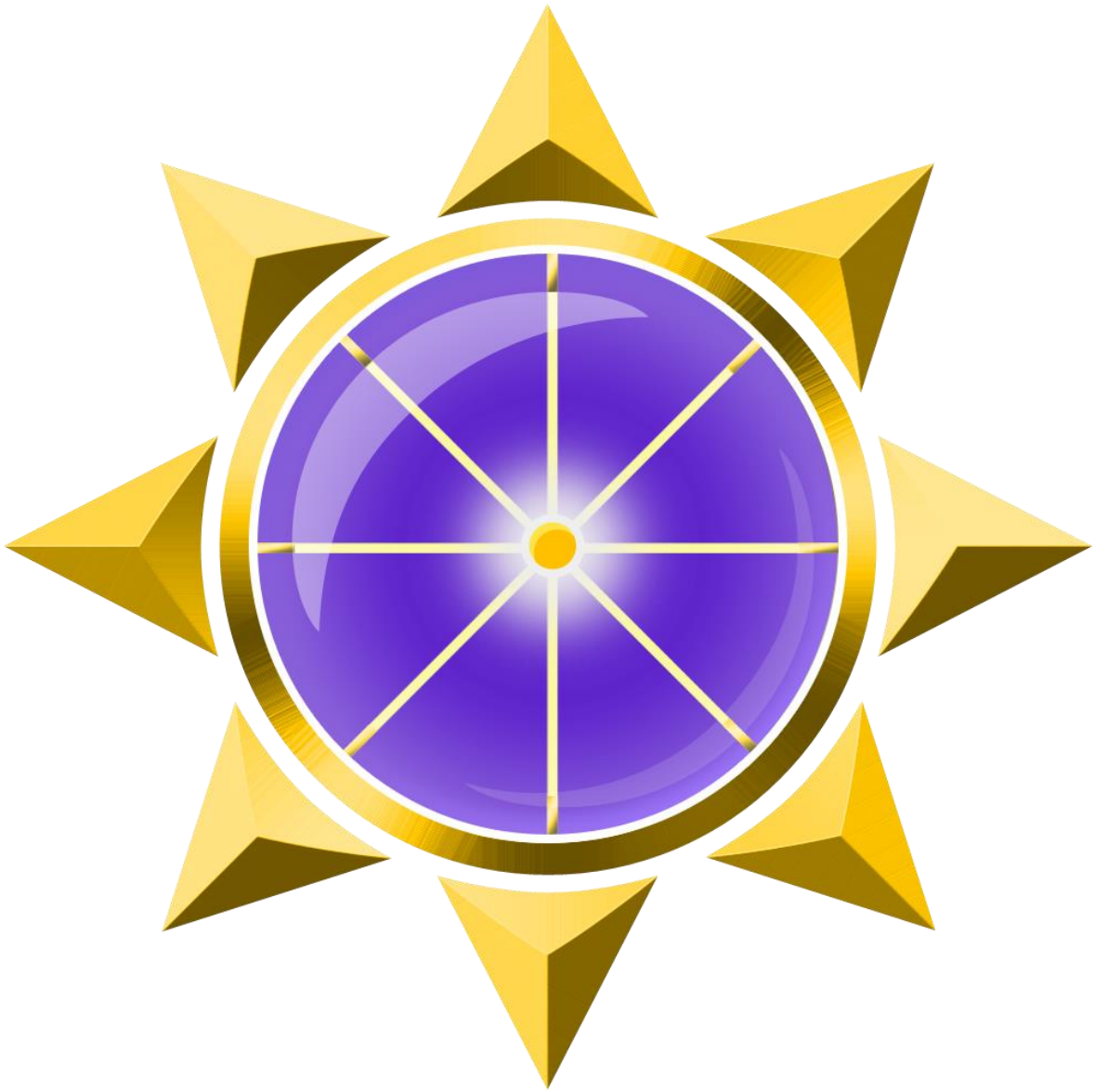
Elemental Meters

By Armathyx (Rafael Gonçalves)



Instruction Manual 1.0

Table of Contents

I. Introduction

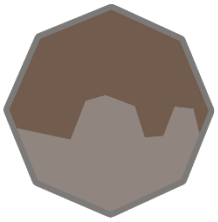
The elemental meters are gauges intended for use with Unity's premade UI tools. They were originally designed as replacements for the more traditional rectangular health, mana or stamina bars, but they may serve other purposes such as loading screen progress bars, in-game clocks, speedometers or even radars.

