

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

Unity 3D İle Oyun Programlama

Bölüm 1: Unity'e Giriş



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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Unity 3D de Neyin Nesi?

Unity 3D, oyunlar ya da başka interaktif uygulamalar (simülasyon, dijital tasarım vb.) oluşturmak için kullanılan, Unity Technologies'in programladığı bir motordur.

Unity iki farklı sürümde sahiptir: ücretsiz sürüm ve Pro sürümü. Ücretsiz sürümde yaptığınız oyunlardan istediğiniz gibi para kazanabilirsiniz ve Unity Technologies'e komisyon da vermeniz gerekmez. Pro sürümü daha yüksek kaliteli oyunlar yapmak için özel araçlara sahiptir ve ücretlidir.

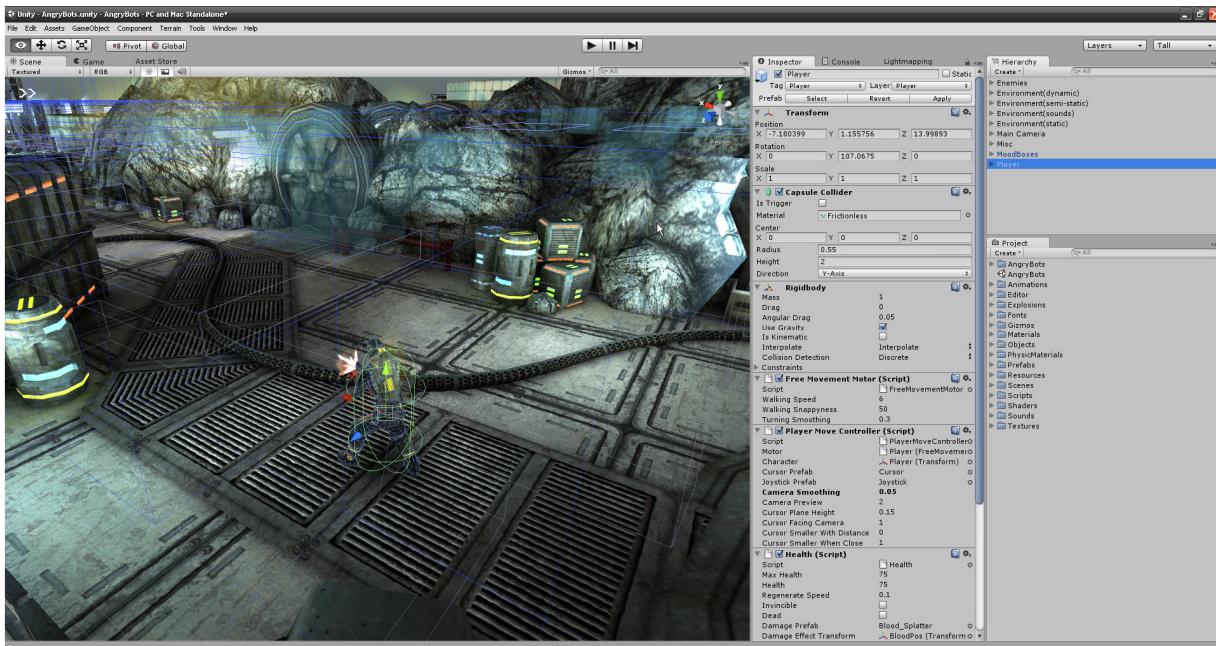
Unity dünya genelinde 500,000+ kişi ve şirket tarafından kullanılmaktadır; örnek vermek gerekirse: Cartoon Network, Coca-Cola, Disney, Electronic Arts, Microsoft, NASA, Amerikan ordusu, Warner Bros ve daha fazlası...

Bilgilendirme

Unity Pro'da Free sürümde olmayan özellikler vardır: post-processing efektler, video oynatma özelliği, dosyaları ve sahneleri stream etme, plugin yüklemek gibi.

Ancak Pro lisansı \$1,500.00 tutmaktadır. Daha fazla bilgi için: <http://unity3d.com/unity/licenses>

Ama canınızı sıkmayın. Ücretsiz sürümde de gayet eğlenceli oyunlar oluşturabilirsiniz!



Görsel 1.1: Unity 3D arayüzü

Unity 3D'yi Nereden Edinebilirim?

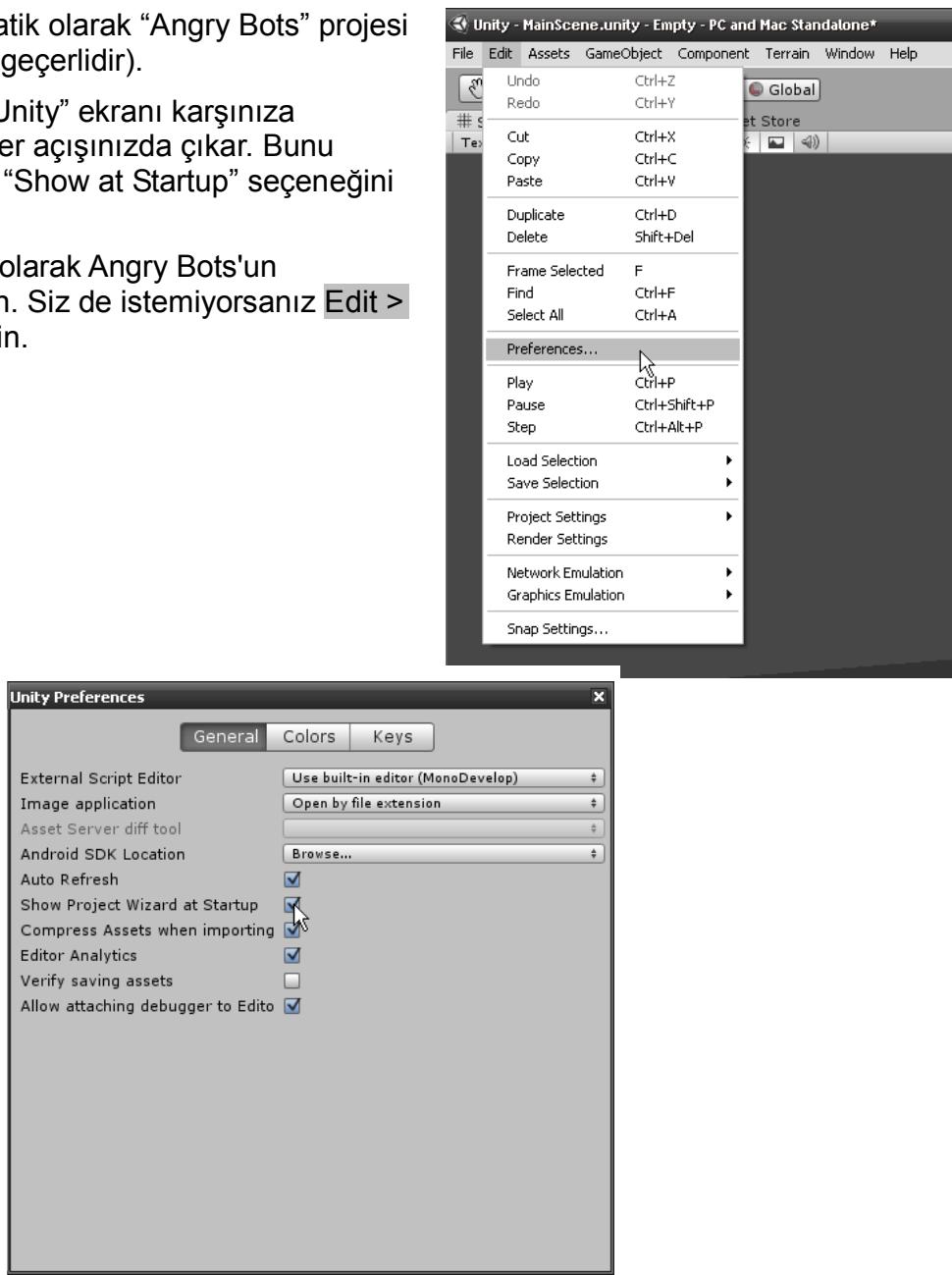
Unity'nin son sürümünü şu adreste bulabilirsiniz: <http://unity3d.com/unity/download/>. İnternete bağlıken Unity'i her açığınızda yeni bir sürüm çıkmış mı diye otomatik olarak kontrol yapılmaktadır.

Unity'e Giriş

Unity'i ilk açığınızda otomatik olarak "Angry Bots" projesi açılacaktır (versiyon 3.4'te geçerlidir).

Ayrıca belki "Welcome To Unity" ekranı karşınıza gelebilir. Bu ekran Unity'i her açığınızda çıkar. Bunu engellemek için sağ alttaki "Show at Startup" seçeneğini kapatın.

Unity açıldığında otomatik olarak Angry Bots'un açılmasını istemiyorum ben. Siz de istemiyorsanız **Edit > Preferences...** yolunu izleyin.



Görsel 1.2: Preferences penceresi.

Buradan "Show Project Wizard at Startup"ı işaretleyin.

Unity'i sonraki açışlarınızda hangi projeyi açmak istediğiniz soran bir pencere gelecek.

Şimdi şu linkteki LessonAssets.zip dosyasını indirin:

<https://dl.dropboxusercontent.com/u/25260770/UnityLessons/LessonAssets.zip?dl>

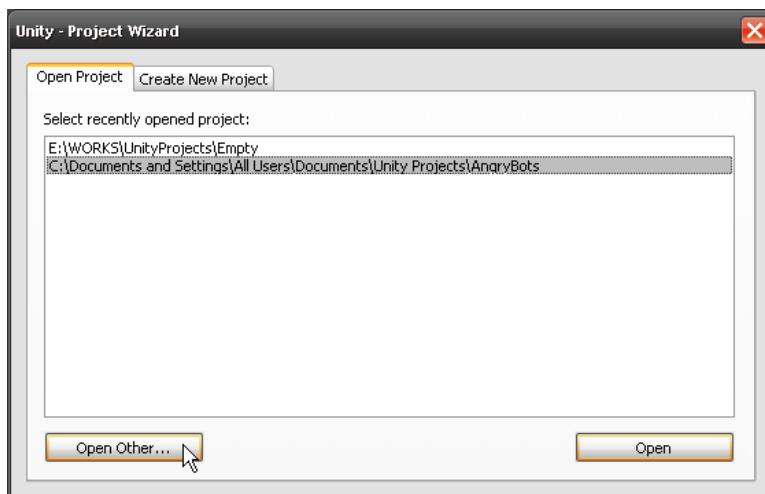
Bize gerekli olan herşey bu arşivde mevcut. Bir de Lesson 1 adında bir klasör mevcut. Bu klasörde ilk projemiz yer alıyor. Şimdi onu beraber açacağız.

Bir Unity projesini açmak için izleyebileceğiniz iki yol var:

Projeyi Unity'den Açımak

Unity'de File > Open Project... yolunu izleyin. Bir pencere gelecek.

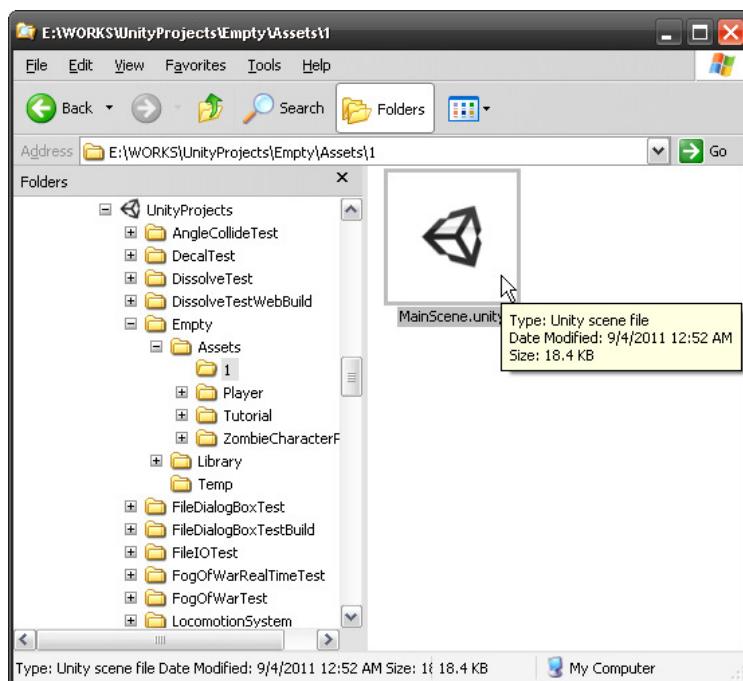
Buradan “Open Other...” butonuna tıklayın ve dosya gezgininde Lesson 1 klasörünün olduğu konuma gelin. Bu konumdayken Lesson 1 klasörüne bir kez tıklayarak onu işaretleyin ve OK butonuna tıklayın. Yani bir proje açarken projenin olduğu ana klasörü seçeceksiniz. Şimdi Unity kapanıp tekrar açılacak. Unity'i sonraki açışlarınızda projenizi listeden seçebilirsiniz:



Görsel 1.3: Project Wizard penceresi.

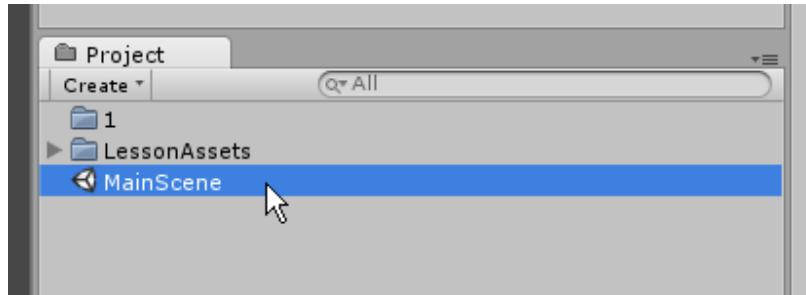
Projeyi İşletim Sistemi Üzerinden Açımak

Windows Explorer (Windows)'ı açın. Unity projenizin olduğu klasörü açın. Orada “Assets” ve “Library” adında iki klasör var. “Assets”i açın. Eğer Lesson 1 projesini açıyorsanız “MainScene.unity” adında bir dosya göreceksiniz. **Dosyaya çift tıklayın**. Proje Unity'de açılacak.



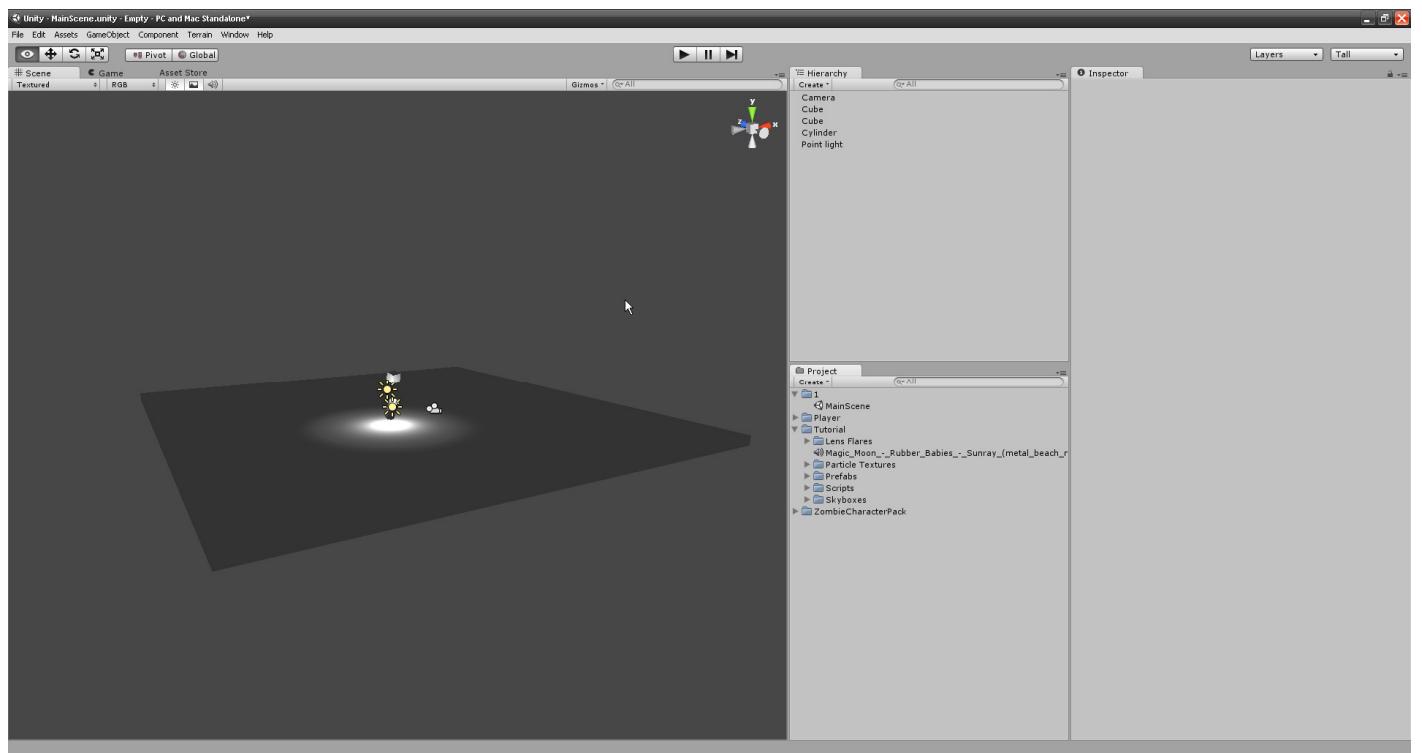
Unity Arayüzüne Tanıma

Lesson 1 projesini açtıktan sonra Project panelindeki MainScene dosyasına çift tıklayın:

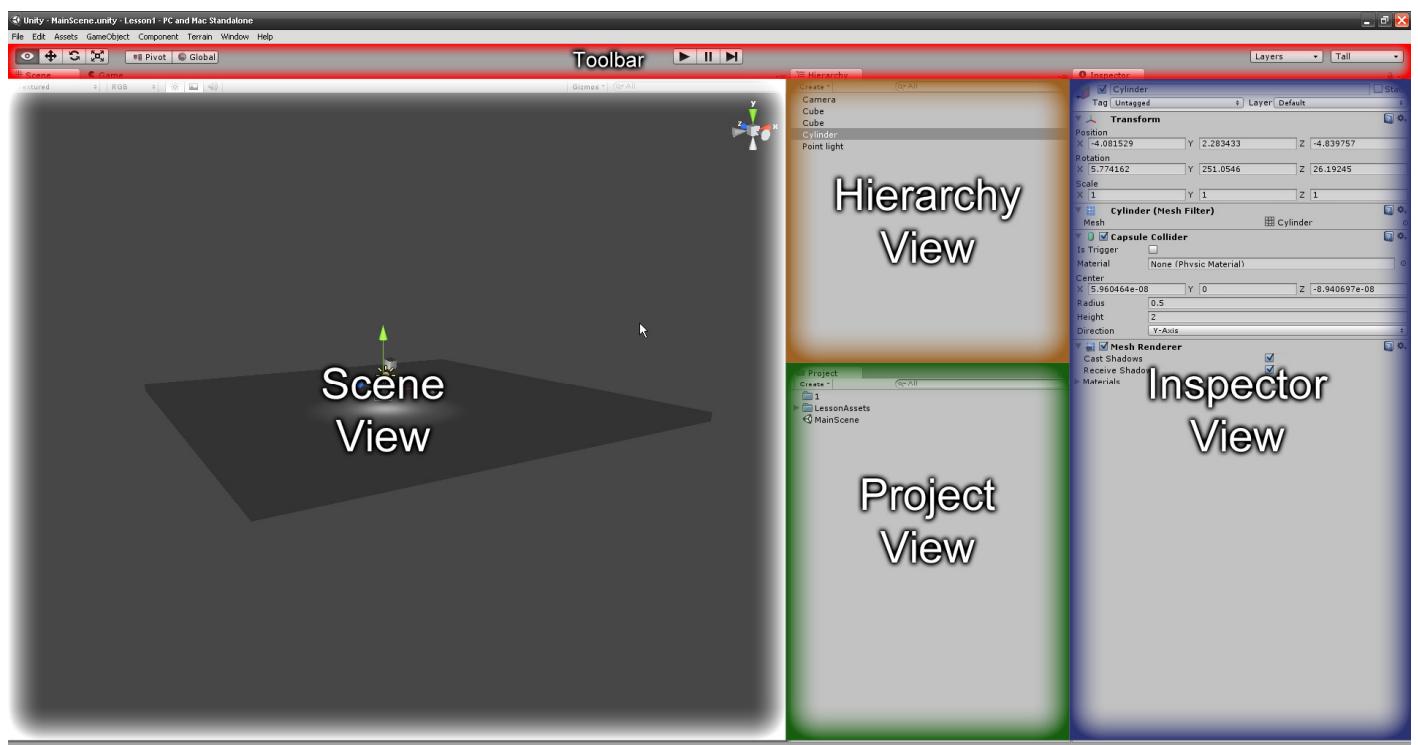
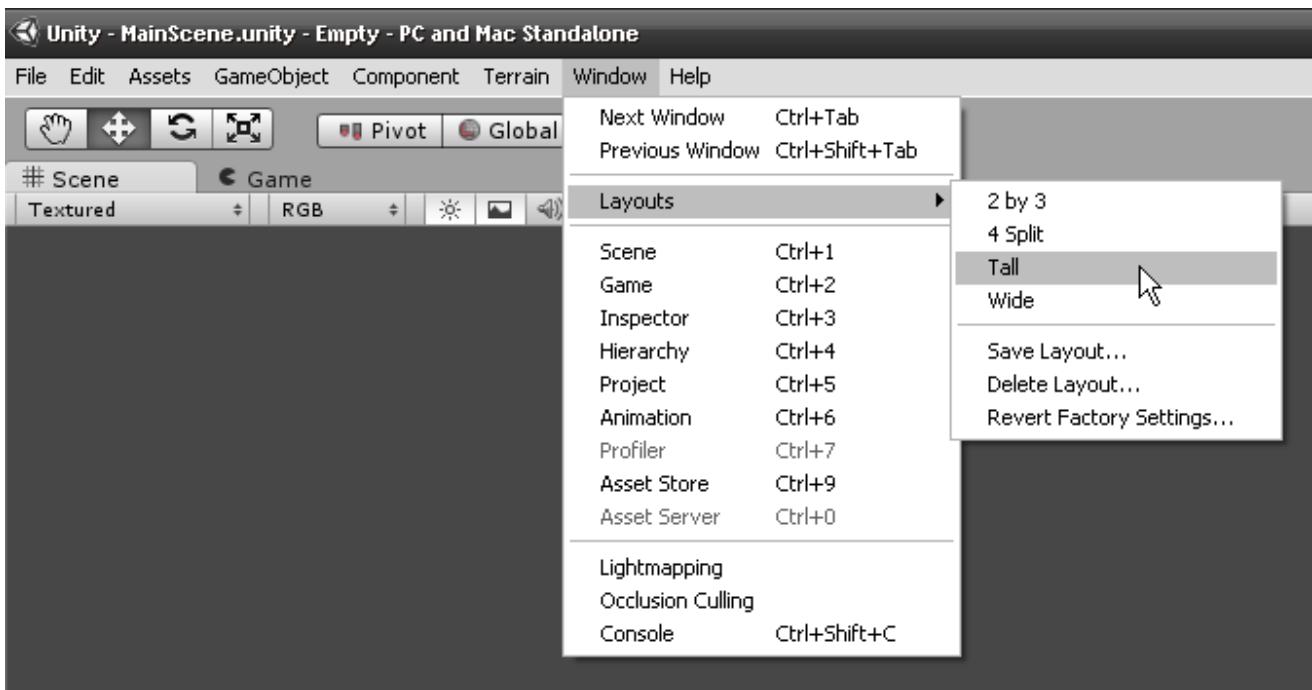


ÇEVİRMEN EKLEMESİ: Unity'nin yeni sürümüyle beraber fark edeceğiniz üzere Project panelinin tipi değişti. Ben resimde gördüğünüz eski Project panelini daha çok seviyorum. Ayrıca ders boyunca da eski tip Project paneli ile çalışıyoruz. Siz de Project panelinin eski tipine dönmek için panelin sağ üstündeki garip ikona tıklayın ve "One Column Layout"u seçin.

Editör ekranı şuna benzer duracaktır:



Arayüzdeki bileşenlerin yerleri sizde farklı olabilir. Bunu düzeltmek için [Window > Layouts > Tall](#) yolunu izleyin. Artık hepimiz biriz!



Unity arayüzü 5 panelden oluşmaktadır:

1. **Scene Paneli:** Oyun alanını (sahne)(scene) 3 boyutlu görmeye yarar. **Unity'de her level bir "scene" olarak adlandırılır.**

2. **Hierarchy Paneli:** Sahnedeki tüm objelerin listelendiği panel. **Unity'de objelere "game object" denmektedir.**
3. **Inspector Paneli:** Seçili obje hakkında detaylı bilgiler burada yer alır.
4. **Project Paneli:** Oyununuzda kullanabileceğiniz tüm kaynak dosyaları burada yer alır. Bunlar 3D objelerden tutun resimlere, ses dosyalarına, fontlara kadar pek çok dosya olabilir.
5. **Araç Çubuğu:** Projenizle etkileşime girmek için çeşitli butonlar.

Sahnedeki (scene) Gezinmek

Her şeyden önce kontrollere aşina olmalısınız.

İmleci Scene paneline götürün. **Orta mouse tuşuna basılı tutun ve fareyi kımıldatın.** Bu işlem kamerayı hareket ettirmeye yarar.

Klavyeden Alt tuşuna ve sol mouse tuşuna basılı tutup fareyi kımıldatın. Bu işlem kamerayı döndürür.

Mouse tekerleği ile ya da Alt+sağ mouse tuşu kombinasyonu ile de kameraya zoom yapabilirsiniz. Şimdi kontrollere iyice alışana kadar pratik yapın.

Bir Objeye Odaklanmak

Hierarchy panelinden "Simple Cylinder" isimli objeye tıklayarak onu seçin. Şimdi imleci Scene paneline hareket ettirin. Klavyeden **F tuşuna basın.** Kamera seçili objeye odaklanacak (focus).

Kamera bir objeye odaklıken onu döndürürseniz (Alt+sol mouse tuşu) kamera objenin etrafında döner. Kamerayı başka yere hareket ettirirseniz artık o objenin etrafında dönmez.

Oyun Alanında Serbestçe Gezinmek

Scene panelinde kameramızla etrafi gezinmenin bir başka yolu da uçarak gezmek. Sağ mouse tuşu ile Scene paneline basılı tutun ve W-A-S-D tuşlarıyla kamerayı hareket ettirin. Q-E tuşlarıyla kamerayı aşağı-yukarı hareket ettirebilirsiniz. Bu esnada fareyi kımıldatarak kamerayı döndürebilirsiniz.

Objelere (Game Object) Giriş

Unity'de oyun alanında gördüğünüz her şey birer game object'tır. Bunlara oyundaki karakterimiz (player), düşmanlar, zemin, duvarlar, ışıklar, arayüzdeki butonlar örnek verilebilir.

Game object'ler ile oluşturduğunuz sahnelere scene denir. Oyununuzda birden çok scene olabilir. Bir scene'de ana menü, öteki scene'lerde oyundaki level'lar yer alabilir.

Game Object Oluşturmak

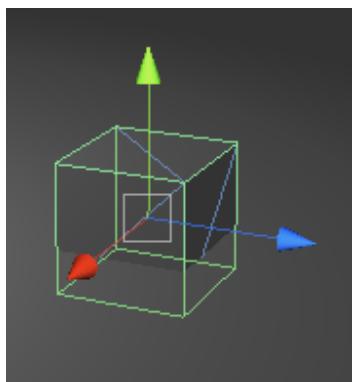
Yukarıdan **GameObject > Create Other > Cube** yolunu izleyin. Kamera görüş alanınızın tam ortasında bir küp olusacak. Küp haricinde oluşturabileceğiniz başka hazır modeller de vardı o menüde ama biz şimdilik küp objesine odaklanacağız.

Objeyi Hareket Ettirmek (Move)(Translate)

Küp objesine tıklayarak onu seçin ve F tuşuna basarak kameralanın seçtiğiniz küp objesine odaklanması sağlayın. Objenin etrafında kırmızı, yeşil ve mavi renkte üç ok göreceksiniz (eğer yoksa W tuşuna basın ya da araç çubuğundan ilgili ikona tıklayın).



Görsel 1.4: Hareket Ettirme Tool'u.



Bu oklar objeyi hareket ettirmeye yarar. Kırmızı ok sağa-sola, yeşil ok yukarı-aşağı, mavi ok ileri-geri oynatmaya yarar.

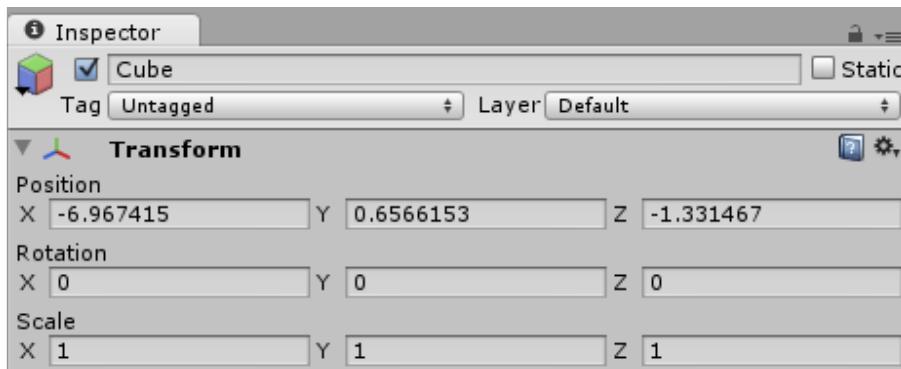
Yapmanız gereken objeyi bu oklardan birinden tutup sürükleyerek istediğiniz konuma getirmek.

Objeyi serbestçe hareket ettirmek için okların merkezindeki beyaz kutucuktan tutarak sürükleyin.

Bilgilendirme

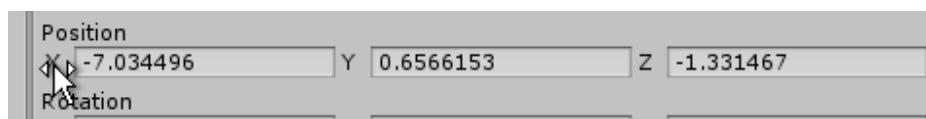
Unity'de mesafeler varsayılan olarak herhangi bir birimle ölçülmemektedir ancak fizik motoru bir birimi bir metre olarak hesaba katmaktadır.

Objeyi hareket ettirirken Inspector'daki değerlerin değiştiğine dikkat ettiniz mi? Bu değerler objenin 3 boyutlu uzaydaki konumunu belirliyor. Bu kutucuklarda değerleri elle istediğiniz gibi değiştirebilirsiniz de...



Görsel 1.5: Inspector paneli objenin x, y, z pozisyonunu ve bazı diğer bilgileri göstermektedir.

X, Y ve Z değerlerini mouse ile de değiştirebilirsiniz. Örneğin imleci X harfinin üzerine getirin ve sol mouse tuşuna basılı tutarak fareyi hareket ettirin.



Objeleri Döndürmek (Rotate)

Bir obje seçin ve E tuşuna basın (ya da araç çubuğundan ilgili ikona tıklayın).

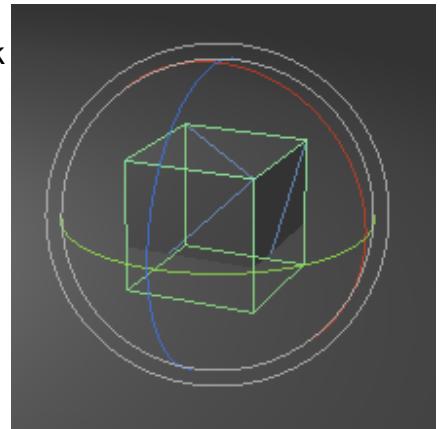


Görsel 1.6: Döndürme (Rotate) Tool'u.

Objeyi çevreleyen üç daire göreceksiniz. İşleyiş hareket ettirme tool'yla aynı. Bu dairelerden birine basılı tutup fareyi hareket ettirerek objeyi o yönde çevirebilirsiniz.

En dıştaki beyaz daireden tutup sürüklerseniz obje kameranın bakış açısı yönünde döner. Test edin ve görün.

Döndürme işlemi yaparken Inspector panelinde Rotation değerlerinin değiştiğini görebilirsiniz. Buradaki X, Y ve Z değerleri birer Euler açı (Euler angle)'dır.

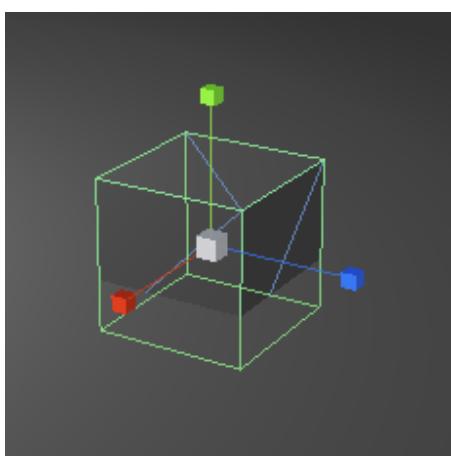


Objeyi Boyutlandırmak (Scale)

Bir objeyi seçin ve R tuşuna basın (ya da araç çubuğundan ilgili ikona tıklayın).



Görsel 1.7: Boyutlandırma Tool'u.



Boyutlandırma aracı seçiliyken objenin merkezinden dışarı doğru üç ok çıkar. Ama bu sefer okların ucunda küp simbolü yer alır.

Önceden yaptığınız gibi bu oklardan tutup sürükleyerek objeyi boyutlandırabilirsiniz. Merkezdeki beyaz küpten tutup sürükleyerek objeyi her yönde eşit olarak boyutlandırabilirsiniz.

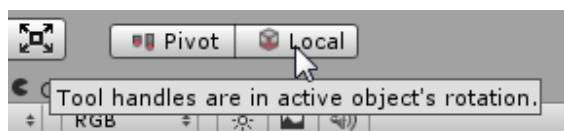
Global Uzay (World Space) ve Yerel Uzay (Local Space)

Bu sahada iki farklı space'ten bahsetmeyi uygun buldum.

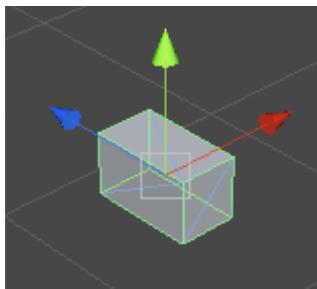
Bir objeyi döndürdüğünüzde (rotate) objenin yönünü değiştirmiş oluyorsunuz. Objeye göre "ileri" yön, objeyle beraber dönmüş oluyor.

Diyelim ki bir objeyi Y ekseni etrafında (yeşil eksen) 45 derece döndürdünüz. Şimdi obje düz değil çapraz duruyor. Objeyi bu halde hareket ettirmek istediğinizde bazı okların yönünün değiştigiğini farkedeceksiniz. Çünkü objenin kendisinin yönü değişti. Okların yönünün objenin yönüne göre değiştiği bu uzaya Local Space deniyor. Okların yönünün hep sabit olduğu uzaya ise World Space deniyor (bu okların yönü Scene panelinin sağ üstündeki gizmo'daki oklarla aynıdır).

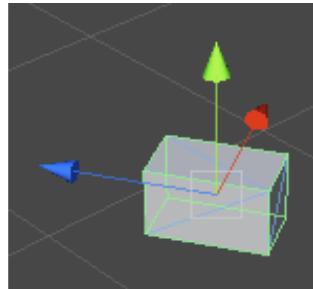
Bir objeye Local Space üzerinden işlem yapmak için araç çubuğundaki ilgili kısmın Local olduğundan emin olun:



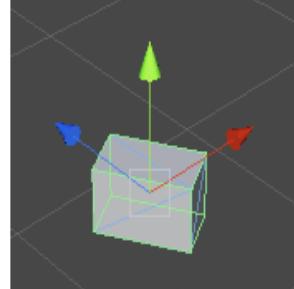
Bu buton Global ve Local uzay arasında geçiş yapmaya yarar.



Görsel 1.8:
Döndürülmemiş bir obje.
Mavi okun yönüne dikkat
edin.



Görsel 1.8: Objeyi
döndürince mavi okun
yönü de değişti. Mavi ok
Local space'te objenin
"ileri" yönünü temsil
etmektedir.



Görsel 1.10: Eğer araç
çubuğundan (toolbar)
World Space'e geçiş
yaparsak oklar hep sabit
bir yöne bakacaktır.

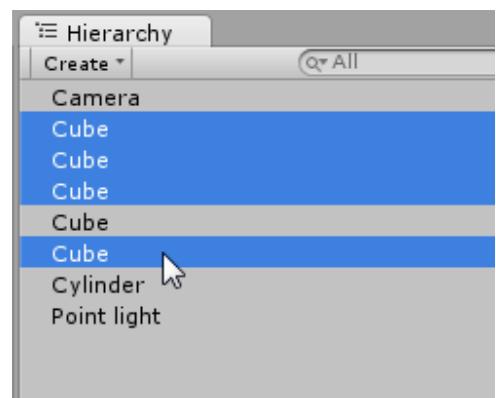
Snapping (Kenetleme)

Objeyi konumlandırırken, döndürürken ve boyutlandırırken Ctrl tuşuna basılı tutarsanız bu işlemler birim birim gerçekleşir (anlamanın tek yolu kendi başınıza denemek).

Birden Çok Objeyi Seçmek

Scene panelinde Ctrl veya Shift tuşuna basılı tutarak birden çok obje seçebilirsiniz. Ardından seçili objeleri aynı anda konumlandırılabilirsiniz.

Hierarchy panelinde de benzer şekilde Ctrl veya Shift tuşuna basılı tutarak çoklu seçim yapabilirsiniz.



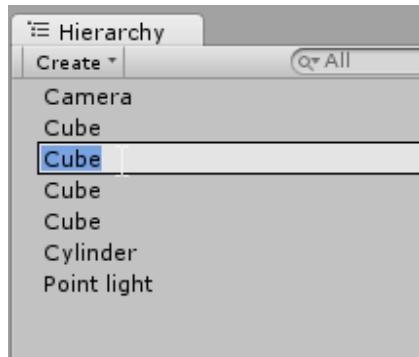
Görsel 1.9: Birden çok küp (Cube)
objesinin seçilmiş hali.

Bir Objenin İsmini Değiştirmek

Bunu yapmanın iki yolu var.

