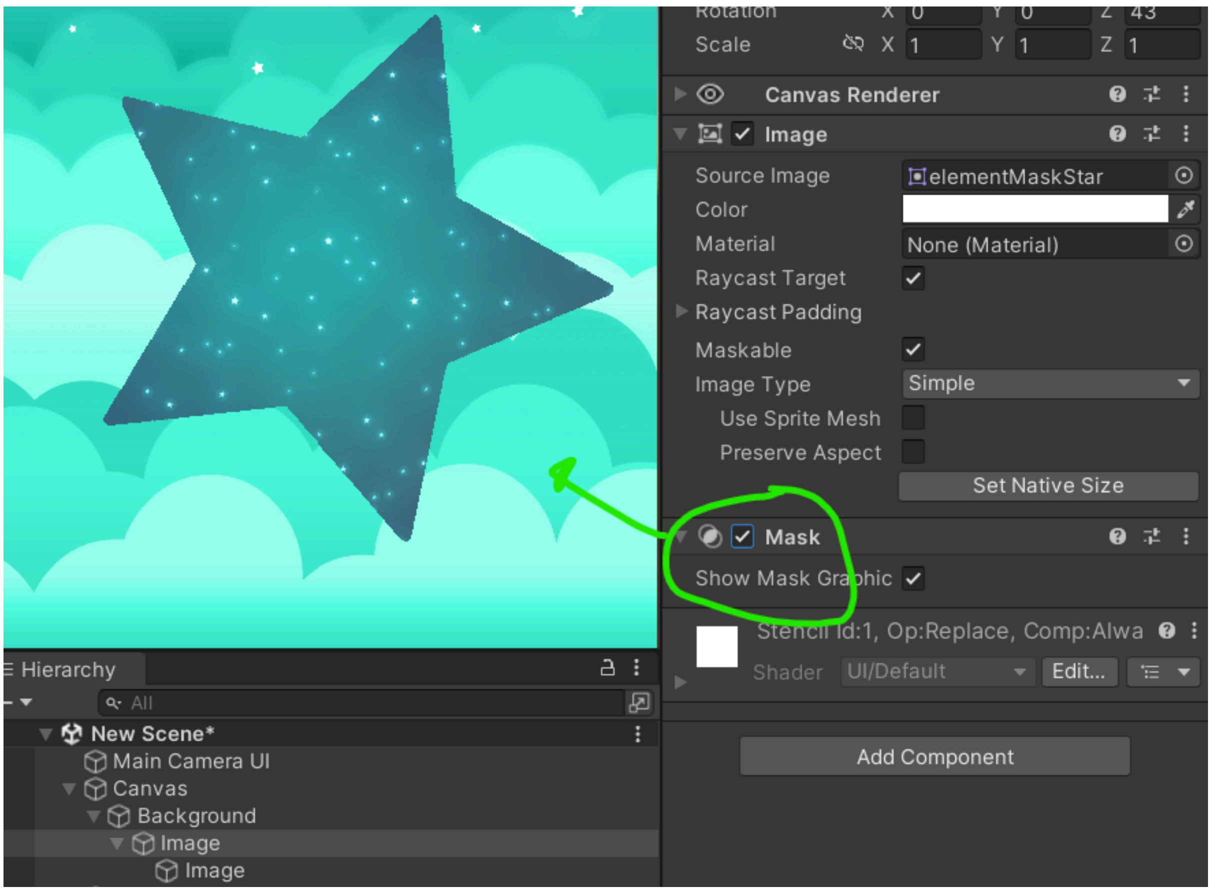
# Awesome Mask

Awesome Mask is a tool from Crazy Minds Game Studio developed to make the developer's lifes easier and practical when working with inverse masking. (among other features)

## Inverse Masking?

Masking is a technique already present in the Unity3D UI framework that allows the developer to create visuals like the one below:



Rotation X 0 Y 0 Z 43
Scale X 1 Y 1 Z 1

Canvas Renderer

Image
Source Image: elementMaskStar
Color
Material: None (Material)
Raycast Target: ✓
Raycast Padding
Maskable: ✓
Image Type: Simple
Use Sprite Mesh
Preserve Aspect
Set Native Size

