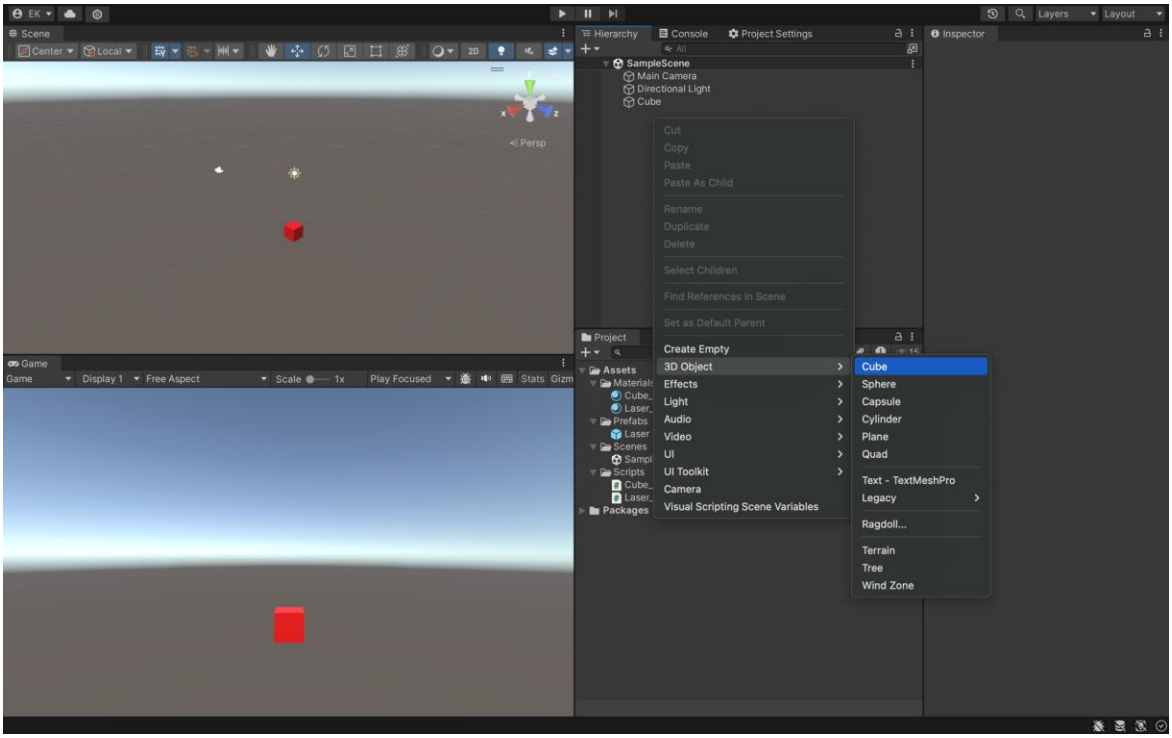
Oyun Programlama Ödevi – Hafta 3

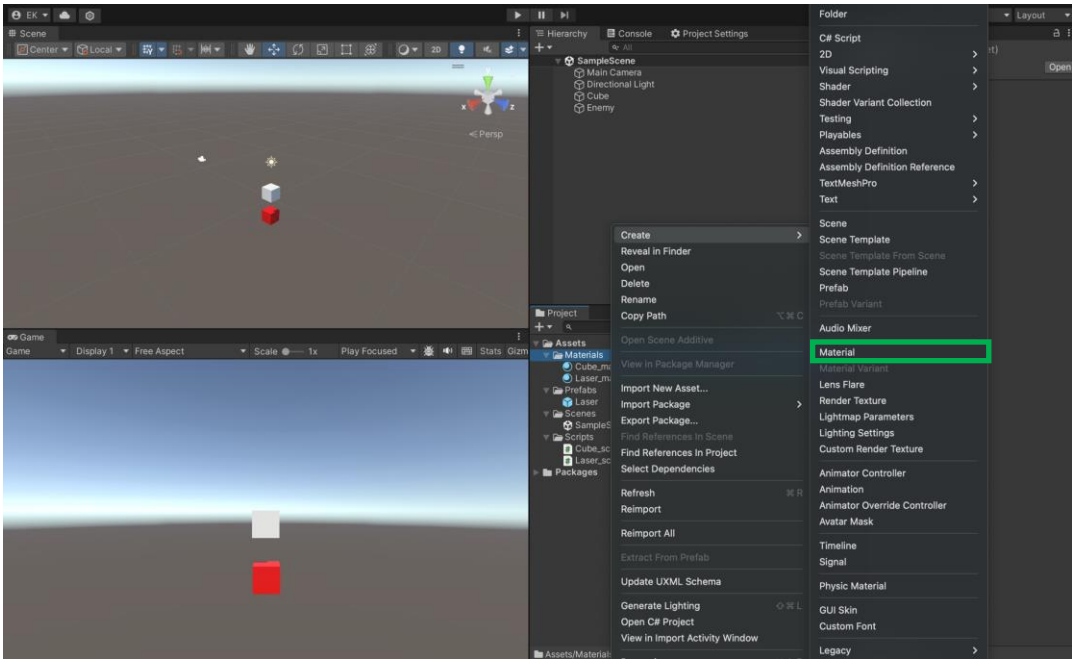
Ad: Eren
Soyad: Köse
Numara: 22360859075
GitHub Kod Linki: https://github.com/erennkose/btu-oyun-programlama/tree/main/Hafta3

