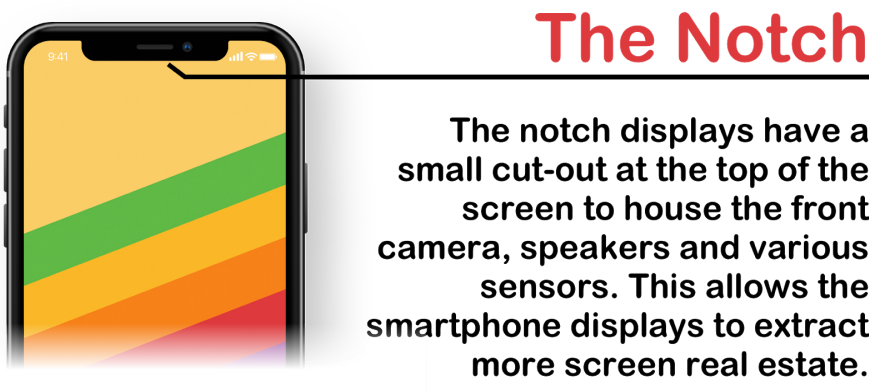
Safe Area Utility

An easy, dynamic, and visual solution
for all your Notches, Cut-outs, and Safe Areas needs.

Overview

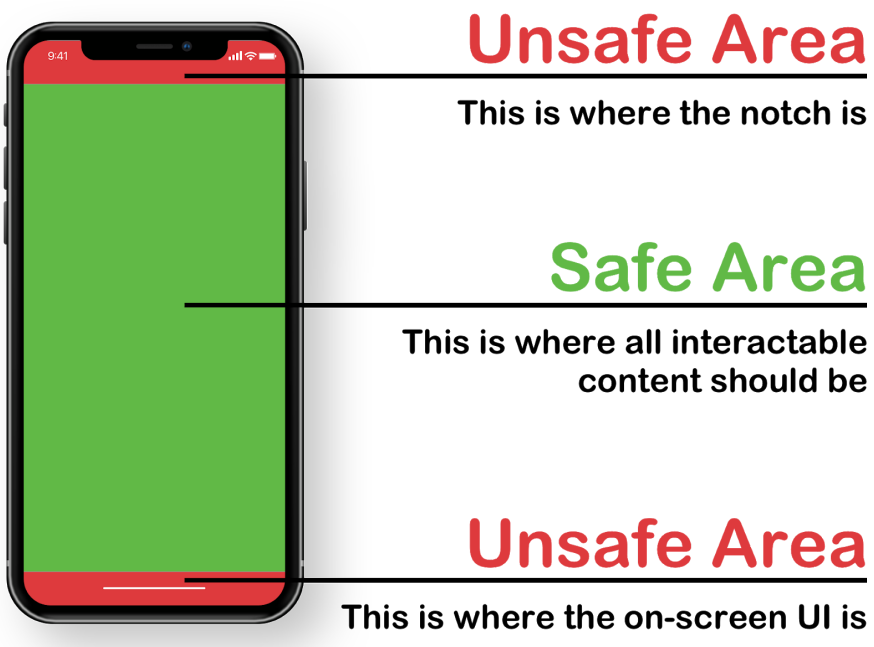
What is SafeArea?

We see more and more devices adopt hardware screens that are not rectangular, or that have software requirements that affect the App/Game screen real estate. A prime example for such device is the iPhoneX. It was the first phone that introduced the notch.



This raises some new challenges to developers, in addition to supporting different screen sizes and resolutions, we now need to support different behaviours for different parts of the screen.

A **Safe Area** is a rectangular region of the screen that is free from cut-outs, or notches.



By defining a **Safe Area**, we also define the **Unsafe Areas**. Where the rule of thumb is, do not put any interactable content, or important visual content outside of the **Safe Area**. What should you so with the **Unsafe Area** then? It really depends on the specific design of your App/Game.