* * * * * *

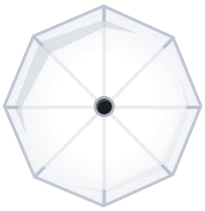
II. Package Contents

This package includes all parts required to setup the various elemental meters. It also includes a preview for each of the premade meters for reference, as well as a demo project. Each meter is built out of three parts:

Frames



Masks



Cases

Many of these parts are interchangeable (you may for instance combine the Sun Meter's case with the Nature Meter's frame and the Water Meter's mask). Additionally, you will need a fill image. A blank sample is included in the pack (\meters > \fill), but it can otherwise be created in Unity. This guide explains how to setup your meter in detail.

III. Importing the assets to Unity

1. In your **Assets** window, right-click anywhere and select "*import new asset…*". Import a frame, mask and case of your choice, as well as the blank fill (circular or hexagonal).

2. Once you've imported the four assets required for setting up the meter, you may adjust their quality in the **Inspector** ("*Max Size*", "*Compression*" …). Increasing these settings will usually ensure the visuals don't lose quality in game.

* * * * *

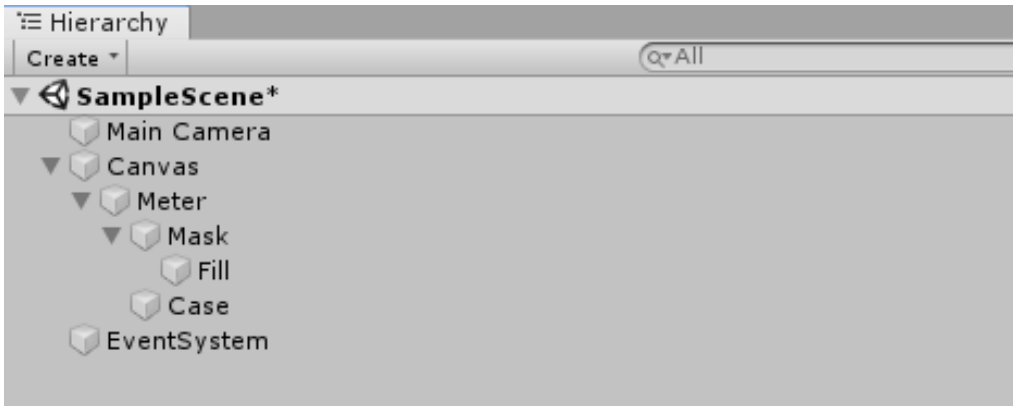
IV. Object hierarchy setup

1. In the **Hierarchy** window, create a new canvas by right-clicking and selecting "**UI**" > "**canvas**".

2. Right-click your canvas object and create a new image within it by selecting "**UI**" > "**image**", name it « *Meter* » (it may be « *health Meter* », « *mana Meter* », or for anything else you'd like to use your meter). This will be your frame.

