Advanced ColorPicker for Unity UI

Readme

Table of Contents

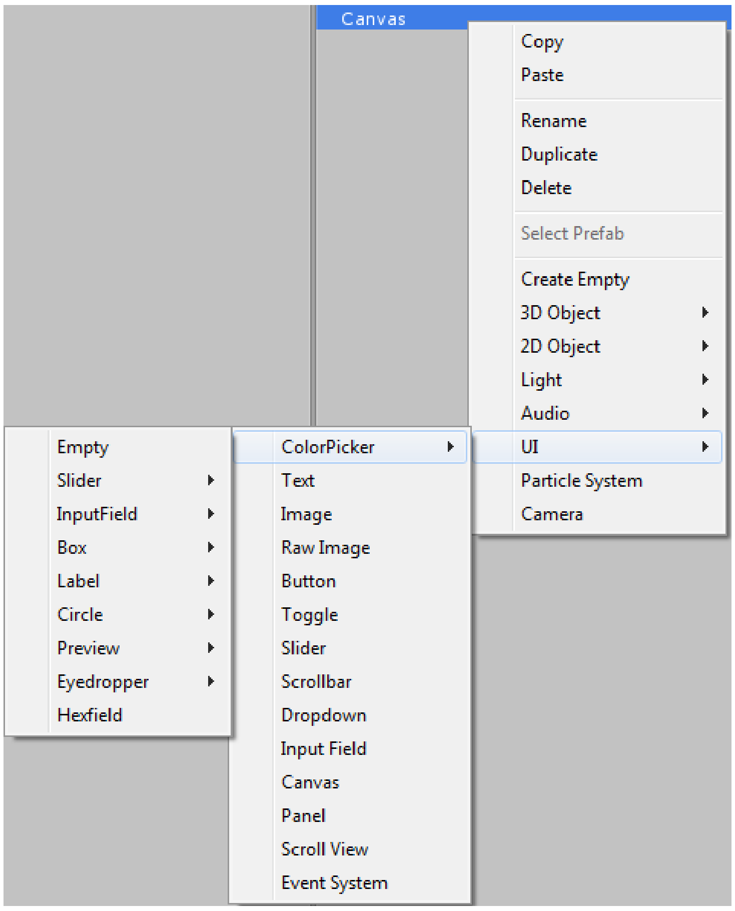
How to use this asset

This asset is meant to help you to quickly create a colorpicker that is tailored to your needs. To do this, it uses MenuItems which allow you to quickly add various components to your colorpicker. And because this colorpicker is made for Unity UI, you can easily modifly the look in the same way you modify the rest of your UI.

The MenuItems are located under “GameObject/UI/ColorPicker” and can be accessed both through the GameObject MenuItem in the top bar as through the context menu in the hierarchy.

Although it is possible to add each script separately, it is advised to use the MenuItems as it can save you a lot of time setting up.

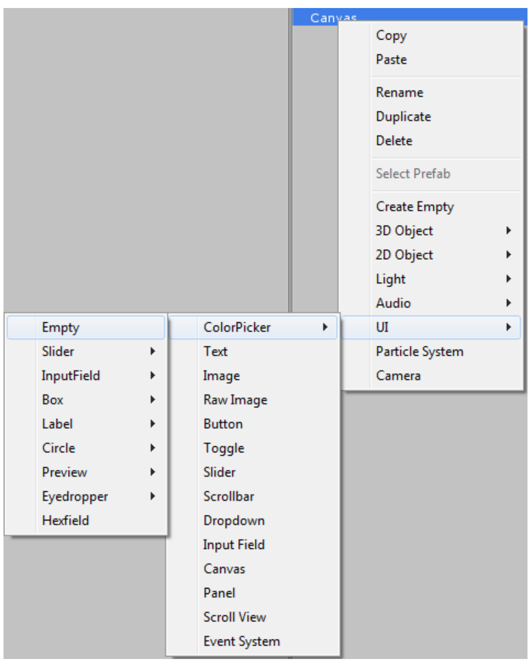
Most components require a reference to a ColorPicker in order to work. When using the MenuItems, if you have something in the hierarchy selected and this something has a ColorPicker component on itself or any of its parents. This ColorPicker is automatically set to speed up the process of creating a new colorpicker even further.

