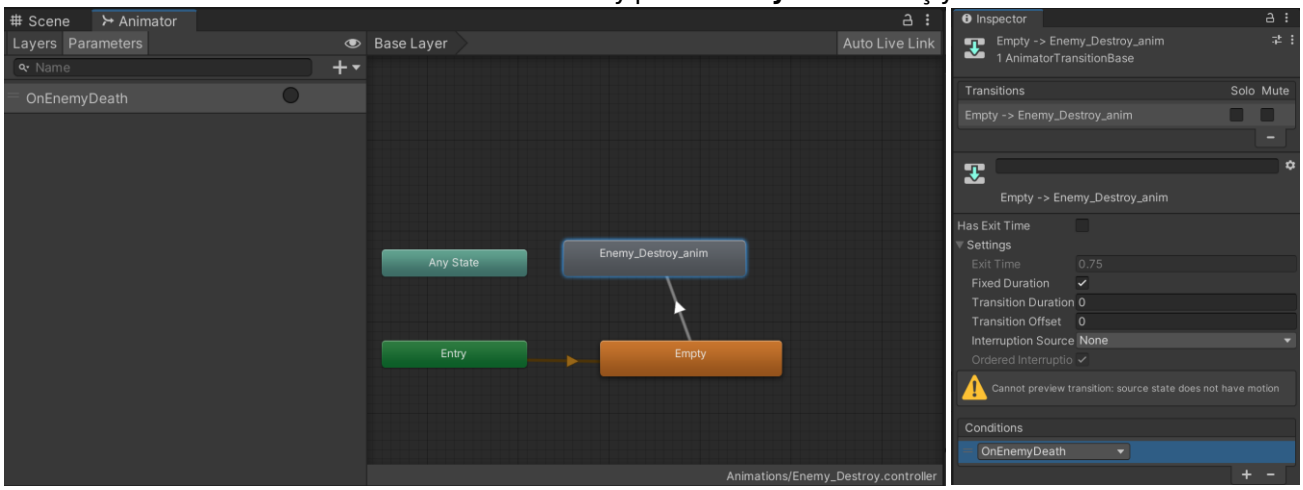
Oyun Programlama Ödevi – Hafta 9

Ad: Eren
Soyad: Köse
Numara: 22360859075
GitHub Kod Linki: https://github.com/erennkose/btu-oyun-programlama/tree/main/Hafta9

- **Düşman Patlama Animasyonu**
  Düşmana patlama animasyonu eklemek için öncelikle Enemy prefabimizi sürükleyip Hierarchye bırakıyoruz ve Animation sekmesine tıklayıp Create butonuna tıklıyoruz. Adını Enemy_destroy_anim yapıp animasyonumuzu oluşturmuş oluyoruz. Sonrasında ise Explosion klasöründe bulunan ilk görseli tıklayıp shift basıp animasyonun sona ereceği görsele tıklayıp oradaki tüm görselleri seçmiş oluyoruz. Sürükleyip animation kısmına bırakıyoruz. Ardından animasyonlarımızı kaydettiğimiz klasörde bulunan Enemy_destroy_anim'e tıklayıp **Loop Time** seçeneğini kapatıyoruz. Ardından oluşturduğumuz animasyona sahip olan nesneye tıklıyoruz ve Animator sekmesine geliyoruz. Buradaki ekranasağ tıklayıp **Create New State** diyerek yeni bir state oluşturuyoruz ve sağ tık atıp **Set As Default Layer State** diyoruz. Ardından yeni oluşturduğumuz Empty stateden Enemy_destroy_anim stateine, Empty state'e sağ tıklayıp **Make Transition** diyerek transition çekiyoruz. Ardından o transitiona (oka) tıklıyoruz ve **Has Exit Time** seçeneğini delay yaşanmaması için kapatıyoruz. Ardından Has Exit Time'ın altında bulunan Settings kısmını açıyoruz ve burada **Transition Duration** kısmını **0** yapıyoruz. Ardından Settingsin altında bulunan **Conditions** kısmındaki **+** butonuna tıklayıp **OnEnemyDeath'i** seçiyoruz.