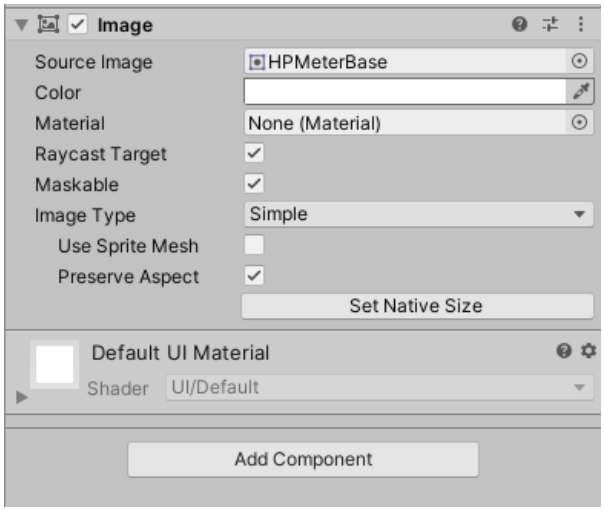
3. Repeat step 2, but this time right-click your "*Meter*" object to create two new child objects, named "*Mask*" and *"Case"*.

4. Finally, create a child object for your "*Mask*", named "*Fill*".



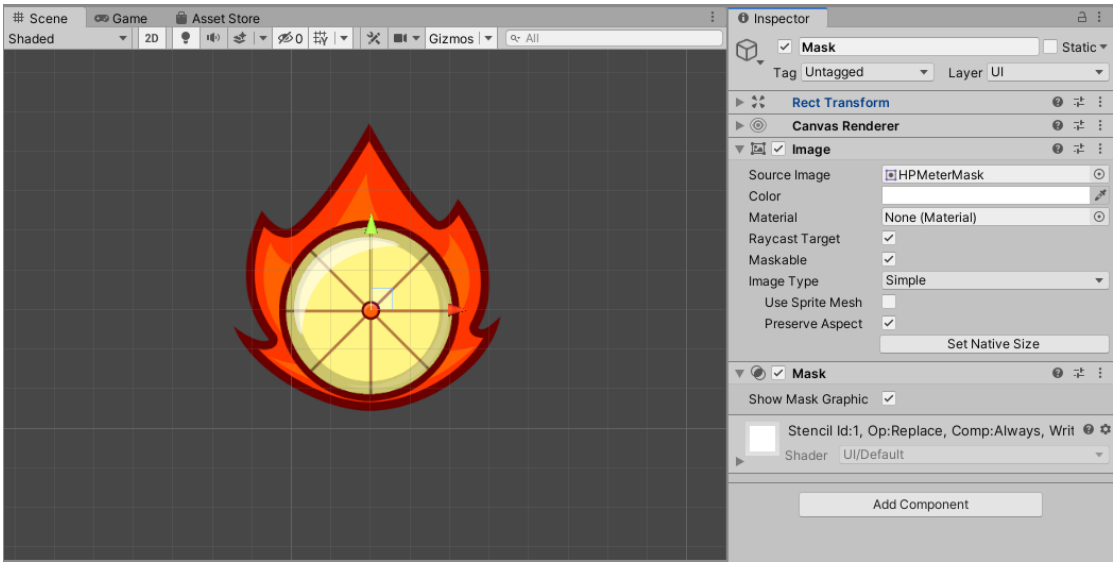
V. Visuals setup

1. Select your "*Meter*" object in the **Hierarchy**. In the **Inspector** window, set its « *source image* » to the frame asset you've imported earlier. If the image appears deformed, tick the « Preserve Aspect » checkbox underneath.