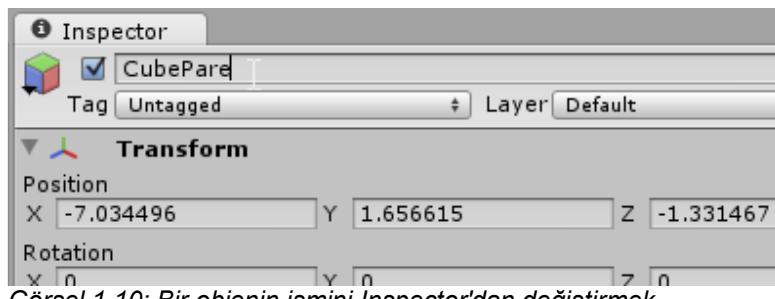
Hierarchy Panelini Kullanmak

Objeyi Hierarchy panelinden seçin ve F2 tuşuna basın (Mac'te Enter tuşuna). Yeni bir isim verip Enter tuşıyla işlemi sonlandırın.



Inspector Panelini Kullanmak

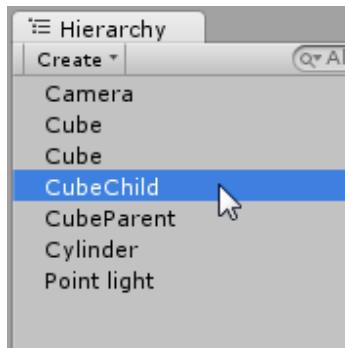
Objeyi Hierarchy ya da Scene panelinden seçin. Inspector panelinin tepesindeki ismi değiştirin ve Enter tuşıyla işlemi sonlandırın.



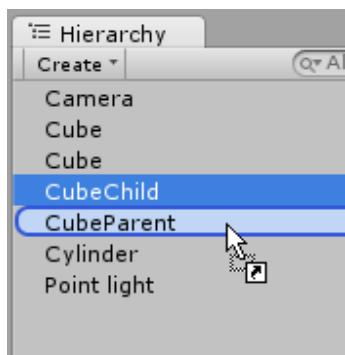
Görsel 1.10: Bir objenin ismini Inspector'dan değiştirmek.

Bir Objeye Ebeveyn Atamak (Parenting)

Birden çok objenin gruplandırılmasına Unity'de parenting denir.



1. Başka bir objenin içinde gruplandırmak istediğiniz objeyi seçin. Seçtiğiniz bu objeye child obje denir.

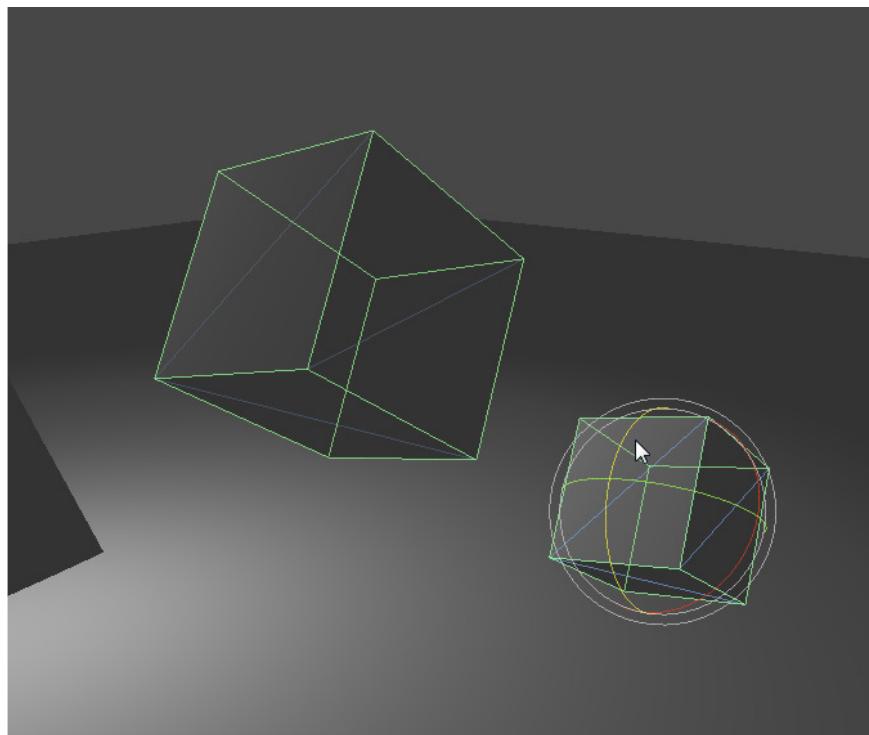


2. Seçili objeyi gruplandırmak istediğiniz objenin üzerine sürükleyin. Üzerine sürüklediğiniz objeye parent (ebeveyn) obje denir.



3. Başarılı bir parenting operasyonu gerçekleştirdiniz! Oluşan grubu Hierarchy panelinde fark edebilirsiniz.

Bir gruptaki parent objeye uygulanan konumlandırma, döndürme ve boyutlandırma işlemleri aynı zamanda child objelere de uygulanır.



Görsel 1.11: Child obje parent objeye yapılan değişiklikleri taklit etmekte.

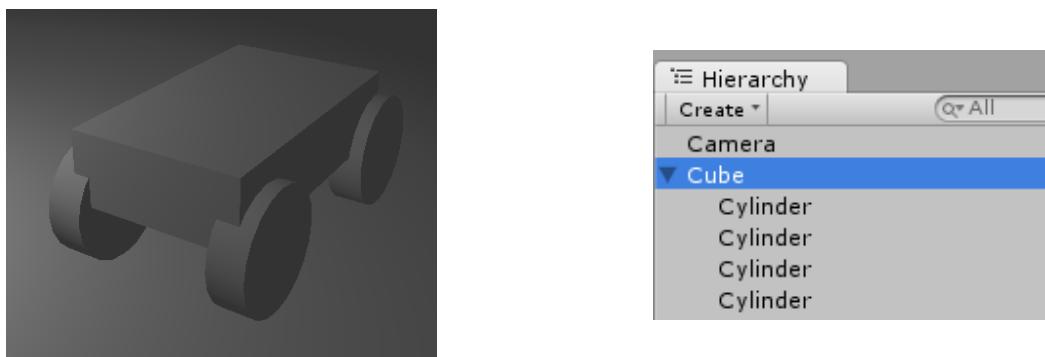
Bir Objeyi Çoğaltmak (Klonlamak)

Objeyi seçin ve Ctrl+D kombinasyonunu uygulayın. Klon obje ile ana objenin özellikleri birebir aynı olacaktır.

Klonlanan obje ana objenin üzerinde yer alır. İkisini birbirinden ayırmak için tekini biraz kımıldatın.

Kendinizi Test Edin

Alttaki resimde gördüğünüz gibi bir obje oluşturmaya çalışın. Ardından silindir objelerini küp objesi ile parent'layın.



Component Terimine Giriş

Unity'de objelere pek çok özellik kazandırabilirsiniz. Kazandırdığınız hemen her özelliğe birer component denmektedir. **Component'ler bir objenin ne olduğunu belirleyen parçalardır: objenin nelere sahip olduğunu, nasıl davranışacağını belirlerler.**

Şu ana kadar bir component ile çoktan tanıştinız: "Transform" componenti. Bu component objenin uzaydaki konumunu (position), eğimini (rotation) ve boyutunu (scale) depolar.

Bir obje seçtiğinizde **Inspector paneli** o objenin tüm component'lerini listeler.

Bir 3D objeyi sahnede render'lamak için bir component gereklidir. Bu objenin şeklini fizik motoruna tanıtmak için ise yine başka bir component gerekmektedir.

"Mesh Filter" componenti, game object'in şeklini aldığı 3D modeli belirlemeye yarar. "Mesh Renderer" bu game object'i ekrana çizdirmeye (render) yarar. "Box Collider" componenti ise bu objenin şeklini fizik motoruna tanıtmaya yarar (Box Collider kullanırsanız şekli fizik motoruna bir dikdörtgenler prizması olarak tanıtırırsınız).

Fiziksel Etkileşimler: Rigidbody Componenti

Siz Deneyin: Lesson 1 projesinde yer alan "Simple Cylinder" objesini seçin. F tuşu ile ona odaklanın.

Objeye **Component > Physics > Rigidbody** yolunu izleyerek Rigidbody componenti verin.



Görsel 1.12: Play butonu basılıyken.

Araç çubuğunda şekildeki gibi üç buton göreceksiniz. Bu butonlar oyunu editörde test etmeye yarar.

En soldaki buton oyunu başlatır. Ortadaki duraklatır. En sağdaki buton da duraklatılmış oyunu kare kare (frame by frame) ilerletmeye yarar.

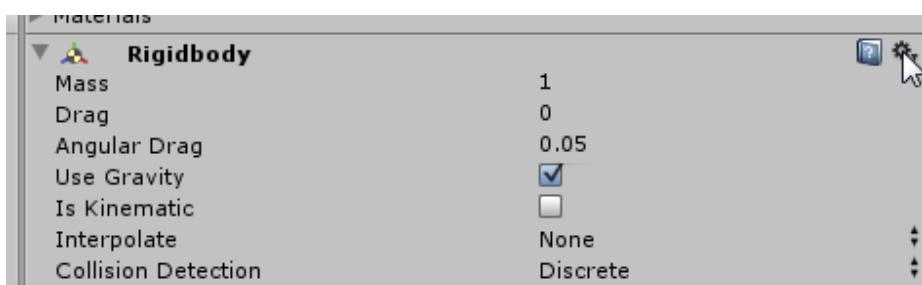
Durmayın ve soldaki butona basarak oyunu başlatın.

Eğer her şeyi düzgün yaptıysanız oyun başladığında silindirin yere düştüğünü göreceksiniz. **İste rigidbody componenti bu işe yarar: objeye fiziksel kuvvetlerin etki etmesini sağlar.**

Play butonuna (soldaki buton) tekrar basarak oyunu durdurun.

Bir Componenti Silmek

Şimdi rigidbody'i objeden silince ne olacağına bakalım. Bir componenti silmek için Inspector panelinden componentin sağ üstündeki dişli ikona tıklayın:

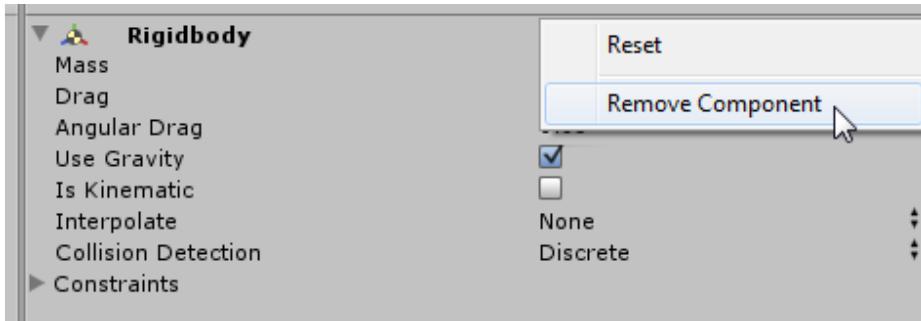


Bilgilendirme

Bir objenin şeklini fizik motoruna tanıtmak genelde görünür şekil tanıtmaz. Çünkü çok poligonlu bir şekil için bu fizik motorunun çok fazla işlem yapması demektir.

Onun yerine basit şekiller kullanılır: küp ya da kapsül gibi. Bu basit şeklin ebatlarıyla oynanarak asıl şeklin etrafını sıkıca kaplaması sağlanır. Sonuç çok poligonlu şekil kendisini fizik motoruna tanıtmak kadar tutarlı olmayabilir ama işlemciye daha az yük bineceği barizdir.

Gelen menüden “Remove Component” seçeneğini seçin.



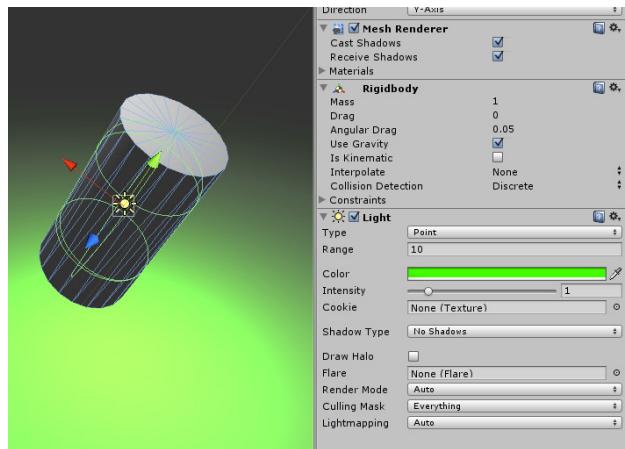
Görsel 1.13: Bir componenti objeden silmek (atmak)

Şimdi tekrar oyunu çalıştırın. Silindirin havada kaldığını göreceksiniz. Artık rigidbody silindirde yer almadığı için silindire fiziksel kuvvetler etki etmiyor (yerçekimi kuvveti gibi).

Light Componenti

Silindir seçili iken **Component > Rendering > Light** yolunu izleyerek ona Light componenti verin. Bu componenti Inspector'dan incelerseniz çeşitli ayarlar göreceksiniz.

Color'ın sağındaki yere tıklayın. Renk paleti gelecek. Buradan istediğiniz rengi seçin. Rengi değiştirdikçe Scene panelindeki görüntünün de değiştiğini görebilirisiniz.



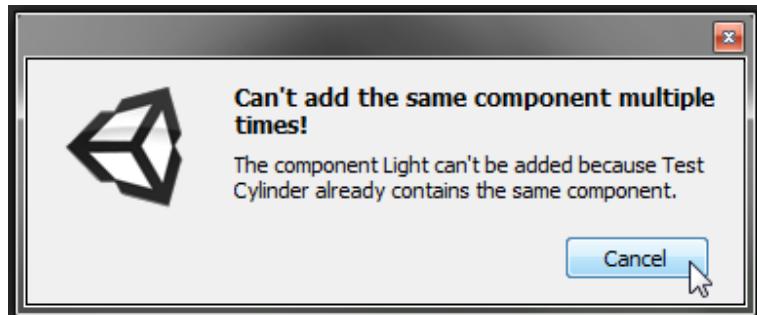
Görsel 1.14: Rengi değiştirdikçe editördeki görüntüyü gerçek zamanlı olarak güncellenecektir.

Silindire tekrar bir rigidbody eklerseniz artık silindirde hem rigidbody hem de light componenti olduğunu göreceksiniz. Oyunu çalıştırınca obje hem yere düşecek hem de etrafına ışık saçacak.

Unity'nin kullandığı component'lere dayalı dizaynın avantajı burada yatıyor. Herhangi bir objeye herhangi bir component kombinasyonunu uygulayarak objenin istediğiniz gibi davranışını sağlayabilirsiniz.

Birden Çok Light Ekleme

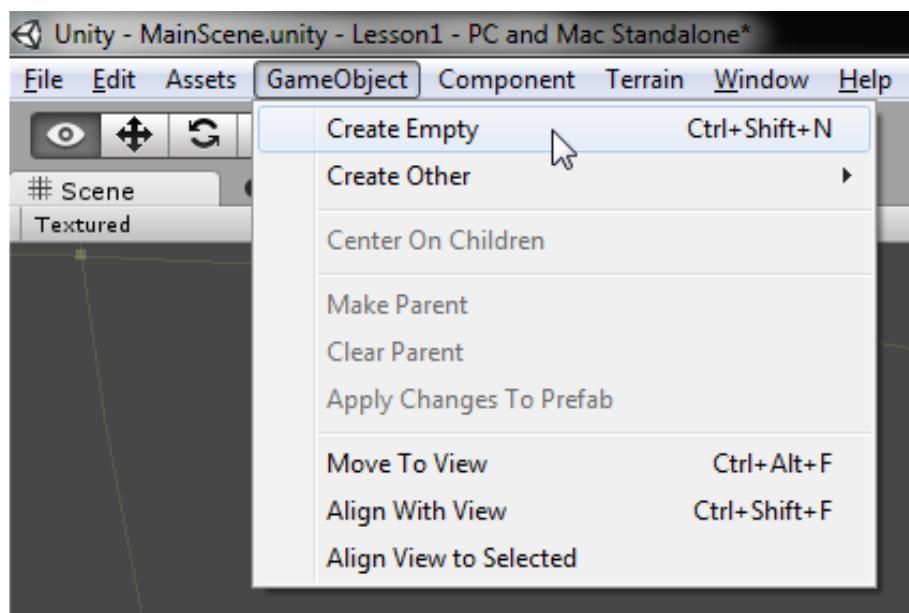
Ya silindire iki Light componenti eklemek istersek? Durmayın ve deneyin. Malesef bir hata alacaksınız:



Bir objeye en çok bir tane Light componenti ekleyebilirsiniz. Çoğu component için bu durum geçerlidir. Tıpkı hata mesajının dediği gibi; **aynı component'ten tek bir objeye birden çok ekleyemezsiniz.**

Bu sorunu nasıl çözeceğiz? **İkinci Light componentini depolayacak olan yeni bir obje oluşturacağız. Bu obje görünmez olacak ve bu objeye parent olarak silindir objesini vereceğiz.** Silindir objesine (veya herhangi başka objeye) istediğiniz kadar child obje verebilirsiniz.

Unity'de görünmez obje oluşturmak için **GameObject > Create Empty** yolunu izleyin ya da **Ctrl+Shift+N** kombinasyonunu uygulayın.



Görsel 1.15: Menü barından boş bir obje (Empty GameObject) oluşturmak.

Özet Geçecek Olursak...

Buraya kadar geldiyseniz tebrikler! Unity ile kendi oyunlarınızı oluşturmak için ilk adımı attınız. Unity'de bir proje açmayı, objeleri konumlandırmayı, component terimini ve görünmez bir obje oluşturmayı gördünüz.

Sonraki derslerde daha çok component göreceksiniz: terrain (arazi), partikül efektleri, lens flare ve müzik gibi.

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

Unity 3D İle Oyun Programlama

Bölüm 2: Basit Bir Level Tasarımı



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



All code snippets are licensed under CC0 (public domain)



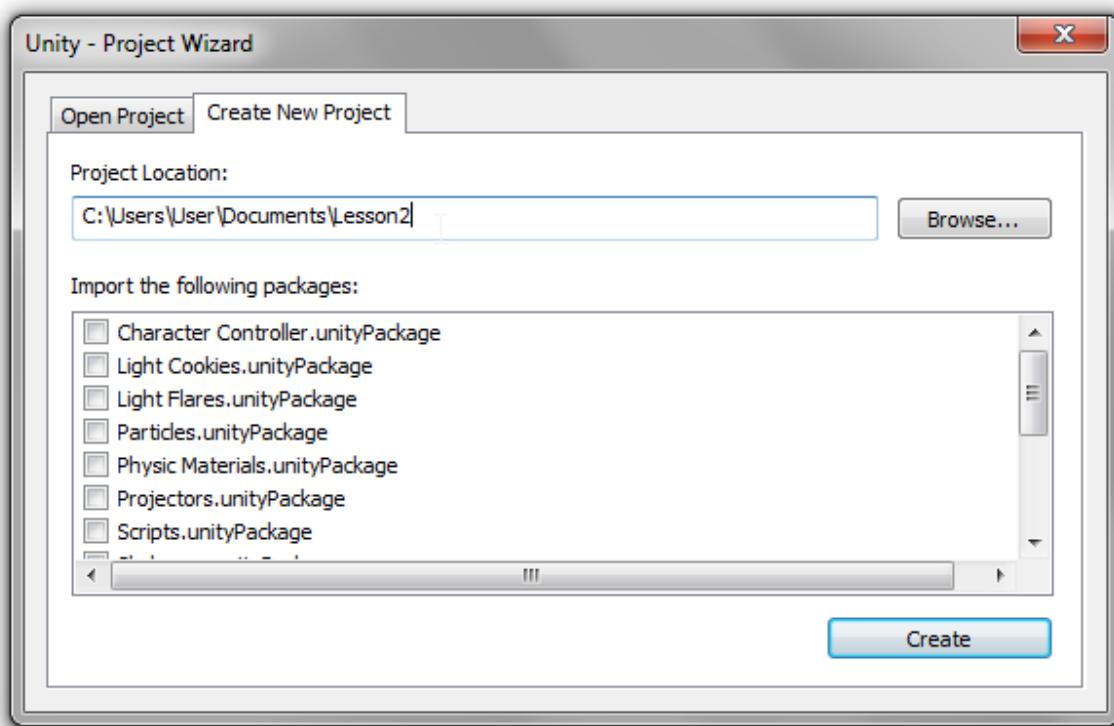
This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Bu bölümde oyuncunun üzerinde yürüyeceği basit bir level tasarlayacağız. Oyunların vazgeçilmezi bazı öğeler kullanacağınız: ağaçlar, tepeler, kamp ateşi ve güzel bir müzik gibi. Bu saylıklarımı yapmak Unity'de çok kolay. Bunlar için kod yazmanız gereklidir.

Zemini Oluşturmak

Öncelikle karakterin üzerinde yürüyeceği zemini oluşturalım.

Unity'i başlatın. File-New Project yolunu izleyin. Harddiskinizin istediğiniz yerinde proje için "Lesson2" adında yeni bir klasör oluşturun ve Browse butonu ile bu klasörü seçin:



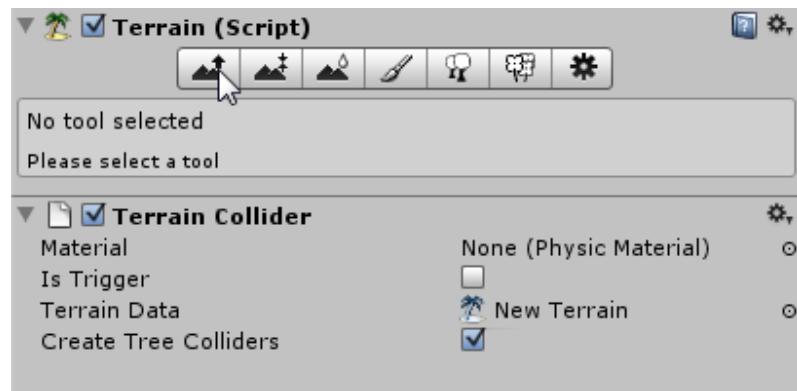
Create butonuna basın. Boş bir sahneyle karşılaşacaksınız.

Şimdi **GameObject > Create Other > Terrain** yolunu izleyerek bir arazi objesi (terrain) oluşturun. Arazi sahnenizde belirecektir.

Tepeler Yapmak: Raise Terrain Aracı

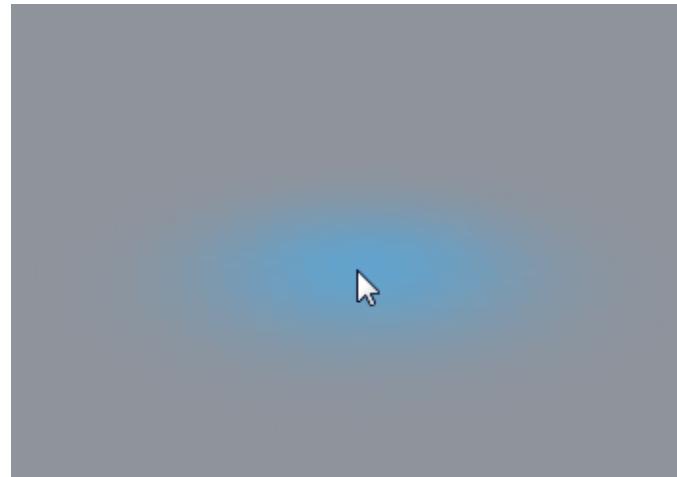
Inspector'da Terrain adında yeni bir component ile karşılaşacaksınız.

Bu component'te yanyana dizilmiş bir takım butonlar bulunmaktadır. Bunlar vasıtasyyla arazimizi şekillendirebiliyoruz.

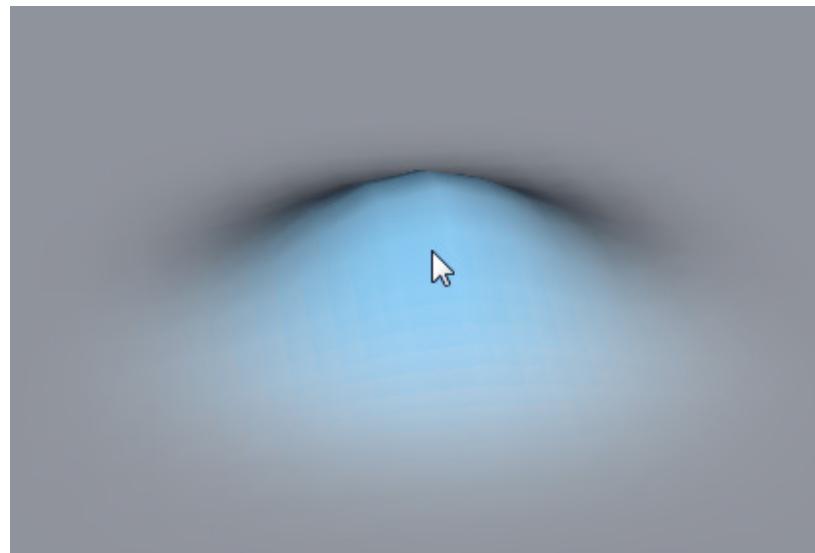


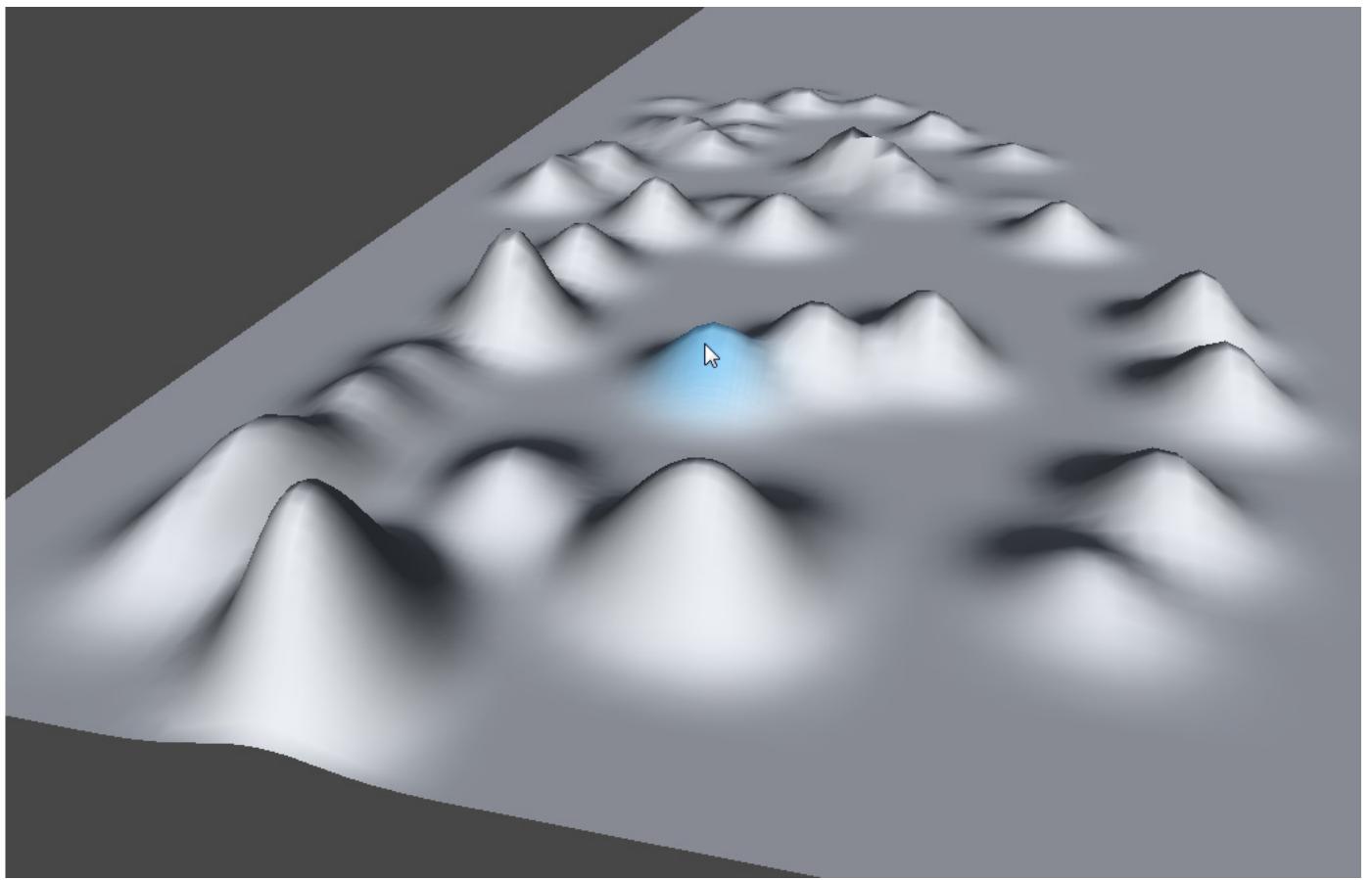
En soldaki butona tıklayın. Bunun adı Raise/Lower Terrain aracı. İmleci sahnedeki düz arazinin üzerine getirin. İmlecin olduğu yerde mavi bir parıldama göreceksiniz. Bu parıldama fırçamızın boyutunu temsil ediyor.

NOT: Kamerayı biraz araziden uzaklaştırmanız gerekebilir.



Arazide bir yere sol tıklayın. Bir yükselti oluşacak.

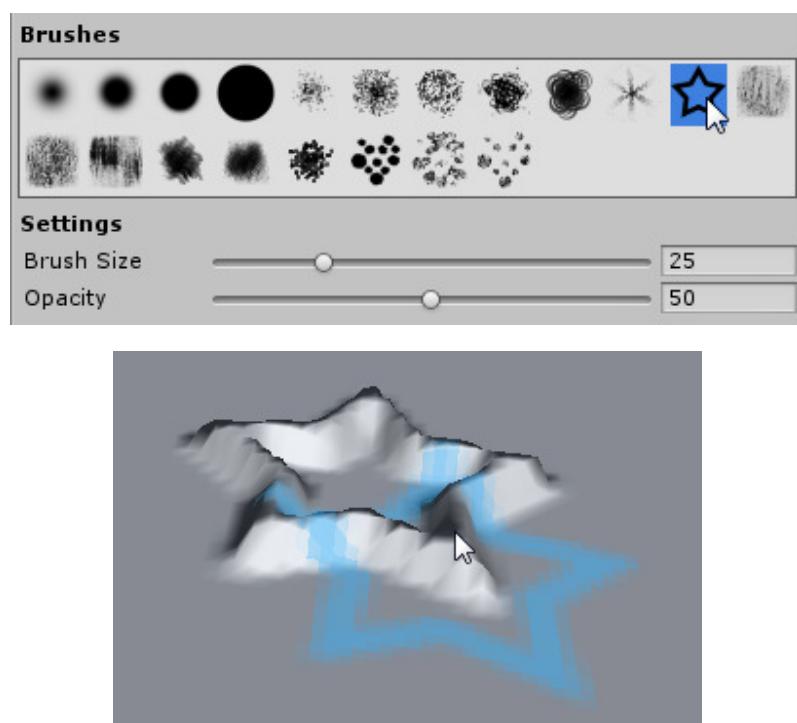




Görsel 2.1: Raise Terrain aracını birkaç kez kullandıkten sonra oluşturduğum arazi (terrain).

Arazinizde istediğiniz gibi tepeler oluşturun. **Hata yapmaktan korkmayın, son işlemi Ctrl + Z kombinasyonu ile geri alabilirsiniz.**

Inspector'daki Terrain componentinde yer alan Brushes başlığı altında çeşitli fırçalar göreceksiniz. Çekinmeyin ve bu fırçalara da birer şans verin.



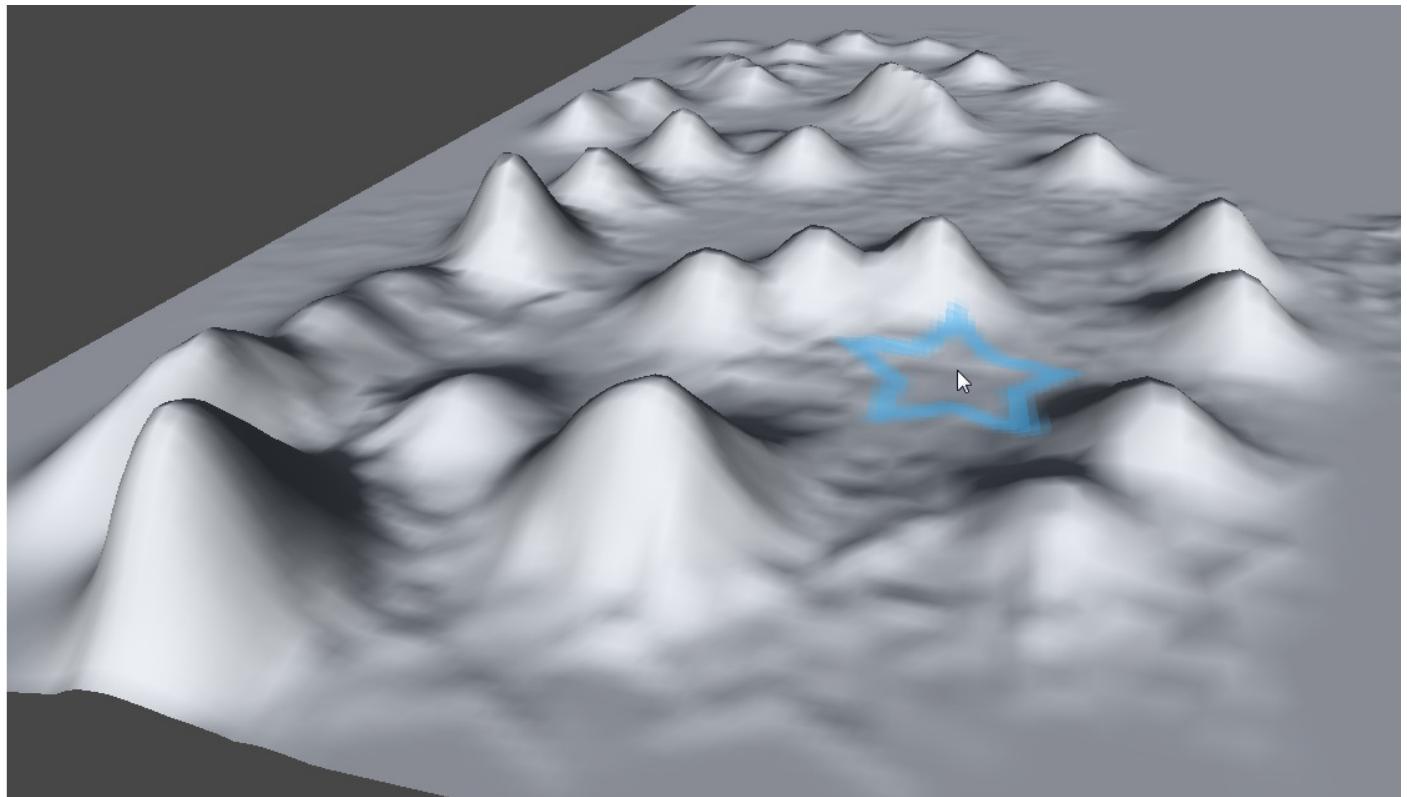
Şimdi “Settings” (ayarlar) kısmı altındaki seçeneklere bakalım. Brush size (fırça boyutu) ile mavi olarak resmedilen fırçanızın boyutunu değiştirebilirsiniz. Opacity ile de fırçanın kuvvetini değiştirebilirsiniz. Araziye tıkladığınızda yükselti o denli kuvvetli olacaktır.

Bunu Deneyin

Yıldız şekilli fırçayı seçin ve opacity'i 3 gibi küçük bir değer yapın.

Şimdi çılgınca sol mouse tuşu basılıken imleci etrafında gezdirin.

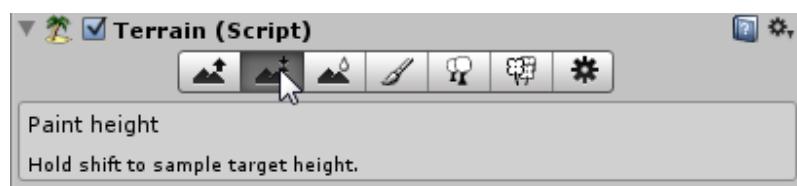
Araziye ufak girinti çıkışları vermek için birebir olduğunu görebilirsiniz.



Görsel 2.2: Yıldız şekilli fırçayı opacity 3 iken kullanmak düz araziyi şekillendirip daha gerçekçi durmasına yardımcı oluyor.

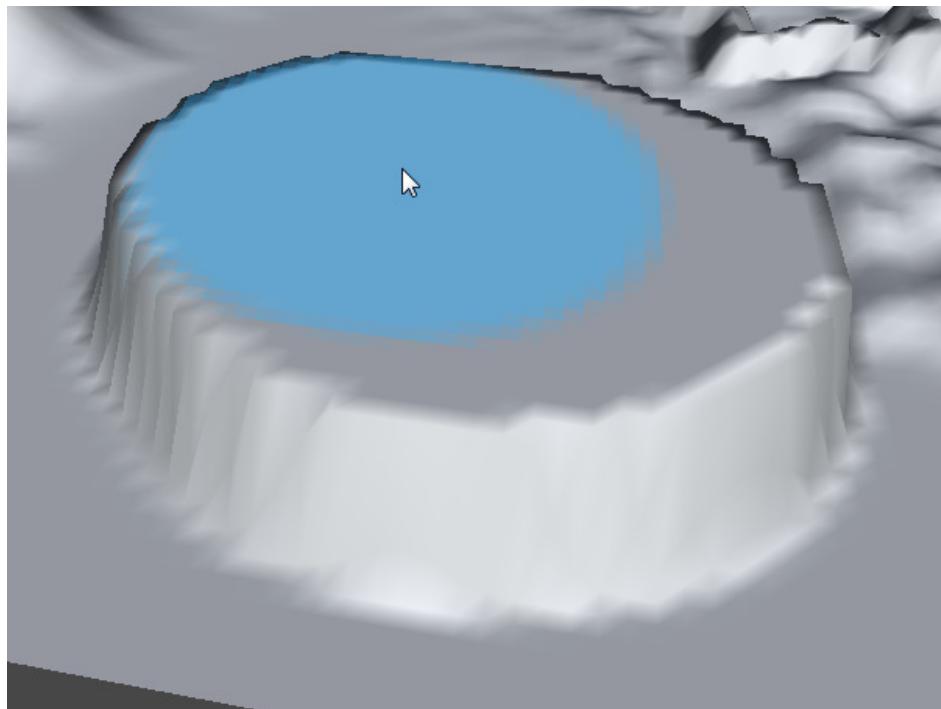
Paint Height Aracı

Şimdi soldan ikinci butonu seçin. Bu, Paint Height aracı. Yaptığınız tepenin yüksekliğinin her yerde sabit olmasını sağlar.



Shift tuşuna basılı tutun ve bir yükseltiye sol tıklayın. Bu aşama önemli çünkü Unity'e tepenin yüksekliğinin ne kadar olmasını istediğiniz burada söylüyorsunuz.

Şimdi en büyük fırçayı (soldan dördüncü) seçin, opacity'i yükseltin ve boyamaya başlayın.

