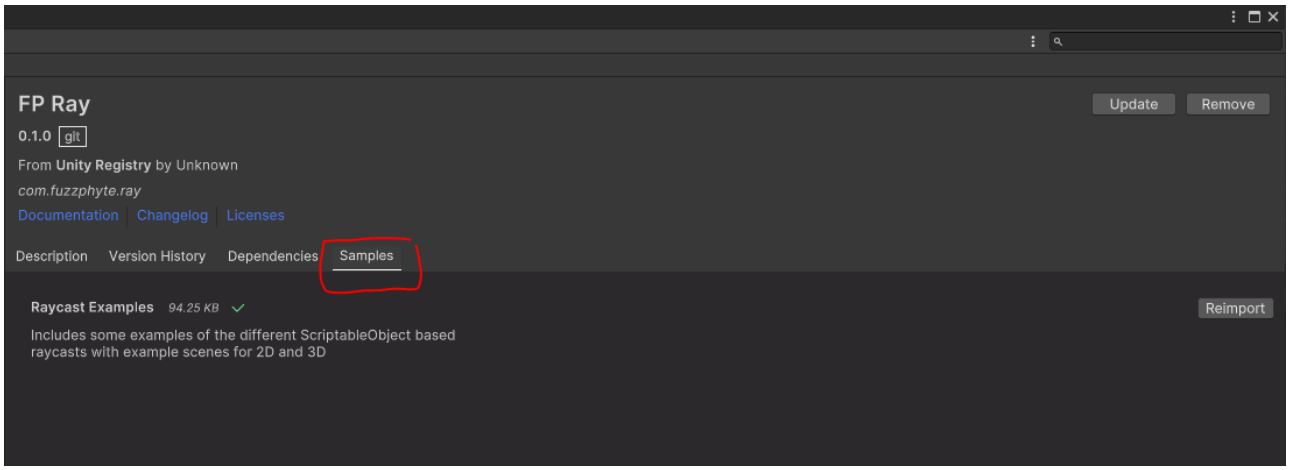
2D Raycast Readme

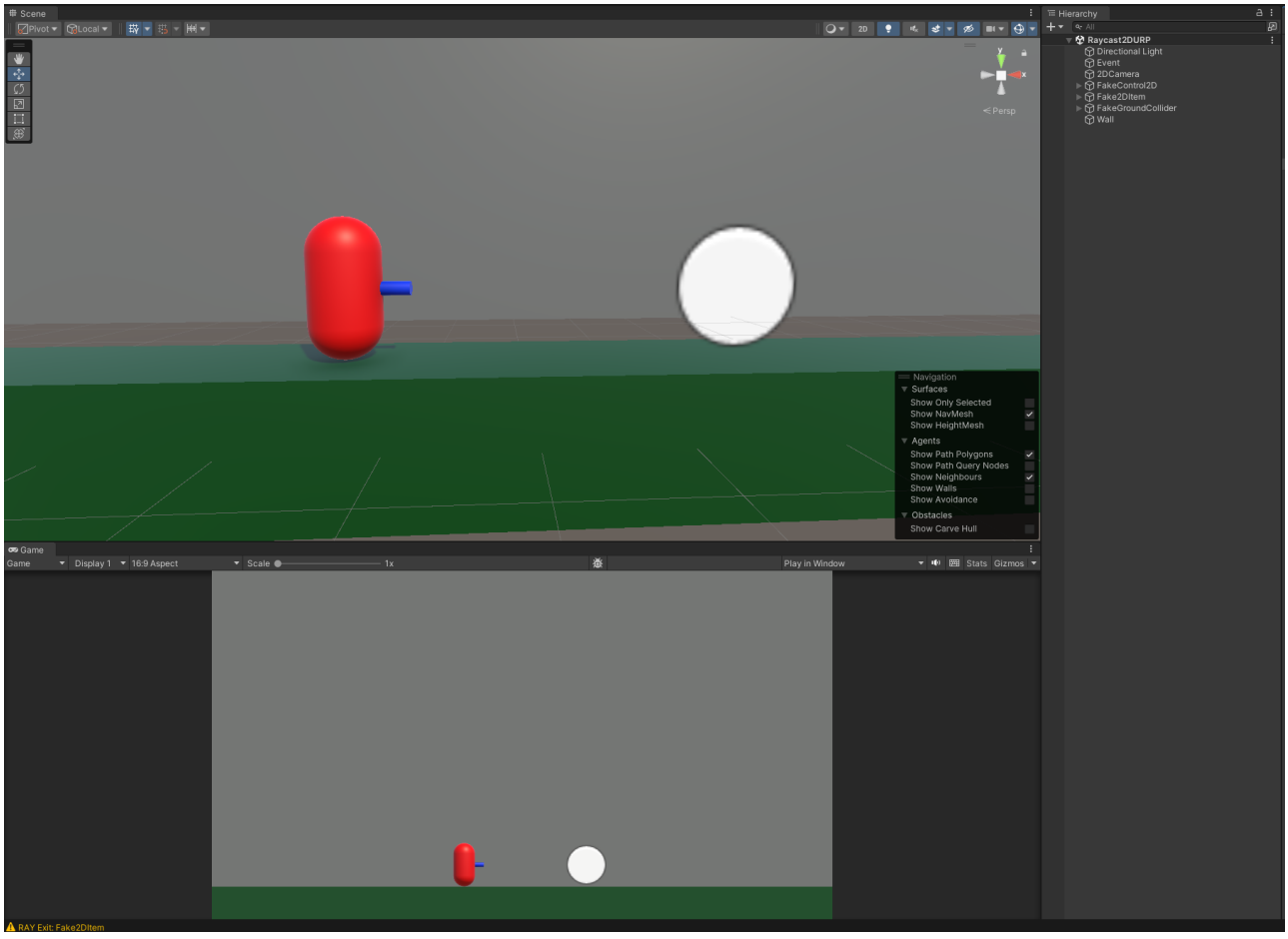
This file is to help walk you through how to utilize this system in a 2D raycasting scenario.