Ardından bu patlamanın temas anında yapılmasını sağlamak için **Enemy_sc** scriptimizi kod editörümüzde açıyoruz. Burada patlamanın gerçekleşmesini sağlamak için öncelikle fonksiyonların üstünde **Animator** türünde anim adında bir değişken tanımlıyoruz. Ardından **Start** fonksiyonunda bu anim nesnesine gerekli atamayı yapıyoruz.

```
Animator anim;
```

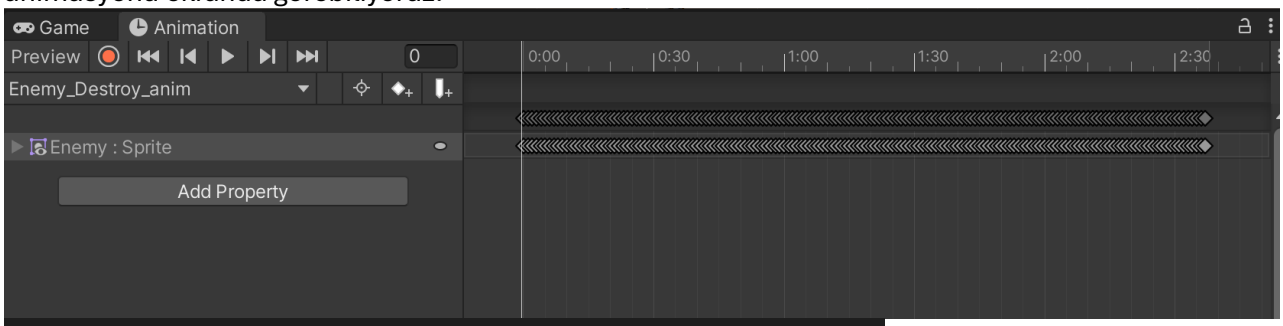
```
anim = GetComponent<Animator>();

if (anim == null){
    Debug.LogError("Enemy_sc::Start anim is null");
}
```

Atamaların ardından **OnTriggerEnter2D** fonksiyonunda Damage fonksiyonunun çağırılmasından sonra **anim.SetTrigger("OnEnemyDeath")** diyip animasyonumuzun başlamasını sağlıyoruz. Bunu hem Laser ile temas tespiti içerisinde, hem de Player ile temas tespiti içerisinde yapıyoruz. Çünkü ikisinde de Enemy nesnemizin patlama animasyonunu gerçekleştirmesini istiyoruz. Ardından Unity'e geri dönüp Hierarchy'de bulunan Enemy nesnemizde **Override** yaptırıyoruz. Bu sayede yaptığımız değişiklikler prefabimizde de geçerli oluyor. Bunun ardından ekrandan nesnemizi siliyoruz.

```
void OnTriggerEnter2D(Collider2D other){
    if(other.tag == "Player"){
        // Canini azalt
        Player_sc playersc = other.GetComponent<Player_sc>();
        playersc.Damage();
        anim.SetTrigger("OnEnemyDeath");
        speed = 0;
        Destroy(this.gameObject, 2.5f);
    }
    else if(other.tag == "Laser"){
        Destroy(other.gameObject);
        if (player_sc != null){
            player_sc.UpdateScore(10);
        }
        anim.SetTrigger("OnEnemyDeath");
        speed = 0;
        Destroy(this.gameObject, 2.5f);
    }
}
```

- **Nesne Yok Etmede Gecikme Eklenmesi**
  Animasyonumuzun ekranda göründükten sonra nesnemizin yok olmasını istiyoruz. Bunun için öncelikle animasyonumuzun ne kadar sürdüğünü hesaplıyoruz. Bizim burada kullandığımız animasyonumuzun süresi 2.5 saniye olduğundan dolayı kodda buna uygun değişiklikler yapacağız. Kod editörümüzde **Enemy_sc** scriptini açıyoruz. Burada **OnTriggerEnter2D** fonksiyonuna gidiyoruz ve burada scripte sahip nesneyi yok ettiğimiz kod parçası olan **Destroy** fonksiyonlarını buluyoruz. Buralarda parametre olarak yalnızca **this.gameObject** veriyorduk. Bu parametrenin yanına virgülle 2.5f yazıyoruz ve bu sayede Destroy işlemimiz 2.5 saniye gecikmeli olarak çalışıyor. Bu sayede nesne yok olmadan önce animasyonu ekranda görebiliyoruz.



