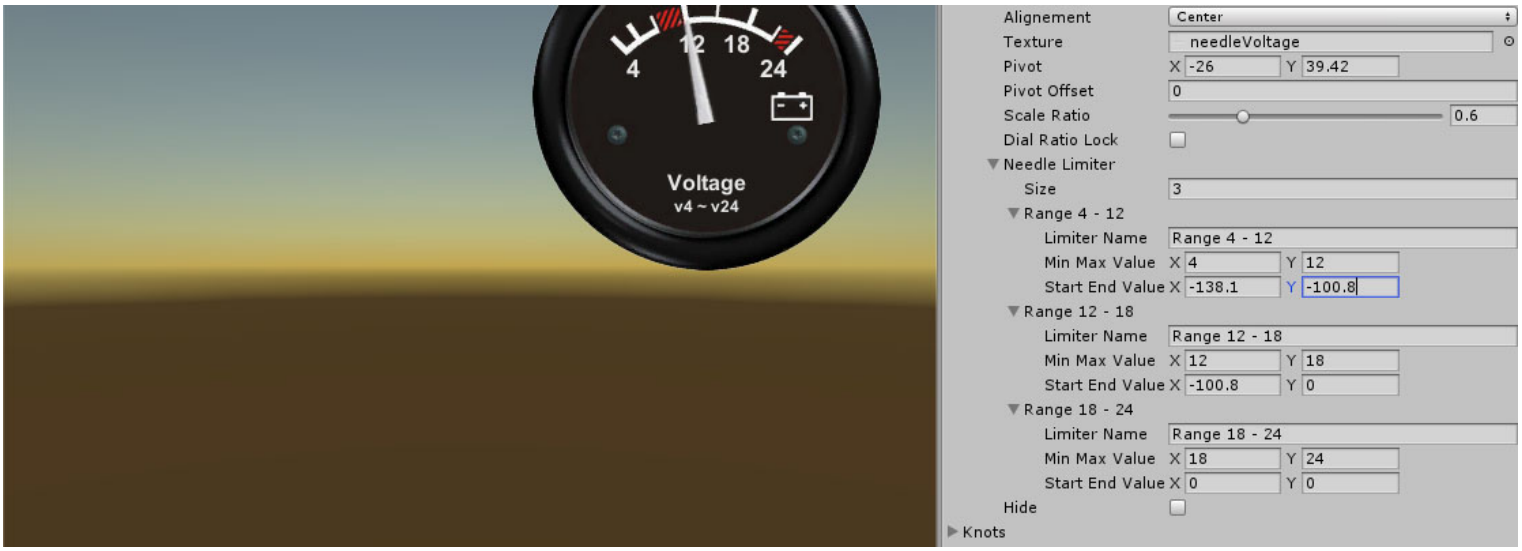
For the clarification of Needle Limiter, this is an aligner for the degree and range of input values. This feature can align the range on inputs with the range of rotation degree, for example we have a different ranges between 4-12 and 12-18 and 18-24 and when we rotate the dial with the same speed the analog dial isn't show the correct value.

For this dial we need to add 3 different ranges, so set the **limiter size to 1** and then you can define the limiter name for better knowing of what is it. Then set the min max values like our ranges. [x 4, y 12] [x 12 – y 18] [x 18 – y 24]

You should have something like image below.

