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

Unity 3D İle Oyun Programlama

Bölüm 9: 3D Modeller ve Animasyonlar

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

Oyunumuzda ciddi bir ilerleme kaydettik. Oyuncuyu kontrol edebiliyoruz, düşmanın bizi kovalamasını sağlayabiliyoruz ve düşmanları vurup öldürebiliyoruz.

Şimdi sahnedeki dandik küp ve kapsül modellerini animasyonlu 3D modellerle değiştirerek görsel anlamda çağ atlayacağız.

Gerçek bir oyun yapıyor olsaydık muhtemelen ekibin 3D modelcisi karakterlerin modellerini kesin olarak tamamlayana kadar yine küp ve kapsüllerle devam ederdik.

Ama bazen sizden, yani ekibin programcısından, bir 3D modeli test amaçlı oyuna ekleyip oyunun bitince nasıl birşey olacağını kabataslak göstermenizi isterler.

Bu, er ya da geç karakterler için 3D model entegrasyonu yapmayı öğrenmelisiniz demektir. İşte bu bölümde tam olarak bunun üzerinde duracağız.

Sahneye 3D modelleri ekledikçe scriptlerimizde şimdiye değin farketmediğimiz hataları (bug) gün yüzüne çıkaracağız.

3D Modeller Hakkında

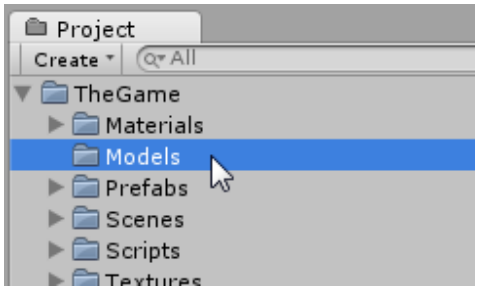
3D modeller, sahnelerimizde yer alan/alacak olan üç boyutlu objelerdir. Genellikle 3D modeller animasyonlara sahip olurlar. Örneğin bir zombi modelinden yürüme, dikilme, saldırı gibi animasyonlara sahip olması beklenir.

Programcı olarak bize düşen görev modelleri oyuna entegre etmek ve uygun durumda uygun animasyonun oynatılmasını sağlamaktır.

3D Zombi Modelini Import Etmek

En başta bahsettiğim gibi eğitim boyunca hazır asset'ler kullanacağız. Buna 3D modeller ve animasyonlar da dahil.

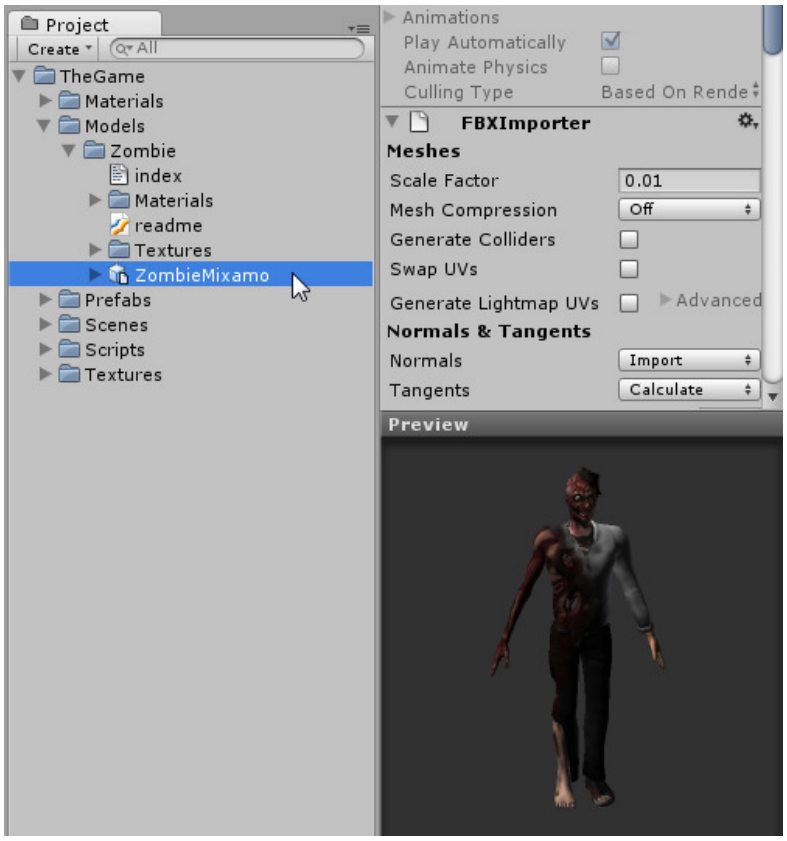
“TheGame” içinde “Models” klasörü oluşturun. Buraya 3D modelleri koyacağız.



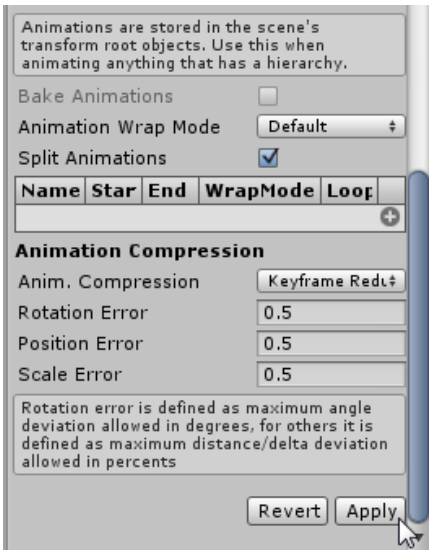
Winrar arşivini çıkardığınız yerde Zombie diye bir klasör var. O klasörü kopyalayıp projenizdeki "Models" klasörünün içine yapıştırın. Ardından dilerseniz Zombie klasöründeki index ve readme dosyalarını silin.

ÇEVİRMEN EKLEMESİ: Başka hiçbir şey yapmadan önce zombi modelinde bir ayar yapacağız yoksa animasyon verirken çok dert olacak (tutorial eski bir Unity sürümü için hazırlanmış ve yeni sürümlerde bu değişikliği yapmak şart.). "ZombieMixamo" asset'ini seçin. Inspector'da yukarıda "**Rig**" adındaki butona basın. "**Animation Type**"ı "Generic"ten "Legacy"e çevirin ve Apply'a basın. Bu kadar!

Zombie klasörünün içinde “ZombieMixamo” isimli bir asset var. Bu, 3D zombi modelinin ta kendisi. Eğer bu dosyayı seçerseniz Inspector'da modelin bir önizlemesi (preview) görünür:



Şimdi ZombieMixamo'yu tutup Scene panelinde bir yere sürükleyin.



Büyük olasılıkla birşey göremeyeceksiniz zira zombi modeli sahnede çok küçük kalmış durumda. Ancak F tuşuna basıp kamerayı zombiye odaklayınca onu görebilirsiniz.

Project panelinden ZombieMixamo'yu tekrar seçin. Inspector'un başlarında “Scale Factor” değeri göreceksiniz. Şu anda 0.01 olarak ayarlanmış. İşte bu yüzden model sahnede çok küçük kalıyor.

Bu değeri 0.58 olarak değiştirin ve Inspector'un aşağı kısımlarında yer alan “Apply” butonuna basın. Değişiklik modele uygulanacak.

Scale Factor modelin büyüklüğünü belirler. Artık zombi, sahnedeki küp modelleri ile hemen hemen aynı boyutlarda.

ÇEVİRMEN EKLEMESİ: Neden boyutlandırma aracını (**R**) kullanmadık diyebilirsiniz. Eğer boyutlandırma aracını kullansaydınız da sonuç kesinlikle birebir aynı olacaktı, ama tek ve önemli bir fark olacaktı: **Scale Factor** kullanıp da boyutlandırma aracını kullanmazsanız bu Unity'de **runtime performans** açısından daha iyi oluyor.

Bilgilendirme

Unity'de küp modelinin Scale değerinin (1,2,1) olduğunu hatırlayın. Yani küp 2 metre yüksekliğindeydi. Zombi modelinin Scale Factor'ünü değiştirerek onun da yaklaşık bu ebatlarda olmasını sağladık.

Görsel 9.1: Zombi modeli artık küp modeliyle neredeyse aynı ebatlarda.

Zombi Modelini Kullanmak

Artık Enemy objelerinde o sıradan küp modeli yerine zombi modelini kullanacağız.

Sahnedeki bir Enemy objesini seçin. Objenin boyutlandırmasını (scale) (1,2,1)'den (1,1,1) olarak değiştirin. Zombi modeli zaten gerçek bir insan ebatlarında olduğundan eğer scale değerini (1,2,1) olarak bıraksaydık zombi 2 kat daha uzun olacak ve "garip" bir görüntü oluşacaktı.

Şimdi küp modelini kullanmayı keselim. “Cube (Mesh Filter)” component'ini bulup Remove Component diyerek silin (component'in sağındaki dişli ikonuna tıklayın). Aynı şeyi "Mesh Renderer" için de yapın.

Ardından Enemy'nin pozisyonunun (position) Y değerini 0.5 yapın. Artık objenin pivot noktası tam zeminin üzerinde olacak. Bunu yapmamızın sebebi şu ki zombi modelini seçecek olursanız onun da pivot noktasının ayak hizasında olduğunu göreceksiniz. İki objenin pivot noktalarını üst üste koyacak olursanız zombi tam yere değiyor konumda olacak.

Eğer zombinin pivot noktasını ayak hizasında değil göbek hizasında görüyorsanız yukarıdaki toolbardan Center değerini Pivot olarak değiştirin.

Şimdi zombi modelini Enemy objesinin içine atacağız. ZombieMixamo'yu Hierarchy'den tutup Enemy'nin üzerine sürükleyerek Enemy objesinin bir **child objesi** yapın.

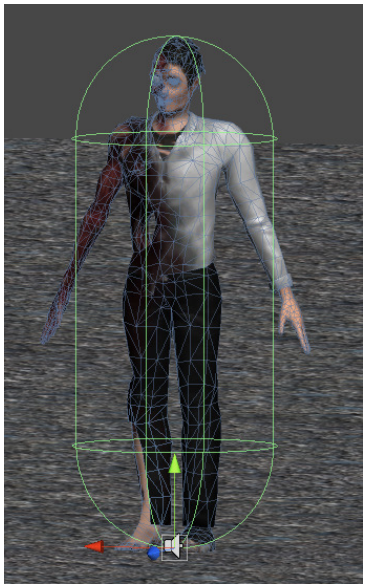
Artık ZombieMixamo, Enemy objesi ile beraber hareket edecek. ZombieMixamo'yu seçip Position değerini (0,0,0) yapın. Bunu yaptıktan sonra zombi ile Enemy objesinin pivotları üst üste gelecek.

