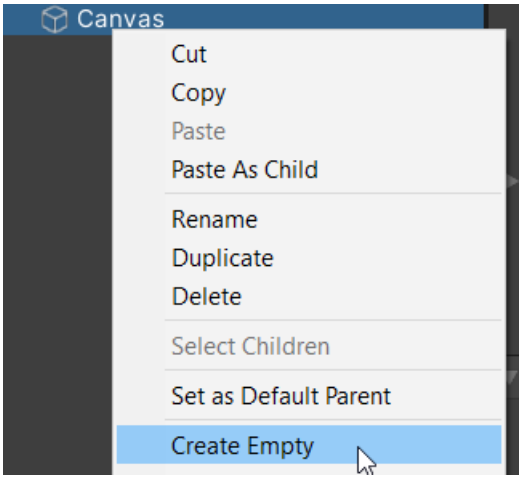
Manual

Table of contents

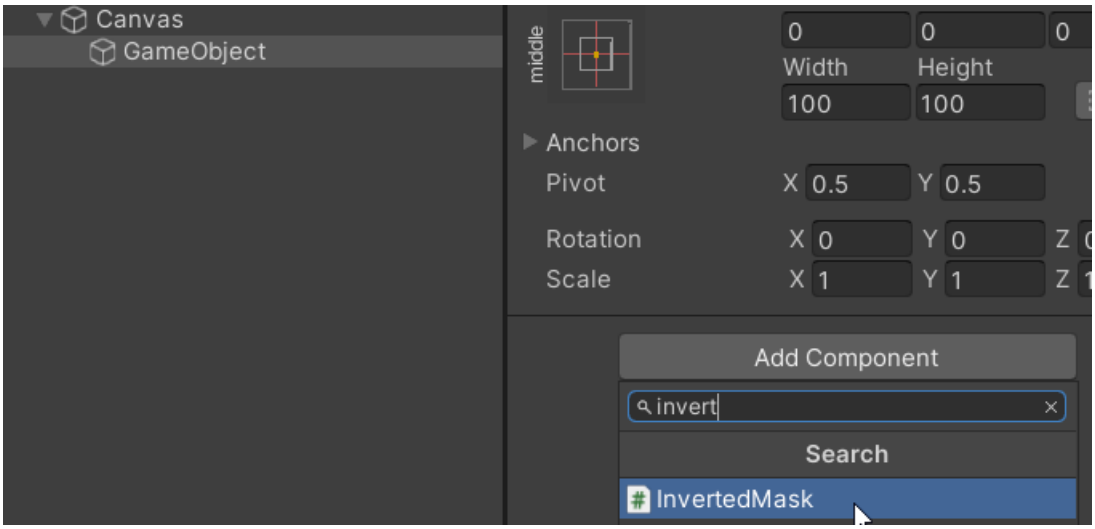
Adding an InverseMask

The inverse mask script uses the IMaterialModifier interface to change the stencil values of the materials of its children (InvertedMaskHole and InvertedMaskGraphic).

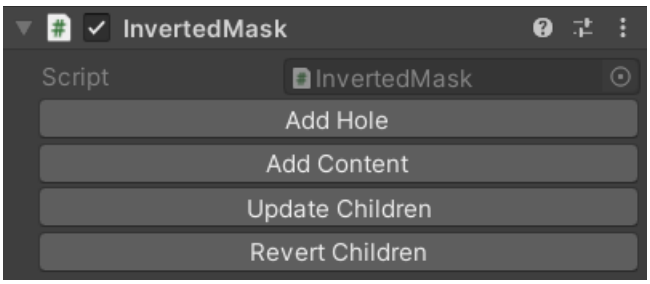
Let's start by using an empty canvas. Within that canvas please create an empty object.