Mask
Show Mask Graphic: ✓

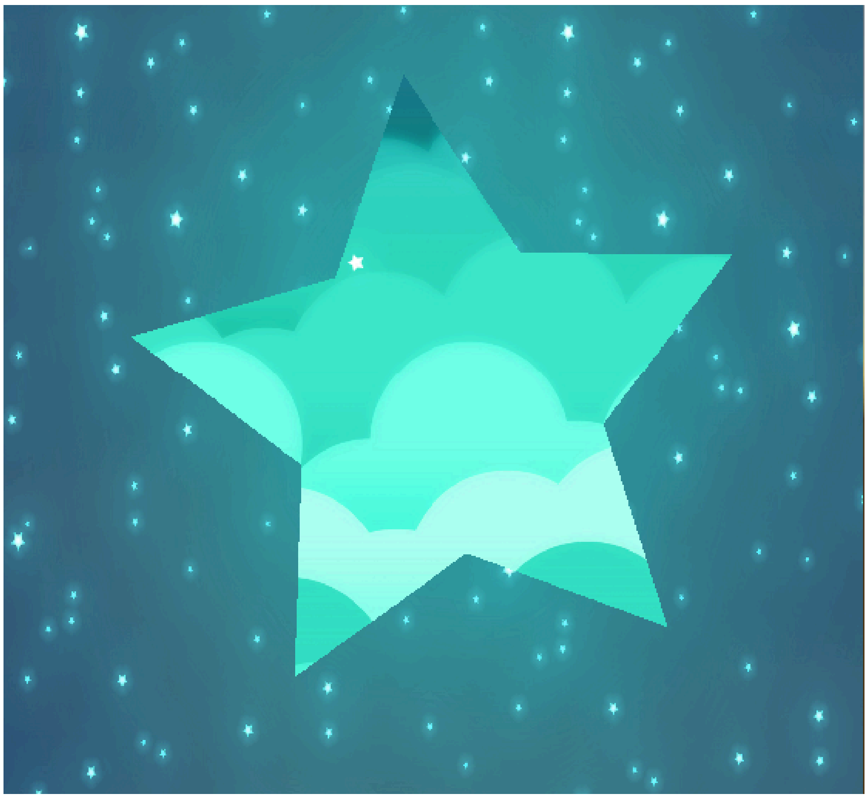
Stencil Id:1, Op:Replace, Comp:Alwa
Shader: UI/Default

Add Component

Hierarchy
New Scene*
Main Camera UI
Canvas
Background
Image
Image

With the use of the component Mask from the Unity3D framework it's possible to shape the format of an Image component according to a sprite provided.

What if you want to make it inverted?



The hollow in the form of a star in the image above is what we call Inverse Masking, you have a sprite with a hollow inside. The Awesome Mask tool allows not only inverse masking with multiple shapes but using infinity hollows.

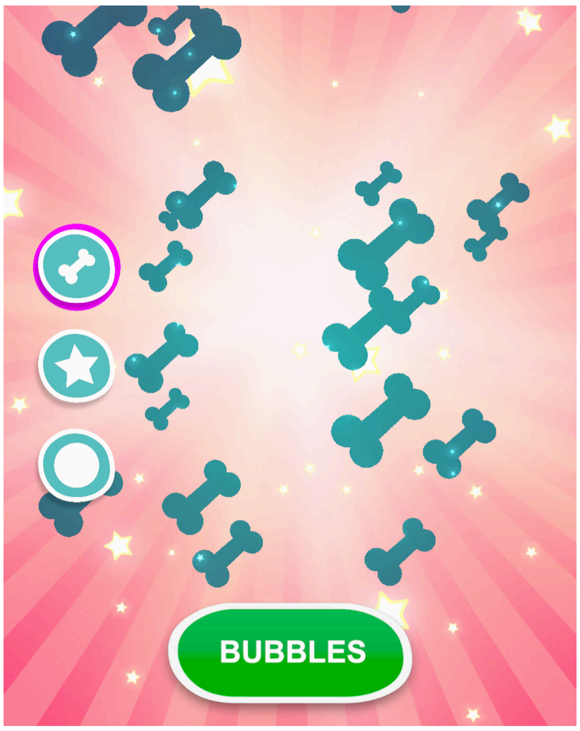
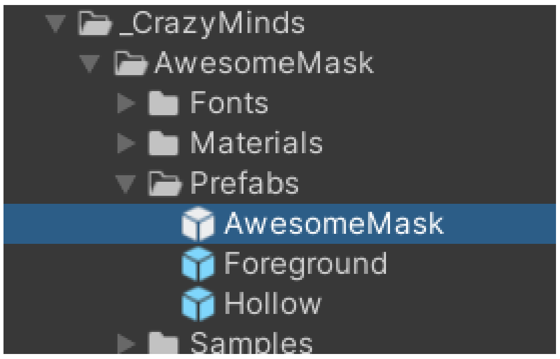


