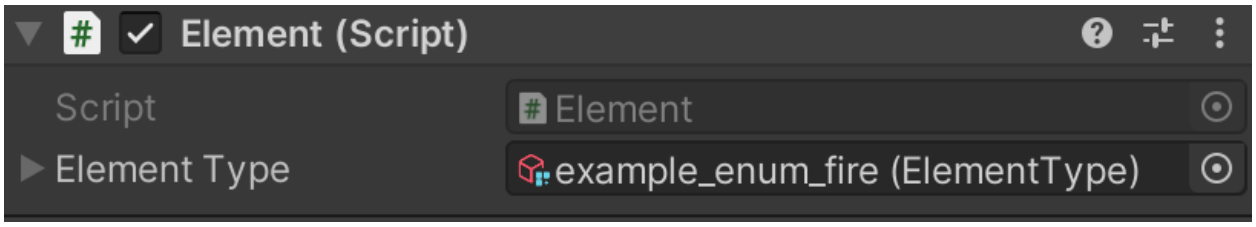
6. Scriptables Enums

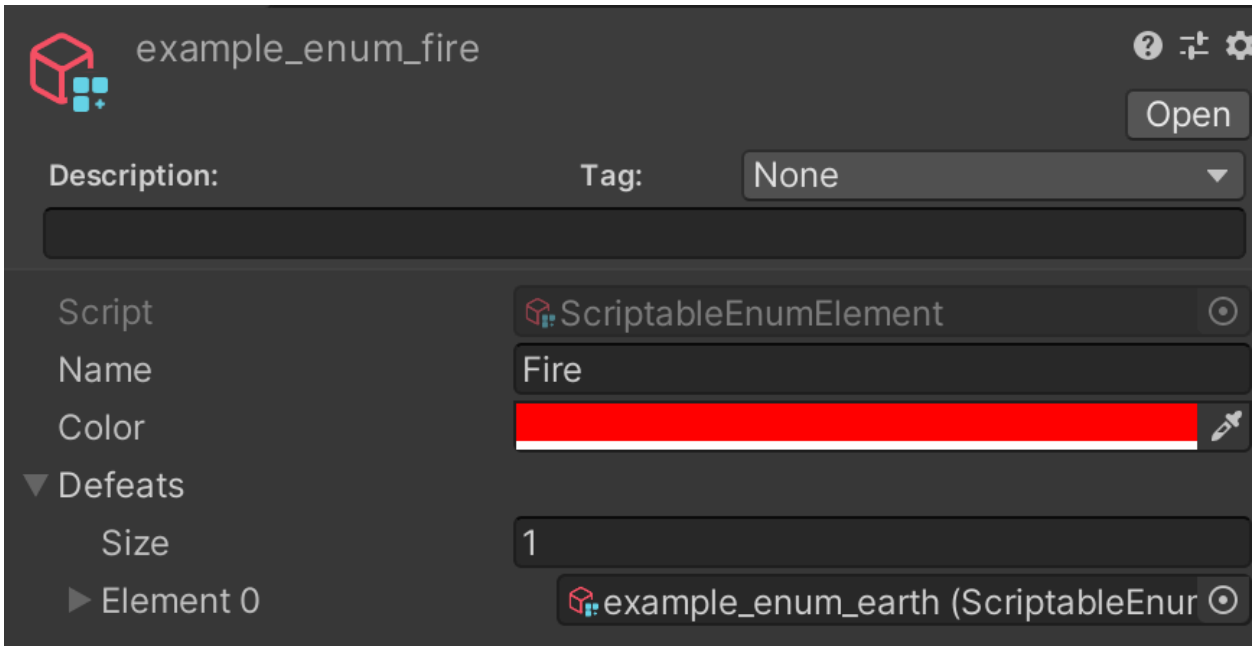
Table of Content

Scene Set up

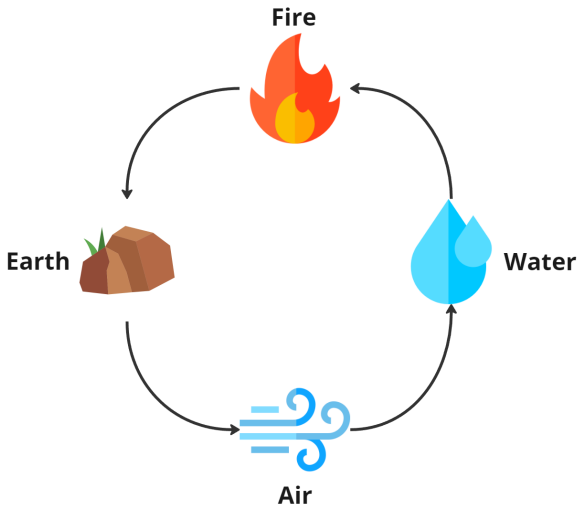
In this scene, you will find four GameObjects named "Element." Each Element has an Element.cs component attached to it, which references a different Scriptable Enum (SE). If you select the first one, you will see that it references the Fire SE as its ElementType.



If you click on this SE, you will see that it contains additional data: a color field and a list (Defeats) of ScriptableObjects. The latter list contains other Scriptable Enums that this element destroys.



If you play the game, use WASD to move the fire element and try to collide with the other elements. As indicated in the Defeats list, if you collide with Earth, you will destroy that element. However, if you collide with Water, the fire element will be destroyed. The current data is set up as follows:



How does it work?

