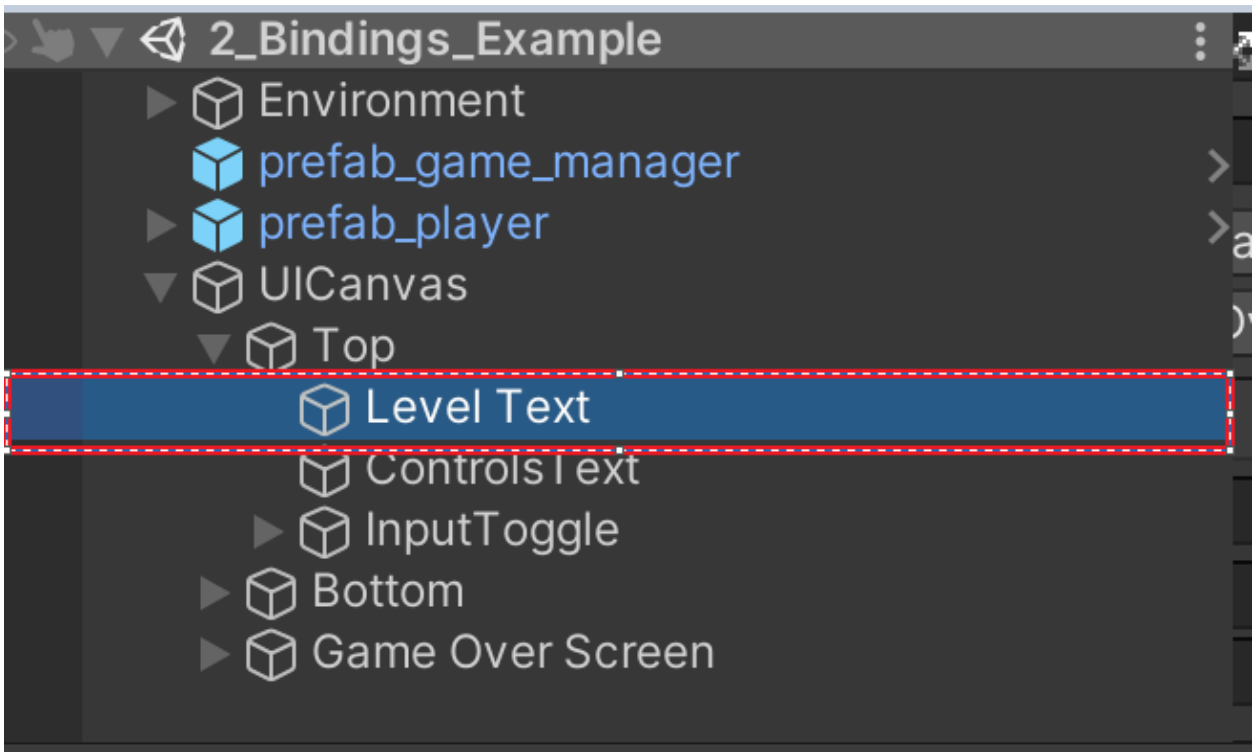
2. Bindings

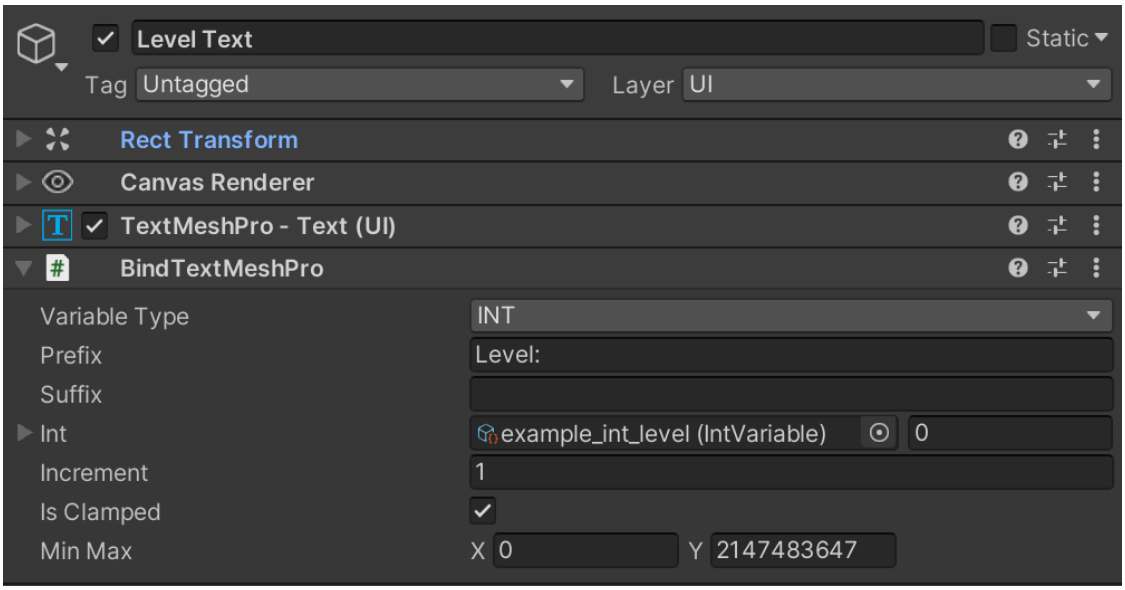
Table of Content

BindText / BindTextMeshPro

Binds a text / text mesh pro to a variable (currently supports only the 4 types : bool, int, float, string). Find the **Level Text** GameObject in the hierarchy.