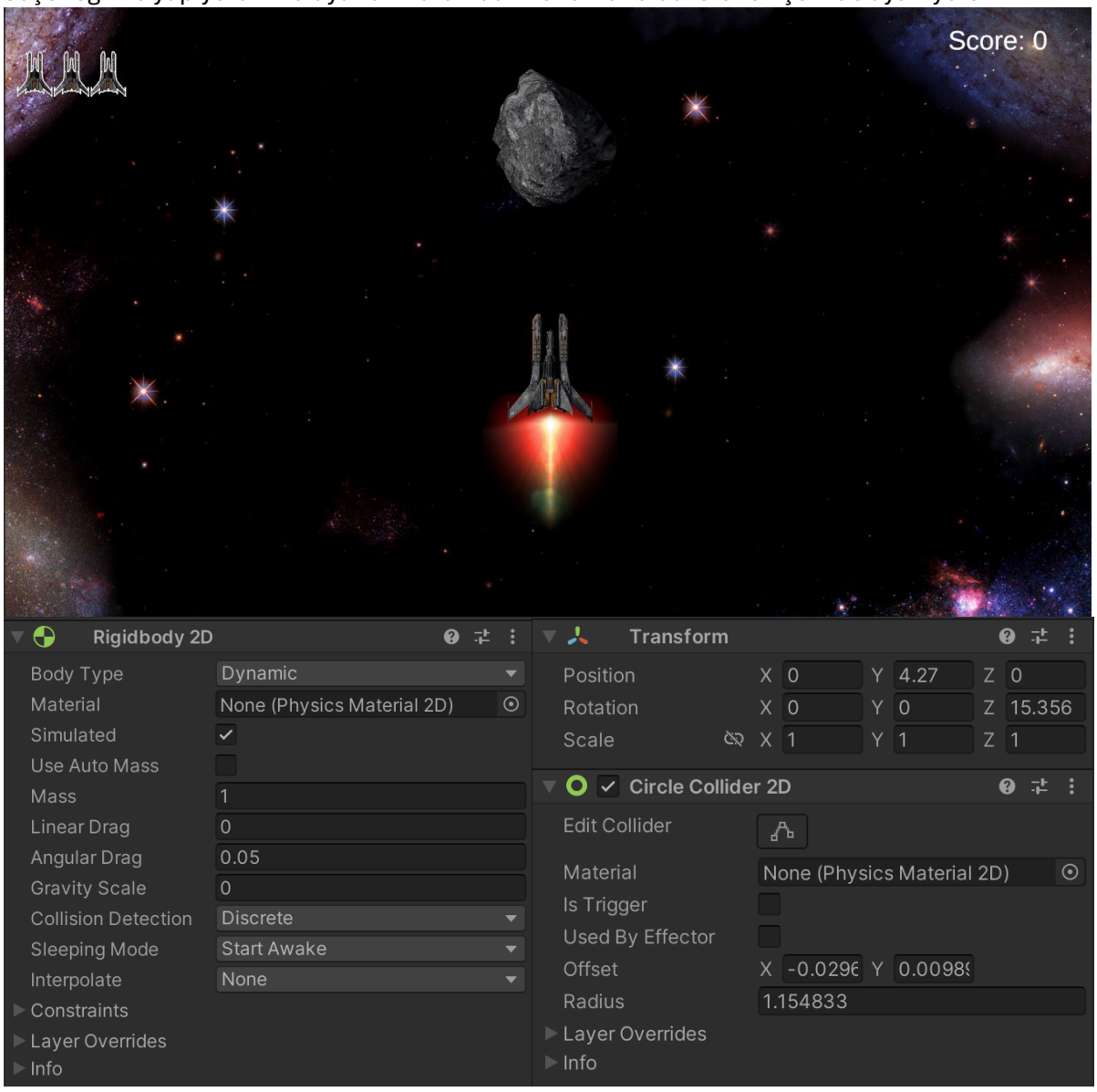

```
Destroy(this.gameObject, 2.5f);
```

- **Hız Sabitleme ile Yok Olan Düşman Gemisinin Zarar Vermesinin Önüne Geçilmesi**
  Enemy nesnemiz artık 2.5 saniye sonra yok edildiğinden dolayı yok edilene kadar animasyon çalışırken bu nesne bizim Player nesnemize çarpıp hasar verebilir. Bu bizim istemediğimiz bir durum. Bu yüzden Enemy nesnemizin yok edilmesi durumundan önce **Enemy** nesnemizin **speed** değerini **0** yapıyoruz. Bu sayede Enemy nesnemiz Laser ile veya Player ile temas ettiği yerde kalıyor.

```
speed = 0;
Destroy(this.gameObject, 2.5f);
```

