Orijinal Kaynak: http://forum.unity3d.com/threads/unity-lesson-1-draft.103421/

Unity 3D İle Oyun Programlama

Bölüm 17: Oyuna Roketatar Eklemek

ÇEVİRİ: Süleyman Yasir KULA

http://yasirkula.com

Ferdinand Joseph Fernandez

Chief Technological Officer, Dreamlords Digital Inc.

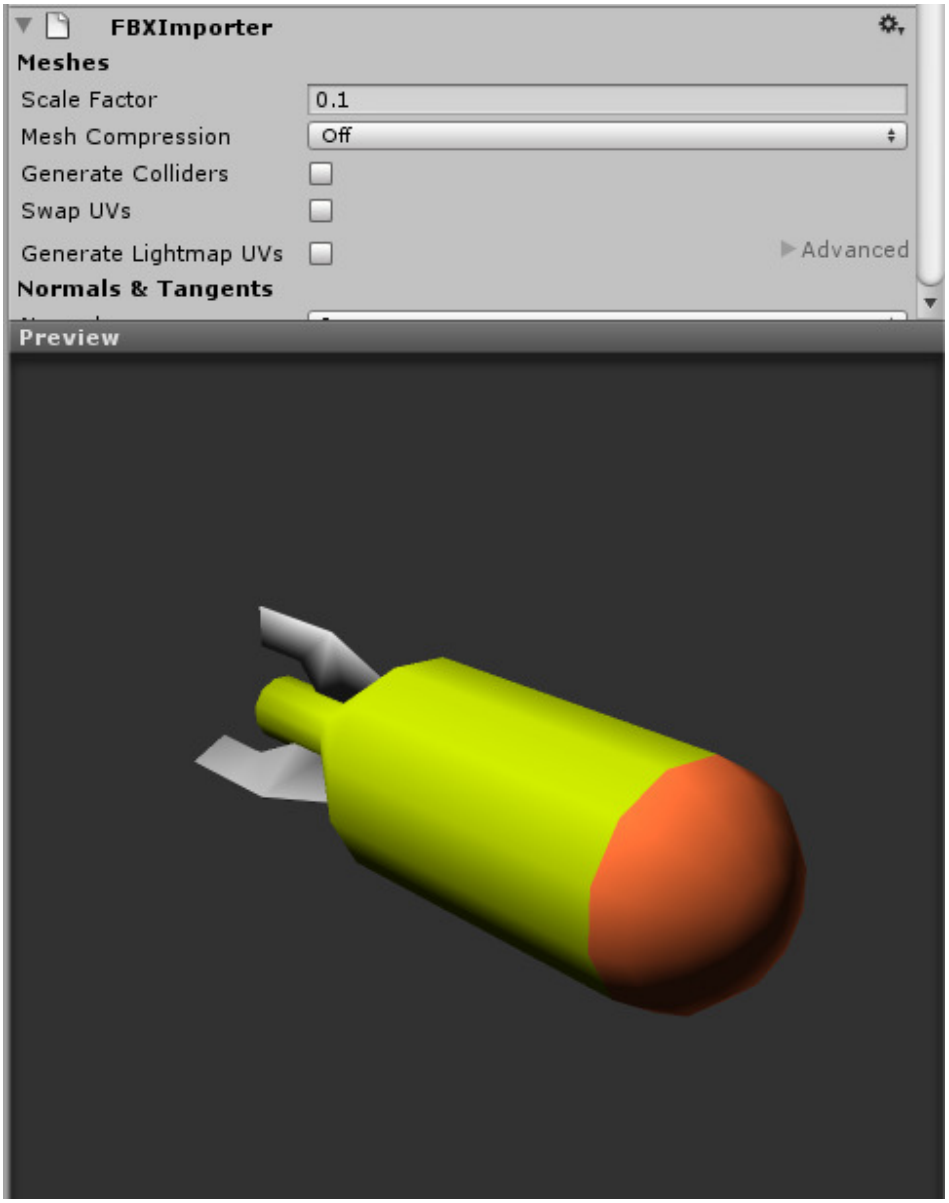
Admin and Co-founder, Unity Philippines Users Group

September 2011

Karakterin taramalı silahından bıktınız mı? O zaman müjdemi isterim: bu bölümde oyuna roketatar ekliyoruz!

