Tutorial for using Event Listener (Observer Pattern)

There are three example cases in the TestEventScene. In all of the three examples, the circle triggers something when it hits the cube.

Example1

Look at script CircleForCollision, it only contains a lifecycle function OnCollisionEnter 2D.
In simple one-to-one cases (or sometimes simple one-to-many cases) where you can easily get the object reference you want, you DON'T have to use observer pattern.

Example2:

Look at script CircleForCounter1 and Counter1. One is on the second circle and one in on the counter UI.
When the circle hits the cube, the counter increases its number. Notice this logic is very importance. It is not letting the circle to change the counter, but have the counter changed by itself. It should make sense since increase count is an "ability" of counter. The counter should "detect" when the circle hits the cube.

How to implement event listener (without toolkit):

1. In the script where you want to trigger the event, declare an Action or Func or unityAction. (All of the three are called delegate or event. Action and Func are C# built-in and unityAction is unity built-in. You may understand them as function pointers, or containers for functions.

2. Action and unityAction accepts function with no return value, and their template variables are for parameter types. Func accepts function with return value, and the first template variable is for the return type.)

3. Get the reference of the object that you want to listen. (drag reference in inspector, or getComponent if you can)

4. In the Start lifecycle function, use += to add function from that object to the event. At where you want the event to be triggered, trigger all the functions in the event using event.Invoke() or event()

Example3:

This example is done by using EventMgr in the toolkit. Notice that it inherits singleton base class, so you need to use EventMgr.Instance.method() to call its method.

In example2, you may have noticed that the counter is not actually "detecting" the event. Everything is done by circle. They are decoupled, but not decoupled completely, since you

still have to get the reference of counter in circle script.

Using EventMgr, every event is paired with a customized string and is centralized in a dictionary container. No need to get reference in the script that you want to trigger the event.

How to use EventMgr:

1. Call EventMgr.Instance.trigger("EventName") at where you want to trigger an event.

2. Add listener in the Start function using EventMgr.Instance.AddListener("EventName", function to be triggered) in the script where you want to listener the event (usually not the same script where you trigger the event) Make sure you don't mistype the name string.

By using EventMgr to send event, the two scripts are completely decoupled, without getting each other's reference.

When should you use EventMgr:

Use it for systems like achievement system, score board, unlocking player skills which have two scripts that cannot easily get each other's reference.
DON'T abuse it. You should not use it if you are simply not bothered to get another object's reference.
When you create an event using EventMgr, think if this event would be reused multiple times. If no, better not to use it.
If there are too many events, managing the strings might be a problem. You should use a consistent naming rule for the event string names.