Now add the InvertedMask Component to the new „GameObject"

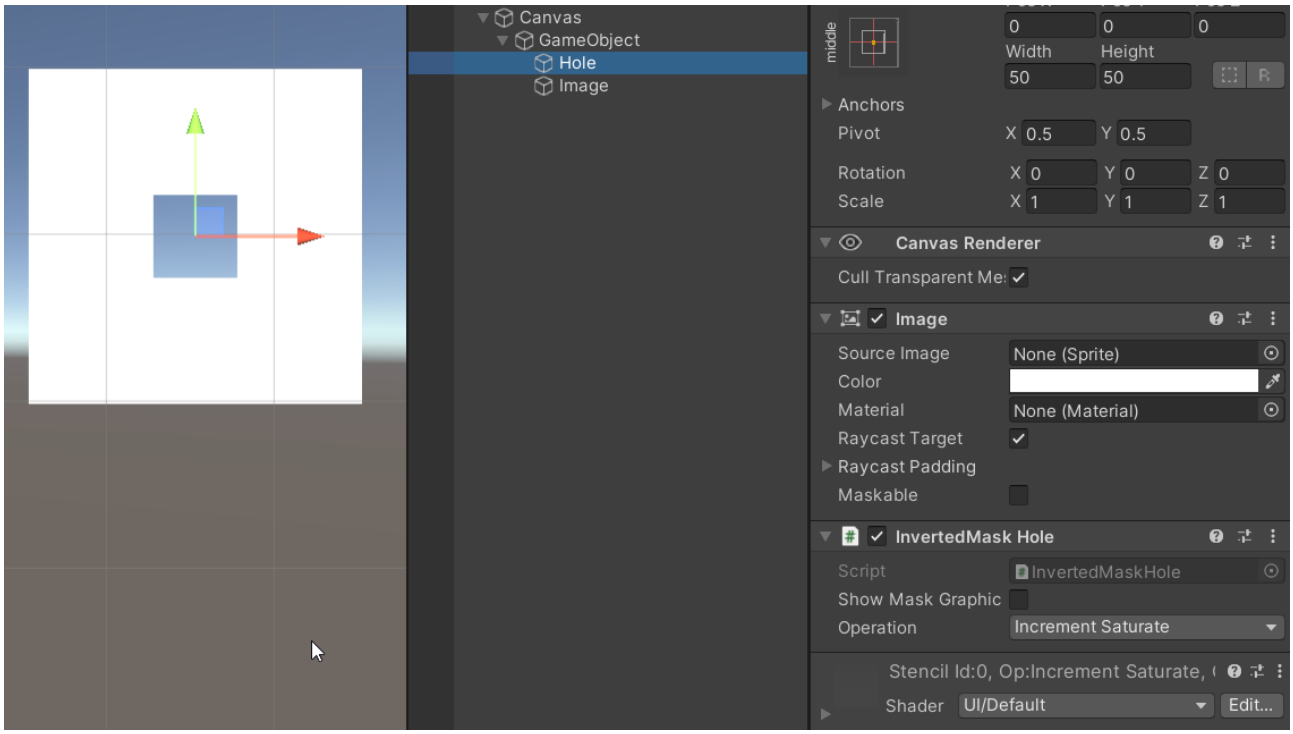


You will see some buttons.



Press „Add Content" and then „Add Hole".

Now it should loook like this:



And that's it.

The „Add ..." buttons are actually just a shortcuts to adding an empty game object and attaching an Image component and a InverteMaskHole/InvertedMaskGraphic component.

InvertedMaskHole

You will notice that each Hole has an „InvertedMaskHole" component attached.

InvertedMaskHole implements the IMaterialModifier interface and controls the stencil value. Holes will get an increased stencil value.

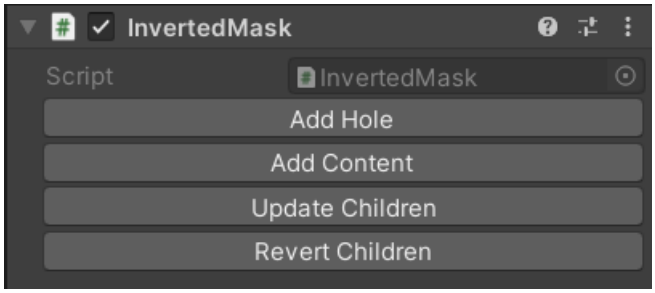
InvertedMaskGraphic

The InvertedMask will automatically add an „InvertedMaskGraphic" component to any graphic within its children. You don't have to add them manually as with many children this would be cumbersome.