Setup

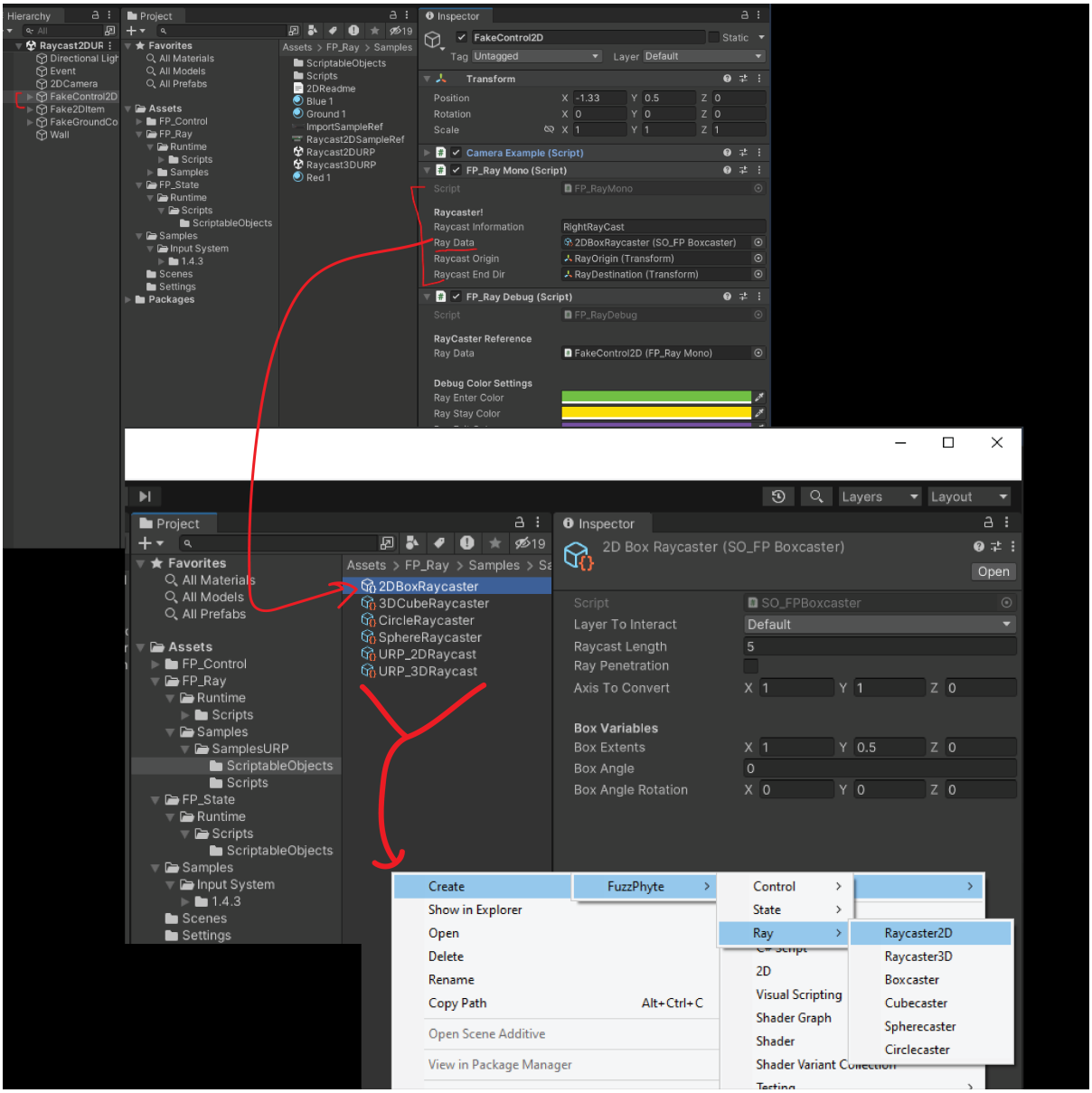
- After installing the package make sure you have installed the samples