Roketi Oluşturmak

Winrar arşivinin oradaki "MinimalMissile" klasöründe yer alan “MinimalMissile1.fbx” modelini projenize import edip ardından bu roket modelini sahneye sürükleyin. "Scale Factor"ü 0.1 yaparak modelin yeterince büyük olmasını sağlayın.



Roketi hareket ettirmek için bir script yazalım. “Missile” adında yeni bir C# scripti oluşturun:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Missile : MonoBehaviour
05 {
06     [SerializeField]
07     float _speed = 50.0f;
08
09     void Start()
10     {
11         rigidbody.AddRelativeForce(new Vector3(0, 0, _speed), ForceMode.VelocityChange);
12     }
13 }
```

Bu scriptte rokete, baktığı yönde hareket etmesi için güç uyguluyoruz.

Scripti MinimalMissile1 objesine verin. Ayrıca bir de rigidbody verin (bence roketin **Rigidbody**'sindeki "**Use Gravity**"i kapatarak yerçekiminin rokete etki etmesini engelleyin).

Oyunu çalıştırınca roket ileri yönde hareket etmeli.

Roketatarı Ateşlemek

Roketatar scripti için RifleWeapon scriptini referans olarak kullanacağız.

RifleWeapon scriptini seçin Ctrl + D kombinasyonu ile klonlayın.

Klonlanan scriptin adını “MissileWeapon” yapın. Ardından scripti açıp **önce class ismini değiştirin**, ardından şu değişiklikleri yapın:

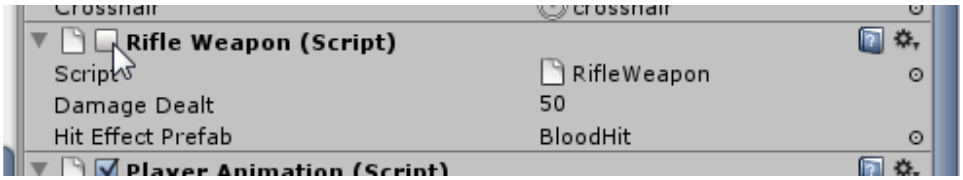
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MissileWeapon : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _missilePrefab;
08
09     [SerializeField]
10     Transform _muzzle;
11
12     void Start()
13     {
14         Screen.lockCursor = true;
15     }
16
17     void Update()
18     {
19         if (Input.GetKey(KeyCode.Escape))
20         {
21             Screen.lockCursor = false;
22         }
23
24         if (Input.GetButtonDown("Fire1"))
```

```
25          {
26              Screen.lockCursor = true;
27
28              Ray mouseRay = Camera.main.ViewportPointToRay(new  Vector3(0.5f, 0.5f, 0));
29              RaycastHit hitInfo;
30
31              if  (Physics.Raycast(mouseRay, out hitInfo))
32              {
33                  Vector3 direction = hitInfo.point - _muzzle.position;
34                  direction.Normalize();
35
36                  GameObject m = Instantiate(_missilePrefab, _muzzle.position, _muzzle.rotation) as GameObject;
37                  m.transform.forward = direction;
38              }
39          }
40      }
41 }
```

Eskiden burada düşmanın Health scriptindeki Damage fonksiyonunu kullanarak düşmana hasar veriyorduk. Şimdiyse "_muzzle" isimli değişkende depolanan objenin olduğu konumda yeni bir roket oluşturuyoruz ve ardından roketin doğru yöne doğru bakmasını "transform.forward = direction;" ile sağlıyoruz. Bu kod roketin "ileri" yönünün (mavi ok) "direction" yönünde olmasını sağlar.

Kodu test etmek için yapmamız gereken ayarlamalar var. Önce MinimalMissile1 objesini Project panelindeki Prefabs klasörüne sürükleyerek bir prefab yapın.

MissileWeapon scriptini Player objesine verin. RifleWeapon scriptini şu an için inaktif hale getirin (isminin solundaki işareti kaldırarak):



Görsel 17.1: RifleWeapon scripti inaktif (disabled) oldu.

Bu işlem RifleWeapon scriptinin çalışmasını engelleyecek ve böylece roketatarı daha rahat test edebileceğiz.

Player'ın "Missile Weapon" component'indeki “Missile Prefab”a "MinimalMissile1" prefab'ını değer olarak verin.