In the **Unsafe Area** you might want to show the game world (to allow an immersive experience), HUD background, or simply set it to black (though not recommended). What ever are your needs, a flexible toolset is needed.

Unity SafeArea

Unity, as a cross platform development platform, have provided such a tool: Screen.safeArea. It is a Rect struct that defines where is the safe area of the screen (the green area in the image above). Unity have provided a way to dynamically retrieve this information at runtime based on the specific device you are running on.

While Screen.safeArea offers the data, it is not always easy to develop and test, due to the lack of editor support for a visual representation and mocking of these specific devices.

SafeAreaUtility Introduction

The safe are utility is built upon Unity's Screen.safeArea, to allow an easy and visual workflow. The tool is fully integrated into the editor, providing visual aid, and allowing testing in Edit Mode.

SafeAreaUtility Key Features

- **Run In Edit Mode:** test everything in the editor, while in edit mode.
- **Visual Safe Area:** draws the unsafe area regions in the editor game view.
- **Flexible:** set any adjustments you need, straight from the inspector - no code needed.
- **Dynamic:** mock any device by simulating its Safe Area in the editor. Easily define new devices - no code needed.
- **Device Overrides:** you can define your own Safe Area definition for specific devices. If you want a bit more paddings or a specific device is not working nice with Screen.safeArea - no problem.
- **Performance:** the tool was built for mobile devices with performance in mind. To achieve that, calculated values are cached and centralized. Any editor related code is being stripped off when building for production. In addition, the package is using assembly definition files to reduce compilation time overhead in development.

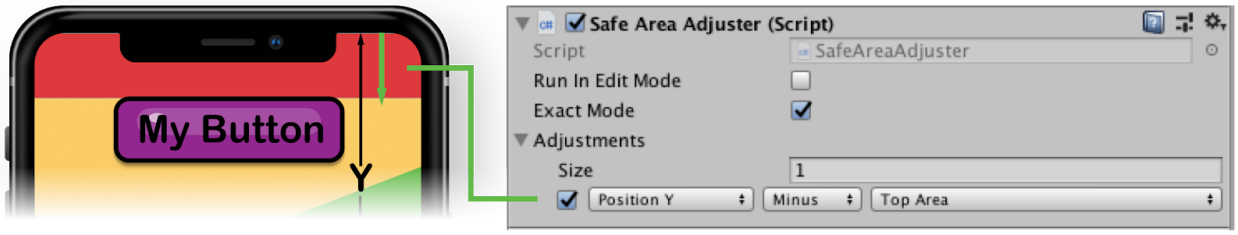
Setup

After adding the package to your project you will see a new folder **EpicBrainGames/SafeAreaUtility**.

Basic Scenario

If your game is not expected to change resolution or orientation during gameplay, this approach is for you, and will save a bit of a performance. **Note** that in order to visually debug in the editor, you will need to temporarily use the method described in the advanced scenario.

All you need to do is to add the **SafeAreaAdjuster** script to the object you want to adjust based on the safe area. After adding the script, you can define in the inspector any adjustment that is needed. For example, moving the button below the unsafe area:

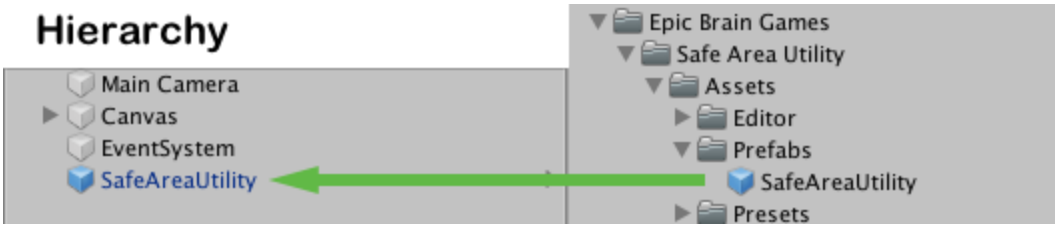


Here we defined one adjustment, where we move the object down along the Y axis, with distance equal to the **Unsafe Top Area**. If the device has no cut-outs, the adjustment will not apply.

