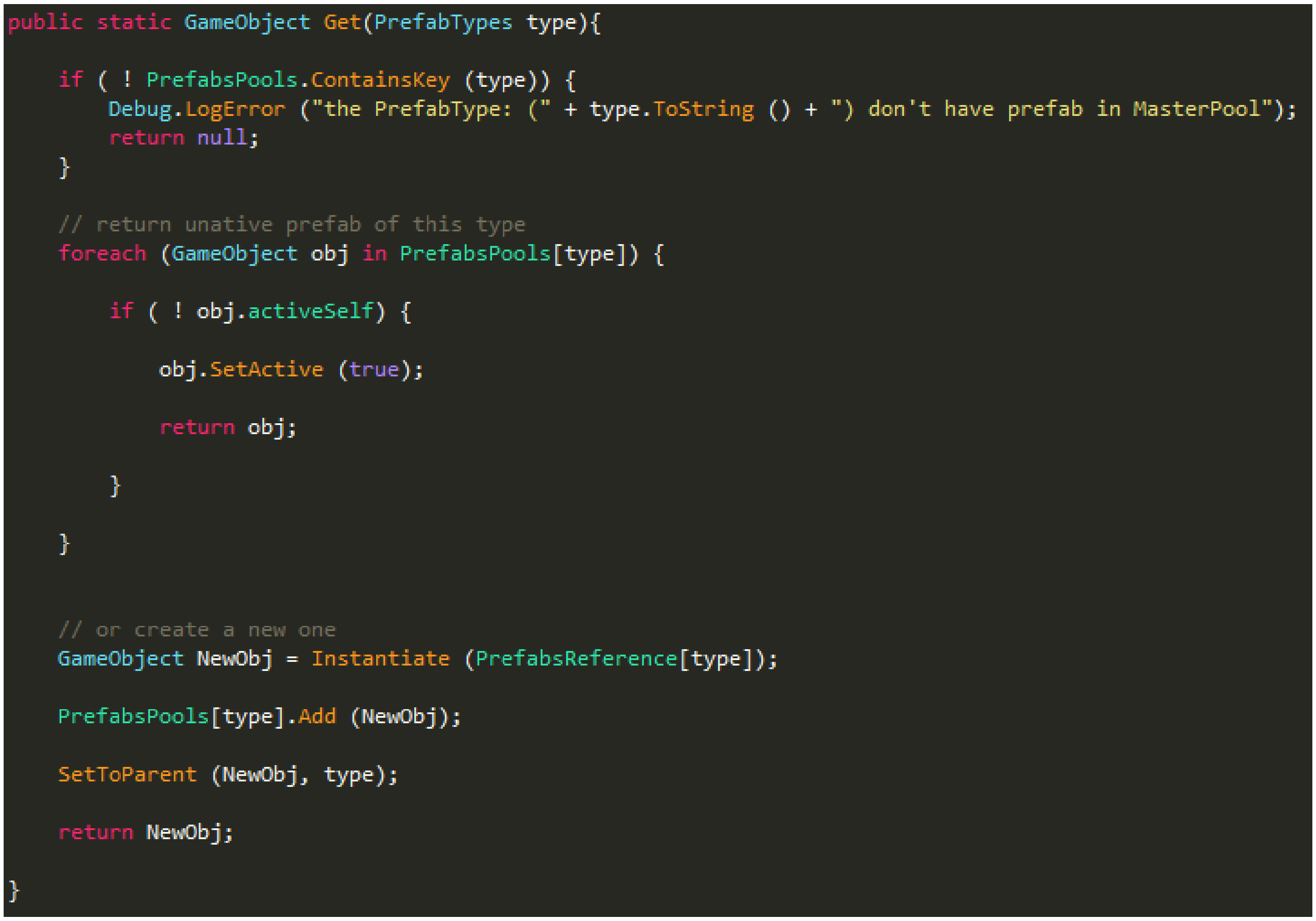
0- Objec pooling

In this two scenes in this file you'll use EventSystem in EventsClass.cs but first let's talk about Object pooling because it's used in every scene in this project

It makes scence to use one script for all our Object pooling logic to release all the other scrips from dealing with creating and reuseing Prefabs.
In order for any scripts to tell the MasterPool.cs what Prefab it needs they need a common variable to talk to each other So I created a namespace called GlobalVars to store all variable that I need all me scripts to know.
We'll use eunm because it's ease to read, write and don't need documentation.