UICanvas/Top/LevelText.



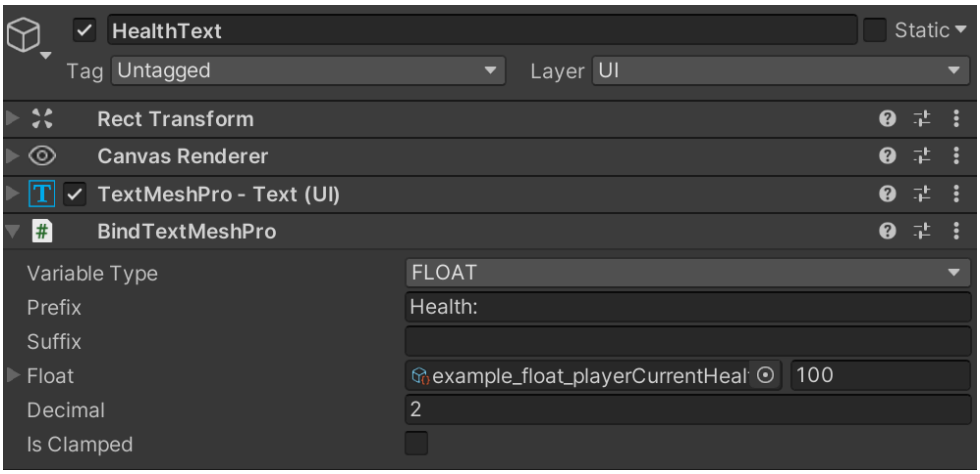


Let's go through the properties:

- **Prefix** adds text before your variable value.
- **Suffix** adds text after your variable value.
- **Increment** adds a number to the int. It mostly used for Level index (as we start at 0, but we want to display 1 to the player).
- **IsClamped :** enable clamping the value shown.
- **Min Max :** minimum and maximum used for clamping the value shown. So in this case, if your variable goes below 0, the value shown in the text will be minimum 0 (won't display negative numbers).

The **HealthText** also has a BindTextMeshPro component.

UICanvas/Bottom/HealthText.

