

PHOTON NETWORKING

1)Multiplayer oyunlarda server olduğu için biz oyuncular birbirlerinin hareketlerini ve yaptıkları işlemleri görebilmeleri için bu işlemleri server tarafında oluşturmamız gerekiyor.

2)Biz server tarafında herhangi bir obje oluşturmak itiyorsak (Character, Particle effect vs.) server tarafında oluşturmamız gerekiyor.

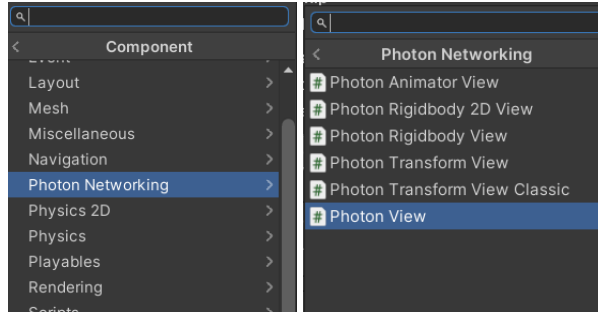
Örneğin ateş esnasında iki oyuncu birbirlerini görebilmeleri için efektleri vs. bu işlemleri server tarafında gerçekleştirmemiz gerekiyor.

3)Yukarıdaki işlemleri yapabilmemiz için,

!Photon sistemi Assets içerisinde Resources (isim çok önemli) klasörü oluşturup, serverda oluşacak objeleri prefab olarak tutmamız gerekiyor. (Eğer klasör ismini yanlış yazarsanız oyun içerisinde herhangi bir obje oluşturma işlemi gerçekleştirmeyecektir.)

4)Bir obje server taraflı obje nasıl oluşturulur.

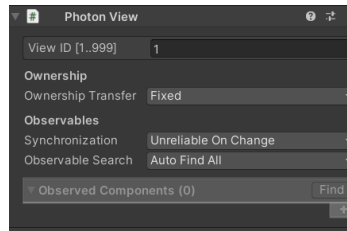
-Örneğin characterimize gidiyoruz --> AddComponent --> Photon Networking --> Photon View komponentini eklediğimiz zaman, artık o objemiz photon için server objesi anlamına geliyor.



PHOTON NETWORKING COMPONENTLER

-Photon View: Server objesi olduğunu belirtmek için Photon View komponentini eklememiz gerekiyor.

Bu komponent bize birçok işlemi yapmamıza olanak tanıyor. Örneğin oyuncu ile ilgili bilgileri alabilmemizi, ve onları kontrol etmemize olanak tanıyor.



View ID: Sistemin çalışma esnasında kullanıcıların bilgilerini otomatik olarak ürettiği bölümdür.

Ownership Transfer: Varsayılan olarak ağa bağlanan nesnelerin sabit bir sahipliği vardır. Yani bazı oyunlarda odanın kurucusu çok önemli olabilir.

Fixed: Sahiplik yok. Her oyuncunun eşit olduğu oyun modeli.

Request: Oyunda bir sahiplik var ve odanın sahibi odayı ilk kuran kişidir. Odayı kuran kişiye bir soru geliyor sahipliği örneğin 5. Kullanıcıya devretmek istiyor musun? Eğer kabul ederse 5. Kullanıcıya sahiplik geçiyor. Bunlara ait özel fonksiyonlar mevcut.

Eğer oyundan direkt çıkarsa oyuna ikinci gelen kullanıcı sistem tarafından sahip olur.

Takeover: Hiçbir talep göndermeden direkt sahipliği istediğimiz oyuncudan alıp, istediğimiz oyuncuya verebiliriz.

Synchorization: Senkronizasyonun nasıl olabileceğini burada belirtiyoruz. Bilgilerin akış şeması mevcut biz onu burada belirtebiliyoruz.

Off: Hiçbir bilgi aktarılamayacak.

Reliable Data Compressed: Verilerin değişmemesi durumunda bile verilerin alınması garanti edilir. En garanti yöntemlerden biridir.

Unreliable: Veriler sıra ile alınır, ancak bazı güncellemeler kaybolabilir. Veriler aktarılırken çok ince bilgiler gelmiyorsa, kullanılabilir.

Unreliable on change: Veriler sıra ile alınıyor ancak güncellemelerde son bilgiler tekrar edilirse bir sonraki değişikliğe kadar güncellemeyi durduracaktır. Performans sağlar.

Observable Serach:

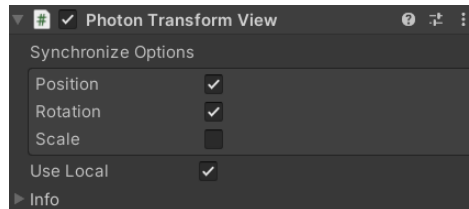
Manual: Eklemiş olduğumuz komponentleri tek tek eklememiz gerekir.

Auto find active: Otomatik olarak sadece aktif olanları ekler.

Auto find all: Eklemiş olduğum komponentleri otomatik olarak ekler.



-Photon Transform View: Transform ile alakalı servera ne gönderilmesi gerekiyorsa, burada belirtiyoruz.