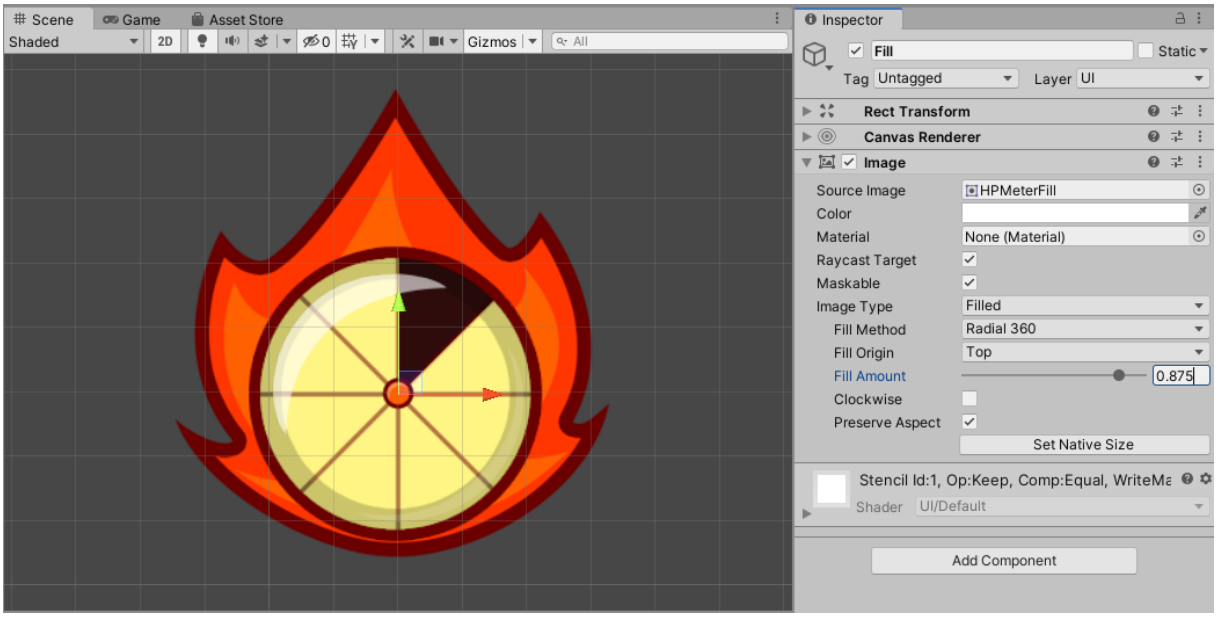


2. Repeat step 1 to assign the assets « BlankFill », « Mask » and « Frame » to the objects « *Fill* », « *Mask* » and « *Frame* » respectively.

3. Select your « *Mask* » object in **Hierarchy** and click « *add component* » in the **Inspector**. Search for « *mask* » and add it. At this point, your fill should be limited to your mask's boundaries. Adjust the size of your mask and fill so that they're positioned correctly inside the frame.



4. Select your "*Fill*" object in the **Hierarchy** and set its "*image type*" to "*filled*" in the **Inspector**. Set the "*fill method*" to "*radial 360*" and "*fill origin*" to "*top*". You should now be able to use the "*fill amount*" slider to test your meter. Adjust the "*fill*" object's position so that it's correctly centred inside the case (try filling it to a quarter, a half, and so on to verify that it stops exactly where the case's section dividers are).



* * * * *

VI. Scripts setup

1. In **Hierarchy**, select your "*Meter*" object. In the **Inspector**, click on "*add component*" and search for "*slider*". Once you've added it, untick "*interactable*" and set "*transition*" to "*none*".