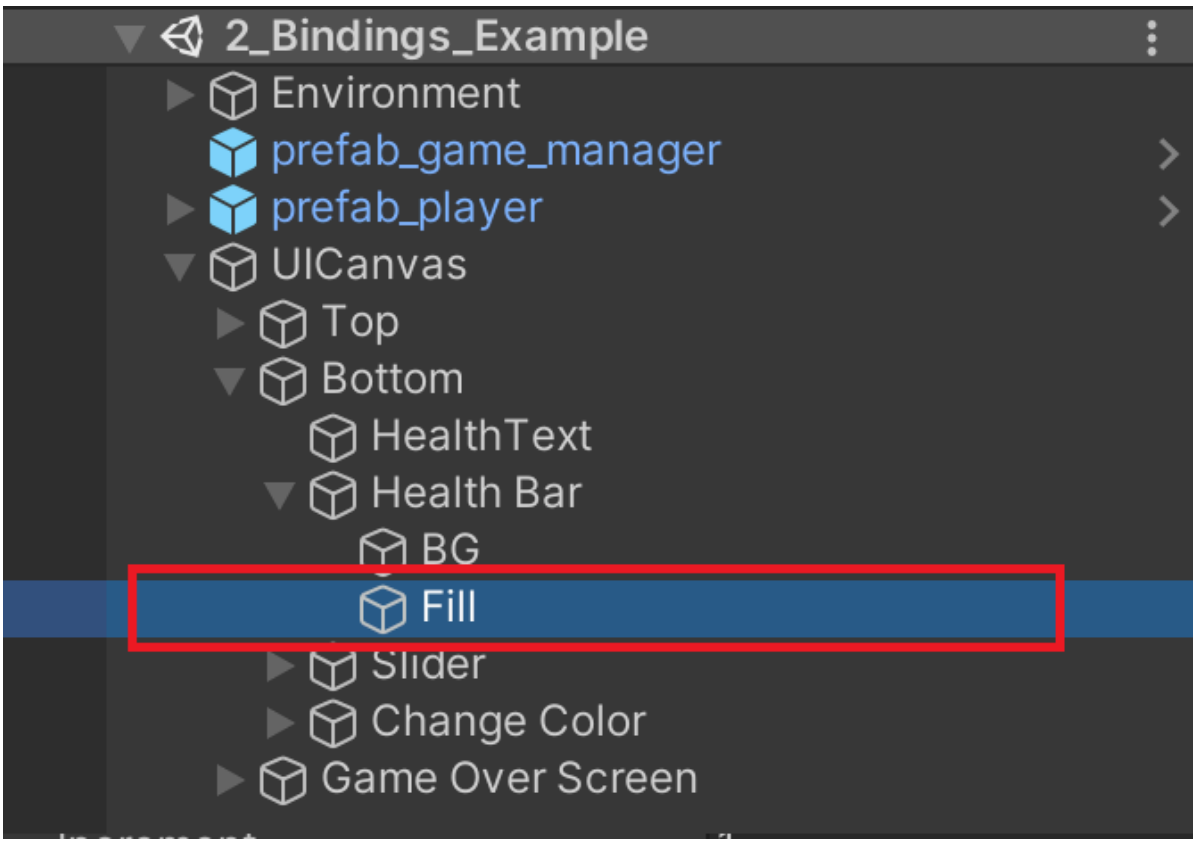


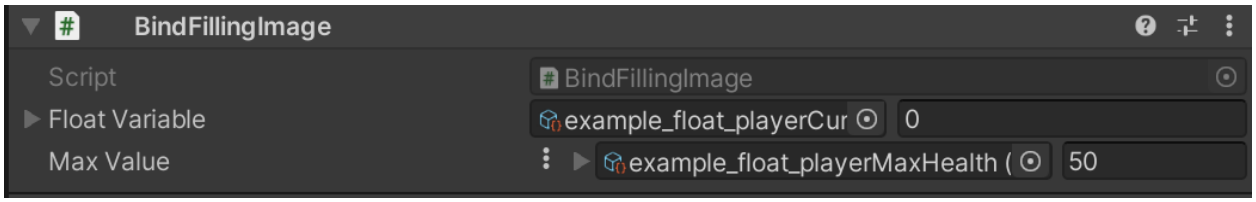
- **Decimal Amount**: how many decimal numbers we want to display.

BindFillingImage

Useful for health, stamina, whatever you need in a filling bar. This is what updates the health bar when you change the health variable.

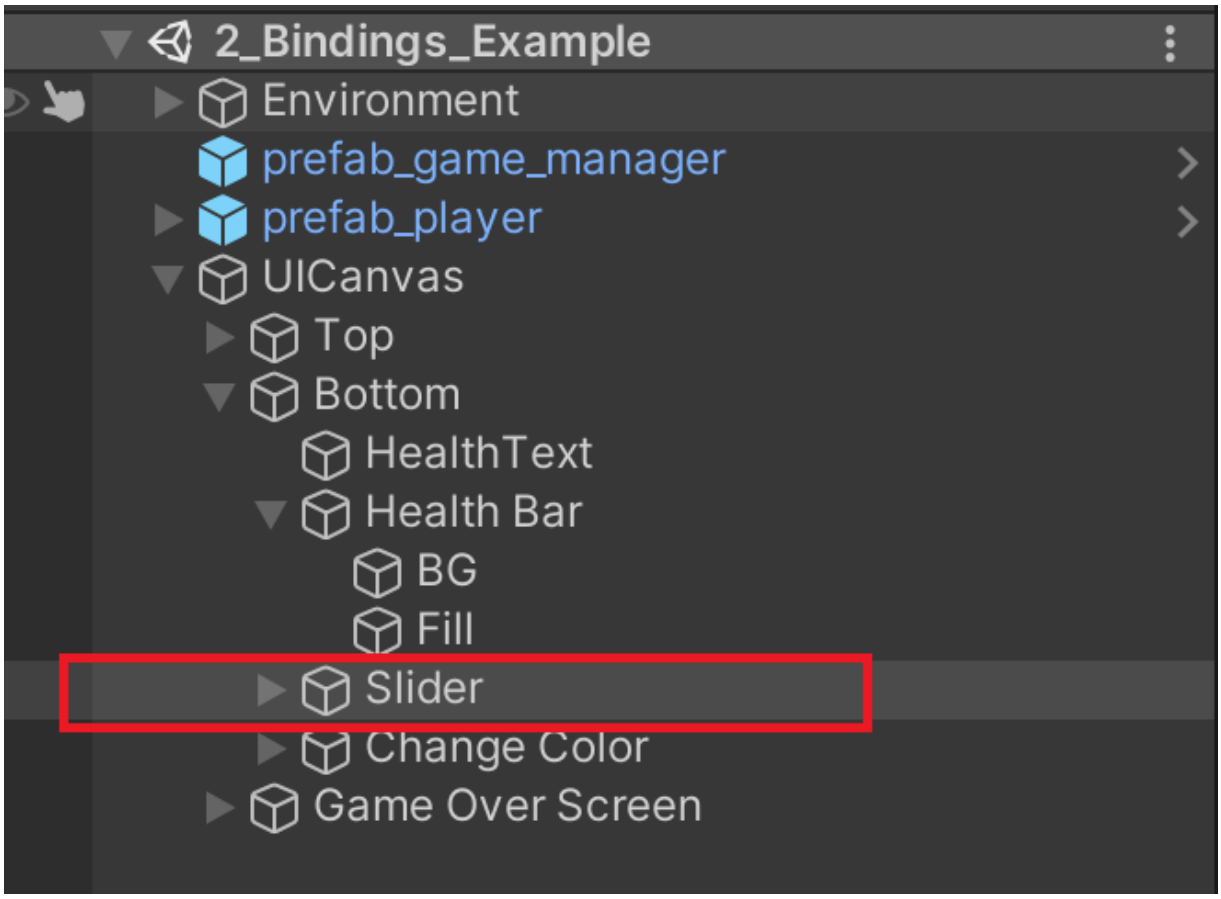
UICanvas/Bottom/Health Bar/Fill.