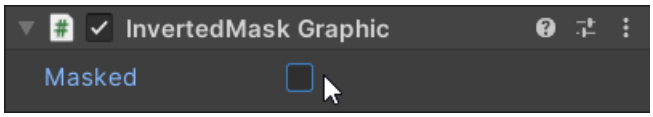
InvertedMaskGraphic implements the IMaterialModifier interface and it sets the stencil compare to „equal or above". This results in leaving out the holes from rendering as the higher stencil value pixels from the holes are no longer drawn.

Removing / Resetting an InverseMask

If you want to remove all InvertedMaskGraphic components within a mask you can use press the **Revert Children** button.



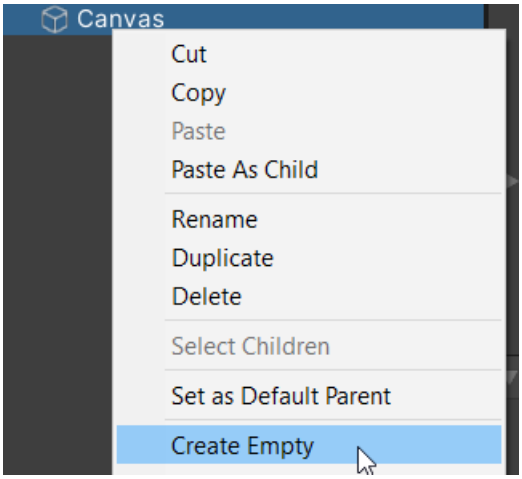
Set the „Masked" toggle to false to exclude a single object from being masked.



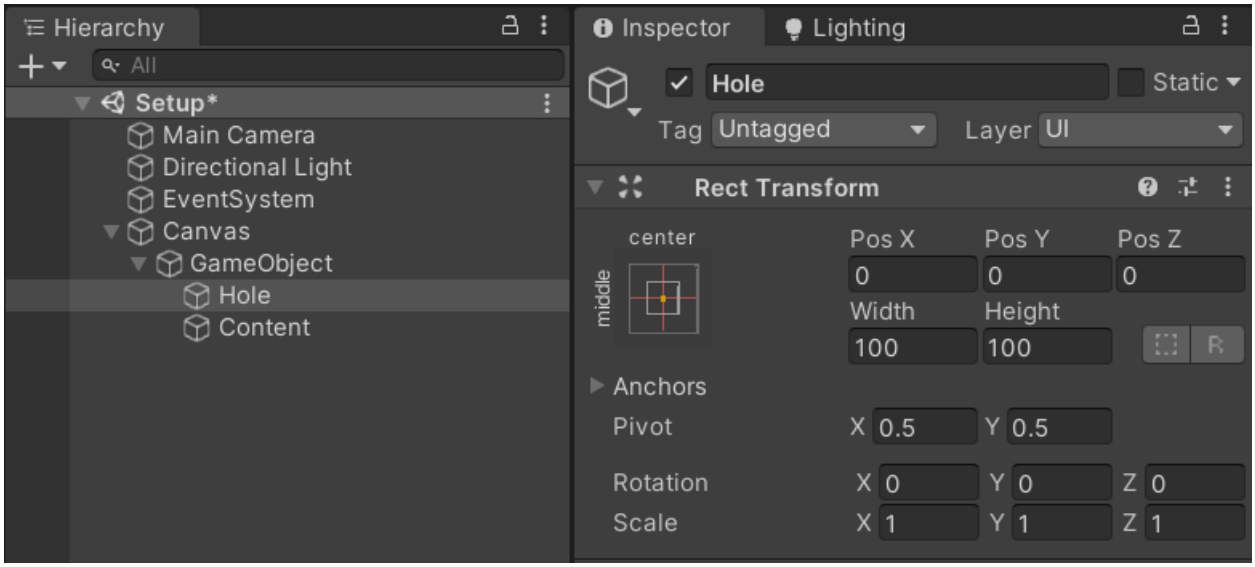
The No-Code Option

If you do not like all these automagic scripts doing stuff with your materials then you can use the „No-Code“ option instead. Here is how to use it:

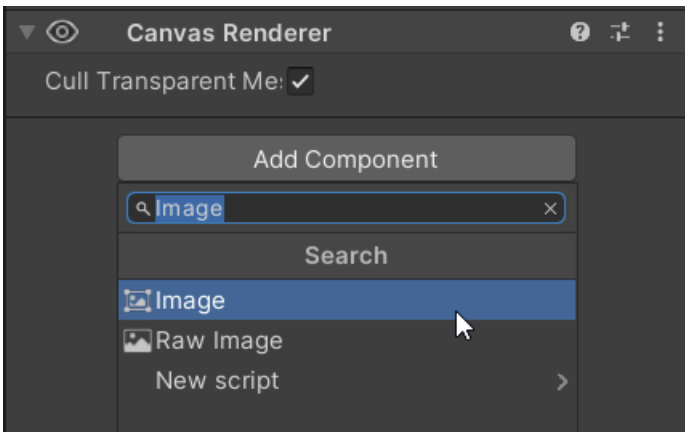
Let’s start by using an empty canvas. Within that canvas please create an empty object.



Now within that we add two new empty objects (let’s call them „**Hole**“ and „**Content**“).

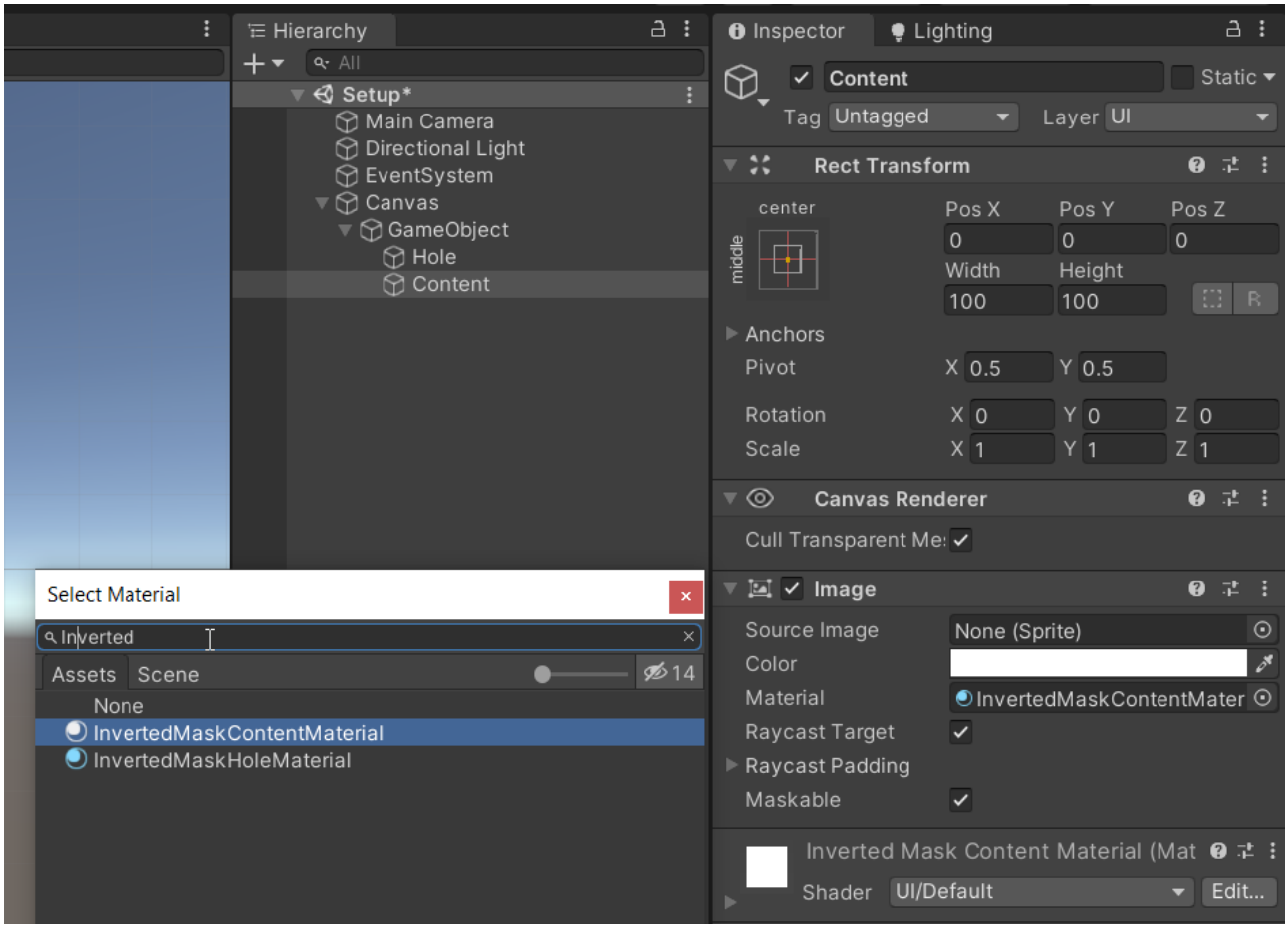


Now add an Image Component to both of them.