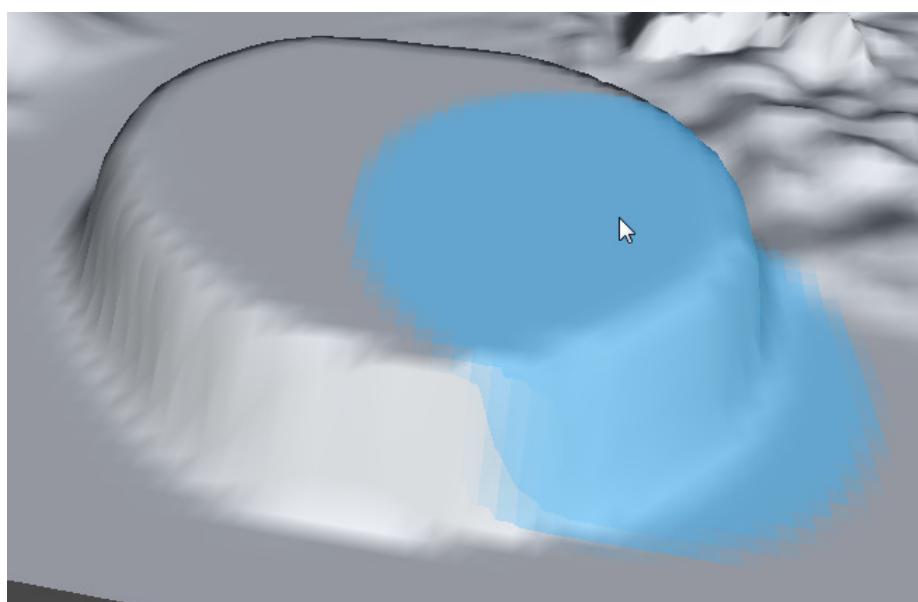


Araziyi Yumuşatma (Smoothing)

Arazinin kenarları çok Minecraft tarzı, gerçek dışı duruyor. Bunu düzeltsek iyi olacak.

Soldan üçüncü arazi aracını seçin. Buna Smooth Height aracı deniyor.

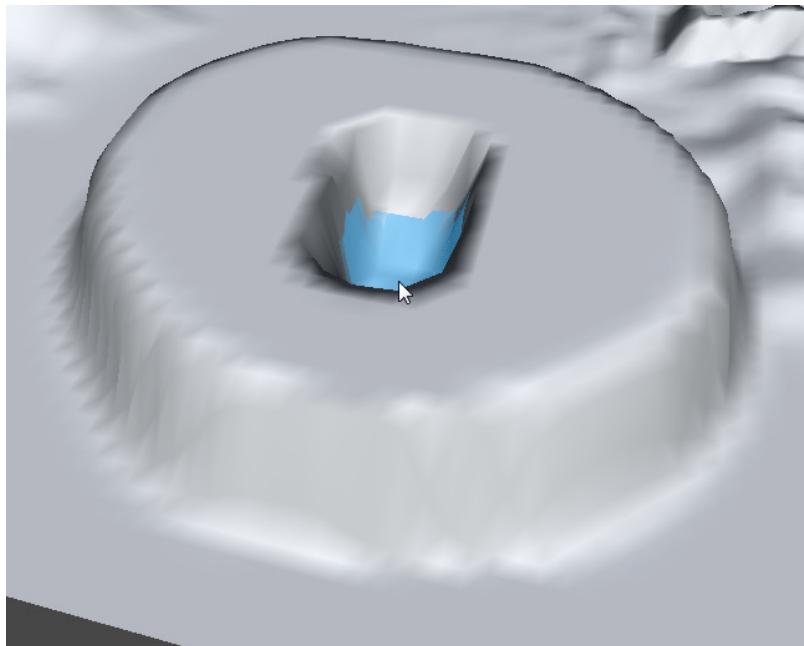
Şimdi kötü duran kenarları bu fırçayla boyayın. Alttaki resimden aradaki farkı görebilirsiniz:



Çukurlar Yapmak: Lower Terrain Aracı

En başta tepeler oluştururken kullandığımız Raise Terrain aracının çukurlar yapmak için de kullanıldığını biliyor muydunuz! Bu yüzden o aracın ismi Raise/Lower Terrain.

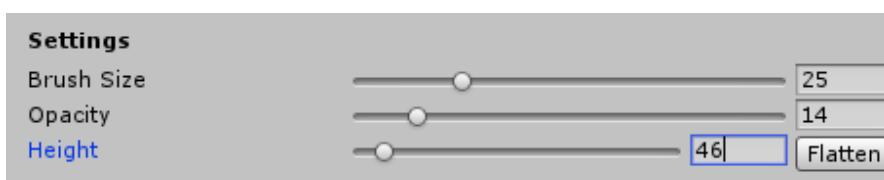
En soldaki butona tıklayarak Raise/Lower Terrain aracını seçin. **Shift tuşuna basılı tutarak etrafı boyayın.** Artık yükseltiler değil çukurlar oluşturmuş olacaksınız.



Neden Düz Zeminde Çukur Oluşmuyor?

Lower Terrain aracını düz zeminde kullanınca çukur oluşturamadığınızı farketmiş olabilirsiniz. Çünkü bu düz zemin olabilecek en düşük seviye. Bu seviyeden aşağı inemezsiniz. Eğer illa ki daha aşağı inmek istiyorsanız tüm araziyi yükseltmeniz lazım.

Unity'de tüm araziyi aynı anda yükseltmek için bir araç mevcut. Paint Height aracını (soldan ikinci) seçin. Height kısmına bir değer girin ve Flatten butonuna tıklayın.

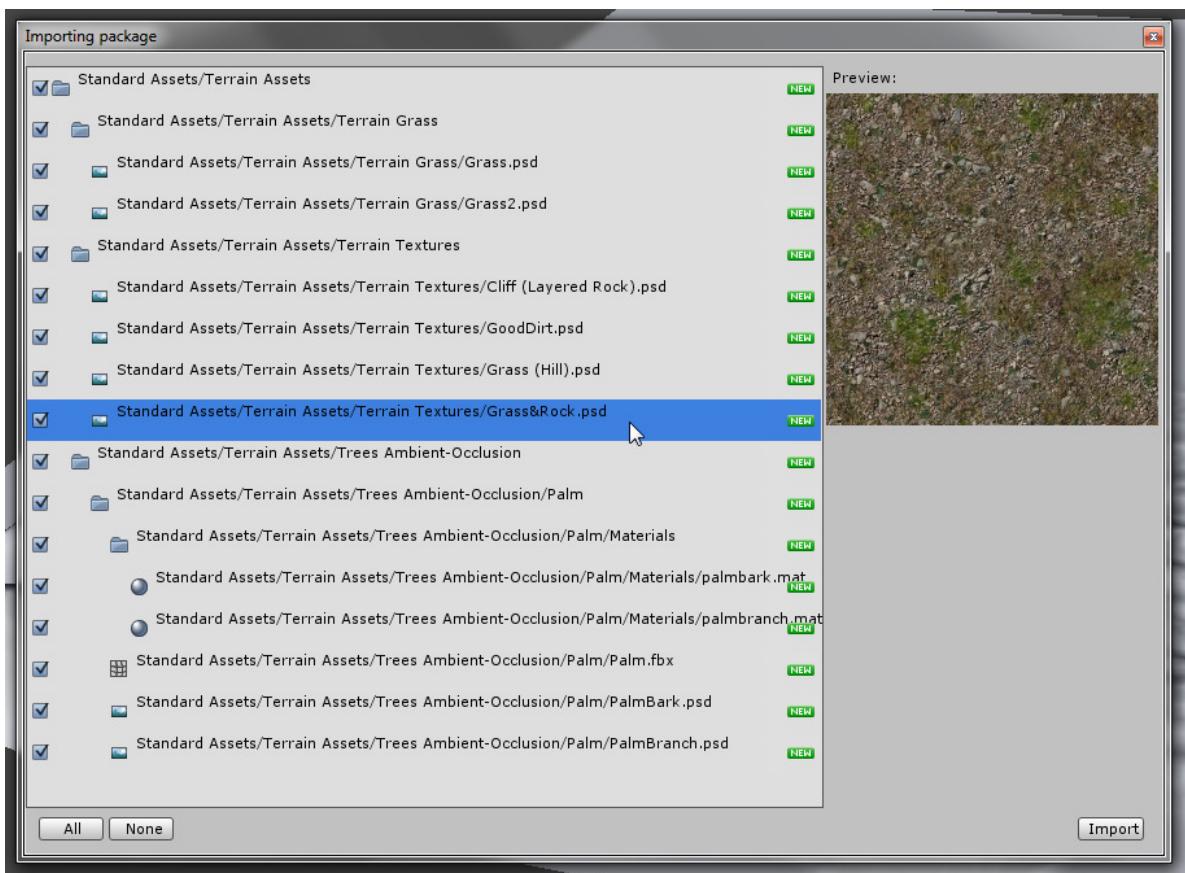


Malesef bu aracın bir yan etkisi var: arazide daha önce yaptığınız değişiklikler kayboluyor.

Kaplama (Texture) Ekleme

Açıkçası arazi böyle gri renkte olunca pek bir çirkin duruyor. Şimdi texture kullanarak araziyi evlere şenlik hale getireceğiz.

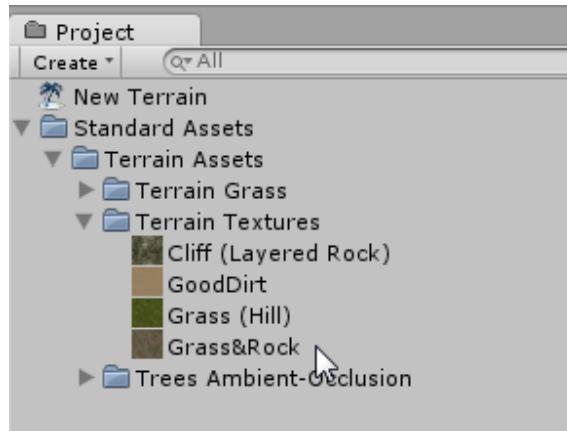
Assets > Import Package > Terrain Assets yolunu izleyin. Bu paketin getireceği dosyaları (asset) gösteren bir pencere gelecek.



Yaptığımız şey projemize yeni dosyalar eklemek. Unity'de bir projeden başka bir projede dosya aktarırken Unity Package denen bir sistem kullanılır. Şu anda import etmekte olduğumuz paket Unity'nin kendi paketidir.

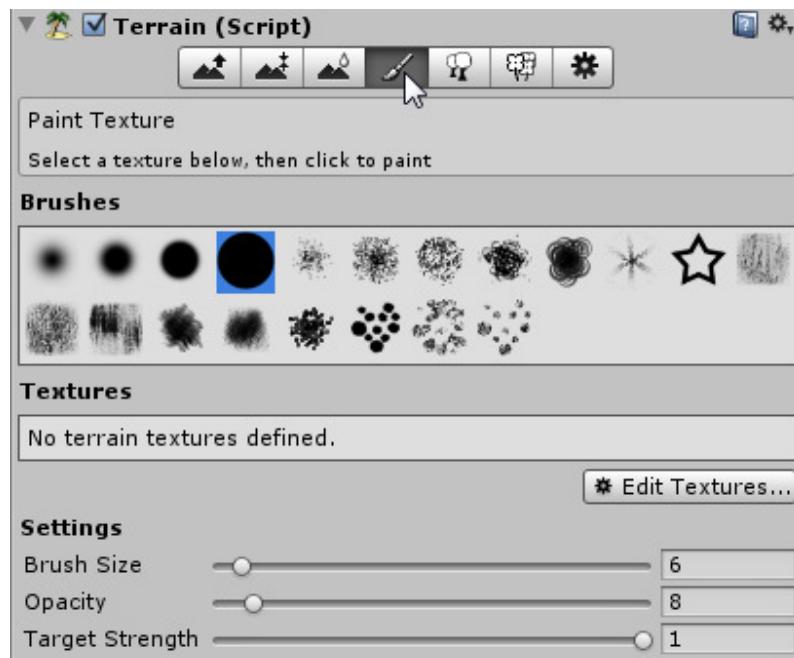
Sağ alttaki Import butonuna tıklayarak dosyaları projenize ekleyin.

Project paneline "Standard Assets" adında yeni bir klasör geldi. Bu klasörde şimdi kullanacağımız birkaç dosya bulunmakta.

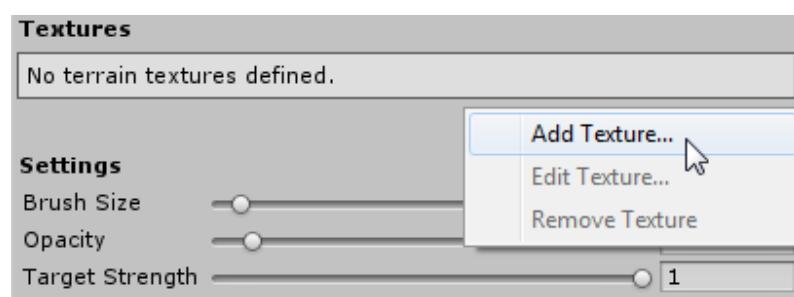


Resimde gördüğünüz “Grass (Hill)” texturesi ile arazimizi kaplayacağız.

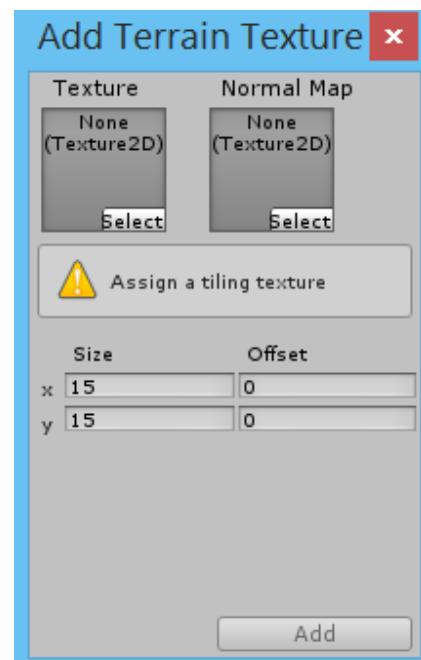
Terrain component'inde soldan dördüncü butona tıklayın. Bunun ismi Paint Texture aracı.



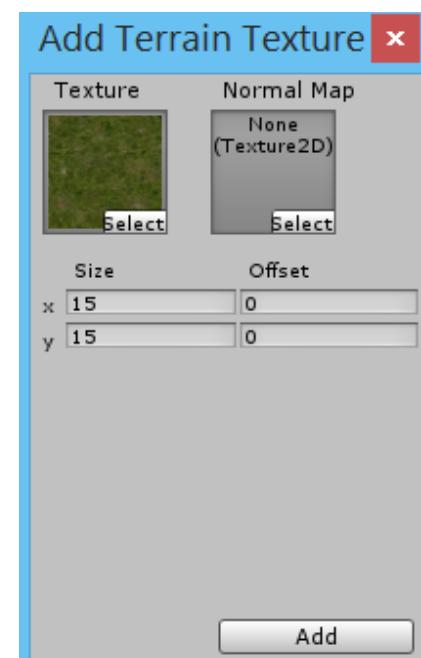
“Edit Textures” butonuna tıklayın.



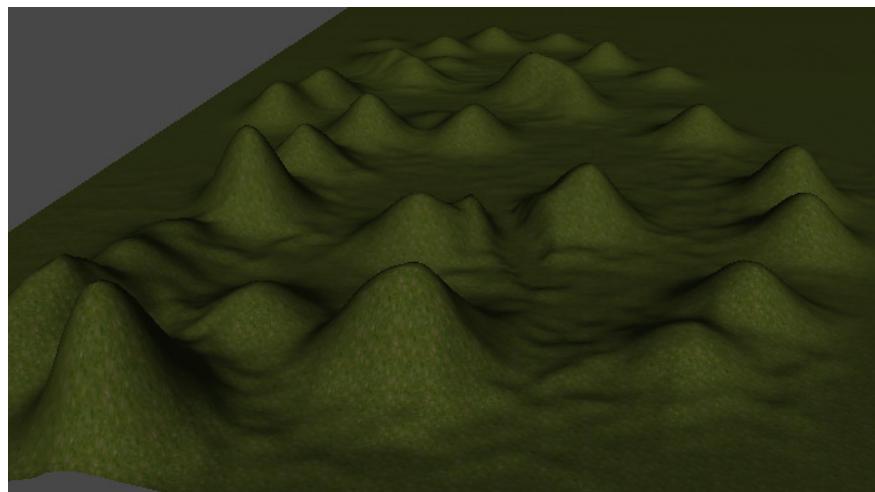
“Add Texture” seçeneğini seçin. “Add Terrain Texture” penceresi açılacak.



“Grass (Hill)” dosyasını Project panelinden tutup sürükleyerek Texture'nin altındaki boşluğa sürükleyin. Ardından Add butonuna tıklayın.



Arazi seçtiğimiz texture ile kaplanacak.



Bunu Deneyin

Araziye işlenen texture'nin boyutunu değiştirebilirsiniz. Edit Textures>Edit Texture... yolunu izleyin ve gelen pencereden Size'in altındaki değerleri (varsayılan olarak 15) değiştirin.

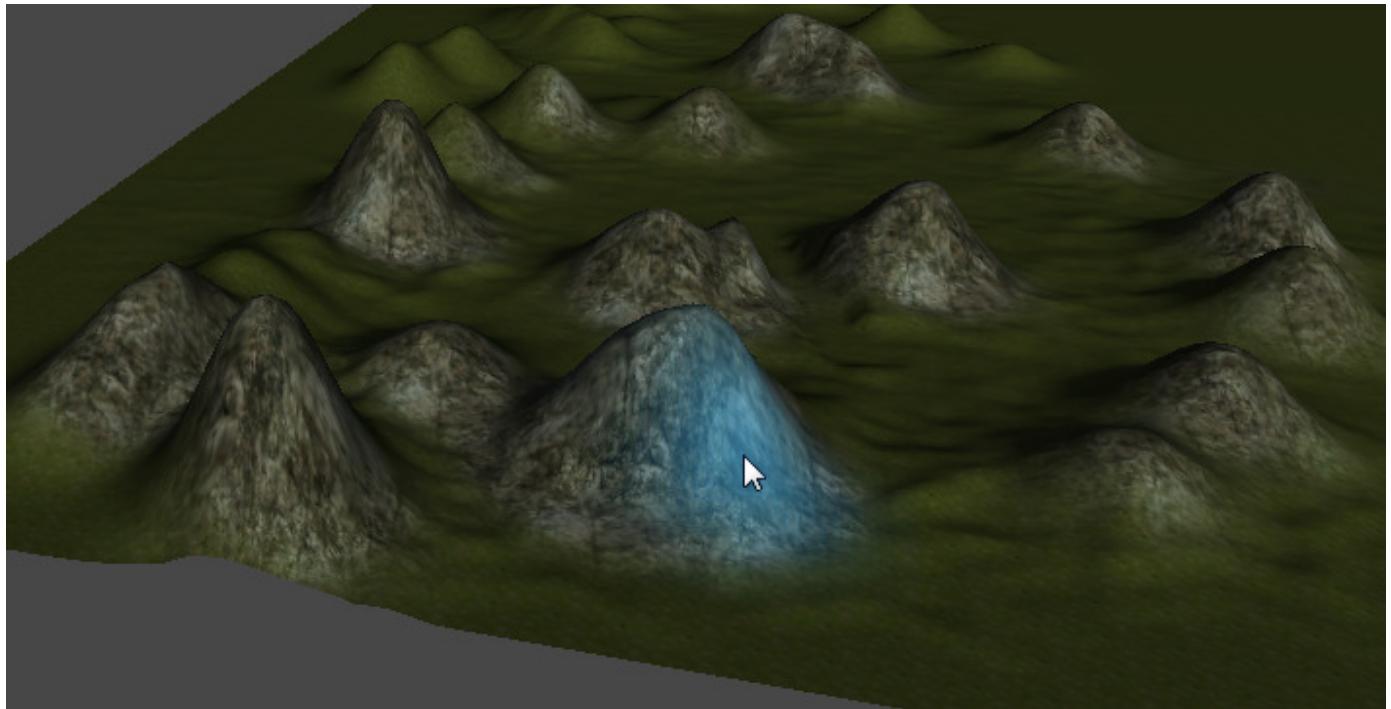
Daha büyük değer demek kaplamanın daha büyük olarak araziye işlenmesi demek. Değerlerle oynayarak istediğiniz gibi bir görünüm elde etmeye çalışın.

Başka Kaplamalar (Texture) Ekleme

Az önce birlikte yaptığımız yöntemi kullanarak yeni bir texture ekleyin. Bunun Texture'sine değer olarak "Cliff (Layered Rock)" verin. Size'in X ve Y değerlerini 50 olarak ayarlayın.

Şimdi Textures altındaki kaplamalar listesinden yeni eklediğiniz texture'yi seçin ve tepeleri boyayın.

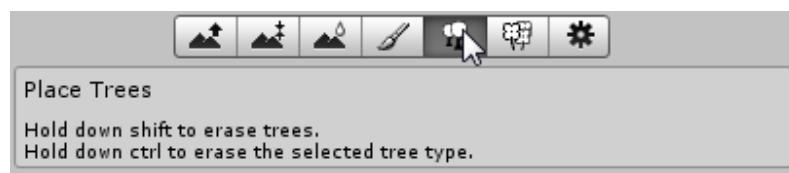




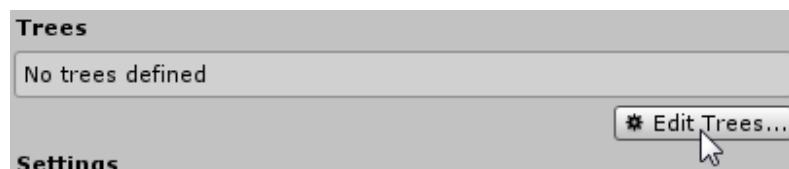
Ağaçlar Ekleme

Şimdi araziye ağaç ekleyelim. Sistem araziyi boyamakla aynı şekilde çalışıyor. Hangi ağaçtan eklemek istediğimizi seçiyor ve araziyi bu ağaçla "boyuyoruz".

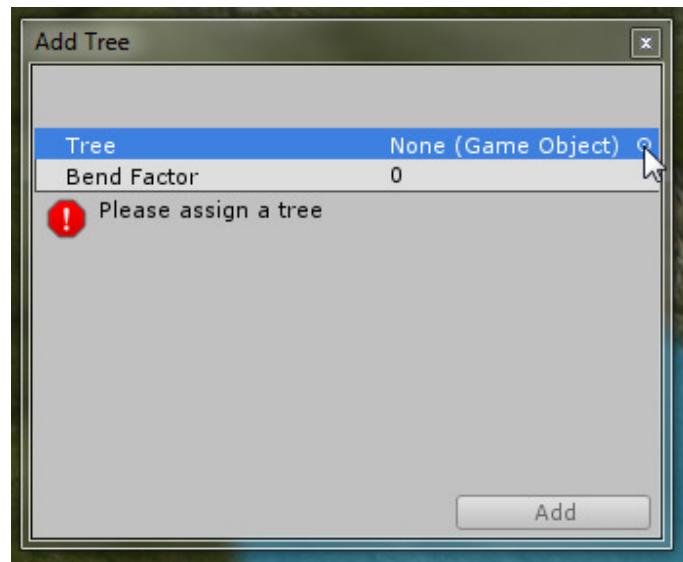
Place Trees aracını seçin.



"Edit Trees" butonuna basıp "Add Tree" seçeneğini seçin:



Şöyledir bir pencere ile karşılaşacaksınız:



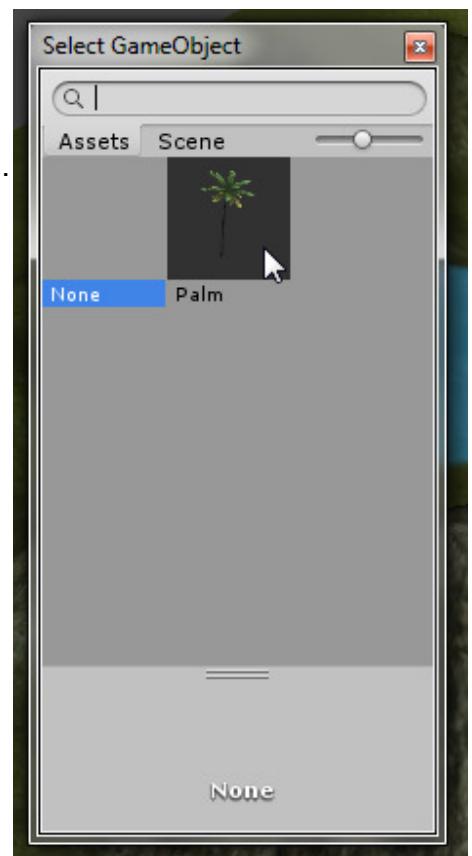
“None (Game Object)” yazan yerin sağındaki daire ikonuna tıklayın.

Bir başka pencere açılacak. Projenizde yer alan 3D modelleri seçebildiğiniz bu pencerede şu anda sadece Palm ağaç mevcut. Bu ağaç Terrain Assets paketi ile geldi.

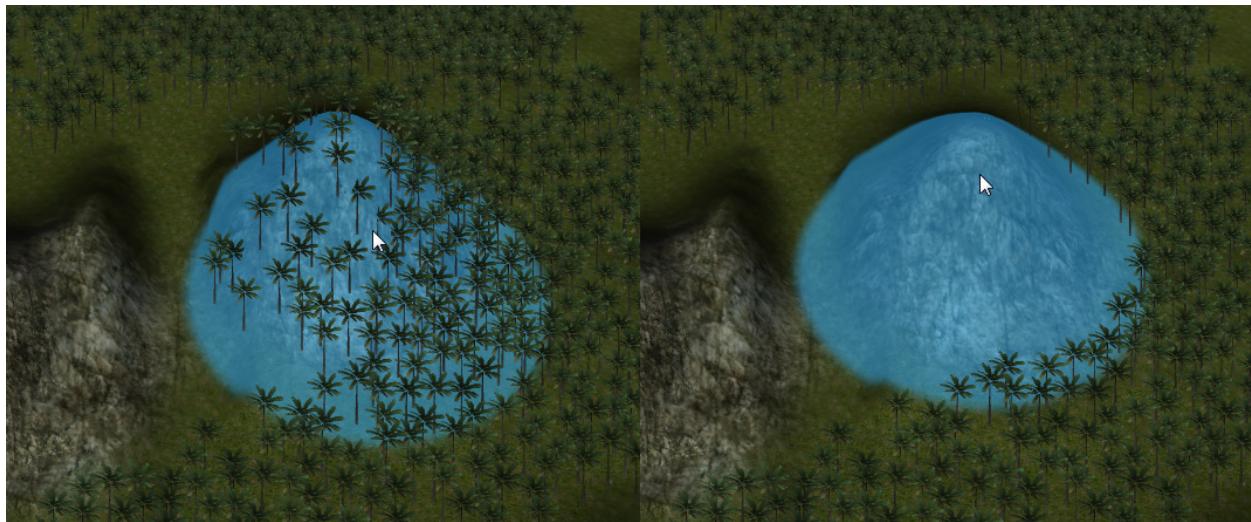
Palm'a çift tıklayın.

Sonra “Add Tree” penceresinin sağ altındaki Add butonuna tıklayın.

Artık araziye ağaç ekleyebilirsiniz!



Araziyi istediğiniz gibi ağaçlandırın. Eğer yanlış bir yere ağaç koyarsanız sorun etmeyin. **Ctrl tuşuna basılı tutarak yanlışlıkla koyduğunuz ağaçları silebilirsiniz.**

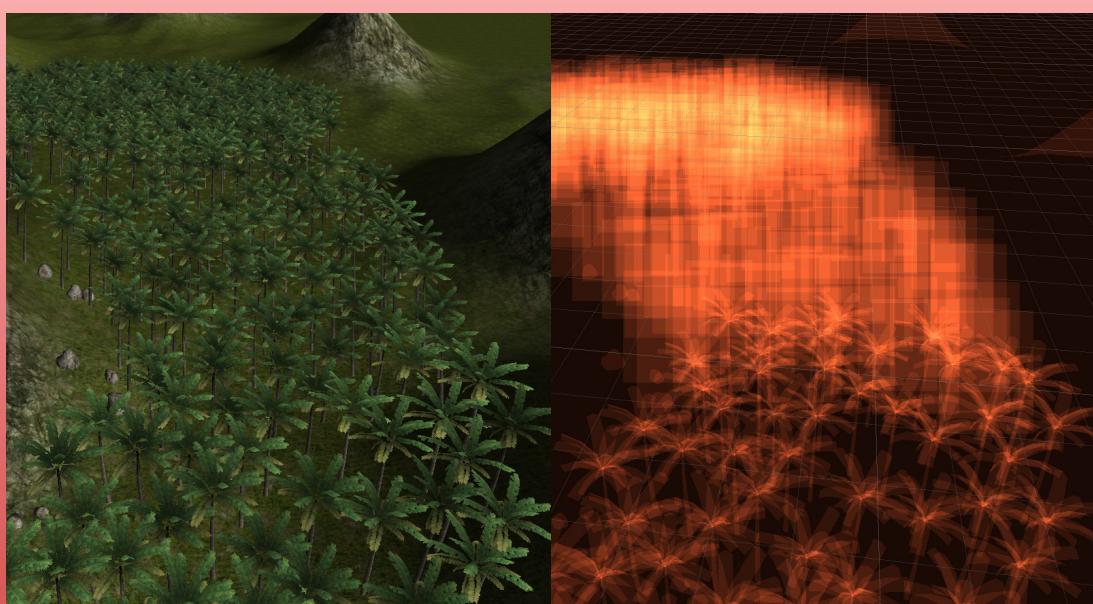


Görsel 2.3: Araziyi ağaçlarla bezerken Ctrl tuşuna basılı tutarak istenmeyen ağaçları silebilirsiniz.

Dikkat!

Çok fazla ağaç eklememeye çalışın. Yoksa bilgisayarınız ağaçları render ederken zorlanabilir.

Unity oyunun en iyi şekilde çalışması için elinden geleni yapmaktadır ama siz yine de bir takılma durumu yaşarsanız birkaç ağaçtan kurtulmayı deneyebilirsiniz.



Görsel 2.4: Unity aslında uzaktaki ağaçları birer 2D resim olarak göstermektedir.

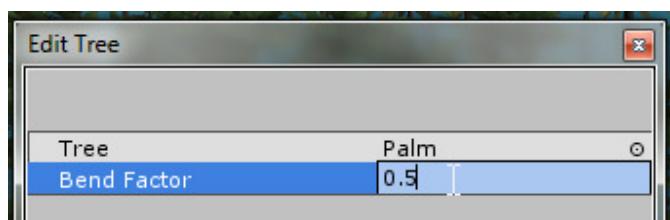
Rüzgar (Wind) Efekti

Ağaçların rüzgarda sallanması için çok basit bir yol var.

Place Trees aracını seçin. Palmiye ağacına çift tıklayın.

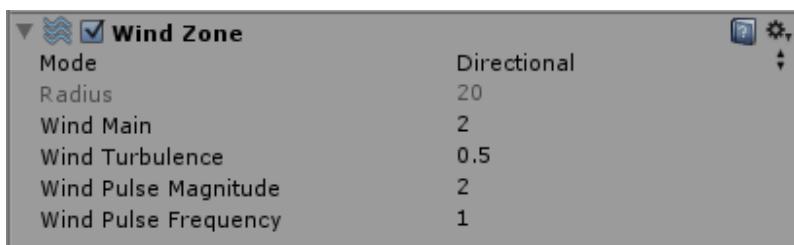


“Bend Factor”’ü 0.5 yapın ve Apply butonuna tıklayın.



Şimdi rüzgar efekti oluşturmak için özel bir objeye ihtiyacımız var. [Game Object > Create Other > Wind Zone](#) ile yeni bir Wind Zone objesi oluşturun.

Inspector'da göreceğiniz üzere bu objenin Wind Zone componenti çeşitli ayarlara sahip. O ayarları resimdeki gibi değiştirebilirsiniz:



Şimdi rüzgarın etkisini izleme vakti. Kamerayı ağaçları yakından rahatça görebileceğiniz bir yere getirin ve oyunu çalıştırın. Ağaçların sallandığını göreceksiniz. Ama muhtemelen bu sallanma efektinin sadece yakındaki ağaçlara uygulandığını da farkedeceksiniz.

Şimdi Terrain componentinin en sağındaki butona tıklayın. Burada araziyle ilgili çeşitli ayarlar mevcut. Ortalarda “Billboard Start” adında bir ayar var. Bunu iyice artırın. Oyunu başlatınca uzaktaki ağaçların da sallandığını göreceksiniz. Bu ayarın yaptığı şey belli bir mesafeden uzaktaki ağaçları oyuncuya birer 2D resim olarak göstererek sistemin daha az yorulmasını sağlamak. Değeri ne kadar artırırsanız sistem o kadar yorulur.

İşik (Light) Ekleme

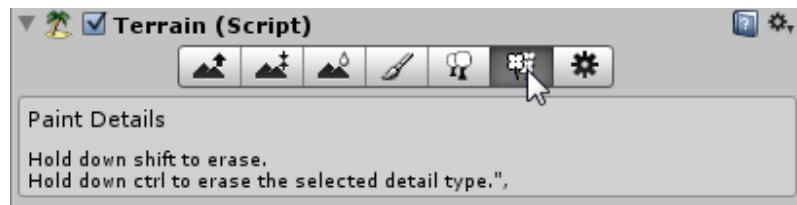
Eğer sahneniz karanlık duruyorsa (özellikle Game panelinde) ışık eklemek en iyi. [Game Object > Create Other > Directional Light](#) yolunu izleyin. Directional Light güneş ışığını taklit eden bir ışık türüdür.

Bu objenin konumu önemli değildir. Önemli olan yönüdür. Sahnedeki ışık objesini döndürerek ışığın istediğiniz yönde verilmesini sağlayın.

İlerleyen zamanlarda diğer ışık türlerini de göreceğiz.

Çimen Ekleme

Bu işlem tipki ağaç eklemek gibi. Ama bu sefer bir 3D obje yerine 2D bir resim dosyası kullanacağız. Sağdan ikinci butona tıklayarak Paint Details aracını seçin.

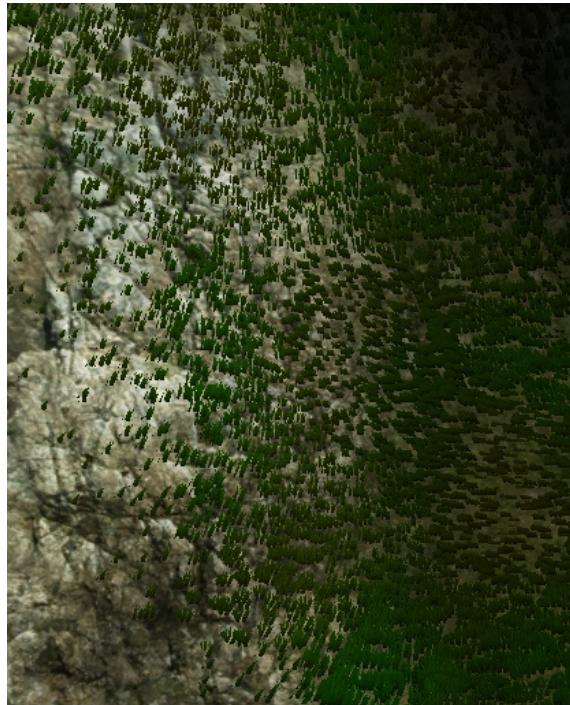


Edit Details-Add Grass Texture yolunu izleyin. "Add Grass Texture" penceresi belirecek.

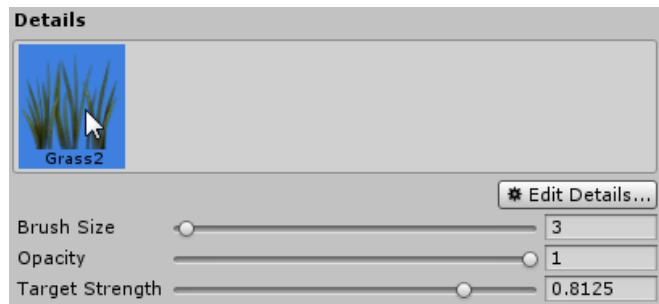
Project panelinden bir tane çimen texture'sini "Detail Texture" yazan yere sürükleyin ve "Add" butonuna basarak işlemi sonlandırın.



Şimdi zemine fırçayla dokunarak çimen ekleyebilirsiniz.
Birşey göremezseniz araziye iyice zoom yapın. **İyi performans için “Target Strength” değerini azaltarak daha az çimen eklemek isteyebilirsiniz.**



Tepeden bakınca çimenin çirkin durduğunu farkettiniz mi? Çünkü çimen 3D uzaya yerleştirilmiş 2D bir resim dosyası. Her ne kadar kendisini iyi göstermeye çalışsa da bazı kamera açılarında havası sönüyor.

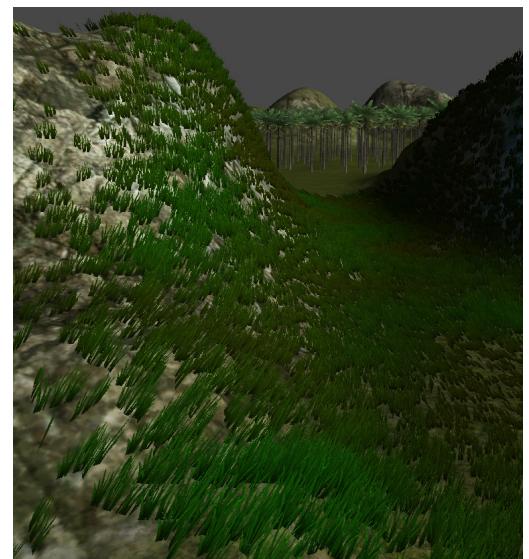


Inspector'dan çimen texture'sine çift tıklayın.

Gelen pencerenin en altında "Billboard" adında bir seçenek yer almaktır.

Bu seçenek seçiliyken çimen kaplaması hep size doğru bakmaktadır.

Billboard'un başındaki işaretü kaldırın. Artık çimen hep size doğru bakmayacaktır. Billboard özelliğini açıp açmamak tamamen bir zevk meselesi.



İpucu

Zemini çimenle benzer renkte bir texture ile boyarsanız zemin daha gerçekçi görünür.