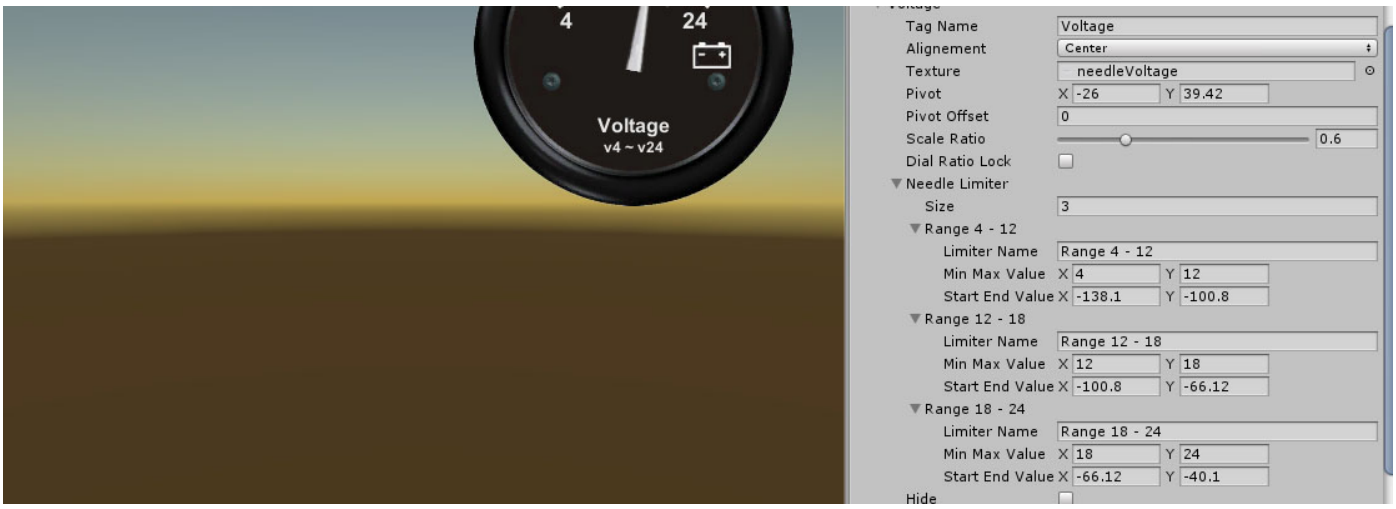


Now for setting up start and end degree of each range, we can set the voltage input value to 4 then can calibrate start value of the Range 4 – 12. So the degree for start end of first range is: -138.1 and then set the value of voltage to 12 then change the start value of the second range to find where is 12 indicator degree, the value was: -100.8, now you should copy this value in Y (End) of the first range. So the Y of the first range is same as the X of the next range.