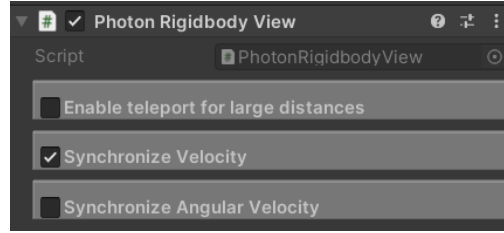


Position: Pozisyon hareketi var mı?

Rotation: Rotasyon hareketi var mı?

Scale: Scale hareketi var mı?

-Photon Rigidbody View: Fiziksel zıplama, patlama vs. ile alakalı servera ne gönderilmesi gerekiyorsa, burada belirtiyoruz.

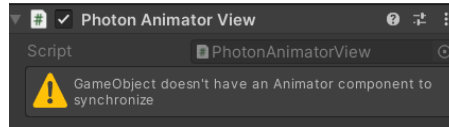


Enable teleport for large distances: Büyük mesafeler için kuvveti etkinleştirmeye yarıyor. Büyük bir boyutlu fiziksel mesafelere ihtiyacın var ise, dataları genişletir ve kullanır.

Synchronize Velocity: Ateş etme, ileri gitme, geri gitme vs. gibi küçük etkileşimler var ise senkronize ederek kullanmamızı sağlar.

Synchronize Angular Velocity: Açısal hız yani çapraz 5 derece vs varsa, bu seçimi de seçebiliriz.

-Photon Animator View: Eğer bir animasyon ile alakalı servera ne gönderilmesi gerekiyorsa, burada belirtiyoruz.



5) Oyuncu odaya girdikten sonra nasıl oluşturulur?

```
public override void OnJoinedRoom()
{
    /*Oyuncu ismindeki resources klasörü içerisindeki prefab dosyasını oluşturuyoruz.*/
    PhotonNetwork.Instantiate("Oyuncu", Vector3.zero, Quaternion.identity);
    Debug.Log("Odaya Girildi.");
}
```

6) IsMine Komutu:

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;

public class Oyuncu : MonoBehaviour
{
    public GameObject partefek;
    PhotonView pv;

    void Start()
    {
        pv = GetComponent<PhotonView>();
    }

    void Update()
    {
        /*Ortak script dosyasını kullanan objeler için is mine komutunu kullanarak, her obje kendi içerisindeki script dosyasını kullanmasını istediğimizi söylüyoruz.*/
        if (pv.IsMine)
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit))
            {
                Vector3 dir = hit.point - transform.position;
                dir.y = 0;
                transform.rotation = Quaternion.LookRotation(dir * Time.deltaTime * 2f);
                Debug.DrawLine(transform.position, hit.point);
            }
        }
    }
}
```

7)Eftikleri oluşturma

```
if (Input.GetButtonDown("Fire1"))
{
    Ray ray2 = Camera.main.ScreenPointToRay(Input.mousePosition);
    if (Physics.Raycast(ray2, out hit))
    {
        Debug.Log(hit.transform.gameObject.tag);

        /*Artık effects de server tarafı iletilmesi için dışarıdan dosyaya alması gerekir. PhotonNetwork'e belirtmek effect isimlerimizi gerçekleştiriyoruz.*/
        PhotonNetwork.Instantiate("AteşParç", transform.position, Quaternion.Euler(-90f, transform.eulerAngles.y, 0f));
    }
}
```

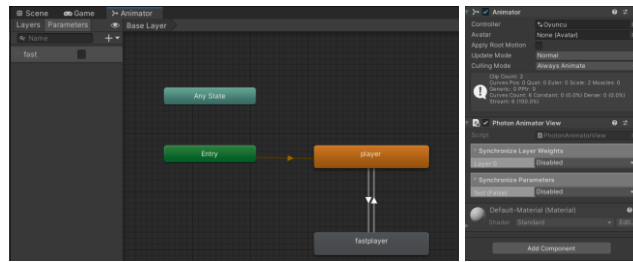
8)Oluşturulan bir objeyi server taraflı yok etme. (Particle Effect)

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using Photon.Pun;
5
6 public class parc : MonoBehaviour
7 {
8     // Start is called before the first frame update
9     void Start()
10     {
11         StartCoroutine(DestroyObject());
12     }
13
14     IEnumerator DestroyObject()
15     {
16         yield return new WaitForSeconds(2);
17         PhotonNetwork.Destroy(gameObject);
18     }
19 }
```

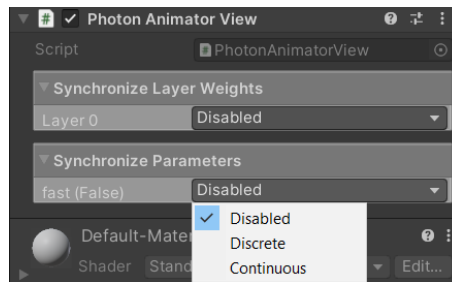
9)Animasyonları server taraflı obje haline getirme

-Eğer static bir animasyonun varsa yani sabit controller tarafından herhangi bir değer gönderilmeyen animasyonun varsa Photon bunu algılar ve herhangi bir şey eklemeye ihtiyaç duyulmaz.

-Eğer animasyonumuz static değil ise, yani parametre göndererek controller yönetiyorsak, örneğin yürüyen bir animasyonumuzu E ye basarak hızını arttırıp koşma animasyonuna geçerse eğer photon'a bunu belirtmemiz gerekiyor.