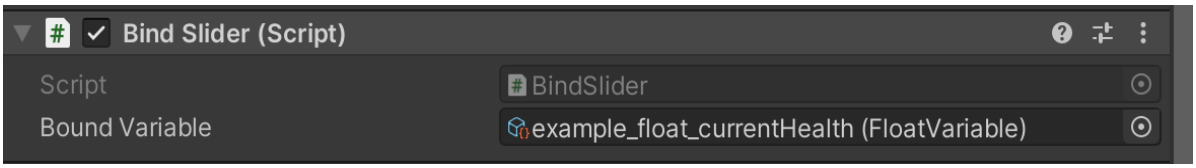




BindSlider

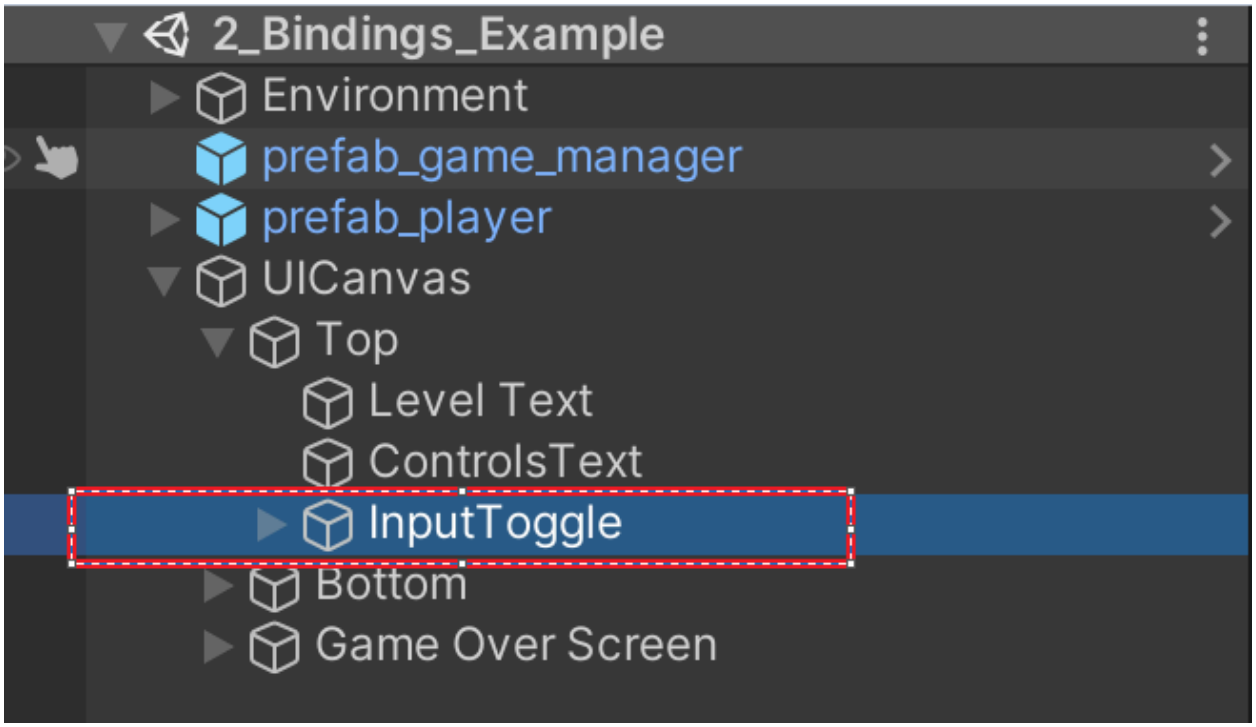
Modifies a float variable using a slider. Useful for debugging. Try messing around with the health slider during play Mode.

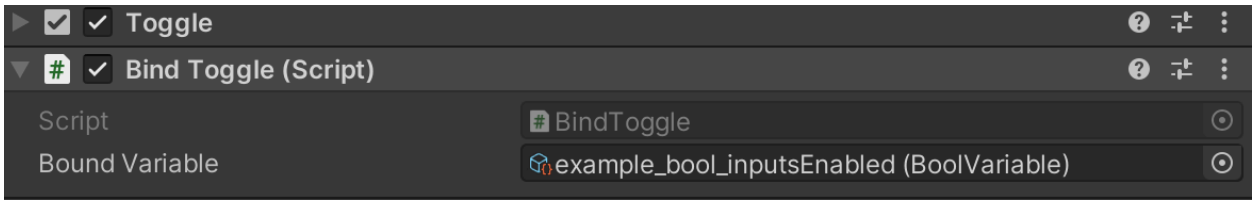


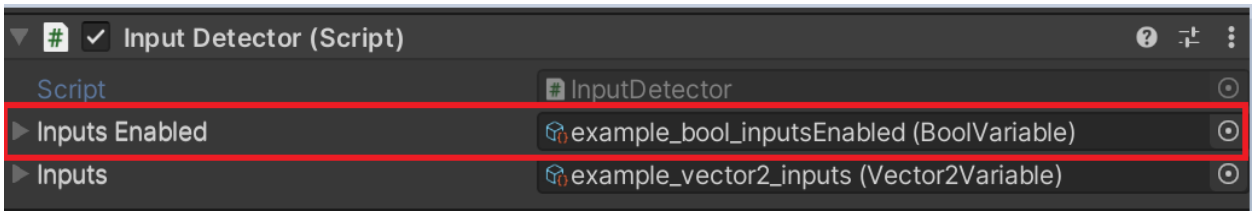


BindToggle

Modifies a bool variable. Useful for debugging. Try disabling the inputs and then see that your character does not react to inputs anymore. It is because the input detector reacts to the example_bool_inputsEnabled variable.