Ardından yeni bir empty gameobject oluşturup onu “Muzzle” olarak isimlendirin. Roketler "Muzzle"ın üzerinden çıkacak. O halde Muzzle objesini Player'ın silahının ucuna yerleştirmeli ve silahla beraber hareket etmesi için onu silahın bir child objesi yapmalıyız.

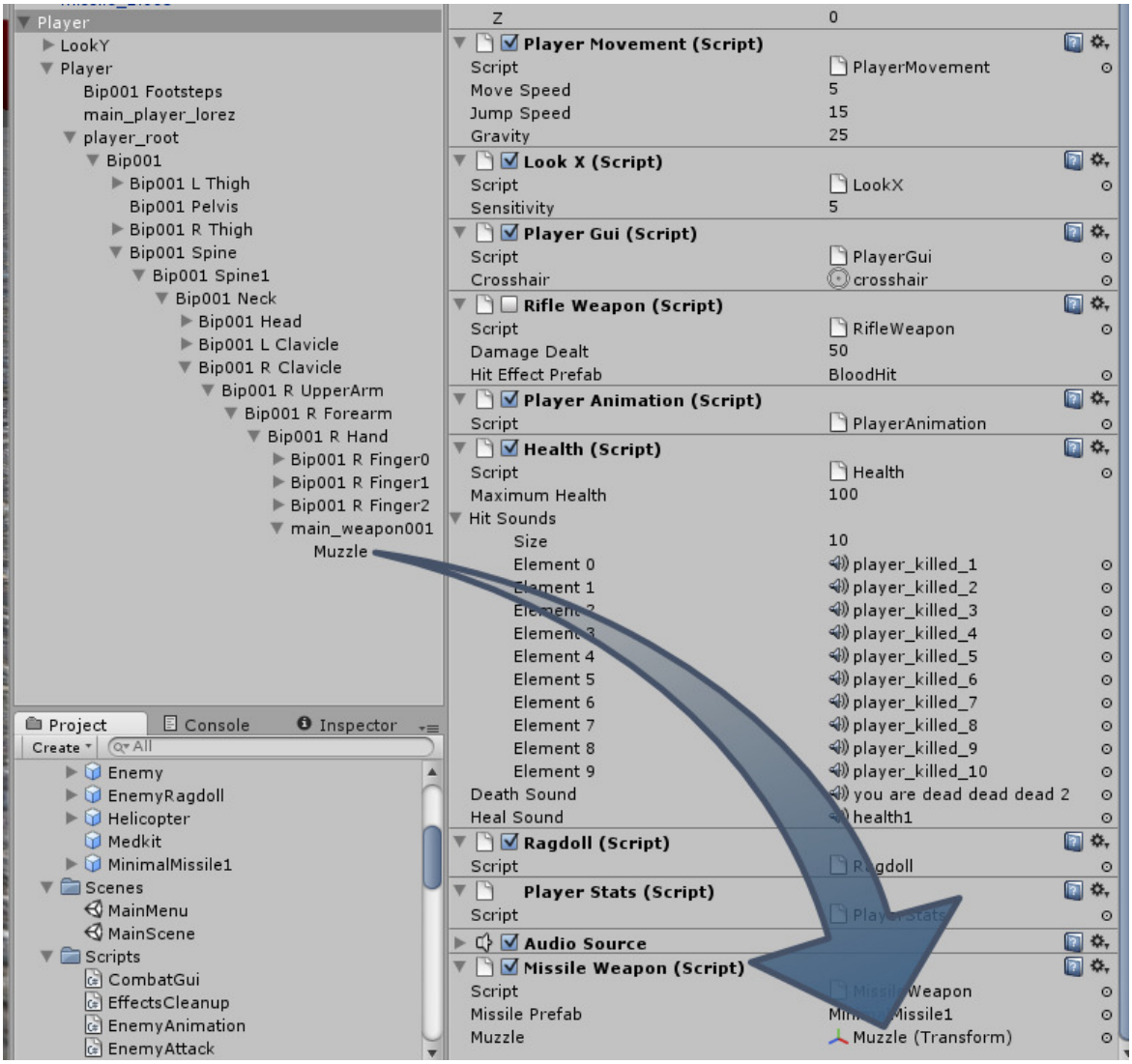
Player objesini seçin. Ardından player'ın silahına tıklayın. Silah seçili hale gelecek (Hierarchy panelinde “main_weapon001” seçilmiş olacak). Muzzle objesini bu objenin child'ı yapacağız.

