**Public Methods in Unity C#**

**Objective:** In this lesson we are going to set up a scoreboard and use public methods that will allow two of our classes to communicate with each other.

Now where is our score going to be handled?

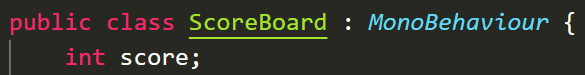
1. Lets create a C# script called ScoreBoard.cs
2. Now every time we destroy an enemy we want the score to go up. That means that the 2 scripts must communicate with each other.

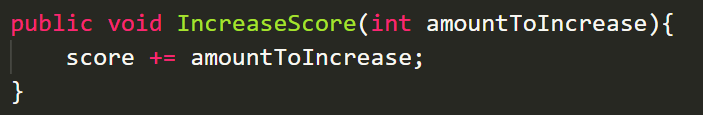
Encapsulation Review

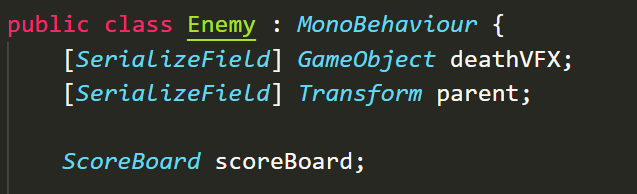
* En-capsule-ating: putting in a capsule
* Different parts of your code have a “need to know basis” level of access
* ie. don’t let everything access everything else

|  |  |
| --- | --- |
| **Class A** | **Class B** |
| Private MethodOne()  Public MethodTwo() | MethodOne()  MethodTwo() |

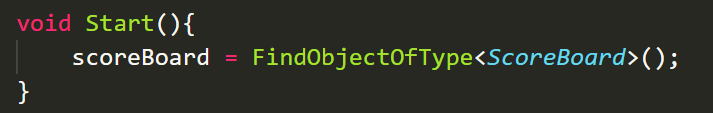
1. Open ScoreBoard script
2. Create a variable to keep track of score



1. Delete the Start and Update method
2. Lets create a public method called IncreaseScore and have it accept an Argument to determine how much to increase the score by.
3. 
4. Create an Empty GameObject and name it ScoreBoard. Rest its position.
5. Attach ScoreBoard.cs script to this GameObject
6. Open up **Enemy.cs** script
7. Lets create an Instance of the **ScoreBoard** script called **scoreboard**. Now remember another way of thinking of this is if I had a class named Dog then my instance is a VERSION of DOG. So it can be a dog name. Cow, Ori, Kairi



1. Create a **Start** method in **Enemy.cs**
2. In the Start method lets assign THE scoreboard to this variable



Now you have to be careful with FindObjectOfType. The way we’re using it is “look for the ScoreBoard and the first thing you find is what we’re referring to.” Theres also a FindObjectsOfType which would return an array of matches. Now you have to be careful with this because FindObjectOfType is VERY resource intensive. But we’re okay because we’re only running it inside of Start and not inside the Update method.

**Challenge:**

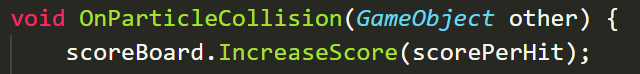
* Create a tunable variable for how much to increase score per hit
* Call our public method to increase score
* Print to the console the score each time it increases

**Solution:**

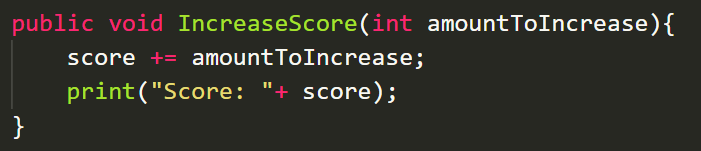
1. Create a [SerializeField] for points



1. Call the method inside of **OnParticleCollision**



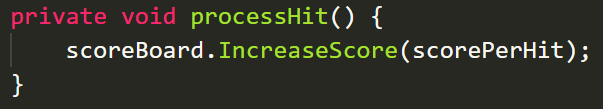
1. In **ScoreBoard.cs** add a print statement to **IncreaseScore** method



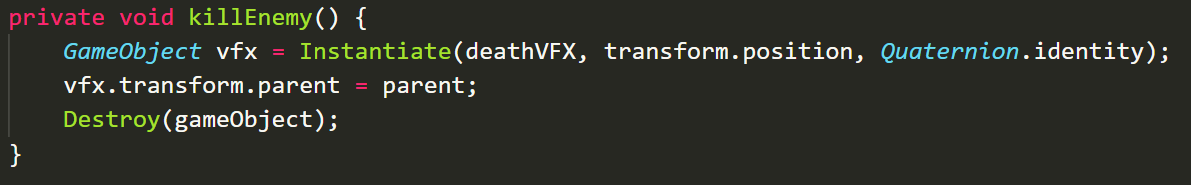
1. Go to Unity and test your changes. THE PROPER WAY TO DEBUG IS TO CHECK IF THE SCORE RESETS WHEN YOU DIE.

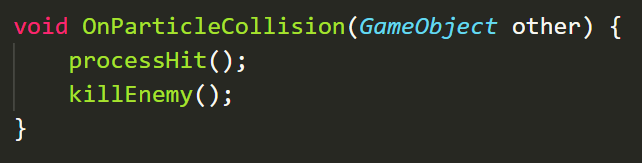
Our Enemy script’s **OnParticleCollision** method is getting clunky.

1. refactor it so that the **IncreaseScore** call happens inside a method called **processHit**



1. The rest should be placed inside of a method called **killEnemy**





FULL SCREENSHOT IN CASE YOURE CONFUSED ABOUT ORDER