*Image taken from the Sample 2 - Bubbles from the Awesome Mask package.*

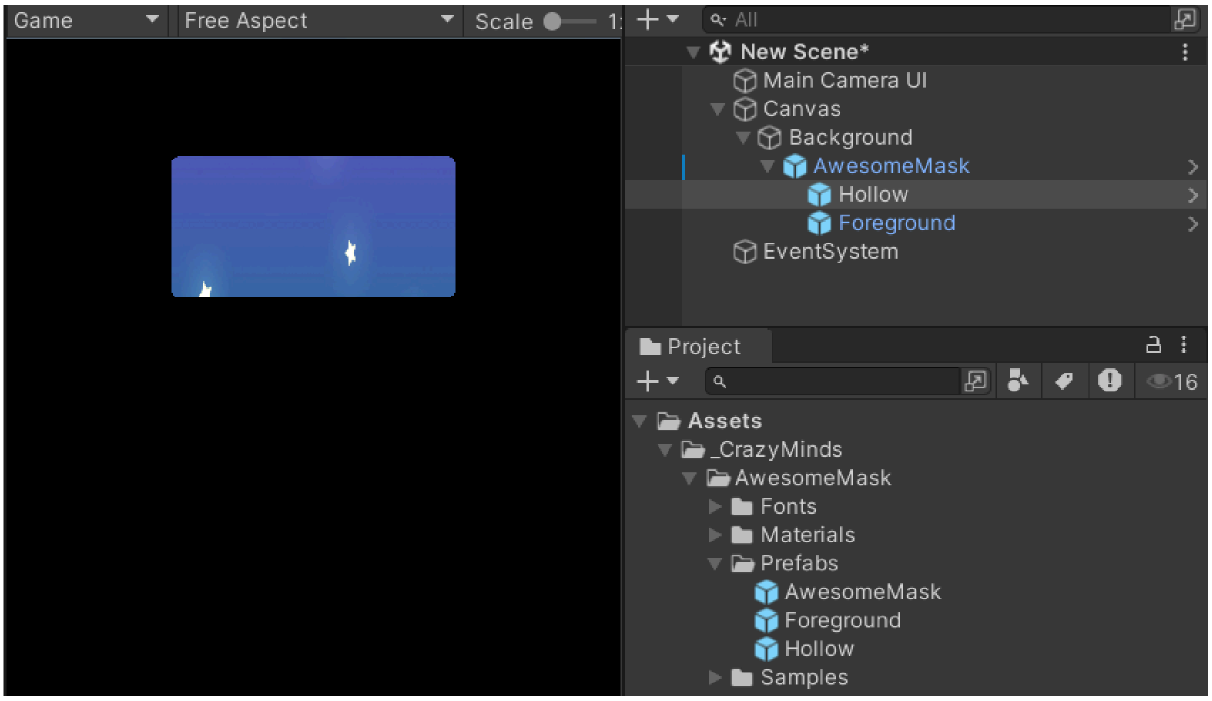
## Let’s get started!

To use Awesome Mask is very simple, you just need to drag and drop the prefab to the scene as a child gameobject of a Canvas. You will find the prefab inside the AwesomeMask folder as seen in the image below:



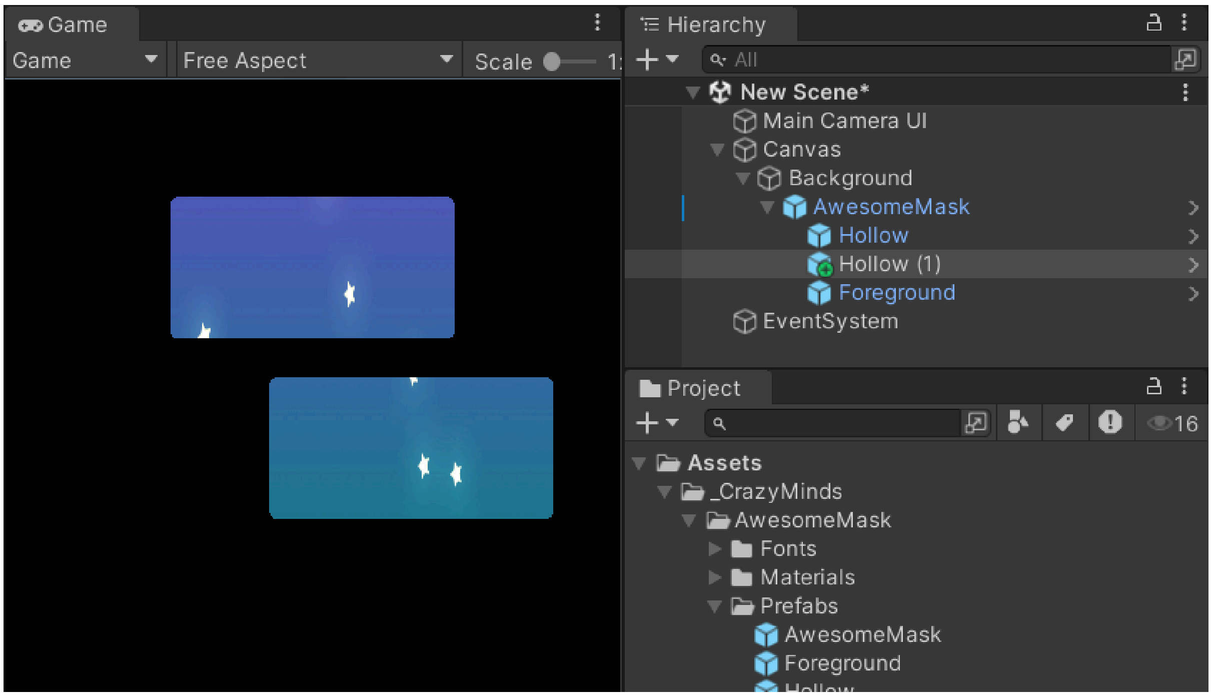
*The prefab you will use to start playing with AwesomeMask.*

This is the result you will get. The default color configuration of AwesomeMask is a black foreground with 1 hollow.