Görsel 17.2: Player'ın silahı seçili halde

O halde hiç durmayın ve Muzzle objesini Hierarchy'den “main_weapon001”in üzerine sürükleyerek onun bir child objesi yapın. Ardından Muzzle'ı silahın namlusunun ucunda bir yere taşıyın.

Muzzle'ın "forward" (ileri) yönünün (Mavi okun) namludan dışarı yönde olduğundan emin olun. Roket bu yönde ateşlenecek. Muzzle'ın rotation'ını (270, 0, 0) yapmak bende iş gördü.

Artık Muzzle doğru konumda olduğuna göre onu, Player'ın MissileWeapon scriptindeki "Muzzle" değişkenine onu değer olarak atayın:



Oyunu çalıştırın. Bir zombiye nişan alıp ateş edin. Herşeyi düzgün yaptıysanız silahınızdan roket çıkmalı (Artık sahnedeki roket objesini silebilirsiniz).

Boşluğa Bile Nişan Alsak Roket Ateşlenmesi

Şu anda boşluğa nişan alınca roket atamıyoruz. Çünkü kodumuzda roketi ancak Raycast bir yere temas ederse oluşturuyoruz.

MissileWeapon scriptini güncelleyin:

```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  MissileWeapon : MonoBehaviour
05 {
06         [SerializeField]
07         GameObject _missilePrefab;
08
```

```
09      [SerializeField]
10      Transform _muzzle;
11
12      void Start()
13      {
14          Screen.lockCursor = true;
15      }
16
17      void Update()
18      {
19          if (Input.GetKey(KeyCode.Escape))
20          {
21              Screen.lockCursor = false;
22          }
23
24          if (Input.GetButtonDown("Fire1"))
25          {
26              Screen.lockCursor = true;
27
28              Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
29              RaycastHit hitInfo;
30
31              Vector3 direction;
32
33              if (Physics.Raycast(mouseRay, out hitInfo))
34              {
35                  direction = hitInfo.point - _muzzle.position;
36                  direction.Normalize();
37              }
38              else
39              {
40                  direction = Camera.main.ViewportToWorldPoint(new Vector3(0.5f, 0.5f, 50)) - _muzzle.position;
41                  direction.Normalize();
42              }
43
44              GameObject m = Instantiate(_missilePrefab, _muzzle.position, _muzzle.rotation) as GameObject;
45              m.transform.forward = direction;
46          }
47      }
48  }
```

Yaptığımız değişiklik çok basit: eğer boşluğa nişan alıyorsak kameranın orta noktasından ileri yönde 50 birim uzaktaki noktayı hedef noktamız olarak ele alıyoruz ve füzenin yönünü ona göre belirliyoruz.

Roketin Hasar Vermesi

Şu anda roket temas ettiği objelerin içinden geçiyor. Ama biz bunu değiştirerek roketin patlamasını sağlayalım!

Missile scriptine şu fonksiyonu ekleyin:

```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  Missile : MonoBehaviour
05 {
06        [SerializeField]
07        float  _speed = 10.0f;
08
09        void  Start()
10        {
11              rigidbody.AddRelativeForce(new  Vector3(0, 0, _speed), ForceMode.VelocityChange);
12        }
13
14        void  OnCollisionEnter(Collision c)
15        {
16              Destroy(gameObject);
17        }
18 }
```

Füze birşeye temas ederse füzeyi yokediyoruz.

Kodun çalışması için füzeye "Box Collider" verin (bunun için sahneye bir tane roket klonu koyun, ona "**Box Collider**" verip collider'ın boyutunu uygun şekilde ayarlayın (**Center** (0, 0, -0.25) ve **Size** (0.2, 0.2, 0.5) iken güzel durdu bende). Sonrasında **Inspector**'dan "**Apply**" deyin ve sahnedeki roketi silin).