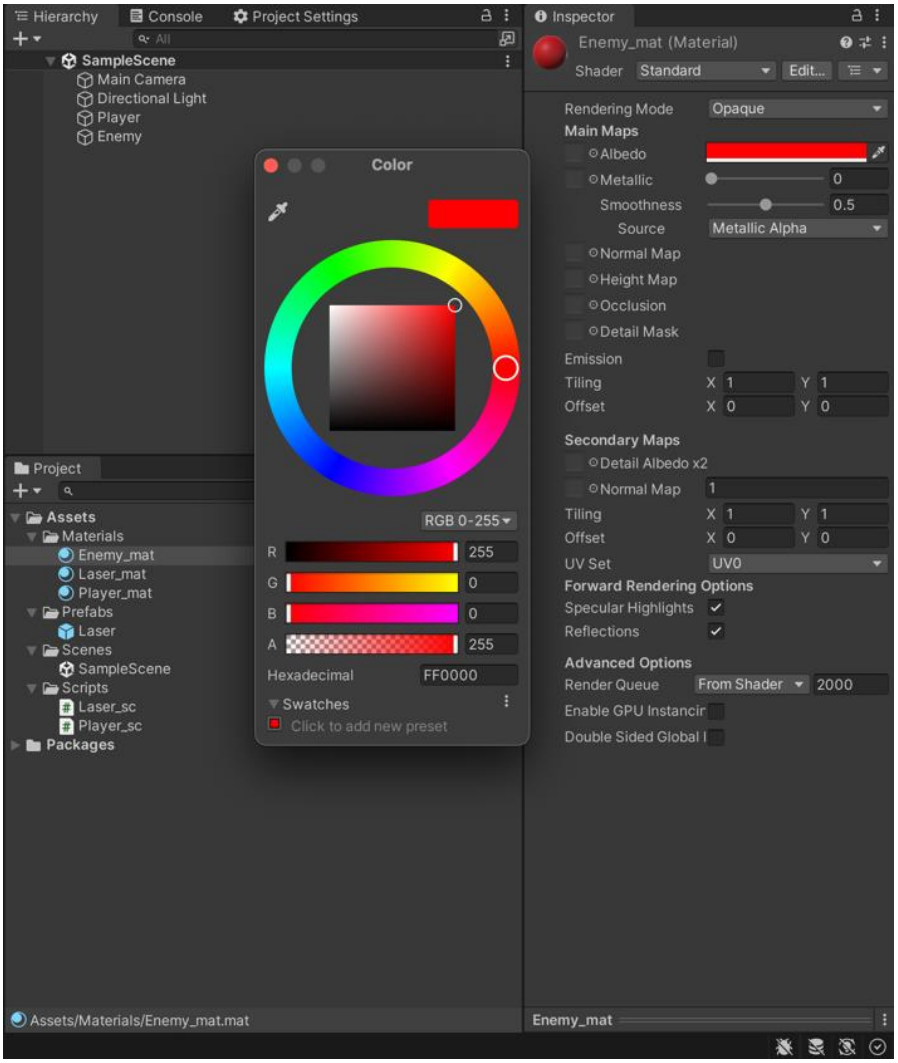
- ## Enemy Prefab'i Oluşturma

  Enemy prefabi oluşturmak için öncelikle bir enemy nesnesi oluşturuyoruz. **Hierarchy** kısmındaki boşluğa sağ tık atıp **3D Objects > Cube** adımlarını kullanarak bu işlemi yapabiliriz.