## Multiple Hollows?

To have multiple hollows you just need to duplicate the Hollow gameobject as seen in the image below:



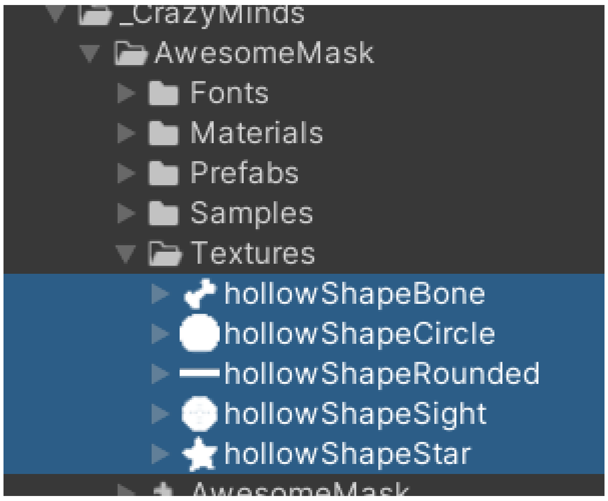
Requirement:

- keep the new Hollow gameObjects inside the same hierarchy, that is, as a child of the AwesomeMask gameobject and above the Foreground gameobject.

You can have as many Hollows you need, the only issue is the performance. So, you just need to take care of not having a huge amount of Hollows because it can degrade the performance of your target platform.

# Custom Hollow Shapes?

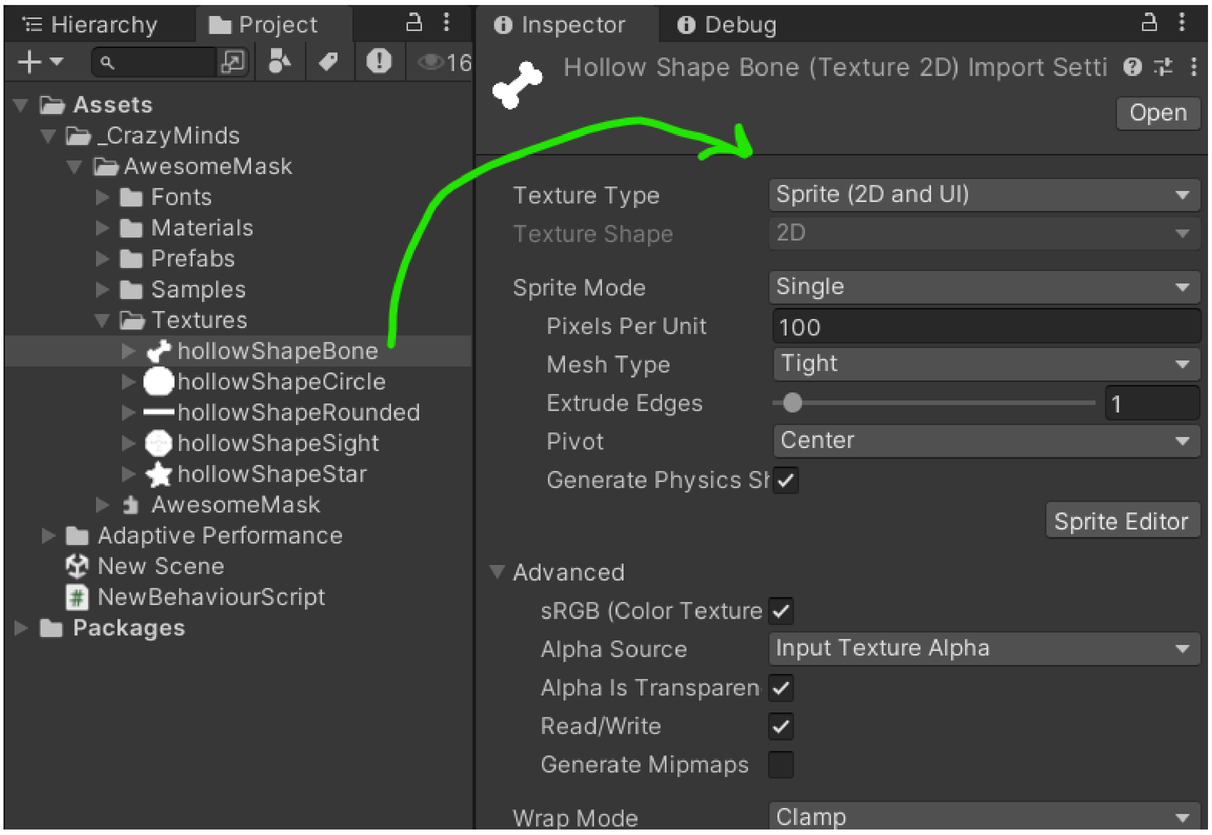
It’s possible to have custom shapes for the Hollows. The AwesomeMask package already comes with some shapes you can use:



_CrazyMinds
AwesomeMask
Fonts
Materials
Prefabs
Samples
Textures
hollowShapeBone
hollowShapeCircle
hollowShapeRounded
hollowShapeSight
hollowShapeStar
AwesomeMask

If you need to add a new one made by yourself you just need to follow the **requirements**:

- The new custom shape must be a Sprite:



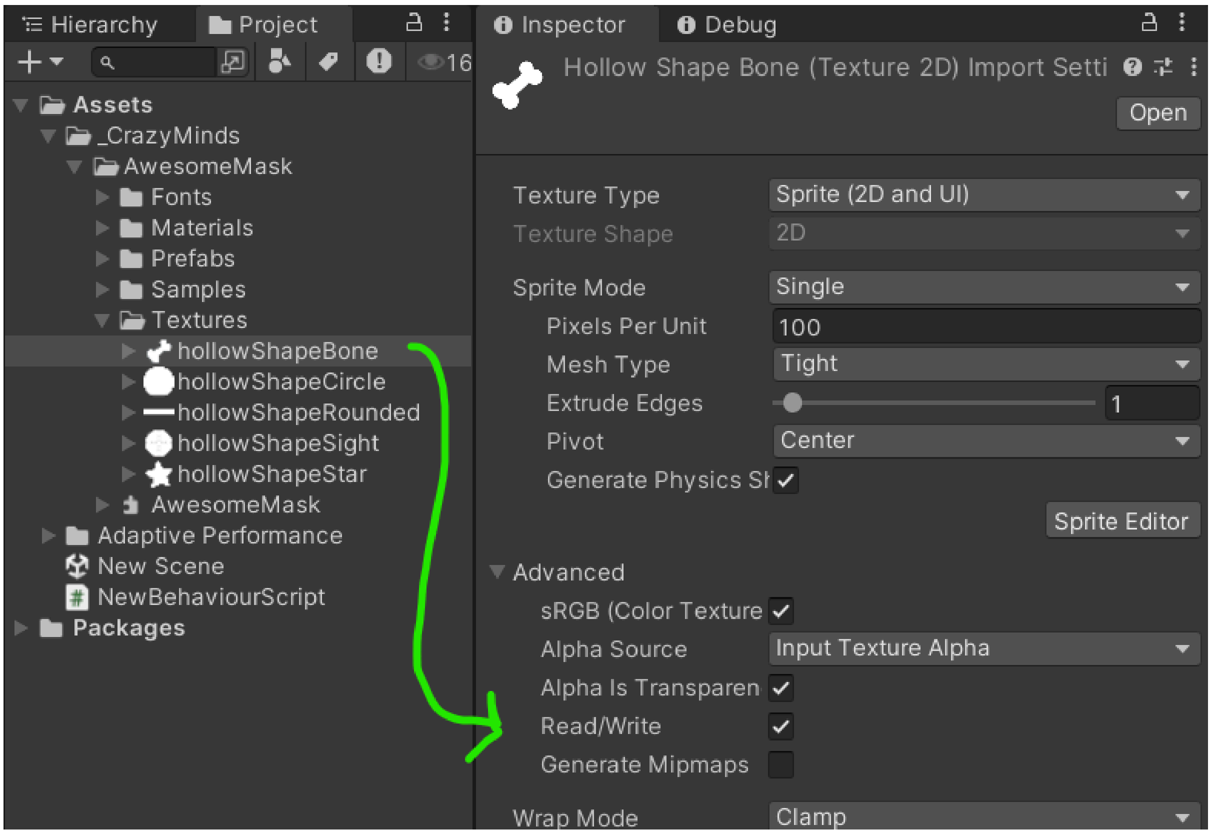
Assets
_CrazyMinds
AwesomeMask
Fonts
Materials
Prefabs
Samples
Textures
hollowShapeBone
hollowShapeCircle
hollowShapeRounded
hollowShapeSight
hollowShapeStar
AwesomeMask
Adaptive Performance
New Scene
NewBehaviourScript
Packages

Hollow Shape Bone (Texture 2D) Import Setti

Texture Type: Sprite (2D and UI)
Texture Shape: 2D
Sprite Mode: Single
Pixels Per Unit: 100
Mesh Type: Tight
Extrude Edges: 1
Pivot: Center
Generate Physics Sh: ✓

Advanced
sRGB (Color Texture: ✓
Alpha Source: Input Texture Alpha
Alpha Is Transparen: ✓
Read/Write: ✓
Generate Mipmaps:
Wrap Mode: Clamp

- The new custom shape Sprite must be configure as **READ/WRITE**



Hierarchy | Project | Inspector | Debug

- Assets
  - _CrazyMinds
    - AwesomeMask
      - Fonts
      - Materials
      - Prefabs
      - Samples
      - Textures
        - hollowShapeBone
        - hollowShapeCircle
        - hollowShapeRounded
        - hollowShapeSight
        - hollowShapeStar
      - AwesomeMask
  - Adaptive Performance
  - New Scene
  - NewBehaviourScript
- Packages

Hollow Shape Bone (Texture 2D) Import Setti

Open

| | |
|---|---|
| Texture Type | Sprite (2D and UI) |
| Texture Shape | 2D |
| Sprite Mode | Single |
| Pixels Per Unit | 100 |
| Mesh Type | Tight |
| Extrude Edges | 1 |
| Pivot | Center |
| Generate Physics Sł | ✓ |