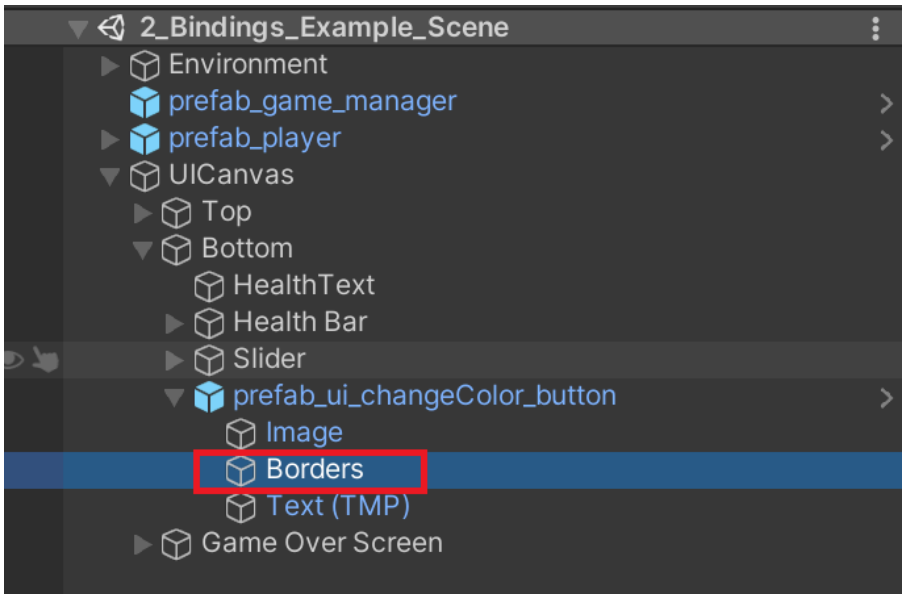


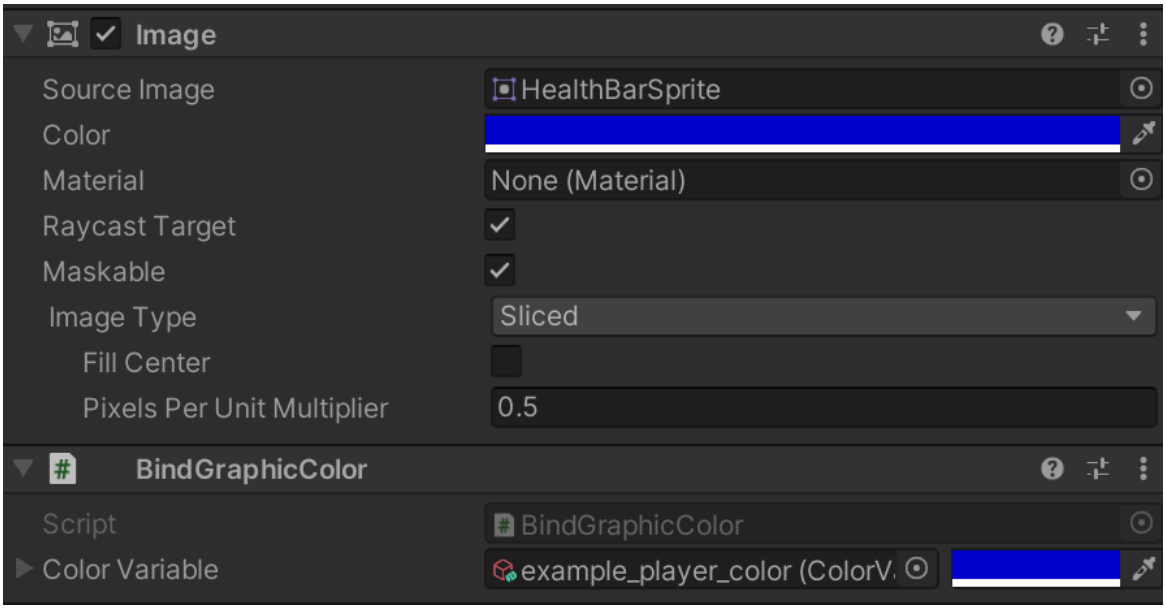


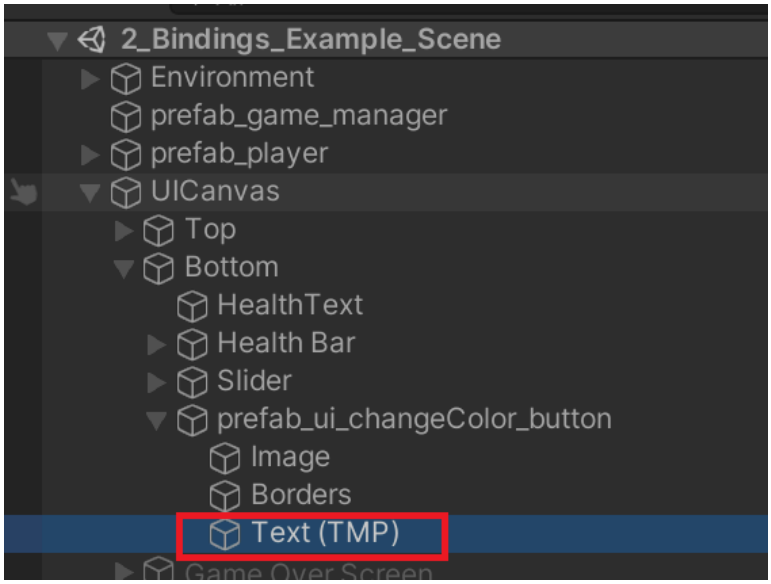


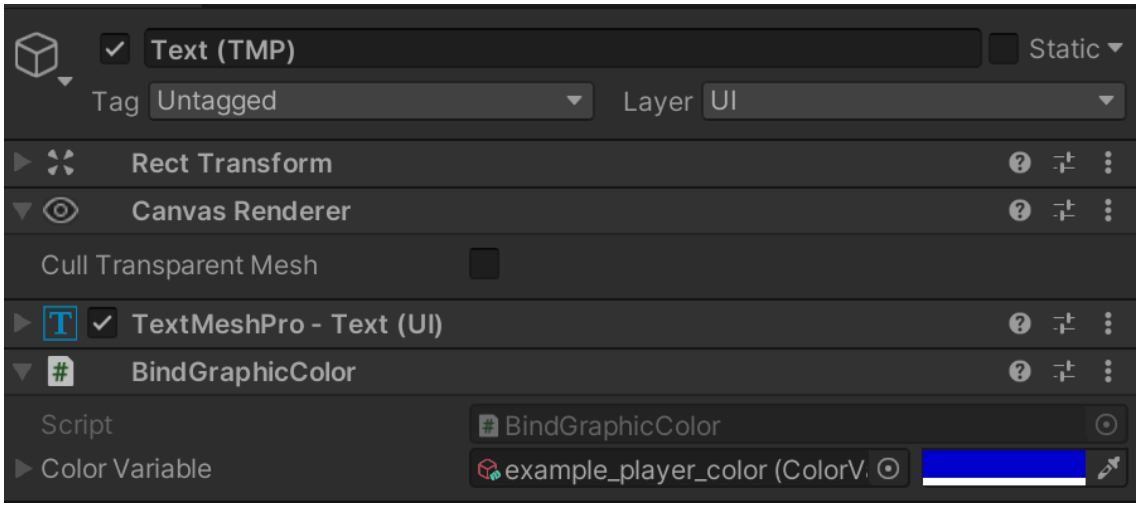
BindGraphicColor

Binds any UI graphic element to a color variable. Press the **Random Color** button in the scene a few times. This updates the color of the image, and the color of the text when the color variable changes. All UI elements inherit from the Graphic class, making this component reusable for various UI elements.






2_Bindings_Example_Scene
Environment
prefab_game_manager
prefab_player
UICanvas
Top
Bottom
HealthText