ScriptableEnums are simply ScriptableObjects used as enums. To compare them, we just compare their references. If you open the Element.cs component located on an Element object, you can see the code.

```
public class Element : MonoBehaviour
{
    [SerializeField] private ScriptableEnumElement _elementType = null;
    private ScriptableEnumElement ElementType => _elementType;

    private void Start()
    {
        GetComponent<Renderer>().material.color = _elementType.Color;
        GetComponentInChildren<TextMeshPro>().text = _elementType.Name;
    }

    private void OnCollisionEnter(Collision other)
    {
        if (other.gameObject.TryGetComponent<Element>(out var element))
        {
            if (_elementType.Defeats.Contains(element.ElementType))
                Destroy(other.gameObject);
        }
    }
}
```

In our case, when an element collides with another, we check if the other element type is contained in our Defeats list. If it is, then we can destroy it. This approach is flexible because you can add or remove element types from this list at runtime or through code. Additionally, if you want to create a new element, all you need to do is duplicate an existing SE and rename it. That's it—no code involved.

Additional Data

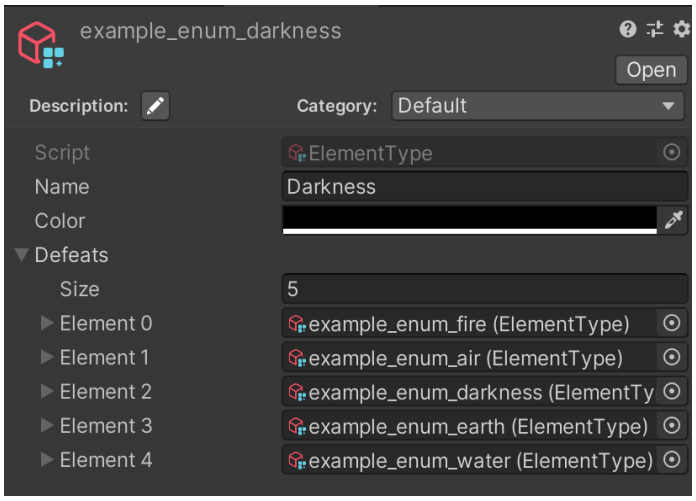
One benefit of SE is that you can add additional data to this enum since it is just a ScriptableObject. In this example, I added three fields: name, color, and Defeats list. In the code above, I use the Name field to set the text of the TextMeshPro component and the color field to set the color of the renderer. This can be very convenient. Other examples of useful data you can add to these enums include an icon, a localization ID, a VFX prefab, and more.

Exercise

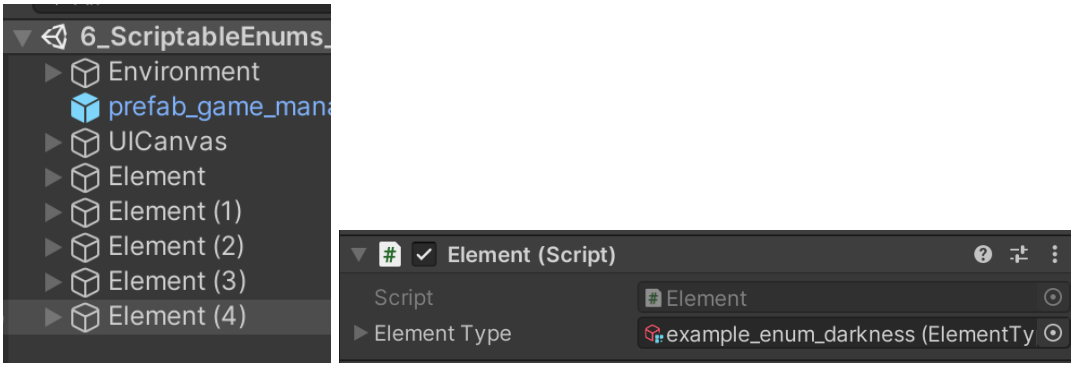
Take a few minutes to modify the data of the different elements. They are located in the Examples folder.

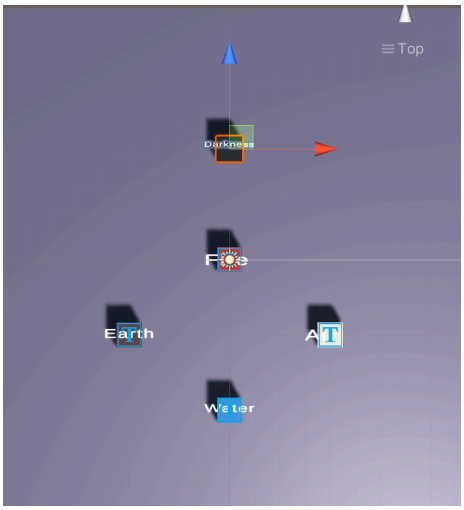
Feel free to change the default rules of "Defeats." After experimenting, we will create a new element: Darkness!

Let's duplicate an existing SE in the project folder, rename it to "Darkness," change its color to black, and add all the element types to its Defeat list. Yes, Darkness can even defeat itself!



Now, let's duplicate an existing Element in the scene, move it somewhere in the scene and assign the Darkness type.





Press play, select that Element in the scene, and then move it around manually onto the different Elements. Admire as it destroys everything!