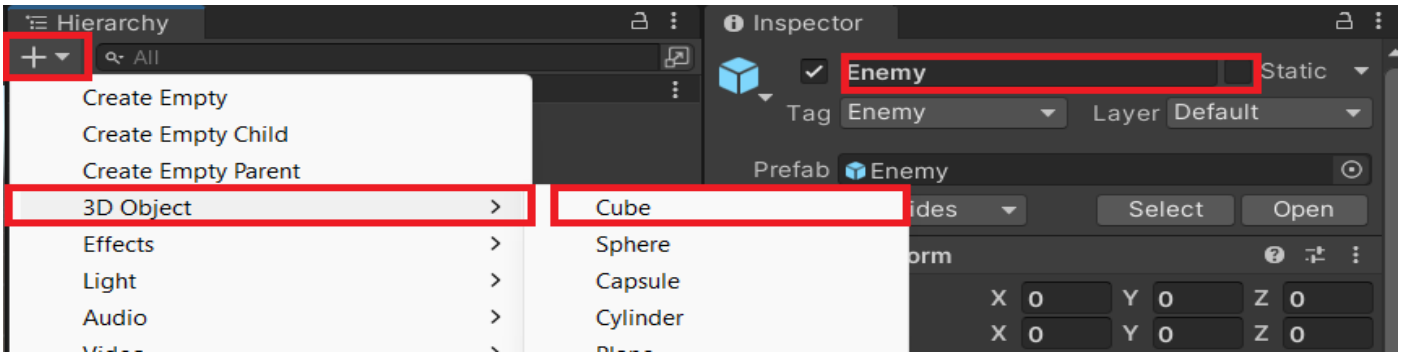
Öğrenci: Ahmet Yusuf Birdir
Numara: 21360859026

Enemy Prefab Oluşturma

Öncelikle Hierarchy panelinden “+” kısmına gelin. Ardından 3D object kısmını seçin ve Cube seçeneğini seçerek sahnenize 1 adet küp nesnesi oluşturun. Oluşturduğunuz küp nesnesinin ismini Enemy yaparak sahnenize bir Enemy nesnesi oluşturmuş olacaksınız.



Ardından Bu nesneyi daha önceden oluşturduğunuz Prefabs klasörüne sürüklediğinizde Enemy prefab oluşmuş olacak.

Enemy Hareketlerinin Kodlanması

İlk olarak istediğimiz şey Düşmanın sahnenin üst kısmından görünmeyen bir kısımdan sahnenin sınırları içine düşecek şekilde belirli bir hızda rastgele şekilde oluşup aşağı düşmesi ve eğer herhangi bir şekilde bir şeye çarpmadan sahnenin altından çıkarsa tekrardan rastgele bir üst konuma geri dönmesi.

Bunun için önce Scripts klasörüne bir adet Enemy_sc adında script klasörü oluşturmak. Daha sonra hareket için gereken değişken ve fonksiyonları eklemek.

Eklenecek ilk şey bir hız değişkeni. Bu değişken Enemy nesnemizin hızını belirleyecek. Yandaki şekilde göründüğü gibi SerializedField ile hızımızı unity uygulamasında da görünecek şekilde ayarlıyoruz ve daha sonra ilk başlangıç noktasını Start fonksiyonu içinde ayarlıyoruz.

```
[SerializeField]
1 reference
float speed = 3.0f;
0 references
void Start()
{
    transform.position = new Vector3(0, 8, 0);
}
```

Daha sonra hareket fonksiyonumuza geçiyoruz. Aşağıdaki görselde olduğu gibi öncelikle Hareketlerimize özel bir fonksiyon belirliyoruz. Fonksiyonun içine ise önce nesnemizin konumunu Translate fonksiyonu ile güncelliyoruz. Bu şekilde update fonksiyonu çağırıldıkça nesnemiz ilerleyecek. Nesnemizin yukarıdan aşağıya doğru hareket etmesini istiyoruz. Bu sebeple Translate fonksiyonu içindeki Vector3.down özelliğini kullanıyoruz. Bu özellik Vector3(0, -1, 0) anlamına gelmekte olan hazır bir durum. Böylelikle nesnemiz sürekli olarak 1 birim aşağı ilerleyecek. Tabi bu durumu saniye başına ve istediğimiz hıza ayarlamak için speed ve Time.deltaTime ile çarpıyoruz ve zaman normalizasyonu işlemi uyguluyoruz.