- **Asteroid Ekleme**

  Asteroidimizi oyuna eklemek için **Sprites** klasörü altında bulunan **Asteroid** nesnesini sürükle-bırak ile **Hierarchy** kısmına bırakıyoruz. Ardından nesnemize **Inspector** kısmından **Add Component** diyerek **Circle Collider 2D** ve **Rigidbody 2D** özelliklerini ekliyoruz. Circle Collider'ımızın Collider sınırlarını **Edit Collider** diyerek istendiği şekilde düzenliyoruz. Sonrasında ise bu nesnemizin Rigidbody'deki yerçekimi sebebiyle yere düşmesini engellemek için Rigidbody 2D componentinin altındaki **Gravity Scale** seçeneğini **0** yapıyoruz. Bu ayarların ardından konumunu da istenen şekilde ayarlıyoruz.



- **Asteroid'in Z Ekseninde Sürekli Hareket Etmesi, Asteroid Patlama Animasyonu**

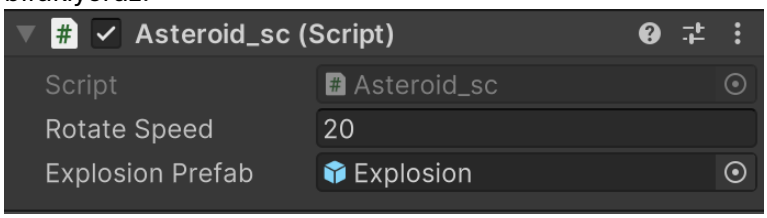
  Asteroidin Z ekseninde sürekli hareketini sağlamak için, öncelikle **Scripts** klasörü altında bir **Asteroid_sc** scripti oluşturuyoruz. Bu scripti kod editörümüzde açıyoruz. Burada **Start** fonksiyonunun üzerinde, dönme hızını belirlemek için **rotateSpeed** adında bir **float** değer oluşturup değerini 20.0f giriyoruz. Tanımlama sonrasında ise **Update** fonksiyonu içerisinde Z ekseninde sürekli hareketi sağlayacak kodu yazıyoruz. **transform.Rotate** fonksiyonu sayesinde bu sürekli hareketi sağlayacağız.

```
void Update()
{
    transform.Rotate(Vector3.forward * rotateSpeed * Time.deltaTime);
}
```

  Bu kod sayesinde asteroidimiz **bir saniyede rotateSpeed** hızında **Vector3.forward** yönünde sürekli dönecek.

  Asteroidimize patlama animasyonu eklemek için ise öncelikle **Sprites** klasörü altındaki **Explosion** klasöründe bulunan ilk patlama görselini sürükleyip **Hierarchy** kısmına bırakıyoruz. **Explosion** ismini verdikten sonra **Animation** sekmesine gelip **Create** butonuna tıklayarak Explosion nesnemiz için bir

animasyon oluşturulmasını sağlıyoruz. Ardından ise **Explosion klasöründe** bulunan tüm görselleri **shift** ile seçerek **Animation** sekmesine bırakıyoruz. Animasyonumuza (Explosion_anim) tıklıyoruz ve **Loop Time** seçeneğini kapatıyoruz. Bu sayede animasyonun sürekli değil bir kere çalışmasını sağlıyoruz. Sonrasında ise bu Explosion nesnemizi sürükleyip **Prefabs** klasörüne bırakıyoruz. Animasyonumuz hazır olduğuna göre, bu animasyonun Laser ile Asteroid teması sonrasında başlamasını sağlamalıyız. Bunun için Asteroid_sc scriptinde bir **GameObject** değişkeni oluşturuyoruz. Bu değişkeni **[SerializeField]** yapıp Unity ekranına geri dönüyoruz. Asteroid nesnemiz altındaki Script componentimizi buluyoruz ve buradaki Explosion Prefab içerisine Explosion prefabimizi sürükleyip bırakıyoruz.



Şimdi kodumuza geri dönüyoruz ve OnTriggerEnter2D fonksiyonu içerisinde temasın Laser nesnesiyle olması durumunda patlama animasyonunu oynatacak, Laser ve Asteroidi yok edecek olan kodu yazıyoruz.

```
void OnTriggerEnter2D(Collider2D other){
    if (other.tag == "Laser"){
        Instantiate(explosionPrefab, transform.position, Quaternion.identity);
        Destroy(other.gameObject);
        Destroy(this.gameObject);
    }
}
```

Bu sayede de asteroidimize **patlama animasyonu** eklenmiş oluyor.