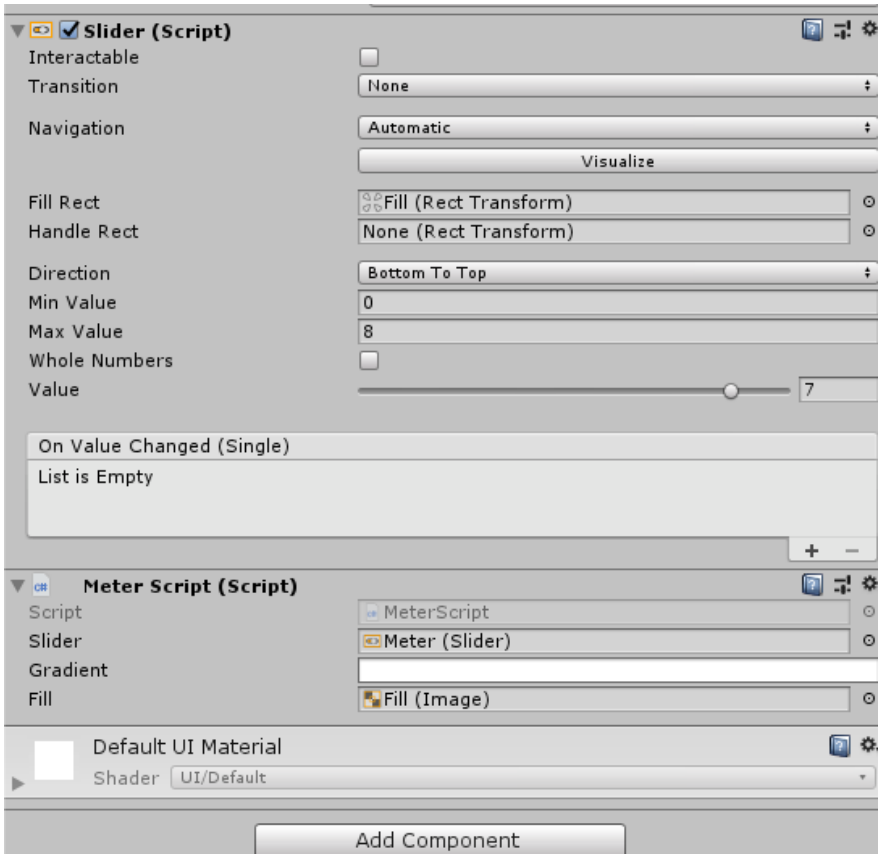
2. Drag and drop your "*Fill*" object from the **Hierarchy** to the "*fill rect*" box in the **Inspector** (you might have to resize your fill again after this, as it'll scale back up to the full size of the mask). You should now be able to control your fill with the "Value" slider.

3. Still in the Inspector, set the "max value" to 8 (in order to match the number of sections in your frame.

4. Add a new script to the "*Meter*" object ("*new component*" > "*new script*"), name it "MeterScript". Fill in the following script. Mind the upper and lowercases, it must be written precisely as follows. For the sake of simplicity, we're naming the variable we're going to be tracking "health", but you can change this later while respecting its lower and uppercases:

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

public class MeterScript : MonoBehaviour
{
    public Slider slider;
    public Gradient gradient;
    public Image fill;

    public void SetMaxHealth(float health)
    {
        slider.maxValue = health;
        slider.value = health;

        fill.color = gradient.Evaluate (1f) ;

    }

    public void SetHealth(float health)
    {
        slider.value = health;
        fill.color = gradient.Evaluate(slider.normalizedValue);