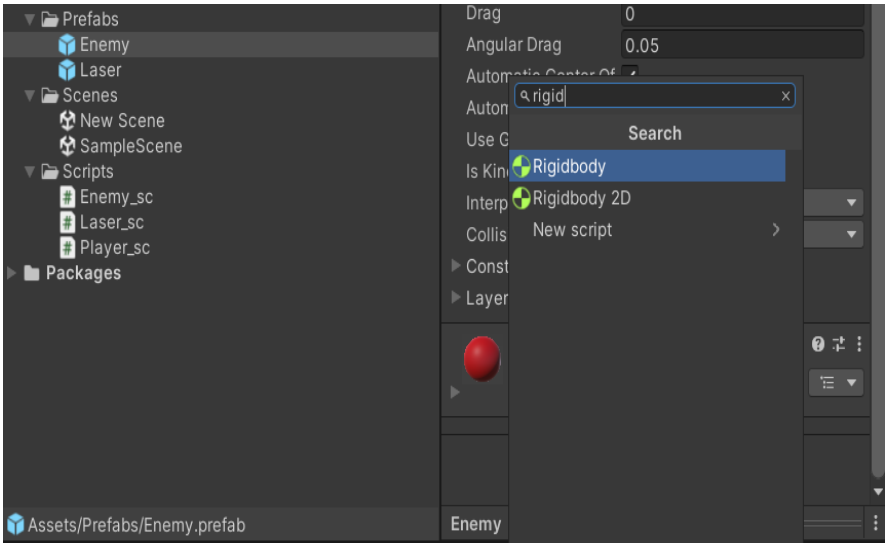
```
void CalculateMovement(){

    transform.Translate(Vector3.down * speed * Time.deltaTime);
    if(transform.position.y < -5.4f){
        int minValue = -9;
        int maxValue = 9;
        float random = Random.Range(minValue,maxValue);
        transform.position = new Vector3(random,7.5f,0);
    }
```

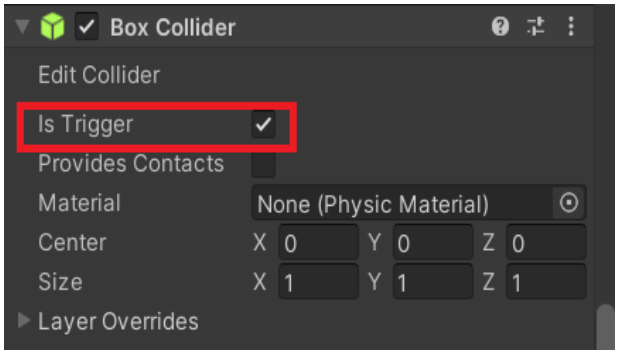
Daha sonra ise bir if bloğu ile eğer Enemy nesnemiz sahnenin altından çıkar ise tekrar rastgele bir x pozisyonundan sahneye geri dönecek şekilde , sahnenin üzerine gönderiyoruz. Sahne yatay sınırlarını belirledikten sonra Random.Range fonksiyonuna yatay sınırlaroı vererek bir x pozisyonu oluşturuyoruz. Daha sonra transform.position ile oluşturduğumuz x ve sahne üst sınırını seçerek nesnemizi oraya ışınlıyoruz. Geri kalan tek şey ise bu script dosyamızı sürükleyip Enemy prefab ile birleştirmek.