Health Bar
Slider
prefab_ui_changeColor_button
Image
Borders
Text (TMP)

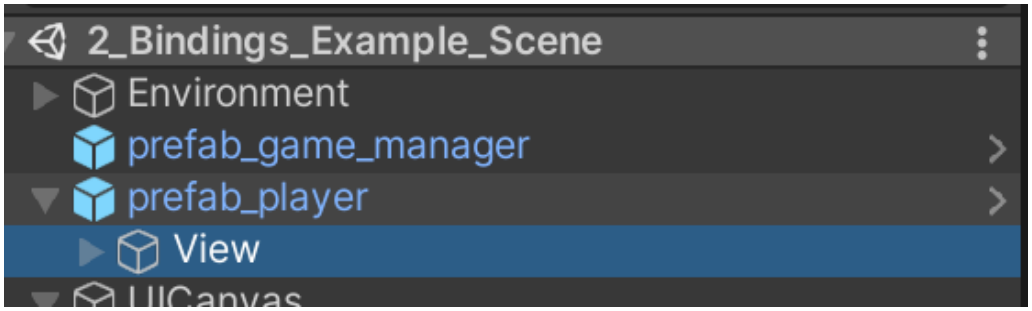

Text (TMP)
Static
Tag
Untagged
Layer
UI
Rect Transform
Canvas Renderer
Cull Transparent Mesh
TextMeshPro - Text (UI)
BindGraphicColor
Script
BindGraphicColor
Color Variable
example_player_color (ColorV

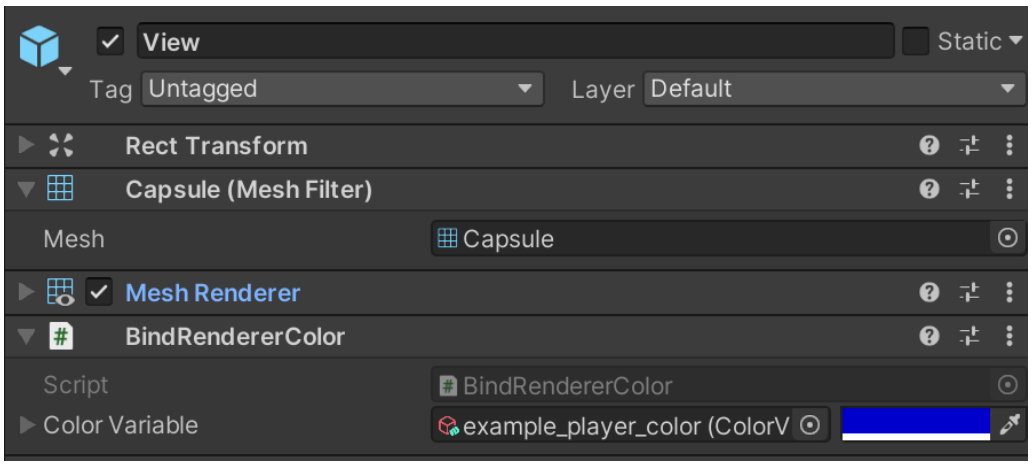
BindRendererColor

Also, if you take a look at the mesh of the player when you click on the Random Color button you will see that the color changes because the player has a component BindRendererColor.cs that binds to the example_player_color variable.