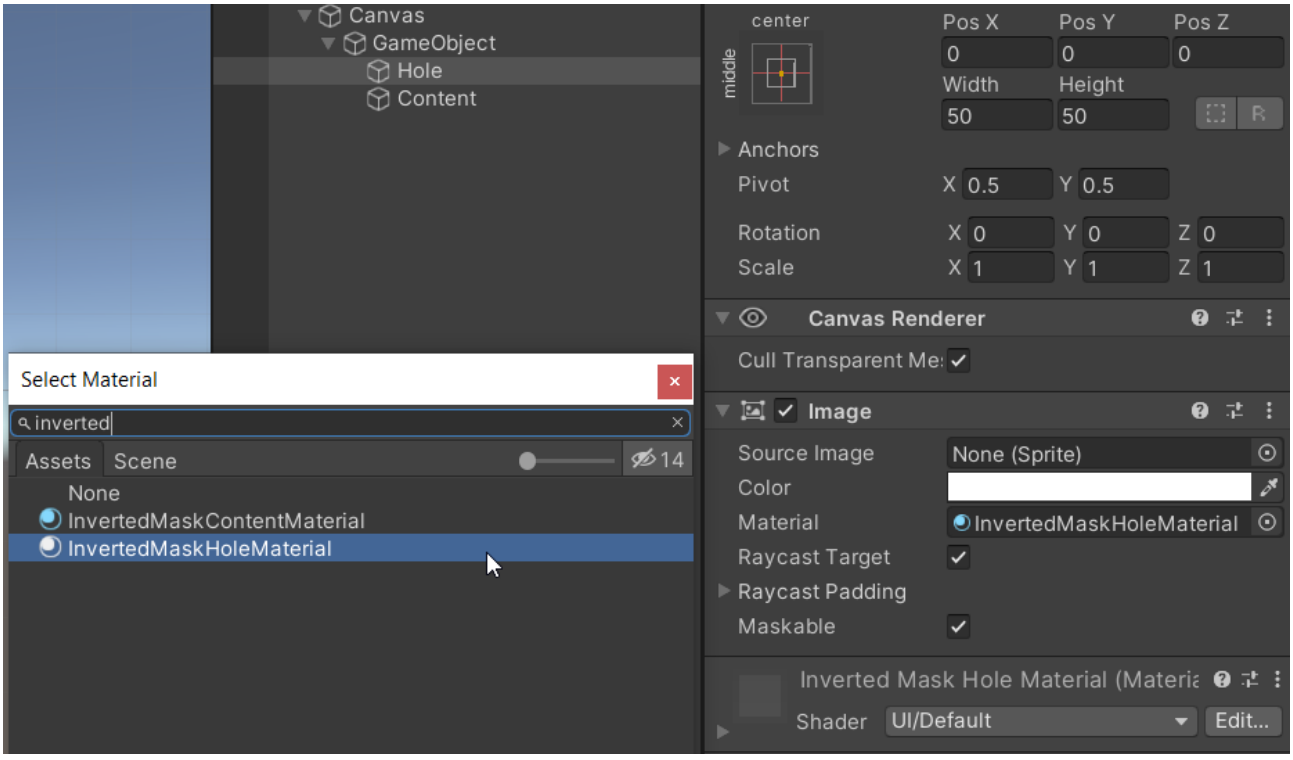


Now we start to apply the masking.

First we need to add the **InvertedMaskContentMaterial** to ALL the things we want