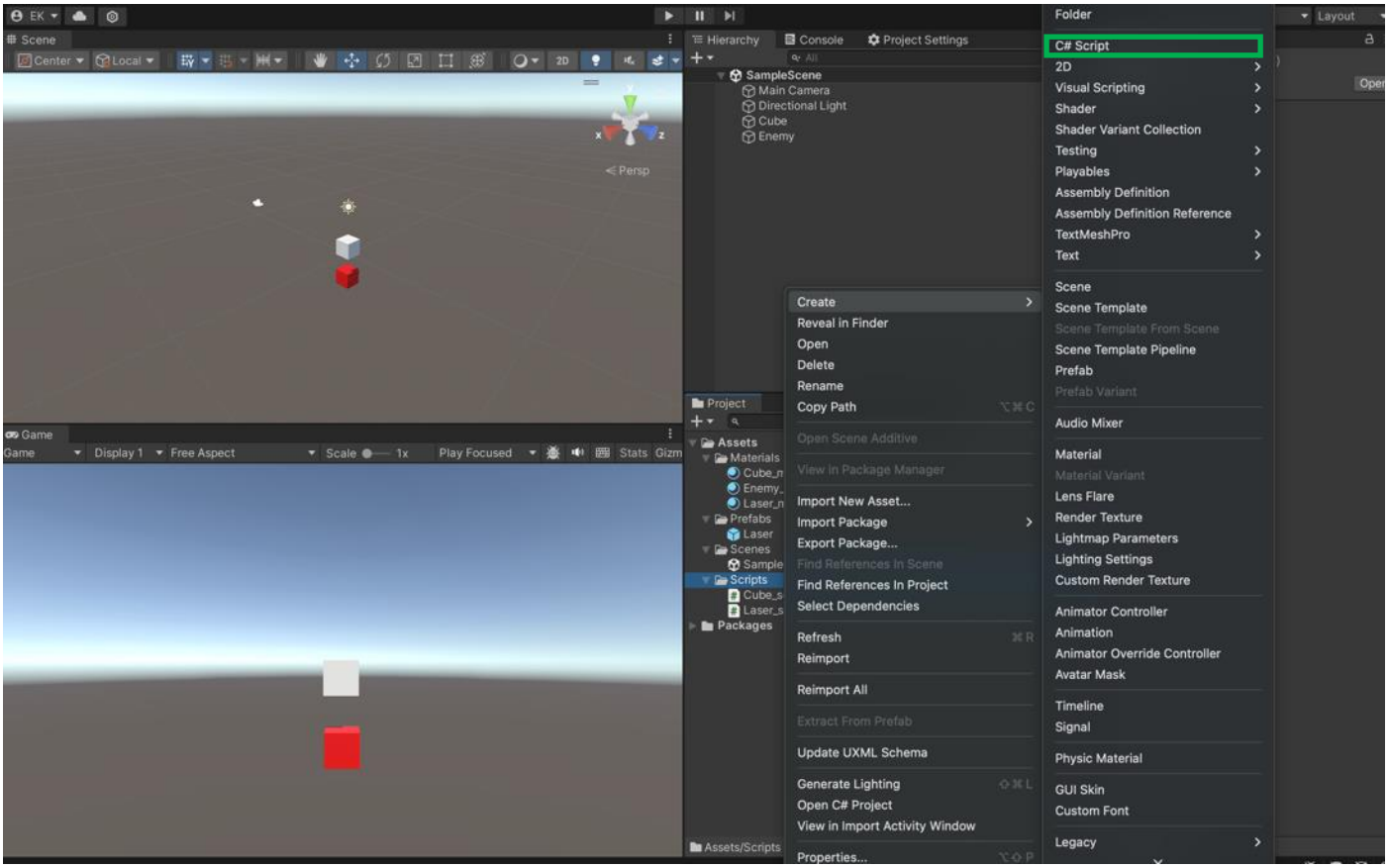


Enemy nesnemizi oluşturduktan sonra Enemy prefabimizde istediğimiz özellikleri Enemy nesnemize eklemeye başlıyoruz. Öncelikle Enemy materyali oluşturuyoruz. Bunun için **Materials** klasörümüze sağ tık atıp **Create > Material** adımlarını takip ederek materyalimizi oluşturup **Enemy_mat** şeklinde isimlendiriyoruz. **Enemy_mat** a tıkladıktan sonra **Inspector** kısmında açılan kısımdan **Albedo** bölümünden Enemymizin rengini kırmızı yapıyoruz.

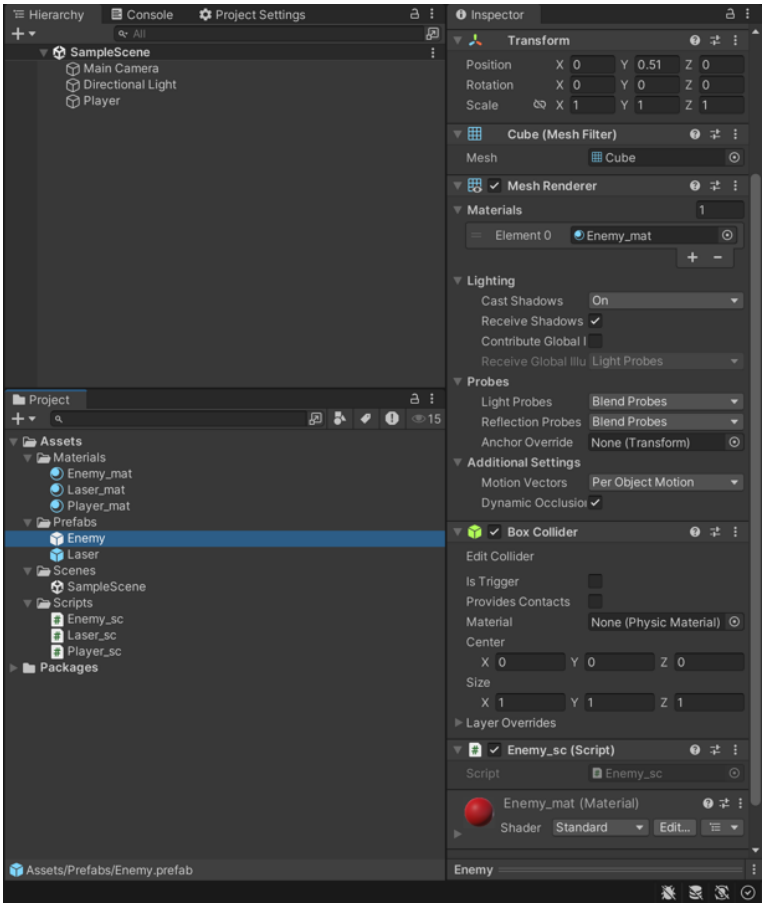




Aynı şekilde **Scripts** klasörümüze sağ tık atıp **Create > C# Script** seçeneklerini izleyerek **Enemy_sc** adlı bir script dosyası oluşturuyoruz.



Bu oluşturmalardan sonra, elimizde artık küpümüz gibi bir de enemy nesnemiz olmuş oldu. Karışıklığın önüne geçmek adına **Cube** adındaki nesnemizin bizim **Playerımız** olduğunu belirtmek istiyoruz. Bu yüzden **Cube** nesnemize sağ tık atıp **Rename** seçeneğini seçtikten sonra **Cube** nesnemizin yeni adını **Player** yapıyoruz.
Scriptimizin adını değiştirmek kod içerisinde sorun yaratabileceğinden dolayı **Player_sc** adında bir script oluşturup, **Cube_sc** içerisindeki kodu kopyalayıp yeni oluşturduğumuz **Player_sc** scriptinin içine olduğu gibi yapıştırıyoruz. Yapıştırdıktan sonra class adındaki **Cube** yerine **Player** yazıyoruz ve kaydediyoruz. Ardından **Unity** uygulamamıza dönüp **Cube_sc** adındaki scripte sağ tık atıp **Delete** seçeneğini seçiyoruz. Aynı zamanda **Player** nesnemizden de **Cube_sc** scriptine sağ tık atıp **Remove Component** seçeneğine tıklayarak nesnemizi bu scriptten kurtarıyoruz. Bu işlemlerden sonra **Player_sc** scriptimizi sürükleyerek **Player** nesnemiz üzerine bırakıyoruz. Ardından Player nesnemize tıklıyoruz ve **Inspector** kısmındaki **Player_sc** bölümünden **Laser Prefab** yazan yerin sağındaki noktaya tıklayarak **Laser** prefabimizi seçiyoruz. Böylece yeni doğru isimli scriptimizi Player nesnemize entegre etmiş oluyoruz.
Aynı şekilde **Cube_mat** adlı materyalimize de sağ tık atıp **Rename** seçeneğini seçiyoruz ve **Cube_mat** materyalimize **Player_mat** adını veriyoruz. Player_mat a tıkladıktan sonra Inspector kısmında açılan kısımdan **Albedo** bölümünden rengini beyaz yapıyoruz.
Bu işlemleri tamamladıktan sonra Enemy prefabimizi oluşturmaya devam edebiliriz.
Oluşturmuş olduğumuz **Enemy_mat** materyalini ve **Enemy_sc** scriptini **sürükle-bırak** yöntemini kullanarak **Enemy** nesnemizin üzerinde bırakıyoruz. Böylece **Enemy** nesnemiz artık **Enemy_mat** ve **Enemy_sc** komponentlerine entegre olmuş oluyor. Tüm bunları yaptıktan sonra ise **Enemy** nesnemizi **sürükle-bırak** yöntemini kullanarak **Prefabs** klasörümüzün içerisine aktarıyoruz. Böylece Enemy Prefabimiz oluşmuş oluyor. Ardından **Hierarchy** kısmından **Enemy** nesnemizi artık silebiliriz.