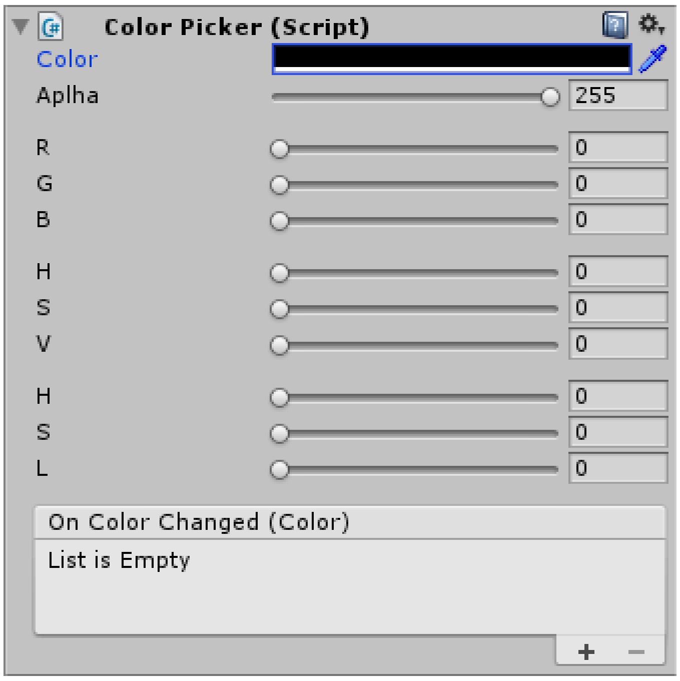


Known problems

The EyeDropper and EyedropperPreview both might return the wrong color in the editor when you set the resolution in the game view to anything other than “Free Aspect”. This is because I can't get the actual size of the gameview window or the size the then added gray borders. And since the mouseposition is based upon the game view and reading pixels is based upon the entire window, the resulting colors are wrong. This problem only exists inside the EDITOR! In the build both the mouseposition and the reading of pixels are based upon the same resolution, even when the game has black borders.

ColorPicker




Color Picker (Script)

Color

Aplha 255

R 0

G 0

B 0

H 0

S 0

V 0

H 0

S 0

L 0

On Color Changed (Color)

List is Empty

- **Color:**
  This displays the current color of the ColorPicker. You can click it to modify the color as expected. Note that although you can se the editor colorpicker of Unity to display HSV, it still returns only RGB values meaning that the actual HSV values of the colorpicker can differ (As there can be multiple HSV values for the same RGB values).
- **Alpha:**
  A slider to modify the alpha value.
- **R/G/B:**
  Sliders to modify red, green and blue values.
- **H/S/V:**
  Sliders to modify hue, saturation and value (this saturation is not the same as the HSL one).
- **H/S/L:**
  Sliders to modify hue, saturation and light (this saturation is not the same as the HSV one).
- **OnColorChanged**
  This field can be used to add listeners to the OnColorChanged event. This event is called every time any of the values of the colorpicker change. It accepts any non-static method with a parameter of the type Color. Note that in order to add this method in the inspector, this method needs to be public.

```
public class Example : MonoBehaviour
{
    // This method can be added as listener in the inspector
    public void OnColorChanged(Color color)
    {
        // your code goes here
    }
}
```

ColorInput

The ColorInput uses Unity's InputField to make one of the values of the ColorPicker editable by the user. You can indicate which value and the min and max value the value should range in between (higher or lower input will automatically be clamped). You can also set the format to use.

See also: Formatter.

Label

The ColorLabel uses Unity's Text component to display one of the values of the ColorPicker. Just like the ColorInput, you can set the min value, max value and the format.

See also: Formatter

Formatter field

Both ColorInput and ColorLabel have a formatter field. In this field you can set a format to use. If you want a specific format, you can almost certainly create it if you provide the correct format. The following links could be followed to learn how to create your format. I personally prefer the Custom numeric format which is also used in my demo's, but they both work. Googling the name will send you to the same site in case the link breaks. But once on the site make sure the .NET framework on the site is set to the one Unity is currently using. (.NET 2.0 at the time of writing this readme)

Standard numeric format strings: https://msdn.microsoft.com/en-us/library/dwhawy9k(v=vs.80).aspx

Custom numeric format strings: https://msdn.microsoft.com/en-us/library/0c899ak8(v=vs.80).aspx

Hexfield

The ColorHexfield is a script that you can use to display the current color of a colorpicker as a hex to the user. It has a few settings that you can change like whether or not you want to show a hashtag inside the field, whether the alpha is displayed and which type(s) of hex input it accepts.