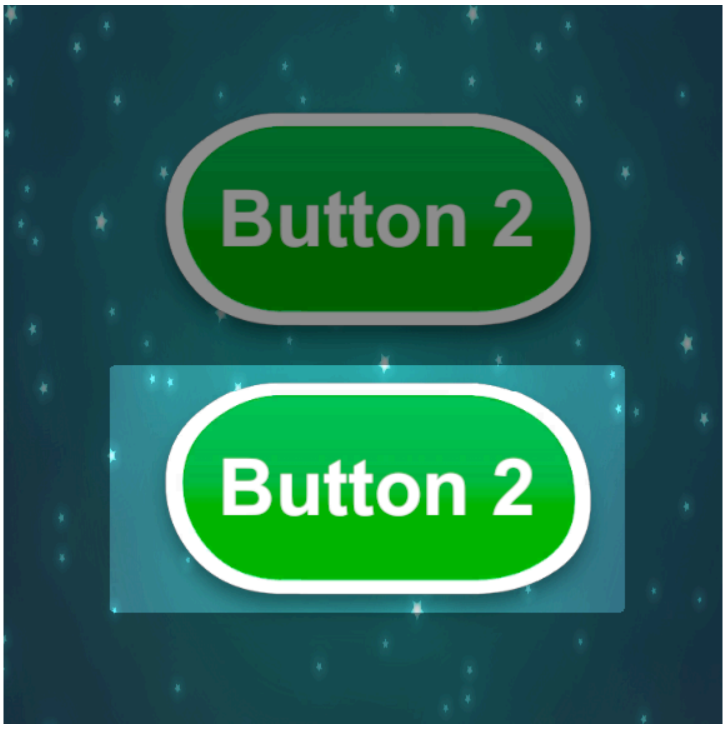
Sprite Editor

Advanced

| | |
|---|---|
| sRGB (Color Texture | ✓ |
| Alpha Source | Input Texture Alpha |
| Alpha Is Transparen | ✓ |
| Read/Write | ✓ |
| Generate Mipmaps | |
| Wrap Mode | Clamp |

## UI interaction

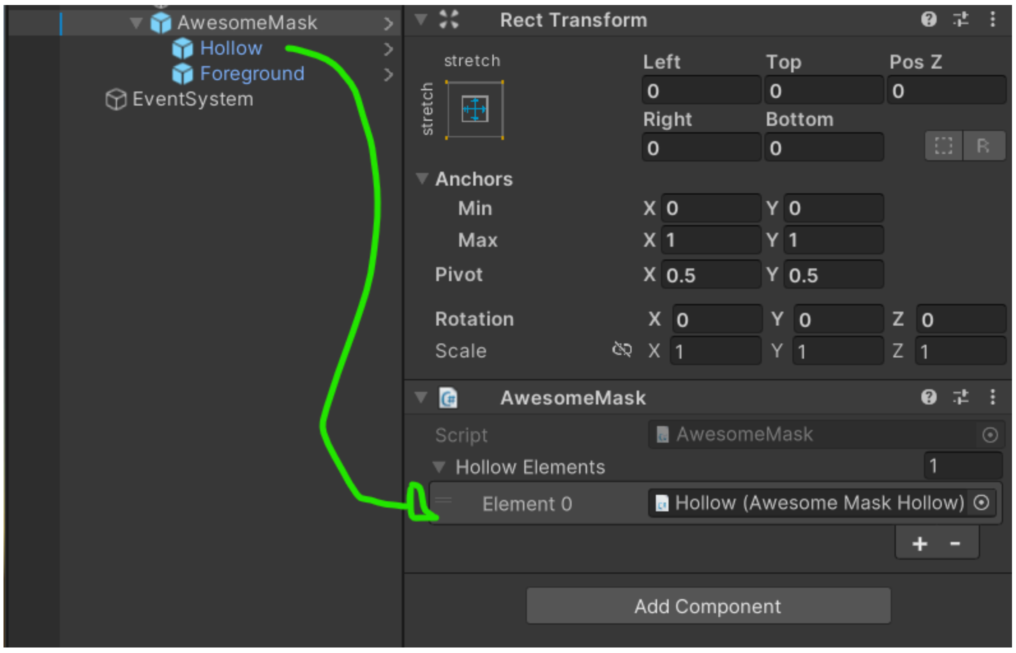
One of the most common applications of the Awesome Mask tool is tutorials. With Awesome Mask you can not only highlight areas of the game but to control the interactivity with components like buttons.