- **Enemy Hareketini Sağlayan Kod**

  Öncelikle bu kodu yazmak için **Enemy_sc** scriptimize iki kez tıklayarak kod editörümüzde bu scripti açıyoruz. Scriptte öncelikle fonksiyonların dışında, classımızın içinde bir **speed** değişkeni tanımlayıp üretilecek enemy nesnemizin hızını belirliyoruz.

```
[SerializeField] float speed = 3.0f;
```

  Hız değişkenimizi oluşturduğumuza göre, enemynin hareketini sağlayacak fonksiyona geçebiliriz. Bu fonksiyonda öncelikle laser ile öldürülmeyen bir nesnenin **mapten aşağı düştüğü zaman tekrar mapin üstüne, yeni bir enemymiş gibi gözükebileceği bir konuma ışınlanması** durumunu sağlayacağız. Sürekli yeni nesne üretmek yerine bu if koşuluyla birlikte optimizasyonu sağlayacağız. Bu if koşulunun else durumu ise nesnenin mapte olduğu durumu belirtmektedir. Bu durumda ise biz enemymizin belirlediğimiz hızda mapte yukarıdan aşağı düşmesini istiyoruz. Yukarıdan aşağı düşmesini sağlamak için else kısmında bir **vektör** oluşturuyoruz ve **y ekseninde 1 birim aşağı** gösterecek şekilde değerlerini giriyoruz. Ardından **transform.Translate** kullanarak ve parametre olarak oluşturduğumuz **vektör, Time.deltaTime, speed değerlerinin çarpımlarını** giriyoruz. Bu sayede bir saniyede belirlediğimiz vektörün yönünde belirlediğimiz hızda oluşan enemy nesnemizi hareket ettirebiliyoruz. if durumumuza bakacak olursak bu durumda da enemy nesnemizin **rastgele bir x konumundan** yeniden mape gelmesini istiyoruz. Bunu sağlamak için ise **Random.Range** fonksiyonuna map sınırlarını parametre olarak girerek rastgele bir değer oluşturabiliyoruz. Bunu da bir değişkene atadıktan sonra **transform.position** fonksiyonuna parametreleri bir vektör üzerinden sırasıyla girerek (x,y,z) nesnemiz laserle öldürülmediyse rastgele bir x konumunda mapin üst kısmından yeniden mape girmesini sağlıyoruz.

```
void Movement(){
    if(transform.position.y < -5.38){
        float randomX = Random.Range(-9.4f,9.4f);
        transform.position = new Vector3(randomX, 7.38f, transform.position.z);
    }
    else{
        Vector3 direction = new Vector3(0,-1,0);
        transform.Translate(direction * Time.deltaTime * speed);
    }
}
```

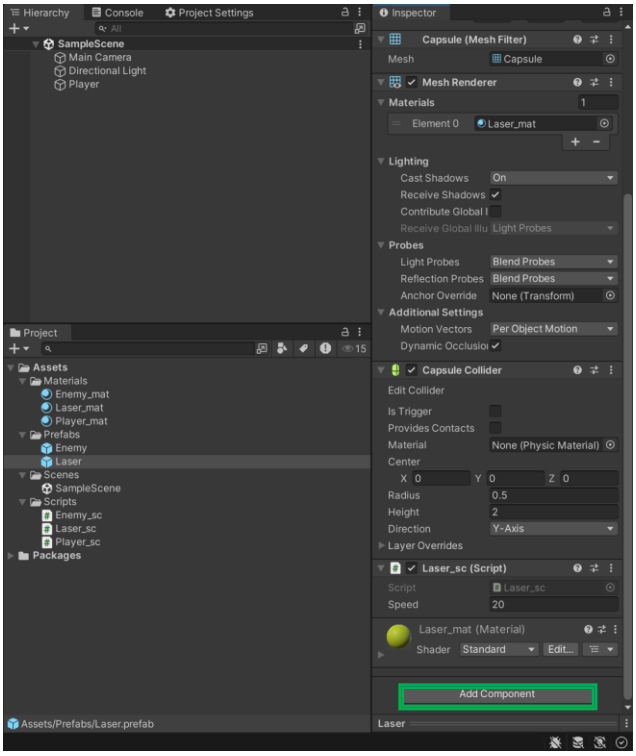
  Bu fonksiyonu oluşturduktan sonra bu if-else durumlarının her framede kontrol edilmesini istediğimiz için **Update** fonksiyonu içerisinde yeni oluşturduğumuz bu fonksiyonu çağırıyoruz.

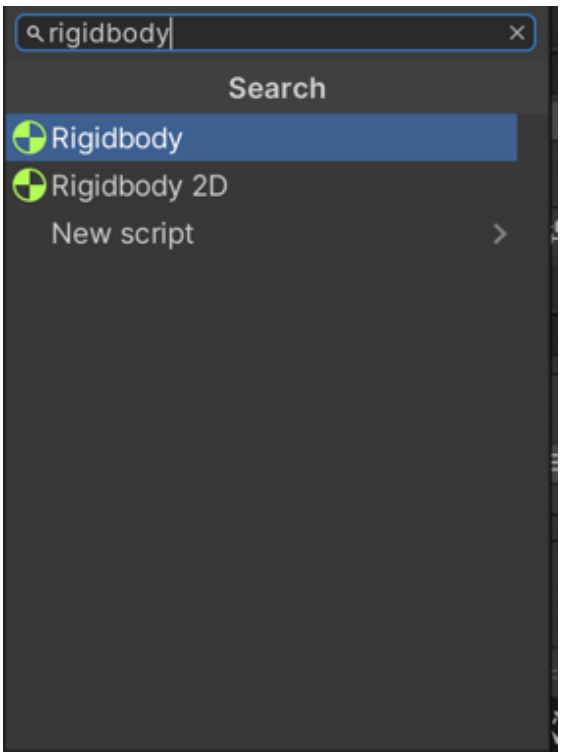
```
void Update()
{
    Movement();
}
```