Senkronizasyonu nasıl gönderdiğimizi de belirtiyoruz.



Disabled: Disabled olarak bırakırsak server taraflı animasyon gösterimlerinde hata alabiliriz.

Discrete: Ara ara bu parametrenin gidişine bakmak.

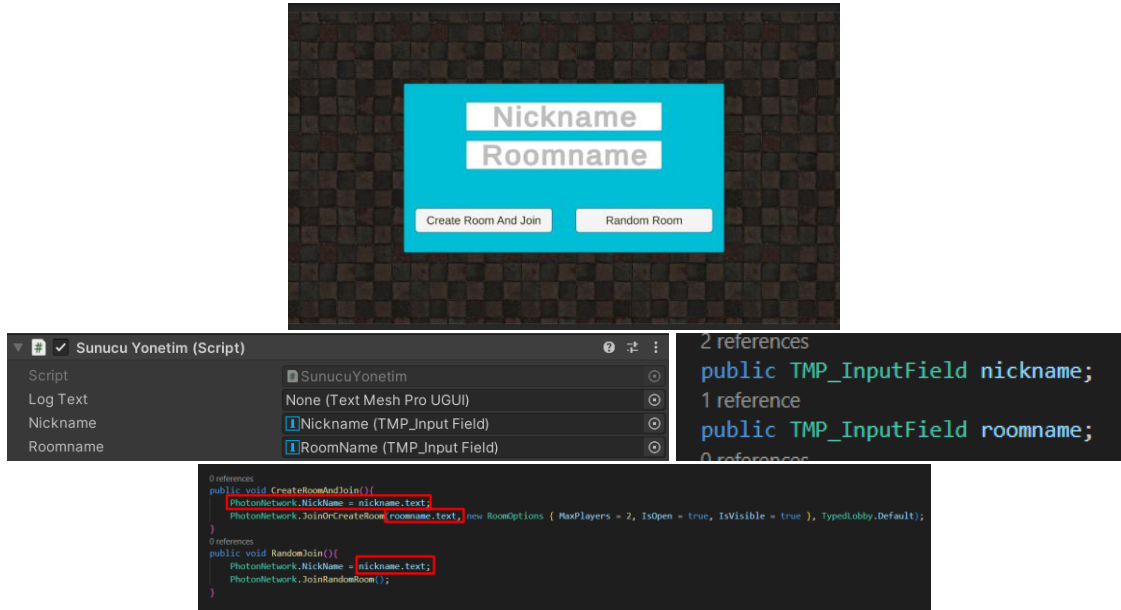
Continuous: Sık sık bu parametreleri kontrol etmesini istiyoruz.

```
PhotonNetwork.Instantiate("AtesParc",transform.position,Quaternion.Euler(-90f, transform.eulerAngles.x, transform.eulerAngles.y));

/*Animasyonları server'a belirtmede kod tarafında sadece is mine içerisinde yazmamız yeterli.
GetKey: Basılı kaldığı sürece*/
if(Input.GetKey(KeyCode.Q)){
    animator.SetBool("fast",true);
}
else{
    animator.SetBool("fast",false);
}
```

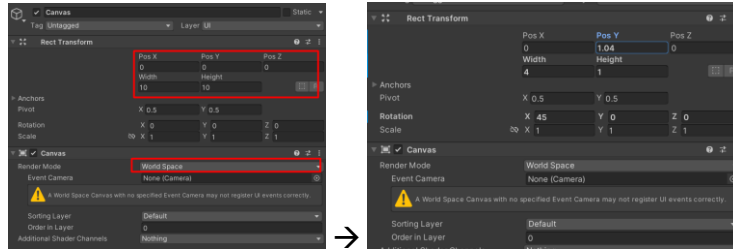
SIMPLE PROJECT

10) Oda kurma ve Ad belirleme paneli

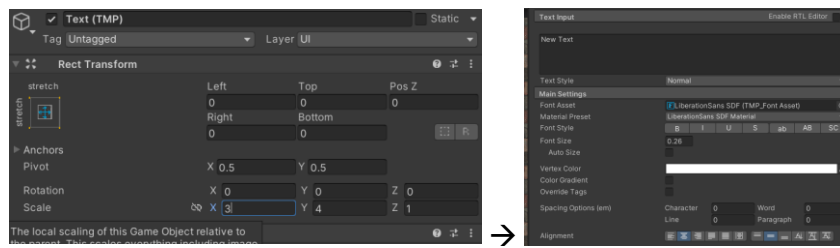


11) Oyuncunun ismini ve healt barını hazırlama

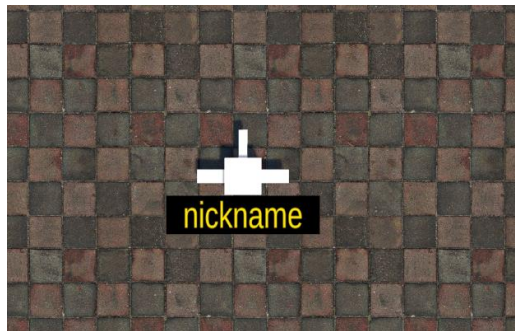
Canvas hazırlama:



TMP_Text hazırlama:



İlk Hali:

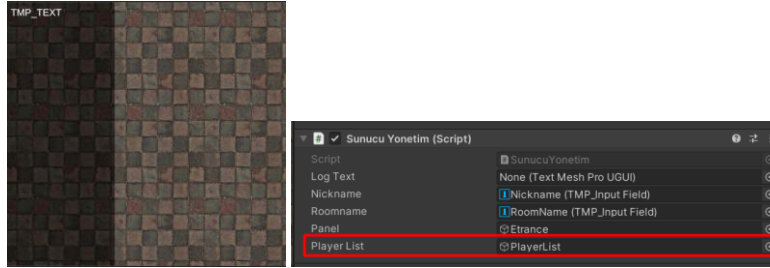


Oda kurarken girdiğimiz ismi oyuncunun scripti içerisinde alma:

```
4 using Photon.Pun;
5 using TMPro;
6 0 references
7 public class oyuncu : MonoBehaviour
8 {
9     3 references
10    PhotonView pw;
11    3 references
12    Animator animator;
13    1 reference
14    [SerializeField] public TextMeshProUGUI nickname;
15    0 references
16    void Start()
17    {
18        pw = GetComponent<PhotonView>();
19        animator = GetComponent<Animator>();
20        nickname.text = pw.Owner.NickName;
21    }
22    0 references
23    void Update()
```



12) Odada bulunan oyuncuların tam listesini alma işlemi



!Burası sürekli tab basılıyken çalışıyor, bir bool çalıştı çalışmadı komutu ekle



13) Oyuncunun kurucu olup olmadığını anlama



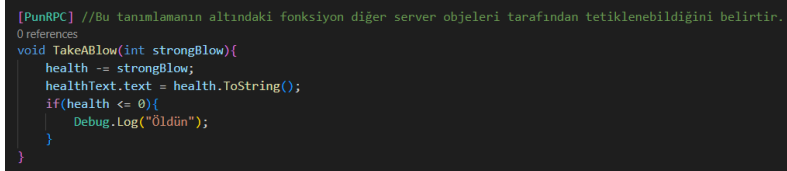
14) Oyuncuların ateş etme sistemi, oyuncuların can kaybetmesi ve ölmesi



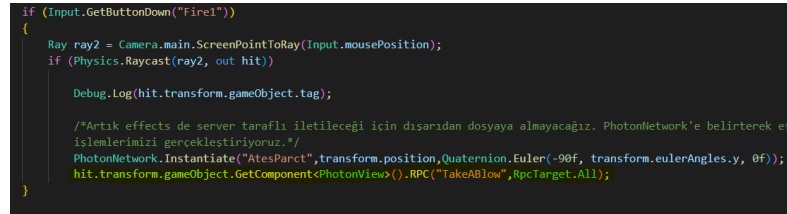
Objelerin birbirleri ile olan etkileşimde birbirlerini tetikleyebilecek bir fonksiyon olması gerekmekte. Örneğin x character ateş ettiğinde y character'e çarptığında y character'in canının azalması gerekmekte.

Server tarafında oluşturulan objelerin birbirleri ile etkileşime girebilmeleri ve birbirlerine sorgu gönderebilmeleri için uygulamamız gereken bir yöntem bulunmakta.

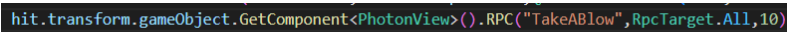
Bu yöntem ilgili objenin script dosyasında olduğu müttetçe sahnede bulunan tüm objeler orayı tetikleyebilmekteler.



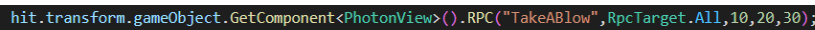
Artık ateş etme içerisindeki bu fonksiyonu çarpan objede tetikliyoruz. Eğer yukarıdaki fonksiyon bir parametre almasaydı sadece TakeABlow() fonksiyonu olsaydı aşağıdaki fonksiyonu yazmamız yeterli.



Eğer parametre de gönderiyorsak,



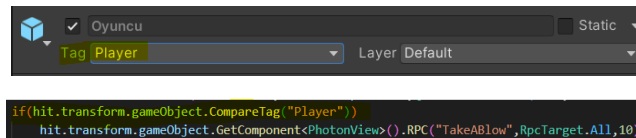
Eğer birden fazla parametre gönderseydik, !parametreleri sırasıyla...



Ateş ettiğimiz de aşağıdaki hatayı alıyoruz.



Bu hatayı düzeltmek için;



Character öldüğünde yok etmek,

```
[PunRPC] //Bu tanımlamanın altındaki fonksiyon diğer server objeleri tarafından tetiklenebildiğini belirtir.  
0 references  
void TakeABlow(int strongBlow){  
    health -= strongBlow;  
    healthText.text = health.ToString();  
    if(health <= 0){  
        Debug.Log("Öldün");  
        PhotonNetwork.Destroy(gameObject);  
    }  
}
```

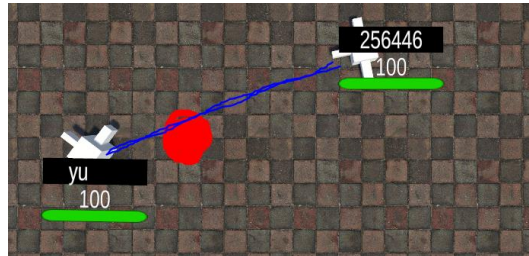
15)Oyuncuya can barı yapımı



16)Biz burada mouse pozisyonuna ışın gönderiyoruz.

