Ultimate Circular Health Bars Guide

1 CONTENTS

2 INTRODUCTION

Thank you for purchasing the Ultimate Circular Health Bars (UCHB) Asset!

This document is a small guide to help you get started with the asset and contains some useful information for scripting.

This asset **NO LONGER REQUIRES URP OR HDRP! Yay!** I wrote a 600+ line shader to copy everything from Shader graph, so now everything **should be compatible with the Built-in RP AS WELL AS URP and HDRP**.

If you encounter any bugs with this asset, or are having difficulties getting things set up, don't hesitate to contact devorengegames@gmail.com with the details of the issue you are having.

I would very much appreciate it if you could write a short review in the asset store once you've become acquainted with this asset!

Check out my other assets if you like this one: https://assetstore.unity.com/publishers/49336 (as of release of this version, this is still the only asset I have published in the asset store)

I also have a YouTube Channel: DevOrenge - YouTube

I hope you get everything you need out of this asset and out of this document!

3 VERSION HISTORY

3.1 V6.0 – CURRENT

Added Arcing

3.2 V5.0

Added Gradients

3.3 V4.0

Built-In RP is now supported!

> Use whatever RP you want, and the asset should just work. Even if you switch RPs back and forth. Contact me if you experience any issues with this so I can fix them for you!

Custom Inspector for clearer editing.

Manager class where you can access all your health bars at runtime.

Health bars can now be duplicated without any issues.

New demo scene and prefabs with updated assets.

Some sprites to get you started with texturing your health bars.

3.4 V3.0

Added texture support!

You can now add textures to:

The inner health bar, the border, the gaps, the health void, and the entire health bar with an overlay texture

Some issues with the shader were also fixed and the shader graph is now more presentable if you want to check out how this asset works under the hood.

All previous keywords were made into Booleans to prevent too many shader variants from being generated. Five new ones were added for texture support.

3.5 V2.0

The current version of UCHB adds 18 new properties and four keywords to the asset. These are:

Properties: *OverlayColor, BorderColor, EmptyColor, SpaceColor, BorderWidth, BorderSpacing, RemoveBorder, OverlayNoise x 3, EmptyNoise x 3, ContentNoise x 3, PulseSpeed, PulseActivationThreshold*

Keywords: *OverlayNoiseEnabled, EmptyNoiseEnabled, ContentNoiseEnabled, PulsateWhenLow*

Other than that, various bugs were fixed, including some aliasing artifacts and some issues with the script.

3.6 V1.0

The initial version of UCHB included seven customizable properties for the shader. These properties were: *SegmentCount, RemovedSegments, Color, Spacing, Radius, LineWidth and Rotation*

4 UPGRADING FROM VERSION 3 OR EARLIER TO VERSION 4 OR LATER

There are a few major changes you have to consider before upgrading to Version 4.

Firstly, and most importantly, **all your health bars will likely be reset to default when you upgrade**, since the way properties are stored has changed significantly, and Unity may not know what to do with your data. **If you are not willing to remake your health bar designs, you should not upgrade to this version!**

Next, you will have to update your references. You can no longer type:

```
<healthBar>.<property> = <value>;
```

You'll have to write:

```
<healthBar>.<property>.Value = <value>;
```

It's a small change, but important to keep in mind, because you will see loads of errors in your scripts that reference health bar properties - if you upgrade - that will require this specific fix.

5 FILES

The base Materials can be found in the **Resources** folder of this asset. There you will also find a blank sprite (among other things). This sprite is used to ensure any SpriteRenderer you are trying to assign this asset to is able to be rendered correctly.

The **ShaderGraphs** folder contains the primary ShaderGraph and SubGraphs.

The **Shaders** folder contains the Built-in RP shader.

The **Editor** folder contains the custom inspector script.

The **Scripts** folder contains the scripts used to populate a given GameObject with the correct Material and which allow you to animate the properties of the UCHB Material.

The **Prefabs** folder is where you'll find finished GameObjects for UI (Image) or non-UI (SpriteRenderer) purposes which can simply be added to your scene.