- After the package samples have installed - open up the 'Raycast2DURP' located in the SamplesURP folder.



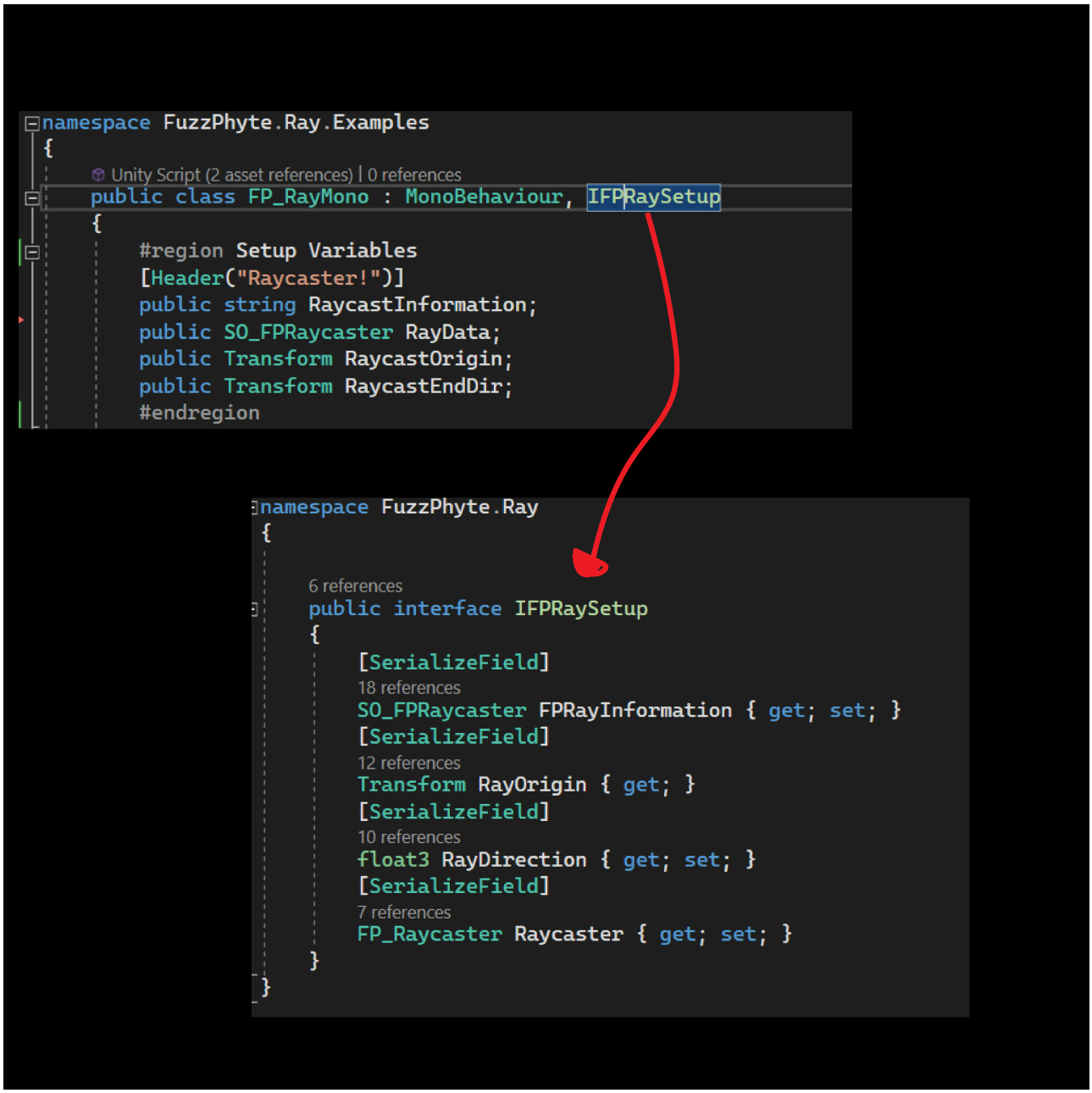
- Notice the FakeControl2D GameObject and the associated FP_Ray Mono Script
    - This script is an example Monobehaviour script on how you can utilize the FP_Ray system via invoking the IFPRaySetup interface.
    - The most important part of this script is the scriptable object class 'SO_FPRaycaster': this tells the FP_Ray system what type of raycast you want to perform.
    - Right click in your project window and look to create a new scriptable object it should be under 'create'-->ScriptableObjects-->FuzzPhyte-->Ray'