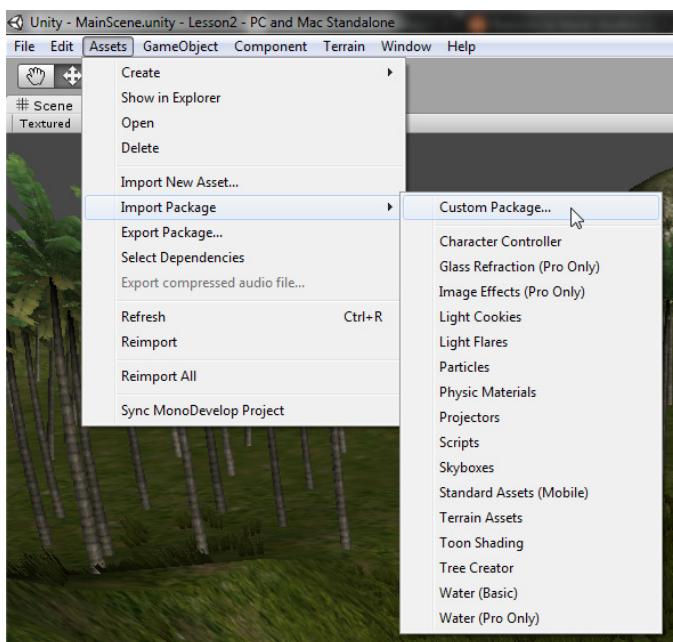


Taş Ekleme

Araziye taş eklerken yine Paint Details aracını kullanıyoruz.

Ama öncesinde elimizde 3 boyutlu bir taş modeli olmalı. Bu taş modelini elde etmek için yeni bir paket import edeceğiz.

Assets > Import Package > Custom Package... yolunu izleyin.



Bilgilendirme

Eğer çimenlerin de taşlar gibi üç boyutlu olmasını isterseniz 3D bir çimen modeli bulup bunu tipki taşlarda yaptığınız gibi "Add Detail Mesh" yoluyla araziye ekleyin.

Dersle gelen Winrar arşivini çıkardığınız klasöre gidin. Buradan Packages klasörüne girin ve "Rock.unitypackage" dosyasını seçin.

Paketle gelen dosyaların (asset) listesini göreceksiniz. Sağ alttaki Import butonuna tıklayarak işlemi tamamlayın.

Artık taşları araziye eklemeye hazırız.

Terrain component'indeki Paint Details aracını (sağdan ikinci buton) seçin. Bu sefer Edit Details-Add Detail Mesh yolunu izleyin.

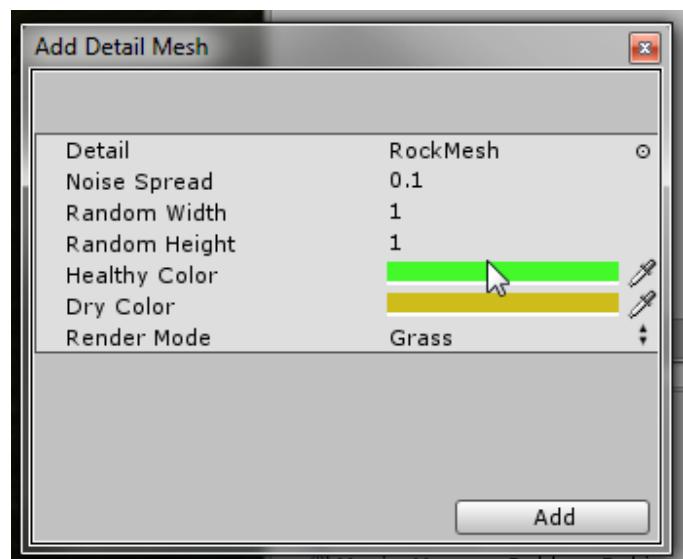
Taş modelini (RockMesh) Detail'in sağındaki boşluğa sürükleyip bırakın (ya da sağdaki küçük daire ikonuna tıklayıp 3D modeli listeden seçin).

Şimdi birkaç ayar daha yapmamız lazım:

"Healthy Color" ve "Dry Color" renklerini beyaz olacak şekilde değiştirin.

"Render Mode"daki "Grass" değerine tıklayın ve onu "VertexLit" olarak değiştirin.

Add butonuna tıklayarak işlemi sonlandırın.



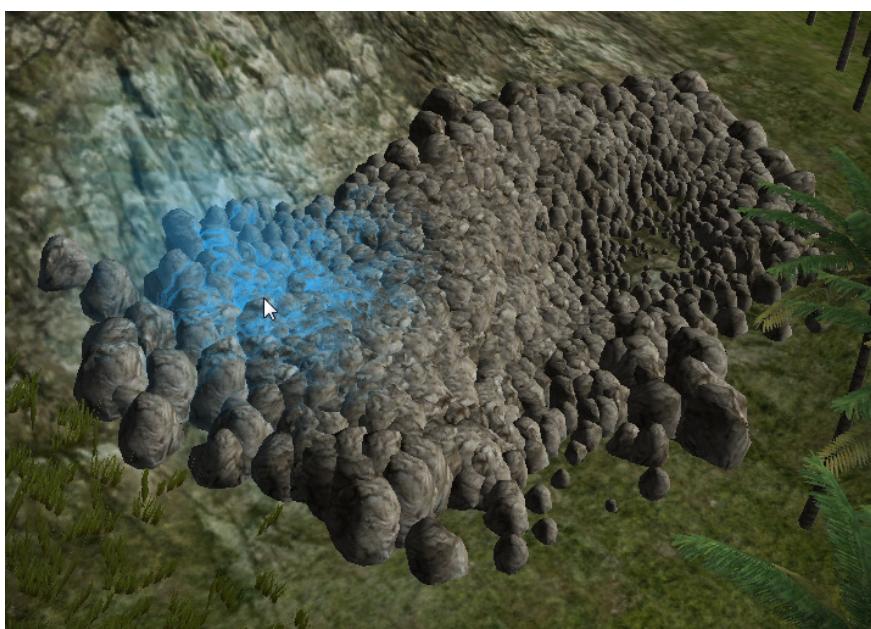
Bilgilendirme

Healthy Color ve Dry Color'un ne olduğunu merak ettiniz mi?

Healthy Color'a ne renk atarsanız ilgili çimen ya da taş o renge bürünür. Bu sayede örneğin kırmızı bir çimen istiyorsanız kırmızı renkte yeni bir texture oluşturmak zorunda kalmazsınız. İşin güzel yanı bu rengin değerini script'ler vasıtasiyla oyun sırasında değiştirip görsel şölen yapabilmeniz.

Dry Color ise ekstra bir renk. Çimen ya da taşın rengi zamanla bu renge bürünür. Aslında olan şey şu: çimen ya da taşın rengi oyun sırasında Healthy Color ile Dry Color arasında zamanla gidip gelir. İşte bu kadar...

Şimdi Terrain component'inden taş brush'ını seçin ve araziyi boyayın. Muhtemelen bir yere çok fazla taş konduğunu göreceksiniz. Eğer hiçbir şey görmüyorsanız araziye zoom yapın.



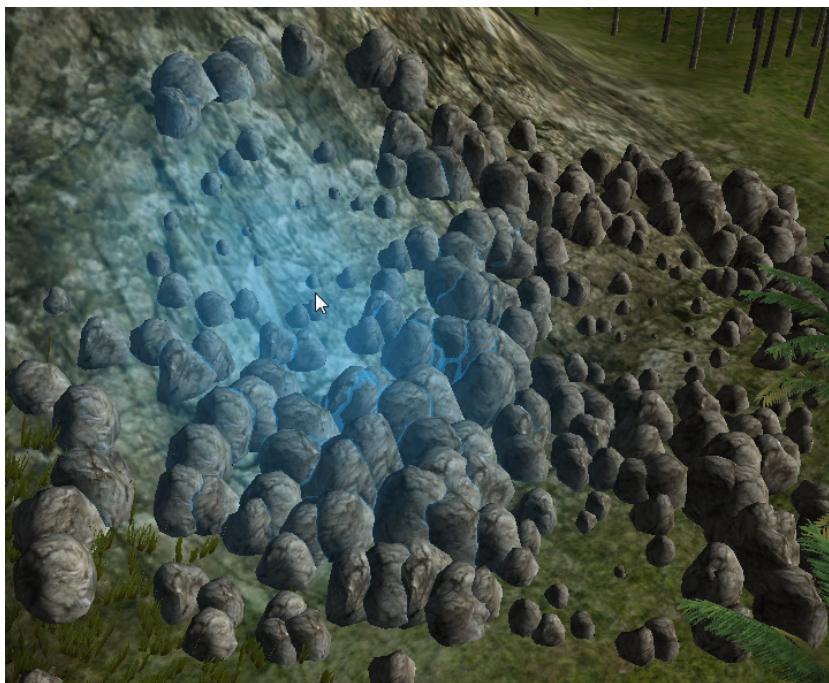
Çok fazla taş koyma sorununu çözmek için izlememiz gereken yol şu:

Sahneye koyduğunuz taşları silin önce. Bunun için Ctrl tuşuna basılı tutun.

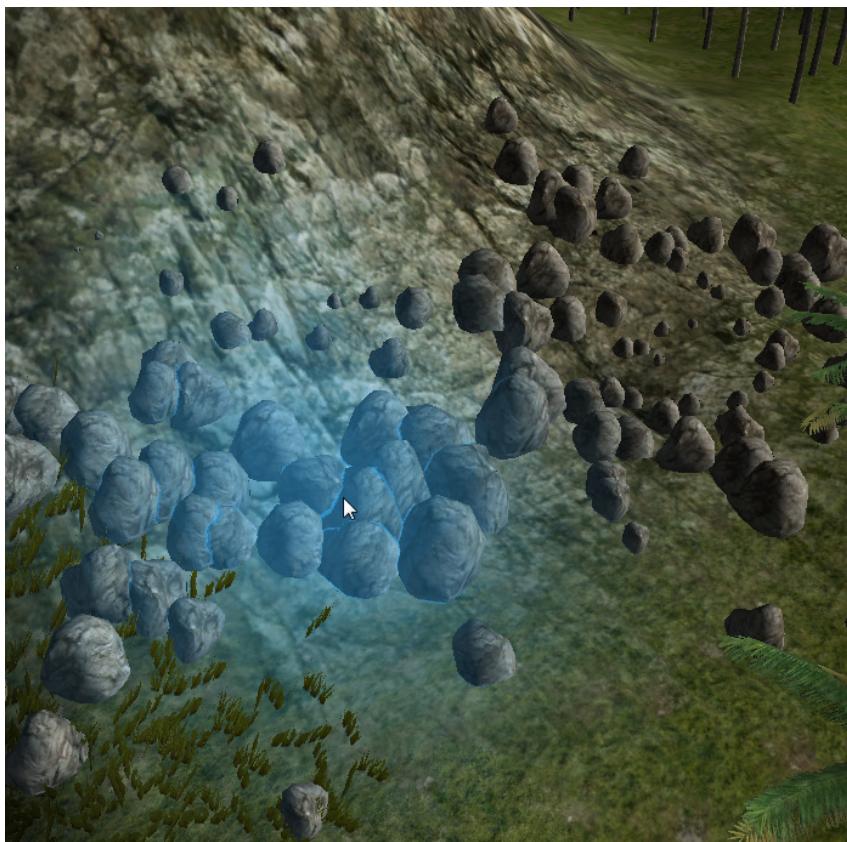
Tek seferde eklenen taş sayısını azaltmak için Paint Details aracının ayarlar kısmındaki Target Strength değerini azaltın. Ama sıfır yapmayın!



Araziye tekrar taş koymaya çalışığınızda artık daha az taş konduğunu göreceksiniz. Ama hâlâ gereğinden fazla taş konuyor.



Ayarlara geri dönün ve bu sefer Opacity'i azaltın. Mesela 0.05 yapın. Sonra fırçayı tekrar deneyin.



Artık daha iyi. Opacity ve Target Strength'i kendinize uygun şekilde değiştirebilirsiniz.

Bilgilendirme

Target Strength ile Opacity arasındaki fark nedir?

Opacity tek bir tıklamada koyduğunuz taş sayısını belirler.

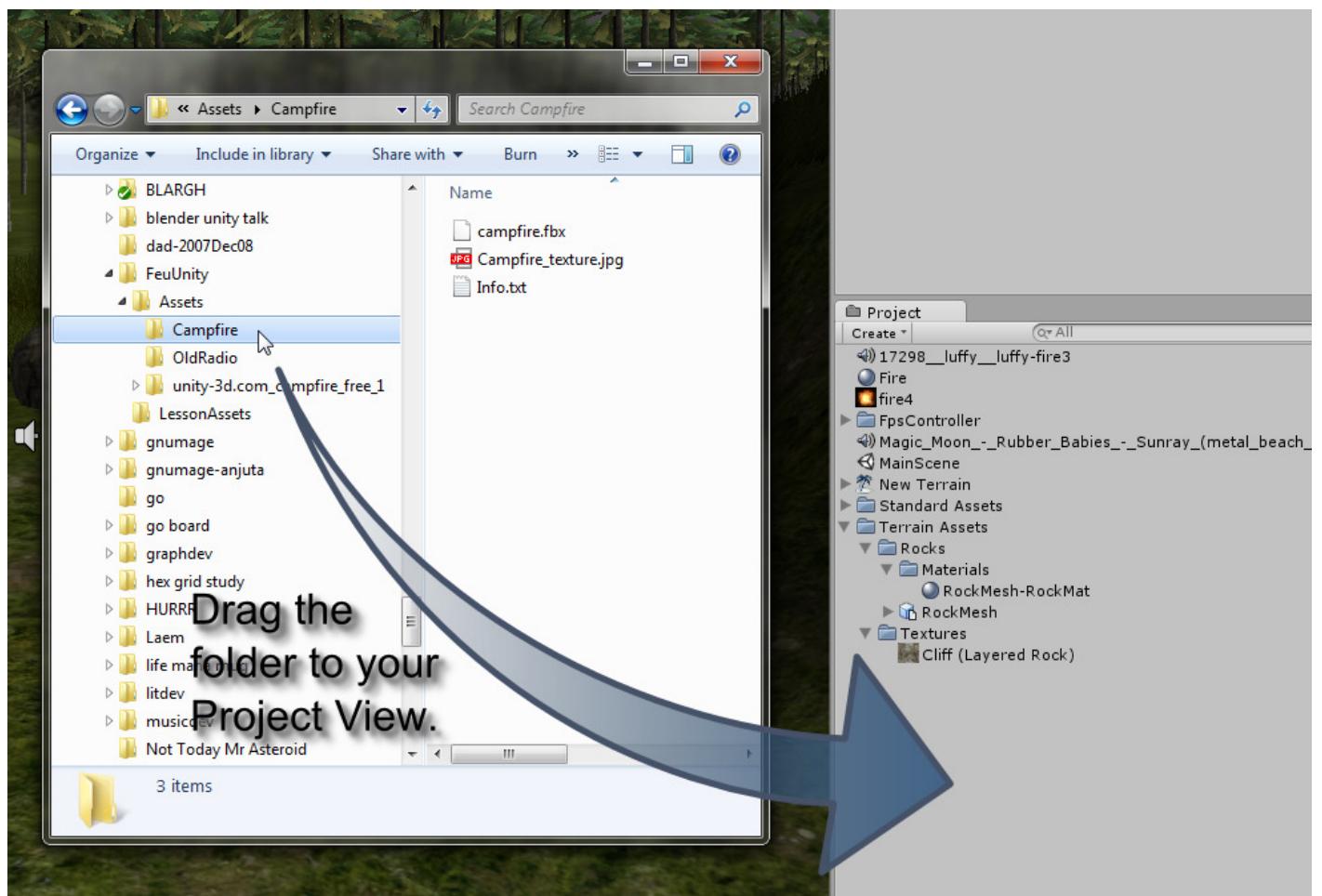
Target Strength ise bir bölgede olabilecek maksimum taş sayısını belirler. O bölgeye fırçayla ne kadar çok dokunursanız dokunun; oraya Target Strength'ten daha çok taş koyamazsınız.

Kamp Ateşi Eklemek

Şimdi araziye şirin bir kamp ateşi ekleyeceğiz ama öncesinde bunun için bir 3D modele ihtiyacımız var.

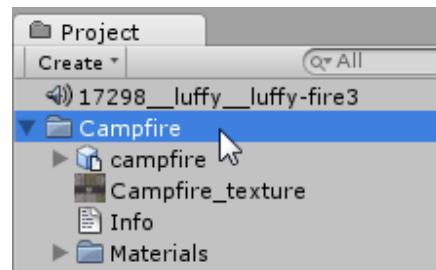
Tahta yaptığımız gibi, 3D modeli projemize ekleyeceğiz. Ama bu sefer Unity Package kullanmak yerine modeli direkt import edeceğiz.

Windows Explorer'ı açın ve Winrar'ı çıkardığınız yerdeki "Campfire" klasörünü bulun. Klasörün içinde ihtiyacımız olan birkaç dosya var. Hepsini aynı Unity'e import etmek için klasörü tutup sürükleyerek Unity'deki Project paneline sürükleyin.



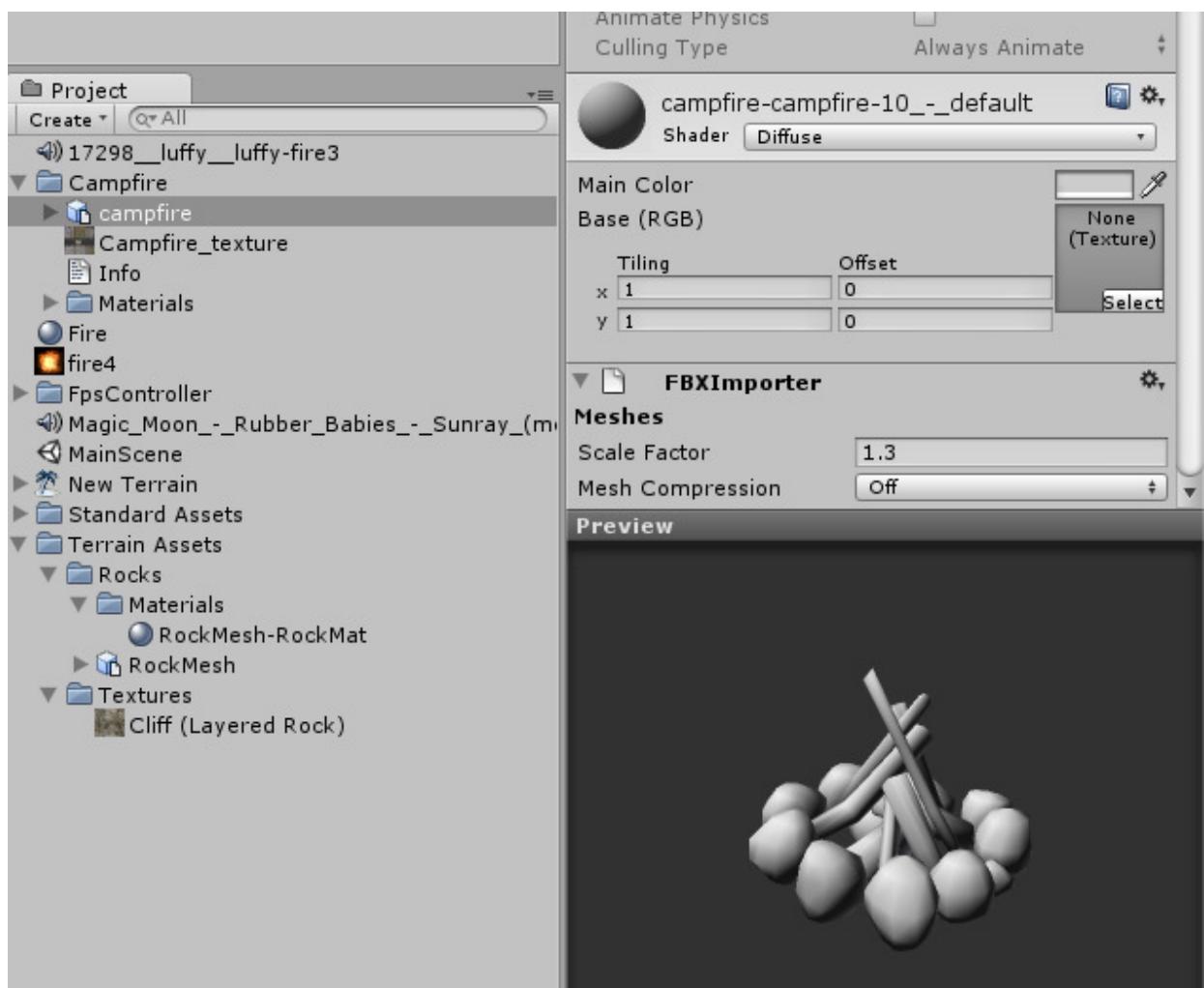
ÇEVİRMEN EKLEMESİ: Klasörü sürükle&bırak yöntemi bende işe yaramadı. Bu yüzden ben de Campfire klasörünü kopyaladım ve Unity projemi oluşturduğum konumda yer alan Assets klasörünün içine yapıştırdım. Sorun böylece çözüldü.

Unity'nin dosyaları import etmesini bekleyin. İşlem bitince Project panelinde "Campfire" klasörünü göreceksiniz.



Klasörü Unity'e import edince otomatik olarak oluşan yeni bir klasörümüz de var: "Materials". Unity bu klasörün içinde 3D modelin sahip olduğu materyalleri depolar.

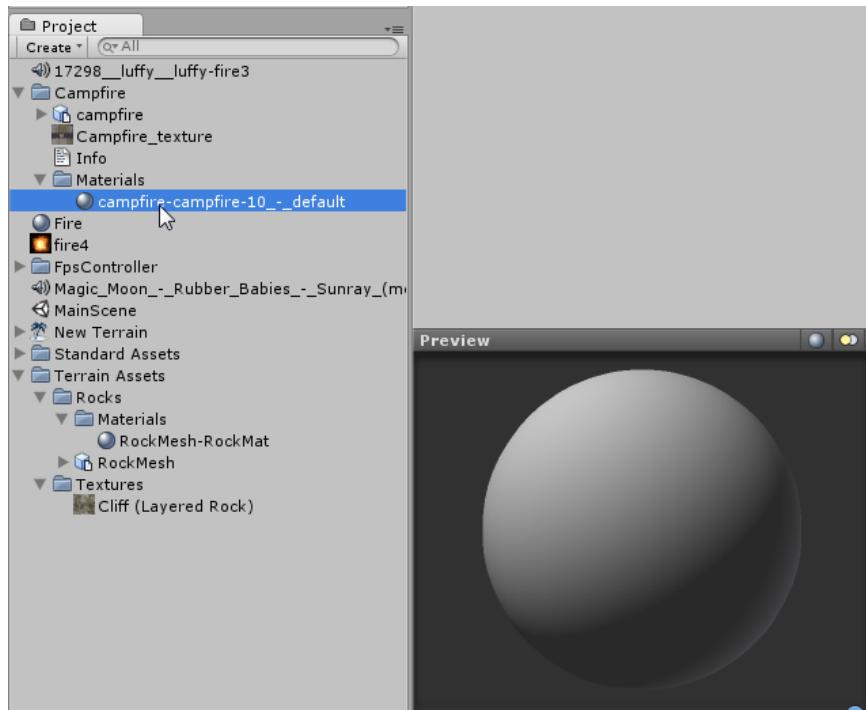
Klasörün içinde campfire dosyasını (asset) mavi bir küp ikonuyla birlikte görebilirsiniz. Bu bir 3D model. Tıklayın kendisine.



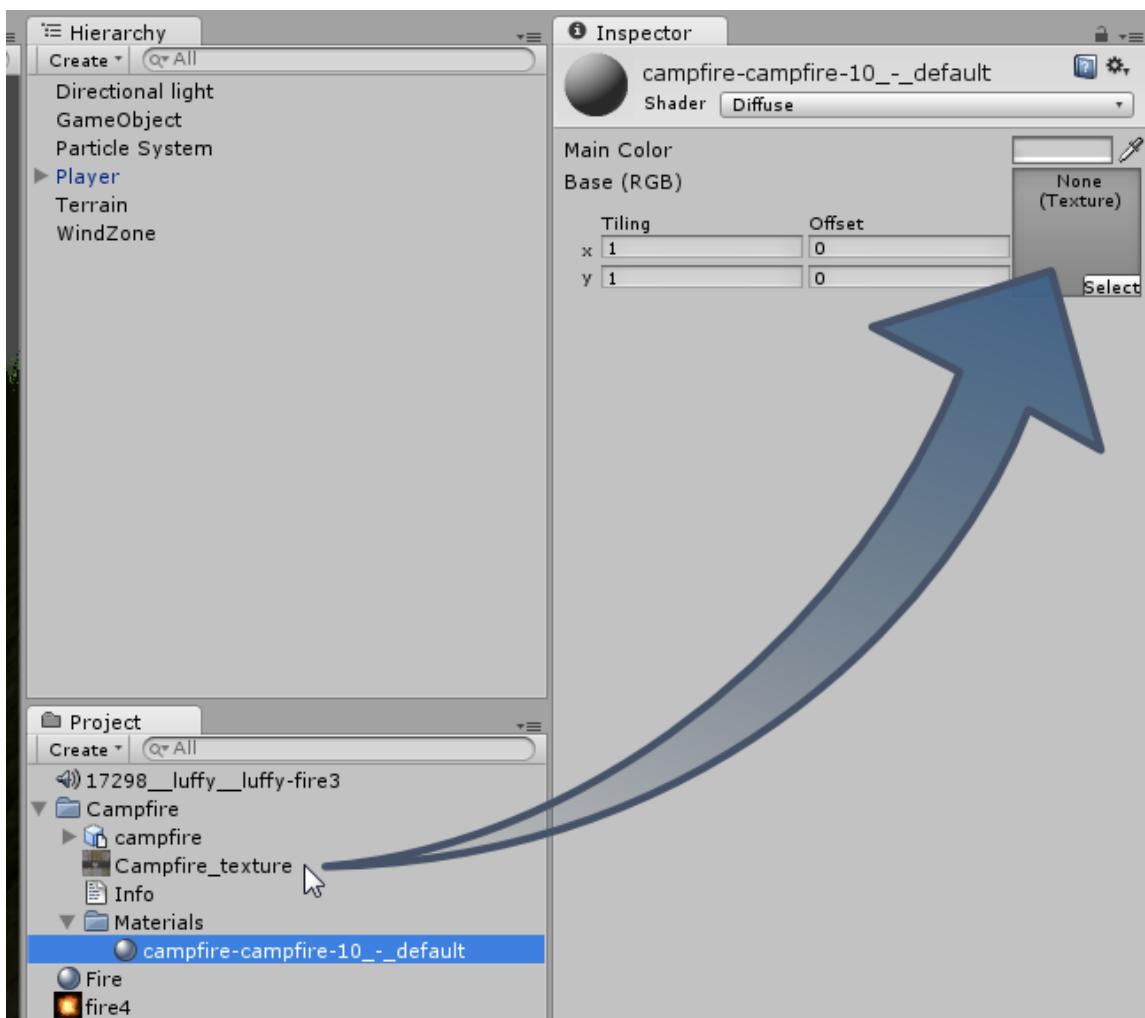
Inspector'da 3D modelin öznemesini görebilirsiniz. Malesef modele henüz bir texture ya da materyal atanmamış durumda, bu yüzden beyaz renkte gözükmüyor. Şimdi bunu çözeceğiz.

"Campfire_texture" adında bir dosya var klasörün içinde. Bu, kamp ateşine renk kazandıran bir texture dosyası.

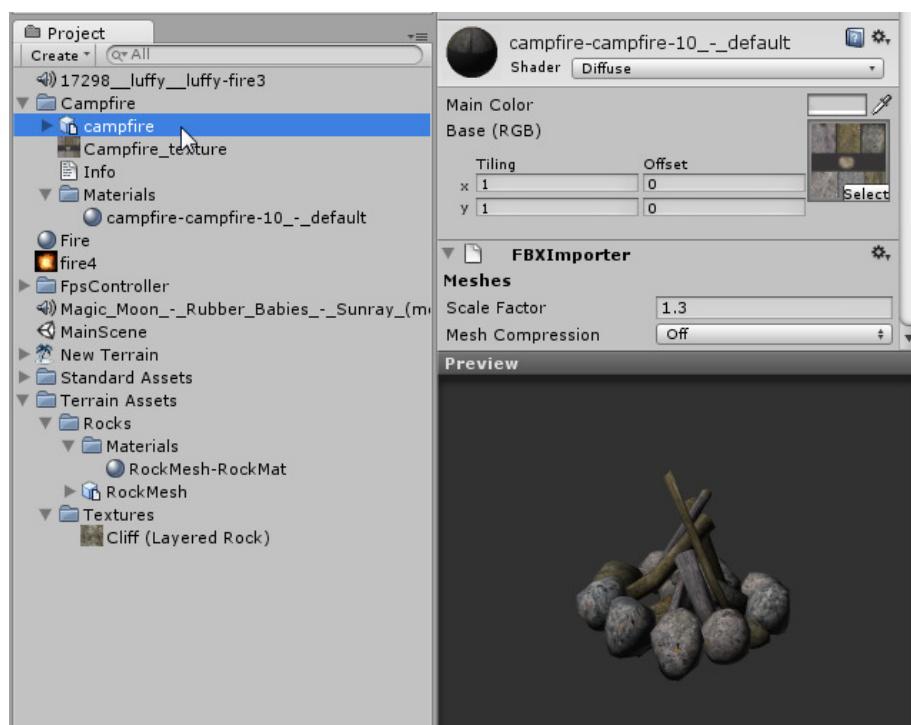
"Materials" klasöründe de bir dosya var. Unity tarafından otomatik olarak isimlendirilmiş bir materyal dosyası. Bu, campfire 3D modelinin kullandığı materyalin ta kendisi. Yapmamız gereken bu materyale ilgili texture dosyasını atamak.



Altaki resimde görüldüğü gibi, Campfire_texture dosyasını Project panelinden tutup sürükleyerek materyalin Texture kısmına bırakın:



Eğer şimdi tekrar 3D modelin önizlemesine bakarsanız modelin renklendiğini göreceksiniz.



NOT: Eğer sizde önizleme penceresi çok küçükse tepesinden tutarak büyütübilirsiniz.

Şimdi arazimize bir tane kamp ateşi koyalım. Bunun için campfire dosyasını Project panelinden tutup Scene paneline sürükleyin. Ardından istediğiniz gibi yerleştirin.

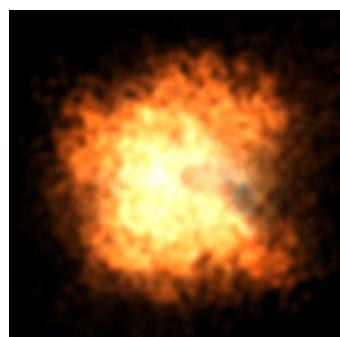


ÇEVİRMEN EKLEMESİ: Ben campfire'ı sahneye sürüklendikten sonra bir türlü bulamadım. En sonunda F tuşu ile kamerayı ona odakladığında objenin çok küçük olduğunu farkettim. Bunu çözmek için ise campfire dosyasını (asset) Project panelinden seçip Inspector'dan **Scale Factor** değerini 3 yaptım ve aşağıdaki **Apply** butonuna tıkladım. Böylece sorun çözüldü.

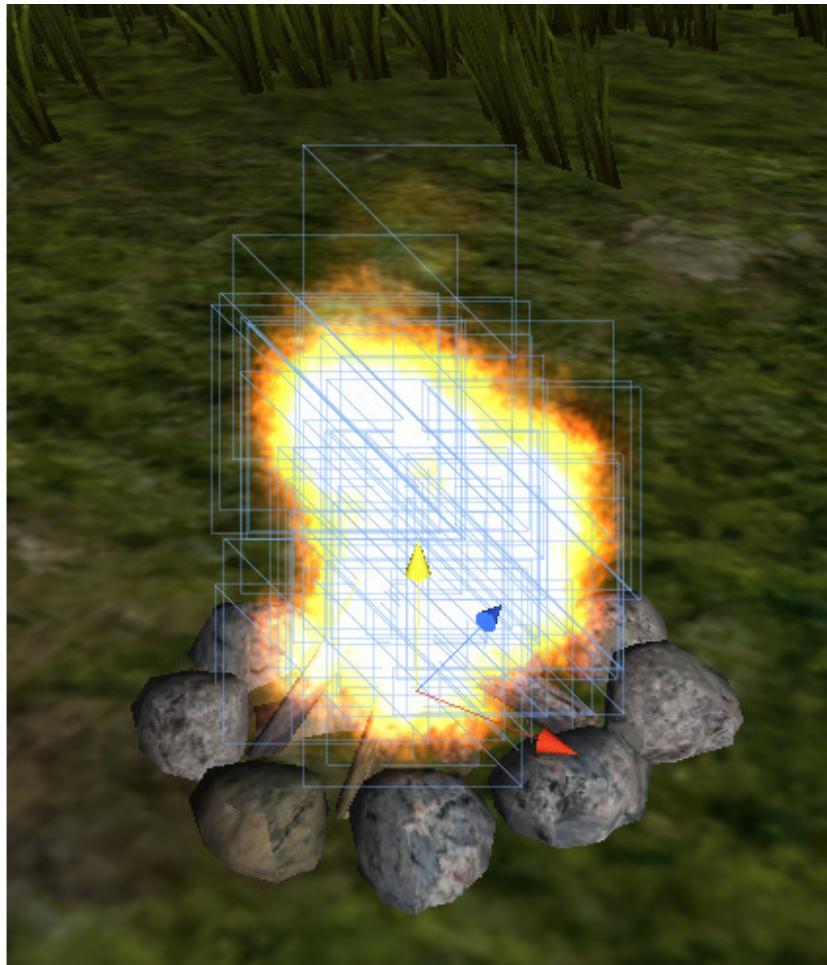
Ateş Eklemek

Ateşsiz bir kamp ateşi hayal edebilir misiniz? Şimdi sahneye yerleştirdiğimiz 3D modelin üzerine ateş efekti ekleyelim. Bunun için partikül efekti (particle effects) denen bir yönteme başvuracağız.

Fikir şu: resimdeki gibi bir ateş texture'mız olacak.



...ve biz bu texture'den aynı anda pek çok tane oluşturup biraz da animasyon katarak bir ateş efekti yapmaya çalışacağız.



Görsel 2.5: Resimde gördüğünüz her kare, biraz önce gördüğünüz ateş texture'sinin bir klonu.

Partikül efektlerinde her bir resim klonu birer "particle" olarak adlandırılır. Her bir partikülün kendine has özellikleri vardır: hızı, kaybolma süresi gibi.

Biz bu derste Unity'nin eski partikül sistemini kullanacağız. **GameObject > Create Empty** ile yeni bir görünmez obje oluşturun. Objenin ismini **Particle System** olarak değiştirin. Sonra sırasıyla şu component'leri ekleyin:

Component > Effects > Legacy Particles > Ellipsoid Particle Emitter

Component > Effects > Legacy Particles > Particle Animator

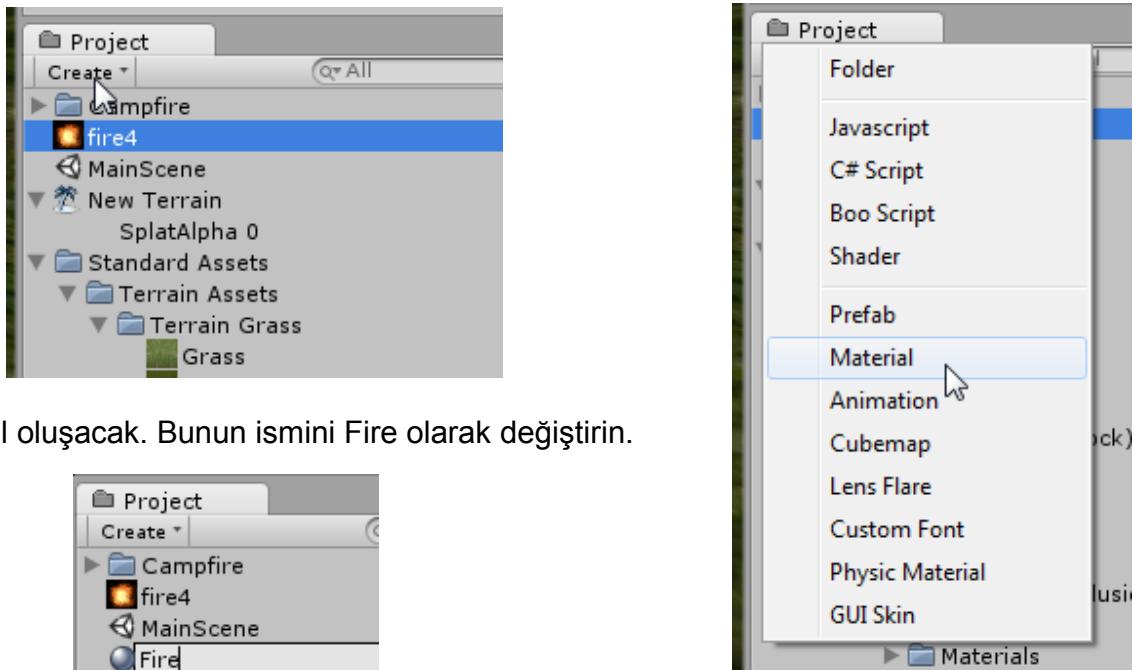
Component > Effects > Legacy Particles > Particle Renderer

Eğer şimdi kamerayı F tuşu ile partikül sisteme odaklısanız pembe partiküller saçtığını göreceksiniz. Biz partiküllerin pembe değil alev şeklinde olmasını istiyoruz.

Öncelikle bir ateş texturesine (kaplama) ihtiyacımız var. Winrar arşivini çıkardığınız yerdeki Images klasöründe "fire4.psd" isimli bir resim dosyası mevcut. Bunu **Assets > Import New Asset...** yolunu izleyerek projenize import edin.

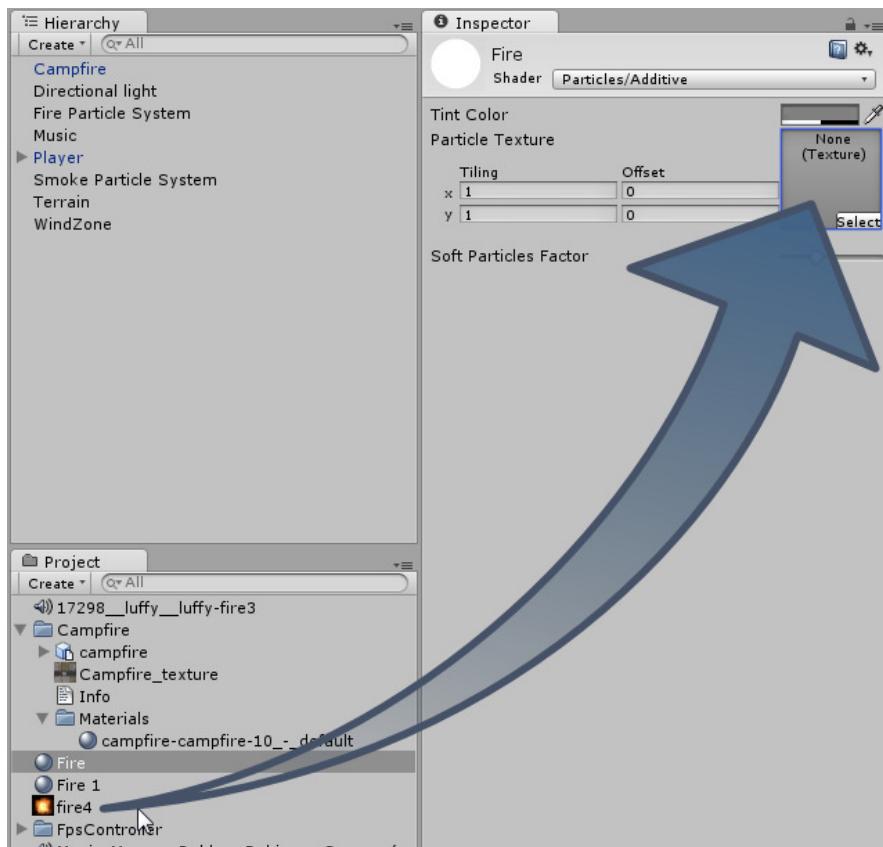
Bu ateş texture'sini direkt partiküllere veremeyiz. Önce texture'yi bir materyale vermelii, sonra materyali partiküllere vermeliyiz.