The **DemoScene** folder contains the demo scene and its corresponding assets.

6 USING THIS ASSET

This asset comes with an easy-to-use script which can be added to any GameObject which has a **SpriteRenderer** or an **Image** component attached to it.

Simply add the script **RadialSegmentedHealthBar** to your desired GameObject and you are good to go!

If something goes wrong, you'll get an error message in the console and the script will remove itself from the GameObject. Everything should work fine, provided you **haven't** removed any essential files from the asset.

6.1 MODIFYING PROPERTIES AND KEYWORDS DIRECTLY

The script has **64 public properties** which can be used for animation or as an easy way to set the UCHB Material properties. To use these, you simply need a reference to the RadialSegmentedHealthBar component and then you can access them like:

```
public RadialSegmentedHealthBar healthBar;
```

…

```
healthBar.<Property>.Value = <value>;
```

There are some public methods to make setting the SegmentCount and RemovedSegments a little easier:

```
healthBar.SetSegmentCount(<float value>);
```

Set the number of segments in this health bar. Minimum for this is 0.

`healthBar.SetRemovedSegments(<float value>);`

Sets the absolute count of removed segments. The result is clamped to 0 and the Maximum value.

`healthBar.SetPercent(<float value>);`

Sets the absolute percentage (0 to 1) of the health bar. The input is clamped to 0 and 1.

`healthBar.AddRemoveSegments(<float value>);`

Add or remove (+ or -) a certain amount of segments from/to the health bar.
This does not alter the segment count. The result is clamped to 0 and the Maximum value.

`healthBar.AddRemovePercent(<float value>);`

Add or remove (+ or -) a certain percent (0 to 1) of the health bar.
This does not alter the maximum value. The result is clamped to 0 and the Maximum value.

The shader has **69 public properties and 5 keywords**[1]. The **properties** can be set like so:

`material.SetFloat(Shader.PropertyToID("<ShaderProperty>"), <value>);`

`material.SetColor(Shader.PropertyToID("<ShaderProperty>"), <value>);`

`material.SetVector(Shader.PropertyToID("<ShaderProperty>"), <value>);`

`material.SetTexture(Shader.PropertyToID("<ShaderProperty>"), <value>)`

The **keywords** can be set like so:

`material.EnableKeyword(<Keyword>);`

`material.DisableKeyword(<Keyword>);`

[1] Keywords are used to enable and disable certain features in a shader.

6.2 THE MANAGER CLASS

There is a new class which makes it easier to access and modify all your health bars!

It's the **StatusBarsManager** class.

When you create a health bar, you need to enter two names at the very top of the inspector.

The parent name: -> **ParentName**

> This name is the parent for a set of health bars. A good example for this is "Player". If you modify a property using only the parent name, ALL properties with that property name of all children will be modified.

The health bar name: -> **Name**

> This name is for the health bar itself. Multiple health bars can have the same name and parent name. If you modify a health bar from StatusBarsManager with a parent name and health bar name, ALL health bars with that name combo will be modified.

Now that that is hopefully clear, let's move on to how you actually access and modify health bars from the manager.

StatusBarsManager.GetHealthBar(<parentName>, <name>);

> Returns all health bars with the given name combo as a **List<ISegmentedHealthBar>**

Other methods all have the syntax:

StatusBarsManager.<SetterMethod>(<parentName>,[name],<value>);

StatusBarsManager.<GetterMethod>(<parentName>,[name], out <value>);

> Where [name] is an optional overload.

If you don't want to use the Manager, you can disable it (and enable it) by calling:

StatusBarsManager.Disable();

StatusBarsManager.Enable();

And there's a final method you can use to clear the StatusBarsManager of all health bars:

StatusBarsManager.Clear();

Finally, the **AddHealthBar** and **RemoveHealthBar** methods are all used internally and, in most cases, shouldn't be used by the user.

6.3 DATA FIELDS

For a more concise version of this list, go to the last chapter of this guide.