Read more about **SafeAreaAdjuster** below.

Advanced Scenario

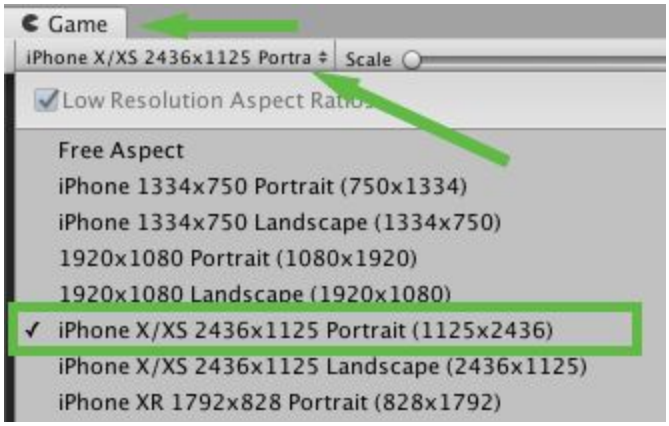
If your game can change resolution or orientation, you must include the **SafeAreaUtility** prefab in your scene hierarchy (or attach the **SafeAreaUtility** script to any game object).



Now the **SafeAreaUtility** will track any screen resolution or orientation changes.

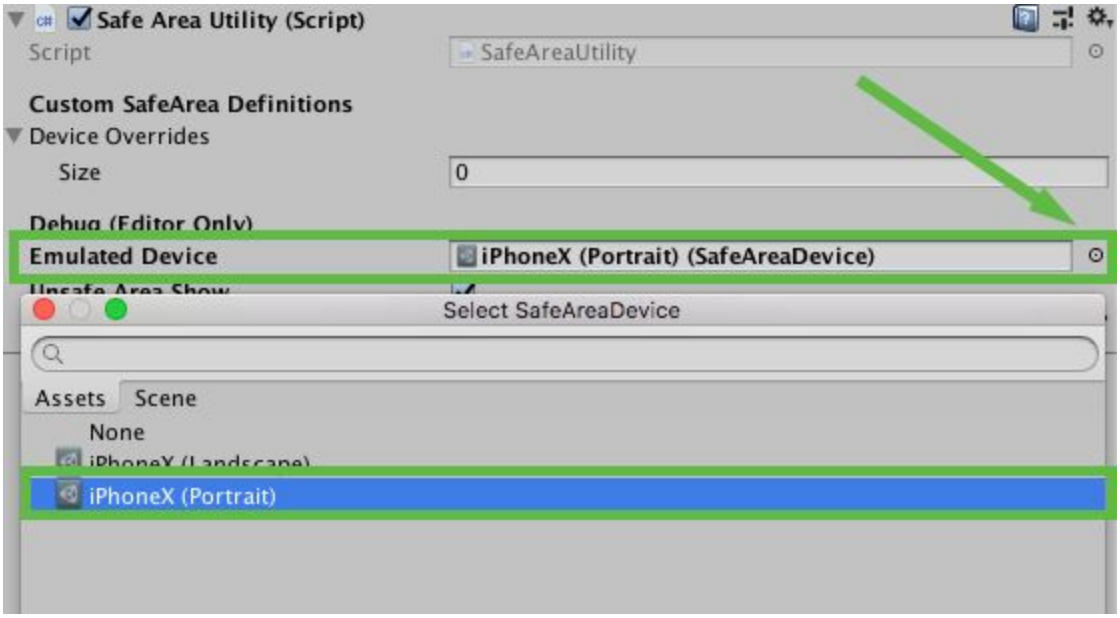
To set up the editor to test the iPhoneX device (the same approach can be applied for any other device). Follow these steps:

1. Change the Game window view **resolution** to match iPhoneX (Portrait).



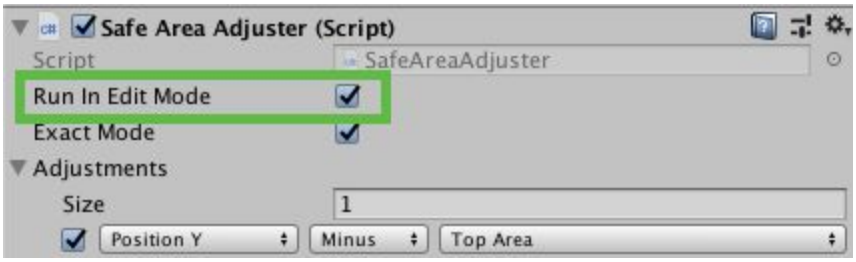
   a. If you don't see iPhone X on the list, simply create a new device with **width = 1125px** and **height = 2436px**.
   b. Note that you don't have to be on the iOS target platform, it will work regardless.

2. Go to the **SafeAreaUtility** script in the inspector, and select iPhoneX (Portrait) for the **Emulated Device**.



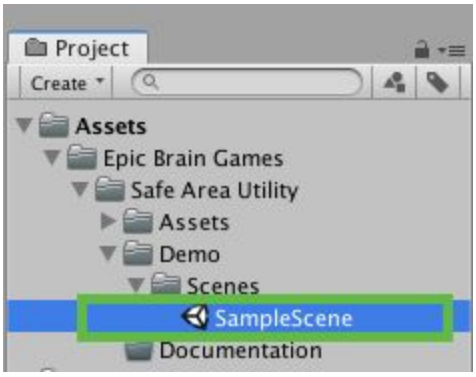
3. Verify that the **Unsafe Area Show** option is enabled. If you followed the above steps, and everything is working as expected, you should now see the unsafe area visible in the game view:

4. Now, if you enter play mode, the iPhone X **Safe Area** will be used. If you want to test in edit mode, you must enable the **Run In Edit Mode** option it the **SafeAreaAdjuster.**



Demo Scene

You can find a demo scene in this path:
Epic Brain Games/Safe Area Utility/Demo/**SampleScene**



Simply double click to open the scene. There you can find a few examples for different cases and how to setup everything as necessary.

SafeAreaUtility Component

The **SafeAreaUtility** is both a static class and a monobehaviour singleton. If you don't want to debug and your game will not change resolution or orientation - there is no need to add this component to your scene.

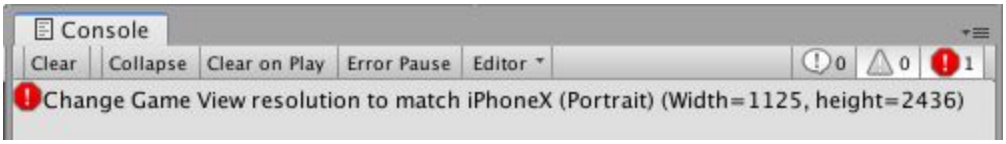
SafeAreaUtility Prefab

If you want to debug the Safe Area or your game supports orientation or resolution changes, you must have the **SafeAreaUtility** component active in your game.

You can either add the provided **SafeAreaUtility Prefab** to your scene or simply add **SafeAreaUtility** component to any game object. Notice that since we are using the singleton pattern, if the gameobject is enabled while there is already an active gameobject with this script attached - it will be destroyed.

Emulated Device

The emulated device is only used in the Editor for testing purposes. The field expects a **ScriptableObject** object of type **SafeAreaDevice** (See the **SafeAreaDevice** section below for more info). You must set the Game view resolution to match the emulated device, if this is not the case, you will see the following error in the console:



You will need to change the game view resolution to the exact values, and then select the emulated device again. Notice that **iPhone X (Portrait)** and **iPhone X (Landscape)** are considered 2 different devices.