Artık düşmana hasar vermeye hazırız. Temas ettiğimiz objede Health scripti var mı diye bakalım ve eğer varsa o scriptin Damage fonksiyonunu çağıralım:

```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  Missile : MonoBehaviour
05 {
06       [SerializeField]
07       float  _speed = 50.0f;
08
09       [SerializeField]
10       int  _damageDealt = 100;
11
12       void  Start()
13       {
14             rigidbody.AddRelativeForce(new  Vector3(0, 0, _speed), ForceMode.VelocityChange);
15       }
16
17       void  OnCollisionEnter(Collision c)
18       {
19             Health h = c.transform.root.GetComponent<Health>();
20             if  (h != null)
21             {
22                   h.Damage(_damageDealt);
23             }
24
25             Destroy(gameObject);
26       }
27 }
```

Roket bundan böyle çarptığı düşmanlara hasar verecek.

Roketin Player'a Hasar Vermesini Engellemek

Bazen kendi silahınızdan çıkan roket ateşlenir ateşlenmez sizi öldürüyor. Çünkü füzenin collider'ı bazen player'ın bir collider'ı ile temas edebiliyor.

Missile scriptini şu şekilde güncelleyerek bu durumu önleyelim:

```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  Missile : MonoBehaviour
05 {
06       [SerializeField]
07       float  _speed = 50.0f;
08
09       [SerializeField]
10       int  _damageDealt = 100;
11
12       void  Start()
13       {
14             rigidbody.AddRelativeForce(new  Vector3(0, 0, _speed), ForceMode.VelocityChange);
15       }
16
```

```
17      void OnCollisionEnter(Collision c)
18      {
19          if (c.transform.root.tag == "Player")
20          {
21              return;
22          }
23
24          Health h = c.transform.root.GetComponent<Health>();
25          if (h != null)
26          {
27              h.Damage(_damageDealt);
28          }
29
30          Destroy(gameObject);
31      }
32 }
```

Eğer roket Player'a temas etmişse "return;" komutunu kullanarak fonksiyonun geri kalanının çalıştırılmasını önlüyoruz.

Aynı Kodu Defalarca Kullanmaya Çözüm

Şu anda RifleWeapon ve MissileWeapon scriptlerinin büyük kısmı birbiriyle tamamen aynı. Aynı kodu iki scriptte tekrar tekrar yazmak yerine parent bir class (script) oluşturabiliriz ve iki scriptteki ortak olan kodu bu parent scripte yazabiliriz.

"Weapon" adında yeni bir C# scripti oluşturun. Bu scripte, RifleWeapon ve MissileWeapon scriptlerinin sahip olduğu ortak kodu yazacağız. Ardından RifleWeapon ve MissileWeapon'ın Weapon scriptinden türemelerini (onun child scripti olmalarını) sağlayacağız. (Konuyla ilgili bilginiz yoksa parent-child class'ın (inheritance diye geçer) ne olduğunu araştırın ve şu videoyu inceleyin: http://unity3d.com/learn/tutorials/modules/intermediate/scripting/inheritance)

Parent class kullanmaktaki amaç birden çok scriptte aynı olan bir kodu tek bir class'a yazmak ve diğer scriptlerde bu ortak kodun kalabalık yapmasını önlemektir (daha başka faydaları da var, bkz. polymorphism).

İki silahın da ateş edebilmesi onların ortak bir özelliği. Ama iki silah bu ateş etme eylemini kendilerine has bir şekilde gerçekleştiriyor. Weapon scriptini yazarken buna dikkat edeceğiz. Weapon class'ında ateş etme fonksiyonu olacak ama fonksiyonun içerisi boş olacak. Bu fonksiyonu RifleWeapon ve MissileWeapon scriptlerinde ayrı ayrı dolduracağız.

Weapon scripti MissileWeapon scriptine çok benziyor. Arada bazı ufak farklılıklar var sadece:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Weapon : MonoBehaviour
05 {
06      [SerializeField]
07      protected Transform _muzzle;
08
09      void Start()
10      {
11          Screen.lockCursor = true;
```

