Read Me

Thank you for purchasing Destructible 2D!

If you haven't already, please consider writing a review. They really help me out, and I read and respond to every one!

If you have any questions, feel free to email me: carlos.wilkes@gmail.com

You can also post to the forum thread: http://forum.unity.com/threads/248661

You can also find me on Twitter: @CarlosWilkes

You can also find me on YouTube: http://www.youtube.com/user/CarlosWilkes

I'm also available for freelance work if you need help with your projects.

Making a Destructible Sprite

To turn a normal sprite into a destructible sprite, you have to:

1 - Drag and drop your sprite into the scene.
2 - Find the **SpriteRenderer** component on your GameObject.
3 - Open the context menu for it (gear icon at the top right).
4 - Select "**Make Destructible**".

Your sprite GameObject will now have the **D2dDestructibleSprite** component attached with basic settings.

Adding Colliders

A destructible sprite is incompatible with Unity's normal 2D colliders (e.g. PolygonCollider2D).

To add colliders, you must use either the **D2dPolygonCollider** component, or the **D2dEdgeCollider** component. This can be done manually, or by pressing the **"+ Polygon Collider"** or **"+Edge Collider"** buttons.

NOTE: D2dEdgeCollider will only work with non-moving GameObjects like the ground. For moving (e.g. physics) objects, you must use the D2dPolygonCollider component.

Optimizing Performance

The best way to optimize performance of your destructible sprites is to use less pixels. If you click the **"Optimize"** button on the **D2dDestructibleSprite** component, then the width & height of your alpha sprite will be halved. This increases the performance of your sprite by 4x, and can be done multiple times. However, each time you optimize it, the visual quality of the destruction will reduce.

NOTE: If you've optimized your sprite too many times, then clicking the **"Rebuild"** button will reset it back to its original state.

You can also optimize your colliders by increasing the **Straighten** setting, don't set this too high though, otherwise you will lose collider accuracy.

Splitting/Slicing Parts Off

By default, a destructible sprite will remain as one single GameObject, even if you slice it in half.

If you want sliced parts to fall off, then you need to add the **D2dSplitter** component. This can be done manually, or by pressing the **"+Splitter"** button.

NOTE: Splitting is a complex operation, and may reduce the performance of your game. To speed this up, make sure you **Optimize** your sprite as much as possible. If your destructible sprite is large then consider splitting it up into multiple smaller parts, or designing your game in such a way that avoids having large singular splittable objects.

Using Semi-Transparent Sprites

By default, destructible sprites read the destruction state/shape from the **Sprite** setting of your **SpriteRenderer** component. However, if your sprite is already semi-transparent then it may not work or render correctly, because the semi-transparent areas will already have some damage due to the reduced alpha.

To fix this, you need to make an alternate/secondary sprite that doesn't have any semi-transparent areas in areas that you want to be solid. Once created, you can drag and drop this sprite into the **Shape** setting of your **D2dDestructibleSprite** component.

Next, you need to replace the **Material** in your **SpriteRenderer** component with the **"Keep Alpha"** material. This material uses both the destruction alpha data, and combines it with the visual sprite's alpha.

Attaching GameObjects to Splittable Destructible Sprites

When a destructible sprite splits in half using the **D2dSplitter** component, it will be cloned/instantiated to make the other side, and **D2dSplitter** will automatically separate the destruction data between the two destructible sprites. This works well for simple destructible sprites, but if your destructible sprite has child GameObjects, then they will also be cloned, and multiply. This will look incorrect because you will now have two copies of your child GameObject, one on the original, and one on the clone. To fix this, you can use the **D2dFixture** component.

To add a fixture you can manually add a child GameObject with the **D2dFixture** component, or open the context menu for the **D2dDestructibleSprite** component, and select the **"Add Fixture"** option. The **D2dFixture** component automatically detects when the current (parent) destructible sprite is splitting, and forces it to stick to only one destructible sprite after the split.

NOTE: The **D2dFixture** component will also destroy itself if/when the pixel underneath it gets destroyed, or if it gets moved off of a solid pixel.

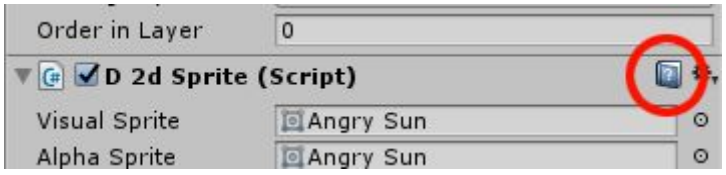
More Tutorials & Demos

For more information and examples of Destructible 2D features I recommend you check the "**Destructible 2D/Examples/Tutorial**" folder. Inside this folder you will see many scenes that go step-by-step through all of the features in Destructible 2D.

If you want to see game examples then check out all the other folders inside **"Destructible 2D/Examples/"**.

Component Documentation

To view documentation about each component in Destructible 2D, simply click the Documentation button at the top right of each component.



Why Does My Sprite Change to a "(Clone)"?

By default, the **D2dDestructibleSprite** component enables the **Crop Sprite** setting. This setting automatically duplicates your original sprite (in the **Sprite Renderer** compoenent), and replaces it with cropped version. This cropped version trims/removes the pixels that no longer exist due to splitting or trimming. This is useful because it reduces overdraw/fillrate, which improves rendering performance if you split or fracture your sprite a lot.

To test this, you can click the render mode dropdown at the top left of the **Scene** tab, and select **Overdraw**. The brighter the pixel, the more overdraw you have. If you disable the **Crop Sprite** setting and split/fracture your sprite a lot, then you will see there will be a lot of overdraw, wasting performance.

I recommend you keep the **Crop Sprite** setting enabled, but in some circumstances you may want to disable it. For example, if you're using sprite animation then keeping this enabled can give you some performance and gcalloc overhead, you may also want to disable it if your sprite has custom components on it that expect the original sprite to always be there.