Project panelinin sol üstündeki Create butonuna tıklayın ve listeden "Material"ı seçin:



Yeni bir materyal oluşacak. Bunun ismini Fire olarak değiştirin.

Şimdi fire4 kaplamasını Fire materyalinin Inspector'undaki Texture boşluğuna sürükleyn:



Artık materyalin önizleme penceresinde ateş texture'si gözükecek.

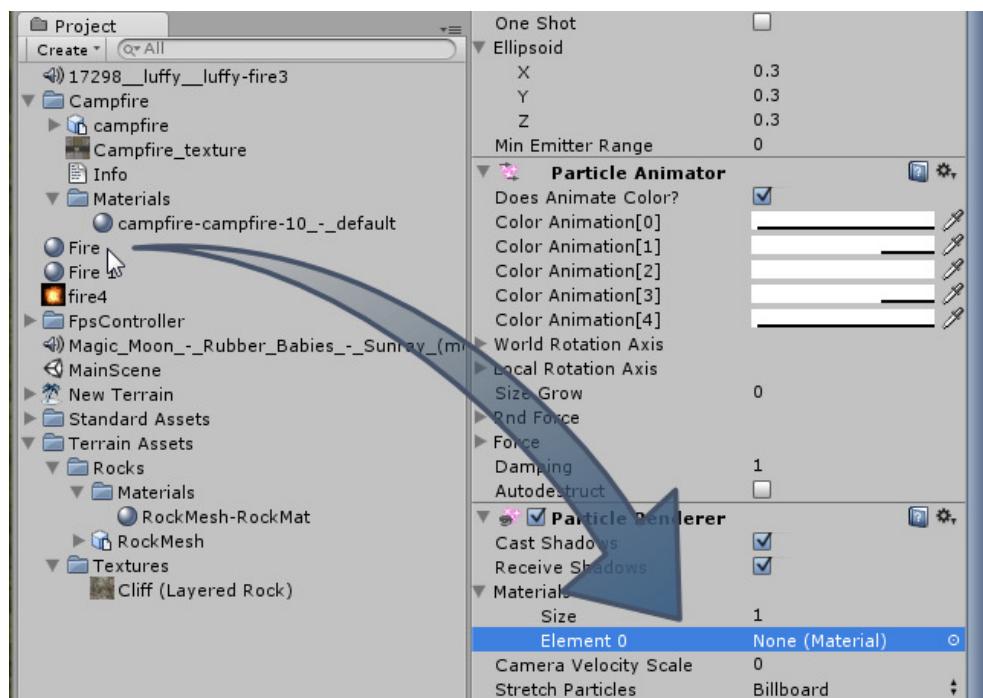
Şimdilik materyali bir kenara bırakıp partikül sisteme odaklanalım. Hierarchy panelinden "Particle System" objesini bulup seçin.

Inspector'da tonla değer göreceksiniz, bunlardan parça parça bahsedeyim.

Objemizde Transform dışında üç tane component var. Bu üç component bir uyum içinde çalışarak partikül sistemin istediği gibi görev yapmasını sağlıyor.

1. Ellipsoid Particle Emitter: kaç tane partikül olacağı, partiküllerin ne kadar süre dayanacağı, partiküllerin hareket yönü ve hızı, partiküllerin boyutu gibi değerler bu component vasıtayıyla ayarlanmakta.
2. Particle Animator: partiküllerin zamanla değişen özellikleri bu component'te ayarlanmaktadır. Örneğin partikülün rengi veya hızı zamanla değişebilir.
3. Particle Renderer: partikülü ekrana çizdirmeye yarayan component. Kendisi partiküllerin ekranda nasıl görüneceğini belirler.

İlk işimiz Particle Renderer componenti ile alakalı. Component'teki Materials değerini bulun ve solundaki üçgene tıklayarak genişletin. Fire materyalini Project panelinden tutup buradaki Element 0'a sürükleyin.



Artık partiküllerimiz pembe değil:

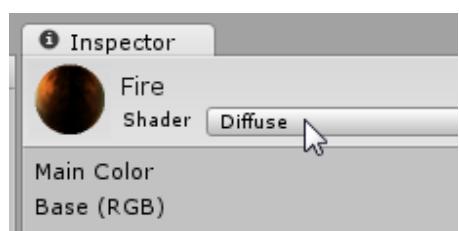


Ne yazık ki alev hiç de istediğimiz gibi durmuyor. Öncelikle alevin arkasındaki siyah arkaplandan kurtulalım.

Shaderlar

Project panelinden Fire materyalini seçin.

Inspector panelinde "Shader" diye bir kısım var.



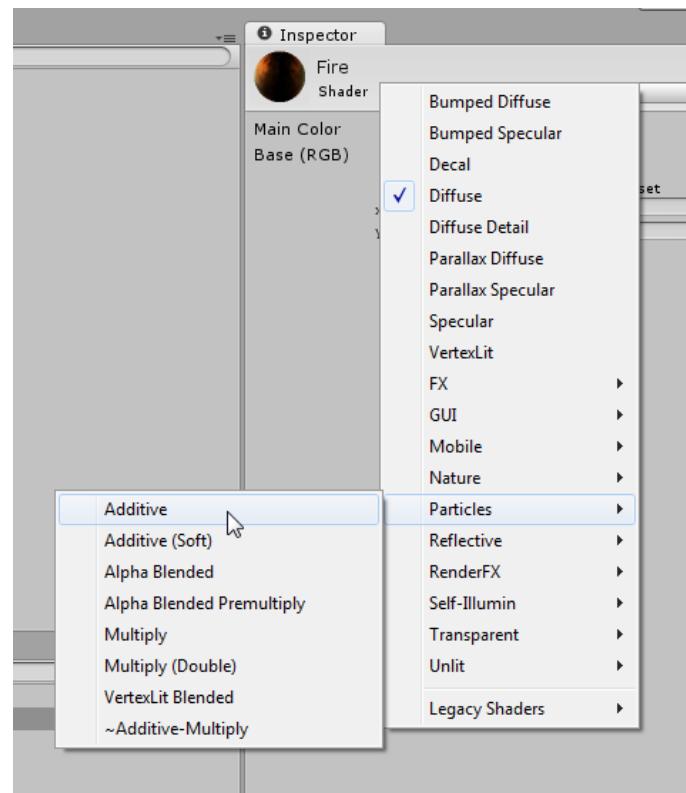
Ona tıklayın ve menüden Particles-Additive'i seçin.

Artık alevin arkasında siyah arkaplan yok.

Dahası artık alev partikülleri daha parlak duruyor. Additive shader'ın yaptığı şey birden çok partikül üst üste binince onları daha parlak göstermek.

Bu, shaderların yapabildiği şeylerden sadece biri. Shaderlar bir objenin ekranada nasıl gözükeceğini düzenleyebilen özel asset'lerdir.

Scene paneline bakarsanız alevlerin daha iyi durduğunu göreceksiniz. Aferin bize!



Particle Büyüklüğü

Partiküllerimiz çok küçük. Şimdi onları büyütelim. Particle System objesini seçin. Ellipsoid Particle Emitter component'indeki Min Size ve Max Size'ı 0.5 yapın.

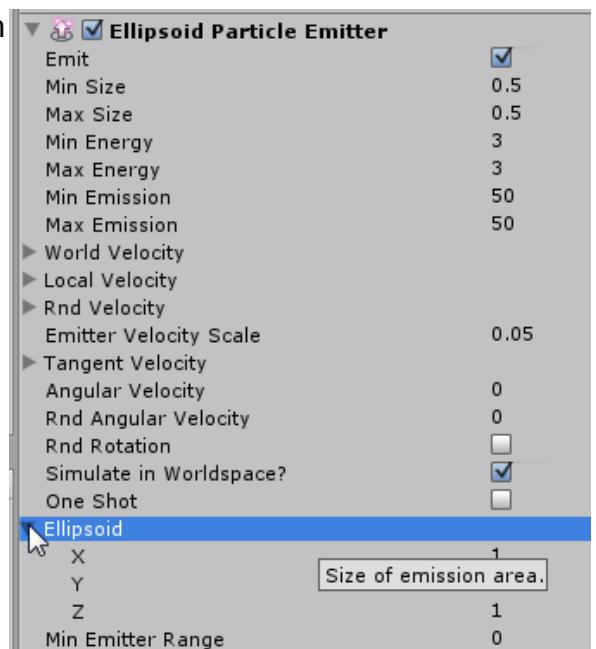


Şimdi bir başka sorunumuz var: alevler campfire objesinin çok dışına taşıyor. Alevlerin sadece campfire'in içinde olmasını sağlamalıyız.

ÇEVİRMEN EKLEMESİ: Bende alevler dışarı taşımadı o yüzden bu adımı kendim yapma gereği duymadım.

Partiküllerimiz aslında ellipsoid bir geometrinin hacminin içinde rastgele bir konumda oluşturuluyor. Biz bu ellipsoidin hacmini küçültürsek partiküllerin oluşabileceği alan daralır.

Ellipsoid parametresini Inspector'dan bulun. X, Y, ve Z değerleri ellipsoidin ne kadar geniş, yüksek ve uzun olacağını belirler. Bu değerleri 0.3 yapın.



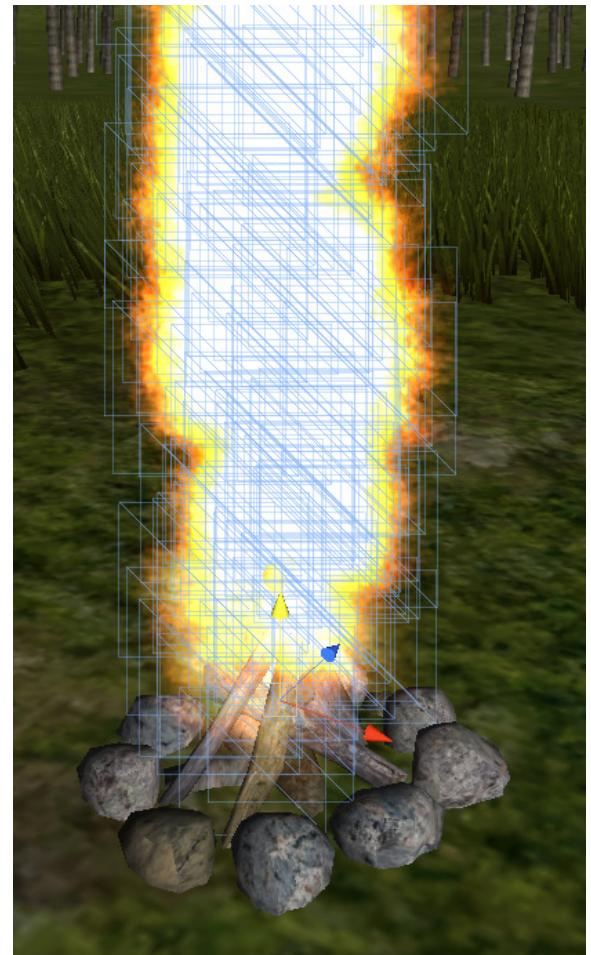
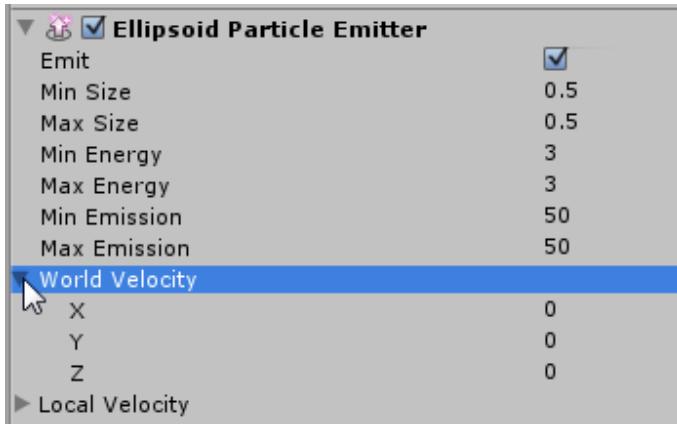


Şimdi daha iyi oldu ve kamp ateşimiz birşeye benzedi.

Partiküllerin Hızı

Şimdi de partiküllerin zamanla yükselmesini sağlayalım. Bunu yapmak için partiküllere birer hız veriyoruz.

World Velocity adında bir parametre var. Bunun Y değerini 1 yapın.



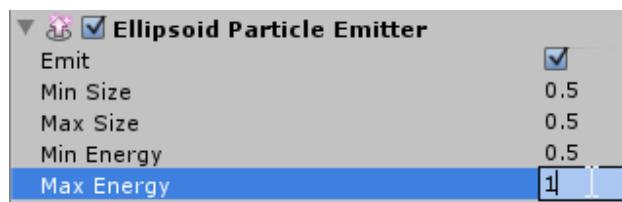
Hmm, sanırım ayarı biraz kaçırıldı! Şimdi bunu çözeceğiz.

Partiküllerin Yaşam Süresi

Sorunumuz şu: partiküller yok olmadan önce çok fazla zaman geçiyor. Partiküllerin yaşam süresini kısaltmalıyız yani.

Unity'de particle emitter component'inin "Energy" parametresi partiküllerin yaşam süresini belirler. Biz minimum ve maksimum değer belirleriz ve her partikül için bu iki değer arasında rastgele bir yaşam süresi belirlenir.

Min Energy değerini 0.5 ve Max Energy değerini 1 yapın.



En nihayetinde güzel bir alevimiz oldu!

Duman Eklemek

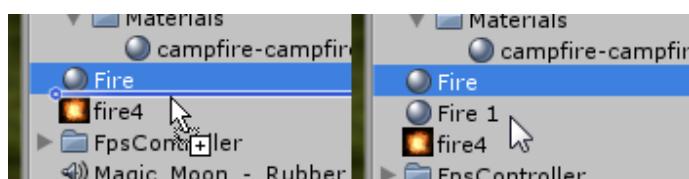
Şimdi alevimizin tepesine siyah bir duman efekti verelim.



Alev için de fire4 texturemizi kullanalım. Tek yapmamız gereken Fire materyalini klonlamak ve duman efekti için klonladığımız materyali kullanmak.

Asset'leri (Dosya) Klonlamak (Duplicate)

Fire materyalini klonlamak için onu Project panelinde bulup seçin. Ardından Ctrl + D kombinasyonunu uygulayın. "Fire 1" adında klon materyal olacak. Bu klonun adını "Smoke" olarak değiştirin (F2 tuşuyla)(Mac'te Enter tuşuyla).

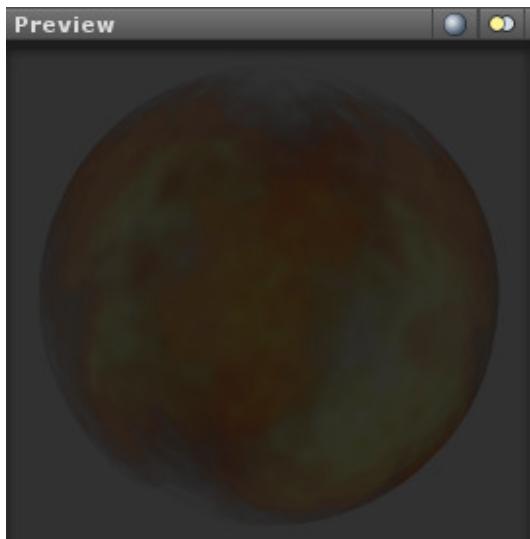


Multiply Shaderı

Şimdi elimizde aynı textureyi kullanan bir materyalimiz daha var ancak biz bu yeni materyali dumana benzetteliyoruz. Bunu yapmak için de materyale texture'yi duman gibi karartan bir shader vereceğiz.

Ama öncelikle fire4 texture'sini Project panelinden seçin ve Inspector'daki Import Settings altında yer alan "Alpha from Grayscale" seçeneğini işaretleyin. Ardından aşağıdaki Apply butonuna tıklayın.

Şimdi Smoke materyalini seçip shaderini Particles-Multiply olarak değiştirin. Multiply shaderı, Additive'in aksine, partiküller üst üste biriktikçe orayı daha karanlık gösterir.



Duman Partikül Sistemi Oluşturmak

Sahnedeki alev partikül sistemini klonlayıp onun üzerinde değişiklik yapacağız. Hierarchy'den "Particle System"ı seçip Ctrl + D ile klonlayın.

Şimdi Particle System adında iki objemiz oldu. İşler ilerde karışmasın diye hemen şimdiki tekini seçin ve "Fire Particle System" (alev) olarak yeniden adlandırın. Ötekini de "Smoke Particle System" (duman) olarak yeniden adlandırın.

Smoke Particle System'ın kullandığı materyali Smoke olarak değiştirin. Önceden yaptığımız gibi sürükle-bırak yöntemini kullanın.

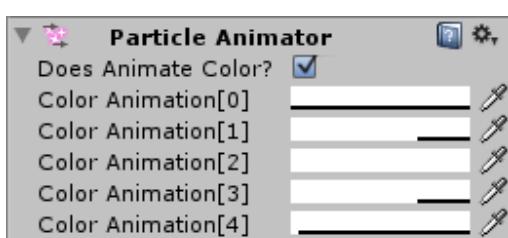
Multiply shaderı sayesinde duman koyu duruyor ama yeteri kadar değil. Dumanın siyah ya da gri olmasını sağlamalıyız, resimdeki gibi kırmızı değil.

Duman partiküllerinin rengini elle ayarlamalıyız. Bunun için doğru yer ise Particle Animator componenti.



Partiküllerin Rengini Değiştirmek

Particle Animator component'inde resimdeki gibi beyaz çubuklar göreceksiniz. Bu çubuklar partikülün zamanla aldığı renkleri gösteriyor.



Color Animation[0] partikülün olduğu andaki, Color Animation[4] de partikülün yok olduğu andaki rengi oluyor. Ortadaki değerler de partikülün yaşam süresi boyunca aldığı renkleri temsil ediyor.

ÇEVİRMEN EKLEMESİ: Duman partikülümüz siyah (ya da en azından kırmızı) ama beyaz değil, bu iş nasıl oluyor diyebilirsiniz. Çünkü Color Animation'daki renk değerleri partiküle atadığınız materyalde yer alan texture'nin **renginin üzerine ekleniyor**. Dijital dünyada bir rengin üzerine beyaz renk eklerseniz yine aynı rengi elde edersiniz.

2, 3 ve 4 için olan renkleri resimdeki gibi griye çalın:



Bir rengi değiştirmek için o renge tıklamanız yeterli.

İşlem bitince partiküllerin daha koyu ve gerçekçi renk aldığına göreceksiniz.

Eğer dumanın hemen yok olmasını istemiyorsanız yaşam süresini artırın. Yapmanız gereken Ellipsoid Particle Emitter'daki Min Energy ve Max Energy değerlerini yükseltmek; mesela 1'e 2 yapmak.



Böylece partikül sistemlerle işimizi halletmiş olduk!

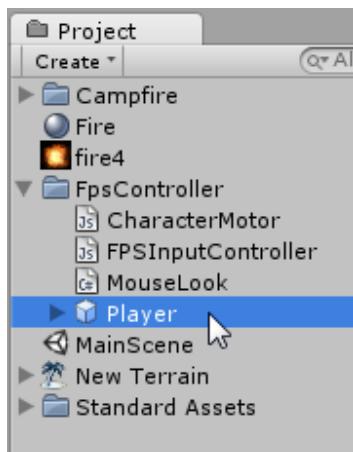
First Person Character Controller'u Kullanmak

Sizce de o kadar üzerinde uğraştığımız arazimizde yürümek güzel olmaz mıydı? Bu bölümde programlama yapmayacağız, onun yerine hazır kaçacağız. Programlama ile ileriki bölümlerde uğraşacağız.

Şimdilik arazide FPS kamera açısıyla dolaşabilmeye yarayan bir paket kullanacağız.

Winrar arşivini çıkardığınız yerdeki Packages klasöründe yer alan "FpsController.unitypackage" paketini **Assets > Import Package > Custom Package...** yolunu izleyerek import edin.

Project paneline FpsController adında yeni bir klasör gelecek. İçinde Player adında bir dosya (asset) yer almaktır. Bunu Scene paneline sürükleyip bırakın.



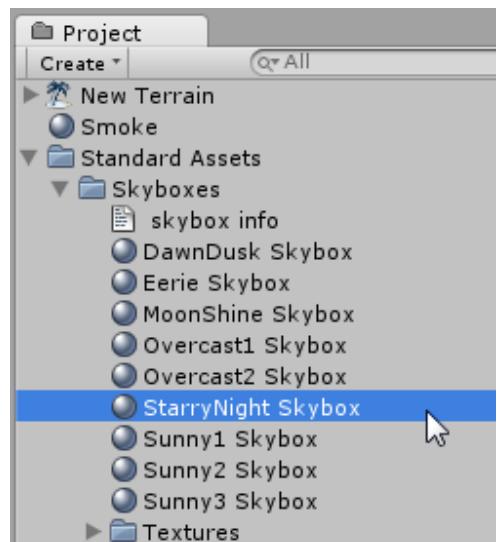
Şimdi Hierarchy panelindeki varsayılan kamerası silmemiz lazım çünkü First Person Controller'in kendi kamerası var ve biz bu iki kameralının çakışmasını istemiyoruz. "Main Camera"yı seçin ve silin. Oyunu başlatın. WASD tuşlarıyla hareket edebilir ve mouse ile etrafa bakabilirsiniz.

ÇEVİRMEN EKLEMESİ: Eğer oyunun başında karakter zeminden aşağı düşüyorsa oyunu durdurun ve **Player** objesini seçip Scene panelinde, zeminle kesinlikle çakışmayacak şekilde **yukarı taşıyın**.

Skybox (Gökyüzü) Eklemek

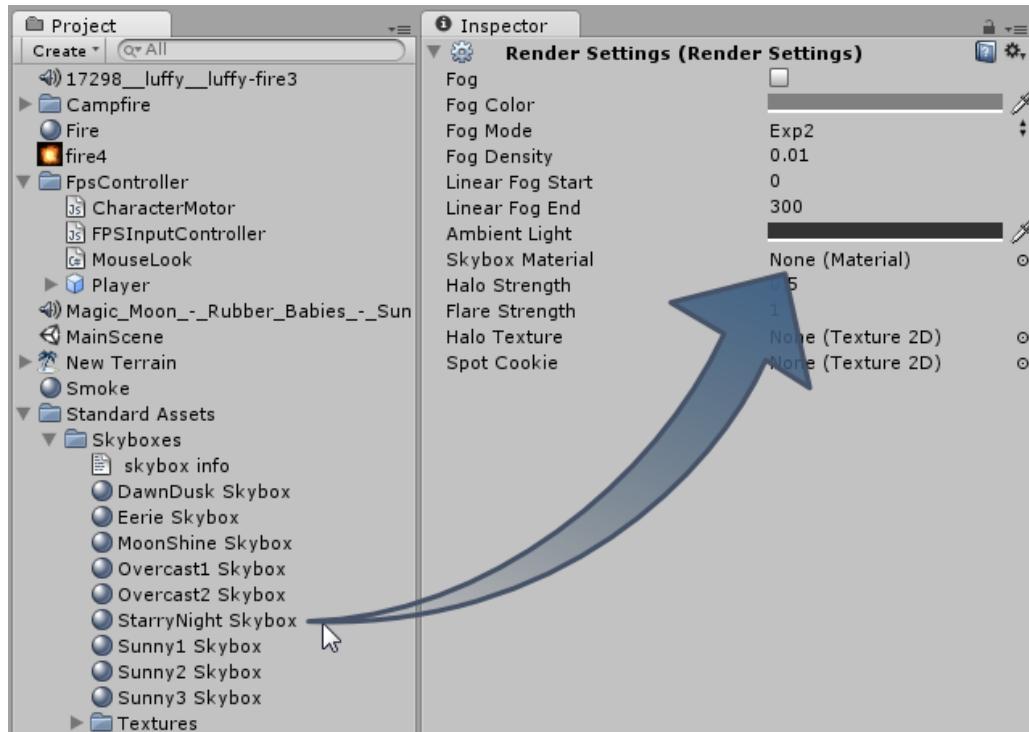
Şimdi oyuna gökyüzü ekleyelim. Unity'de bu iş için "skybox" denen bir sistem kullanılıyor.

Unity sağolsun bazı hazır skybox'lar sağlıyor bize. Bunlara erişmek için **Assets > Import Package > Skyboxes** yolunu izleyin ve paketi import edin.

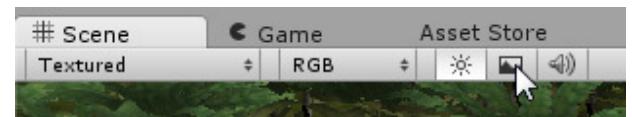


Bir skybox belirlemek için **Edit > Render Settings** yolunu izleyerek Render Settings'e erişin.

Buradan kullanmak istediğiniz skybox'ı tutup "Skybox Material" kısmına sürükleyin. Ben StarryNight skybox'ını kullanacağım.



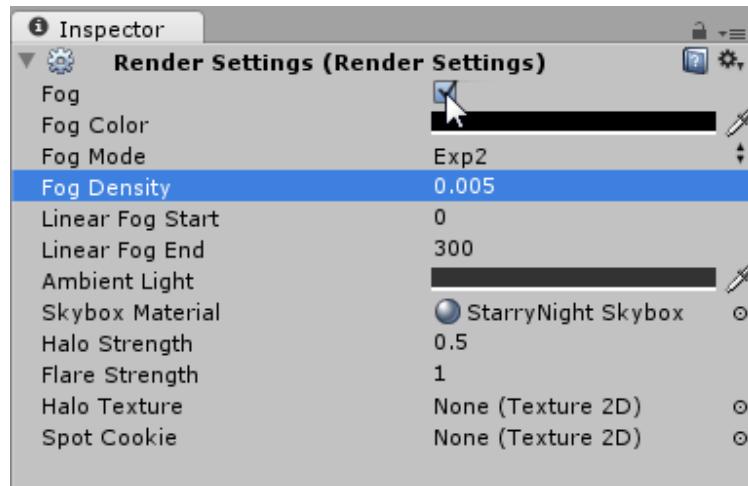
Değişikliği Scene panelinde gökyüzüne bakarak görebilirsiniz. Eğer bir fark yoksa değişikliği önizlemeye yarayan butona tıklamayı deneyin:



Sis (Fog) Eklemek

Şimdi Render Settings'teki bir başka seçeneğe bakalım.

Baştaki Fog seçeneğini açın, Fog Color'u siyah yapın ve Fog Density'i 0.005 yapın.



Farkı şu iki resmi kıyaslayarak görebilirsiniz:



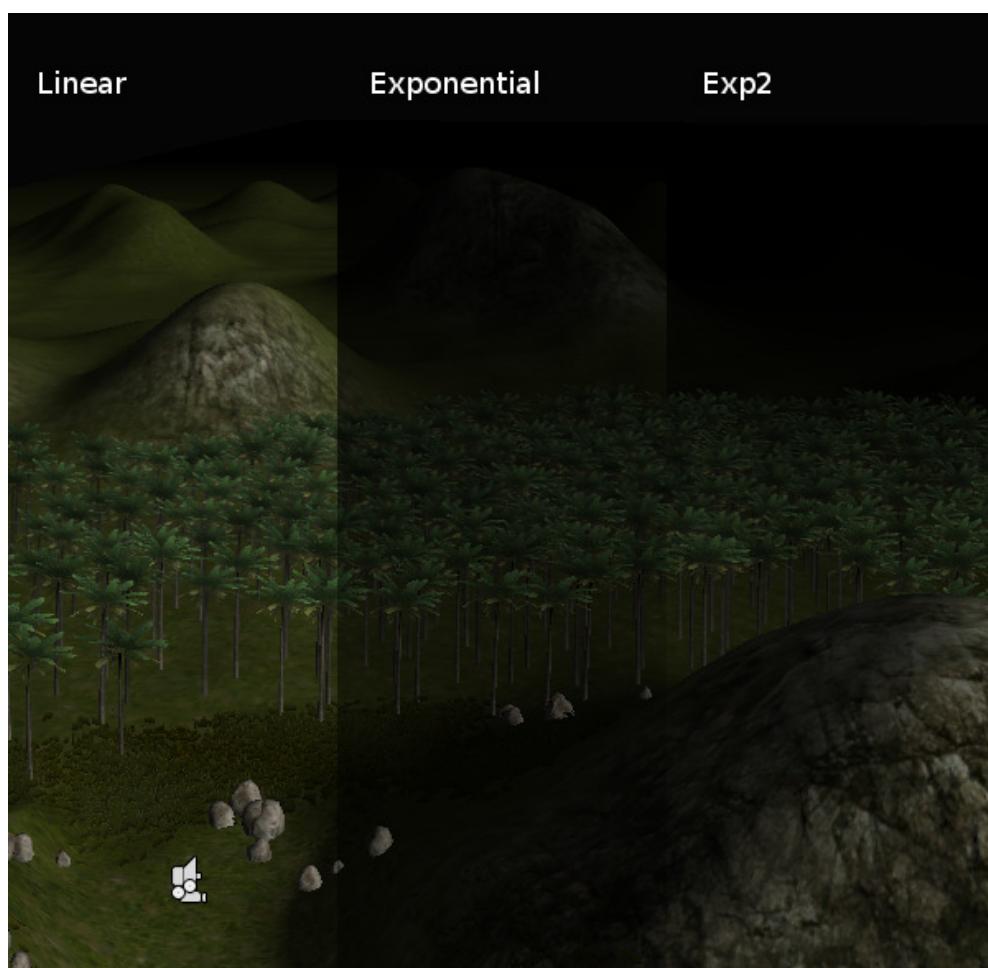
İpucu

Eğer sisin rengi gökyüzünün rengiyle uyuşursa ortaya çok güzel bir manzara çıkabilir.



Fog Mode

Fog Mode ayarı sisin etrafa nasıl dağılacagını belirler.



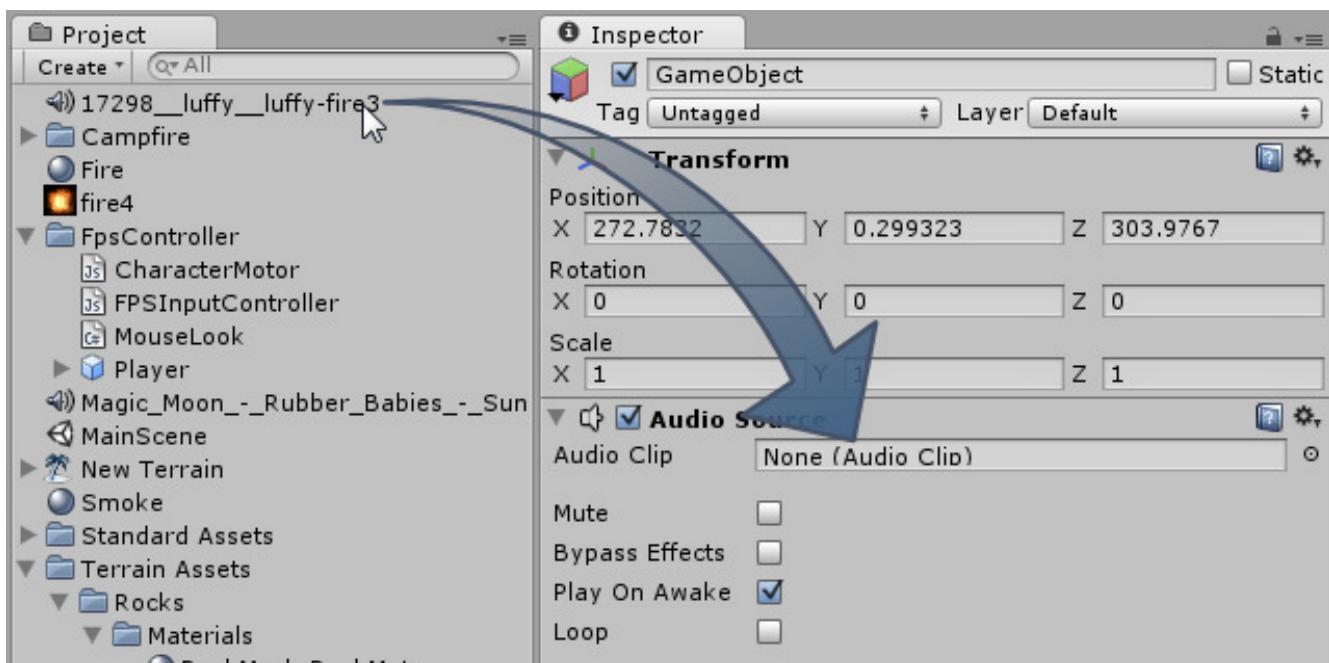
Ses Ekleme

Bence kamp ateşimize bir de ses efekti verelim; böylece iyice şahane olsun.

Önce ses dosyasını projemize eklemeliyiz. Winrar arşivini çıkardığınız yerdeki Sounds klasöründeki "17298_luffy_luffy-fire3.wav" dosyasını Assets > Import New Asset... ile projenize dahil edin.

Yeni bir boş obje oluşturun (GameObject->Create Empty). Objeyi kamp ateşiyle aynı yerde olacak şekilde konumlandırın. Sesin bu obje çalacak ve şu anda sesin, ateşin olduğu yerden çıkıştı en mantıklı şey gibi duruyor. Component > Audio > Audio Source yapın. Audio Source componenti bir objenin ses çalabilmesini sağlar.

Biraz önce import ettiğimiz ses dosyasını Audio Source'taki Audio Clip kısmına sürükleyin:



Loop seçeneğini işaretleyin; böylece ses dosyası bitince tekrar başa sarıp çalışmaya devam etsin.

Oyunu çalıştırın. Kamp ateşinin yanına yaklaşınca ateşin çıktıığı sesi duyabilirsiniz. Ateşten uzaklaşıkça sesin şiddeti azalacak.

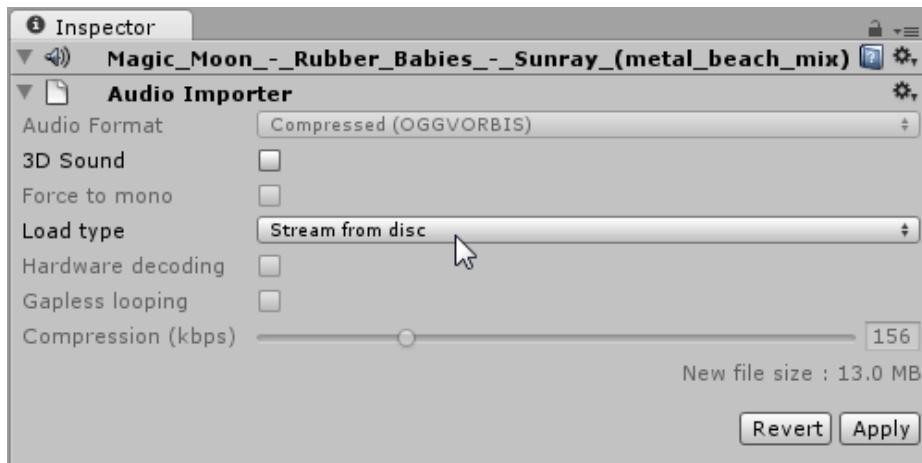
ÇEVİRMENDEN İPUCU: Bir objeyi istediğiniz yere yerleştirirken (örneğin ses objesini kamp ateşinin olduğu yere taşıırken) işlemi hızlandırmak için bir yöntem var. Scene panelinde kameralı, objeyi taşımak istediğiniz yere götürün. Ardından taşımak istediğiniz objeyi (ses objesi) seçip GameObject->Align With View yolunu izleyin. Böyle yaptığınızda seçtiğiniz obje kameralan bulduğu konuma ıshınlanır.

Müzik Ekleme

Müzik eklemek için de aynı şeyleri yapacağız. Öncelikle müzik dosyasını import edeceğiz. Winrar arşivinin orada Music klasörü var. İçindeki müziklerden istediğiniz herhangi birini projenize import edin.

Import ettiğiniz ses dosyasını seçin ve Inspector'a dikkatinizi verin.

Müziğin hep aynı şiddette çalmasını istiyoruz. Yani ateş sesinin aksine karakteri nereye hareket ettirirsek ettirelim şiddeti değiştirmeyecek. Bunu yapmak için 3D Sound'u kapatın. Müzik büyük bir dosya boyutuna sahip olduğu için "Load type" olarak "Stream from disc"'i seçmeniz önerilir. Böylece oyun başlarken müziğin yüklenmesi için daha az süre beklersiniz. Ardından Apply'a basın.



Yine boş bir obje oluşturun (GameObject->Create Empty) ve Audio Source componenti verin. Müzik dosyasını Audio Clip kısmına sürükleyin.

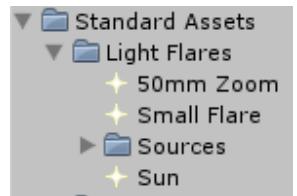
Ses dosyası 3D olmadığı için artık sesi çalan objenin konumunun hiçbir önemi yok. Ses her zaman için aynı şiddette çalışacak.

Lens Flare Kullanmak

Oyunumuza son ekleyeceğimiz özelliğin ismi lens flare. Resimde örnek bir lens flare görebilirsiniz:

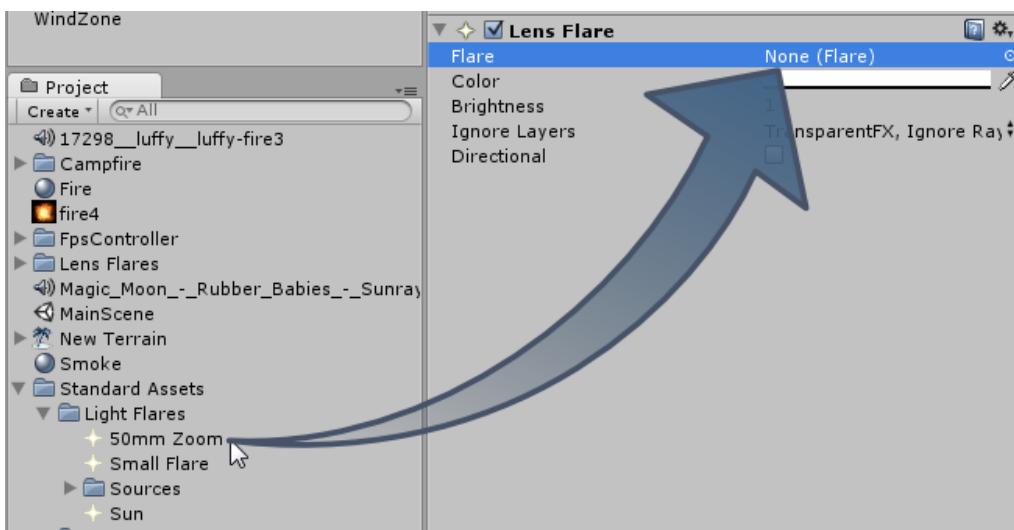


Unity bize üç tane hazır lens flare sunmaktadır. Onlara erişmek için önce **Assets > Import Package > Light Flares** ile ilgili paketi import edin.