Select the player view:

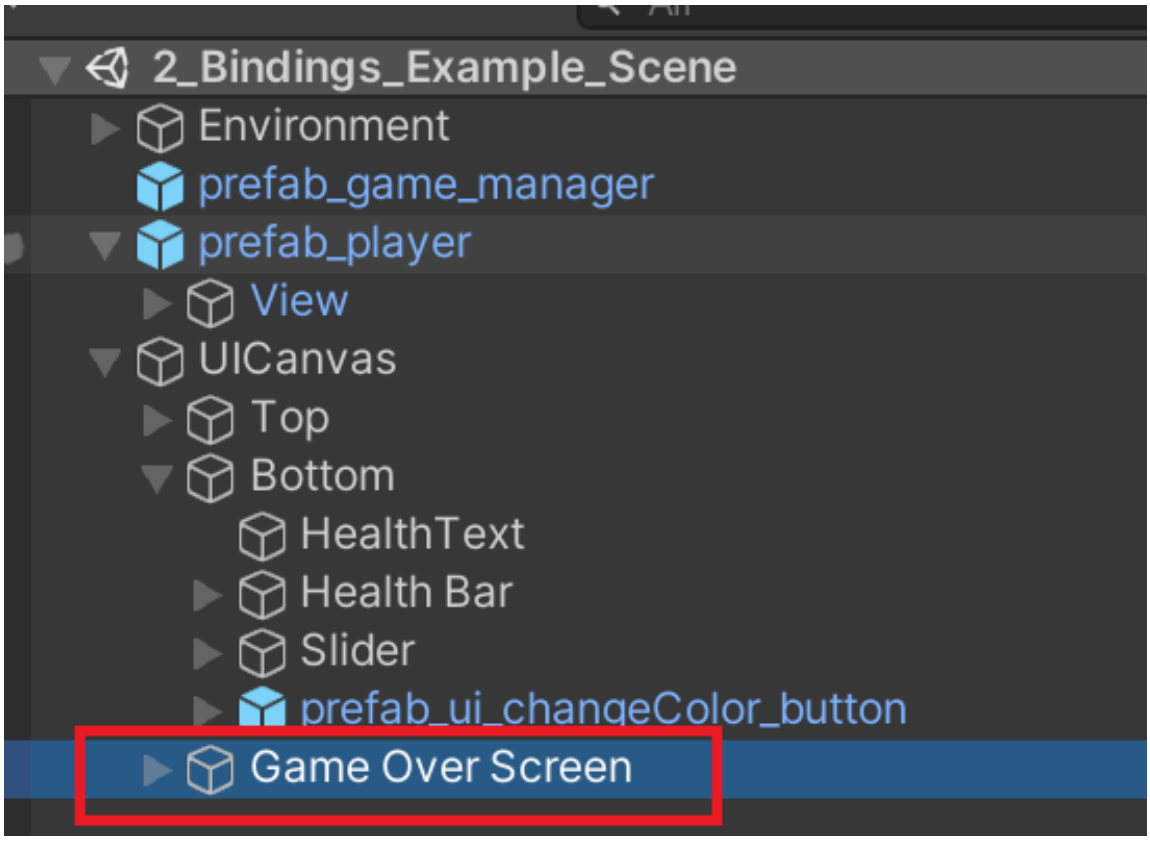
Prefab_player -> View



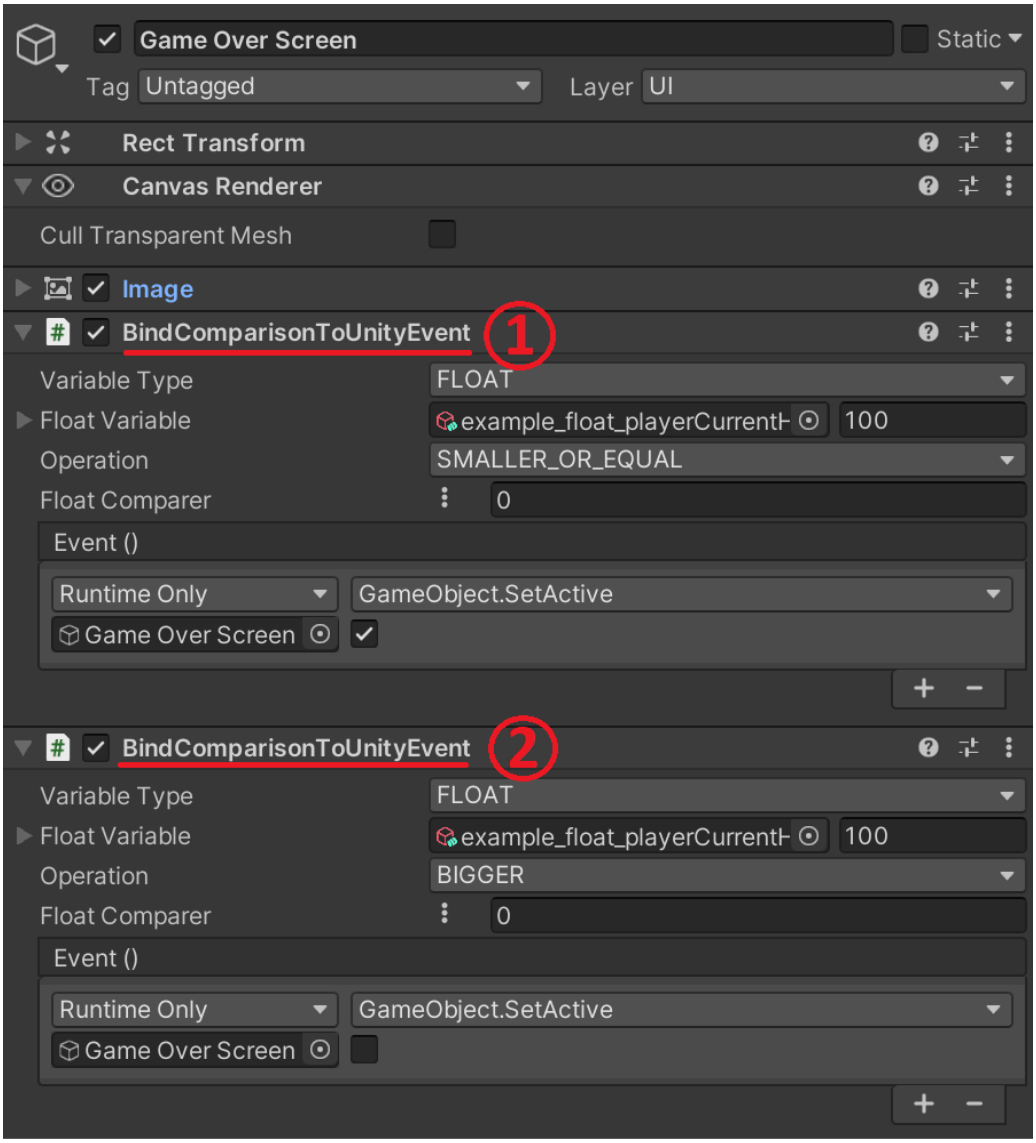


BindComparisonToUnityEvent

This one is a bit trickier but is the most useful. Find the **Game Over Screen**.



You can see that there are 2 BindComparisonToUnityEvent components on this panel.



Now enter play mode and set the health to 0 using the health slider. Then, once the game over panel appears, set the health back to 100.

It might seem a bit overwhelming at first, but it is simple. Here is what happens:

The first component **(1)** does the following:

- Binds itself to example_float_currentHealth variable.
- Checks if the variable value is smaller or equal to 0 (when it changes).

- If true, triggers the unity event. In this case -> enable the gameObject.

The second component **(2)** does the following:

- Binds itself to example_float_currentHealth variable.
- Checks if the variable value is bigger to 0 (when it changes).
- If true, triggers the unity event. In this case -> disable the gameObject.

This check happens also on Start(), to make sure the conditions are validated at the start of the game (in this case the game over screen will be hidden as the health is set to 100).