```
12      }
13
14      void Update()
15      {
16          if (Input.GetKey(KeyCode.Escape))
17          {
18              Screen.lockCursor = false;
19          }
20
21          if (Input.GetButtonDown("Fire1"))
22          {
23              Screen.lockCursor = true;
24
25              Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
26              RaycastHit hitInfo;
27
28              Vector3 direction;
29
30              if (Physics.Raycast(mouseRay, out hitInfo))
31              {
32                  direction = hitInfo.point - _muzzle.position;
33                  direction.Normalize();
34              }
35              else
36              {
37                  direction = Camera.main.ViewportToWorldPoint(new Vector3(0.5f, 0.5f, 50)) - _muzzle.position;
38                  direction.Normalize();
39              }
40
41              Shoot(hitInfo, direction);
42          }
43      }
44
45      virtual protected void Shoot(RaycastHit hitInfo, Vector3 direction)
46      {
47      }
48 }
```

Artık Shoot adında "virtual" bir fonksiyonumuz var. Bir fonksiyonun "virtual" olması, o fonksiyonun child scriptler tarafından "override" edilebilmesini sağlar (yani child scriptlerin o fonksiyonun yapacağı şeyleri kendilerine göre düzenleyebilmelerini sağlar).

"_muzzle" değişkenini "protected" yapıyoruz. "protected" değişkenlere child class'lar tarafından erişilebilir ama "private" değişkenlere child class'lar tarafından erişilemez. Biz "_muzzle"ı MissileWeapon scriptinde kullanacağımızdan dolayı onu "protected" yaptık.

MissileWeapon scriptini şöyle güncelleyin (4. satıra özellikle dikkat edin):

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MissileWeapon : Weapon
05 {
06      [SerializeField]
07      GameObject _missilePrefab;
08
09      override protected void Shoot(RaycastHit hitInfo, Vector3 direction)
10      {
```

```
            GameObject m = Instantiate(_missilePrefab, _muzzle.position, _muzzle.rotation) as GameObject;
            m.transform.forward = direction;
        }
}
```

Artık MissileWeapon scripti Weapon scriptinin bir child'ı oldu. Weapon scripti ise MonoBehaviour'ın child class'ıydı. Haliyle MissileWeapon otomatik olarak MonoBehaviour'ın da bir child'ı vaziyetinde.

Unutmayın, bir script (class) başka bir class'ın child'ı ise sanki parent class'taki kodlar child class'ta yazılıymış da görünmez vaziyetteymiş gibi bir durum olur. Yani MissileWeapon scriptimizin her ne kadar göremesek de Start ve Update fonksiyonları mevcut (bu fonksiyonları parent'ı olan Weapon scriptinden alıyor).

Weapon class'ı ateş ederken Shoot fonksiyonunu kullanıyor. MissileWeapon scriptinde Shoot fonksiyonunu "override" ediyoruz, yani bu child scriptte Shoot fonksiyonunun farklı davranmasını sağlıyoruz. Böylece ateş edince Weapon scriptindeki boş Shoot fonksiyonu değil MissileWeapon scriptindeki dopdolu Shoot fonksiyonu çağrılıyor.

Şimdi sıra geldi RifleWeapon'u da Weapon class'ının child'ı yapmaya:

```
using UnityEngine;
using System.Collections;

public class RifleWeapon : Weapon
{
    [SerializeField]
    int _damageDealt = 10;

    [SerializeField]
    GameObject _hitEffectPrefab;

    override protected void Shoot(RaycastHit hitInfo, Vector3 direction)
    {
        if (hitInfo.transform == null)
        {
            return;
        }

        Health enemyHealth = hitInfo.transform.GetComponent<Health>();
        if (enemyHealth != null)
        {
            enemyHealth.Damage(_damageDealt);

            Vector3 hitEffectPosition = hitInfo.point;
            Quaternion hitEffectRotation = Quaternion.FromToRotation(Vector3.forward, hitInfo.normal);
            Instantiate(_hitEffectPrefab, hitEffectPosition, hitEffectRotation);
        }
    }
}
```

RifleWeapon scripti de Shoot fonksiyonunu "override" ederek kendine has bir şekilde çalıştırıyor. Eğer boşluğa ateş etmişsek "hitInfo.transform == null" ifadesi "true" döndürüyor ve kodun geri kalanını çalıştırmıyoruz. Düşmana ateş etmişsek de ona eskiden olduğu gibi hasar veriyoruz.

Weapon class'ı RifleWeapon ile MissileWeapon'un sahip olduğu ortak kodları depoluyor. Bu iki script sadece kendilerini has fonksiyonu override ediyor: Shoot. Birisi roket atarken öbürü mermi yağdırıyor. Farkettiyseniz "inheritance" sayesinde RifleWeapon ile MissileWeapon scriptleri oldukça rahatladı.