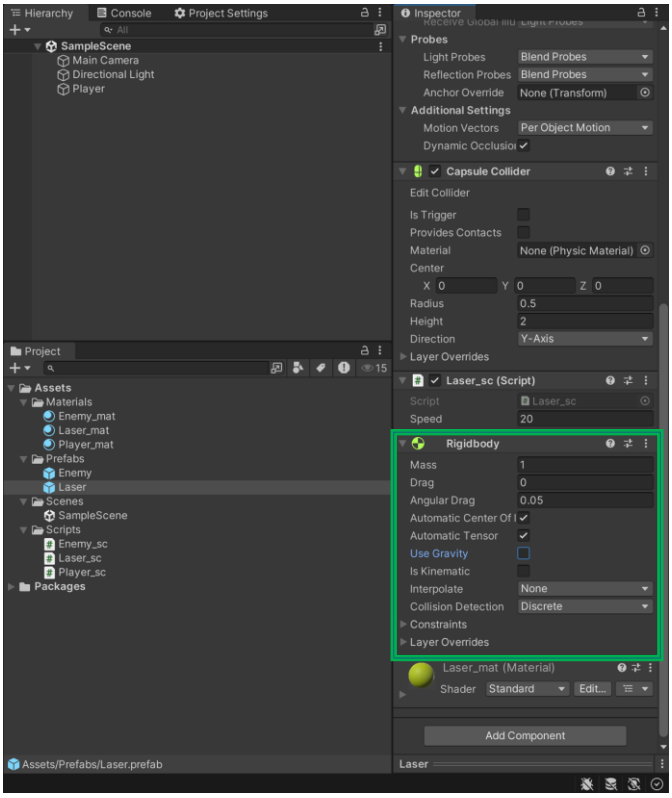
  Bu şekilde prefab üzerinden oluşturulacak olan Enemy nesnelerimizin hareketini sağlamış oluyoruz.

- **<u>RigidBody ve Collision Component'lerinin Eklenmesi</u>**

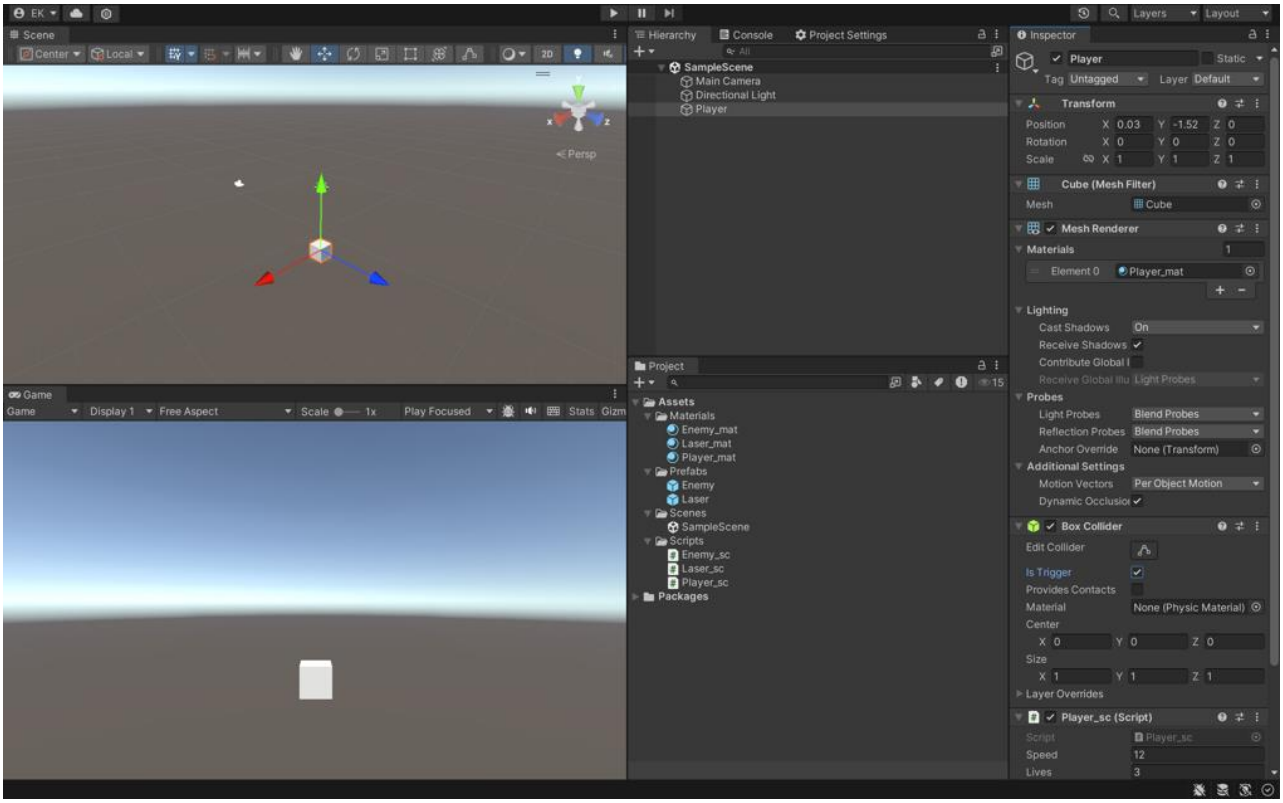
  RigidBody adlı bileşen, oyunumuz içerisinde fiziksel hareketlerin taklit edilmesini sağlamaktadır. Bundan dolayı oyunumuzda bazı nesnelerde RigidBody bileşenini kullanmak istiyoruz. Biz de oyunumuzda **Laser** ve **Enemy** prefableri için RigidBody bileşenini kullanacağız. Bunun için öncelikle Laser prefabimize tıklıyoruz. Sağ tarafta açılan **Inspector** kısmında en altta bulunan **Add Component** kısmına tıklıyoruz. Ardından açılan Pop-Upta arama kısmına **RigidBody** yazıyoruz. Karşımıza çıkan RigidBody bileşenine tıklıyoruz ve bu bileşeni Laser prefabimize eklemiş oluyoruz. Ekledikten sonra ise yerçekimi kullanmak istemediğimiz için **Inspector** kısmındaki **RigidBody** bölümünden **Use Gravity** seçeneğini **kapatıyoruz**. Bunu aynı şekilde Enemy prefabimize de uyguluyoruz. Bu sayede fiziksel hareketleri bu prefablerden oluşacak olan tüm nesnelerimize eklemiş oluyoruz.