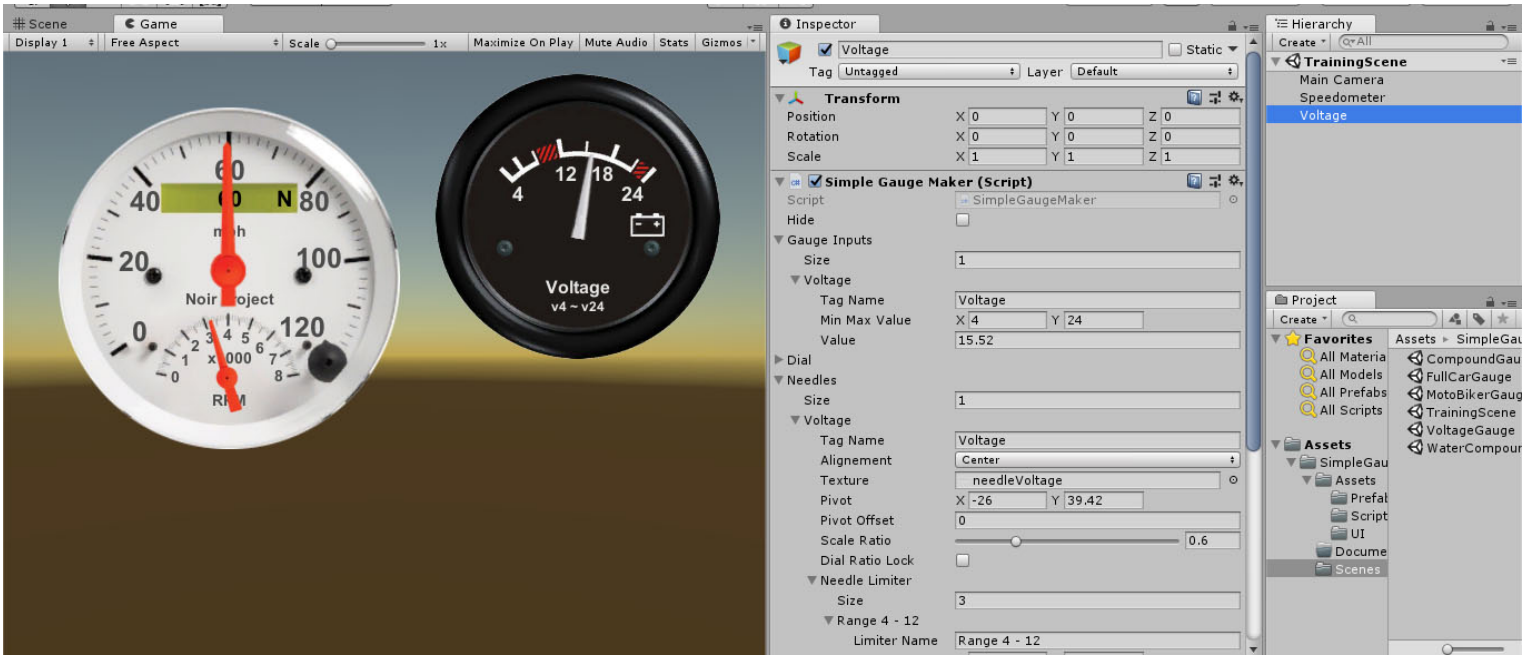


Apply this steps for all of the ranges and then you should have something like this setup:

For the last range end degree set the input voltage value to the 24 (the end of range) and find the degree like this image:



Now when you change the input value, the needle correctly show the value of the dial, this is an compound dial. The scene called **TrainingScene** contain this gauge too.



You can place infinite different Gauge component in one scene.

Scripting

Ok now we have custom gauge and it works, now we need to know how to connect out player speed to the dial. Now I will create a new C# script, inside script folder and call the script **gaugeValueSample.cs** and then I have some code like this:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class gaugeValueSample : MonoBehaviour {

        // Use this for initialization
        void Start () {

        }

        // Update is called once per frame
        void Update () {

        }
}
```

For finding or getting a game object many different approaches are available, for now I just create a transform an get the Gauge component from it.

Add

```
public Transform SpeedGaugeTransform = null;
private SimpleGaugeMaker speedGauge = null;
```

before the start, so you can attach any transform on scene to the **`SpeedGaugeTransform`** later to this transform.

And then add this code:

```
if (SpeedGaugeTransform != null)
{
    if (SpeedGaugeTransform.GetComponent<SimpleGaugeMaker>() != null)
    {
        speedGauge = (SimpleGaugeMaker)GetComponent<SimpleGaugeMaker>();
    }
}
```

Inside the Start() method.

Note: in the next screen when I show the almost completed code, you may see this line of code before class start:

```
[AddComponentMenu("Noir Project/Simple Gauge Maker/Gauge Sample Value Modifier")]
```

This will handle the script location inside the component add menu, so you can easily find your script at desired location. Also you can change or write down a new one for your scripts.

So your code should be look like this now.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[AddComponentMenu("Noir Project/Simple Gauge Maker/Gauge Sample Value Modifier")]
public class gaugeValueSample : MonoBehaviour
{
    public Transform SpeedGaugeTransform = null;
    private SimpleGaugeMaker speedGauge = null;

    // Use this for initialization
    void Start()
    {
        if (SpeedGaugeTransform != null)
        {
            if (SpeedGaugeTransform.GetComponent<SimpleGaugeMaker>() != null)
            {
                speedGauge = (SimpleGaugeMaker)GetComponent<SimpleGaugeMaker>();
            }
        }
    }
    // Update is called once per frame
    void Update()
    {

    }
}
```

Now we want to increase the speed gauge value in time and then reverse it down by creating the simple script that command the gauge for now value in time. Because this is a sample for how to change the value of the gauge and isn't a coding training, so I just made everything here so simple. So for the better and optimizer way to change the value in time please read the unity reference.