Following Notation:

<ScriptProperty> -> <**ShaderProperty or Keyword**>

SegmentCount -> **_SegmentCount**

This is used to define how many segments the health bar has. This is a float value to allow for more flexibility, however, it is recommended to only use integers for this field to avoid funky looking health bars. Use a value of 1 to define a monolithic health bar with only one segment. Don't use 0…

RemovedSegments -> **_RemoveSegments**

How many segments have been removed from the health bar's total segment count. This is also a float value, here you can input something like 1.5 to remove 1.5 segments. It is also worth noting that if you have 1 set for the segment count this value acts like a percentage with 0 being 0% removed and 1 being 100% removed.

6.4 COLOR FIELDS

OverlayColor -> **_OverlayColor**

The overlay color for the entire health bar

InnerColor -> **_InnerColor**

The color for the value part of the health bar.

BorderColor -> **_BorderColor**

The health bar's border color

EmptyColor -> **_EmptyColor**

The color of the space where health has been removed

SpaceColor -> **_SpaceColor**

The color of the segment gaps, or the color between the segments.

6.5 General Settings

Spacing -> **_SegmentSpacing**

The amount of spacing between each segment.

Arc -> **_Arc**

The amount of arcing on the health bar

Radius -> **_Radius**

The relative size of the health bar. Values greater than 0.5 usually make the health bar exceed the bounds of the sprite.

Line Width -> **_LineWidth**

The thickness of the health bar. Setting a low value for the radius and cranking up the line width results in a pie looking health bar.

Rotation -> **_Rotation**

The rotation of the health bar in degrees.

Offset -> **_Offset**

How much the health bar is offset on the x and y axes. This is useful when combining a small arc with a large radius, to keep the health bar in visible range.

6.6 Border Settings

BorderWidth -> **_BorderWidth**

The border width.

BorderSpacing -> **_BorderSpacing**

The spacing relative to line spacing. 0 is flush with the primary spacing property, and everything above that closes the border spacing gap until there is no gap at all.

RemoveBorder -> **_RemoveBorder**

The inverted opacity of the border in the region where health has been removed 0 is fully visible and 1 is fully transparent. This also removes the empty space which would be normally filled by *EmptyColor.*

6.7 Noise Settings

OverlayNoiseEnabled -> **_OverlayNoiseEnabled**

This is a Boolean to disable or enable overlay noise. It is the noise that encompasses the entire health bar. Booleans are represented in the shader as floats.

EmptyNoiseEnabled -> **_EmptyNoiseEnabled**

This Boolean or disables empty noise. It is the noise that fills the empty space where health has been removed. Booleans are represented in the shader as floats.

ContentNoiseEnabled -> **_ContentNoiseEnabled**

This Boolean enables or disables content noise. It is the noise that fills the inner health bar. Booleans are represented in the shader as floats.

OverlayNoiseScale -> **_OverlayNoiseScale**

EmptyNoiseScale -> **_EmptyNoiseScale**

ContentNoiseScale -> **_ContentNoiseScale**

Set the scale of each noise type.

OverlayNoiseStrength -> **_OverlayNoiseStrength**

EmptyNoiseStrength -> **_EmptyNoiseStrength**

ContentNoiseStrength -> **_ContentNoiseStrength**

Set the intensity of the noise. When this is set to a value greater than one, the corresponding area begins to fade until it is completely transparent.

OverlayNoiseOffset -> **_OverlayNoiseOffset**

EmptyNoiseOffset -> **_EmptyNoiseOffset**

ContentNoiseOffset -> **_ContentNoiseOffset**

The noise offset for each noise area. This can be used add a fog-like appearance to the health bar when animated.

6.8 PULSE SETTINGS

PulsateWhenLow -> **_PulsateWhenLow**

Boolean to enable pulsating when the health bar value is low. Booleans are represented in the shader as floats.

PulseSpeed -> **_PulseSpeed**

The speed of pulsating.