Aşağıdaki görselde mouse kırmızı noktada ama ateş ettiğimizde particle effect oyuncuya çarpıyor, ancak canı düşmüyor.

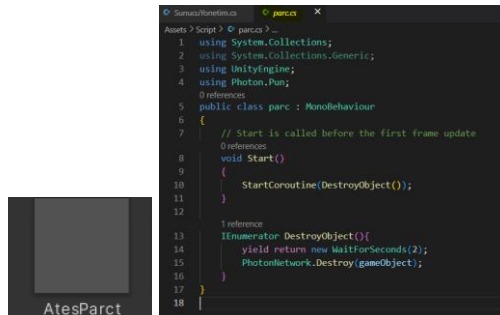
Bunun sebebi ise şu bir Raycast ışınımızı mouse göre gönderiyoruz. Yani mouse pozisyonuna gönderiyoruz. Bu nedenle oyuncu üzerinde değil ise mouse ışın gitmeyecektir.



1.Çözüm ise;

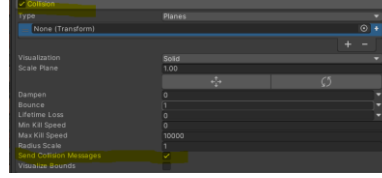
Gönderilen particle effectin çarptığında çarpılan objenin RPG methoduna ulaşmayı ve canını düşürmeyi göreceğiz. Böylece mouse konumu önemini kaybediyor ve mouse pozisyonunda çalışan effect önem kazanıyor.

1-Ateş ettiğimizde çalışan particle effectimizin içerisindeki C# dosyamızı açıyoruz.

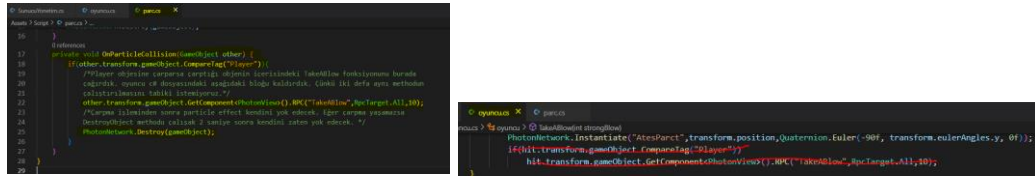


2-Particle effectlerde biz çarpışmaları nasıl yakalayabiliriz.

Particle effects içerisindeki collision parametresini açıyoruz. Ve çarpışmaları alabilmek için ise Send Collision Messages seçeneğini tikliyoruz. Artık script tarafında çarpışmalar sonucunda cevap gönderebilecek.



Script tarafında ise,

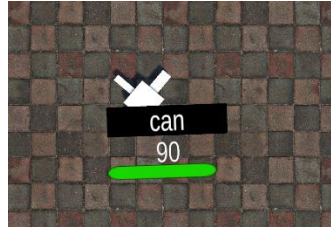


!Collision parametresi içerisindeki çarpışma tipini world yapmayı unutmayalım.



3-Şu an ise bir sorununuz mevcut. Sahnede hiçbir oyuncu yokken bile ateş ettiğimizde hem particle effect çalışmıyor hem de kendi canımız otomatik olarak düşüyor.

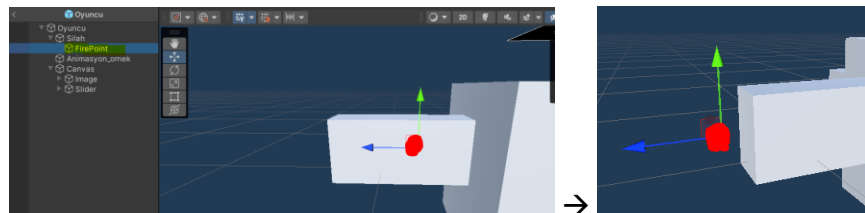
!Bunun sebebi şu, particle effect oyuncunun pozisyonunda oluşturuyoruz. Bundan dolayı particle effect ileriye gitmeden kendi oyuncusuna çarptığı için, kendi canını azaltıyor ve çarptığında particle effect yok ettiğimiz için ise, particle effecti görmüyoruz.



Şimdi ise bu sorunu çözelim,

1..Oyuncumuza gidiyoruz ve silah objemizin içerisine bir adet firepoint oluşturuyoruz.

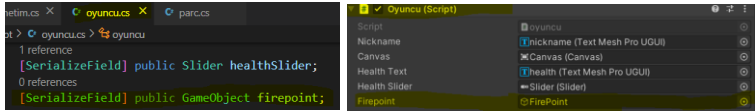
Sonrasında bu obje yine player tagından etkilenmemesi için biraz öne oluyoruz.



2..Particle effec artık oyuncunun transform pozisyonundan üretilmeyecek verdiğimiz firepoint'den üretilecek.

```
işlemlerimizi gerçekleştiriyoruz."/
PhotonNetwork.Instantiate("AtesParct", transform.position, Quaternion.Euler(-90f, transform.eulerAngles.y, 0f));
```

Firepoint objesi pozisyonunda efektimizi oluşturma



```
PhotonNetwork.Instantiate("AtesParct", firepoint.transform.position, Quaternion.Euler(-90f, transform.eulerAngles.y, 0f));
```

Artık sorunsuz çalışacaktır.