- **<u>Spawn Routine'lerinin Asteroid Yok Edildikten 3 Saniye Sonra Başlaması</u>**

Bu işlemi sağlamak için öncelikle Asteroid_sc scriptine gidiyoruz. Burada SpawnManager_sc scripti tipinde bir değişken oluşturuyoruz.

```
SpawnManager_sc spawnManager_sc;
```

Ardından Start içerisinde **GameObject.Find** kullanarak **Spawn_Manager** nesnesini buraya atıyoruz.

```
void Start()
{
    spawnManager_sc = GameObject.Find("Spawn_Manager").GetComponent<SpawnManager_sc>();
    if (spawnManager_sc == null){
        Debug.LogError("Asteroid_sc::Start spawnManager_sc is null");
    }
}
```

Atama işleminden sonra ise SpawnManager_sc scriptine gidiyoruz ve Start içerisinde yapmakta olduğumuz iki Coroutine başlatma işlemlerini public olan **StartSpawning** adında bir fonksiyon açıp onun içerisinde yaptırıyoruz. Bu coroutine fonksiyonların 3 saniye sonrasında çalışması için ise fonksiyonların içine ilk satırlarına **yield return new WaitForSeconds(3.0f);** kod parçasını ekliyoruz.

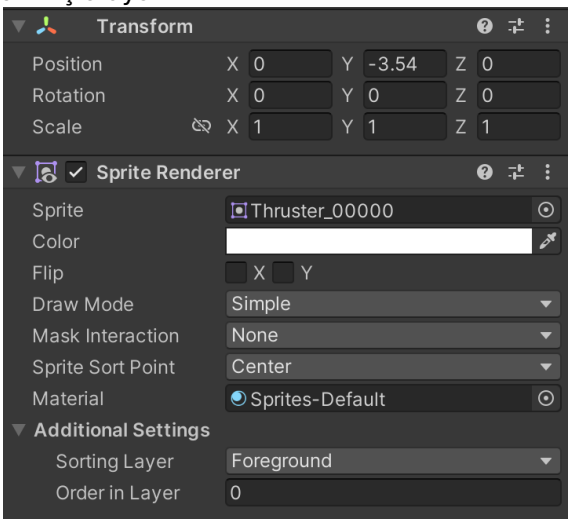
```
public void StartSpawning()
{
    StartCoroutine(SpawnEnemyRoutine());
    StartCoroutine(SpawnBonusRoutine());
}
IEnumerator SpawnEnemyRoutine()
{
    yield return new WaitForSeconds(3.0f);
    while(stopSpawning == false)
    {
        Vector3 position = new Vector3(Random.Range(9.4f,-9.4f), 7.4f, 0);
        GameObject new_enemy = Instantiate(enemyPrefab, position, Quaternion.identity);
        new_enemy.transform.parent = enemyContainer.transform;
        yield return new WaitForSeconds(5.0f);
    }
}
1 başvuru
IEnumerator SpawnBonusRoutine(){
    yield return new WaitForSeconds(3.0f);
    while(stopSpawning == false)
    {
        int randNum = Random.Range(0,3);
        Vector3 position = new Vector3(Random.Range(9.4f,-9.4f), 7.4f, 0);
        Instantiate(bonusPrefabs[randNum], position, Quaternion.identity);
        yield return new WaitForSeconds(10.0f);
    }
}
```

Sonrasında ise Asteroid_sc scriptimizdeki **OnTriggerEnter2D** fonksiyonunda Laser ile çarpışmayı kontrol ettiğimiz koşulun altındaki Destroylar arasında **StartSpawning** fonksiyonunu çağırıyoruz. Bu sayede Spawn işlemlerimiz asteroid nesnesi yok edildikten **3 saniye sonra** başlamış oluyor.