To know when is a GameObject ready to be reused againg we'll check if it active or not, so when we're done with a GameObject we'll set it to false.

The rest is simple, ask the MasterPool for Prefab using the eunm if there's an unused Prefab of that type return it, if not create one and return it.

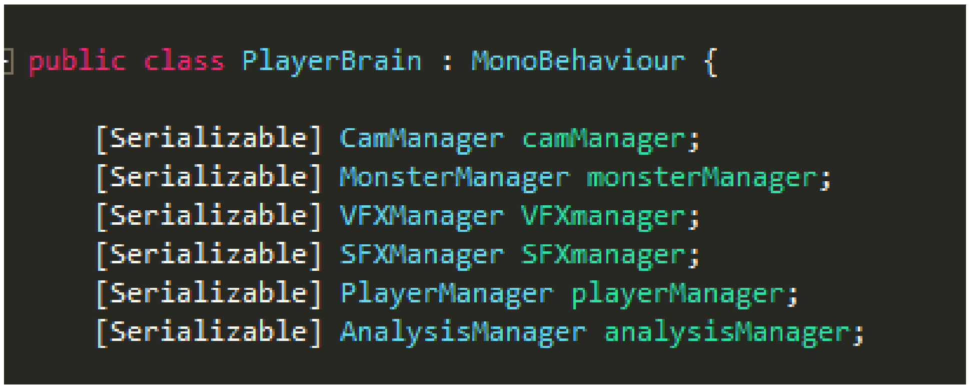


```
public static GameObject Get(PrefabTypes type){

    if ( ! PrefabsPools.ContainsKey (type)) {
        Debug.LogError ("the PrefabType: (" + type.ToString () + ") don't have prefab in MasterPool");
        return null;
    }

    // return unative prefab of this type
    foreach (GameObject obj in PrefabsPools[type]) {

        if ( ! obj.activeSelf) {

            obj.SetActive (true);

            return obj;

        }

    }


    // or create a new one
    GameObject NewObj = Instantiate (PrefabsReference[type]);

    PrefabsPools[type].Add (NewObj);

    SetToParent (NewObj, type);

    return NewObj;
}
```