*A button being highlighted by Awesome Mask.*

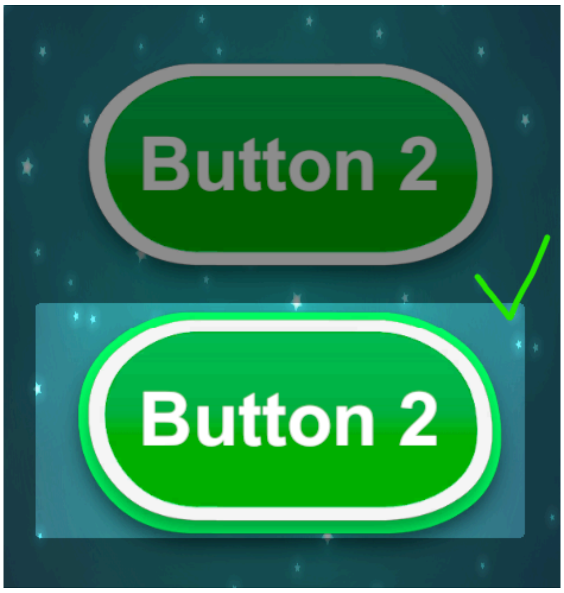
## Allowing interactivity with UI

In order to allow interactivity to the UI components below the Awesome Mask component you just need to add the Hollow on top of the component you highlighted to the Hollow Elements list in the Inspector:



```
AwesomeMask
  Hollow
  Foreground
EventSystem

Rect Transform
stretch
stretch      Left        Top         Pos Z
             0           0           0
             Right       Bottom
             0           0
Anchors
  Min        X 0         Y 0
  Max        X 1         Y 1
Pivot        X 0.5       Y 0.5

Rotation     X 0         Y 0         Z 0
Scale        X 1         Y 1         Z 1

AwesomeMask
Script              AwesomeMask
Hollow Elements                              1
  Element 0         Hollow (Awesome Mask Hollow)
                                           + -
Add Component
```

*Hollow added to the interaction List.*

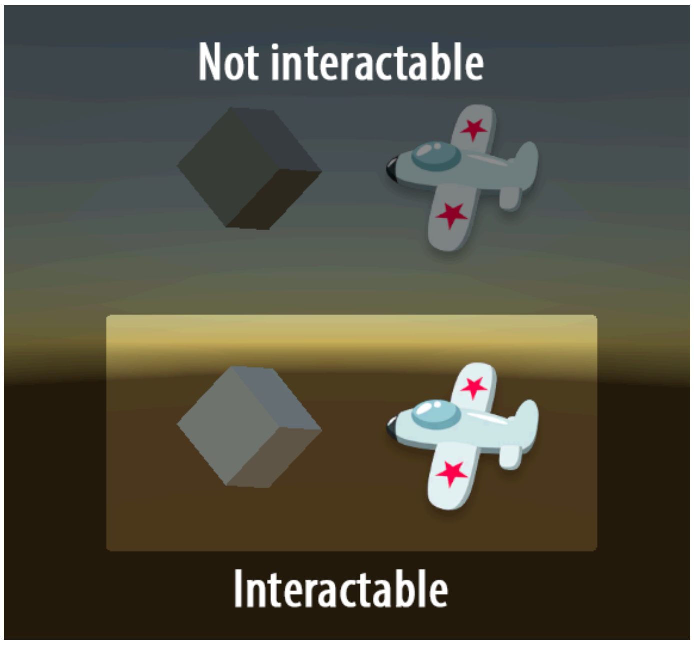


*A button being clicked through a Hollow.*

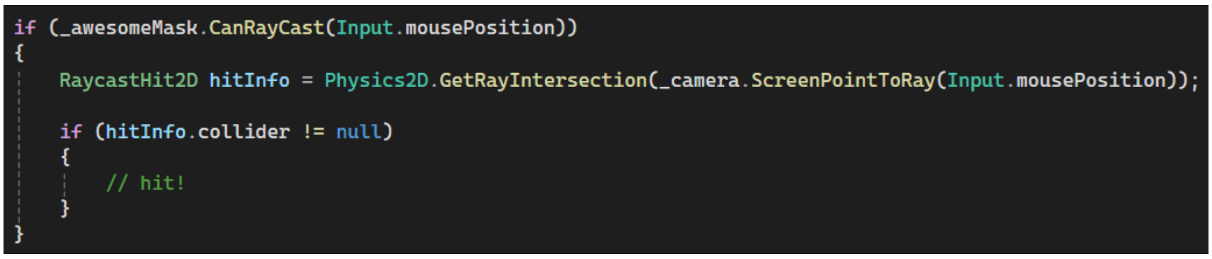
Hollow added to the hierarchy that is not inside the Hollow Elements list will still highlight the components behind it but will not allow interactivity.

## 2D/3D Interactivity

Awesome Mask doesn't block raycasts done by the developer by script using Physics2D/Physics3D APIs, the developer must first check if the elements in the 2D/3D layer is behind one of the intractable ares (Hollows).



To do so, it's required, to before calling your Raycast API of preference, to check using CanRayCast() method from AwesomeMask, as can be seen below:



```
if (_awesomeMask.CanRayCast(Input.mousePosition))
{
    RaycastHit2D hitInfo = Physics2D.GetRayIntersection(_camera.ScreenPointToRay(Input.mousePosition));

    if (hitInfo.collider != null)
    {
        // hit!
    }
}
```

*Calling the Method CanRaycast() from AwesomeMask.*

The Awesome Mask API then verifies if the position in the screen (MousePosition or Touch) is inside the Hollow rect.

The method CanRaycast() returns a bool value. If true, it means the point in the screen being tested is inside a Hollow.

Requirement:

- This Hollow must be inside the Hollow Elements list to get computed.
- The position passed to CanRaycast() must be in the screen space.