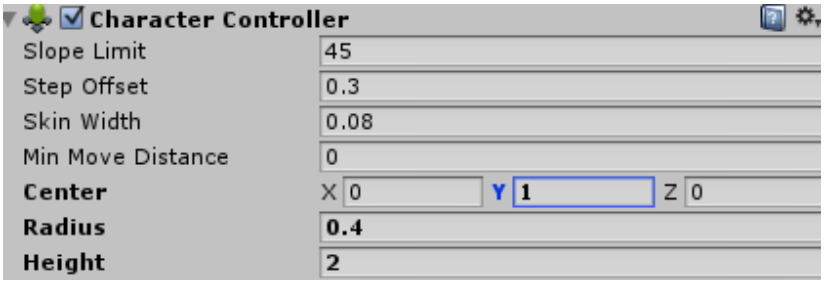
Zemindeki yeşil küreyi görüyor musunuz? Bu küre, Enemy'nin Character Controller component'inin temas alanı olarak kabul ettiği ve öyle hesaplamalar yaptığı alan. Bu temas alanını zombi modeline uygun şekilde düzenlememiz lazım. Bunun için Character Controller'da şu değişiklikleri yapın:

Height: 2

Radius: 0.4

Center: (0,1,0)

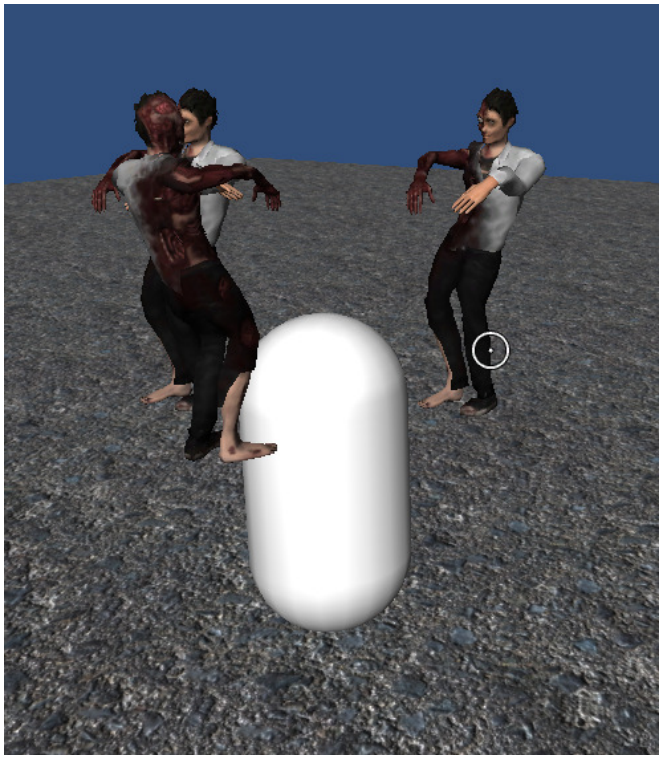




Artık Character Controller'un temas alanı zombinin etrafını güzelce kaplıyor. Kolların biraz dışarıda kalması sorun değil.

Zombi modelini Enemy objesine verdiğimize göre yaptığımız tüm bu değişiklikleri sahnedeki tüm Enemy objelerine uygulayalım. Tahmin edeceğiniz üzere bunu yapmanın yolu da değişiklikleri prefab'a uygulamak. Ve bunu başarmak da size sadece tek bir tıklama kadar uzak! Enemy objesi seçiliyken Inspector'daki Apply butonuna basın, böylece tüm Enemy objeleri ve prefab'ın kendisi otomatik olarak güncellenecek.

Eğer sahnedeki diğer Enemy objeleri havadaysa onların Y pozisyonunu da 0.5 yapın ve oyunu çalıştırın. Bir hata ile karşılaşacaksınız: oyunumuzdaki zombiler uçuyor!



Zombiler uçuyor çünkü onlara etki eden bir yerçekimi kodlamadık.

Ama PlayerMovement scriptinde yerçekimi kodladık bile. Yapmamız gereken aynı konsepti EnemyMovement scriptine de uygulamak.

Bilgilendirme

Aslında PlayerMovement ile EnemyMovement'un çok ortak yönü var. Bu iki class'ın yaptığı ortak şeyleri tek bir class'ta yazsak daha güzel olurdu. İlerleyen safhalarda bununla da uğraşacağız.

Zombilere Yerçekimi Uygulamak

Yapacağımız tek şey PlayerMovement'taki birkaç satırı EnemyMovement'a da yazmak. EnemyMovement scriptini açıp şöyle güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyMovement : MonoBehaviour
05 {
06     CharacterController _controller;
07     Transform _player;
08
09     [SerializeField]
10     float _moveSpeed = 5.0f;
11
12     [SerializeField]
13     float _gravity = 2.0f;
14
15     float _yVelocity = 0.0f;
16
17     void Start()
18     {
19         GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
20         _player = playerGameObject.transform;
21
22         _controller = GetComponent<CharacterController>();
23     }
24
25     void Update()
26     {
27         Vector3 direction = _player.position - transform.position;
28         direction.Normalize();
29
30         Vector3 velocity = direction * _moveSpeed;
31
32         if (!_controller.isGrounded)
33         {
34             _yVelocity -= _gravity;
35         }
36
37         velocity.y = _yVelocity;
38
39         _controller.Move(velocity * Time.deltaTime);
40     }
41 }
```

Tıpkı PlayerMovement'taki gibi artık düşmanların da bir _yVelocity değişkeni var ve bu değişken dikey düzlemdeki hızı temsil ediyor. Değişkenin değeri negatif olunca Move fonksiyonu vasıtasıyla düşman yere düşmeye başlıyor.

Oyunu çalıştırdığınızda zombilerin artık uçmadığını göreceksiniz.

Zombilerin Player'a Doğru Bakmasını Sağlamak

Şu anda zombilerin bize bakmak gibi bir dertleri yok anlaşılan. Ama biz bu sorunu şimdi çözeceğiz.

EnemyMovement scriptini güncelleyelim:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyMovement : MonoBehaviour
05 {
06     CharacterController _controller;
07     Transform _player;
08
09     [SerializeField]
10     float _moveSpeed = 5.0f;
11
12     [SerializeField]
13     float _gravity = 2.0f;
14
15     float _yVelocity = 0.0f;
16
17     void Start()
18     {
19         GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
20         _player = playerGameObject.transform;
21
22         _controller = GetComponent<CharacterController>();
23     }
24
25     void Update()
26     {
27         Vector3 direction = _player.position - transform.position;
28         direction.Normalize();
29
30         Vector3 velocity = direction * _moveSpeed;
31
32         if (!_controller.isGrounded)
33         {
34             _yVelocity -= _gravity;
35         }
36
37         velocity.y = _yVelocity;
38
39
40         transform.rotation = Quaternion.LookRotation(direction);
41
42
43         _controller.Move(velocity * Time.deltaTime);
44     }
45 }
```

Yaptığımız şey zombilerin eğimini (rotation) değiştirerek bize doğru bakmalarını sağlamak. “transform.rotation” komutu ile eğimi değiştirebiliyoruz.

27. satırda elimizde bir yön (direction) değişkeni oluyor ve bu yön vektörünün ucu player'a doğru bakıyor. Hatırlarsanız bu vektörü kullanarak da düşmanı hareket ettiriyoruz.

Quaternion.LookRotation fonksiyonu bir yön vektörü alır ve bunu transform.rotation'ın anlayabileceği bir veri türüne (Quaternion) çevirir.

Quaternion, tıpkı Euler açılar gibi eğimi temsil etmek için kullanılan bir veri türüdür.

Oyunu çalıştırın. Malesef zombiler yanımıza gelince sapıtmaya başlıyor ve zıpladıkça havaya doğru dönüyorlar.

Bilgilendirme

Niçin “quaternion” veri türünü kullanıyoruz?

Eğimi Euler türünde temsil etmek tercih edilmemiş Unity'de. Çünkü Euler açılar “gimbal lock” adı verilen bir handikaptan mağdurdurlar. Bu "gimbal lock"un ne olduğunu anlatan güzel bir video var: http://www.youtube.com/watch?v=rrUCBOlJdt4.

Quaternion veri türünde ise gimbal lock sorunu yoktur. Başka artı noktaları daha olsa gerek ki Unity tarafından tercih edilmektedir.

Malesef quaternion'ları anlamak zordur. Bir quaternion dört değerden oluşur: x, y, z ve w. Bu değerler Euler açıların aksine 0'dan 360'a kadar bir değer aralığına sahip değildir. İşleyişi daha farklıdır yani.

Quaternion veri türünü kullanarak eğimi temsil etmenin nasıl işlediği şurada anlatılmaktadır: http://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation. Eğer ilk saniyede kafanız karışırsa sorun etmeyin. Quaternion'ları müthiş bir şekilde bilmeniz gerekmiyor. Sadece onları Unity'de kullanabilecek kadar temel bilgiye sahip olsanız yeter.

Görsel 9.2: Kod pek de istediğimiz gibi çalışmıyor.

Zombilerimiz yukarı-aşağı bakıyor çünkü yön vektörü (direction) bir Y değerine sahip. Y değeri dikey eksende hareketi sağlar.

Çözüm basit. Yön vektörünün y değerini sıfırlayıp ondan sonra eğimi değiştiriyoruz:

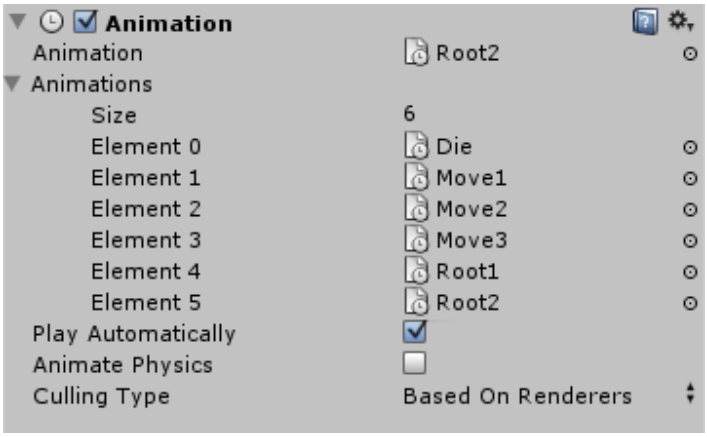
```
25 void  Update()
26      {
27          Vector3 direction = _player.position - transform.position;
28          direction.Normalize();
29
30          Vector3 velocity = direction * _moveSpeed;
31
32          if  (!_controller.isGrounded)
33          {
34              _yVelocity -= _gravity;
35          }
36
37          velocity.y = _yVelocity;
38
39
40          direction.y = 0;
41          transform.rotation = Quaternion.LookRotation(direction);
42
43
44          _controller.Move(velocity * Time.deltaTime);
45      }
46 }
```

Artık zombiler bizi "normal" bir şekilde takip ediyorlar.

Zombilere Yürüme Animasyonu Vermek

Geldik oyunu renklendirecek bir detaya: bu bölümü bitirdiğimizde zombilerimiz yürüyebilecek! Bize doğru kayarak gelmeyecekler artık.

“ZombieMixamo” objesini seçip Inspector'a bakın. Animation component'ine sahip olduğunu göreceksiniz. Bu component'teki Animations değerini genişlettiğinizde modelin sahip olduğu tüm animasyonlar gün yüzüne çıkacak:



Bu animasyonların arasında “Move1”, “Move2” ve “Move3” adında üç animasyon da var. Bunlar zombi için hazırlanmış yürüme animasyonları.

Aslında kod yazmadan da bir animasyonu oynatmak mümkün ama biz kod yazarak animasyon üzerinde tam hakimiyet sahibi olacağız.