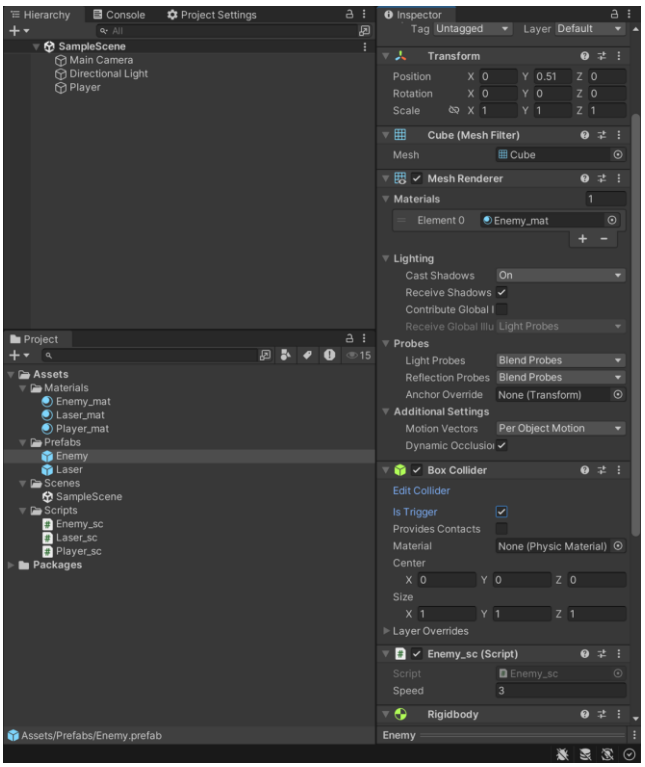


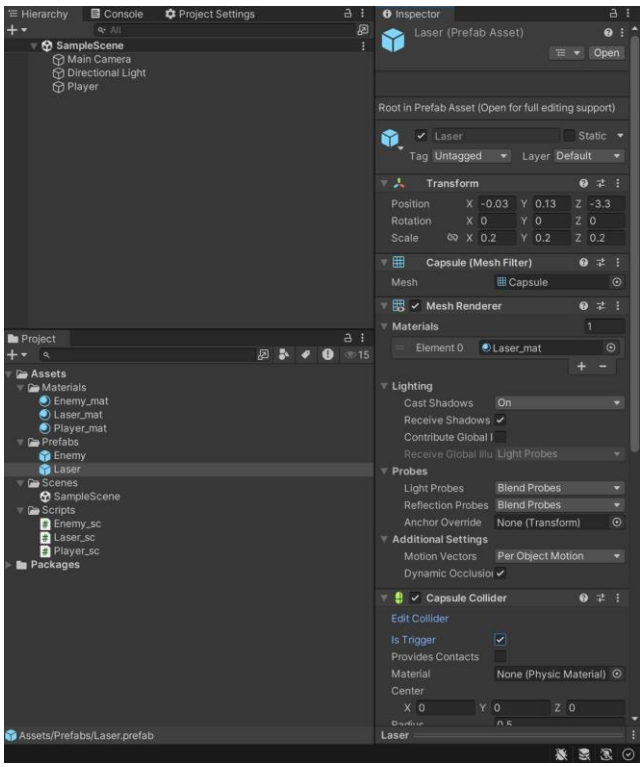




RigidBody bileşenimizi gerekli yerlere ekledikten sonra ise Collision işlemlerimize geçebiliriz. **Collision**, eklediğimiz nesnenin ve diğer ekli nesnelerin temas ettiğinde etkileşimde bulunmasını sağlayan bileşendir. Elimizde Player nesnesi, Laser prefabi ve Enemy prefabi bulunmakta. Bu iki prefab ve nesnemizin etkileşime geçme durumlarını göz önüne alacağımız için üçünde de collision bileşeni aktifleştirilmeli. Collision bileşenimizi aktifleştirmek için öncelikle **Player** nesnemize tıklıyoruz. Tıkladıktan sonra **Inspector** kısmında **Box Collider** bölümüne geliyoruz. Colliderın Box olmasının sebebi Player nesnemizi küp olarak oluşturmamızdır. Aynı işlemleri Laser için yaparken de **Capsule Collider** yazdığını görebileceğiz. Box Collider bölümünün altında yazan **Is Triggerd** seçeneğinini işaretleyerek aktif hale getiriyoruz. Böylece Player nesnemiz için Collision bileşeni ayarlanmış oluyor. Aynı yöntemle Enemy prefabine ve Laser prefabinin de Collision bileşenini aktifleştiriyoruz. Bu sayede Collision komponentleri nesnelerimiz için aktifleştirilmiş oluyor.