Once setup with no errors in the console, the device will be used as a reference for the SafeArea definitions.

Unsafe Area Show

Enabling this option will draw the unsafe areas, of the emulated device, to the game view. You can change the color if you like. This entire feature is stripped off when building for a device, so no fear to see the unsafe areas in production.

(Additional overlays are the **Cut-outs** and **Tight Unsafe Area**)

Device Overrides

This feature allows you to override Unity's Screen.safeArea definition with any definition you desire per device. This could be specifically useful for 2 reasons:

- If a new device is out, but it is still not supported properly with Screen.safeArea, you can fix this issue immediately by providing your own definition.
- You want to provide screen margins for specific devices, which are more than the basic definition of the device default SafeArea. You can apply this to ant device, even if it does not have a SafeArea definition.

To use, simply add the **SafeAreaDevice** you want to override to the **Device Overrides** list (See the **SafeAreaDevice** section below for more info).

SafeAreaAdjuster Component

The **SafeAreaAdjuster** is the script you to an object you want to adjust based on the SafeArea. If the screen resolution or orientation will change (and you have included the **SafeAreaUtility** in your scene), the **SafeAreaAdjuster** will re-adjust the values to match the current SafeArea configuration.

Adjustments

To define one or more adjustments, simply add them to the Adjustments list. Each adjustment has the following structure (from left to right):

1. **Enable or disable** the specific adjustment. This is useful when testing in the editor and you want to see how the specific adjustment behave.
2. **What to adjust** is the target property that you want to change. For example, if you want to move the object on the Y axis, you set it to “Position Y”.
3. **How to adjust** the target property. Meaning, should you add or subtract - For example, move up or down the Y axis.
4. **Based on what** to adjust the target property. There are 4 areas to consider: Top, Bottom, Left, Right. Some areas might not be relevant for specific devices. For example, the iPhone X has a Top Area for the Notch and Bottom Area for the on screen button.

So, you can create any adjustment needed with the above combination. In the example, you can read the adjustment like this: Move the object down with a value based on the Top Area size.

Tight Unsafe Area

While the **Safe Area** defined by the device manufacture may include paddings around the notches/cut-outs and for curved screen corners etc, the **Tight Unsafe Area** only surrounds the notches/cut-outs of the device - with no added padding. This value is calculated based on the Screen.cutouts and is supported from Unity 2019.2 and higher.

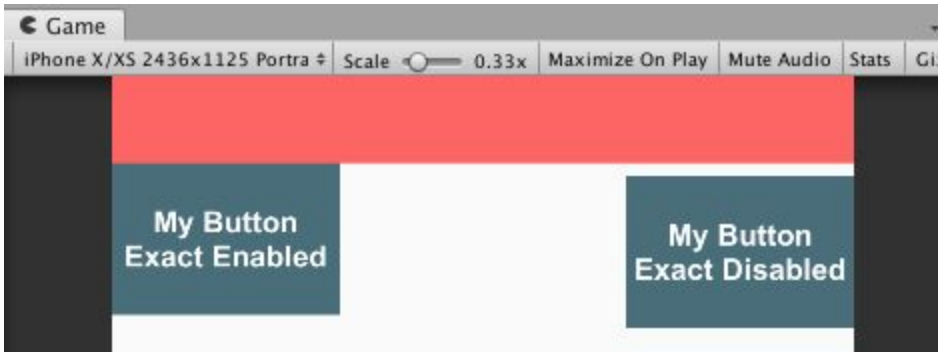
You may define your overrides for the cutouts in the override devices on earlier unity versions as well.

A usage for this feature may be to align UI elements to the cut-outs and avoid the extra spacing, as you might get in iPhoneX.