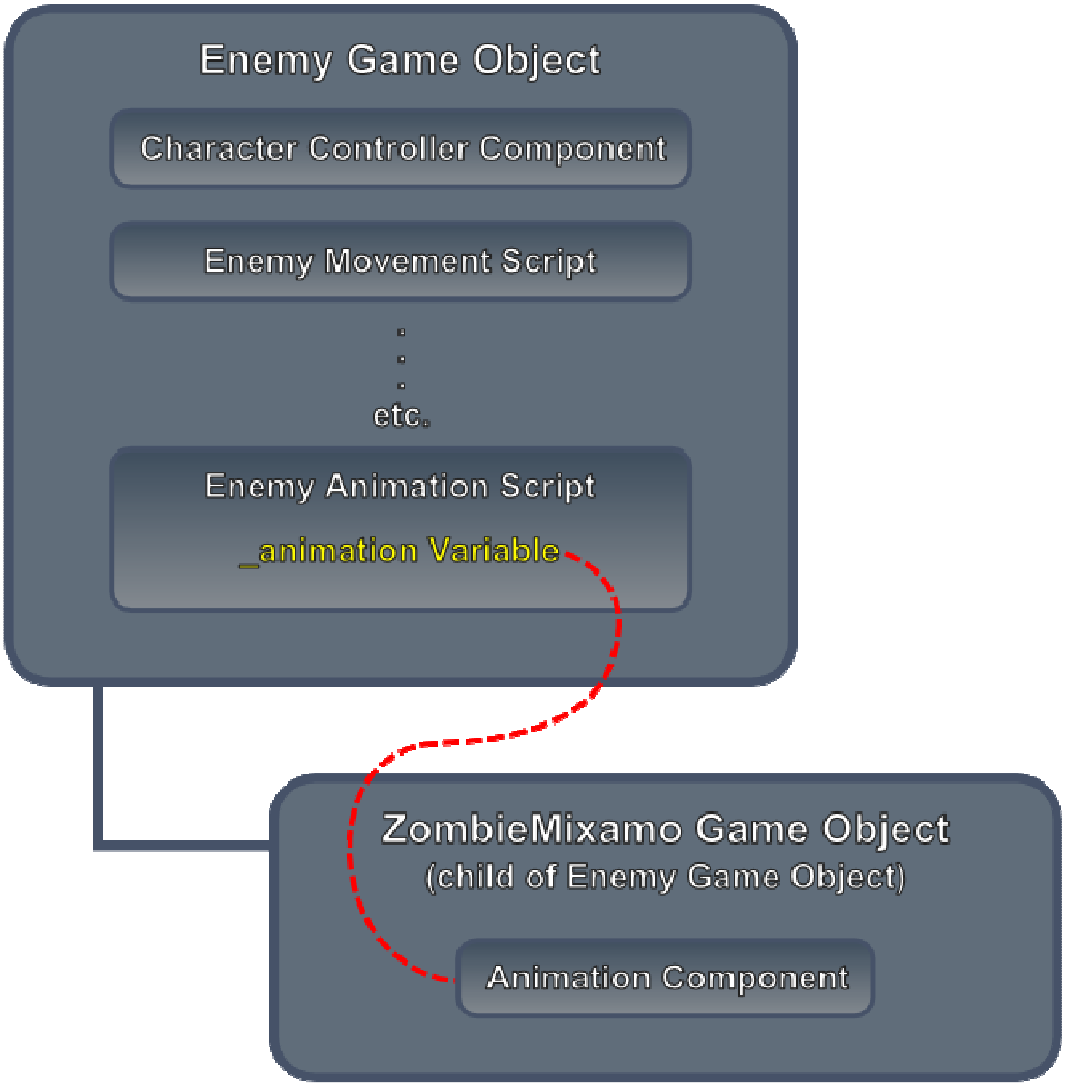
“EnemyAnimation” adında yeni bir C# scripti oluşturun:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAnimation : MonoBehaviour
05 {
06     Animation _animation;
07
08     void Start()
09     {
10         _animation = GetComponentInChildren<Animation>();
11
12         _animation["Move1"].wrapMode = WrapMode.Loop;
13         _animation.Play("Move1");
14     }
15 }
```

Zombi objesinin Animation component'ine hızlıca erişebilmek için onu _animation isminde bir değişkende depoluyoruz. Depolama işlemini 10. satırda GetComponentInChildren ile yapıyoruz.

Neden GetComponent kullanmadık? Çünkü scripti “Enemy” objesine vereceğiz ama Animation component'i Enemy'nin child'ı olan “ZombieMixamo” objesinde. GetComponentInChildren fonksiyonu child objedeki bir component'i döndürmeye yarar.

GetComponentInChildren fonksiyonunu çağırınca olan biteni şu diyagramda görebilirsiniz:



GetComponentInChildren ile ZombieMixamo'nun Animation component'ini çekiyoruz ve onu "_animation" değişkenine değer olarak veriyoruz.

Dediğim gibi, GetComponentInChildren<Animation>() komutu, GetComponent<Animation>()'ın aksine aramaya child objeleri de dahil eder (bu esnada parent obje de aranır, sadece child objeler aranmaz.) ve bulduğu ilk Animation component'ini döndürür.

13. satırda bu Animation component'inin "Move1" isimli animasyonu oynatmasını sağlıyoruz. Böylece zombinin yürüme animasyonu oynatılıyor.

Yürüme animasyonunun sürekli gerçekleşmesi makul olur, yani bir kere oynayıp sonra durması değil. Bu yüzden 12. satırda animasyon için "wrapMode = WrapMode.Loop" yaparak onun sürekli oynamasını (loop) sağlıyoruz.

EnemyAnimation script'ini bir Enemy objesine verip Apply butonuna basarak değişikliği tüm prefab klonlarına uygulayın. Ardından oyunu çalıştırın ve zombilerin güzel yürüme animasyonlarının tadını çıkarın!

Yürüme Animasyonuna Değişiklik Katmak

Zombilerimizin yürüme animasyonu eş zamanlı oynuyor ve bu da bir yapaylık katıyor oyuna. Bu sorunu çözmek için her zombinin animasyonunun başlangıç karesini değiştirebiliriz. Varsayılan olarak bir animasyon ilk kareden (frame) oynatılmaya başlar ama biz bunu değiştirerek animasyonun farklı bir kareden oynatılmaya başlamasını sağlayabiliriz:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAnimation : MonoBehaviour
05 {
06     Animation _animation;
07
08     void Start()
09     {
10         _animation = GetComponentInChildren<Animation>();
11
12         _animation["Move1"].wrapMode = WrapMode.Loop;
13
14         _animation.Play("Move1");
15         _animation["Move1"].normalizedTime = Random.value;
16     }
17 }
18
```

15. satırda animasyonun başlangıç karesini o düşman objesi için değiştiriyoruz. Random.value fonksiyonu 0.0 ile 1.0 arasında rastgele (random) bir değer döndürmeye yarar, mesela 0.5...

Bu rastgele değeri animasyonun normalizedTime'ına veriyoruz. “normalizedTime”, animasyonun hangi karesinde olduğumuzu belirler.

Eğer normalizedTime 0.5 olursa animasyonun ortadaki karesindeyizdir, 0.0 ise ilk karesindeyizdir (varsayılan değer budur) ve 1.0 ise sondaki karesindeyizdir...

normalizedTime'a rastgele bir değer verdiğimiz için her zombinin animasyonu farklı bir kareden oynamaya başlayacak ve animasyonları eş zamanlı oynuyor izlenimi uyandırmayacak.