Bundan böyle her iki silahta da olmasını istediğimiz ortak şeyleri sadece Weapon class'ına yazmamız yeterli olacak. Ne kadar da harika birşey!

NOT: Weapon scriptinde "_muzzle" değişkeni olduğu için artık Player'ın "Rifle Weapon" component'inde de "Muzzle" adında bir değişken var. Buna değer olarak MissileWeapon'a verdiğiniz Muzzle objesini verin.

Player'ın Arkasına Doğru Ateş Etmesini Önlemek

Nişangahımızın ucundaki zombi arkamızdaysa ona yine de ateş edebiliyoruz. Çünkü Weapon scriptimiz ateş ederken raycast kullanıyor ve bu raycast silahın namlusundan değil kameradan çıkıyor.

Yapmamız gereken, düşmanın Player'ın arkasında olup olmadığını anlamak. Bunun için Weapon scriptini şu şekilde güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Weapon : MonoBehaviour
05 {
06     [SerializeField]
07     protected Transform _muzzle;
08
09     void Start()
10     {
11         Screen.lockCursor = true;
12     }
13
14     void Update()
15     {
16         if (Input.GetKey(KeyCode.Escape))
17         {
18             Screen.lockCursor = false;
19         }
20
21         if (Input.GetButtonDown("Fire1"))
22         {
23             Screen.lockCursor = true;
24
25             Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
26             RaycastHit hitInfo;
27
28             Vector3 direction;
29             bool foundTarget = false;
30
31             if (Physics.Raycast(mouseRay, out hitInfo) && (Camera.main.WorldToScreenPoint(hitInfo.point).z >=
                   Camera.main.WorldToScreenPoint(_muzzle.position).z))
32             {
33                 foundTarget = true;
34                 direction = hitInfo.point - _muzzle.position;
35                 direction.Normalize();
36             }
37             else
```

```
38                  {
39                      direction = Camera.main.ViewportToWorldPoint(new  Vector3(0.5f, 0.5f, 50)) - _muzzle.position;
40                      direction.Normalize();
41                  }
42
43                  RaycastHit gunHitInfo;
44                  if  (Physics.Raycast(_muzzle.position, direction, out gunHitInfo))
45                  {
46                      foundTarget = true;
47                  }
48
49                  Shoot(foundTarget, gunHitInfo, direction);
50              }
51      }
52
53      virtual protected  void  Shoot(bool  foundTarget, RaycastHit hitInfo, Vector3 direction)
54      {
55      }
56 }
```

"foundTarget" adında bir değişken oluşturduk. Bu değişken önümüzde bir düşman olup olmadığını belirliyor. Varsayılan değeri "false".

31. satırda Camera class'ının WorldToScreenPoint fonksiyonunu kullanıyoruz. Bu fonksiyon 3 boyutlu uzaydaki bir noktanın (vector3), ekran koordinatlarında hangi pixele denk geldiğini döndürür. Döndürülen Vector3'ün x ve y değerleri pixelin konumunu ifade ederken z değeri ise o noktanın kameradan kaç birim uzakta olduğunu döndürür.

Eğer düşman kameraya player'dan daha yakınsa düşman arkamızda demektir. Yok düşman daha uzaksa o zaman düşman önümüzde demektir ve bu durumda "if" koşulunun içine giriyoruz.

43. satırda bir raycast daha yapıyoruz. Bu raycast'i yapma amacımız şu: eğer nişangahımızın ucundaki düşman arkamızdaysa ama aynı zamanda önümüzde de bir düşman varsa önümüzdeki düşmana ateş edebilmek. Eğer bu ikinci raycast'i yapmazsak ve nişangahımızın ucundaki düşman arkamızdaysa "foundTarget" false olacak ve silahımız ateş etmeyecekti.

Şimdi RifleWeapon ve MissileWeapon scriptlerini uygun şekilde güncelleyelim.

RifleWeapon:

```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  RifleWeapon : Weapon
05 {
06      [SerializeField]
07      int  _damageDealt = 10;
08
09      [SerializeField]
10      GameObject _hitEffectPrefab;
11
12      override protected  void  Shoot(bool  foundTarget, RaycastHit hitInfo, Vector3 direction)
13      {
14          if  (!foundTarget)
15          {
16              return;
17          }
18
```