Diving into the Code

The code base is nothing more than a humble pattern utilizing an interface to decouple a C# class from being forced into a monobehaviour class and listening for messages/callbacks from the system. Open up the FP_RayMono.cs and follow along with the images below to walk through the basic use of the system.

- FP_RayMono.cs is an example you can create your own. It uses the IFPRaySetup interface which is shown below between the FP_RayMono.cs file and the IFPRaySetup.cs file.



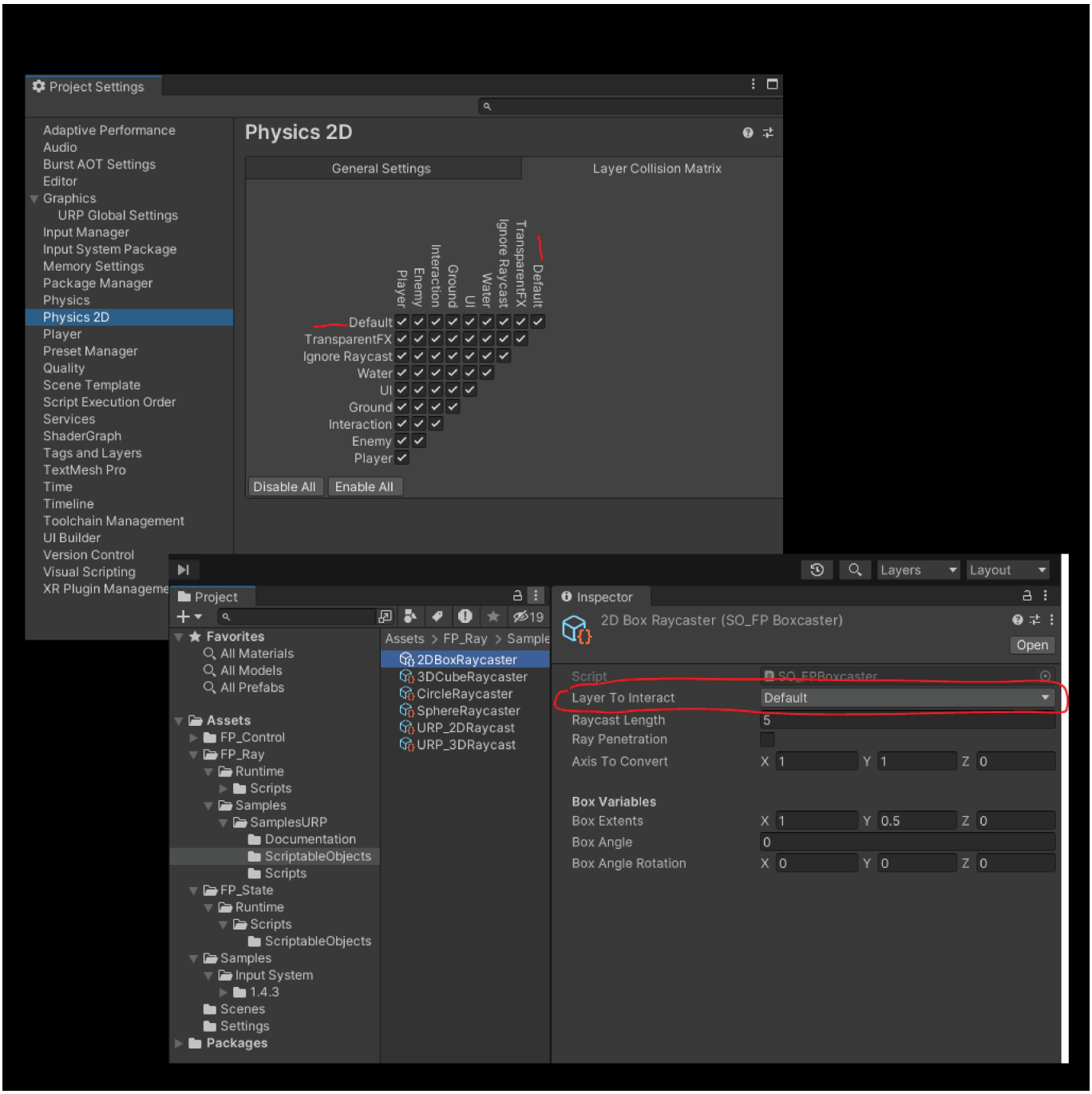
- FP_RayMono.cs needs to implement the requirements for the interface, those are in the image below which is part of the FP_RayMono.cs file

```
Unity Script (2 asset references) | 0 references
public class FP_RayMono : MonoBehaviour, IFPRaySetup
{
    Setup Variables
    #region Interface Requirements
    18 references
    public SO_FPRaycaster FPRayInformation
    {
        get { return RayData; }
        set { RayData = value; }
    }
    12 references
    public Transform RayOrigin {
        get { return RaycastOrigin; }
        set { RaycastOrigin = value; }
    }
    10 references
    public float3 RayDirection
    {
        get { return Vector3.Normalize(RaycastEndDir.position - RaycastOrigin.position); }
        set { RayDirection = value; }
    }
    private FP_Raycaster _raycaster;

    private FP_RayArgumentHit _rayHit;

    7 references
    public FP_Raycaster Raycaster { get { return _raycaster; } set { _raycaster = value; } }

    1 reference
    public void SetupRaycaster()
    {
        _raycaster = new FP_Raycaster(this);
    }
    #endregion
    Unity Message | 0 references
    private void Awake()
    {
        SetupRaycaster();
    }
```