Farklı Yürüme Animasyonu Oynatmak

Bildiğiniz gibi zombinin üç farklı yürüme animasyonu var. Kodu biraz daha genişleterek her zombinin bu üç animasyondan rastgele birisini kullanmasını sağlayabiliriz:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAnimation : MonoBehaviour
05 {
06     Animation _animation;
07
08     void Start()
09     {
10         _animation = GetComponentInChildren<Animation>();
11
12         string animationToPlay = "";
13         switch (Random.Range(0, 3))
14         {
15         default:
16         case 0:
17             animationToPlay = "Move1";
18             break;
19         case 1:
20             animationToPlay = "Move2";
21             break;
22         case 2:
23             animationToPlay = "Move3";
24             break;
25         }
26
27         _animation[animationToPlay].wrapMode = WrapMode.Loop;
28
29         _animation.Play(animationToPlay);
30         _animation[animationToPlay].normalizedTime = Random.value;
31     }
32 }
33
```

Bir animasyonu oynatırken hep animasyonu ismiyle çağırıyorduk: "Move1" şeklinde. Elimizde üç tane animasyon var. Bu üç animasyondan birini seçip ismini, string türündeki animationToPlay değişkenine değer olarak veriyoruz ve artık "Move1" diye belli bir animasyon ismi kullanmak yerine animationToPlay değişkeninde depolanmış animasyon ismini kullanıyoruz.

13. satırda Random.Range(0, 3) fonksiyonunu kullanıyoruz. Bu fonksiyon 0'dan 3'e kadar bir tamsayı (integer) döndürür. Bu aralığa 0 dahildir ama 3 değildir. Yani fonksiyon sadece 0, 1 ya da 2 döndürebilir.

Bu döndürülen değeri bir switch koşulunun içine sokuyoruz ve üç koşul için üç farklı yürüme animasyonundan birisinin ismini animationToPlay değişkeninde depoluyoruz.

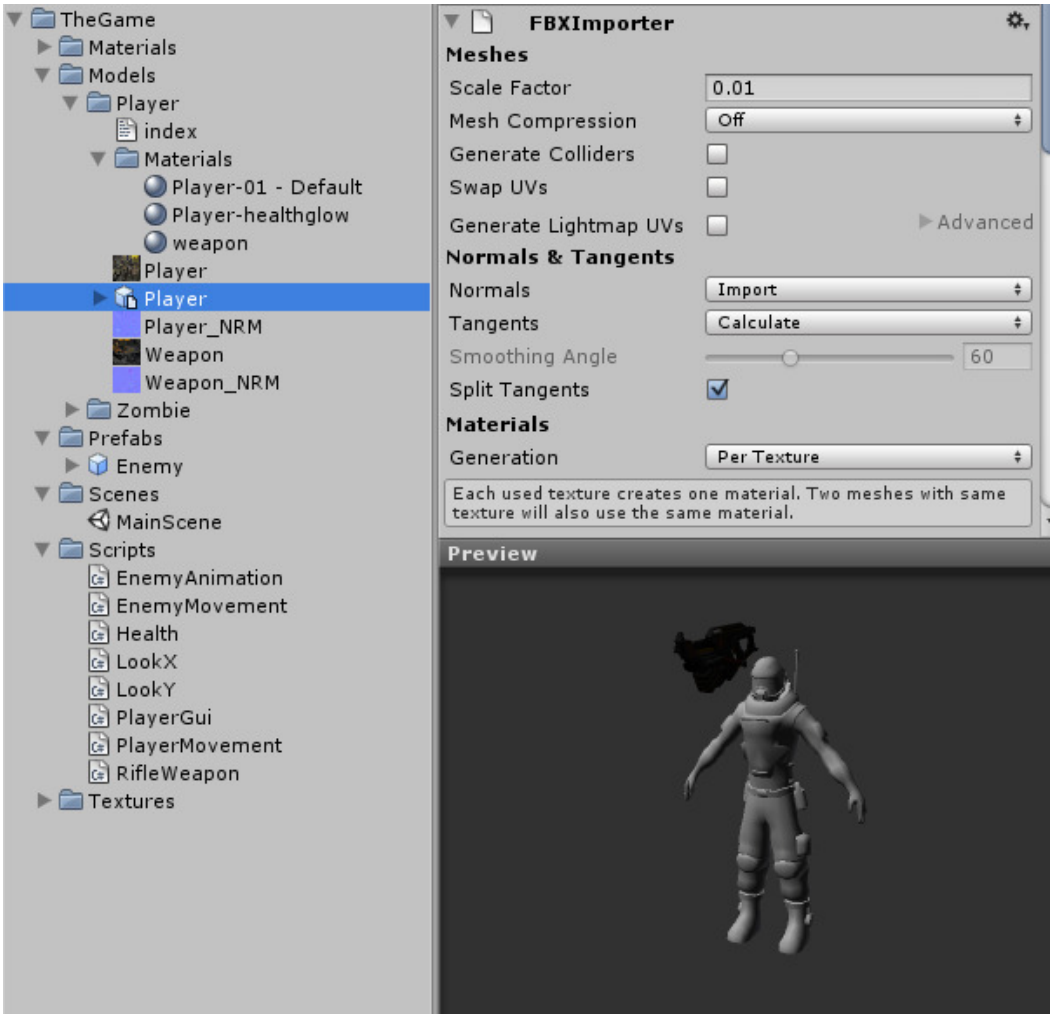
Kaydetmeyi Unutmayın!

Yaptığınız hemen her değişiklikte CTRL+S ile projeyi kaydetmeyi unutmayın.

Oyuncu (Player) Modelini Import Etmek

Zombilerimize makyaj uyguladık. Sıra Player objesinde. Artık onun da dandik kapsül formundan kurtulup detaylı bir 3D modele sahip olmasının vakti geldi de geçiyor!

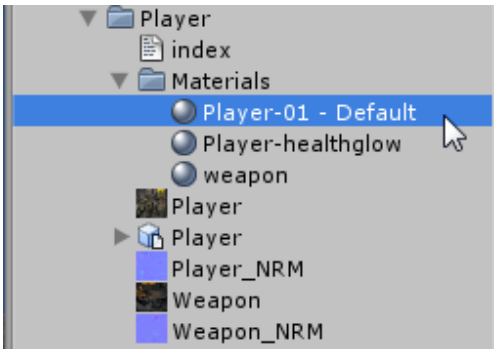
Winrar arşivinin oradaki “Player” klasörünü projenizin Models klasörünün içine kopyalayın. Klasördeki "index" dosyasını isterseniz silebilirsiniz.



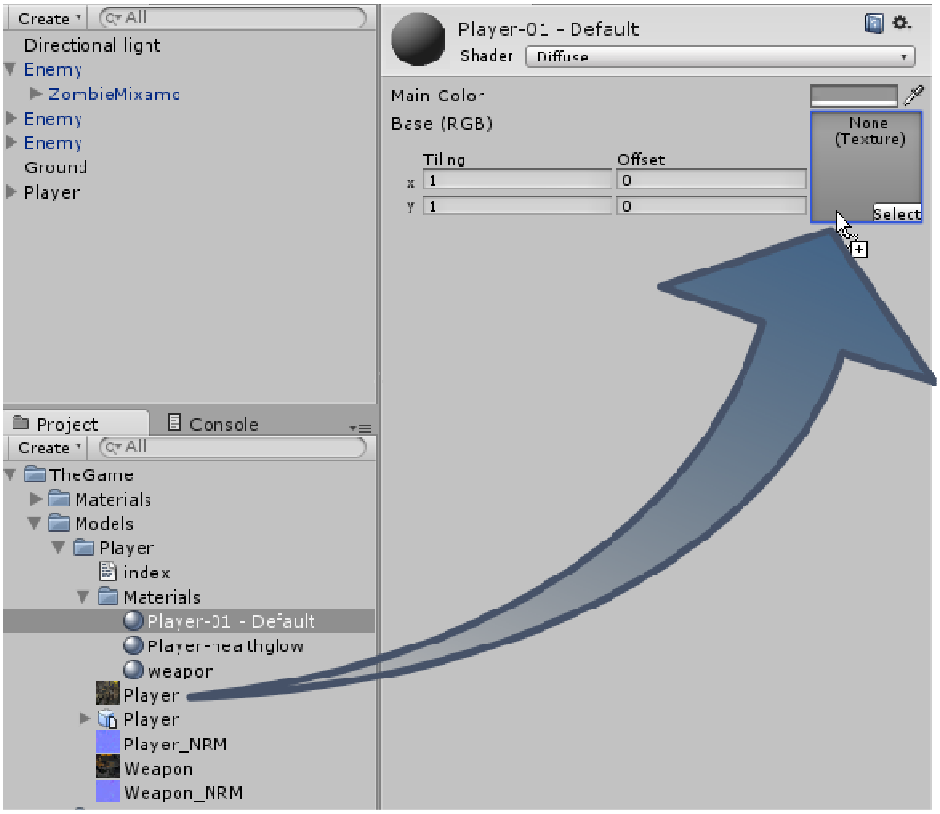
Modeli Project panelinden seçerseniz gri olduğunu göreceksiniz, çünkü henüz modele kaplaması (texture) atanmamış.

ÇEVİRMEN EKLEMESİ: Unity'nin eski sürümünde Player modeliyle beraber gelen materyalin ismi "**Player-01 - Default**" idi ama yeni sürümlerde bu isim "**main_player_diffuse_deleteme**" olarak değişmiş. Yani yazıda "Player-01 - Default" diye geçen materyali siz "main_player_diffuse_deleteme" olarak görün.

Materials klasöründe "Player-01 – Default" isimli bir materyal var. Bu materyal player'ın kullandığı ana materyal. Gerekli işlemi bunun üzerinde yapacağız.

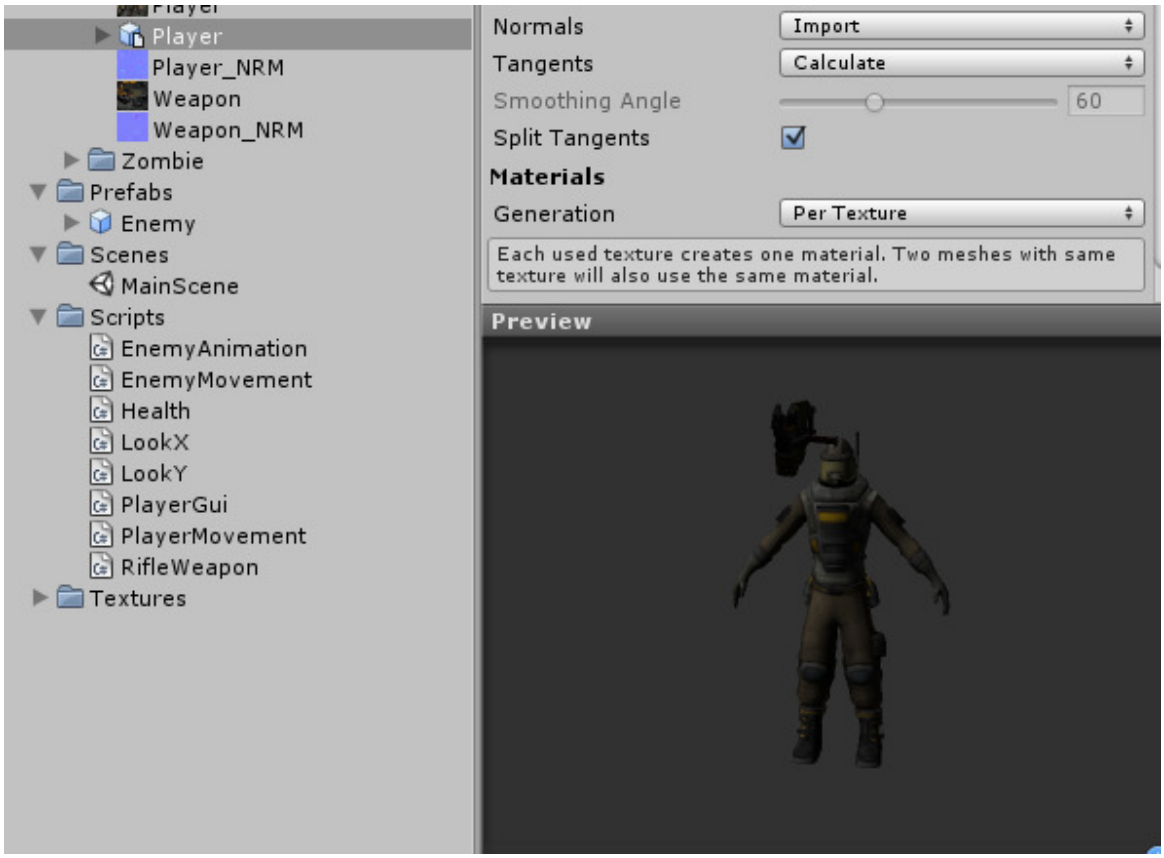


Aslında yapacağımız işlem çok basit. Project panelinden Player isimli kaplamayı (texture) sürükleyip "Player-01 – Default" materyalindeki ilgili kısma yerleştirin:



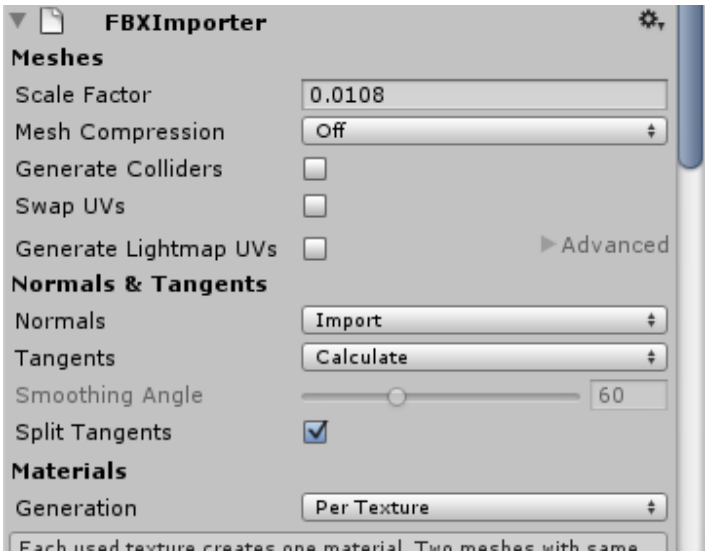
ÇEVİRMEN EKLEMESİ: "Player-01 – Default" materyalinin rengi (**Main Color**) gri. Bu, kaplamanın modelin üstünde gereğinden daha koyu gözükmesini sağlıyor. Büyük ihtimalle yazar bu detayı söylemeyi unuttu. Eğer **Shader**'ı **Diffuse**'tan **Mobile-Diffuse**'a değiştirirseniz hem modeli render etmek daha hızlı gerçekleşir (Mobile-Diffuse shader'ı daha basit işlemlere sahiptir) hem de kaplama, modelin üstünde istendiği gibi (daha aydınlık) durur.

Artık Player modelini seçtiğinizde önizleme ekranında rengarenk bir model sizi bekliyor olacak:



Şimdi Player modelini Project panelinden Scene paneline sürükleyin.

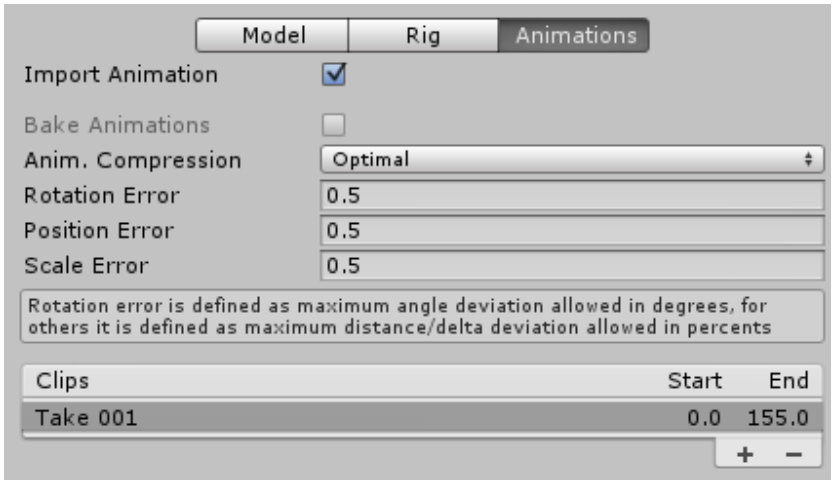
Scale Factor'ü 0.0108 yaparak modelin boyunun kapsülün boyuyla eşdeğer düzeye gelmesini sağlayın. Scale Factor'ü değiştirdikten sonra Apply demeyi unutmayın.



Bu 3D modelde animasyonlar zombi modelinden biraz daha farklı yapılmış. Karakterin tüm animasyonları tek bir uzun animasyonun farklı kare (frame) aralıklarına yerleştirilmiş. Nedeni bilinmiyor.

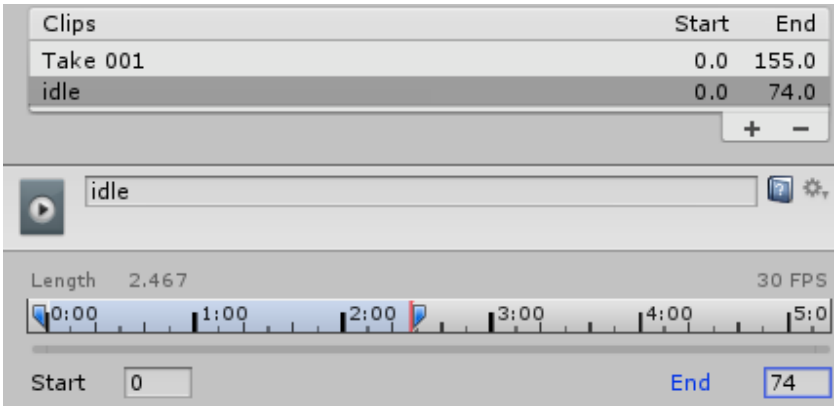
Nedeni ne olursa olsun bu uzun animasyonu parçalara bölmek zorundayız. Böylece karakterin istediğimiz animasyonunu oynatabileceğiz. Bu işlemi Inspector'dan yapıyoruz.

Player modelini Project panelinden seçin. Inspector'un tepesindeki Animations sekmesine geçiş yapın. Orada şu anda Clips kısmında "Take 001" adında tek bir animasyon listelenmekte:



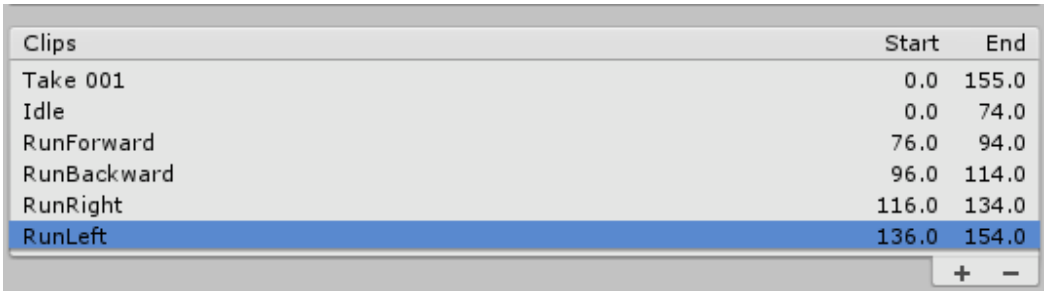
Klibi önizleme penceresinde oynatırsanız tüm animasyonların tek bir büyük animasyonda birleştirilmesinden neyi kastettiğimi görebilirsiniz.

Şimdi biz bu animasyonu parçalara ayıralım. Bunun için "Take 001"in sağ altındaki + işaretine tıklayın. "Take 0010" adında yeni bir klip oluşacak. Bunun ismini “Idle” olarak değiştirin (**NOT:** Ben resimde yanlışlıkla "idle"ı küçük harfle başlattım ama siz "Idle" diye büyük harfle başlatın). "Start" kısmına değer olarak 0, "End" kısmına da değer olarak 74 girin. Bunlar karakterin hareketsiz durduğu animasyonun başlangıç (0) ve bitiş (74) kareleri:



Eğer Idle animasyonunu önizleme penceresinde oynatırsanız animasyonun düzgün oynatıldığını göreceksiniz. Animasyon her ne kadar önizleme penceresinde devamlı (loop halinde) oynasa da oyunda varsayılan olarak animasyonun oynama şekli tek seferlik. Yani animasyon bir kere baştan sonra oynadıktan sonra duruyor, kendini tekrar etmiyor. Bunu çözmek için "Start"ın altındaki "Loop Time" seçeneğini işaretleyin. Artık animasyon oyunda da sürekli (loop) oynayacak.