Çarpışma Eklentileri

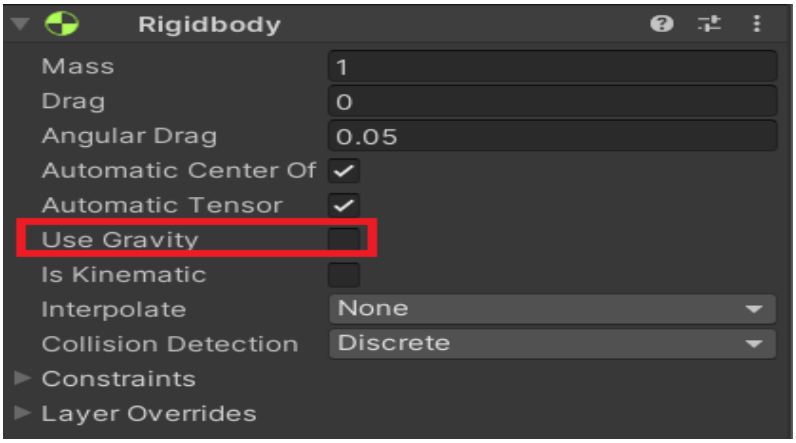
Nesneleremizin teması durumunda bu temasın oluşması ve algılanması için gerekli 2 eklentiyi Enemy nesnemize ekleyeceğiz. Bunlardan biri daha önce de kullandığımız rigidbody eklentisi. Bu eklentiyi inspector panelinin en alt kısmındaki add component kısmından enemy prefab'a ekliyoruz.



Bir diğer eklenti ise nesneler ile hazır gelen Collider eklentisi. Bizimm durumumuzda Enemy nesnesi üzerinde Box Collider şeklinde gözükecek olan bir eklentidir. Collider eklentisi bir çarpşma olup olamdığını tespit etmek için kullanılacak ve bu sebeple ” is trigger” seçeneğini aktif hale getiriyoruz.



Unutmadan Rigidybody eklentisi ise nesnemize fiziki bir yapı sağlıyor olacak. Yani nesnemiz fizik kurallarına göre hareket edecek. Bu durumda biz zaten nesnemizi aşağı hareketini tanımladığımızdan “Use gravity” özelliğini inaktif hale getiriyoruz, aksi halde nesnemiz aşağı yönde hızlanacaktır.



Çarpışma Tespiti ve OnTriggerEnter Fonksiyonu

Çarpışma tespiti ve sonrasında yapılacaklar için Enemy script fonksiyonumuza OnTriggerEnter isminde bir fonksiyon oluşturuyoz.

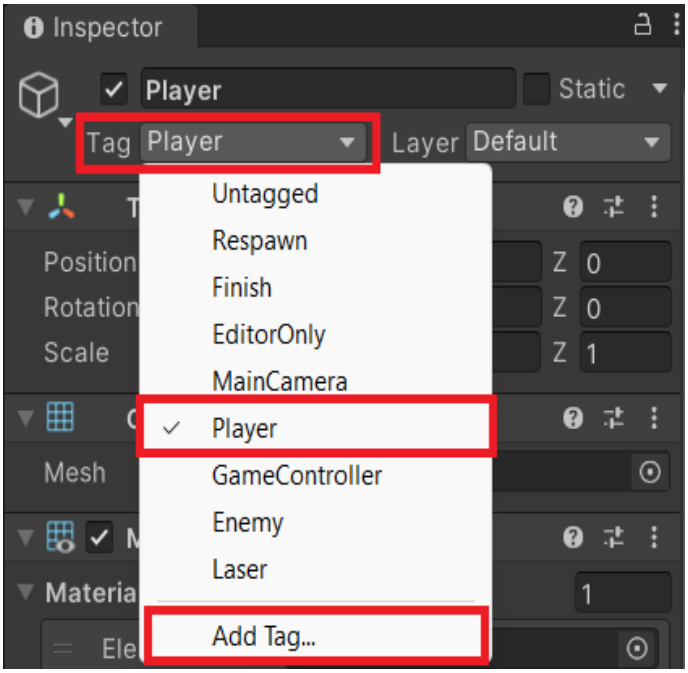
Bu fonksiyonumuz aşağıdaki gibi bir Collider Nesnesini parametre olarak alıyor. Bu ona çarpan nesnenin kendisi olacak. Player nesnemizi kullanabilmek adına öncelikle Player_sc açıldığında oluşan Player_sc sınıfına ait player nesnemizi çağırıyor ve other nesnemiz ile birleştiriyoruz. Bu şekilde player sınıfından methodlar çağırabileceğiz.

```
0 references
void OnTriggerEnter(Collider other){
    Player_sc player = other.GetComponent<Player_sc>();
```

Çarpışmalara geldiğimizde ise burada 2 durum söz konusu: Enemy ile Player çarpışması veya Enemy ile Laser çarpışması.