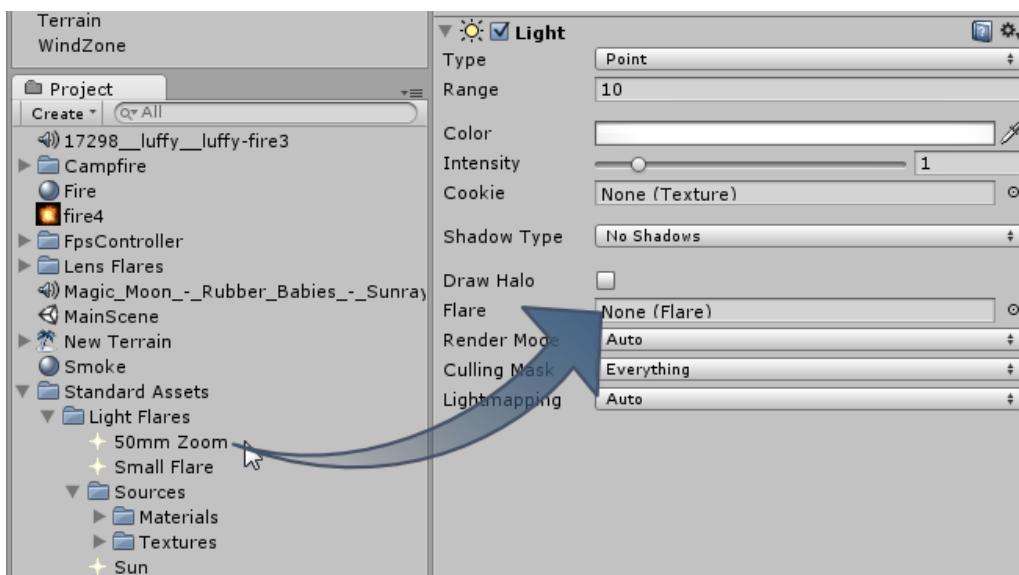


Lens flare kullanmak için de yeni bir boş objeye ihtiyacımız var. Bir tane oluşturun ve **Component > Effects > Lens Flare** ile Lens Flare componenti verin.

Ardından az önce import ettiğiniz lens flare'lerden birini Lens Flare component'indeki Flare kısmına sürükleleyin.



Lens flare'ler genelde ışığın gözü alması durumunu simüle ettiği için ışık objelerine verilir. Bu yüzünden ki ışık objelerinin Light component'lerinde de lens flare koymak için Flare adında bir değer bulunur. İsterseniz lens flare'i buraya sürükleyebilirsiniz.



Peki sizce neden o kadar uğraşıp yeni bir obje oluşturduk ve ona Lens Flare componenti verdik? Çünkü herhangi bir ışıktan bağımsız olarak bu objeyi istediğimiz yere sürükleyip sanki orada ışık varmış efekti verebiliriz, gerçekte bir ışık olmasa bile.

Bu avantajı örneğin skybox'taki güneş veya ay resimlerinin ışık saçtırmış gibi durması için kullanabilirsiniz.

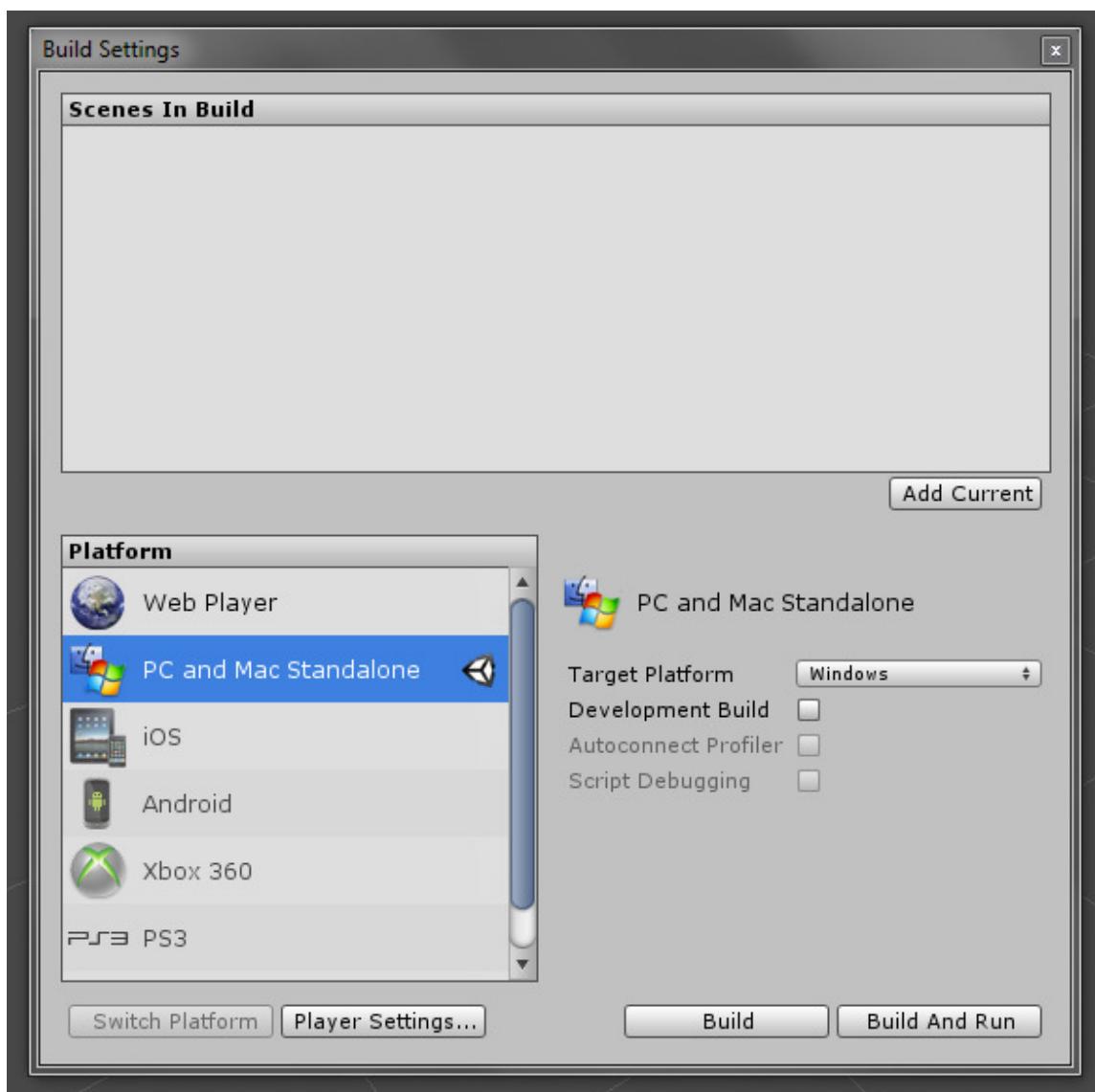
Bu durumda Lens Flare component'inde yer alan Directional seçeneğini işaretleyin. Çünkü directional light güneş ışığını (ve haliyle ay ışığını) simüle eder ve biz lens flare'i güneş veya ayın parıldaması efektini vermek için kullanacağız.

Lens Flare'de Directional seçeneğini işaretleyince Lens Flare'in konumu önemini yitirir. Tıpkı Directional Light gibi, sadece objenin eğimi önem arzeder. Örneğin lens flare objesi dimdik yere bakıyorsa sanki güneş tam yukarıdaymış da güneş ışınları dimdik aşağı yönde geliyormuş gibi simüle edilir ve göz alma efekti sadece dimdik yukarı bakınca görülebilir.

Oyunu .EXE Olarak Build Etmek

Bu bölümde çok basit bir oyun oluşturduk. Şimdi bunu arkadaşlarımıza gösterme zamanı. Oyunu .exe olarak derleyip arkadaşlarınıza sunabilirsiniz, böylece Unity'si olmayan arkadaşlarınız da oyunu oynayabilir.

File > Build Settings... yolunu izleyin ya da Ctrl + Shift + B kombinasyonunu uygulayın.



Gelen pencere, oyunu derlerken (build) kullanılır.

Hangi sahnelerin (scene)(level) .exe halindeki oyuna dahil olacağını da buradan belirlememiz gereklidir. "Add Current" butonuna basarak sahnenin listeye eklenmesini sağlayın.

"PC, Mac & Linux Standalone"un seçildiğinden ve "Target Platform"un "Windows" olduğundan emin olun.

Şimdi "Build" butonuna tıklayın. EXE dosyası için bir isim belirleyip işlemi tamamlayın.

Unity'nin oyunu derlemesini bitirmesini bekleyin. Ardından .exe dosyanızı oluşturduğunuz klasörü gösteren bir pencere gelecek.

Bu klasörün içinde .exe'ye ek olarak oyununuzun ismiyle başlayıp "_Data" ekiyle sona eren bir klasör daha var. Oyunun çalışması için gerekli tüm dosyalar bu klasörde depolanmaktadır. Oyunu başkalarıyla paylaşırken .exe ile birlikte bu klasörü de paylaşmayı unutmayın.

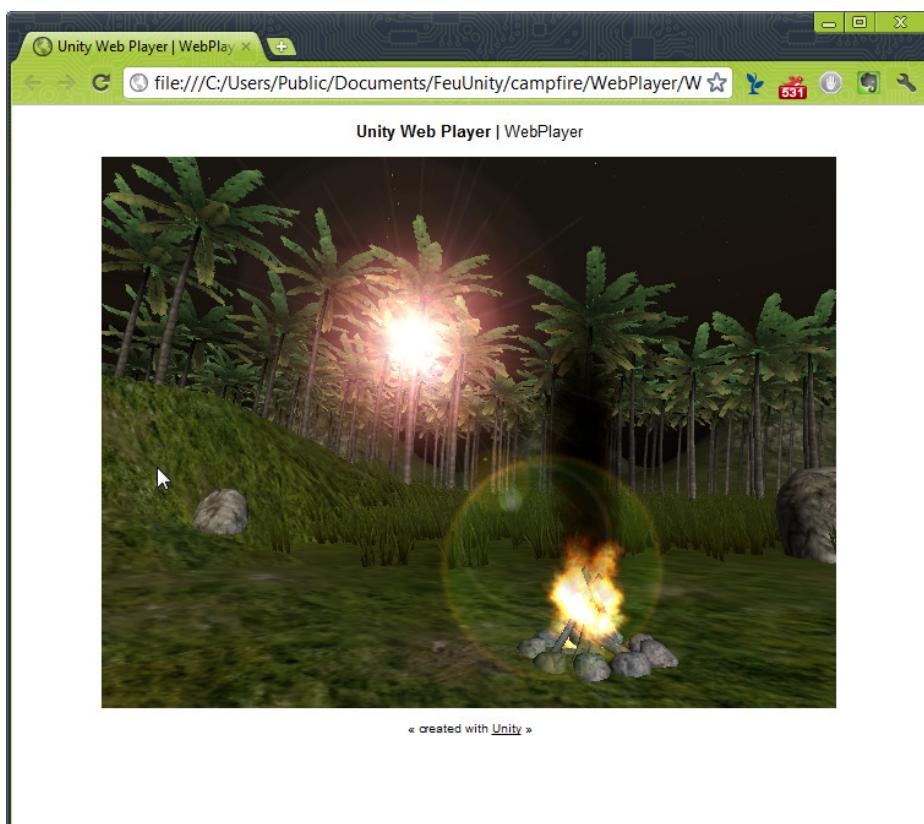
Oyunu Web Tarayıcısında Çalışacak Şekilde Build Etmek

Dilerseniz oyununuzu web tarayıcısında çalıştırabilirsiniz. Performans .exe kadar iyi olmayacağından emin olun ama ilginç bir deneyim olacağı kesin.

File > Build Settings...'ten "Web Player"ı seçin ve "Switch Platform'a tıklayın. Bir yükleme ekranı çıkarsa biraz bekleyin. Sonra "Build" butonuna tıklayın. Yeni bir klasör oluşturup hedef olarak bu klasörü seçin.

Build işlemi bitince klasör açılacak. Klasörde ".unity3d" ve ".html" uzantılı iki dosya yer almaktadır.

HTML dosyasına çift tıklayarak onu tarayıcınızda açın. Harika!



Özet Geçecek Olursak...

Bu bölümde Unity'nin çeşitli özelliklerini gördük: arazi oluşturma, 3D model ve başka asset'ler import etme, partikül sistemi kullanma vb.

Artık doğada geçen basit bir level yapmayı ve onu ağaçlarla, taşlarla, gökyüzüyle ve hatta rüzgarla bezemeyi biliyorsunuz.

Sonraki bölümde kod yazmaya başlayacağız. Objelerinizin davranışlarına kodla nasıl müdahale edebileceğinizi öğreneceksiniz.

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

Unity 3D İle Oyun Programlama

Bölüm 3: Script Yazmaya Giriş



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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Bu bölümde Unity'de kod yazmaya başlayacağız. Unity'de yazılan kodlar diğer oyun motorlarına göre biraz değişktir çünkü Unity component bazlı bir sistem kullanmaktadır.

Scriptler

Unity'de yazdığınız her script aynı zamanda tıpkı Transform, Rigidbody ya da Light gibi birer component'tir. Scriptin çalıştırılması için onu, çalışmasını istediğiniz objeye verirsiniz.

Derse temiz bir proje üzerinden devam edelim. "Lesson3" adında yeni bir proje oluşturun.

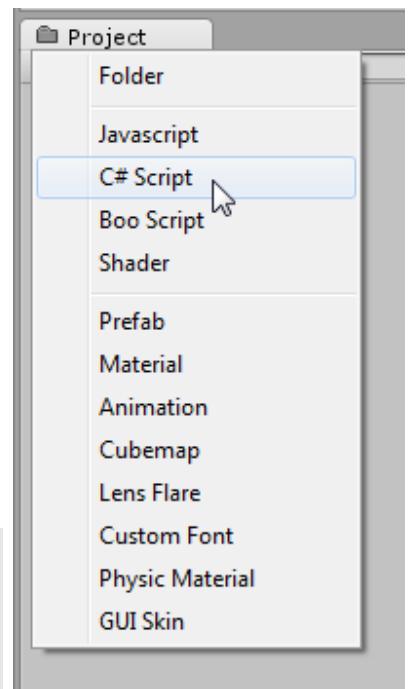
Şimdi bir script oluşturalım. Project panelindeki "Create" butonuna tıklayın ve "C# Script'i seçin.

Scriptin ismini "Rotate" yapın.

Rotate scriptine çift tıklayın. Varsayılan olarak MonoDevelop açılacaktır. MonoDevelop vasıtısıyla kod yazabilir/düzenleyebilirsiniz.

Şimdi ilk kodumuzu yazalım. Sarı renkle vurgulanmış kodu siz de scriptinizin Update fonksiyonuna yazın:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Rotate : MonoBehaviour
05 {
06     // Use this for initialization
07     void Start()
08     {
09
10     }
11
12     // Update is called once per frame
13     void Update()
14     {
15         transform.Rotate(2, 0, 0);
16     }
17 }
```



Her yeni C# scripti bazı hazır kodlar yazılmış halde gelir.

Bu hazır kodlardan ilki en baştaki satır. Bu satırda "UnityEngine" kütüphanesini scriptimizde kullanacağımızı belirtiyoruz. Böylece kütüphanenin sunduğu fonksiyonlardan, değişkenlerden faydalanaibileceğiz.

Scriptimizle aynı isimli bir class'ımız yer almaktır. Class isminin script ismiyle birebir aynı olması çok önemli yoksa Unity istenmeyen hatalar verebilir.

Bizim class'ımız varsayılan olarak MonoBehaviour class'ını extend ediyor. MonoBehaviour, UnityEngine kütüphanesinde yer alan bir class. Unity'de çok kullandığımız fonksiyonlar, değişkenler hep bu class'te tanımlanmış vaziyette.

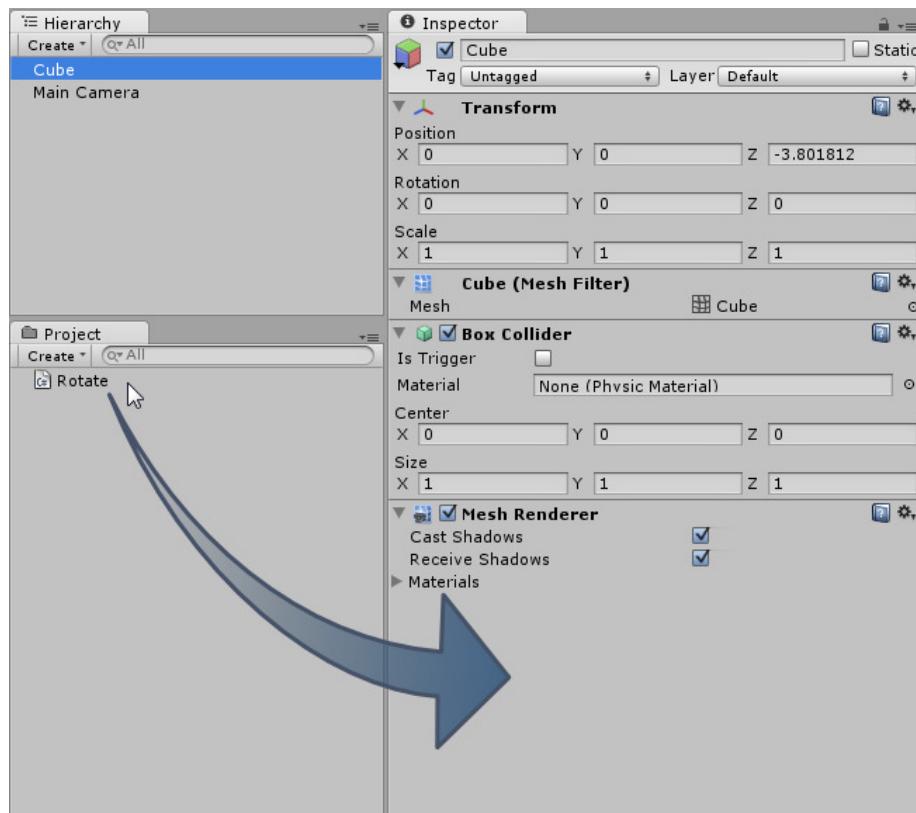
Update fonksiyonunda eriştiğimiz "transform", MonoBehaviour'un değişkenlerinden birisi. "transform" ile bir objenin pozisyonunun, eğiminin ve boyutunun depolandığı Transform component'ine erişiriz. Ardından script vasıtıyla Transform component'indeki bu değerleri değiştirebiliriz.

Yazdığımız kod çok basit: Update fonksiyonu her çağrıldığında kod hangi objedeyse o objeyi X ekseninde 2 derece döndürüyoruz.

Update fonksiyonu her frame'de (kare) otomatik olarak çağrılır.

GameObject > Create Other > Cube ile sahnede bir küp oluşturun. Objeyi kameranın görebileceği bir yere yerleştirin.

Küp seçiliyken Rotate scriptini Project panelinden sürükleyerek Inspector panelinin altındaki boşluğa bırakın. objeye component olarak atanacak.



Oyunu başlattığınızda küpün durmadan döndüğünü göreceksiniz. Küpü daha iyi görmek için sahneye bir ışık koyabilirsiniz (**GameObject > Create Other > Point Light**).

Rotate scriptini istediğiniz kadar objeye verebilirsınız. Birkaç tane daha obje oluşturun. Mesela bir kapsül oluşturun (**GameObject > Create Other > Capsule**). Rotate scriptini kapsüle de verip oyunu başlatın. Hem küp hem de kapsül sürekli döneceklerdir.

Unity'de scriptlerin işleyiş mantığı böyle. Hangi objelerin scriptte verdığınız komutları yerine getirmesini istiyorsanız o objelere o script'i verirsiniz. Eğer bir scripti hiçbir objeye vermezseniz o script bir işe yaramaz. Bu duruma bazı istisnalar var tabii (static class'lar).

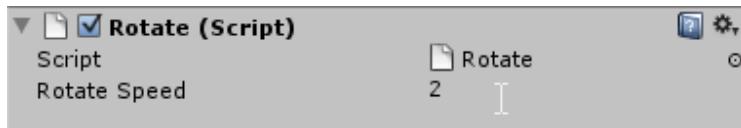
Değişkenlerle Çalışmak

Az önceki kod objeyi her Update çağrılığında 2 derece döndürüyordu. Buradaki 2 değeri bir sabit sayydı. Ama genelde böyle sabit sayılarla çalışmazsınız. Şimdi 2 dereceyi bir değişken ile değiştirelim:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Rotate : MonoBehaviour
05 {
06     public float _rotateSpeed = 2;
07
08     // Use this for initialization
09     void Start()
10     {
11
12     }
13
14     // Update is called once per frame
15     void Update()
16     {
17         transform.Rotate(_rotateSpeed, 0, 0);
18     }
19 }
```

Aklınızda bulunsun, bu derste değişken isimlerinin başına hep `_` işaretini koyacağım.

Scripti kaydedip Unity'e geri dönün. Inspector'da Rotate Speed değeri belirecek.



İşin güzel yanı Rotate Speed'i direkt Inspector üzerinden değiştirebilmeniz. Yani scriptte herhangi bir değişiklik yapmanız gerekmiyor.

Inspector'daki değerin scriptteki varsayılan değere göre üstünlüğü bulunmakta. Eğer scripti açıp `_rotateSpeed`'i 2'ye değil de 3'e eşitleseniz bile Rotate Speed'in değeri yine de Inspector'daki 2 değeri olarak işleme sokulacaktır.

Private Değişkenler

Scriptlerde public değişkenlerle çalışmak genel olarak iyi bir şey değildir. Örneğin `_rotateSpeed`'in private olması script yazma prensiplerine daha uygun. Private değişkenlerde yaşanan sorun ise değerin Inspector'da gözükmemesi. Peki bu sorunu nasıl aşarız?

Neyse ki Unity'de private bir değişkenin değerini Inspector'da görüntülemeye zorlayan bir komut mevcut. `_rotateSpeed` değişkeninin tanımlandığı (declaration) kısmı şöyle değiştirin:

```
01 [SerializeField]
02 float _rotateSpeed = 2;
```

SerializeField, bir değişkenin değerini Inspector'da görünmeye zorlayan bir anahtar kelimedir. Değişkenin başındaki "public" takısını sildiğimiz için de artık değişkenimiz private olmuş oldu.

Oyun Hızına (Frame Rate) Adapte İşlemi

Aslına bakarsanız Update fonksiyonunda objeyi belli bir derece döndürmek iyi birsey değildir. Oyunu yavaş bir bilgisayarda oynarken Update fonksiyonu daha az kez çağrılmaktır. Çünkü yavaş bilgisayarın gücü bir saniyede daha fazla Update çağrırmaya yetmemektedir.

Bu durumda daha az Update çağrıldığı için de obje daha az kez dönecektir (Rotate). Hızlı bilgisayarlarda ise bunun aksine bir saniyede daha fazla kez Update çağrıldığından obje daha çok dönecektir. Yani objenin dönme miktarı bilgisayarın hızıyla doğru orantılı olacaktır.

Bu doğru orantı sorununu çözmek ve objenin her türlü sistemde (hızı ne olursa olsun) aynı sürede hep aynı miktar dönmesini sağlamak için adapte işlemi yapacağımız. "her karede 2 derece döndür" demek yerine "her saniyede 2 derece döndür" diyeceğiz. Kodda yaptığımız değişiklik şöyle:

```
01 void Update()
02 {
03     transform.Rotate(_rotateSpeed * Time.deltaTime, 0, 0);
04 }
```

Burada _rotateSpeed'in değerini Time.deltaTime ile çarpıp objeyi o kadar derece döndürüyoruz. Time class'ı statiktir ve UnityEngine kütüphanesinde yer alır. **Bu class'taki deltaTime değişkeni bir float'tur ve en son iki kare (frame) arasında kaç saniye geçtiğini depolar.**

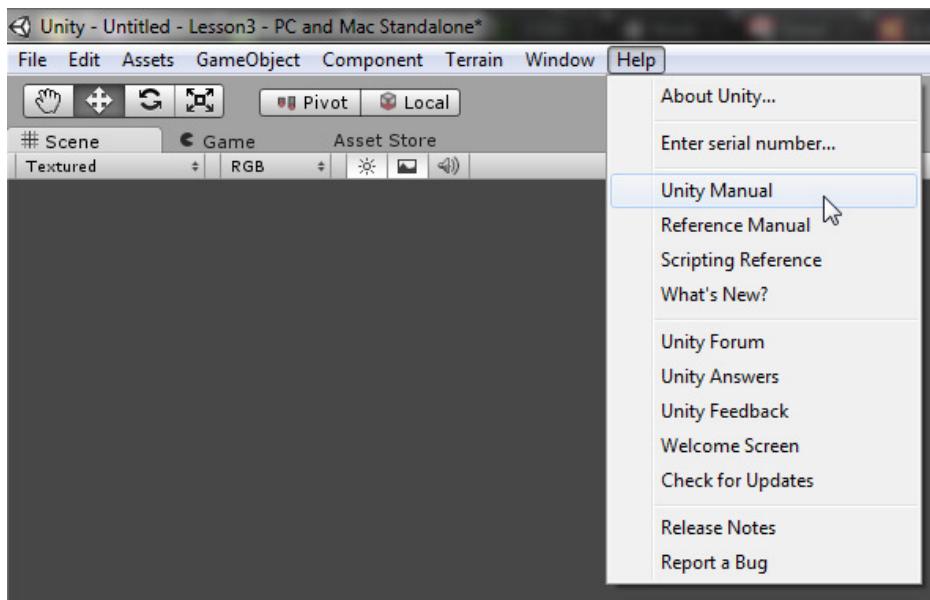
Artık her karede değil her saniyede rotateSpeed kadar döneceğimizden rotateSpeed'in değerini Inspector'dan artırmayı düşünebilirsiniz.

Online Destek Hakkı

Oyunumuzu yapmaya başlamadan önce faydalı bulduğum bazı kaynakları paylaşacağım.

Unity'nin Kendi Kullanma Kılavuzu

Unity, internet gerektirmeden okunabilen bir kullanma kılavuzu ve script referans arşivi ile birlikte gelmektedir.



Kullanma kılavuzuna Help > Unity Manual yoluyla, script referans arşivine Help > Scripting Reference yoluyla ulaşabilirsiniz.

Resmi Forum Sayfası

Unity hakkında sorularınıza şurada cevap arayabilirsiniz: <http://forum.unity3d.com/>.

A screenshot of the Unity Community forum website. The top navigation bar includes links for Unity, Gallery, Store, Support, and Company. Below that are Documentation, Resources, and Community. The main header says "Unity Community" and has a search bar. The "Forum" section shows a "General" category with several threads: "Announcements" (9,505 posts in 169 topics, 12 Hours Ago), "Collaboration" (38,220 posts in 5,609 topics, 44 Minutes Ago), "Gossip" (81,464 posts in 6,533 topics, 6 Minutes Ago), "Showcase" (86,693 posts in 5,827 topics, 7 Minutes Ago), "Commercial Work" (4,859 posts in 1,203 topics, 1 Hour Ago), and "Assets and Asset Store" (14,421 posts in 1,000 topics, 27 Minutes Ago). Below this is another "Unity" category with "Unity Support" (125,608 posts in 26,599 topics, 13 Minutes Ago) and "Scripting" (152,764 posts in 30,588 topics, 2 Minutes Ago).

Wiki Sayfası

Unity'nin hayranları tarafından yapılmış wikipedia tarzı bir destek sitesi olduğunu biliyor muydunuz? Bu siteye şuradan ulaşabilirsiniz: <http://www.unifycommunity.com/wiki/>. Burada bulduğunuz kodlar ticarî kullanıma açık olabilir de olmayı bilir de. İlgilendiğiniz kodun açıklamasını dikkatle okumanız yararınıza olur.

Soru&Cevap Sitesi

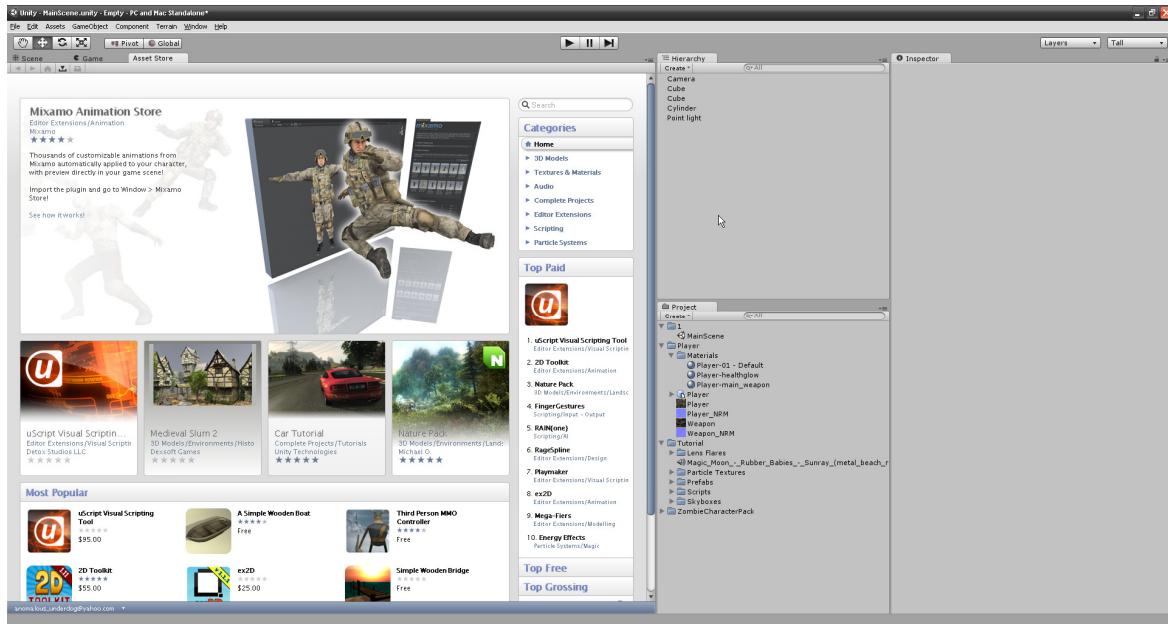
Unity'nin Stack Exchange sitesine benzer bir sitesi var: [http://answers.unity3d.com/.](http://answers.unity3d.com/)

All questions				active	newest	hottest	most voted
0 votes	1 answers	65 views	Preventing characters falling through world when altering terrain at runtime	2 mins ago mouseman919 1			
3 votes	2 answers	33 views	Point Light not acting correctly	4 mins ago syclamoth 205			
0 votes	3 answers	54 views	How to display *.txt file on the cube model?	9 mins ago leokvn 1			
0 votes	1 answers	73 views	Big FPS drop when using SetHeights on terrain	10 mins ago syclamoth 205			
0 votes	1 answers	7 views	gameobject movement problem	12 mins ago syclamoth 205			
2 votes	1 answers	277 views	Particle Colliders and manual particles	15 mins ago atm 486			
0 votes	-1 answers	23 views	How to handle saving dozens of tiny objects	19 mins ago syclamoth 205			
0 votes	1 answers	20 views	Screen VS Viewport What is the difference	22 mins ago syclamoth 205			
1 votes	1 answers	15 views	Orient vehicle to ground normal (terrain hugging)	22 mins ago aldonaleto 5.3k			
0 votes	1 answers	28 views	Performance issue loading resource	29 mins ago cj.coimbra 31			

Unity Asset Store (Dükkan)

Asset Store, Unity'de kullanmak üzere her türlü asset'i alıp satabileceğiniz bir mekandır. Bu asset'ler 3D modeller, hazır scriptler, plug-inler, müzik ve ses efektleri gibi çeşitli kategorilere ayrılmaktadır. Dükkanındaki bazı asset'ler tamamen ücretsizdir.

Window > Asset Store ya da Ctrl + 9 kombinasyonu ile Asset Store'a Unity içinden erişebilirsiniz.



Özet Geçeceğ Olursak...

Unity'de kod yazmaya bu dersle başlamış olduk. Private da olsa bir değişkenin değerini Inspector'da göstermeyi öğrendiniz.

Ayrıca sorunlarınızı şifa olabilecek birkaç site öğrendiniz.

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

Unity 3D İle Oyun Programlama

Bölüm 4: Player Scriptine Giriş



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

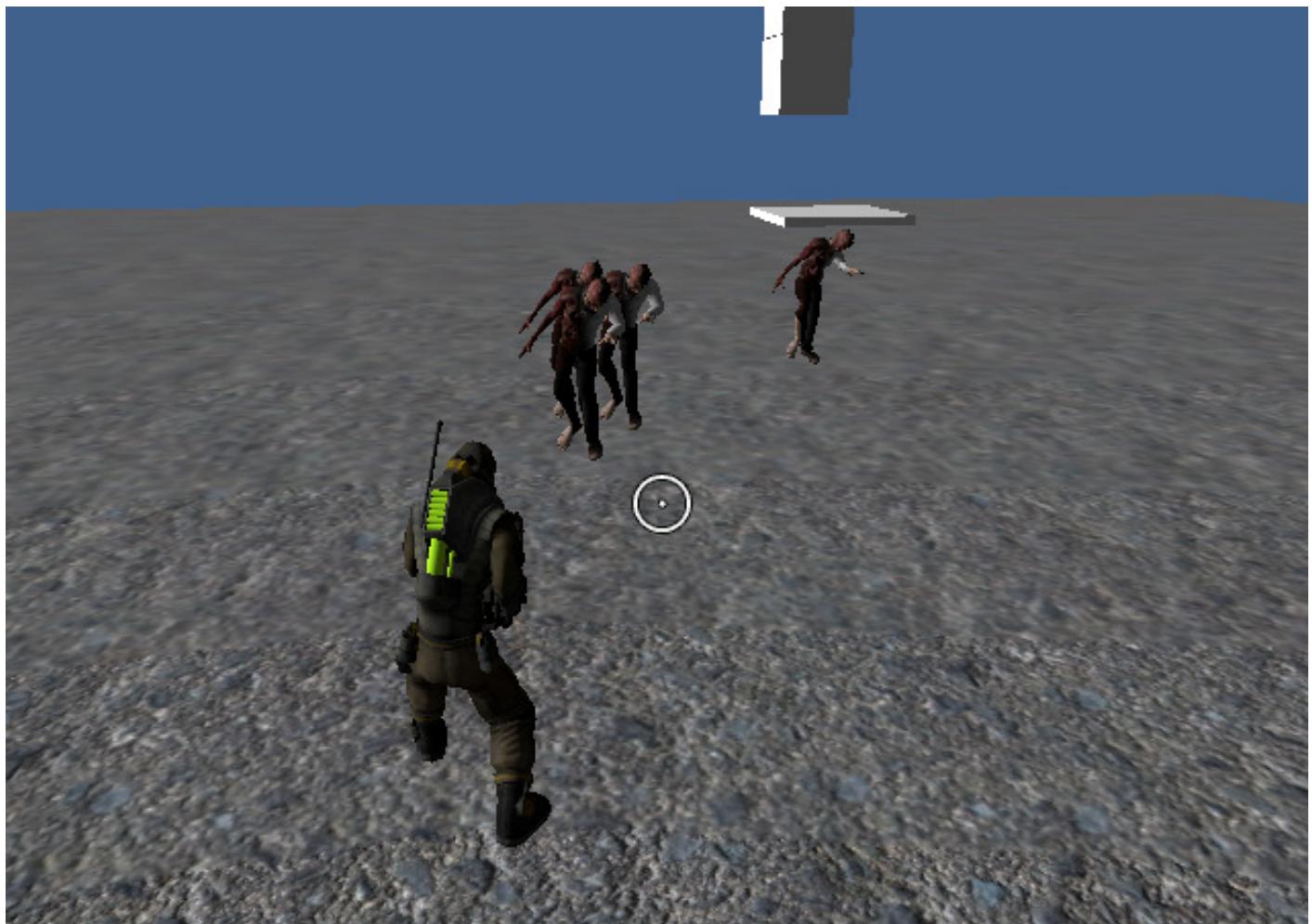


All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with
Creative Commons Attribution-NonCommercial 3.0 Unported

Bu bölümle birlikte oyunumuzu yapmaya başlıyoruz. Yapacağımız oyun TPS bakış açısı ile oynanan bir zombi-shooter olacak.



Bu dersin genel olarak amacı programlamaya odaklanmak. Bu yüzden oyunumuzda hep hazır modeller, kaplamalar, ses dosyaları vb. kullanacağız. Örneğin resimde gördüğünüz karakter ve zombi modelleri Unity Asset Store'daki ücretsiz modeller arasından alınmıştır.

Dersin bu bölümünde oyuncuyu (player) oluşturacak ve klavye ile hareket ettirmeye çalışacağız.

“TheGame” adında yeni bir proje oluşturun.

Oyun Alanı

Önce karakterimizin yürüyeceği zemini oluşturalım. Bir küp oluşturun (**GameObject > Create Other > Cube**) ve onu (0, 0, 0) koordinatlarına yerleştirin (Position değerlerini değiştirerek). Scale değerini (50, 1, 50) yaparak küp objesini boyutlandırın.

Küpün ismini “Ground” olarak değiştirin.

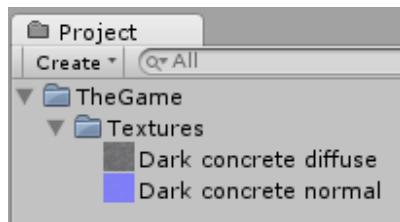
Şimdi **GameObject > Create Other > Directional Light** ile sahneye ışık ekleyin ve ışığı (30, -30, 0) şeklinde döndürün.

Zemin gri renkte güzel durmuyor. Onun üzerine bir kaplama atalım. Bunun için Winrar arşivinin oradaki Images klasöründe yer alan “Dark concrete diffuse.png” ve “Dark concrete normal.png” resimlerini projenize import edin.

