Q! EVENTS

Q! Events (Quantum! Development's Event System) is a simple system that allows you to easily add events to your project using Unity Events and C# Events technology.

SET UP

There are two main types of events: Unity and C#. Depending on what and how you would like to configure the events, you should choose one or the other. There are two event subtypes within the Unity events: *Area* Event and *Trigger* Event.

UNITY EVENTS

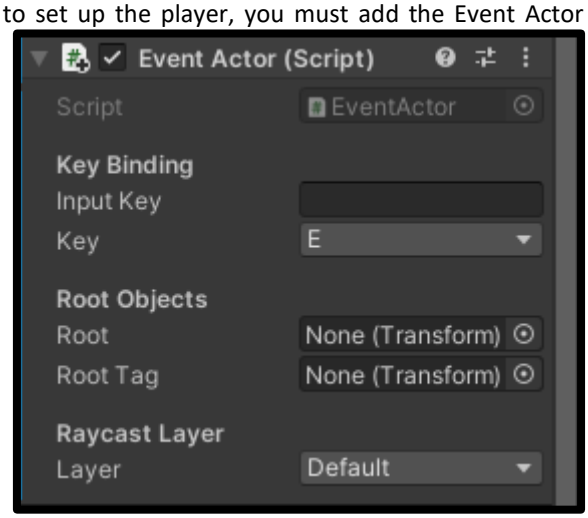
If you don't want to get involved with code and you want to make this experience as beginner-friendly as possible, you should definitely use Unity Event Triggers. The main reason is because you will be assigning everything on the inspector, you don't need to access any script nor create any new code. In this case, add the Event Trigger component to any game object in your scene… just that.

C# EVENTS

If you want to be able to dynamically create event handlers, you could use the system variant that uses the potential of C# events. There are two components under the "C#" folder, you must add the Event Publisher component to preferably an empty game object and the Event Subscriber component could be added to any game object directly or added using the Add Component method. Both are valid as you can pre-set actions to be executed when the event is triggered or you could simply assign actions using code.

PLAYER

The demo scene contains a player which was added for obvious reasons, testing. If you already have a custom player prefab, use that instead. In order to set up the player, you must add the Event Actor component to your player's camera.



This component has its variables separated in groups for easier organization.

KEY BINDING

Use the one of your preference, keep in mind that if you use Input Key, you must set Key to None, and if you use Key, you should leave Input Key empty.

ROOT OBJECT

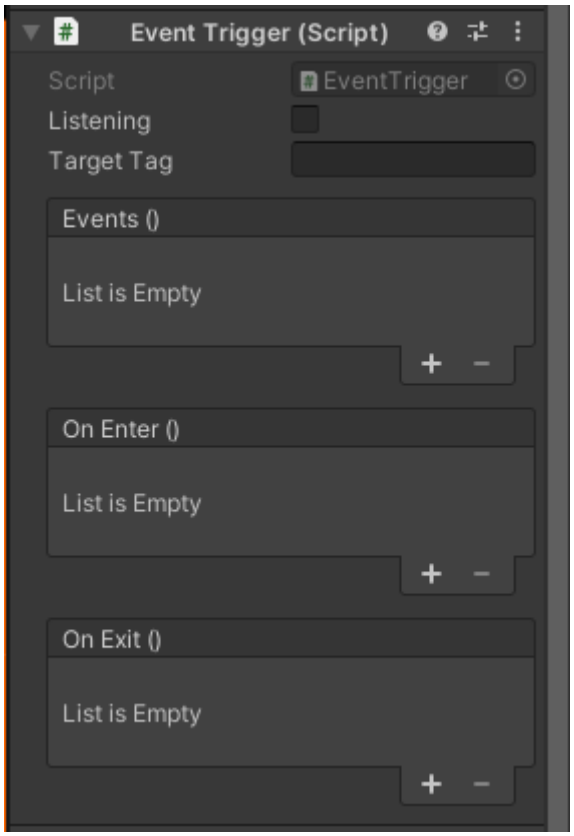
Root will be automatically set to the Main Camera Transform. If no object is 'Main Camera' tag, please assign the player's camera to this variable.

The Root Tag is the transform that will be tested when comparing tags. It should be assigned to the main transform (player's collider). If your player has the common structure (player transform, camera child transform), you should assign the player transform to the Root Tag.

Finally, the Layer is the layer that will be used when raycasting (triggers).

USAGE

Both systems are easy to use, specially the first one, using Unity Events. It is very user-friendly. Here in this image, you can see that there are a few variables. The first one “is listening” will simply ignore the trigger and won’t fire the events.



Target tag is what the name implies, the tag of the object which the component is listening to.

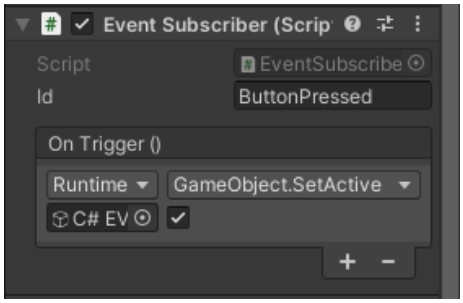
Then you can see three sets of events: events, on enter and on exit. ‘Events’ will fire when manually triggered (using Trigger public method). On Enter and On Exit will execute if the component finds a collider attached to its game object and the object with that tag has detected on trigger enter, or on trigger exit.

The main difference is that you can add the Event Actor script to the main camera, when the right key is pressed, a ray cast function will be executed. If a Event Trigger is hit, it will trigger its Events.

If no event is set within the Events variable, no action will be taken when ray casted on it. Same thing with area, you need to have a trigger collider, otherwise it won’t work and pretend as nothing happened.

C# EVENTS

There are two components regarding the C# Events: Event Publisher and Event Subscriber. The event publisher component has no variables to set. There is one public method called Trigger which needs a string **id** argument. When you call this method and pass an **id**, the event publisher will send an event signal to all event subscribers with that **id**, if any of them has the same **id** variable, it will fire the events you assigned in “On Trigger”.



For example, I call the method Trigger and pass as an argument: “Destroy Base”. If I have any subscriber with that string assigned to its ID, then the events will fire.

There is one very important difference with the event triggers using Unity, you can dynamically set the events fired when called. There are two variables within the subscriber event: On Trigger and OnAction.

```
/// <summary>
/// Initializes the EventSubscriber component.
/// </summary>
/// <param name="eventID">Event ID to listen to.</param>
/// <param name="onAction">Coded actions to be fired when triggered.</param>
/// <returns>This event subscriber.</returns>
0 referencias
public EventSubscriber Initialize(string eventID, Action onAction)
{
    this.eventID = eventID;
    this.onAction = onAction;
    return this;
}
```

The second variable cannot be modified using the inspector as it is a System.Action. You can instantiate a subscriber and code the actions that will be fired when triggered. Use this to your advantage.