Now add these 4 definition line before start,

```
public float speedMultiplier = 40.0f;
private float _tempValue = 0;
private float currentSpeed = 0;
private bool reverse = false;
```

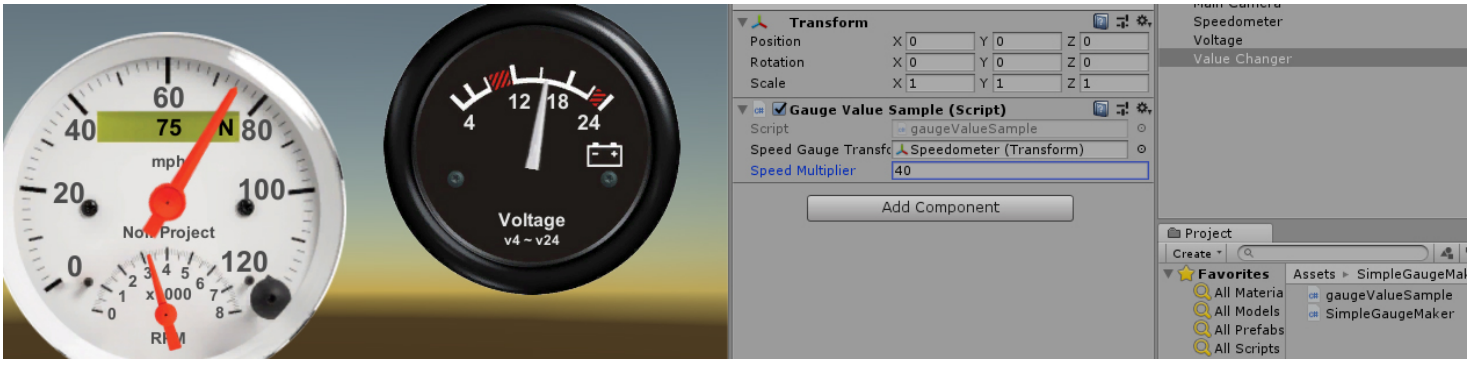
and then inside Update Method add this codes:

```
    //this will increase and reduce the number by time and time speed Multiplier
    if (!reverse)
    {
        //increase the value by time multuiplier when reverse is false.
        currentSpeed += speedMultiplier * Time.deltaTime;
    } else {
        //vice versa, reduce the value when reverse is true.
        currentSpeed -= speedMultiplier * Time.deltaTime;
    }

    //when the value is equal or over the 120 set the rever on
    if (currentSpeed >= 120)
        reverse = true;

    //when the speed is equal or under 0 set re reverse off
    if (currentSpeed <= 0)
        reverse = false;

    //this line send the current value to the gauge desired input
    speedGauge.setInputValue("Speed", currentSpeed);
```

so your code should be similar to this:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[AddComponentMenu("Noir Project/Simple Gauge Maker/Gauge Sample Value Modifier")]
public class gaugeValueSample : MonoBehaviour
{

    public Transform SpeedGaugeTransform = null;
    private SimpleGaugeMaker speedGauge = null;

    public float speedMultiplier = 40.0f;
    private float _tempValue = 0;
    private float currentSpeed = 0;
    private bool reverse = false;

    // Use this for initialization
    void Start()
    {
        if (SpeedGaugeTransform != null)
        {
            if (SpeedGaugeTransform.GetComponent<SimpleGaugeMaker>() != null)
            {
                speedGauge = SpeedGaugeTransform.GetComponent<SimpleGaugeMaker>();
            }
        }
    }

    // Update is called once per frame
    void Update()
    {
        //this will increase and reduce the number by time and time speed Multiplier
        if (!reverse)
        {
            //increase the value by time multuiplier when reverse is false.
            currentSpeed += speedMultiplier * Time.deltaTime;
        } else {
            //vice versa, reduce the value when reverse is true.
            currentSpeed -= speedMultiplier * Time.deltaTime;
        }

        //when the value is equal or over the 120 set the rever on
        if (currentSpeed >= 120)
            reverse = true;

        //when the speed is equal or under 0 set re reverse off
        if (currentSpeed <= 0)
            reverse = false;