```
            Health enemyHealth = hitInfo.transform.GetComponent<Health>();
            if (enemyHealth != null)
            {
                enemyHealth.Damage(_damageDealt);

                Vector3 hitEffectPosition = hitInfo.point;
                Quaternion hitEffectRotation = Quaternion.FromToRotation(Vector3.forward, hitInfo.normal);
                Instantiate(_hitEffectPrefab, hitEffectPosition, hitEffectRotation);
            }
        }
}
```

Çok basit: eğer namlumuzun ucunda düşman yoksa ateş etmiyoruz.

MissileWeapon:

```
using UnityEngine;
using System.Collections;

public class MissileWeapon : Weapon
{
    [SerializeField]
    GameObject _missilePrefab;

    override protected void Shoot(bool foundTarget, RaycastHit hitInfo, Vector3 direction)
    {
        GameObject m = Instantiate(_missilePrefab, _muzzle.position, _muzzle.rotation) as GameObject;
        m.transform.forward = direction;
    }
}
```

Silahın Sürekli Ateş Etmesi

Silahlarımız, sol mouse tuşuna basılı tuttuğumuz sürece belli bir aralıkla ateş etse güzel olmaz mı? Tıpkı piyasadaki FPS oyunlarındaki gibi olur. Düşmanı nasıl belli aralıklarla spawn ettiysek silahı da belli bir aralıkla ateşleyelim.

Weapon scriptini güncelleyin:

```
using UnityEngine;
using System.Collections;

public class Weapon : MonoBehaviour
{
    [SerializeField]
    protected Transform _muzzle;

    [SerializeField]
    float _fireDelay = 0.3f;

    float _nextFireTimeAllowed = -1.0f;

    void Start()
    {
        Screen.lockCursor = true;
```

```
17      }
18
19      void Update()
20      {
21          if (Input.GetKey(KeyCode.Escape))
22          {
23              Screen.lockCursor = false;
24          }
25
26          if (Input.GetButton("Fire1") && Time.time >= _nextFireTimeAllowed)
27          {
28              _nextFireTimeAllowed = Time.time + _fireDelay;
29
30              Screen.lockCursor = true;
31
32              Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
33              RaycastHit hitInfo;
34
35              Vector3 direction;
36              bool foundTarget = false;
37
38              if (Physics.Raycast(mouseRay, out hitInfo) && (Camera.main.WorldToScreenPoint(hitInfo.point).z >=
                        Camera.main.WorldToScreenPoint(_muzzle.position).z))
39              {
40                  foundTarget = true;
41                  direction = hitInfo.point - _muzzle.position;
42                  direction.Normalize();
43              }
44              else
45              {
46                  direction = Camera.main.ViewportToWorldPoint(new Vector3(0.5f, 0.5f, 50)) - _muzzle.position;
47                  direction.Normalize();
48              }
49
50              RaycastHit gunHitInfo;
51              if (Physics.Raycast(_muzzle.position, direction, out gunHitInfo))
52              {
53                  foundTarget = true;
54              }
55
56              Shoot(foundTarget, gunHitInfo, direction);
57          }
58      }
59
60      virtual protected void Shoot(bool foundTarget, RaycastHit hitInfo, Vector3 direction)
61      {
62      }
63  }
```

Artık Input.GetButtonDown() yerine Input.GetButton() kullanıyoruz. Bu fonksiyon, ilgili tuşa basılı tuttuğumuz sürece "true" döndürür. GetButtonDown() fonksiyonu sadece bir kere "true" döndürüyordu.

EnemySpawnManager'daki gibi, ateş etme vaktinin (_nextFireTimeAllowed) gelip gelmediğini kontrol ediyor ve ancak vakti geldiğinde ateş ediyoruz. Buradaki "_fireDelay" değişkenimiz, silahın kaç saniyede bir ateş edebileceğini belirliyor.

İpucu

Player ateş ettiğinde bir ateş etme sesinin çalmasını sağlayın.

Özet Geçecek Olursak...

Parent class kullanarak kodlarımızı nasıl daha organize hale getirebileceğimizi gördük. Ayrıca silah sistemini de geliştirdik. Böylece bir bölümün daha sonuna geldik.