to be masked (i.e. the **Content**).

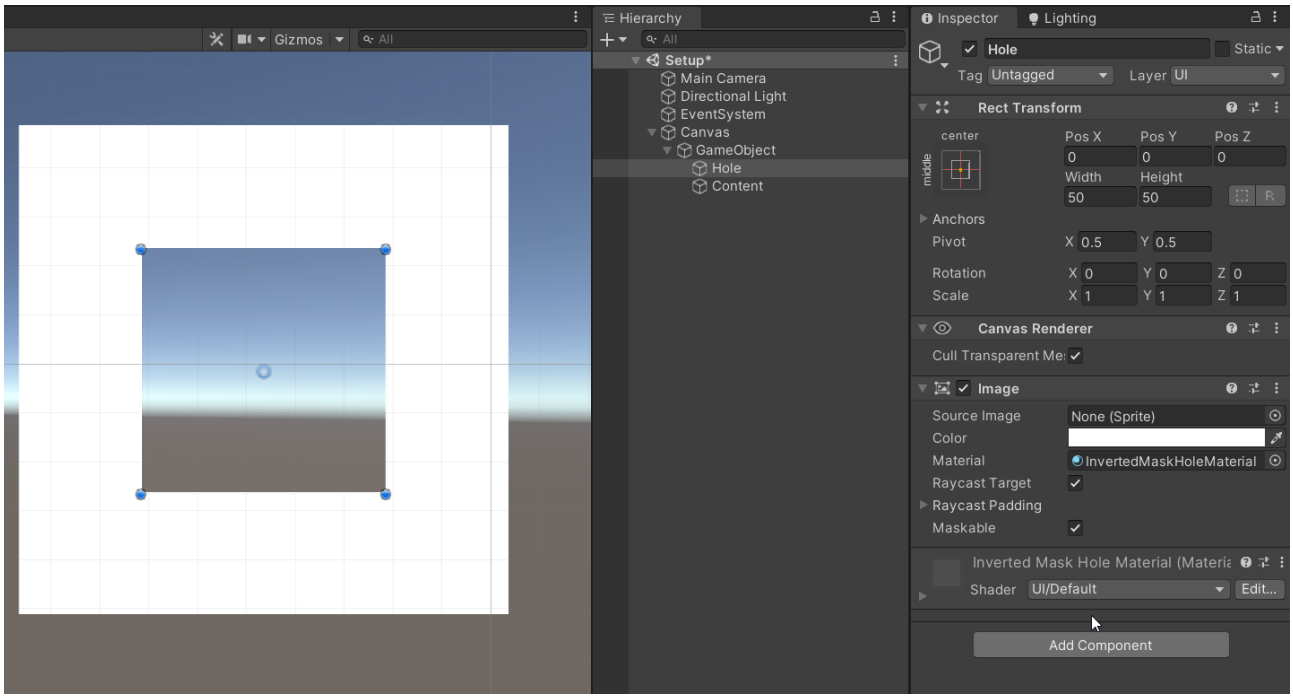


Second we need to add the **InvertedMaskHoleMaterial** to the **Hole** Image.