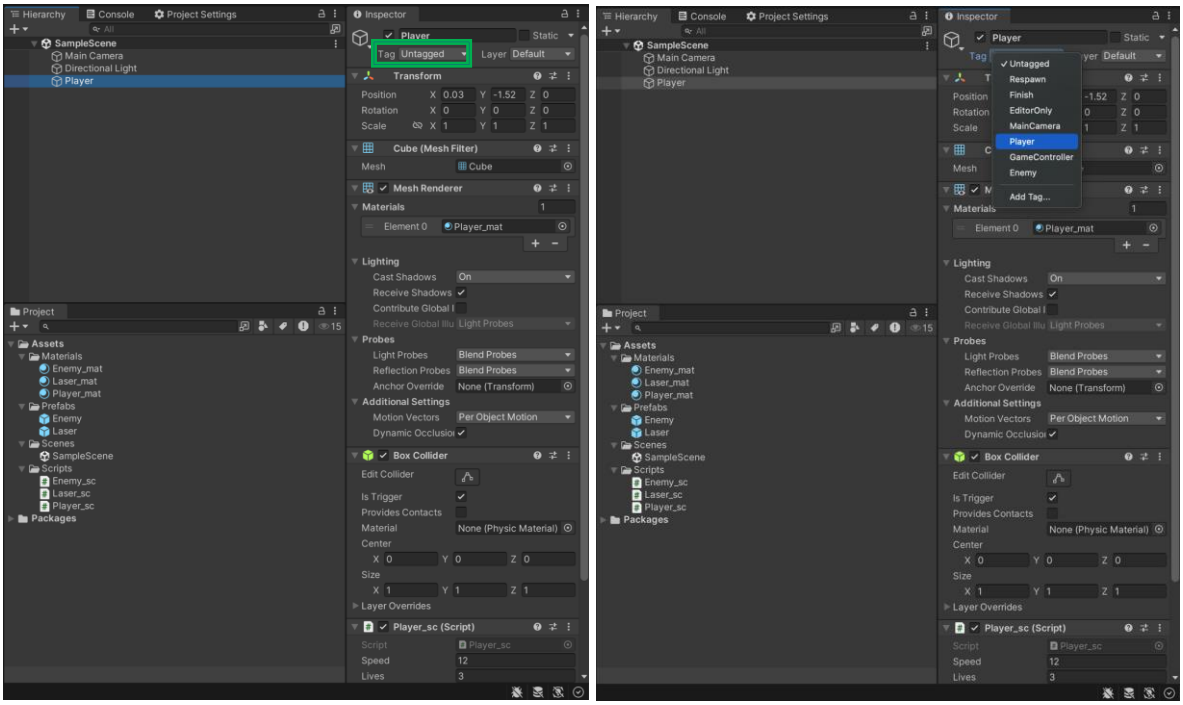




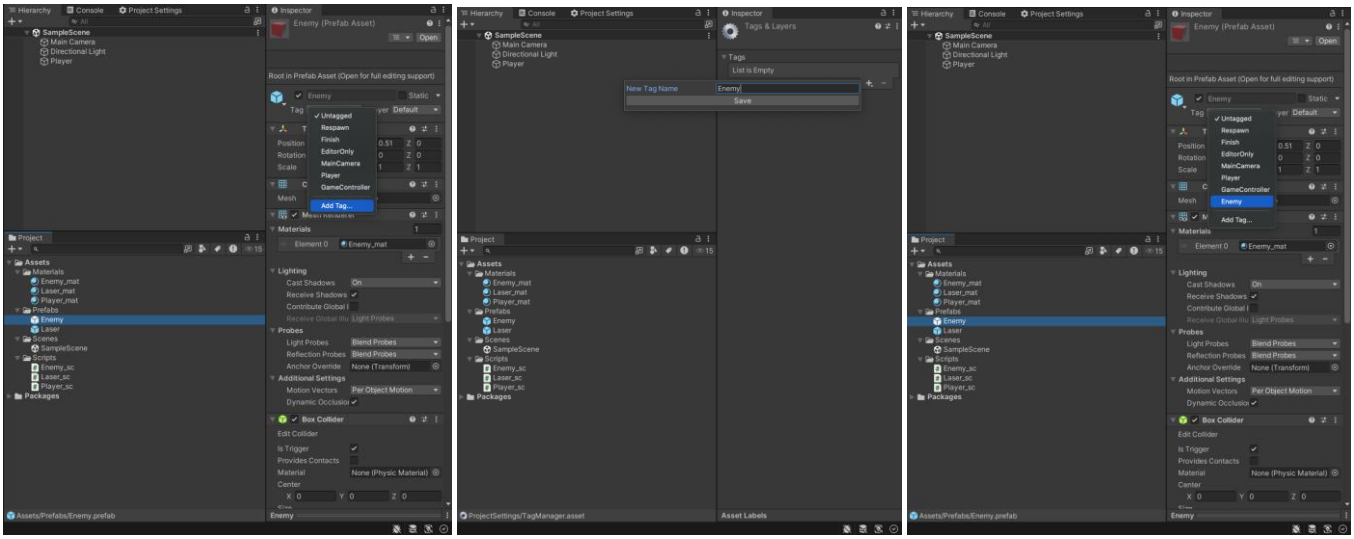


- **<u>OnTriggerEnter Fonksiyonu ve Çarpışmaların Tespiti ile Gerekli Aksiyonların Alınması</u>**

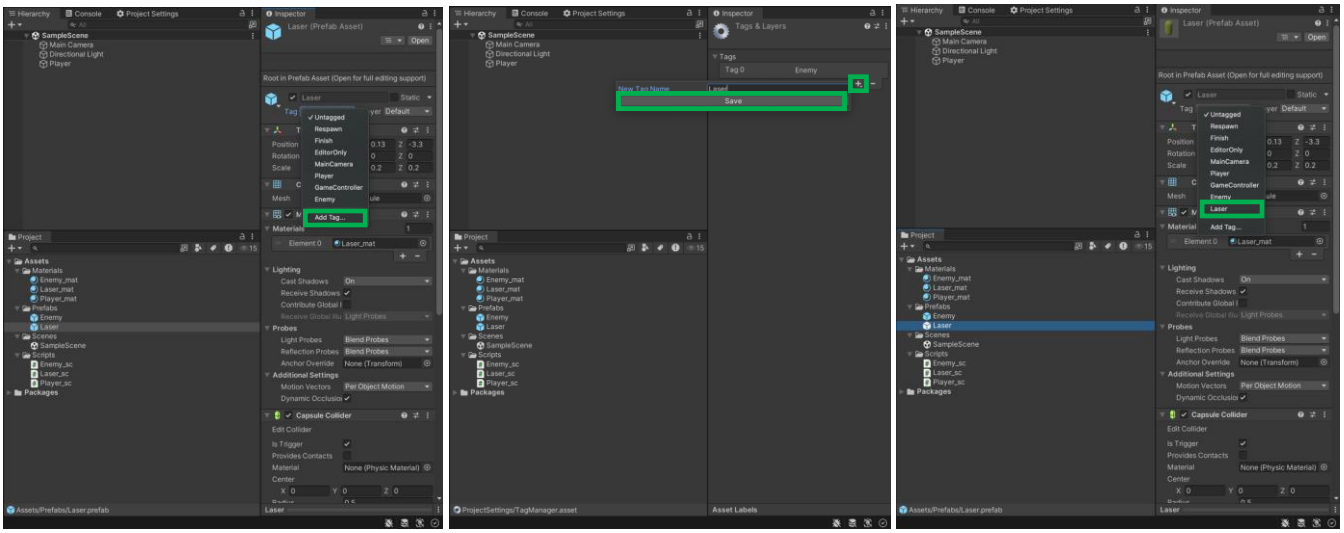
  OnTriggerEnter fonksiyonumuz Enemy prefabimizden oluşturan nesnelerin Player nesnesi ve Laser prefabinden oluşturan nesneler ile etkileşime geçtiğinde yapılması gerekli olan işlemleri belirten bir fonksiyondur. Bu fonksiyona parametre olarak **(Collider other)** veriyoruz. Bu parametre sayesinde bir Collider ile temas olduğu zaman bu fonksiyon tetikleniyor. Fonksiyonun içinde ise fonksiyonun çalışmasını tetikleyen Collider hangi nesneye aitse Enemymizin ona göre tepki vermesini istiyoruz. Bu yüzden if-else koşullarıyla Colliderın kime ait olduğunu tespit edeceğiz. Bunun için ise kodda nesnelerimizi ve prefablerimizi kullanabilmemiz gerekiyor. Bunun için ise Player nesnemize, Laser prefabimize ve Enemy prefabimize **Tag** ataması yapmamız gerekiyor. Bu durumu sağlamak için öncelikle **Unity** uygulamamıza dönüyoruz. **Player** nesnemize tıklıyoruz ve **Inspector** kısmında en üst bölümde yer alan **Tag** seçeneğine tıklıyoruz. Default olarak **Untagged** yazan yeri seçeneklerde bulunan **Player** seçeneği ile değiştiriyoruz. Bu sayede Player nesnemizi Player olarak etiketlemiş oluyoruz.