**Hollows not added to the Hollow Element list will still highlight elements behind it, but will always return false when checked positions (with CanRaycast) are inside its area.**

## Repositioning Hollows

To reposit Hollows in the screen at runtime can be done by any script as it is a simple game object, but, you can use one of the APIs from AwesomeMask if you like.

- ***SetElementPosition(int index, Vector3 position3D)***

This API will reposit the Hollow with Index equal to the index passed as parameter. The position3D is a world space position..

```
// reposit the element of index 1 in the Hollow Elements List:
_awesomeMask.SetElementPosition(1, someGameObject.transform.position);
```

*Repositioning at runtime a Hollow in the list of Hollow Elements by its index.*

- ***SetElementPosition(AwesomeMaskHollow hollow, Vector3 position3D)***

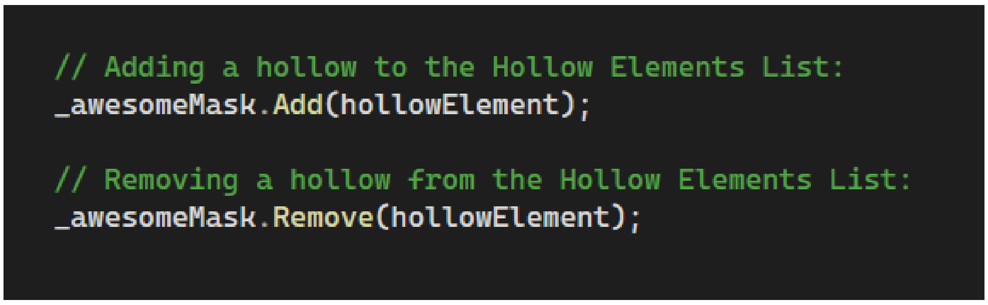
This API will reposit the Hollow object passed as parameter. The position3D is a world space position. This object is not required to be inside the Hollow Elements list.

```
// reposit the hollow object:
_awesomeMask.SetElementPosition(hollowElement, someGameObject.transform.position);
```

*Repositioning at runtime a Hollow object.*

## ADD/REMOVE APIs

The Add() and Remove() APIs are responsible for adding a Hollow to the Hollow Elements list and then being used in the interactivity calculation.



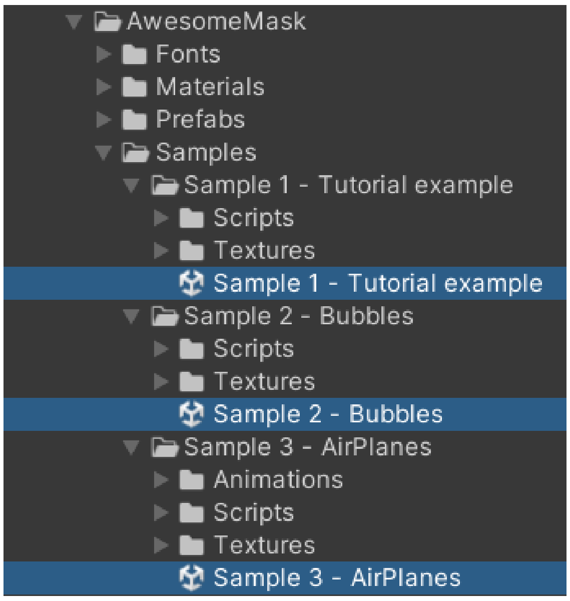
```
// Adding a hollow to the Hollow Elements List:
_awesomeMask.Add(hollowElement);

// Removing a hollow from the Hollow Elements List:
_awesomeMask.Remove(hollowElement);
```

It’s always important to control which Hollow is inside the Hollow Elements List as it is a requirement for the Interactivity calculation.

# Samples

Inside the AwesomeMask package you will find several examples to follow, learn and get new ideas for using the Awesome Mask tool.

You will find this samples inside the folder Samples:



```
AwesomeMask
  Fonts
  Materials
  Prefabs
  Samples
    Sample 1 - Tutorial example
      Scripts
      Textures
      Sample 1 - Tutorial example
    Sample 2 - Bubbles
      Scripts
      Textures
      Sample 2 - Bubbles
    Sample 3 - AirPlanes
      Animations
      Scripts
      Textures
      Sample 3 - AirPlanes
```

To start playing with it, load the specific scene from the sample you want to learn and hit play.

At the time this document is written there are 3 samples, Tutorial, Bubbles and Airplanes:

**Tutorial Sample**: in this sample you will learn how AwesomeMask can make your life easy in the development of tutorials.

**Bubbles Sample**: this is a simple example to show the power of Awesome Mask and how any developer, with a simple but creative idea, can create amazing games with real visual appeal.

**AirPlanes Sample**: is more than a simple sample but a tutorial on how to use the features from AwesomeMask, mainly the interactivity features, how to interact with 2D/3D elements, and manage the AwesomeMask components at runtime.

# Thank you!

If you are here it means you bought a product developed by me (Julio Dutra). It means a lot to me you are enjoying this piece of work of mine. I do my work with love and I really hope it brings you an additional step to your success.

Crazy Minds Game Studio
Website
Playstore
Instagram
Twitter
Facebook
Linkedin