17) Her oyuncuya kamera ekleme işlemi

1. Her silahın farklı darbe gücü olabilir. Şimdi biz darbe işlemini particle effecte attığımız için, darbe gücümüzü değiştiremiyoruz. Peki değiştirmek istersek nasıl yapabiliriz:

Effect oluşturduğumuzda bir game objeye atıp effect içerisindeki değişkene ulaşip bu değişkenin değerini değiştirip, ateş gücünü ayarlayabiliriz.

```
if (Input.GetButtonDown("Fire1"))
{
    Ray ray2 = Camera.main.ScreenPointToRay(Input.mousePosition);
    if (Physics.Raycast(ray2, out hit))
    {
        Debug.Log(hit.transform.gameObject.tag);

        //Artık effects de server tarafı iletilmesi için dışarıdan donaya alacağız. PhotonNetwork'a belirterek effect
        //işlemlerimizi gerçekleştiriyoruz.*/
        GameObject effect = PhotonNetwork.Instantiate("AtesParct", firepoint.transform.position, Quaternion.Euler(-90f, transform.eulerAngles.y, 0f));
        effect.GetComponent<AteşGücü>().atesGücü = 20;
    }
}
```

2.Sıradaki işlemimiz ise,

SERVER MODLAR

RpcTarget.All	→ O an odada bulunan tüm oyunlara değişikliği yansıtır ve yapılan işlemin sonucunu gösterir. RPC işlemini RPC'yi gönderen oyuncuda hemen işler diğer oyunculara sıra ile işler
RpcTarget.AllBuffered	→ O an odada bulunan tüm oyunlara değişikliği yansıtır ve odaya sonradan giren oyunculara da geçmişte yapılan işlemlerin güncel halini gösterir
RpcTarget.AllViaServer	→ All yönteminin aksine, bu seçenek seçilirse RPC işlemleri tüm oyunculara aynı sıra ile aktarılır. Gönderende herhangi bir öncelik söz konusu olmaz
RpcTarget.AllBufferedViaServer	→ Buffered ile viaServer yöntemlerinin birleştirilmiş halidir.
RpcTarget.MasterClient	→ Sadece oda kurucusu olan oyununcun RPC fonksiyonunu çalıştırır
RpcTarget.Others	→ RPC'yi gönderen oyuncu haricindeki tüm oyuncuların RPC fonksiyonlarını çalıştırır.
RpcTarget.OthersBuffered	→ Buffered yönteminin kullanıldığı şeklidir. Odaya sonradan dahil olan RPC gönderen oyuncu haricindeki tüm oyunculara geçmiş RPC belleğini gösterir.

-Örnekler;

--**RpgTarget.All**: Sunucu tarafında yapılan işlemlerde bir sıra mevcuttur. Taleplerin gönderilme sırası örneğin, 5 adet talep vardır. Örneğin server bu talepleri sıraya sokar ve sıra ile bu işlemleri gerçekleştirir.

Bir oyuncu ateş ettiğini düşünelim, oyuncu RPC talebi gönderiyor. O oyuncuda o değişiklik hemen yansırken diğer oyuncuların RPC sıra ile gerçekleştirir.

--**RPCTarget.AllBuffered**: Örneğin, odamız 3 kişilik ve 2 oyuncu odaya giriş yaptı. Birbirleri ile etkileşime girdiler, işte savaştılar puan kazandılar vs. sonradan gelen 3. oyuncu da odaya girdiğinde kendinden önceki çağırılan tüm RPC işlemlerini güncel haliyle görmesini sağlar.

AllBuffered komutu bir ön bellek sistemidir. Oyun başladığında kayıtlar başlar ve yapılan tüm işlemler ön belleğe alınır ve daha sonradan gelen oyuncu ön bellekteki bu verileri alır ve güncel olarak oyuna dahil olur.

Bunun şöyle bir handikapı mevcut, RPC methodlarını ön belleğe alacağı için çok fazla RPC method kullanılır ise ön bellekte fazla yer kaplayacaktır. Haliyle sistem bir yerden sonra kasabilir ya da ağırlaşabilir.

Çok fazla kullanılan bir yöntem olmasa da oyun ihtiyacına göre kullanılabilir.

--**RPCTarget.AllViaServer**:

Gönderilen RPC herhangi bir öncelik sağlanmadan her oyuncuya aynı anda yansır.

--**RPCTargetAllBufferedViaServer**:

RPCTarget.AllBuffered ve RPCTarget.AllViaServer modlarının birleştirilmiş halidir.

--**RPCTarget.MasterClient**:

Sadece oda kurucusu olan kişinin RPC fonksiyonunu çağırmak istenildiğinde bu mod kullanılır.

--**RPCTarget.Others**:

Ben bir kullanıcıyım oyunda yaptığım herhangi bir işlem 3 oyuncununda canını azaltacağını düşünelim. Örneğin oyunda bir bomba attım ve bombanın menzili 4 oyuncuyu etkilediğini düşünelim o zaman aynı anda farklı oyuncuların da RPC fonksiyonunu çalıştırmak için Others methodunu kullanırız.