Ardından **Enemy** prefabimize tıklıyoruz. Aynı şekilde **Inspector** kısmında en üstte bulunan **Tag** seçeneğine tıklıyoruz. **Untagged** olan yeri değiştireceğiz. Fakat bu kez Enemy için kendimiz bir tag oluşturmamız gerekiyor. Bunun için **Untaggede** tıkladıktan sonra seçeneklerin en altında bulunan **Add Tag** seçeneğine tıklıyoruz. **List is empty** yazan kısımın en sağında bulunan **artı** işaretine tıklayarak **Enemy** adında bir **tag** oluşturuyoruz. Oluşturduktan sonra tekrar **Enemy** prefabimize tıklıyoruz ve **Untagged** olan kısmı **Enemy** olarak değiştiriyoruz.



Enemy prefabimiz için yaptığımız aynı işlemleri **Laser** prefabimiz için yapıyoruz. Laser prefabimiz için **Tagimizi Laser** olarak oluşturuyoruz ve ekliyoruz.



Taglerimizi eklediğimize göre artık OnTriggerEnter fonksiyonumuz içerisinde nesnelerimize ve prefablerimize erişebileceğiz.

OnTriggerEnter fonksiyonumuzdaki parametrede tespit edilen nesne eğer Player nesnesi ise Player nesnemiz ile Enemy nesnesi çarpışmış demektir. Bu durumda önce Player nesnemizin canını azaltacak fonksiyon olan **Damage** fonksiyonunu çağıracağız. Bu fonksiyon Player_sc scriptinde bulunacağından dolayı önce **Player_sc** scriptinden bir **nesne** oluşturup bu nesneye **other.GetComponent<Player_sc>()** kod parçasını atarsak Player_sc içerisindeki fonksiyonlara erişebiliriz. Bu şekilde **Damage** fonksiyonunu çağırıyoruz ve Player nesnemizin canını bir adet düşürüyoruz. Ardından ise Player ile çarpışan Enemy nesnemizi yok etmek için **Destroy** fonksiyonunu kullanıyoruz ve Enemy nesnemizi yok etmiş oluyoruz.

Eğer Enemy nesnemizin çarpıştığı nesne bir Laser nesnesi ise **önce Laser** nesnesini, **sonra** ise **Enemy** nesnesini **Destroy** kullanarak yok ediyoruz.

Bu sayede de OnTriggerEnter fonksiyonu ile çarpışılan nesnenin türünü tespit edip, çarpışmalara göre gerekli nesneleri yok etmiş oluyoruz.

```
void OnTriggerEnter(Collider other){
    if(other.tag == "Player"){
        // Canini azalt
        Player_sc playersc = other.GetComponent<Player_sc>();
        playersc.Damage();
        Destroy(this.gameObject);
    }
    else if(other.tag == "Laser"){
        Destroy(other.gameObject);
        Destroy(this.gameObject);
    }
}
```

- **<u>Oyuncu Hasar Alma İşlemlerinin Yapılması</u>**
  Playerımıza bir Enemy nesnesi çarpması durumunda Player nesnemizin canının azalması durumunu sağlamalıyız. Bu durumu sağlamak için ise Player_sc scriptimiz içerisinde Player nesnemizin canını tanımlamamız gerekmekte. Bunun için classımız içerisinde, fonksiyonların dışında **lives** adında bir integer değişken tanımlıyoruz.

```
[SerializeField]int lives = 3;
```

  lives integer değişkenimizi tanımladıktan sonra fonksiyonların dışında, classımızın içerisinde **public** olarak **Damage** adında bir fonksiyon tanımlıyoruz. Bu fonksiyonu public tanımlamamızın sebebi Enemy_sc scriptinde, Player_sc nesnesi oluşturarak bu fonksiyonumuza erişmek istememizdir. Çünkü Enemy nesnesi ile Player nesnesinin çarpışmasını durumunu Enemy_sc içerisinde kontrol ediyoruz. Temas etmeleri durumunda bu **Damage** fonksiyonunu çağırarak Player nesnemizin canını bir azaltmayı sağlıyoruz. Canımızın 1'den aşağı olması durumunda ise Player nesnemizin yok edilmesini istiyoruz. Bunu sağlamak için ise Destroy fonksiyonunu canının 1'den aşağıda olması if koşulu altında çağırıyoruz. Bu sayede de oyuncunun hasar alma işlemleri tamamlanmış oluyor.

```
public void Damage(){
    lives--;
    if(lives < 1){
        Destroy(this.gameObject);
    }
}
```