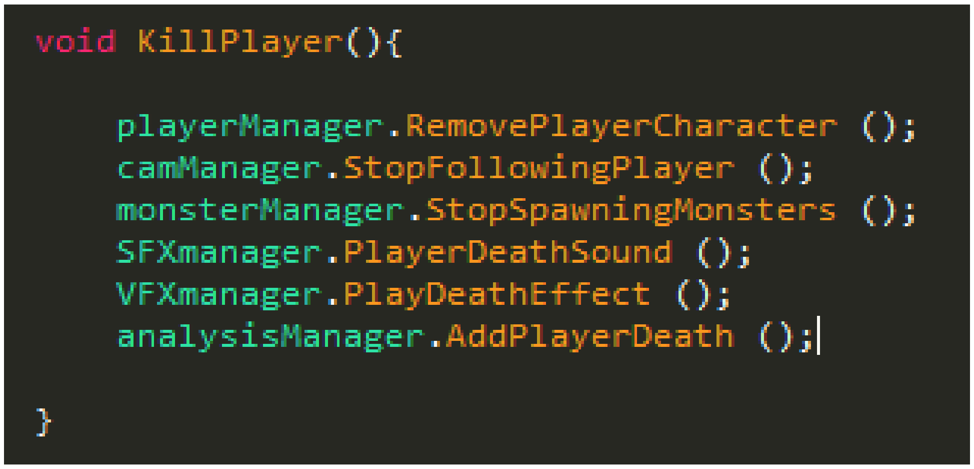
1-Event System

So now that Object Pooling is out of the way let's talk about Events system

when a simple scripts is handling the responsibility of telling a lot of other scripts what to do it's a good sign to create an event to release that script from that responsibility.
For expable let's say that you have a Player that needs to tell some managers to do something when he die.



```
public class PlayerBrain : MonoBehaviour {

    [Serializable] CamManager camManager;
    [Serializable] MonsterManager monsterManager;
    [Serializable] VFXManager VFXmanager;
    [Serializable] SFXManager SFXmanager;
    [Serializable] PlayerManager playerManager;
    [Serializable] AnalysisManager analysisManager;
```



```
void KillPlayer(){

    playerManager.RemovePlayerCharacter ();
    camManager.StopFollowingPlayer ();
    monsterManager.StopSpawningMonsters ();
    SFXmanager.PlayerDeathSound ();
    VFXmanager.PlayDeathEffect ();
    analysisManager.AddPlayerDeath ();

}
```

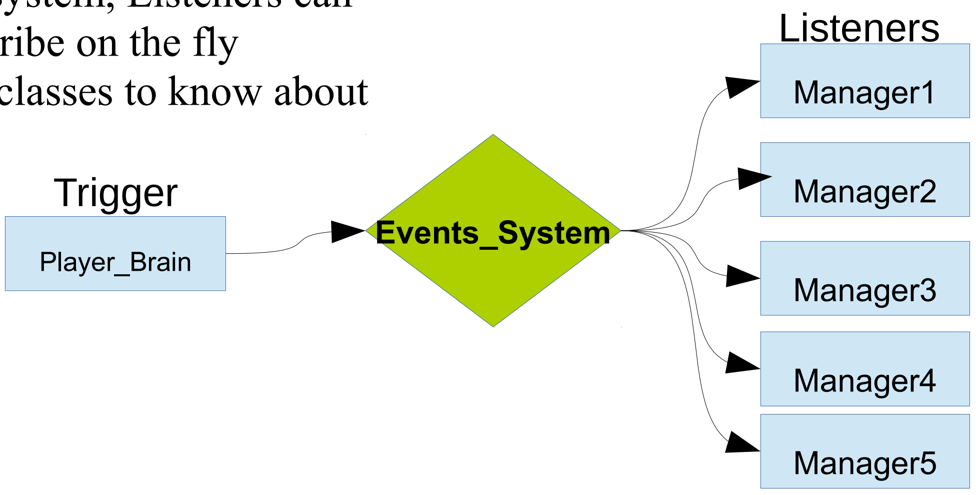
You can notice that the PlayerBrain.cs is handling a lot of responsibilities at once when the player die

This's a bad way to tell managers what to do for couple of resonse:

-Managers functions are public.

-No code dependence, meaning the PlayerBrain needs to find all managers in every scene to be able to function, and managers needs PlayerBrain to call them.

By creating an Event system, Listeners can subscribe and unsubscribe on the fly
Since we need all our classes to know about possible events it's good idea to keep all our events in GlobalVars.cs
So Triggers and Listeners can access them freely.