PulseActivationThreshold -> **_PulseActivationThreshold**

How much health the player needs to activate the pulsating. 0 is no health 1 is full health.

6.9 TEXTURE SETTINGS

InnerTextureEnabled -> **INNER_TEXTURE_ON**

OverlayTextureEnabled -> **OVERLAY_TEXTURE_ON**

BorderTextureEnabled -> **BORDER_TEXTURE_ON**

EmptyTextureEnabled -> **EMPTY_TEXTURE_ON**

SpaceTextureEnabled -> **SPACE_TEXTURE_ON**

Five keywords to determine whether textures will be used or not.

AlignInnerTexture -> **_AlignInnerTexture**

AlignBorderTexture -> **_AlignBorderTexture**

AlignEmptyTexture -> **_AlignEmptyTexture**

AlignSpaceTexture -> **_AlignSpaceTexture**

Determines whether the textures follow the flow of the radial health bar (true) or if they are tiled normally (false).

InnerTextureScaleWithSegments -> **_InnerTextureScaleWithSegments**

BorderTextureScaleWithSegments -> **_BorderTextureScaleWithSegments**

EmptyTextureScaleWithSegments -> **_EmptyTextureScaleWithSegments**

Determines whether the tiling of some textures is tied to the segment count (true) or not (false).

InnerTextureOpacity -> **_InnerTextureOpacity**

OverlayTextureOpacity -> **_OverlayTextureOpacity**

BorderTextureOpacity -> **_BorderTextureOpacity**

EmptyTextureOpacity -> **_EmptyTextureOpacity**

SpaceTextureOpacity -> **_SpaceTextureOpacity**

How much the texture is visible. This isn't transparency, it's just a lerp between a solid color and the texture (0 is solid color, 1 is full texture)

InnerTextureTiling -> **_InnerTextureTiling**

OverlayTextureTiling -> **_OverlayTextureTiling**

BorderTextureTiling -> **_BorderTextureTiling**

EmptyTextureTiling -> **_EmptyTextureTiling**

SpaceTextureTiling -> **_SpaceTextureTiling**

The tiling of the textures. If the texture is aligned (e.g. AlignInnerTexture = true), then the tiling usually only works on one axis.

InnerTextureOffset -> **_InnerTextureOffset**

OverlayTextureOffset -> **_OverlayTextureOffset**

BorderTextureOffset -> **_BorderTextureOffset**

EmptyTextureOffset -> **_EmptyTextureOffset**

SpaceTextureOffset -> **_SpaceTextureOffset**

The texture offset. If the texture is aligned (e.g. AlignInnerTexture = true), then the offset usually only works on one axis. Offset can be used to animate a tiled texture to display a loading bar for example.

6.10 Gradient Settings

InnerGradient -> **_InnerGradient**

The gradient to be applied to the health bar. Shader graph and shaders in general don't support importing gradients, so this property is a texture. A helper function exists to convert from a gradient to a texture so you can simply call:

material.SetTexture("_InnerGradient", gradient.ToTexture2D());

InnerGradientEnabled -> **_InnerGradientEnabled**

Enable or disable the use of the inner gradient. **This has no effect if the inner texture is disabled.**

ValueAsGradientTimeInner -> **_ValueAsGradientTimeInner**

Use the health bar value as the gradient evaluator.

EmptyGradient -> **_EmptyGradient**

The gradient to be applied to the **depleted portion** of the health bar. Shader graph and shaders in general don't support importing gradients, so this property is a texture. A helper function exists to convert from a gradient to a texture so you can simply call:

material.SetTexture("_InnerGradient", gradient.ToTexture2D());

EmptyGradientEnabled -> **_EmptyGradientEnabled**

Enable or disable the use of the empty gradient. **This has no effect if the empty texture is disabled.**

ValueAsGradientTimeEmpty -> **_ValueAsGradientTimeEmpty**

Use the health bar value as the gradient evaluator.

