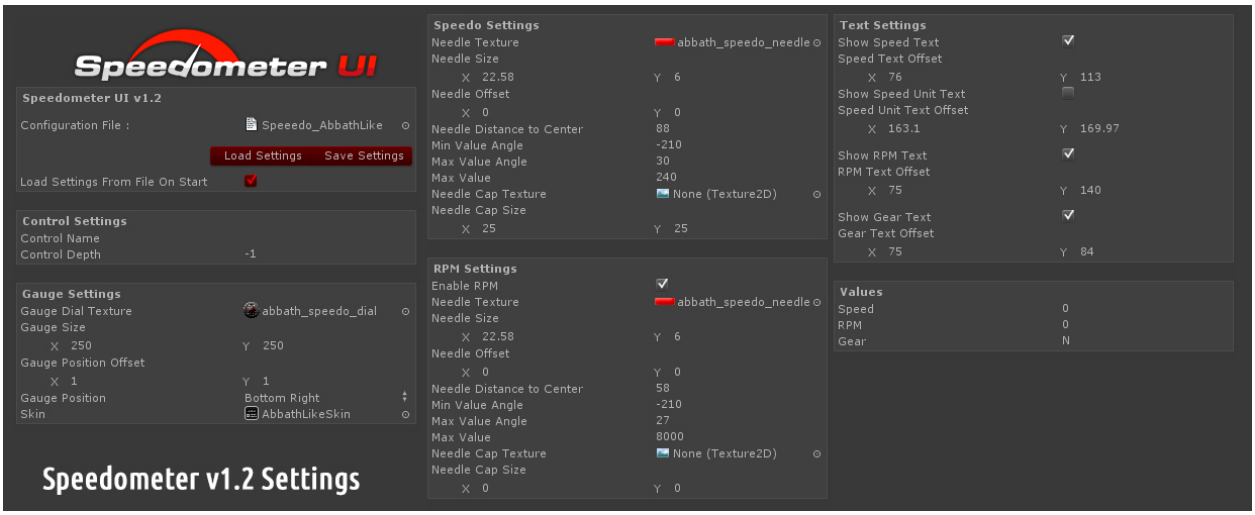
Speedometer UI v1.2 Manual

Yusuf AKDAG

6/10/2013

Speedometer UI v 1.2

Speedometer UI Created for visualize vehicle speed, rpm and gear values on screen.



Speedometer UI can have one gauge texture only. You may use separated Speedometer UI component for multiple gauges or one combined texture for both Speedo and RPM.

JSON based configurations files contains every settings and can be load in game with different settings. Also you can load/save different configurations from Unity Editor.

Settings

Control Settings

Control Name : A unique name of this control. If you use one Speedometer UI on same object, you DONT have to set this property. A detailed usage explained in Multiple Speedometer Usage subject in this topic..
Control Depth : GUI Depth of this control.

Gauge Settings

Gauge Dial Texture : Gauge itself or background texture.
Gauge Size : Gauge size on screen
Gauge Position : Gauge position on screen.
Gauge Position Offset : Position offset on screen
Skin : Skin file that contains font and size data. This skin is required to work. Each template have its own skin.

Speedo & RPM Settings

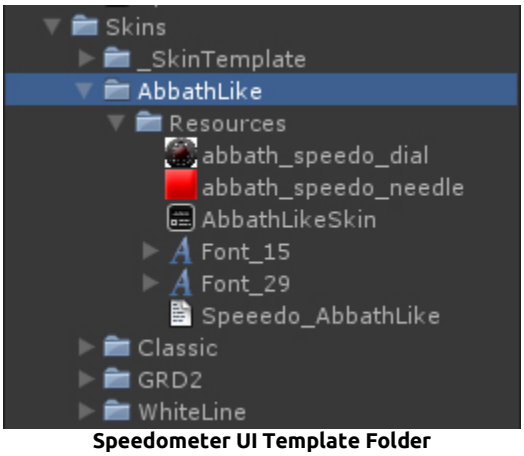
(I explained both settings here because they're exactly same)
Enable RPM : This settings only available on RPM Settings. It enables/disables rpm visualization. In case like you want to use separated gauges.

Needle Texture: Texture of needle
Needle Size : Size of needle on screen
Needle Offset : Needle offset to center
Needle Distance to Center : Needle distance to center, you can see the usage from AbarthLike template
Min Value Angle : Minimum rotation angle of needle. This will be 0 value position.
Max Value Angle : Maximum rotation angle of needle. This will most possible rotation of needle.
Max Value : Max value is like your vehicles maximum speed. Its always better to set exact value of shown on your gauge. Rotation calculation based the this max value.

Text Settings

Show Speed Text : Enables/Disables visualization of speed text
Speed Text Offset : Speed text position offset relative to gauge.
Show Speed Unit Text : Enables/Disables visualization of speed (or rpm or any other thing) text.
Speed Unit Text Offset : Speed unit text position offset relative to gauge.
Show RPM Text : Enables/Disables visualization of RPM text
RPM Text Offset : RPM text position offset relative to gauge.
Show Gear Text : enables/Disables visualization of gear text.
Gear Text Offset : Gear text position offset relative to gauge.

Skin Structure



Speedometer UI Template Folder

Speedometer UI can load templates on fly. Because of Unity's requirement, all dynamic content must be located in Resources folder. And also all content in resources folder requires unique name. You may have two different resources folder but you cant have two textures named "needle"

With this requirement, Speedometer UI template folder located at "Skins/Skin Name/Resource/" folders as seen the image above.