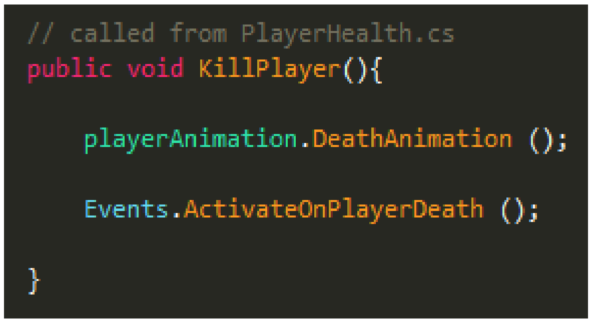


However, every design has its cons

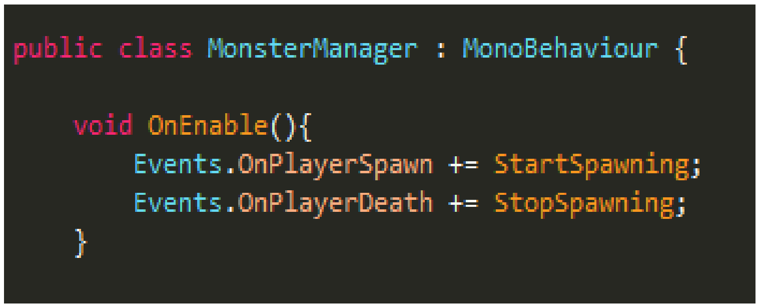
-To subscripe a function to a delegate it has to have the same parameters.

-You can only call the event form the same class where it has been created, so you'll need to use separated fucntions to call them (ActivateOnPlayerDeath)

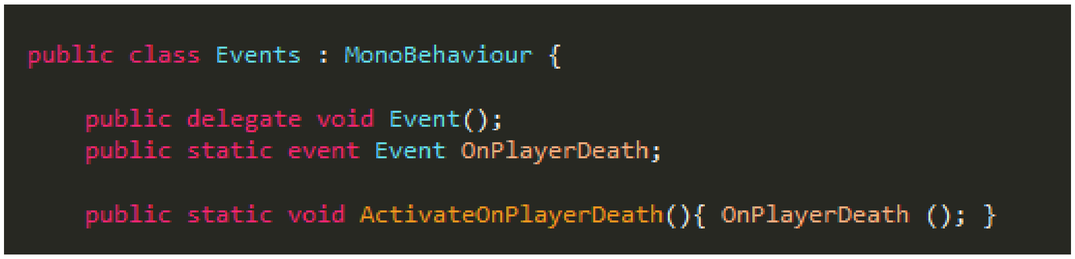
-Because we're using Statics for easy access to events we'll also have to clean the events before leaving the scene because Statics hold to their values between Scenes, to work around that I prefer creating separated event to clean from all events, and the Trigger for it will be the scrip responsible for leaving the scene.



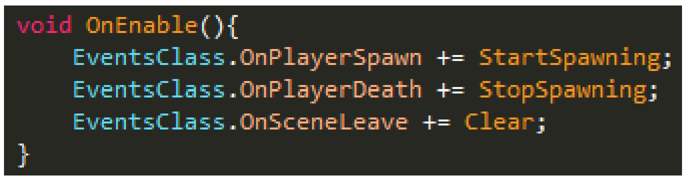
```
// called from PlayerHealth.cs
public void KillPlayer(){

    playerAnimation.DeathAnimation ();

    Events.ActivateOnPlayerDeath ();

}
```



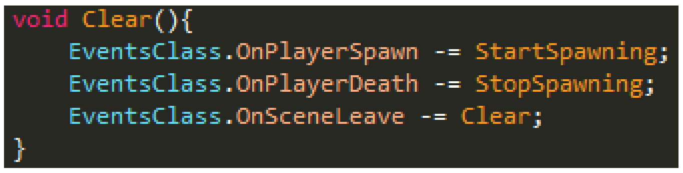
```
public class MonsterManager : MonoBehaviour {

    void OnEnable(){
        Events.OnPlayerSpawn += StartSpawning;
        Events.OnPlayerDeath += StopSpawning;
    }
```



```
public class Events : MonoBehaviour {

    public delegate void Event();
    public static event Event OnPlayerDeath;

    public static void ActivateOnPlayerDeath(){ OnPlayerDeath (); }
```



```
void OnEnable(){
    EventsClass.OnPlayerSpawn += StartSpawning;
    EventsClass.OnPlayerDeath += StopSpawning;
    EventsClass.OnSceneLeave += Clear;
}
```



```
void Clear(){
    EventsClass.OnPlayerSpawn -= StartSpawning;
    EventsClass.OnPlayerDeath -= StopSpawning;
    EventsClass.OnSceneLeave -= Clear;
}
```