7 SHADER REFERENCE

Following Notation:

<ScriptProperty> -> <**ShaderProperty or Keyword**>

7.1 PROPERTIES

SegmentCount -> **_SegmentCount**

RemovedSegments -> **_RemoveSegments**

OverlayColor -> **_OverlayColor**

InnerColor -> **_InnerColor**

BorderColor -> **_BorderColor**

EmptyColor -> **_EmptyColor**

SpaceColor -> **_SpaceColor**

Spacing -> **_SegmentSpacing**

Arc -> **_Arc**

Radius -> **_Radius**

LineWidth -> **_LineWidth**

Rotation -> **_Rotation**

Offset -> **_Offset**

BorderWidth -> **_BorderWidth**

BorderSpacing -> **_BorderSpacing**

RemoveBorder -> **_RemoveBorder**

OverlayNoiseEnabled -> **_OverlayNoiseEnabled**

EmptyNoiseEnabled -> **_EmptyNoiseEnabled**

ContentNoiseEnabled -> **_ContentNoiseEnabled**

OverlayNoiseScale -> **_OverlayNoiseScale**

EmptyNoiseScale -> **_EmptyNoiseScale**

ContentNoiseScale -> **_ContentNoiseScale**

OverlayNoiseStrength -> **_OverlayNoiseStrength**

EmptyNoiseStrength -> **_EmptyNoiseStrength**

ContentNoiseStrength -> **_ContentNoiseStrength**

OverlayNoiseOffset -> **_OverlayNoiseOffset**

EmptyNoiseOffset -> **_EmptyNoiseOffset**

ContentNoiseOffset -> **_ContentNoiseOffset**

PulsateWhenLow -> **_PulsateWhenLow**

PulseSpeed -> **_PulseSpeed**

PulseActivationThreshold -> **_PulseActivationThreshold**

AlignInnerTexture -> **_AlignInnerTexture**

AlignBorderTexture -> **_AlignBorderTexture**

AlignEmptyTexture -> **_AlignEmptyTexture**

AlignSpaceTexture -> **_AlignSpaceTexture**

InnerTextureScaleWithSegments -> **_InnerTextureScaleWithSegments**

BorderTextureScaleWithSegments -> **_BorderTextureScaleWithSegments**

EmptyTextureScaleWithSegments -> **_EmptyTextureScaleWithSegments**

InnerTextureOpacity -> **_InnerTextureOpacity**

OverlayTextureOpacity -> **_OverlayTextureOpacity**

BorderTextureOpacity -> **_BorderTextureOpacity**

EmptyTextureOpacity -> **_EmptyTextureOpacity**

SpaceTextureOpacity -> **_SpaceTextureOpacity**

InnerTextureTiling -> **_InnerTextureTiling**

OverlayTextureTiling -> **_OverlayTextureTiling**

BorderTextureTiling -> **_BorderTextureTiling**

EmptyTextureTiling -> **_EmptyTextureTiling**

SpaceTextureTiling -> **_SpaceTextureTiling**

InnerTextureOffset -> **_InnerTextureOffset**

OverlayTextureOffset -> **_OverlayTextureOffset**

BorderTextureOffset -> **_BorderTextureOffset**

EmptyTextureOffset -> **_EmptyTextureOffset**

SpaceTextureOffset -> **_SpaceTextureOffset**

InnerGradient -> **_InnerGradient**

InnerGradientEnabled -> **_InnerGradientEnabled**

ValueAsGradientTimeInner -> **_ValueAsGradientTimeInner**

EmptyGradient -> **_EmptyGradient**

EmptyGradientEnabled -> **_EmptyGradientEnabled**

ValueAsGradientTimeEmpty -> **_ValueAsGradientTimeEmpty**

7.2 KEYWORDS

InnerTextureEnabled -> **INNER_TEXTURE_ON**

OverlayTextureEnabled -> **OVERLAY_TEXTURE_ON**

BorderTextureEnabled -> **BORDER_TEXTURE_ON**

EmptyTextureEnabled -> **EMPTY_TEXTURE_ON**

SpaceTextureEnabled -> **SPACE_TEXTURE_ON**