Its highly recommend to dublicate _SkinTemplate folder before create your own skin.

Speedometer UI v1.2 Manual

Basically a GUI skin requires two textures; Gauge Dial and Needle. Also if you're going to use any text, i highly recommend to convert them to static fonts. Do not forget to assign your fonts to in skin file.

Skins

Speedometer UI Skins

Speedometer UI comes with 4 different skins.
AbarthLike is example of use speedometer and rpm in one gauge.
GRD2 is example of use of single speedometer usage
Classic is example of use speedometer and rpm in two gauges.
Whiteline is example of use separated gauges of speedometer. In this example same object have 3 different speedometer components assigned.

Vehicle Connections

Speedometer have 3 values for visualization; Speed, RPM and Gear. You can send this values as shown in examples below (You can find this examples in SpeedometerUIVehicleConnector.cs script);

RGKCar ;

```
using UnityEngine;
using System.Collections;
using RacingGameKit;
using RacingGameKit.RGKCar;

public class SpeedometerUIVehicleConnector : MonoBehaviour
{
    SpeedometerUI SpeedometerUIComponent;
    RGKCar_Engine RGKEngine = null;

    void Start()
    {
        SpeedometerUIComponent = base.GetComponent<SpeedometerUI>();
        RGKEngine = base.GetComponent<RGKCar_Engine>() as RGKCar_Engine;
    }

    void Update()
    {
        if (RGKEngine!=null){
            if (SpeedometerUIComponent != null)
            {
                SpeedometerUIComponent.Speed = RGKEngine.SpeedAsKM;
                SpeedometerUIComponent.RPM = RGKEngine.RPM;
                string Gear = RGKEngine.Gear.ToString();
                if (Gear == "-1") Gear = "R";
                if (Gear == "0") Gear = "N";
                SpeedometerUIComponent.Gear = Gear;
            }
        }
    }
}
```

UnityCar ;

```
using UnityEngine;
using System.Collections;
[AddComponentMenu("Racing Game Kit/Speedometer UI/Speedometer UI - Vehicle
Connector"), ExecuteInEditMode]
public class SpeedometerUIVehicleConnector : MonoBehaviour
{
    SpeedometerUI SpeedometerUIComponent;
    Drivetrain UnityCarDriveTrain = null;

    void Start()
    {
        SpeedometerUIComponent = base.GetComponent<SpeedometerUI>();
        UnityCarDriveTrain = base.GetComponent<Drivetrain>() as Drivetrain;
    }

    void Update()
    {
        if (UnityCarDriveTrain)
        {
            if (SpeedometerUIComponent != null)
```

```
            {
                // This conversation required because UnityCar scripts lack of
vehicle speed property..
                SpeedometerUIComponent.Speed =
float.Parse((UnityCarDriveTrain.velo*3.6).ToString());
                SpeedometerUIComponent.RPM = UnityCarDriveTrain.rpm;
                string Gear = UnityCarDriveTrain.gear.ToString();
                if (Gear == "-1") Gear = "R";
                if (Gear == "0") Gear = "N";
                SpeedometerUIComponent.Gear = Gear;
            }
        }
    }
}
```

Multiple Speedometer Usage

In some cases, you may want different sized gui textures for your vehicle. For that scenario, you have to use separated SpeedometerUI components for each type.
In this example I created 3 different gauges for Speedometer, RPM and Turbo. With gameObject.GetComponents method we can numerate every speedometer components. For determining right component for right usage, you can use ControlName property for naming the component. In this example i named Speedometer gauge as Speedo, RPM as RPM and turbo as Turbo. And Created 3 different object reference for each. At start, i assign this references with right speedometer component and later, I use their properties.
(This sample created for RGKCar only but same can be applied for UnityCar too)

```
using UnityEngine;
using System.Collections;
using RacingGameKit;
using RacingGameKit.RGKCar;

[AddComponentMenu("Racing Game Kit/Speedometer UI/Speedometer UI - Vehicle Connector
- 3Way"), ExecuteInEditMode]
public class SpeedometerUIVehicleConnector3Way : MonoBehaviour
{
    SpeedometerUI[] SpeedometerIUs;
    RGKCar_Engine oEngine=null;

    SpeedometerUI oSpeedoForSpeed;
    SpeedometerUI oSpeedoForTurbo;
    SpeedometerUI oSpeedoForRPM;
    void Start()
    {
        SpeedometerIUs = base.GetComponents<SpeedometerUI>();
        oEngine = base.GetComponent<RGKCar_Engine>() as RGKCar_Engine;

        foreach (SpeedometerUI oSUI in SpeedometerIUs)
        {
            if (oSUI.ControlName == "Speed")
            {
                oSpeedoForSpeed = oSUI;
```

```
            }
            else if (oSUI.ControlName == "Turbo" )
            {
                oSpeedoForTurbo=oSUI;
            }
            else if (oSUI.ControlName == "RPM")
            {
                oSpeedoForRPM = oSUI;
            }
        }
    }

    void Update()
    {
        if (oEngine!=null){
            if (oSpeedoForSpeed != null)
            {
                oSpeedoForSpeed.Speed = oEngine.SpeedAsKM;
                string gear = oEngine.Gear.ToString();
                if (gear == "-1") gear = "R";
                if (gear == "0") gear = "N";
                oSpeedoForSpeed.Gear = gear;
            }
            if (oSpeedoForRPM!=null) oSpeedoForRPM.Speed = oEngine.RPM;
            if (oSpeedoForTurbo!=null) oSpeedoForTurbo.Speed =
oEngine.TurboFill*100f;
        }
    }

}
```