ColorSlider

The ColorSlider uses Unity's Slider component to control one of the values of the colorpicker (red, green, hue, etc.). It is advised to add it using the MenuItems, as then the HandleRect and Gradientbackground will be added automatically with the correct default settings and ready to go.

GradientBackground

The GradientBackground is one of the few components that can't be added directly through the MenuItems, but is added when you create other Components through the MenuItems. (Sliders and some color previews)

This script is made so that it can create any gradient you would like, so you could in theory just use it for another background not related to a colorpicker.

- **Display Checkboard:**
  If checked, a checkboard will be drawn behind the gradient. You should only check this box when your gradient is (or can be) transparent, as it will use 4 vertices extra to draw it.
- **Checkboard size:** (only visible when checkboard is enabled)
  The size of the checkboard, bigger value means bigger squares.
- **Gradient**
  If checked, this component behaves normally, if set to false this could behave more like a slider fill area. Checking this box is slightly better for performance as setting it to false uses 2 extra vertices.
- **Direction**
  Self explanatory
- **Center type**
  What controls the center position of the gradient. In most cases, this value should be set to custom with the center position set to 0.5. You can however set it to any of the color types, if you do that and you uncheck gradient it can behave like a fill area for a slider (See the alpha GradientBackground of the "GameObject/UI/ColorPicker/Preview/Seperated")
- **Center Position** (Only visible if center type is set to custom)
  Where this center position of the gradient is (0 = start, 0.5 is half way and 1 is at the end). You should keep this value at 0.5 in almost every scenario.
- **Border size**
  The border size in pixels
- **Color**
  The background color (Only drawn when Border size is > 0) Note that this background color is also drawn when the gradient is transparent. It that case I would advise to either check DisplayCheckboard or set the border size to 0 and draw your border yourself.
- **Colors amount**
  Out of how many colors this gradient should exist. Note that this and the following colors are set to defaults whenever you add a slider through the MenuItems. It is advised to use the MenuItems to set the majority of these settings as the default values are a great starting point (especially for hue slider backgrounds).
  - **Type**
    What controls this Color of the gradient. This can be set to Custom for a custom color, or to any of the ColorPicker's values.
  - **Fixed 'X'**
    When the type is set to any of the colorpickers values, the gradient will go from that value 0 to that value 1 (normalized). The other values of the colorpicker that influence the resulting color can be set to a fixed value (normalized between 0 and 1, with the exception of alpha which is between 0 and 255)

Box

The GradientBox component uses the Slder2D to control 2 of the values of a ColorPicker. Note that these values need to belong to the same group, with the groups being RGB, HSV and HSL (So for instance, you can’t mix Hue with red, green or blue). The remaining value controls the look/color of the box gradient and is controlled by the ColorPicker. Like with the GradientBackground, you can choose to set this value to a fixed value which is normalized between 0 and 1.

Slider2D

This component is a 2 dimentional slider. It is based on the Unity Slider. You can set a min and max value, the values itself and whether or not to inverse the X and/or Y values.

Circle

The GradientCircle component uses the SliderCircle2D to control 2 of the values of a ColorPicker. Note that these values need to belong to the same group, with the groups being RGB, HSV and HSL (So for instance, you can't mix Hue with red, green or blue). The remaining value controls the look/color of the circle and is controlled by the ColorPicker. Like with the GradientBackground, you can choose to set this value to a fixed value which is normalized between 0 and 1.

SliderCircle2D

This component is a 2 dimensional slider based upon an angle (ranges between 0 and 360) and a distance (ranges between 0 and 1). It is based upon the Unity Slider but with some modifications to make it 2 dimensional and circular.

- **Corners**
  This value must at least be 6. Please keep this value as low as you think you can keep it while still looking good. As increasing the corners decreases performance. A high number of corners might increase the accuracy in some cases because the color displayed by the GradientCircle is not 100% accurate in all cases. This is because of how the triangles are positioned (they all combine in the center). If the center is not the same color at any degree (Like when displaying any of the RGB values) then the center won't be accurate as with the current triangle positions I can't possibly lerp between the colors correctly. Values which don't have this problem are: HS, HV, HS and HL. For all the other values, setting the corners to 60 or higher will make this problem barely noticeable and increase the accuracy of the display.
- **Inverse angle**
  Inverses the angle of the circle (clockwise/counterclockwise). Note that you can also rotate the RectTransform if setting this value alone doesn't give you the desired result.

