**SendMessage() between components**

**Objective:** We are going to use a messaging model which is really powerful and useful moving forward, to make sure that when we hit something, controls get enabled or rather disabled.

1. Go to the **WaypointCircuit.cs** file in your standard assets and go to **line 64.**

We have **Length == 0**. Now, I've had warnings from Visual Studio about that and quite rightly. WHY DO YOU THINK THIS IS WRONG? Explore the code and find more info about Length.

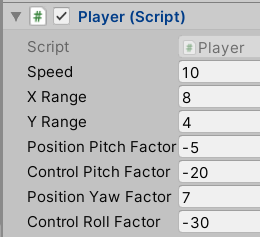
ANSWER: We're trying to compare a **float**. Length is a float here, for some reason it has a capital L even though it's a variable. If you want to change it its only the ones on lines 20, 64, 66, 69, 173, and TWO on 179.

More importantly you can’t compare a float to ZERO. This will cause bugs. So lets replace the **==0** with **<=Mathf.Epsilon**



All right lets move on

1. Lets go to your **SpaceShip** and minimize all components except the Player script.



Now our script has gotten complicated. This is the point where we need to create another script. Before we move on let’s clean up the Player script.

Our Player script currently deals with only with movement except **OnTriggerEnter.** I would say, this **OnTriggerEnter** belongs somewhere else.

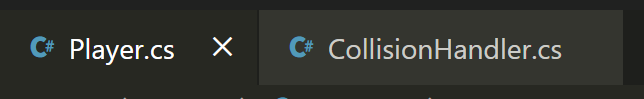
1. Click on Player and lets Add Component. What do we need? are we managing collisions? My gut tells me CollisionManager but we don’t EVER want to call a class Manager. Its not very descriptive. So thanks to [StackOverflow](https://stackoverflow.com/questions/1866794/naming-classes-how-to-avoid-calling-everything-a-whatevermanager) we can see some alternatives.

|  |  |  |  |
| --- | --- | --- | --- |
| Coordinator | Builder | Writer | Reader |
| Handler | Container | Protocol | Target |
| Converter | Controller | View | Factory |
| Entity | Bucket |  |  |

If we’re going to have a component that deals with Collision lets call it a **CollisionHandler.** Add it to the PlayerShip

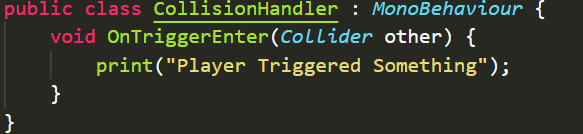
1. Lets move the script into the appropriate folder

What code to we want to move from **Player** to **CollisionHandler**?

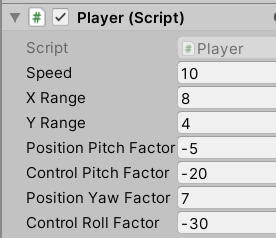
Lets open up both scripts. .

On the far right there should be a **Split** button  it should open your script on the right so you can easily compare the 2.

1. Lets move the **OnTriggerEnter** method to the **CollisionHandler**. Also lets delete the **Start** method from **Player.cs**.



1. Go back to Unity press Play and check that collisions still trigger the message.
2. Now we want to specialize the Player script. Its really about Player Movement now. Now we don’t just want to rename the class cause we’re going to lose our SerializeField values. Lets take a screenshot of it first using Snip

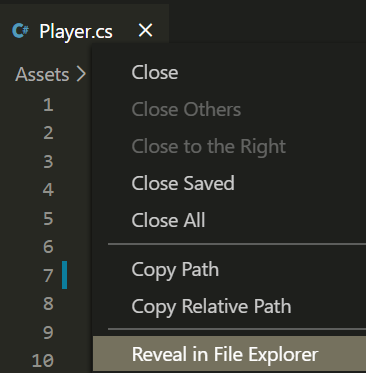


1. Lets rename it to **PlayerController**. First lets rename the class.

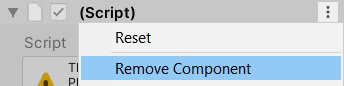


1. Then lets rename the actual file. Right click on the **Player.cs** tab in VScode and look for “**Reveal in file Explorer**”. Right click this file and rename it to **PlayerController.**

**REMEMBER THE CLASS NAME AND FILE NAME HAVE TO MATCH.**



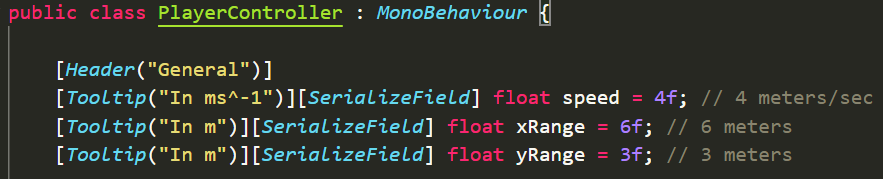
1. This move broke the Player Ship link to the script so in the inspector remove the component and add the **PlayerController** to it.