Exact Mode

This mode is enabled by default. When enabled, the values are factored by the parent canvas size - resulting in an exact positioning on the screen. This is specifically important if you have a CanvasScaler set up with the **Scale With Screen Size** :

Not enabling the Exact mode will result in a small percentage of error based on your specific configuration and the specific device resolution. Here is an example with the iPhone X:



If you don't use the **Scale With Screen Size** setup, or you don't care about an exact positioning, you can disable the **Exact Mode** to reduce the performance overhead (a small one)

Run In Edit Mode

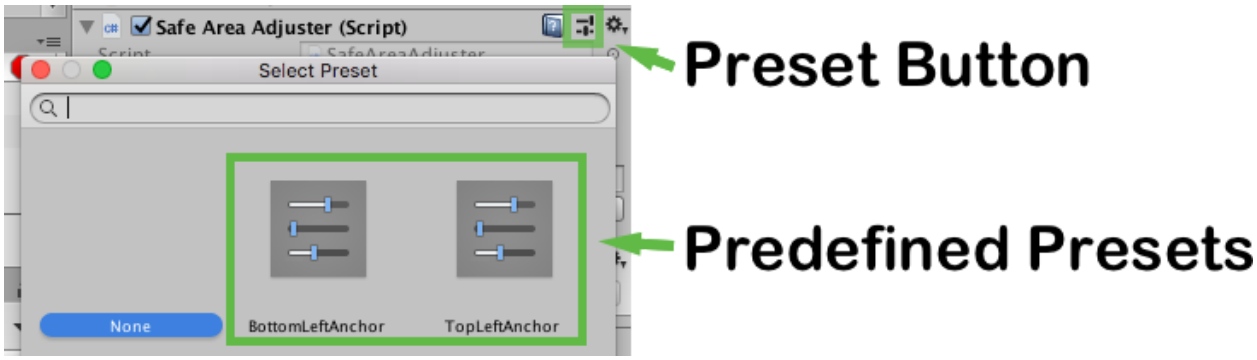
Enabling this mode will allow you for easy testing in edit mode. Note that while his option is enabled - you can't (and should not) change the object transform directly. Running in edit mode will require setting up an **Emulated Device** as described above.

Suggested workflow:

1. Set the game object positioning as you normally would.
2. Add the **SafeAreaAdjuster** script to the object
3. Mount an **Emulated Device** and enable the **Run In Edit Mode**
4. While seeing the red Unsafe Areas in the **Game View**, configure the Adjustments until the new position is as you desire.
5. Disable the **Run In Edit Mode**

Presets

After a while you might find yourself reconfigure the same Adjustments over and over. To solve that, consider using the component **preset**. A few predefined presets already available to use, but you could create your own presets if needed.



SafeAreaDevice

The **SafeAreaDevice** is a ScriptableObject used to define devices to be used by the **SafeAreaUtility**. The object contains all the data required to generate the **SafeArea**.
The predefined devices are located in this folder:
Epic Brain Games/Safe Area Utility/Assets/ScriptableObjects/Devices/*

Creating a new device

To create a new SafeAreDevice, select from the menu:
Assets -> Create -> EpicBrainGames -> SafeAreaUtility -> SafeAreaDevice

This will create a new file in the project. Let's examine iPhoneX as an example:

- **Display Name (Optional)** is the name is used internally.
- **Orientation (Required)** should be either Portrait or Landscape. Note that a device has different files for each orientation.
- **Model Names (Optional)** is used to identify the device at runtime by the Device Overrides feature. The value is then compared to SystemInfo.deviceModel. If there is a match, the override device definition will be used.
- **Screen Resolution (Required)** is the resolution of the screen in the given orientation.
- **Screen Unsafe Areas (Required)** is the size of the unsafe

areas. These values are used when debugging in the editor or when overriding devices. Changing these values will be reflected immediately in the editor in case the device is being emulated.

Contact Us

Questions? Bugs? Suggestions? Requests? Or anything else, please reach out:

Eyal Biran
Epic Brain Games
epicbraingames@gmail.com