This is useful for menus, pop ups, VFX and other stuff when you don't want to make a script to handle simple behaviors.

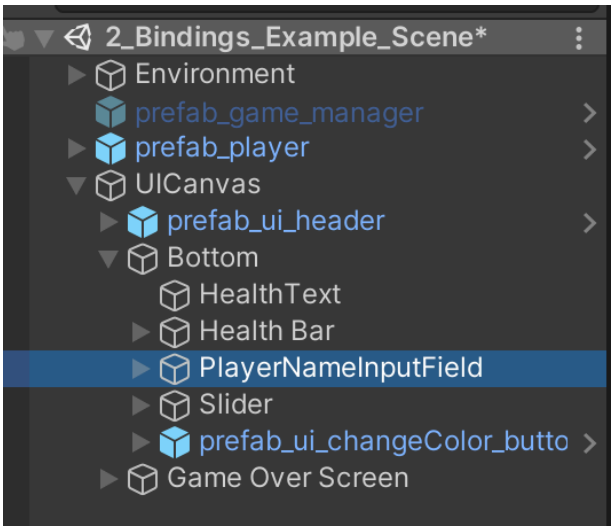
Many things can be resolved with Unity Events. I recommend using unity events for visuals feedback and juice and not for actual game logic. You can consider them as "hooks" where you can attach nonessential things to your game.

Note: Unity events creates more garbage than traditional C# events, so they are not the most performant. However, I have used them in mobile games and so far, never had an issue with them.

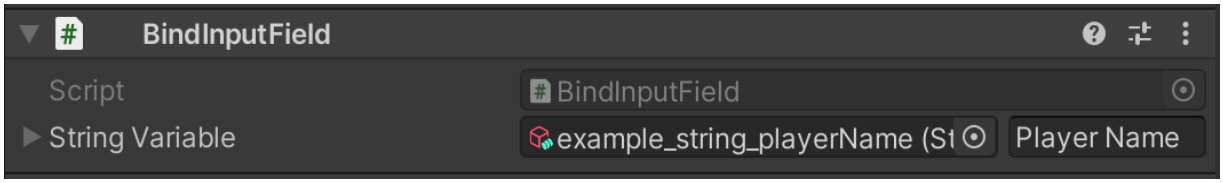
BindInputField

It can be useful sometimes to save a string from an input field, for example the Player Name. Let's see how the BindInputField bindings works.

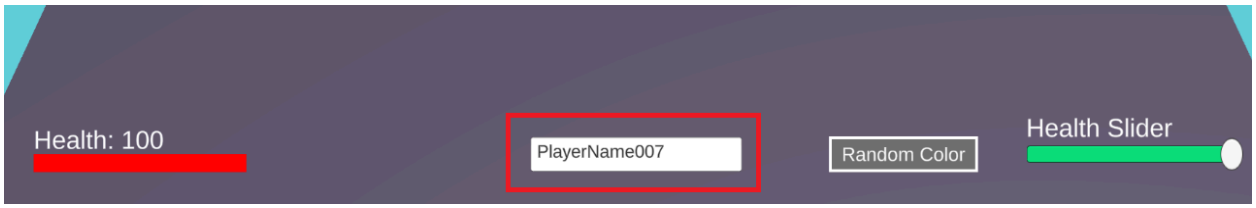
Select the PlayerNameInputField in the scene:



Now, on this object you should find the BindInputField component:



Play the game and enter a random name in the input field at the bottom of the screen.



Now stop the game and play again. You can see that the name persisted. In this example, the string variable: example_string_playerName is saved. Depending on your case, you can decide to save it or not.