--**RPCTarget.OthersBufferd**:

Buffered merhodunu others içerisine dahil edilmiş halidir.

```
other.transform.gameObject.GetComponent<PhotonView>().RPC("TakeABlow",RpcTarget.Others,10);
other.transform.gameObject.GetComponent<PhotonView>().RPC(["TakeABlow",RpcTarget.All],10);
```

3.RPC ile ilgili kullanılabilecek bir diğer yöntem.

RPC methodunda önemli verilerin mevcut ise şifreleyerek gönderebilirsin. RpcSecure dedikten sonra true derse önemli veriler servera şifreleyerek gidecektir.

```
other.transform.gameObject.GetComponent<PhotonView>().RpcSecure("TakeABlow",RpcTarget.All,true,10);
```

4.Şimdi asıl konumuza geri dönersek, her oyuncunun kendi kamerası varsa biz her kamerayı oyunculara özel nasıl gerçekleştirebiliriz.

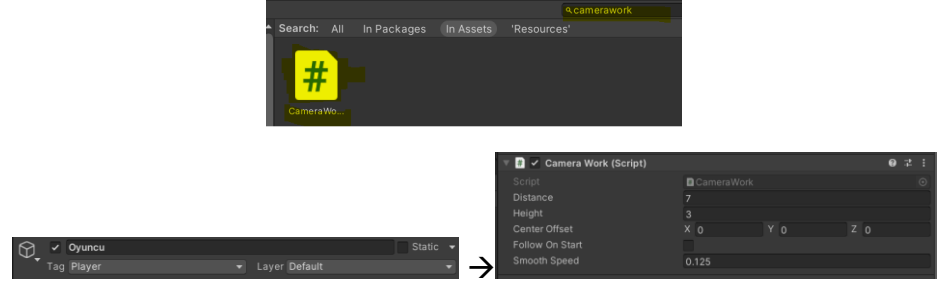
Biz oyunumuzdaki ekrandaki kameraya şunu dememiz gerekiyor. Senin local oyuncun kim ise onu takip edeceğini söylüyeceğiz.

Bu işlem kameran ne olursa olsun hiçbir sorun yaratmayacaktır.

1-Bunun için,

1..Photon'un bize sunduğu CameraWork scripti mevcut o script dosyasını oyuncumuzun içerisine atıyoruz.

Bu script dosyasında takip mesafesi yüksekliği vs. gibi çeşitli ayarları oyuna uygun yaparsın.



2..Sıradaki işlem ise photonun bize sunmuş olduğu Camerawork scriptin çalışması için oyuncunun içerisine bir kod bloğu eklememiz gerekiyor. Aşağıdaki txt dosyasının içerisindeki kamera takip methodunu alıp, kullanabilirsin.



kameratkip.txt

1...İlk olarak gerekli using parametresini alıyoruz.

```
using Photon.Pun.Demo.PunBasics;
```

2...Sonrasında ise oyuncumuzun start methodunun içerisine txt dosyasındaki kodları yapıştırıyoruz. Artık kameramız oyuncularımızı takip edecektir.



18) Score sistemi

Oyun esnasında bizim bazı verilerimiz bulunmakta örneğin can, skor, altın, para gibi oyunculara özgü bazı değerler olabilir ve her oyuncu da bu değerler farklı olabilir. İşte bu işlemleri nasıl yapacağımızı göreceğiz.

Birden fazla değer tutmamız gerekiyorsa yazacağımız fonksiyon bu işlemleri rahatlıkla bize sağlayacak.

Eğer oyununda sadece bir score gibi tek bir veri tutacaksan bunu bir listede vs. tanımlamadan bu işlemi yapabilme imkanı da sunuyor.

Score sistemine bakalım:

1-Eğer ki tek bir veri tutacaksak:

```
PhotonNetwork.LocalPlayer.AddScore(0);  
PhotonNetwork.LocalPlayer.GetScore();  
PhotonNetwork.LocalPlayer.SetScore(PhotonNetwork.LocalPlayer.GetScore() + 1);  
PhotonNetwork.LocalPlayer.SetScore(PhotonNetwork.LocalPlayer.GetScore() - 1);
```

2-!Konu Dışı

Eğer bir objeyi yok etmek istiyorsak mutlaka ve mutlaka o objenin kendisi olup olmadığını kontrol etmeliyiz. Yok ise hatalar ile karşı karşıya geliriz.

```
public class Parc : MonoBehaviour  
{  
    2 references  
    PhotonView pw;  
    0 references  
    void Start()  
    {  
        pw = GetComponent<PhotonView>();  
        StartCoroutine(DestroyObject());  
    }  
  
    1 reference  
    IEnumerator DestroyObject()  
    {  
        yield return new WaitForSeconds(2);  
        DestroyParc();  
    }  
  
    0 references  
    private void OnParticleCollision(GameObject other) {  
        if(other.transform.gameObject.CompareTag("Player")){  
            /*Player objesine carparsa carptigi objenin icerisindeki TakeAblow fonksiyonunu burada  
            caldirarak oyuncu et dogayindaki emgideki bilgi kaldirdik. Cunku iki defa aynı methodun  
            calistirilmesini tabiki istemiyoruz.*/  
            other.transform.gameObject.GetComponent<PhotonView>().RPC("TakeAblow", RpcTarget.All, 10);  
            /*Carpa işleminden sonra particle effect kendini yok edecek. Eğer carpa yasanmazsa  
            DestroyObject metodu calışık 2 saniye sonra kendini zaten yok edecek.*/  
            DestroyParc();  
        }  
    }  
  
    2 references  
    void DestroyParc(){  
        if(pw.IsMine)  
            PhotonNetwork.Destroy(gameObject);  
    }  
}
```