Şimdi bu işlemi diğer animasyonlar için de yapalım. Dört farklı animasyon daha oluşturun ve başlangıç (Start) ile bitiş (End) değerlerini resimdeki gibi belirleyin:

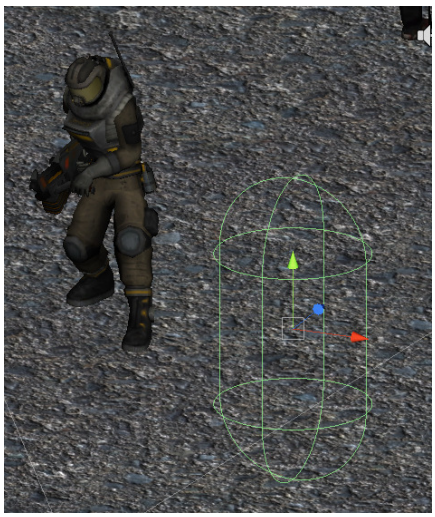


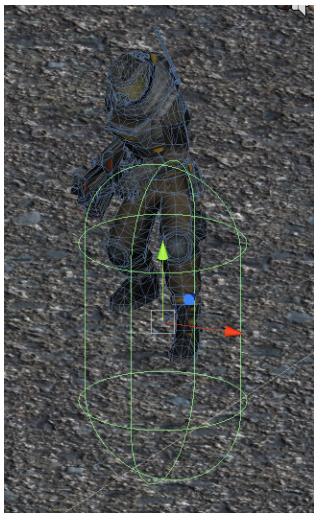
Tüm animasyonları resimdeki gibi ayırdıktan sonra "Take 001" animasyonunu seçin ve + işaretinin sağındaki - işaretine tıklayın. "Take 001" animasyonuna ihtiyacımız olmadığı için onu listeden bu yöntemle silelim. Ardından Apply butonuna basarak değişiklikleri modele uygulayın.

Player Objesi İçin 3D Modeli Kullanmak

Zombide olduğu gibi, kapsül modelinden kurtulup yerine Player modelini kullanacağız.

Player objesini seçin. “Capsule (Mesh Filter)” ve Mesh Renderer component'lerini silin. Objenin Y pozisyonunu 0.5 yapın. Ardından sahnedeki 3D Player modelini, Player objesinin bir child'ı yapın ve 3D modelin pozisyonunu (0,0,0) yapın. Böylece model ile Player'ın pivot noktaları üst üste gelecek.



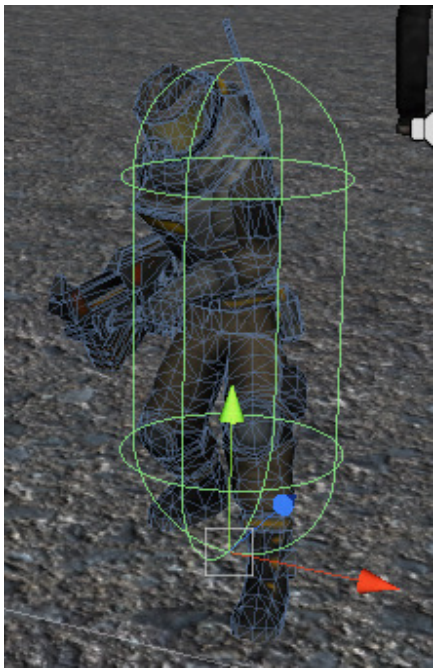


Şimdi Player'ın Character Controller'unda şu değişiklikleri yapın ki yeşille gösterilen temas alanı Player'ın etrafını sarsın:

Height: 1.8

Radius: 0.4

Center: (0, 0.9, 0)



Bazı poligonların kapsülün dışında kalması önemli değil. Modelin genel olarak kapsülün içine girmesi yeterli.

Oyunu çalıştırın. Karakterimiz yerin içinden geçip aşağı düşüyor. Çünkü karakter oyunun başında yere çok yakın. Player'ın Y pozisyonunu 0.51 yapın. Artık karakter aşağı düşmeyecektir.

Nişangahın Konumunu Ayarlamak

Siz de crosshair'in biraz aşağıda kaldığını düşünüyor musunuz?

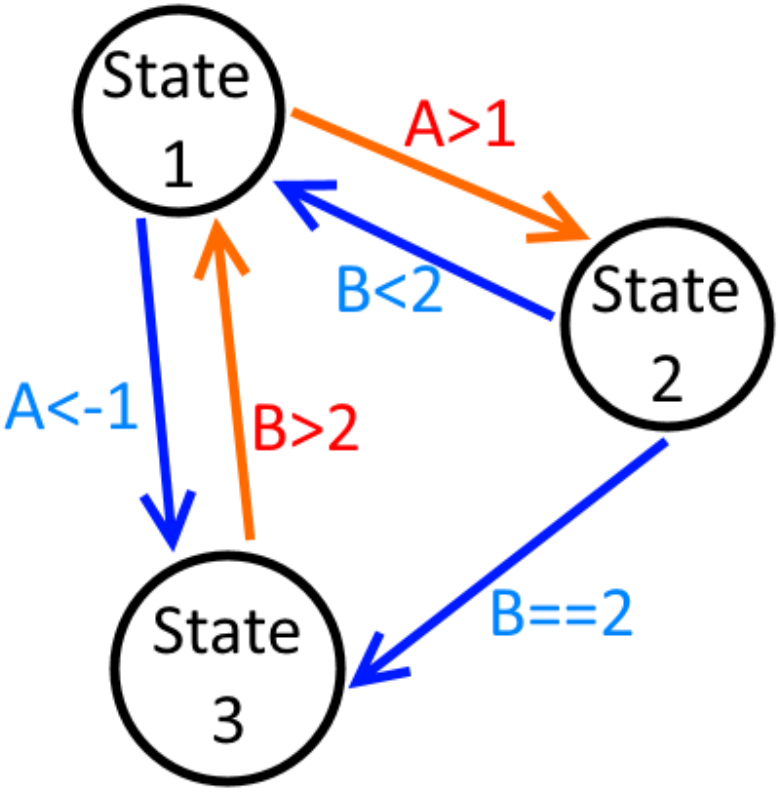
Yapmamız gereken Player'ın child objesi olan LookY objesinin pozisyonunu değiştirmek. Hatırlarsanız LookY objesi kamera objesinin pivot noktası gibi davranıyor ve kamera LookY etrafında dönüyor. İmleç her zaman için ekranın tam ortasında yer alıyor. O halde LookY objesini biraz yukarı taşırsak imleç de daha yukarıda duracak.

LookY objesini seçip pozisyonunu (1, 1.25, 1.5) yapın. Artık kamera daha yukarıda yer almakta ve böylece imleç de daha yukarıda durmakta. Oyunu test edip farkı kendiniz de gözlemleyebilirsiniz.

Karaktere Yürüme Animasyonu Vermek

ÇEVİRMEN EKLEMESİ: Karaktere animasyon verdiğimiz bu kısmı kendi başıma tamamen sıfırdan yazdım. Orijinal metinde Animation component'i kullanarak animasyon veriliyordu ama kod hem biraz karmaşıktı hem de bu aşamayı Unity'nin yeni animasyon sistemi olan Mecanim ile yapmayı görmenizi istedim.

Mecanim sisteminin çalışma yapısı state-machine adında bir konsepte dayanıyor. Bu konsept üniversitede dijital devre tasarımı dersimde de karşıma çıkmıştı, yani köklü bir sistem bu. Peki nasıl çalışır? Örnek bir şema üzerinde göstereyim:



NOT: Bazı oklar turuncu iken bazı oklar mavi. Oklar ile onlara bağlı olan koşullar diğerleriyle karışmasın diye böyle birşey yaptım, yoksa mavi oklar ve turuncu oklar arasında hiçbir fark yok.

Resimde üç adet state var. Oyunun başında bu state'lerden bir tanesi varsayılan olarak aktif olur (**Aynı anda sadece ama sadece bir state aktif olabilir!**). Diyelim ki en başta aktif olan state'imiz "State 1". "State 1"den "State 2"ye ve "State 3"e birer ok çıktığını farkettiniz mi?

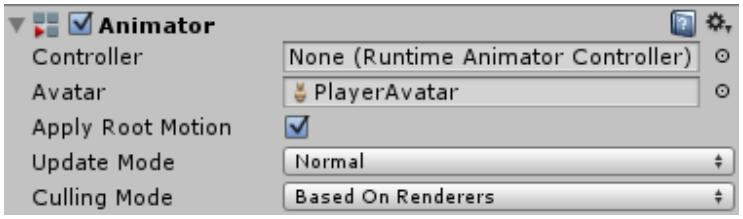
"State 1"den "State 2"ye çıkan okun üzerinde "A>1" yazıyor ve "State 1"den "State 3"e çıkan okun üzerinde "A<-1" yazıyor. Bu oklar birer "iki state arası geçiş" (**transition**)'dir ve yanlarında yazan yazılar da bu transition'ın gerçekleşmesi için gerekli olan koşullardır (**condition**).

Şöyle ki buradaki A ve B değerleri birer değişkendir. Bu değişkenlerin değerlerini kod ile değiştirebilmekteyiz. Şu an "State 1"deyiz ve bu state'den başka bir state'e geçiş yapmak (transition) için iki koşuldan birisinin doğru olması gerekmekte: A'nın değeri ya 1'den büyük olacak (bu durumda "State 2"ye geçilir) ya da -1'den küçük olacak (bu durumda "State 3"e geçilir). Eğer "State 1"deysek ve A'nın değeri -1 ile 1 arasındaysa o zaman koşulların hiçbiri sağlanmamış olur ve biz "State 1"de kalmaya devam ederiz.

Kendi state'lerinizi, hangi state'ler arasında geçiş olacağını (ok çizileceğini)(transition), bu transition'larda koşulların (condition) neler olacağını tamamen siz belirlersiniz. Örneğin "State 2"den "State 3"e geçiş varken "State 3"ten "State 2"ye geçiş yok. Bu tamamen tercih meselesi.

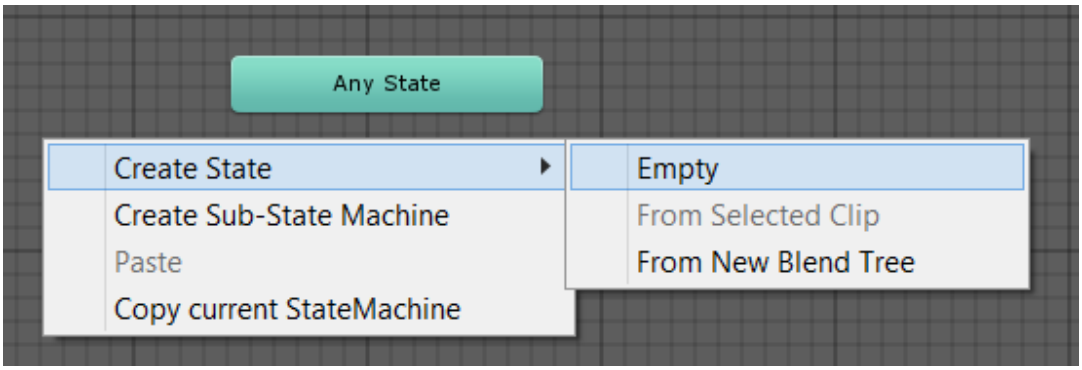
Peki bir state'teyken neler gerçekleşir? Dijital devre tasarımında örneğin yeşil bir LED ışık yanabilir, bir yere bir ışın yollanabilir vb. Unity'nin Mecanim sisteminde ise ne gerçekleşeceği bellidir: animasyon oynatılır.

Unity'de Mecanim sisteminin çalışması için gerekli olan component **Animator**'dur. İşin aslı, yeni bir model import ettiğinizde Animator component'i otomatik olarak gelir. Eğer Player objesinin child objesi olan Player modelini Hierarchy'den seçerseniz Inspector'da Animator component'ini görebilirsiniz.

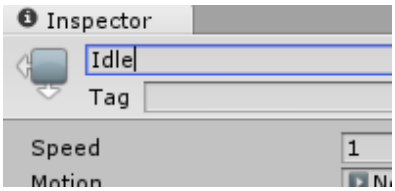


Bu component'te "Controller" adında bir değer yer almakta. Buraya bir Animator Controller (Mecanim state-machine) bağlamalıyız. Bunun için Project panelinde "**Create-Animator Controller**" yolunu izleyin ve oluşan asset'in ismini "PlayerAnimatorController" olarak değiştirin. Ardından bu controller'ı **Models** klasöründe yer alan **Player** klasörünün içine taşıyın. Son olarak da Player modelinin Animator component'indeki **Controller** kısmına değer olarak "PlayerAnimatorController"u verin.

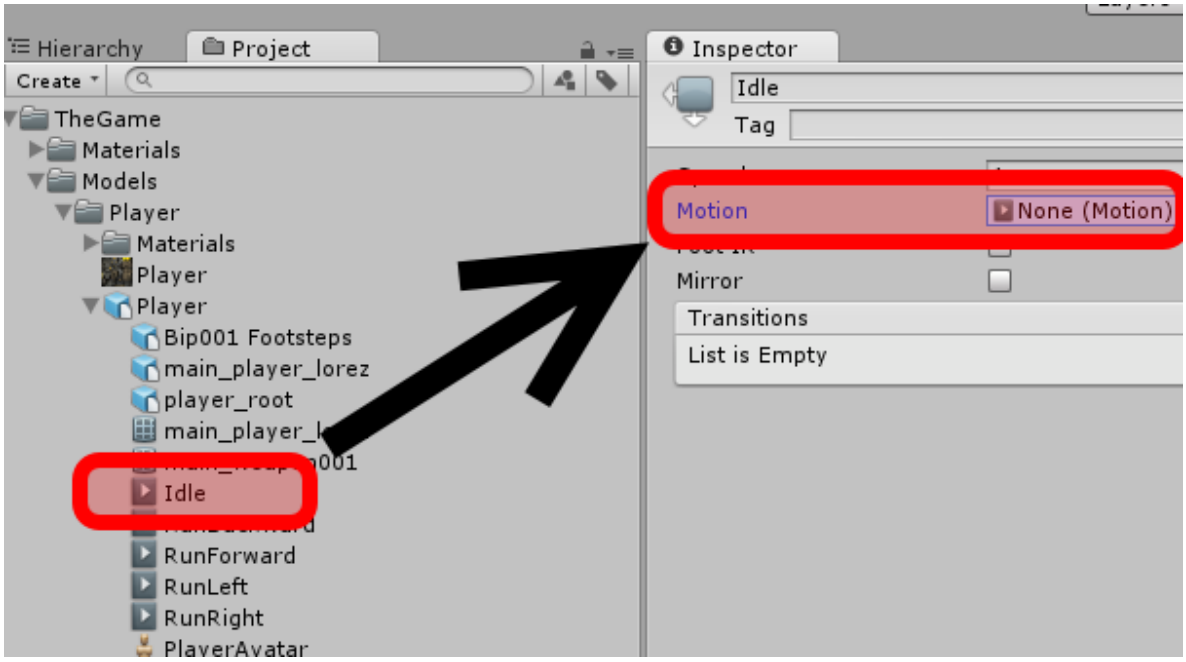
Şimdi "PlayerAnimatorController"a çift tıklayın. Editör arayüzünde **Animator** adında bir pencere belirecek. İşte burada state-machine'imizi kuruyoruz. Şu anda ekranda sadece "Any State" var. Bu state'i kullanmayacağız, nolduğuyla ilgili kafanızı yormayın. Ekranda boş bir yere sağ tıklayın ve "**Create State-Empty**" yolunu izleyin:



Tıkladığınız yerde "New State" adında yeni bir state oluşacak. Bu state'in rengi turuncu, bunun anlamı oyunun başında aktif olacak olan state'in "New State" olacağıdır. "New State"e tıklayın ve Inspector'dan state'in ismini "**Idle**" olarak değiştirin:



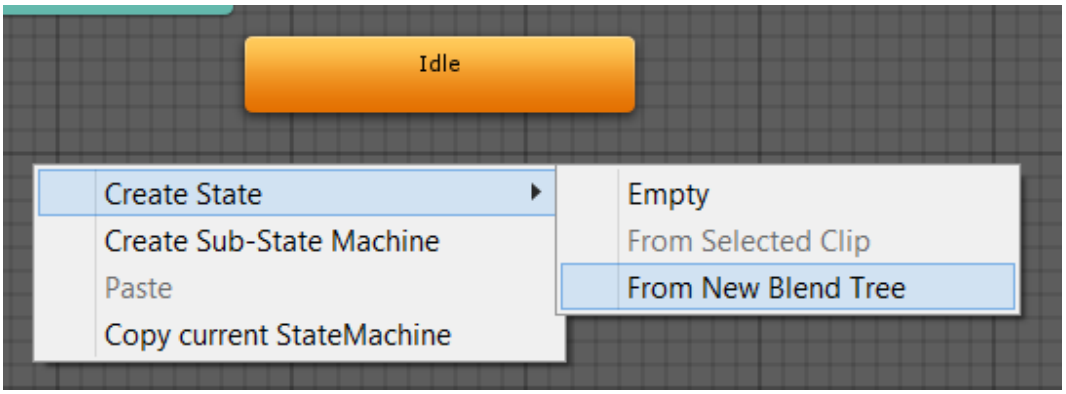
Bu state'te bulunduğumuz zaman karakterin Idle, yani olduğu yerde sabit durma animasyonu oynayacak. Ama önce Idle animasyonunu Idle state'ine tanıtmalıyız. Bunun için ise Idle state'ini seçin ve **Motion** kısmına değer olarak Project panelinden Idle animasyonunu sürükleyin:



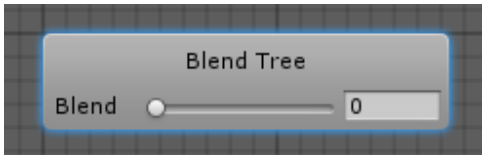
Oyunu şu halde çalıştıracak olursanız karakterin Idle animasyonunu loop halinde oynattığını göreceksiniz. İlk aşamayı tamamlamış olduk. Sırada hareket etme animasyonları var.

Hareket etme animasyonlarında özel bir durum var: elimizde ileri, geri, sağa ve sola koşma olmak üzere sadece dört hareket animasyonu var. Peki karakter ileriye ve sağa hareket ederken hangi animasyon oynayacak? Cevap ilginizi çekebilir: hem ileri koşma hem de sağa gitme animasyonları aynı anda oynayacak. Her iki animasyon da 20 kare (frame) uzunluğunda ve eğer iki animasyonu birleştirirsek (aynı anda oynatırsak) elimizde güzel bir ileri-sağa gitme animasyonu oluyor. Bunu yapmak Mecanim ile mümkün.

Animator penceresinde boş bir yere sağ tıklayıp "Create State-From New Blend Tree" yolunu izleyin:



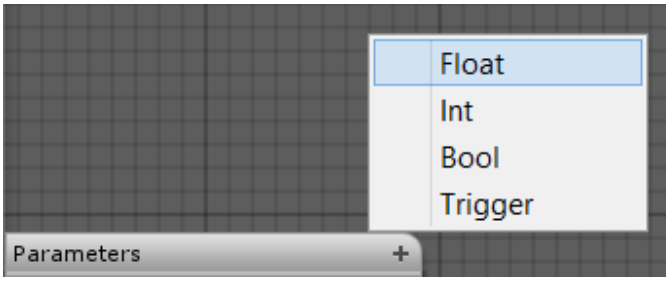
Oluşan state'in ismini "RunForward" olarak değiştirin. Şimdi RunForward state'ine çift tıklayın. Farklı bir state ekranı karşınıza gelecek:



Blend Tree formatı birden çok animasyonu harmanlamak için idealdir. Idle'da Blend Tree kullanmadık çünkü Idle animasyonu tek bir animasyon. Ama ileri koşma animasyonunda harmanlama işlemi yapmamız zorunlu zira dediğim gibi, ileri giderken sağa veya sola da gitmek isteyebiliriz.

Hatırlarsanız örnek olarak gösterdiğim state-machine diyagramında çeşitli koşullar (condition) vardı. Bu koşullarda A ve B isminde iki değişken kullanılıyordu. Bizim Mecanim state-machine'imizde de iki state arasında geçiş yaparken condition'lar olacak. Başka birşey yapmadan önce state-machine'imizde kullanacağımız tüm değişkenleri oluşturmak istiyorum.

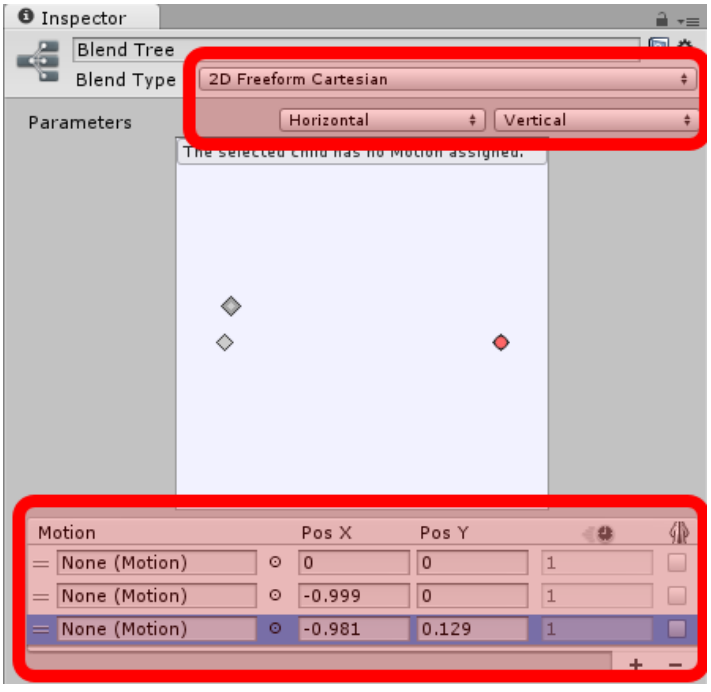
Animator penceresinin sol alt kısmında Parameters var. Burası vasıtasıyla Mecanim'de kullanacağımız değişkenlerimizi oluşturuyoruz. Parameters'ın sağındaki **+** işaretine tıklayıp **Float** deyin:



Oluşan değişkenin ismini "Horizontal" olarak değiştirin. Bir başka Float değişken daha oluşturun ve onun ismini "Vertical" yapın. Son olarak **Bool** türünde bir başka değişken oluşturup ismini Motion yapın:

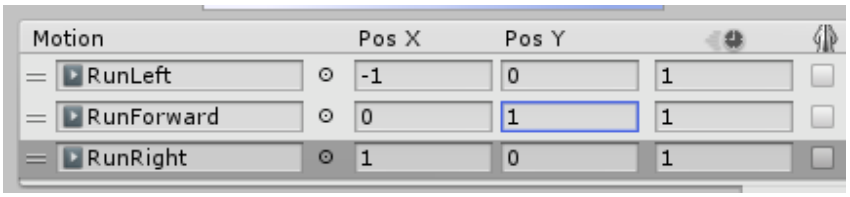
Bu değişkenlerin değerlerini ileride script ile dinamik olarak ayarlayacağız. "Horizontal" değişkeni klavyenin sağ-sol ok tuşlarından aldığımız input'u "**Input.GetAxis("Horizontal")**" değer olarak alacak ve "Vertical" da klavyenin ileri-geri ok tuşlarından aldığımız input'u "**Input.GetAxis("Vertical")**" değer olarak alacak. "Motion" iki state arası geçiş yaparken kolaylık sağlaması için oluşturduğum bir değişken. Eğer klavyeden herhangi bir yön tuşuna basıyorsak o zaman karakter hareket ediyordur (in motion) ve bu durumda Motion true olacak. Ok tuşlarına basmadığımız durumda false olacak.

İleri koşma animasyonunu yapmaya geri dönelim. Blend Tree'yi seçin. Inspector'da Blend Type'ı "**2D Freeform Cartesian**" yapın. Parameters'ı "**Horizontal**"a "**Vertical**" yapın. Sonrasında Motion yazan boş listenin sağ altındaki **+** tuşuna basıp "Add Motion Field" deyin. Bu işlemi iki kere daha yapın, böylece Motion listesinde toplam üç animasyonluk yer olsun.



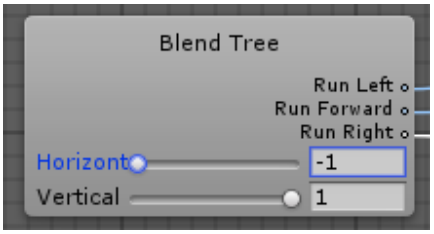
Bu üç animasyonun neler olduğunu tahmin edebilirsiniz: sola koşma animasyonu, ileri koşma animasyonu ve sağa koşma animasyonu. İleri doğru koşarken bu üç animasyonu gerektiği gibi harmanlayacağız.
İleri giderken harmanlayacağımız animasyonlar arasında geri koşma animasyonu olması saçma olurdu.

Motion kısmındaki animasyonlara sırası ile Player'ın "RunLeft", "RunForward" ve "RunRight" animasyonlarını verin ve Pos X ile Pos Y değerlerini de resimdeki gibi güncelleyin:



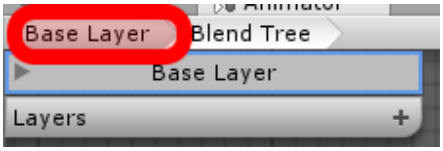
"Pos X" ve "Pos Y" sırasıyla parametre olarak girdiğimiz "Horizontal" ve "Vertical"ı temsil ediyor. Şöyle ki eğer Horizontal'ın değeri -1, Vertical'ın değeri 0 olursa (klavyeden sadece sol ok tuşuna basılıyor) sadece RunLeft animasyonu oynarken Horizontal 1, Vertical 0 iken (sadece sağ ok tuşuna basılıyor) sadece RunRight ve Horizontal 0, Vertical 1 iken (ileri ok tuşuna basılıyor) sadece RunForward animasyonu oynuyor. Peki Horizontal -1 ve Vertical 1 iken (sol ve ileri ok tuşlarına basılıyor) ne oluyor? İleri koşma (RunForward) ve sola koşma (RunLeft) animasyonları aynı anda oynuyor. Benzer şekilde Horizontal ve Vertical 1 iken de RunForward ve RunRight animasyonları beraber oynuyor.

Bu dediğimi kendiniz görerek daha iyi anlayabilirsiniz. Inspector'un alt kısmında Preview kısmı var. Eğer orada Player modeli yoksa ve "No model is available for preview." yazıyorsa Project panelinden Player modelini sürükleyip o Preview alanına bırakın. Artık orada karakterimiz gözükecek ve animasyonları buradan canlı olarak test edebileceğiz. Şimdi o önizleme kısmındaki oynatma tuşuna basın. Ardından Blend Tree'deki parametrelerin değerlerini değiştirerek animasyonun harmanlanmasını izleyin:

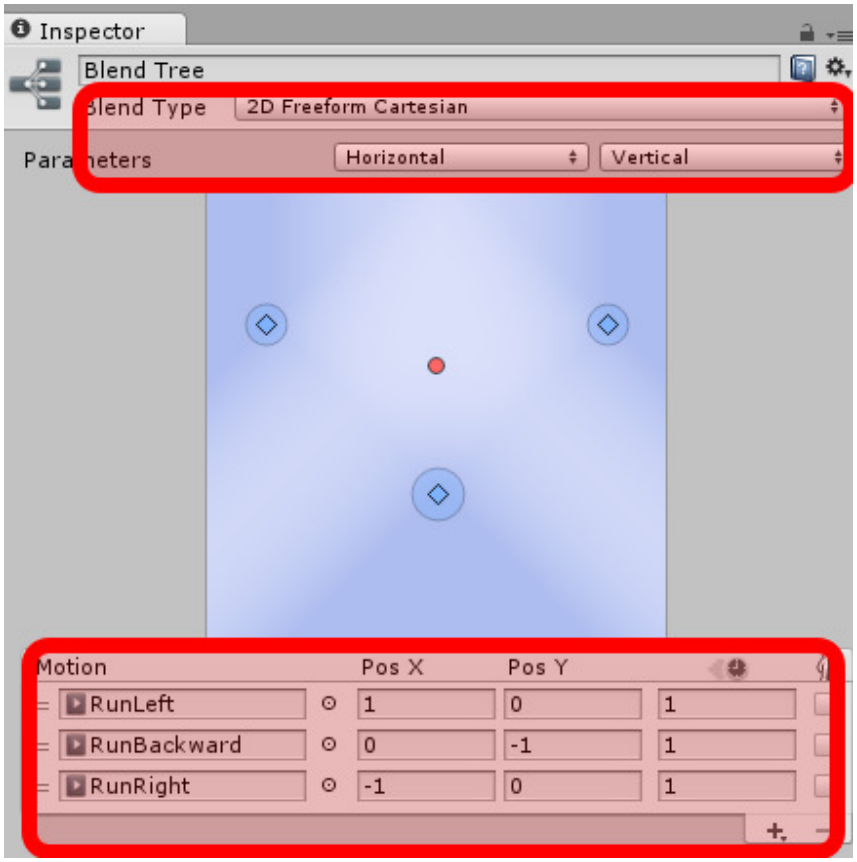