    }

}
```

5. Once the script has been added, you should have three new parameters under your « MeterScript » in the **Inspector**. Drag and drop the slider component (from the **Inspector**) into the « *slider* » bar, and the "*Fill*" object (from the **Hierarchy**) into the « *fill* » bar.



6. Click on the « Gradient » parameter's bar to select the colours of your choice. We will only be able to test the gradient later, when the meter is assigned to an in-game object.

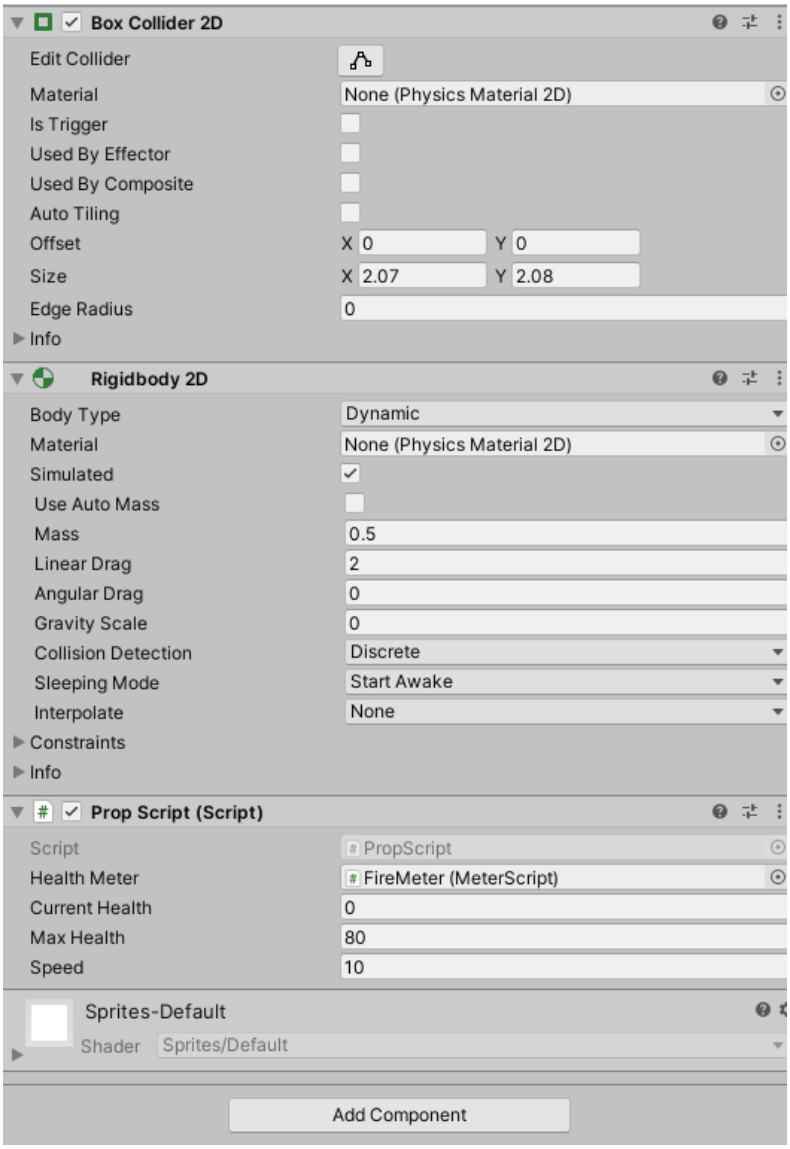
* * * * *

VII. Assigning your meter to an object

Let's say we'd like to use our meter to keep track of your player character's health in game. In your character's script, you will need the following lines of code (marked with "//meter code") in order to link your meter to it:

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

public class PlayerController : MonoBehaviour
{
    private BoxCollider2D propBoxCollider;
    public MeterScript healthMeter; //meter code
    public int currentHealth; //meter code
    public int maxHealth = 80; //meter code
    float movementSpeed = 10f;
    [SerializeField]
    private float speed;
    private Rigidbody2D rb2D;

    // Start is called before the first frame update
    void Start()
    {
        propBoxCollider = GetComponent<BoxCollider2D>();
        rb2D = GetComponent<Rigidbody2D>();
        currentHealth = maxHealth; //meter code
        healthMeter.SetMaxHealth(maxHealth); //meter code

    }

    // Update is called once per frame
    void FixedUpdate()
    {
        healthMeter.SetHealth(currentHealth); //meter code

        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
        Vector2 movement = new Vector2(moveHorizontal, moveVertical);
        rb2D.AddForce(movement * speed);
    }

    private void OnCollisionEnter2D(Collision2D other)
    {
        if (other.gameObject.tag == "DamageSource")
        {
            currentHealth -= 10; //meter code
        }

    }
}
```

1. Once this script is assigned to your character, a new parameter will appear in your character's **Inspector** window named « *Health Meter*». Drag and drop your « Meter » object from the **Hierarchy** into it.



2. We will then need a function that increases or decreases your health meter's fill value. In this example, the OnCollisionEnter2D function at the bottom of the script reduces your character's health by 10, therefore decreasing the health meter's fill by 10.

Your health meter should now be fully functional!

VIII. Troubleshooting & Contact

In this package you'll find a demo containing a fully functional example of this tutorial. Should you run into any problems during the setup, please refer to it for troubleshooting. Additionally, the demo also contains buttons scripted to increase and decrease a meter's fill value, should you wish to explore that functionality.

If you need further assistance, wish to report a bug or would like to make a request for a different style of meter, please contact the following address:

armathyx.games@gmail.com