Project Panelini Düzenli Tutmak

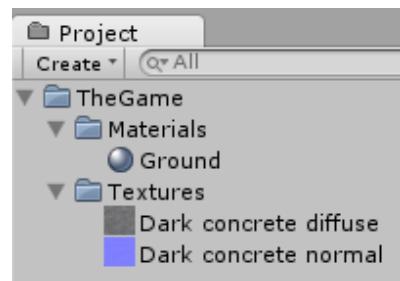
İstediğimiz şeye en hızlı şekilde erişmenin en iyi yolu Project panelini klasörler aracılığıyla düzenli tutmak.

Project panelindeki Create butonunu kullanarak “TheGame” adında bir klasör (folder) oluşturun. Klasörün içine “Textures” adında başka bir klasör oluşturup az önce import ettiğiniz kaplama asset'lerini bu klasöre taşıyın.

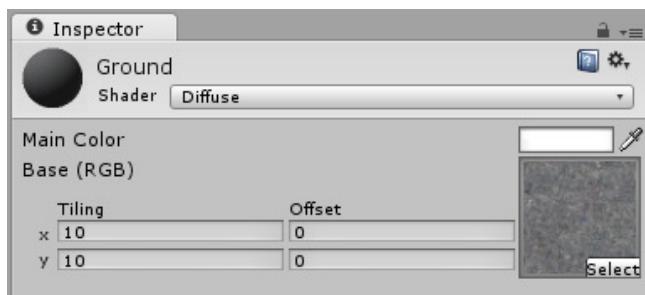


Zemine Kaplama Atamak

“TheGame” klasörünün içinde “Materials” adında başka bir klasör daha oluşturun. Bu klasörü seçip içinde yeni bir materyal (material) oluşturun, ismini "Ground" yapın.



Ground materyalini seçip Texture kısmına değer olarak “Dark concrete diffuse” texture'sini verin. Tiling değerinin X ve Y'sini 10 yapın.

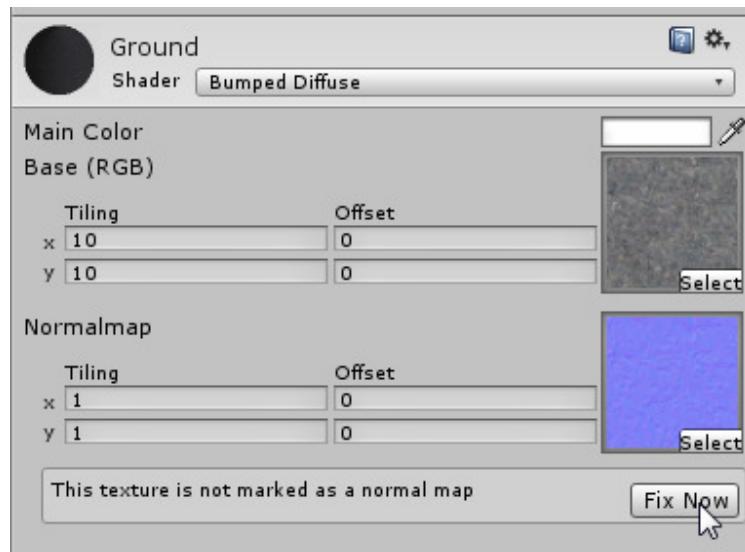


Şimdi bu materyali zemine atayalım. Bunun için zemini seçip Inspector'unda yer alan Mesh Renderer component'inin Materials kısmındaki Element 0'a değer olarak Ground materyalini verin.



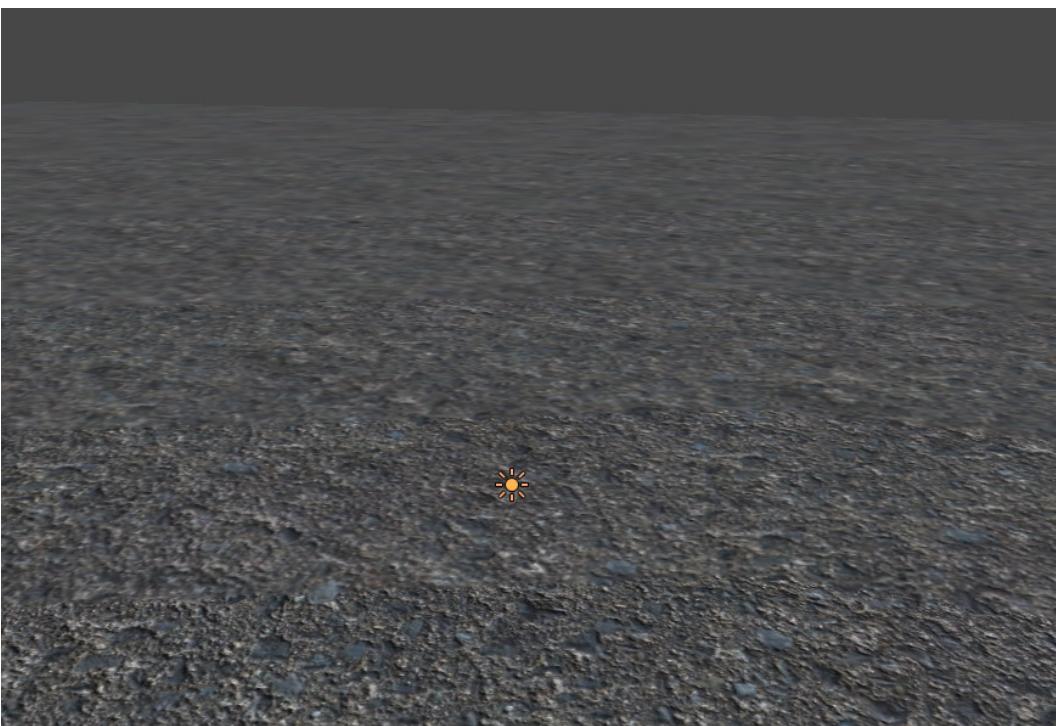
Şimdi materyalin kullandığı shaderı değiştirerek normal-map desteklemesini sağlayacağız. Ground materyalini seçip shader'ını "Bumped Diffuse" olarak değiştirin. "Dark concrete normal" texture'sini Normalmap kısmına değer olarak verin.

"This texture is not marked as a normal map" şeklinde bir uyarı verecek. Yanındaki "Fix Now"'a tıklayın.

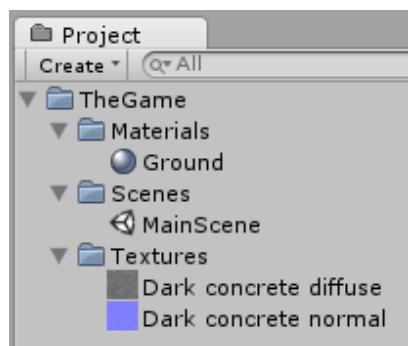


Şimdi Normalmap'in Tiling'ini de 10'a 10 yapın.

Sonuç olarak zemin girintili çıktınlı gibi, daha gerçekçi duracak:



Oyunu kaydetmek için şimdilik iyi bir zaman olabilir. Ctrl + S ile sahneyi kaydetmek istediğinizde nereye kaydetmeyi istediğiniz soracak. "TheGame" klasörünün içinde "Scenes" adında yeni bir klasör oluşturun ve bu klasörün içine sahneyi "MainScene" adıyla kaydedin.



Character Controller

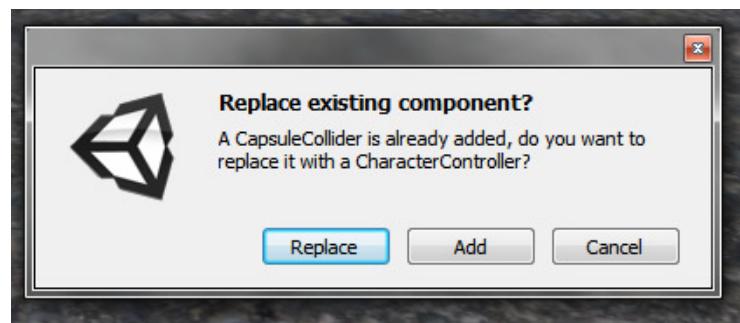
Unity'de bir karakteri hareket ettirmenin kolay yolu Character Controller componenti kullanmaktır. Bu component tek başına bir işe yaramaz ama ona komut veren bir script olunca işte o zaman akan sular durur.

Şimdilik karakterimiz için karmaşık bir 3D model değil basit bir kapsül kullanalım. Yeni bir kapsül oluşturup (`GameObject > Create Other > Capsule`) (0, 1.75, 0) pozisyonuna yerleştirin.

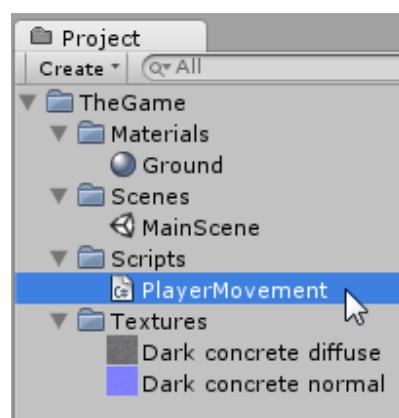
Kapsüle "Player" ismini verin.

Şimdi kapsüle `Component > Physics > Character Controller` ile Character Controller verin. Eğer bir uyarı penceresi çıkarsa "Replace"e basın.

ÇEVİRMEN EKLEMESİ: Eğer böyle bir uyarı penceresi bende çıkmadığı gibi sizde de çıkmazsa Player objesinin Inspector'undan Capsule Collider'in sağındaki dişli ikonuna tıklayın ve Remove Component seçeneğini seçin. Yoksa karakter hareket ederken garip bir şekilde uçmaya başlıyor.



Oyuncuyu hareket ettirmek için bir script yazmamız lazım. "TheGame"in içinde "Scripts" klasörü oluşturun. Bu klasörde "PlayerMovement" adında bir C# scripti oluşturun.



PlayerMovement scriptini Player objesine component olarak atayın. Sonra scripti açın.

Keyboard'dan Input Almak

Scripti şu şekilde güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerMovement : MonoBehaviour
05 {
06     public float h = 0;
07
08     // Use this for initialization
09     void Start()
10     {
11
12     }
13
14     // Update is called once per frame
15     void Update()
16     {
17         h = Input.GetAxis("Horizontal");
18     }
19 }
```

Değişkenimizin public olmasını sıkıntı etmeyin, o değişkeni geçici olarak kullanıyoruz. Değişkeni public yapmamızın sebebi değerini kolayca Inspector'dan takip edebilmek.

Oyunu çalıştırın. Klavyeden sol ya da sağ ok tuşuna basılı tutup Inspector'daki H'nin değerini kontrol edin. Değeri bastığınız tuşa göre 0.0'dan 1.0'a ya da -1.0'a kayacaktır. Dilerseniz ok tuşları yerine A ve D tuşlarını da kullanabilirsiniz.

Dışarıdan input almanın bir yolunu gördünüz. Input.GetAxis, hangi ok tuşuna bastığınıza göre -1.0'dan 1.0'a kadar bir değer döndürür. "Horizontal" (yatay) anahtar kelimesi kullandığımız için döndürülen değer sadece sağ ve sol ok tuşlarından (ve A-D tuşlarından) etkileniyor.

Character Controller'a Erişmek

Şimdilik Input kodunu silelim. Scriptin son halinin şöyle olduğundan emin olun:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerMovement : MonoBehaviour
05 {
06     CharacterController _controller;
07
08     // Use this for initialization
09     void Start()
10     {
11         _controller = GetComponent<CharacterController>();
12     }
13
14     // Update is called once per frame
15     void Update()
16     {
17     }
18 }
```

Karakteri hareket ettiren component Character Controller; yazmakta olduğumuz script ise Character Controller'a gerekli olan input'ları sağlıyor. Bu scriptten Character Controller'a erişmek için Character Controller component'ini _controller isimli bir değişkende tutuyoruz.

Değişkene değerini Start fonksiyonunda veriyoruz. Burası önemli. Yazdığınız class MonoBehaviour class'ını extend ediyorsa sakın ola yazdığınız class'ın constructor'ını kullanmayın / ellemeyin / kurcalamayın. Yoksa sebebini bir türlü anlamadığınız hatalarla karşılaşabilirsiniz.

GetComponent fonksiyonu, MonoBehaviour class'ı tarafından sunulmaktadır. Kendisi, bir component'e erişmek için kullanılır. Player objesine bir Character Controller componenti vermiştık; burada GetComponent ile de o component'e erişip onu değişkenimize atıyoruz.

Karakteri Hareket Ettirmek

Şimdi Input kodunu fonksiyona geri ekleyelim ve onu kullanarak karakteri hareket ettirelim:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerMovement : MonoBehaviour
05 {
06     CharacterController _controller;
07
08     // Use this for initialization
09     void Start()
10     {
11         _controller = GetComponent<CharacterController>();
12     }
13
14     // Update is called once per frame
15     void Update()
16     {
17         Vector3 direction = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
18         _controller.Move(direction);
19     }
20 }
```

Input.GetAxis'i kullanarak input alıyoruz. Parametre olarak "Horizontal" yazınca sağ-sol ok tuşlarının, "Vertical" yazınca ileri-geri ok tuşlarının döndürdüğü değeri alıyoruz. Ardından bu değerleri bir Vector3 değişkeninde depoluyoruz.

Vector3 yapısı (struct) x, y ve z değerlerine sahiptir. 3 boyutlu uzaydaki pek çok şeyi (pozisyon, eğim, boyut...) ifade etmek için Vector3 ideal yapı tipidir.

"Horizontal" input'u Vector3'ün X değerine, "Vertical" input'u da Vector3'ün Z değerine atıyoruz. Y değerine ise 0 veriyoruz çünkü Y ekseni dikey hareketi temsil eder ve karakterimiz sadece ileri-geri, sağa-sola hareket edecek. Henüz zıplama gibi bir durumumuz yok.

18. satırda Character Controller component'inin Move fonksiyonunu kullanıyoruz. Bu fonksiyon karakteri hareket ettirir. Vector3 değişkenimizi fonksiyona parametre olarak veriyoruz. Böylece karakteri sağa-sola ve ileri-geri hareket ettirebiliyoruz.

Şimdi oyunu çalıştırıp karakteri ok tuşlarıyla ya da WASD tuşlarıyla hareket ettirmenin keyfini yaşayabilirsiniz.

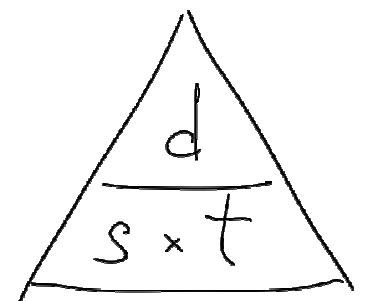
Saniye Cinsinden Hesap Yapmak

Önceki bölümde bahsettiğim gibi Update içinde sabit sayılarla işlem yapmak iyi değil. Çünkü oyunun hızına göre Update'in çalışma sıklığı değişir. O yüzden hesaplamızı bir saniyeyi baz alarak yapmalıyız:

```
18     _controller.Move(direction * Time.deltaTime);
```

Fizikten bildiğiniz üzere Hız (speed) = Mesafe (distance) / Zaman (time).

18. satırda hızı zaman ile çarpıyoruz ve haliyle karakterin gittiği mesafeyi ayarlamış oluyoruz. (Sonraki sayfaya bakınız...)



Görsel 4.1: Basit bir fizik kanunu. Lise derslerinizden hatırlarsınız belki.

ÇEVİRMEN EKLEMESİ: Time.deltaTime'in ne işe yaradığını size hatırlatmak istedim. Update fonksiyonunda Time.deltaTime kullanmadan bir işlem yaparsanız o işlemde adı geçen değerler, Update'in çağrılmış sıklığına göre güncellenir. Bu tavsiye edilmiyor, çünkü oyunun yavaş ve hızlı bilgisayarlarında birebir aynı şekilde kullanıcı deneyimi sunmasını engelliyor. Eğer ki Update fonksiyonunda işlem yaparken üzerinden geçtiğiniz değerleri Time.deltaTime ile çarparsanız o zaman artık yaptığınız işlem bir karede (frame) değil bir saniyede gerçekleşecek şekilde anlam kazanır, kullanıcı deneyimi bilgisayardan bilgisayara değişmez. İşte bu, tavsiye edilen yöntem.

Hızı Ayarlayabilmek

Şimdi sorunumuz karakterin çok yavaş hareket etmesi. Input.GetAxis, -1.0'dan 1.0'a bir değer döndürüyor. Yani objeyi saniyede 1 metre hareket ettirebiliyoruz. Bu değeri bir başka değerle çarparak hızı ayarlayabilmeliyiz. Kodu şu şekilde güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerMovement : MonoBehaviour
05 {
06     CharacterController _controller;
07
08     [SerializeField]
09     float _moveSpeed = 5.0f;
10
11     // Use this for initialization
12     void Start()
13     {
14         _controller = GetComponent<CharacterController>();
15     }
16
17     // Update is called once per frame
18     void Update()
19     {
20         Vector3 direction = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
21         Vector3 velocity = direction * _moveSpeed;
22         _controller.Move(velocity * Time.deltaTime);
23     }
24 }
```

Artık direction'ın x, y ve z değerlerini _moveSpeed ile çarpıyoruz. Bu sayede, eğer _moveSpeed 5 ise -1.0'dan 1.0'a olan değer yelpazesи -5.0'dan 5.0'a genişlemiş oluyor. Player'ı böylece saniyede 1 değil 5 metre ilerletebiliyoruz.

Karakteri Zıplatmak (Jump)

Space tuşuna basınca karakterin zıplamasını sağlayalım:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerMovement : MonoBehaviour
05 {
06     CharacterController _controller;
07
08     [SerializeField]
09     float _moveSpeed = 5.0f;
10
11    [SerializeField]
12    float _jumpSpeed = 20.0f;
13
14    [SerializeField]
15    float _gravity = 1.0f;
16
17    float _yVelocity = 0.0f;
18
19    // Use this for initialization
20    void Start()
21    {
22        _controller = GetComponent<CharacterController>();
23    }
24
25    // Update is called once per frame
26    void Update()
27    {
28        Vector3 direction = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
29        Vector3 velocity = direction * _moveSpeed;
30
31        if (_controller.isGrounded)
32        {
33            if (Input.GetButtonDown("Jump"))
34            {
35                _yVelocity = _jumpSpeed;
36            }
37        }
38        else
39        {
40            _yVelocity -= _gravity;
41        }
42
43        velocity.y = _yVelocity;
44
45        _controller.Move(velocity * Time.deltaTime);
46    }
47 }
```

Birkaç yeni değişken ekledik: “_jumpSpeed” ne kadar yükseğe zıplayabileceğinizi belirler. “_gravity” ise yerçekiminin gücünü belirler. “_yVelocity” değişkeni objenin dikey eksendeki (Y) hızını depolar ve farkettiğiniz üzere değeri Inspector'da gözükmez çünkü objenin mevcut karedeki (frame) dikey hızını Inspector'dan değiştirmek çok saçma olurdu.

31. satırda kullandığımız isGrounded fonksiyonu Character Controller'ın yere değip degmediğini döndürür. Eğer karakter yere deziyorsa ancak o zaman zıplama işlemini gerçekleştiriyoruz.

33. satırdaki Input.GetButtonDown fonksiyonu bir butona basılıp basılmadığını kontrol etmemize yarar. Parametre olarak “Jump” yazarsak spacebar tuşuna basılıp basılmadığına bakılır. Eğer basmışsa karakterin dikey hızını (_yVelocity) aniden zıplama gücüne eşitliyoruz, böylece karakter zıplıyor.

Eğer karakter havadaysa (isGrounded false döndürürse) onun Y eksenindeki hızını yerçekimi kadar azaltıyoruz. Böylece obje önce yavaşlayıp duruyor, sonra aşağı düşmeye başlıyor (Y hızı negatif olduğunda).

“_yVelocity” değişkeninin değerini 43. satırda velocity değişkeninin y parametresine veriyoruz. Hatırlayın: Move fonksiyonunda velocity değişkenini kullanıyoruz. Az önce velocity'nin y parametresinin değerini değiştirdiğimiz için de obje dikey eksende hareket edebiliyor.

Özet Geçecek Olursak...

Bu bölümde kullanıcıdan input almayı ve bu input'u kullanarak karakteri hareket ettirmeyi gördük. Ayrıca klasörler oluşturarak Project paneline biraz çeki düzen verdik.

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

Unity 3D İle Oyun Programlama

Bölüm 5: Kamera Scripti



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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

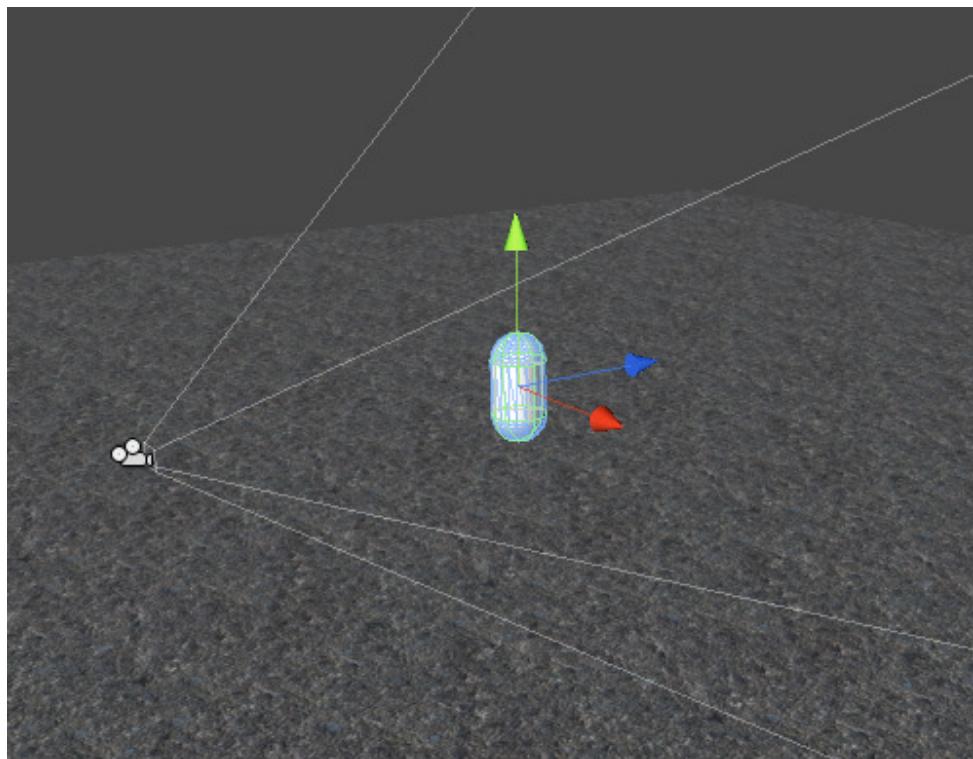
Karakteri hareket ettirebiliyoruz. Şimdi karakter ile etrafa bakabilemeye (kamerayı döndürmeyi) yapalım.

Burada kullanacağımız kamera açısı, TPS (third person shooter) oyunlarındaki gibi karakteri arkasından takip eden bir kamera açısı olacak. Kamerayı klavye ile değil fare ile kontrol edeceğiz.

Kamerayı Konumlandırmak

Kameranın oyuncuyu takip etmesi için onu karakterin bir child objesi yapmalısınız (Player objesi Main Camera'nın parent objesi olmalı). Main Camera'yı Hierarchy'den Player'ın üstüne sürüklemeniz yeterli.

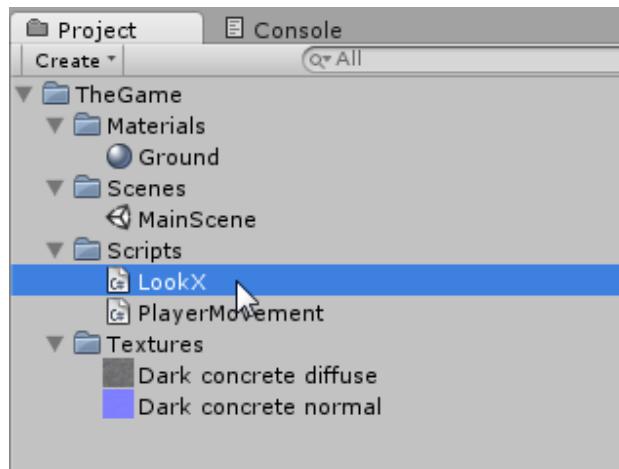
Şimdi kamera objesini karakterin arkasına doğru taşıyalım: kamerayı (0,0,-7) koordinatlarına taşıyın ve eğimini (0,0,0) yapın.



(Projeyi sık sık kaydetmeyi unutmayın.)

Sağ-Sola Bakmak

“Scripts” klasöründe “LookX” adında yeni bir C# scripti oluşturun.



Scriptin içeriğini şu şekilde değiştirin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class LookX : MonoBehaviour
05 {
06     [SerializeField]
07     float _mouseX = 0.0f;
08
09     void Update()
10     {
11         _mouseX = Input.GetAxis("Mouse X");
12     }
13 }
```

Ardından LookX scriptini Player objesine verin.

Oyunu çalıştırıp imleci sağa sola hareket ettirince Mouse X'in değerinin değiştiğini göreceksiniz. Bu sefer -1.0'dan 1.0'a bir aralık yok. O frame'de imleç yatay eksende (X eksen) ne kadar hareket ettiyse o kadar değeri oluyor. İmleci sola kaydırınca negatif, sağa kaydırınca pozitif değer alıyoruz.

Şimdi kodu şöyle değiştirin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class LookX : MonoBehaviour
05 {
06
07     float _mouseX = 0.0f;
08
09     void Update()
10     {
11         _mouseX = Input.GetAxis("Mouse X");
12
13         Vector3 rot = transform.localEulerAngles;
14         rot.y += _mouseX;
15         transform.localEulerAngles = rot;
16     }
17 }
```

Mouse X değişkeninin değerini Inspector'da görmek gereksiz birşey. Bu yüzden oradaki [SerializeField]'ı kaldırıldı.

Update fonksiyonunda da önce objenin Transform component'indeki rotation değerini alıp "rot" adında bir değişkene kaydettik (Transform component'ine erişmek için transform yazmak yeterli)(localEulerAngles Inspector'da objenin Rotation değerinde gördüğünüz Vector3'ü verir).

Fareyi sağa sola kımıldattıkça karakteri **Y ekseni (dikey eksen) etrafında** döndürüyoruz.

Oyunu çalıştırın ve fareyi oynattıkça karakterin döndüğüne şahit olun!

Mouse Hassaslığını (Sensitivity) Ayarlamak

Belki siz de benim gibi farenin hassaslığının az olduğunu düşünüyorsunuzdur. Ufak bir değişiklikle bu sorunu aşalım:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class LookX : MonoBehaviour
05 {
06     [SerializeField]
07     float _sensitivity = 5.0f;
08
09     float _mouseX = 0.0f;
10
11     void Update()
12     {
13         _mouseX = Input.GetAxis("Mouse X");
14
15         Vector3 rot = transform.localEulerAngles;
16         rot.y += _mouseX * _sensitivity;
17         transform.localEulerAngles = rot;
18     }
19 }
```

Yaptığımız tek şey, değerini Inspector'dan belirleyebildiğimiz Sensitivity ile Mouse X'i çarparak hassaslığı ayarlamak.

Kamerayı döndürürken Time.deltaTime ile çarpmadığımıza dikkat edin. Oyun ne kadar yavaşlsa da fare yavaşlamayacaktır. Belki imlecin konumunun güncellenmesi yavaşlayacaktır ama bizim işimiz imlecin konumuyla alakalı değil.

Yukarı-Aşağı Bakmak

Scripts klasöründe "LookY" diye bir C# scripti oluşturun. Karaktere verip içini güncelleyin:

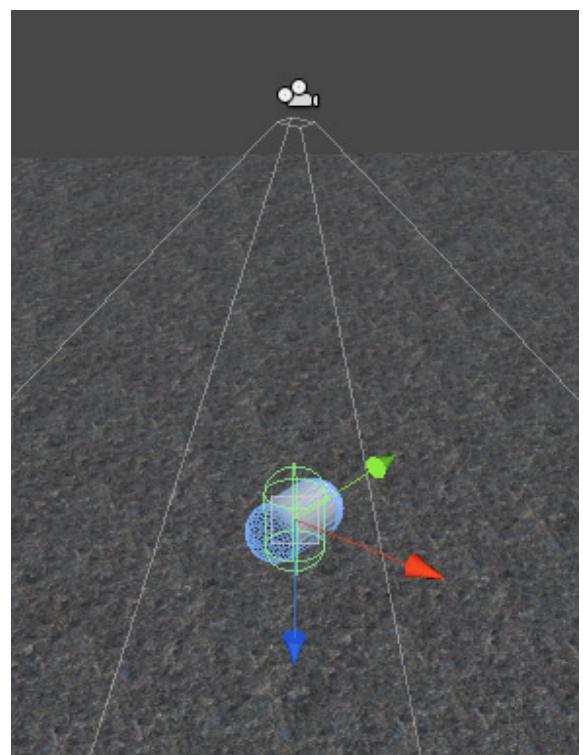
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class LookY : MonoBehaviour
05 {
06     [SerializeField]
07     float _sensitivity = 5.0f;
08
09     float _mouseY = 0.0f;
10
11     void Update()
12     {
13         _mouseY = -Input.GetAxis("Mouse Y");
14
15         Vector3 rot = transform.localEulerAngles;
16         rot.x += _mouseY * _sensitivity;
17         transform.localEulerAngles = rot;
18     }
19 }
```

Burada _mouseY değişkenine değer verirken GetAxis fonksiyonunun döndürdüğü değeri -1 ile çarpıyoruz. Çünkü imleci yukarı oynattıkça GetAxis("Mouse Y") pozitif değer döndürür. Ama biz kamerayı yukarı oynatmak için fareyi aşağı hareket ettirmeliyiz. Döndürulen değeri -1 ile çarptığımız için artık istediğimiz şey gerçekleşiyor.

Fareyi yukarı-aşağı hareket ettirince resimdeki görüntüyle karşılaşacaksınız. Karakter objesi fareyi hareket ettirdikçe devriliyor-düzeliyor.

Objenin x eksenindeki eğimini değiştirdiğimiz için obje devriliyor. Scripti hangi objeye verirsek o obje fareyi yukarı-aşağı oynattıkça devrilecek-düzelecek. Ama biz sadece kameranın yukarı-aşağı oynamasını istiyoruz.

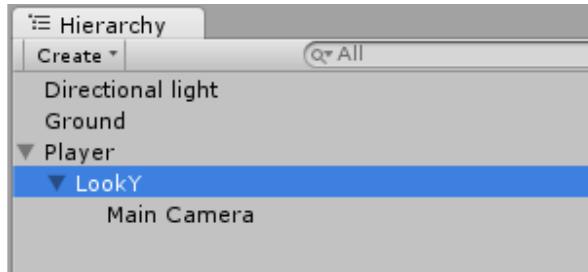
Scripti niye kameraya vermiyoruz diyebilirsiniz. Eğer kameraya verirsek fareyi yukarı-aşağı oynattıkça artık kamera karaktere odaklanmaktan çıkar. Bunu anlamanın en basit yolu Player'dan Mouse Y component'ini Remove Component ile silmek ve onun yerine Main Camera'ya vermek.



Eğer çalışıyorsa oyunu durdurun. Player'da ya da kamerada LookY scripti varsa Remove Component ile objeden atın.

"LookY" adında yeni bir boş obje oluşturun. Player objesinin bir child objesi yapın. Ardından Transform component'inin sağındaki dişli simgesine tıklayıp "Reset"e basın.

Şimdi de Main Camera objesini LookY objesinin child objesi yapın:



Görsel 5.1: Player objesinin hiyerarşisi bu şekilde olmalı.

Şimdi LookY scriptini LookY objesine verip oyunu çalıştırın. Sistem kusursuz çalışmalı.

Scripti boş objeye verdigimiz için aslında kamera ile birlikte boş obje de devriliyor-düzeliyor ama adı üstünde "boş obje" olduğu için devrilen birşey görmüyoruz biz. Player ise artık devrilmiyor çünkü parent objeler (Player) child objeleri (LookY) etkileseler bile child objeler (LookY) parent objeleri (Player) etkilemez.

Karakterin Doğru Yände Hareket Etmesini Sağlamak

Kamerayı döndürüp karakteri hareket ettirirseniz yanlış yönde gittiğini göreceksiniz. Bu, global uzay (world space) ve yerel uzay (local space) ile alakalı bir sorun.

Bölüm 1'de bahsetmiştim; global uzayda X, Y ve Z eksenlerinin yönü daima sabittir. Her objenin bir de yerel uzayı vardır ve obje döndükçe yerel uzayındaki X, Y, Z eksenlerinin yönü de değişir. Yerel Z ekseni objenin yüzü nereye bakıyorsa hep oraya bakar.

Global uzayda bir referans noktası yoktur ama yerel uzayda bir referans noktasına ihtiyacınız vardır. Örneğin "evim marketin 3 blok yanında" ibaresi bir yerel uzay örneğidir. Buradaki referans noktamız "market" ve "3 blok" da uzaklıktır.

Problemimize dönelim. Sorunun kaynağı CharacterController'un Move fonksiyonunun global uzayda bir vektör istemesidir. Objemize göre ileri olan ekseni yerel uzaydan global uzaya çevirmeli ve Move fonksiyonuna parametre olarak bunu sunmalıyız.

Neyse ki bir objenin yerel uzayındaki bir vektörü global uzaydaki bir vektöre çevirmeye yarayan hazır bir fonksiyon bulunur. PlayerMovement scriptine şu eklemeyi yapın:

```

25 // Update is called once per frame
26 void Update()
27 {
28     Vector3 direction = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
29     Vector3 velocity = direction * _moveSpeed;
30
31     if (_controller.isGrounded)
32     {
33         if (Input.GetButtonDown("Jump"))
34         {
35             _yVelocity = _jumpSpeed;
36         }
37     }
38     else
39     {
40         _yVelocity -= _gravity;
41     }
42
43     velocity.y = _yVelocity;
44
45     velocity = transform.TransformDirection(velocity);
46
47     _controller.Move(velocity * Time.deltaTime);
48 }
49 }
```

TransformDirection, Transform class'ındaki bir fonksiyondur. Kendisi bir Vector3 alır ve hangi transform tarafından çağrılmışsa Vector3'ü o transform'un yerel uzayındaymış gibi farzeder. Ardından bu Vector3'ün global uzaydaki Vector3 karşılığını döndürür (return).

TransformDirection'ın döndürdüğü değeri geri velocity değişkenine veriyoruz. Artık velocity global uzaydaki bir vektörü depoluyor. Ardından her zamanki gibi Move fonksiyonuna parametre olarak velocity'i veriyoruz.

Bilgilendirme

Bazı durumlarda yerel uzayda, bazı durumlarda ise global uzayda işlem yapmak daha mantıklıdır.

Örneğin "Kendime göre ileri yönde saniyede 5 metre gitmek istiyorum" demek "Saniyede X ekseninde 3.535 metre, Z ekseninde -3.535 metre gitmek istiyorum." demeye göre daha basit ve mantıklıdır. İlk örnek tahmin edebileceğiniz üzere yerel uzaya, ikinci örnek global uzaya örnekti.

Tam tersi şekilde, "Giriş (3.4, 5.9, 12.0) koordinatlarında" demek "Giriş tabelaya göre (-1.3, 0, 0,), tabela da çıkış kapısına göre (45.23, 0, 3.2) koordinatlarında" demmeye göre daha basittir. İlk örnek global uzayı, ikinci örnek yerel uzayı temsil etmektedir.

Gördüğünüz gibi bazen global uzayda çalışmak bazen de yerel uzayda çalışmak daha mantıklıdır.

Nişangahı (Crosshair) Oluşturmak

Nişangah oluşturarak grafik arayüzüne (GUI) giriş yapacağız. Simdilik ekranın ortasında bir resim çizdireceğiz.

“PlayerGui” adında yeni bir C# scripti oluşturun:

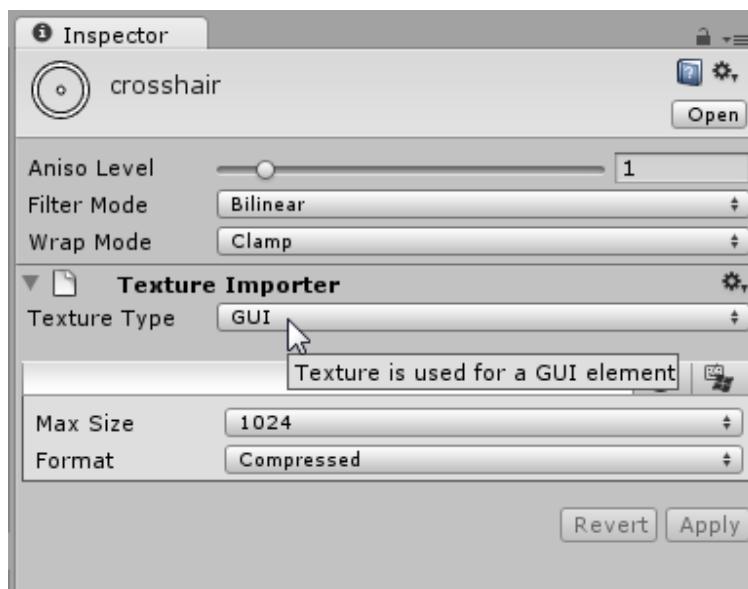
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _crosshair;
08
09     void OnGUI()
10     {
11         GUI.DrawTexture(new Rect(0, 0, 30, 30), _crosshair);
12     }
13 }
```

Texture2D, bir resim dosyasını barındırmaya yarayan bir veri türüdür. Bu veriyi OnGUI fonksiyonunda kullanarak ekrana resmi çizdiriyoruz. Eğer script kullanacaksanız GUI elemanları her zaman için OnGUI fonksiyonunda çizdirilir. Koda alternatif olarak GUI Texture componenti kullanılabilir.

GUI.DrawTexture ekranda dikdörtgen şeklinde bir alanı (Rect) parametre olarak alır ve resmi bu alanda çizdirir. Rect'in ilk iki parametresi dikdörtgenin sol üst noktasının koordinatını (X ve Y) belirlerken son iki parametresi dikdörgenin genişliğini (width) ve yüksekliğini (height) belirler.

Winrar arşivini çıkardığınız yerdeki "Images" klasöründe yer alan "crosshair" resmini projenize import edip "Textures" klasörüne taşıyın.

Project panelinden crosshair'ı seçin. Inspector panelindeki Texture Type seçeneğini "GUI" (yeni sürümlerde "GUI (Editor \ Legacy)" olarak geçmektedir) olarak değiştirin ve "Apply" butonuna tıklayın.



Şimdi PlayerGui scriptini Player objesine verin. Player Gui component'ının Crosshair kısmına ilgili resim dosyasını verin.

Oyunu çalıştırınca ekranın sol üstünde nişangahı göreceksiniz.