İleri koşma animasyonunu hayranlıkla izlemeniz bittiyse şimdi geri koşma animasyonunu yapmaya başlayalım. Neden hem ileri hem de geri gitme animasyonunu tek bir state'te yapmadık diye haklı olarak sorabilirsiniz. Ben de bunu çok isterdim ama ilginç bir şekilde hem geri hem sağa giderken harmanlanması gereken animasyonlar RunBackward ve RunLeft. Eğer RunBackward ve RunRight animasyonlarını harmanlarsak geri-sola doğru bir animasyon oynuyor. O yüzden ileri koşma ve geri koşma animasyonlarını ayrı birer state olarak oluşturmak durumunda kaldık.

Hâlâ RunForward state'inin içindeyseniz sol üstteki "Base Layer" butonuna basarak ana ekrana dönün:



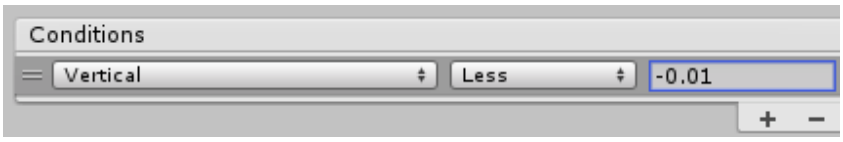
Boş bir yere sağ tıklayıp "Create State-From New Blend Tree" ile yeni bir Blend Tree oluşturun ve ismini "RunBackward" koyun. Çift tıklayarak Blend Tree'ye geçiş yapın. Sonrasında Blend Tree'nin ayarlarını şöyle değiştirin:



Gördüğünüz gibi klavyeden sol-geri ok tuşlarına basılınca geri koşma ve sağa koşma animasyonlarını harmanlarken sağ-geri ok tuşlarına basılınca geri koşma ve sola koşma animasyonlarını harmanlıyoruz. Önizleme penceresini kullanarak animasyonun nasıl durduğuna bakmanız en iyisi.

İster inanın ister inanmayın Mecanim state-machine'ini tamamladık sayılır! Elimizde ihtiyacımız olan tüm state'ler var. Şimdi sadece bu state'ler arası geçişleri (transition) yapmak kaldı.

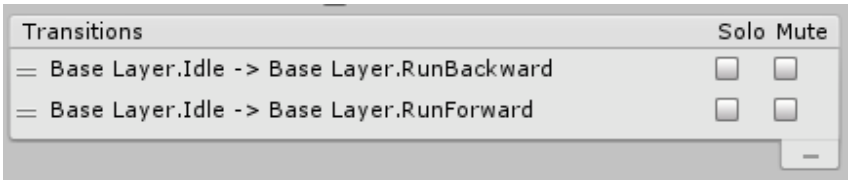
Base Layer'a geri dönün. Idle state'ine sağ tıklayıp "**Make Transition**" deyin ve okun ucunu "RunBackward" state'inde bırakın. Şimdi Idle'dan RunBackward'a giden transition okuna tıklayın. Inspector'da Conditions adında bir kısım var. Oradaki "Exit Time" diye geçen koşulu seçin (biraz soluna doğru tıklamanız gerekebilir) ve sağ alttaki **-** tuşuna basarak bu condition'ı silin. Biz buraya kendi koşulumuzu koyacağız. Bunun için **+** tuşuna basın. Koşulu şöyle değiştirin:



Eğer Vertical değişkeni -0.01'den küçükse (geri ok tuşuna basıyorsak) Idle'dan RunBackward state'ine geçiş yapıyoruz. Bu kadar basit!

Şimdi de Idle'dan RunForward'a giden bir transition yapın. Yine koşullar arasındaki "Exit Time"ı silin ve kendi koşulunuzu şu şekilde ekleyin:

Eğer klavyeden herhangi bir ok tuşuna basarsak Idle'dan RunForward'a geçiyoruz. Peki geri ok tuşuna basınca da geçiş yapmaz mı? Hayır çünkü geri ok tuşuna basınca RunBackward state'ine geçiş yapılmasını az önce özellikle ayarladık. Eğer Idle state'ini seçerseniz şu tablo gelecek:



Burada **Idle'dan çıkan** Transition okları listelenmekte. Gördüğünüz gibi önce Idle'dan RunBackward'a giden Transition'ın koşulu test ediliyor ve eğer o durum sağlanmıyorsa ancak o zaman RunForward'a giden Transition kontrol ediliyor. Yani biz geri ok tuşuna bastığımızda Motion true oluyor ama aynı zamanda Vertical'ın değeri de -0.01'den küçük oluyor ve ilk önce RunBackward koşulu test edildiği için RunBackward state'ine geçiş yapılıyor.

NOT: RunForward'a geçiş yaparken niçin "**Motion == true**" diye test ettik de "**Vertical > 0.01**" diye test etmedik diye sorabilirsiniz. Gerçekten güzel soru. Cevabı şu: RunForward state'ini sadece ileri koşarken (veya ileri-sağa, ileri-sola koşarken) kullanmayacağız, aynı zaman sadece sola koşarken veya sadece sağa koşarken de kullanacağız. Zaten bu yüzden state-machine'mizde "RunRight" ve "RunLeft" gibi bir state yok. Eğer "Vertical > 0.01" yapsaydık RunForward'a sadece ileri ok tuşuna basınca geçilecekti ama "Motion == true" yaptığımız için artık ileri-sol-sağ ok tuşlarından en az birine basınca RunForward state'ine geçiş yapıyoruz.

Idle'dan diğer state'lere geçiş yaptık ama bu, o state'lerden geri Idle'a dönebileceğimiz anlamına gelmiyor. Bu geri dönüş için ayrı bir transition oluşturmak lazım. RunForward'dan Idle'a bir transition oluşturup "ExitTime" koşulunu silin ve yerine "Motion == false" koşulunu ekleyin. Aynı şeyi RunBackward'dan Idle'a bir transition oluşturup onun için de yapın. Böylece bu state'lerden birindeyken tüm ok tuşlarını bıraktığımızda karakter Idle state'ine geçecek ve olduğu yerde durma animasyonunu (Idle) oynatmaya başlayacak.

Tek bir sorunumuz kaldı: diyelim sağ ok tuşuna basıp RunForward state'ine geçtik. Sağ ok tuşundan elimizi çekmeden geri tuşuna bastığımızda RunBackward state'ine geçiş yapıp geri-sağa gitme animasyonunu oynatmamız lazım ama ne RunForward'dan RunBackward'a transition'ımız var ne de RunForward'dan önce Idle'a, sonra Idle'dan RunBackward'a geçebiliriz çünkü Motion'ın değeri false olmadığı için RunForward'dan Idle'a geçiş mümkün değil. Bu sorunu çözmenin en uygun yolu RunForward ve RunBackward state'leri arasında da geçişler (transition) oluşturmak diye düşündüm ben.

RunForward'dan RunBackward'a bir transition oluşturun ve koşul olarak "Vertical < -0.01" (Vertical, Less, -0.01) verin. "Exit Time" koşulunu yine silin, ne olduğunu bilmiyorum ama hiçbir transition'da ona ihtiyacımız yok. Geri ok tuşuna basınca Vertical'ın değeri 0'dan küçük olur ve bu durumda RunForward'dan RunBackward'a geçiyoruz. Tam tersi durumda (geri ok tuşundan eli çekince) Vertical'ın değeri negatif olmaz ve bu durumda da RunForward'a geri gitmemiz lazım. Bunun için ise RunBackward'dan RunForward'a transition yapıp koşulunu "Vertical > -0.01" (Vertical, Greater, -0.01) yapın.

Artık tüm state-machine'imiz tamamlandı. State'ler olsun transition'lar olsun hepsi mantık çerçevesinde uygun duruyor. Geriye Horizontal, Vertical ve Motion değişkenlerinin (parameter) değerlerini script ile ayarlamak kaldı.

“PlayerAnimation” adında yeni bir C# scripti oluşturun. Scripti Player objesine verin (child obje olan 3D modele değil, parent obje olan ve "Look X", "Player Gui" gibi component'leri tutan objeye verin). Sonrasında scripti şu şekilde güncelleyin:

```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  PlayerAnimation : MonoBehaviour
05 {
06       Animator _anim;
07
08       void  Start()
09       {
10            _anim = GetComponentInChildren<Animator>();
11       }
12
13       void  Update()
14       {
15            float  h = Input.GetAxis("Horizontal");
16            float  v = Input.GetAxis("Vertical");
17
18            _anim.SetFloat( "Horizontal", h );
19            _anim.SetFloat( "Vertical", v );
20
21            if( !Mathf.Approximately( h, 0f ) || !Mathf.Approximately( v, 0f ) )
22                 _anim.SetBool("Motion", true);
23            else
24                 _anim.SetBool("Motion", false);
25       }
26 }
27
```

Başlarken child objedeki Mecanim (Animator) component'ini _anim değişkenine veriyoruz. Ardından Update fonksiyonunda SetFloat ile Mecanim'deki "Horizontal" ve "Vertical" değişkenlerine değerlerini veriyoruz. Son olarak eğer herhangi bir ok tuşuna basılıyorsa SetBool ile "Motion"u true, yoksa false yapıyoruz. Mathf.Approximately fonksiyonu, iki float değerin eşit olup olmadığına bakmaya yarar. Neden direkt "**if(h != 0 || v != 0)**" yazmadık diye sorabilirsiniz. Çünkü float sayıları bu şekilde karşılaştırmak yanlıştır. Siz mesela bir float'u 0.5 diye ayarlarsınız ama o bilgisayarda 0.5000001 diye tutulur. Bunun sizinle alakası yok, donanımsal birşey bu. O yüzden **her zaman için iki float sayıyı karşılaştırırken Mathf.Approximately fonksiyonunu kullanın!**

Mecanim'le o kadar uğraştınız. Şimdi derhal oyunu çalıştırın ve animasyonlu karakterinizin tadını çıkarın!

NOT: Animasyonlar arası geçiş mükemmel olmayabilir, şu an elimizden bu kadarı geldi. Eğer dilerseniz Mecanim'le ilgili dökümasyonlara bakarak animasyonları daha iyi bir hale getirmeye çalışabilirsiniz.

Zıplama Animasyonu

Malesef model bir zıplama animasyonuyla birlikte gelmediği için zıplarken özel bir animasyon oynatmayacağız.

Özet Geçecek Olursak...

Bu bölümde azımsanmayacak düzeyde konular işledik. 3D model ve animasyon kullanımından quaternion'lara, rastgele sayı oluşturmadan GetComponentInChildren'a kadar...

Eğer ki Mecanim sistemini ve yaptığımız state'leri, transition'ları az da olsa anladıysanız eli öpülesi insanlarsınız. Çünkü Mecanim state-machine'ini belki daha önce hiç kullanmadınız ve gerçekten Mecanim karmaşık duran bir sistem. Bizim hiç ellemediğimiz tonla ayar vardı daha orada. Ben de o ayarların en azından yarısını bilmiyorum zaten. Ama bu kısıtlı bilgimizle bile istediğimiz gibi bir animasyon sistemi elde etmeyi başardık; bu da Mecanim'in gücünü kanıtlar nitelikte bir durum.