Preview

To preview your color, you can use different approaches depending on your needs. Using the MenuItems you can create 6 default ones

- **Opaque**
  Uses the GradientBackground component to display an opaque version of your color
- **Transparent**
  Uses the GradientBackground component to display the color. When the alpha is 0, you'll see a checkboard but since the alpha is 0 not the color.
- **Half Transparent**
  Uses the GradientBackground component to display the Color from full opaque in the top to color's transparency in the bottom. That way, you can always see the color, even when the transparency is 0, while still displaying the transparency.
- **Seperated**
  Uses the GradientBackground component to display an opaque version of your color and the alpha as a black/white bar below it (The same way Unity displays color in the inspector).
- **Graphic**
  This adds an empty image component with a PreviewGraphic script attached. You can add the PreviewGraphic script to any Graphic component, see Demo1 for more info.
- **Renderer**
  This adds a cube to the scene with a PreviewRenderer script attached to it. The PreviewRenderer can be added to any MeshRenderer. And will work as long as the Material's main color is used by the MeshRenderer.

If you need to display the color in a way that isn't supported above, you can always listen to the OnColorChanged event of the ColorPicker and use that color to color you own custom component or material.

Eyedropper

The Eyedropper is a script which is usually added to a button. Which calls ColorEyedropper.Activate when clicked. When added through the MenuItems this is done automatically, but in case you add it in a different way, make sure you call Activate at some point to make it function.

There are a few settings that you can change in the inspector.

- **Changes Color Instantly**
  When set to true, it changes the current ColorPicker's color instantly while hovering over a color (else it would only change after clicking). Advised is to keep this unckecked in most cases. As holding it over components that change appearance when the color changes might start it to quickly change between different colors as the color under you mouse cursor changes when the color changes, thus changing the color again and again and again.
- **OnActivated**
  Called when activated. If you use an EyedropperPreview that shouldn't always be activated/enabled. Make sure you activate that component using this callback.
- **OnDeactivated**
  Called when deactivated. If you use an EyedropperPreview that shouldn't always be activated/enabled. Make sure you deactivate that component using this callback.

For examples of how to use these callbacks in combination with an EyedropperPreview, see the Options or the DefaultColorPicker demo scenes.

EyedropperPreview

The EyedropperPreview is a script that basically is nothing more than a zoom window. By adjusting the settings you can change the amount of pixels displayed, the size of each pixel displayed and the color of the borders. Note that this component is quite expensive on vertice usage, as well as overall performance due to overhead on Texture2D.ReadPixels. Therefore, it is advised to deactivate this component when not in use! Also, in order to make this display accurate and update correctly, it uses a separate drawcall, so keep this in mind.

- **Type**
  The amount of pixels visible on screen can be linked to 3 values, in type you set to which of the 3 values it is linked. This can be: Pixel size, Horizontal pixel amount and Vertical pixel amount.
- **Pixel size**
  The size of one pixel in pixels. So for instance, when this value is 3, every 1 pixel under the mouse cursor will be enlarged to a block of 3x3 pixels in the preview.
- **Horizontal pixels**
  How many pixels there will be displayed to the left or right of the center pixel. Total amount of horizontal pixels is: “1 + (2 * horizontalPixels)”.
- **Vertical pixels**
  How many pixels there will be displayed below and above the center pixel. Total amount of vertical pixels is: “1 + (2 * verticalPixels)”.
- **VisiblePixels**
  How many pixels are displayed by this preview. Note that this is not the actual size of the rect, but the amount of pixels under the mouse where this component focusses on. This value is calculated based upon the following formula:
  ```
  totalHorizontalPixels    = 1 + (CeilToInt(horizontalPixels) * 2)
  totalVerticalPixels      = 1 + (CeilToInt(verticalPixels) * 2)
  VisiblePixels            = totalHorizontalPixels * totalVerticalPixels
  ```
  You should try to keep this value low, as this is better for performance.
- **Vertex count**
  This value indicates how many vertices will be used with the current settings. You should try to keep this value as low as possible but it <u>must</u> stay below 65000! In order to achieve this you should try to keep the amount of pixels visible low. You can do that by changing the values above or changing the size of the RectTransform. This value is calculated using the following formula
  ```
  VertexCount = 8 + (VisiblePixels * 4)
  ```
- **Border size**
  The width of the borders in pixels
- **Background color**
  The color of the background (and thus the borders)
- **Selection box color**
  The color of the selection box in the center of the component
- **Activated**
  If true, this component will display the pixels under the mousepointer, if false not. If you don’t use or display the component, make sure this is false or the component/gameobject itself is disabled/inactive. As keeping this activated costs resources.