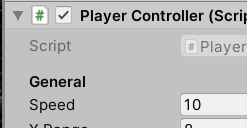
1. Fix your values and have them match to your screenshot.
2. Notice if you try to apply overrides you cant. So drag your PlayerShip from the Hierarchy on to your Spaceship prefab in your **Assets>Prefab** folder to re-link them.

What we’re putting into practice here is called the [**Open Closed Principle**](https://stackify.com/solid-design-open-closed-principle/)

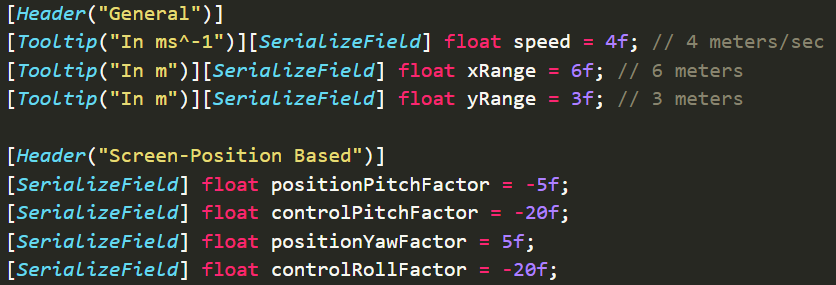
1. Open the PlayerController script.
2. Above all your Tooltips lets add this code **[Header(“General”)]**



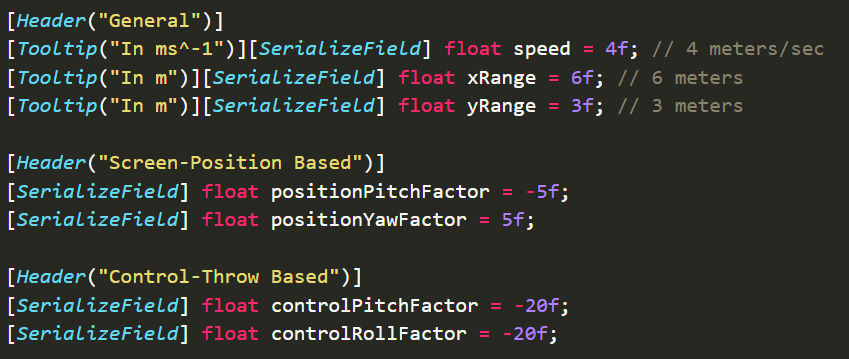
1. Go back to Unity and see that we created a group.



1. Above your SerializeFields lets created another one called **Screen-Position Based**



1. Lets created one more called **Control-Throw Based** and group everything accordingly.



1. Lets rename the **speed** variable to **controlSpeed** and make sure it changed in Unity. If it didn’t then undo and try again lol
2. We are done with the **PlayerController**. Lets close the script and minimize the component.

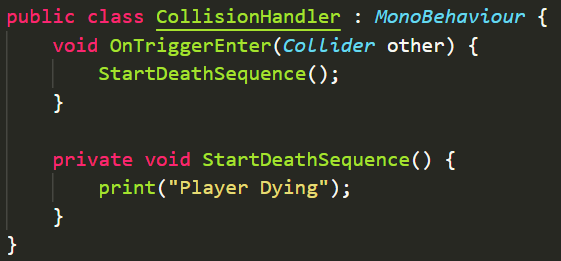
**SendMessage()**

So now that we split our Player script into two, we've now got a CollisionManager. In the future we may have some other script. How are we going to communicated between them? So this paradigm of having things talk to each other through messages like this. Is something called the [actor model](https://en.wikipedia.org/wiki/Actor_model).

**Messaging Other Components**

SendMessage(“Method Name”) – calls methods called MethodName() on all other scripts on the same GameObject.

1. In our **CollisionHandler** script lets create a **StartDeathSequence()** method and call it inside of **OnTriggerEnter**



**Challenge:** Use the SendMessage() method to freeze player controls on impact.

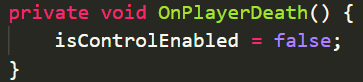
1. Create a **OnPlayerDeath** method in **PlayerController** than just prints **Controls Frozen**

|  |  |
| --- | --- |
| **CollisionHandler.cs** |  |
| **PlayerController** |  |

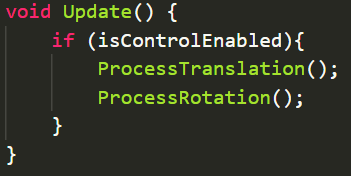
1. Lets put next to the **OnPlayerDeath** method in **PlayerController** a comment that says **//Called by String reference**
2. Test it in Unity. Press play etc
3. One last thing lets actually freeze the controls. Under float xThrow, yThrow; (line 22) lets create a **bool** called **isControlEnabled**



1. In **OnPlayerDeath** method change the print statement to **isControlEnabled = false;**



1. In the **Update** method we’re only going to process the Translation and Rotation if isControlEnabled is true.



1. Lets test in Unity. As soon as I hit something I can’t move.