Her iki durum farklı olduğundan bu iki duruma özel if, else if blokları oluşturuyoruz.

İf koşullarımız içinde kullanacağımız other.tag ile nesnemizin player veya laser olup olmadığını anlayacağız ancak bunun için daha önceden oluşturduğumuz nesnelerimizin tag kısmını seçmeliyiz. Bunun için nesnelerimizin inspector panellerindeki tag kısmına gelerek seçim yapıyoruz. Bu kısımda player tag'i hazır olmasına rağmen Enemy ve Laser tagleri olmayacaktır. Bunları Add Tagg kısmından kendiniz ekleyebilirsiniz. Eklemeleriniz yaptıktan sonra bunları Prefab'lara kaydetmeyi unutmayın. Böylelikle oluşan her nesne birbirinden ayrı tutulmuş olacak.



Ardından ilk olarak Laser ile Enemy çarpışmasını inceleyelim. Ardından bu çarpışmanın olup olmadığını anlamak için öncelikle else if bloğu içerisine bir koşul koyuyoruz. Eğer Enemy Laser ile çarpıştıysa her ikisi de yok olmalı. Bunun için Destroy fonksiyonu ile

```
else if(other.tag == "Laser"){
    //? Always destroy the object of script which you are coding
    //* So.. destroy laser first then destroy enemy
    Destroy(other.gameObject);
    Destroy(this.gameObject);
}
```

hem Laser'i hem de Enemy nesnesini yok ediyoruz. Ancak önemli bir husu olarak öncelikle Laser nesnesi ok edilmeli çünkü bu işlem enemy nesnesi üzerinden yapılmakta. İlk olarak Enemy nesnesi yok edilir ise ona çarpan laser nesnesine erişim kalmayacağıdan yok edilemeyecektir.

Şimdi ise Player ve Enemy çarpışmasına gelelim. Burada if bloğu ile çarpışmayı kontrol ettikten sonra yapılacak şey Enemy nesnesinin yok edilmesi ve Player nesnesinin canının azalması. Burada ilk olarak Daha önceden Player_sc sınıfndan bir player

```
if(other.tag == "Player"){
    player.Damage();
    Destroy(this.gameObject);
}
```

nesnesi çağırmıştık ve other nesnemizi ona bağlamıştık. Çarpışma eğer player ile olursa buradan player nesnesinin içindeki Damage fonksiyonu çağrılarak nesnemizin canı azaltılacak. Bu fonksiyon sunumun bir sonraki başlığında anlatılacak. Daha sonra ise Enemy nesnesi Destroy fonksiyonu ile yok edilecek.

Player Hasar Alma İşlemi

Burada Player_sc doyamıza geliyoruz ve öncelikle bir can değişkeni, sonra ise Damage adında bir fonksiyon oluşturuyoruz. Can miktarını istediğiniz şekil ve miktara göre ayarlayabilirsiniz. Burada yapılmış olan mantık Player'ın 3 can hakkı olması ve her çarpışmanın 1 can hakkı götürmesi şeklinde olacak.

```
public int playerHealth = 3;
```

```
1 reference
public void Damage(){
    this.playerHealth--;
    if(playerHealth == 0){
        Destroy(this.gameObject);
    }
}
```

Damage fonksiyonu içerisinde ilk olarak çalıştığında aktive olacak şekilde çarpılan nesnenin canı 1 azaltılmalı bu sebeple this.playerHealt - - ile çappışan Player nesnesinin canı 1 azaltılır. Daha sonra ise can haklarının konrolü gerçekleştirilir. Eğer can hakkı bitmiş yani 0'a ulaşmış ise Player nsenesi yok edilir ve oyun biter.

GitHub : https://github.com/ahmetyusufbirdir03/Oyun-Programlama-Dersleri