19) Oyunculara özellikler Tanımlama, Güncelleme – Hashtable – Set Custom Properties

Yukarıdaki tek bir değer tutma işlemi gördük. Şimdi ise birden fazla veri tutma işlemi görelim. Biz toplu verileri veya tutulması gereken verileri array veya listelerde tutuyoruz. Photon networkte ise Hashtable kullanma yöntemini göreceğiz.

!Hashtable normal oyun geliştirilmede de kullanılan yöntemdir.

1-İlk olarak Unity'nin Hashtable kullanmayacağız. PhotonNetwork'ün bize sunduğu Hashtable kullanacağız. Onun için ilk olarak aşağıdaki namespace ekliyoruz.

```
using Hashtable = ExitGames.Client.Photon.Hashtable;
```

2-Sonrasında bir Hashtable değişkeni oluşturuyoruz ve içerisine gerekli olacak default key ve value değerlerini veriyoruz.

```
Hashtable props;
```

```
props = new Hashtable  
{  
    {"key", "value"},  
    {"arrangement", 1},  
    {"win", 0},  
    {"Lose", 0}  
};
```

3- Sonrasında PhotonNetwork ile local oyuncuya gidip Set Custom Properties diyerek, oluşturduğumuz Hashtable oyuncuya veriyoruz.

```
PhotonNetwork.LocalPlayer.SetCustomProperties(props);
```

4-İki yöntem ile biz Hashtable verilerini alabiliriz. (İlk yöntem uzun olan yöntem)

1.İlk yöntem: Yeni bir obje türü oluşturuyoruz. Hashtable anahtarından verdiğimiz key'i bu obje türüne çıkarıyor ve biz bu objeyi yazdırabiliyoruz.

```
if(Input.GetKeyDown(KeyCode.Alpha1)){
    object arrangementValue;
    PhotonNetwork.LocalPlayer.CustomProperties.TryGetValue("arrangement",out arrangementValue);
    print(arrangementValue);
}
```

2.İkinci Yöntem: Objeyi dışarıda tanımlamak yerine içeride tanımını yaparak, ekstra bir satırdan da kurtuluyoruz.

```
if(Input.GetKeyDown(KeyCode.Alpha1)){
    PhotonNetwork.LocalPlayer.CustomProperties.TryGetValue("arrangement",out object arrangementValue);
    print(arrangementValue);
}
```

5- Özelliğimizi alıyoruz. Şimdi ise özelliği güncelleme işlemine geçelim.

Önceki Hashtable değerimizi alıyoruz, üzerinde gerekli değişiklikler yaptıktan sonra set ediyoruz.

```
if(Input.GetKeyDown(KeyCode.Alpha2)){
    props.TryGetValue("arrangement", out object arrangementValue);
    props["arrangement"] = (int) arrangementValue + 1;
    PhotonNetwork.LocalPlayer.SetCustomProperties(props);
}
```

20) Oyunculara özellikler Silme, Ekleme – OnPlayerPropertiesUpdate

Artık Hashtable işlemlerini öğrendik. Oyun esnasında bazı değerleri silebilir bazı değerler ekleyebilirsin. Bu işlemleri görelim

1-Silme

```
if(Input.GetKeyDown(KeyCode.Alpha3)){
    //Server içerisinden kaldırdığımız gibi, oyuncu scriptin içerisindeki
    Hashtable içerisinden de kaldırmamız gerekiyor.*/
    PhotonNetwork.LocalPlayer.CustomProperties.Remove("arrangement");
    props.Remove("arrangement");
}
```

2-Ekleme

```
if(Input.GetKeyDown(KeyCode.Alpha4)){
    props.Add("arrangement",1);
    PhotonNetwork.LocalPlayer.SetCustomProperties(props);
}
```

3-Oyuncunun herhangi bir özelliği değiştiğinde bunu bize söyleyen Photon'un bir fonksiyonu daha mevcut.

Yani en ufak değişimde bile biz haberdar oluyoruz. Böylece sürekli güncellenip güncellenmediğini kontrol etmiyoruz ve daha dinamik bir yapıya sahip oluyoruz.

```
4 references
public override void OnPlayerPropertiesUpdate(Player targetPlayer, Hashtable changedProps)
{
    targetPlayer.CustomProperties.TryGetValue("arrangement",out object value);
    print("Değişenler: " + value);
}
```

4-Son olarak bütün oyuncuların, bütün özelliklerini nasıl alacağımızı göreceğiz.

```
if(Input.GetKeyDown(KeyCode.Alpha5)){
    foreach (Player player in PhotonNetwork.PlayerList)
    {
        player.CustomProperties.TryGetValue("arrangement",out object arrangement);
        print(player.NickName + " oyuncunun sırası: " + arrangement);
    }
}
```