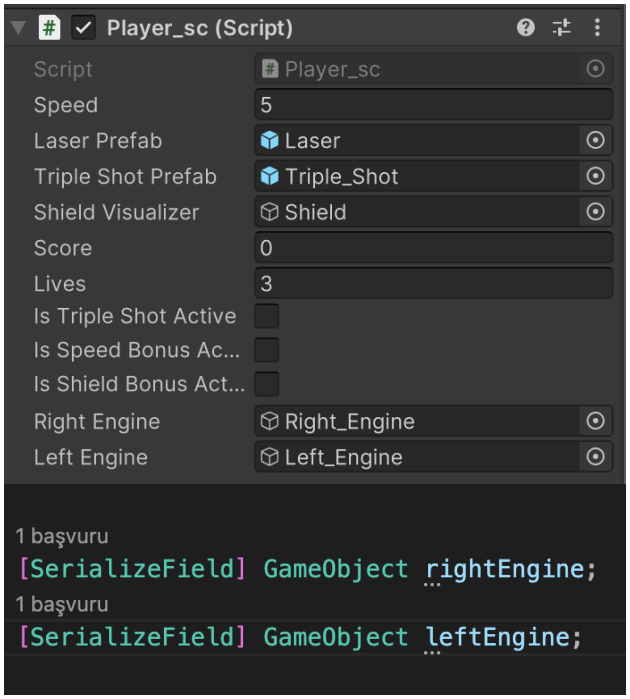
```
void OnTriggerEnter2D(Collider2D other){
    if (other.tag == "Laser"){
        Instantiate(explosionPrefab, transform.position, Quaternion.identity);
        Destroy(other.gameObject);
        spawnManager_sc.StartSpawning();
        Destroy(this.gameObject);
    }
}
```

- **<u>Thruster Ekleme, Sağ ve Sol Motor İçin Hasar Animasyonlarının Eklenmesi</u>**

Thruster eklemek için öncelikle **Sprites** klasörü altındaki **Thruster** klasöründeki ilk görseli sürükleyip, **Hierarchy** kısmındaki Player nesnemizin üzerine bırakıyoruz. Bu sayede Thruster görselimiz, Player nesnemizi childe'ı olarak atanmış oluyor. Ardından Player nesnemizin arkasına gelecek şekilde **konumunu** ayarlıyoruz. Sorting Layer'ını ise **Foreground** yapıyoruz. Sonrasında ise Thruster nesnemiz seçiliyken **Animation** sekmesini açıp **Create** diyerek Thruster_anim adında bir animasyon oluşturuyoruz. Buraya **Sprites > Thruster** klasörü altındaki görselleri shift ile seçip bırakıyoruz. Bu sayede Thruster ve animasyonu eklenmiş oluyor.



Sol ve sağ motor için hasar animasyonları için ise, öncelikle Sprites > Player_Hurt klasörü altındaki ilk Fire_00000 isimli görseli sürükleyerek Hierarchy kısmındaki Player nesnesinin üstüne bırakıyoruz. Adını Left_Engine yapıyoruz ve sol kanada denk gelecek şekilde konumunu ayarlıyoruz. Sorting Layer'ını Foreground yapıyoruz ve Order in Layer'ını 1 yapıyoruz. Ardından Animation sekmesine gelip Create butonuna basıyoruz (Left_Engine seçiliyken). Adını Engine_Failure_anim yapıyoruz. Sonrasında Sprites > Player_Hurt içerisindeki görselleri shift ile seçip Animation sekmesine bırakıyoruz. Aynı şeyleri yaparak adı Right_Engine olan bir nesne daha oluşturuyoruz, onun konumunu sağ motora denk getiriyoruz ve yeni animasyon oluşturmak yerine Engine_Failure_anim adlı animasyonu kullanıyoruz. Bunların sonrasında ise bu motorların hasar animasyonları canımız azaldıkça görüneceğinden oyun başlayınca görünmesini engellememiz lazım. Bunun için öncelikle her iki Engine için de Inspector kısmında adlarının yanındaki tiki kaldırıyoruz. Sonrasında ise Player_sc scriptimizde canımızı azalttığımız kısım olan Damage fonksiyonuna gidiyoruz. Burada canımızın 2 değerine eşitlenmesi durumunda sol motorun, 1 değerine eşitlenmesi durumunda ise sağ motorun görünmesini aktive edeceğiz. Bunun için öncelikle leftEngine ve rightEngine adında iki adet [SerializeField] olan GameObject değişkenleri oluşturuyoruz. Bunları Unity ekranına dönüp sürükle-bırak ile tanımlıyoruz. Kodda ise koşulları kullanarak SetActive fonksiyonu ile aktifleştiriyoruz. Bu sayede de motorlarımızın hasar animasyonunu ayarlamış oluyoruz.




```
public void Damage(){
    if (isShieldBonusActive){
        isShieldBonusActive = false;
        shieldVisualizer.SetActive(false);
        return;
    }
    else {
        lives--;
        if (lives == 2){
            leftEngine.SetActive(true);
        }
        else if (lives == 1){
            rightEngine.SetActive(true);
        }
```