- Once you have implemented the interface, your MonoBehaviour is now ready to talk with the FP_Ray system - but in order to hear back from the system you need to add the correct listeners to your class. Those are outlined below.

```
Unity Script (2 asset references) | 0 references
public class FP_RayMono : MonoBehaviour, IFPRaySetup
{
    Setup Variables
    Interface Requirements
    Unity Message | 0 references
    private void Awake()...
    Unity Message | 0 references
    public void OnEnable()
    {
        _raycaster.OnFPRayFireHit += OnRayStay;
        _raycaster.OnFPRayEnterHit += OnRayEnter;
        _raycaster.OnFPRayExit += OnRayExit;
        _raycaster.ActivateRaycaster();
    }
    Unity Message | 0 references
    public void OnDisable()
    {
        _raycaster.OnFPRayFireHit -= OnRayStay;
        _raycaster.OnFPRayEnterHit -= OnRayEnter;
        _raycaster.OnFPRayExit -= OnRayExit;
        _raycaster.DeactivateRaycaster();
    }
    #region Callback Functions for Raycast Delegates
    2 references
    public void OnRayEnter(object sender, FP_RayArgumentHit arg)
    {
        Debug.LogWarning($"RAY Enter: {arg.HitObject.name}");
        _rayHit = arg;
    }
    2 references
    public void OnRayStay(object sender, FP_RayArgumentHit arg)
    {
        Debug.LogWarning($"RAY Stay: {arg.HitObject.name}");
        _rayHit = arg;
    }
    2 references
    public void OnRayExit(object sender, FP_RayArgumentHit arg)
    {
        Debug.LogWarning($"RAY Exit: {arg.HitObject.name}");
        _rayHit = arg;
    }
```

At this point as long as your Scriptable Object (data) is correctly setup and your Physics2D/3D Matrix allow you to have the correct collisions the system will report back and activate to these callbacks when those messages fire: on enter, on exit, on stay.



Feel free to copy and use the FP_RayMono.cs as an example - just be aware that this script is always going to be subject to change into future versions.