You may be surprised that you see … nothing. But that's correct. Your content and your hole are identical at the moment and thus the hole „covers" all the content and nothing is visible.

Simply resize the hole to something smaller and you will see the content appear, like this:

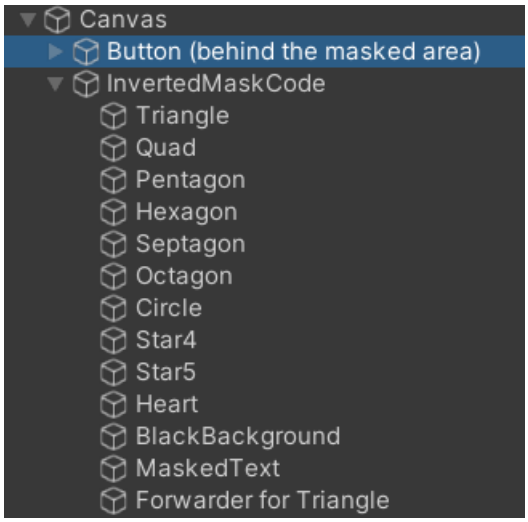


That's it, well done. Enjoy your inverted mask.

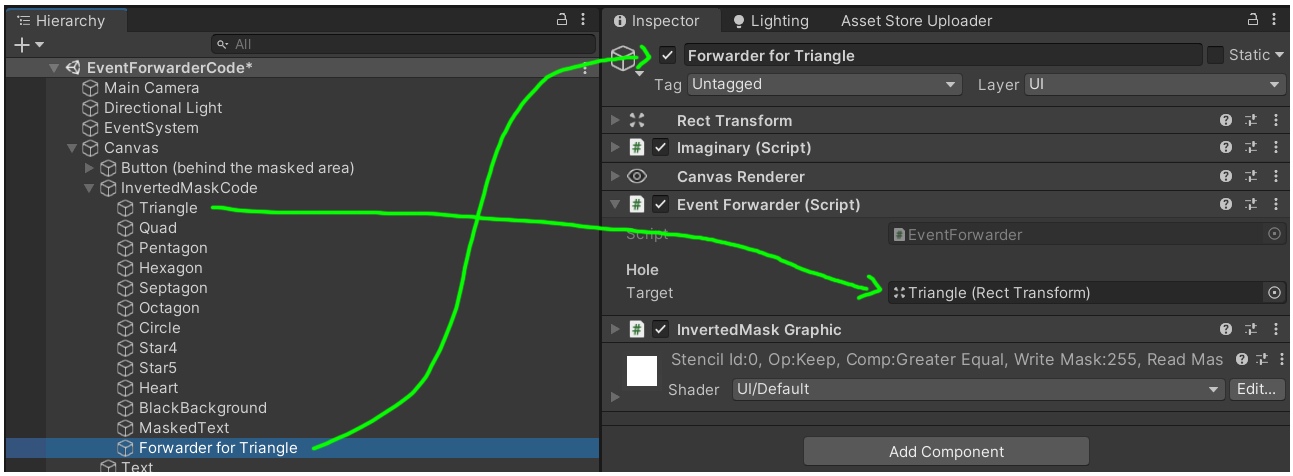
Clicking through holes

While the inverted mask gives you a visual hole in your UI, logically your image in front will still capture any UI events. To work around that there is a helper component called EventForwarder. Please check out the examples under InvertedMask/Exmaples/EventForwarderExamples.

Let's say you have a setup like this (notice the button BEHIND the masked image):



For reach hole you have you can add a forwarder like this:



NOTICE: The forwarder uses the rect shape of the hole to make an area clickable. The „Target" mustn't be a hold object. It can be any transform its area you want to enable click-through. The forder has to be IN FRONT (meaning at the lowest position in the hierarchy) within the mask to work.

The reason is that the forwarder will intersect any normal raycast and instead does a custom raycast on the UI that calls any PointClick Event or EventTrigger it finds behind the mask.