        //this line send the current value to the gauge desired input
        speedGauge.setInputValue("Speed", currentSpeed);
    }
}
```

Now add newly created code on a game object, and then set the gauge transform on the created gauge transform field. Then play the scene, I do this on the training scene and the result is this: (this code is also available in script folder)



This line of code can inside Update() method parse the value to the Gauge:

speedGauge.setInputValue(Tagname, value);

Tagname: is the input tag name you would like to set the value on.

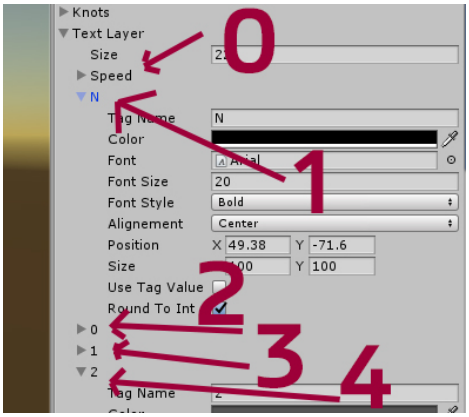
Value: is the numeric value of the current tag name you will set.

Now I would like to change the N sign (Text Layer) in LCD of the speedometer to different Text, just add new line like the above and this time use this:

```
speedGauge.setTextLayerTag(int LayerIndex, String NewTagName, bool ValFromInput)
```

NewTagname: is the new text you would like to show, or also you can change it to the tag name available on the gauge inputs so the value will shown on the text layer, just you should set the ValFromInput to true for replicating value from gauge input.

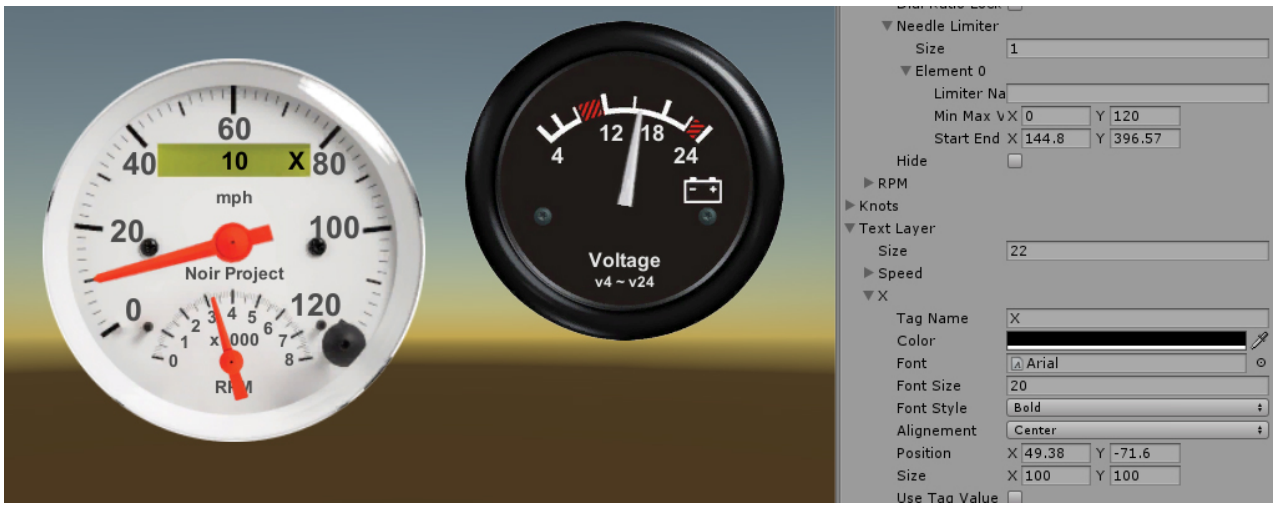
LayerIndex: because you may change the tagname in time for shoing the texts, you may cant find the tagname of the text layer, so you should set the current index of text layer you added on the guage component. You should find the index from your gauge Text layer like image below, index is ordered and started from 0.



So are N is index 1, the indexes are always are the same.

So for setting N to X we should add this line: (you can set the gear number on this)

```
speedGauge.setTextLayerTag(1, "X", false); //index is 1, new text set to X, do not read from input values
```




km/h
MPH
SHADOWNOIR.COM