Öncelikle nişangahın çizildiği dikdörtgensel alanın genişlik ve yüksekliğinin 30 pixel yerine "crosshair" resim dosyasının genişlik (width) ve yüksekliğiyle (height) aynı olmasını sağlayalım:

```
11     GUI.DrawTexture(new Rect(0, 0, _crosshair.width, _crosshair.height), _crosshair);
```

Sonrasında basit bir işlem ile nişangahı ekranın ortasına taşıyalım:

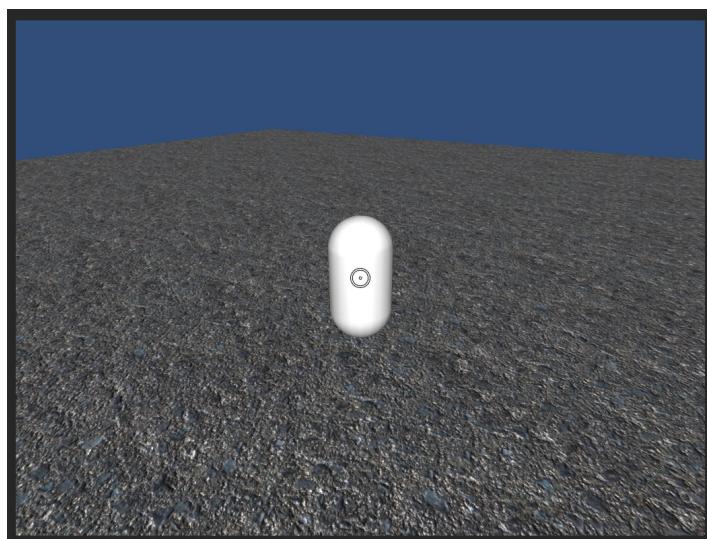
```
11     float x = (Screen.width - _crosshair.width) / 2;
12     GUI.DrawTexture(new Rect(x, 0, _crosshair.width, _crosshair.height), _crosshair);
```

Screen.width, ekranın pixel cinsinden genişliğini verir. Aynı işlemi Y koordinatına da uygulayalım:

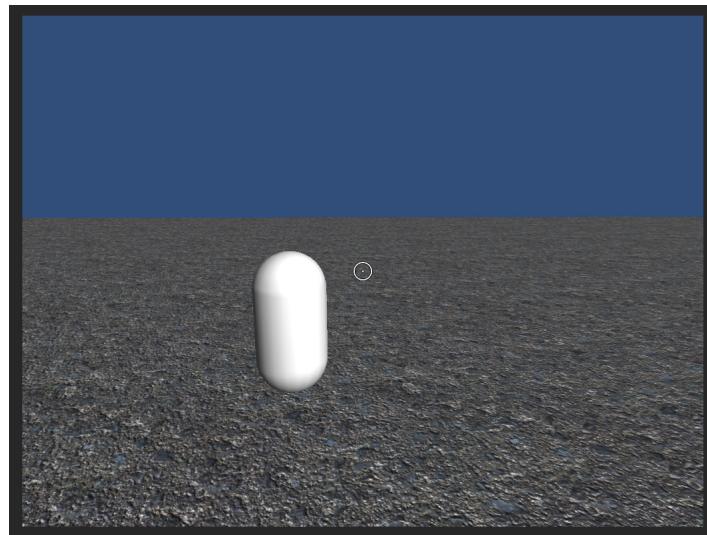
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _crosshair;
08
09     void OnGUI()
10     {
11         float x = (Screen.width - _crosshair.width) / 2;
12         float y = (Screen.height - _crosshair.height) / 2;
13         GUI.DrawTexture(new Rect(x, y, _crosshair.width, _crosshair.height), _crosshair);
14     }
15 }
```

ÇEVİRMEN EKLEMESİ: Yaptığımız "basit işlem"i açıklayayım. Rect'in X ve Y koordinatının dikdörtgenin "sol üst" noktasının koordinatını verdiginden bahsetmiştim. (Screen.width / 2) komutu ekranın genişliğinin yarısını verir. Eğer Rect'e X koordinatı olarak bu değeri verseydik nişangahın sol noktası ekranın ortasına denk gelirdi ve nişangah tam ortalanmamış olurdu. Ama eğer X koordinatını crosshair'ın genişliğinin yarısı kadar daha sola taşırsak (_crosshair.width / 2) crosshair X ekseninde mükemmel bir şekilde ortalanmış olur.

Artık crosshair ekranda ortalanmış vaziyette. Ama bu sefer de crosshair Player'ın üzerinde kalıyor.



Player'ın "LookY" isimli child objesini seçin ve Transform component'inden pozisyonunu (1, 0.5, 1) olarak değiştirin. Artık kamera oyuncuya "omuz hizasından" bakacak ve nişangahın önü serbest kalacak.



Özet Geçecek Olursak...

Artık kamerayı fare ile hareket ettirebiliyoruz. Bu bölümde grafik arayüzüne de basit bir giriş yaptık. Tebrikler, çok iyi gidiyorsunuz!

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

Unity 3D İle Oyun Programlama

Bölüm 6: Düşman Yapay Zekasına Giriş



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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Bu bölümde düşman yapay zekası ile uğraşacağız. Simdilik yapay zekanın yaptığı tek şey bizi kovalamak olacak.

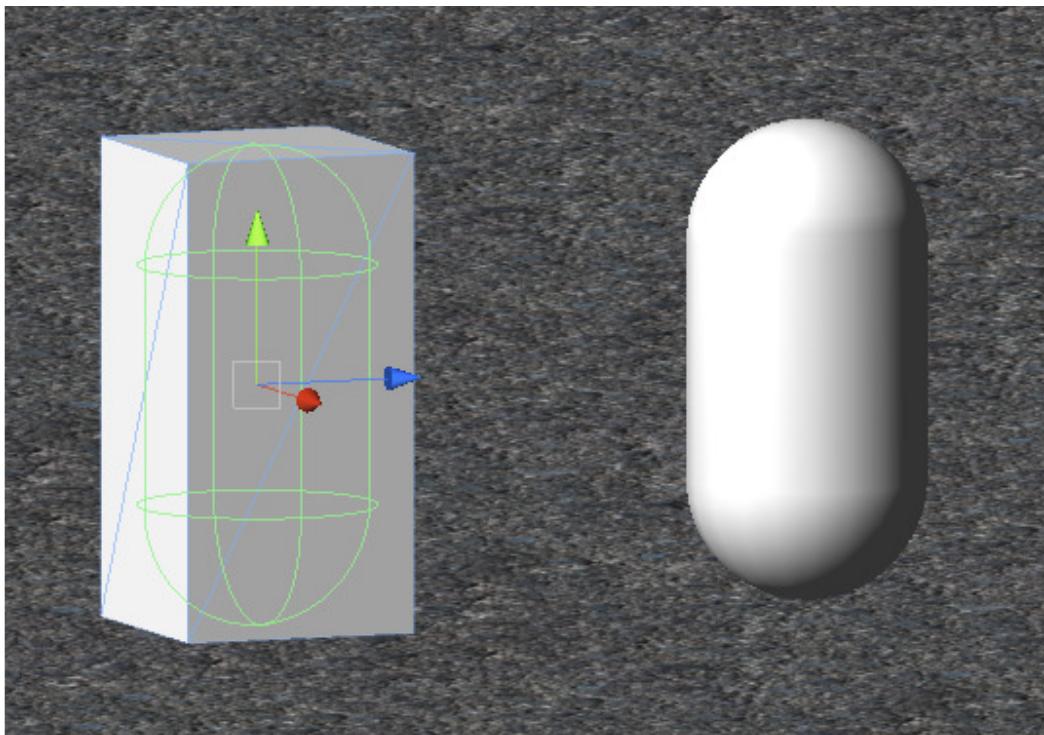
Düşman Oluşturmak

Şu an için düşman yerine temsili küp objeleri kullanacağız.

GameObject > Create Other > Cube ile küp oluşturup (1, 2, 1) şeklinde boyutlandırın (scale).

Player'da olduğu gibi, düşmanın hareket etmesini de Character Controller ile sağlayacağız. O yüzden küp objesine bir Character Controller verin (Component > Physics > Character Controller).

Şimdi objeye "Enemy" (düşman) adını verin. (**ÇEVİRMEN EKLEMESİ**: Tıpkı Player'da yaptığımız gibi, **Enemy** objesinin **Box Collider'ını** da **Remove Component** ile silin. Zira Character Controller'in kendi görünmez collider'i var.)



Görsel 6.1: Düşman objesinin ebatları Player ile aşağı yukarı aynı.

Scripti Yazmaya Başlamak

“EnemyMovement” diye bir C# scripti oluşturun.

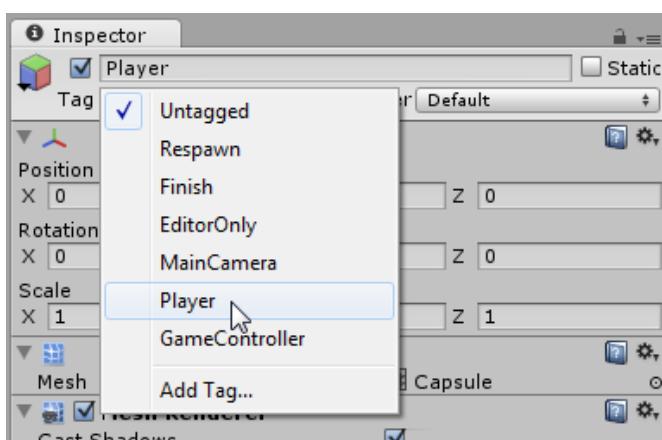
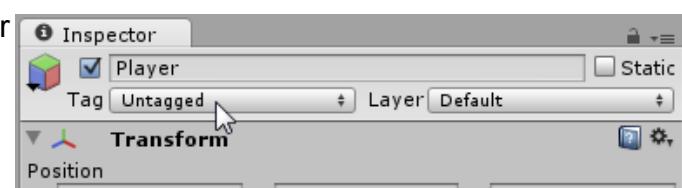
Düşman bizi takip edeceği için haliyle nerede olduğumuzu bilmelidir. Pozisyonumuzu Player'ın Transform componenti depoluyor. Bu component'e düşman objelerinden sık sık erişeceğimiz için bu componenti tutan bir değişken oluşturalım:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyMovement : MonoBehaviour
05 {
06     Transform _player;
07
08     void Start()
09     {
10     }
11
12     void Update()
13     {
14     }
15 }
```

“_player” değişkeni Player objesinin Transform component'ini içinde tutacak. Bu değişkene değer vermek için kullanabileceğimiz çeşitli yollar var. Biz “tag” (etiket) sistemini kullanacağız.

Tag Kullanımı

Hierarchy'den Player'ı seçin. Yukarıda Tag adında bir değer yer almaktır. Varsayılan değeri “Untagged” olarak verilmiş.



“Untagged”e tıklayınca karşınıza seçenekler gelecek. Bunlar arasından “Player”ı seçin. Artık objemizin tag'ı “Player” oldu.

EnemyMovement script'ini şu şekilde güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyMovement : MonoBehaviour
05 {
06     Transform _player;
07
08     void Start()
09     {
10         GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
11         _player = playerGameObject.transform;
12     }
13
14     void Update()
15     {
16     }
17 }
```

Player objesini bizim için Unity'nin bulmasını istiyoruz. FindGameObjectWithTag fonksiyonu parametre olarak bir string alır ve bu tag'a sahip objeyi döndürür. Parametre olarak tahmin edeceğiniz üzere "Player" verdik. Artık bu fonksiyon Player objesini bulup playerGameObject değişkenine atacak.

Sonraki satırda playerGameObject'in, yani Player objesinin Transform component'ine erişip onu _player değişkenine veriyoruz.

Character Controller'a Erişmek

Nasıl Player objemizin CharacterController component'ine script ile erişiyorsak düşman için de ayını yapacağız. Bunun için 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     void Start()
10     {
11         GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
12         _player = playerGameObject.transform;
13
14         _controller = GetComponent<CharacterController>();
15     }
16
17     void Update()
18     {
19     }
20 }
```

Oyuncuyu Kovalamak

Düşmanı hareket ettirmek için de Move fonksiyonunu kullanacağız. Ama bu sefer input'u klavyeden almayacağız. Bir formül kullanarak zombinin gideceği yönü kendimiz hesaplayacağız:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyMovement : MonoBehaviour
05 {
06     CharacterController _controller;
07     Transform _player;
08
09     void Start()
10    {
11        GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
12        _player = playerGameObject.transform;
13
14        _controller = GetComponent<CharacterController>();
15    }
16
17     void Update()
18    {
19        Vector3 direction = _player.position - transform.position;
20
21        _controller.Move(direction * Time.deltaTime);
22    }
23 }
```

Öncelikle scripti test edelim. Scripti düşman objesine verin ve oyunu çalıştırın. Zombinin sizi kovalaması lazım.

Hemen hemen tüm işi 19. satırda yapıyoruz. Player ve Enemy arasındaki yön vektörünü buluyoruz. Bunu yapmanın yolu da Player'ın pozisyonundan Enemy'nin pozisyonunu çıkarmak. Böylece bu iki obje arasındaki yön vektörünü elde ediyoruz ve vektörün ucu Player'a doğru bakıyor. Eğer Enemy'nin pozisyonundan Player'ının pozisyonunu çıkarsaydık okun ucu Enemy'e bakacaktı:

$$\Delta x = x_2 - x_1$$

Bir başka deyişle:

$$\text{YÖN} = \text{HEDEF KONUM} - \text{KENDİ KONUMUM}$$

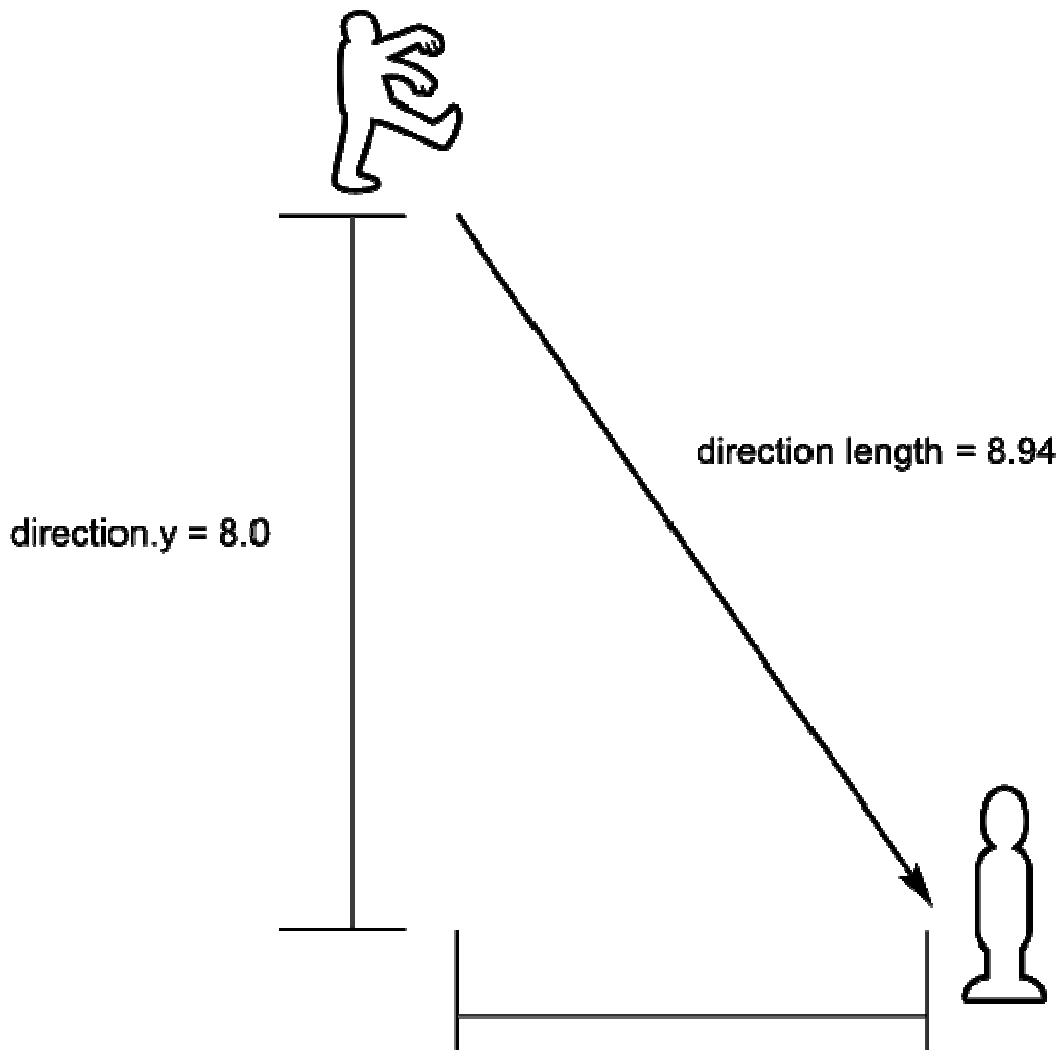
Bu formül sayesinde Player'ı kovalamak için gerekli vektörü elde ediyoruz.

19. satırda "KENDİ KONUMUM" olarak transform.position'i, "HEDEF KONUM" olarak da _player.position'i kullanıyoruz. Bu iki değişken birer Vector3 (yani x, y ve z değerlerine sahip). Bu yüzden elde ettiğimiz veri de Vector3 türünde oluyor.

Vektörü Normalize Etmek

Düşman oyuncuya yaklaşıkça yavaşlıyor. Çünkü düşman ile oyuncu arasındaki yön vektörünün boyu (length) gitgide kısalıyor.

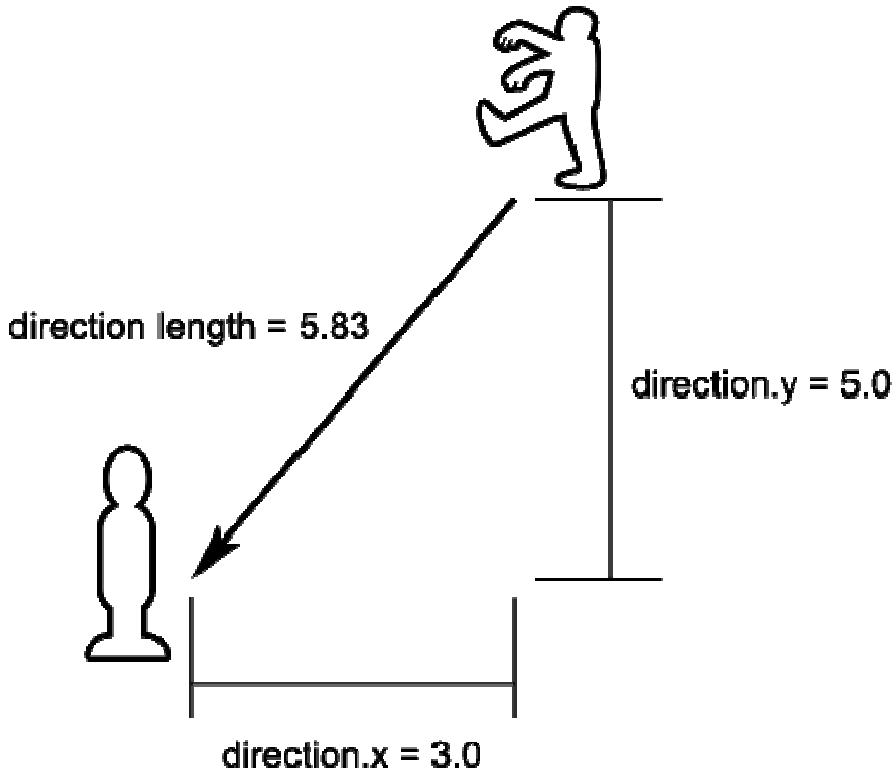
Şu örneği inceleyelim:



Yön vektörünün (direction) görselleştirilmesi

Bu örnekte düşman ile oyuncu arasındaki yön vektörünün uzunluğu (length) 8.94 birim.

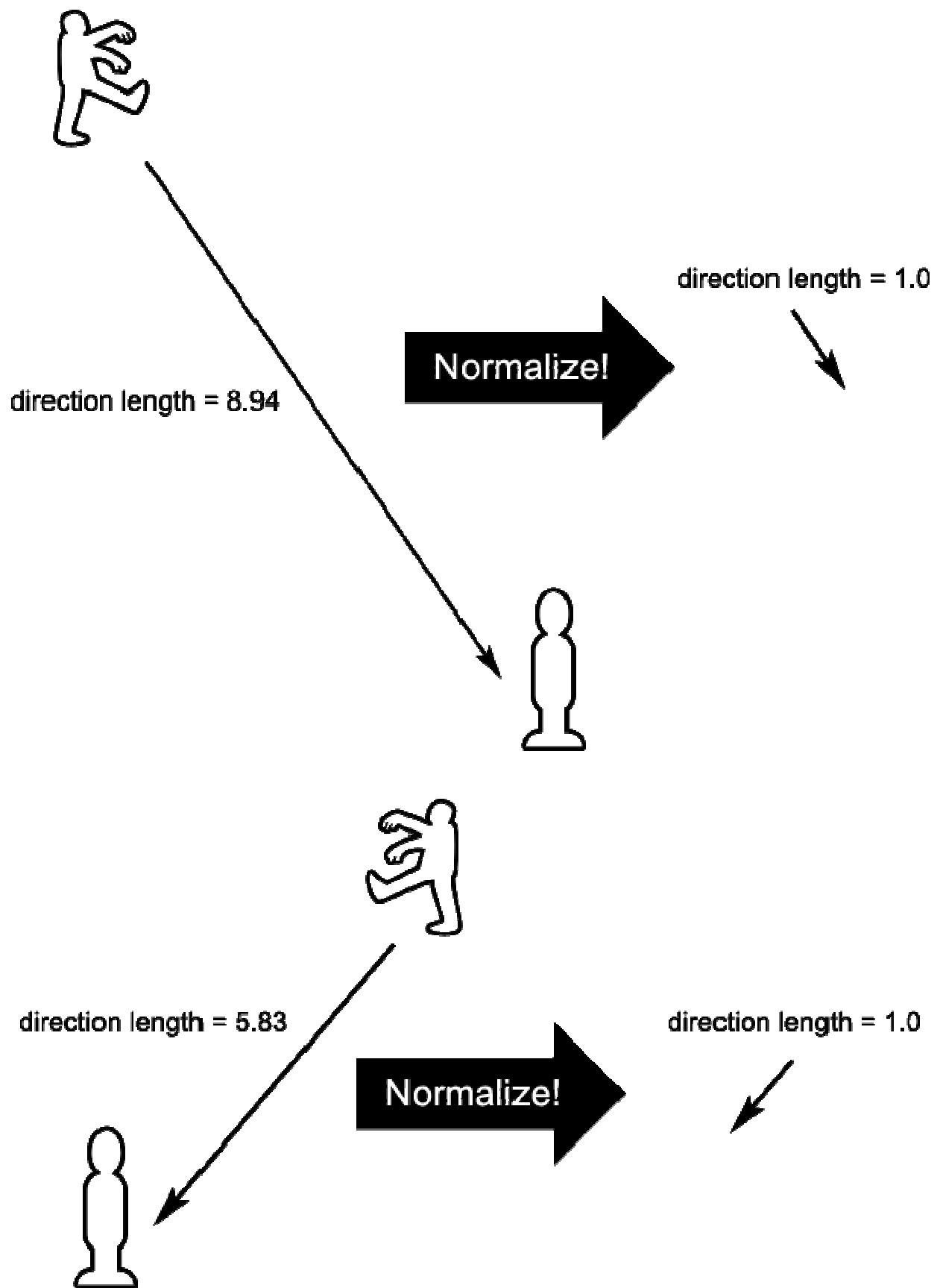
Şimdi sıradaki örneğe bakalım. Düşmanın bize daha yakın olduğu bir senaryo bu. Yön vektörü daha da kısaltılmış durumda. Hatırlarsanız Move fonksiyonuna parametre olarak bu yön vektörünü verdik. Yani yön vektörü kısaltıkça zombi daha yavaş hareket etmeye başlıyor.



Biz böyle olsun istemiyoruz. Zombilerimiz hep aynı hızda hareket etsinler. Bunun için de bir şekilde yön vektörünün uzunluğunu belli bir değere sabitlemeliyiz. Bu işleme Normalize deniyor.

Bir vektörü Normalize etmek onun uzunluğunu 1.0 yapar. Ama vektörün yönü aynı kalır!

Üstteki iki örnekteki yön vektörlerini normalize edersek ne olur birlikte görelim:



Gördüğünüz gibi Normalize edilen vektörün uzunluğu 1.0 oluyor ama yönü aynı kalıyor.

Unity'de Normalize için hazır bir fonksiyon mevcut. Biz de bu fonksiyondan faydalanalalım:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyMovement : MonoBehaviour
05 {
06     CharacterController _controller;
07     Transform _player;
08
09     void Start()
10     {
11         GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
12         _player = playerGameObject.transform;
13
14         _controller = GetComponent<CharacterController>();
15     }
16
17     void Update()
18     {
19         Vector3 direction = _player.position - transform.position;
20         direction.Normalize();
21
22         _controller.Move(direction * Time.deltaTime);
23     }
24 }
```

20. satırda Vector3 class'ının Normalize fonksiyonunu kullanıyoruz. Böylece vektörümüz Normalize ediliyor. Normalize etmenin bir başka yolu da şu:

```
20     direction = direction.normalized;
```

"normalized" değişkeni ise vektörün kendisini değiştirmez ama Normalize edilmiş halini döndürür. Biz de döndürülen bu değeri istediğimiz değişkene atabiliriz (üstteki kodda yine direction vektörüne atıyoruz).

Oyunu çalıştırınca zombinin yavaşladığını ama hızının sabitlendiğini göreceksiniz. Yavaşlamanın sebebi artık yön vektörünün uzunluğunun hep 1.0 olması. Yani saniyede 1 metre hareket ediyoruz. Bir hız değişkeni ekleyerek düşmanı hızlandırıralım derim ben.

Düşmana Hız Vermek

EnemyMovement'ı düzenleyelim:

```
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     void Start()
13     {
14         GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
15         _player = playerGameObject.transform;
16
17         _controller = GetComponent<CharacterController>();
18     }
19
20     void Update()
21     {
22         Vector3 direction = _player.position - transform.position;
23         direction.Normalize();
24
25         Vector3 velocity = direction * _moveSpeed;
26
27         _controller.Move(velocity * Time.deltaTime);
28     }
29 }
```

Player scriptinde yaptığımız işlemin aynısını yaptığımız için açıklama yapmama gerek yok. Düşmana çok hız vermeyin ki kaçma imkanınız olsun!

Özet Geçeceğ Olursak...

Bu bölümde düşmanı oluşturup onu hareket ettirmekle kalmadık; aynı zamanda vektör matematiğine de basit bir giriş yaptık. Ayrıca tag (etiket) sistemi ile uğraştık.

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

Unity 3D İle Oyun Programlama

Bölüm 7: Prefab Sistemi



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



All code snippets are licensed under CC0 (public domain)



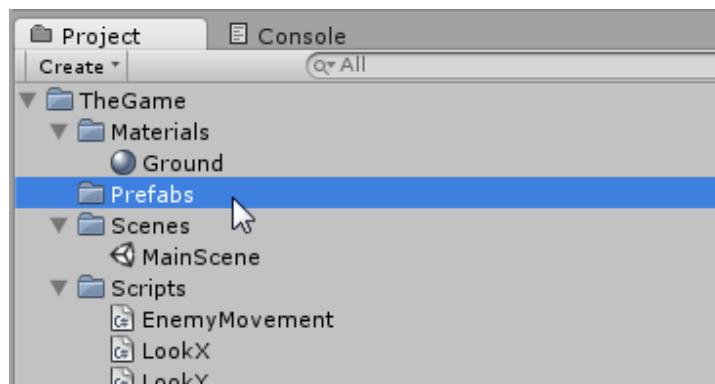
This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Madem geçen bölümde düşman objesi oluşturduk, bu bölümde de bu düşmanı çoğaltarak oyunun zorluğunu artıralım.

Unity'de bir objeyi taslaç olarak kullanıp daha sonra bu taslağı kullanarak o objeden istediğimiz kadar çoğaltabiliriz. Bu sisteme "prefab" adı veriliyor.

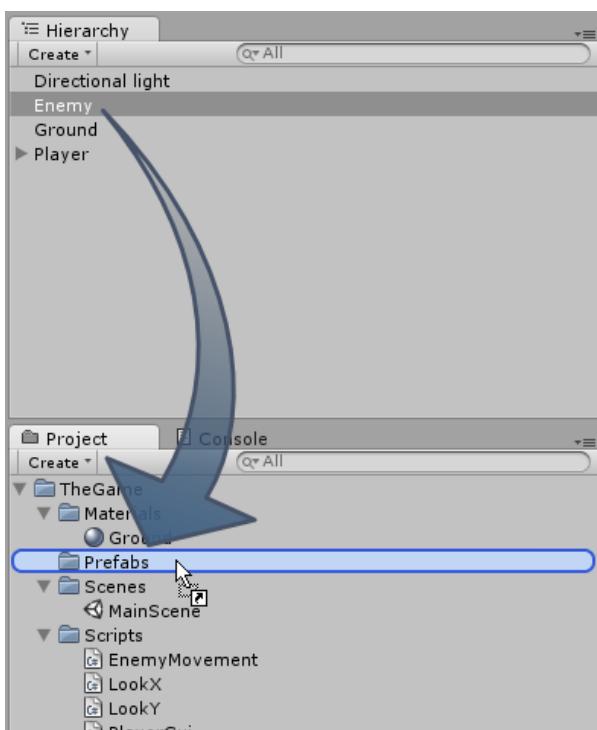
Prefab'lar İçin Klasör Oluşturmak

Prefab'lar da birer asset'tir ve Project panelinde görünürler. Oyunda kullanacağımız tüm prefab'ları depolamak üzere yeni bir klasör oluşturun. Klasöre "Prefabs" adını verin:

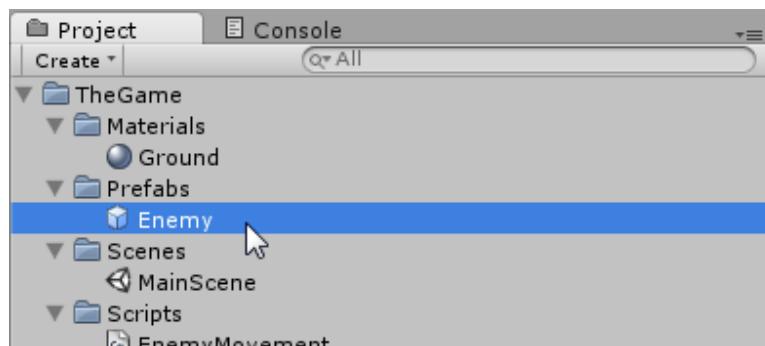


Yeni Bir Prefab Oluşturmak

Unity 3.4 versiyonundan itibaren prefab oluşturmak kolaylaştı. Hierarchy panelinden "Enemy" objesini tutup Project panelindeki "Prefabs" klasörünün içine sürükleyin. İşte bu kadar!



Prefabs klasöründe Enemy prefab'ı bir asset şeklinde belirecek. Bu asset'in bir prefab olduğunu ikonuna bakarak anlayabilirsiniz: mavi bir küp.



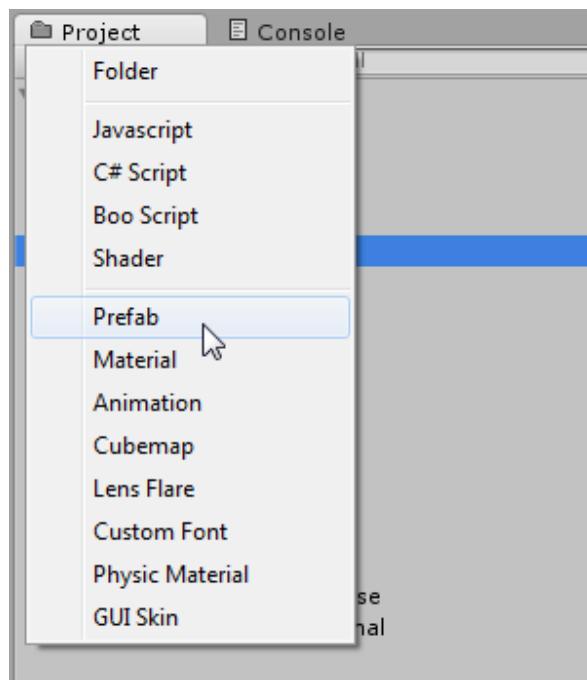
Alternatif Yol

Unity'nin 3.4 sürümünden önceki bir sürümü kullanıyorsanız alternatif yolu kullanmalısınız.

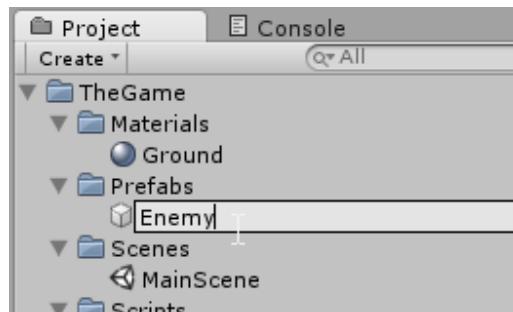
Eğer önceki metodu kullanarak prefab oluşturduysanız bu aşamayı atlayabilirsiniz.

Project panelinden Prefabs klasörünü seçin.

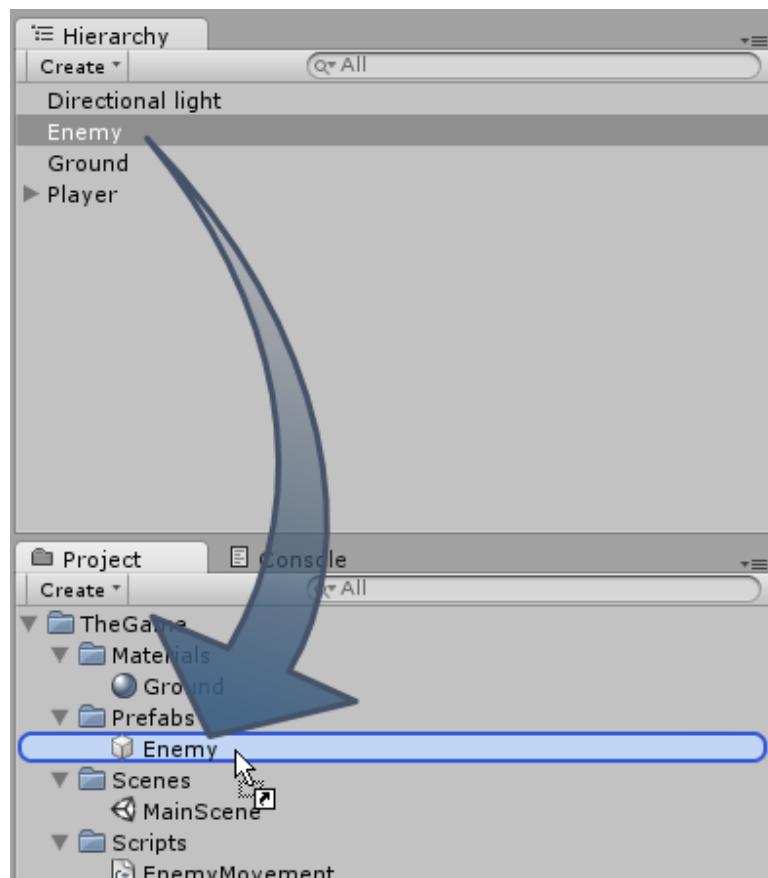
“Create” butonuna basın ve Prefab'ı seçin.



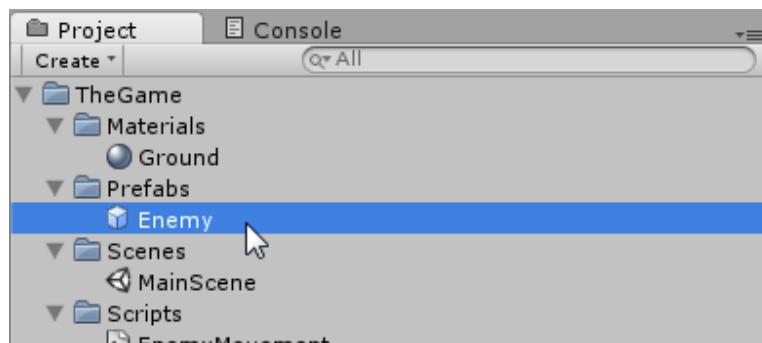
Boş bir prefab olacak. İsmini "Enemy" yapın. Prefab'ın ikonu gri bir küp. Bunun anlamı prefab'in içinden henüz boş olduğunu göstermektedir.



Şimdi "Enemy" objesini Hierarchy panelinden tutarak Project panelindeki "Enemy" prefab'ının üzerine sürükleyin.

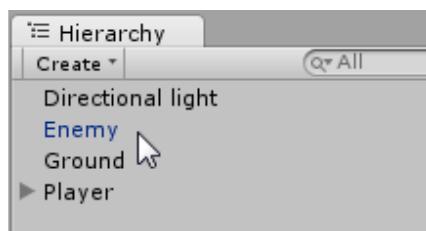


Bu aşamadan sonra Enemy prefab'ının ikonu mavi küp olacak ve böylece içinin dolu olduğunu bize bildirecek.



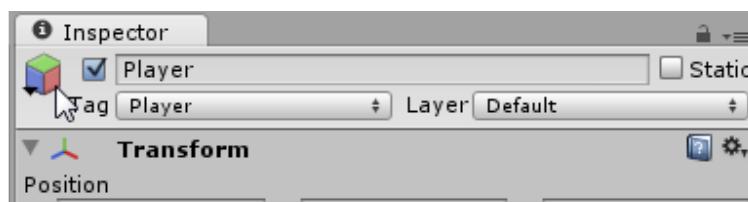
Prefab'ları Ayırt Etmek

Hierarchy'deki Enemy objesinin ismine bakacaksınız mavi renkte olduğunu göreceksiniz. Bunun anlamı Enemy objesinin bir prefab'a bağlı olduğunudır.

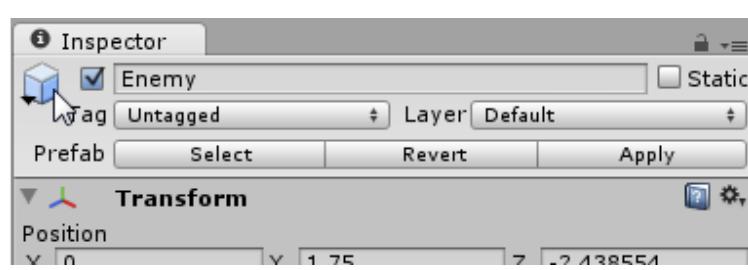


Görsel 7.1: Prefab'dan türemiş objelerin isimleri mavi renkte görünür.

"Prefab"ı klonlanmaya hazır bir obje olarak düşünebilirsiniz. Kendisi sahnenizde yer almaz, sadece Project panelinde bir asset olarak yer alır. Oluşturduğu klon objeler ise sahnede (scene) yer alır.



Görsel 7.3: Normal bir objenin ikonunun kırmızı, yeşil ve mavi renklerden oluşan bir küp olduğunu dikkat edin.

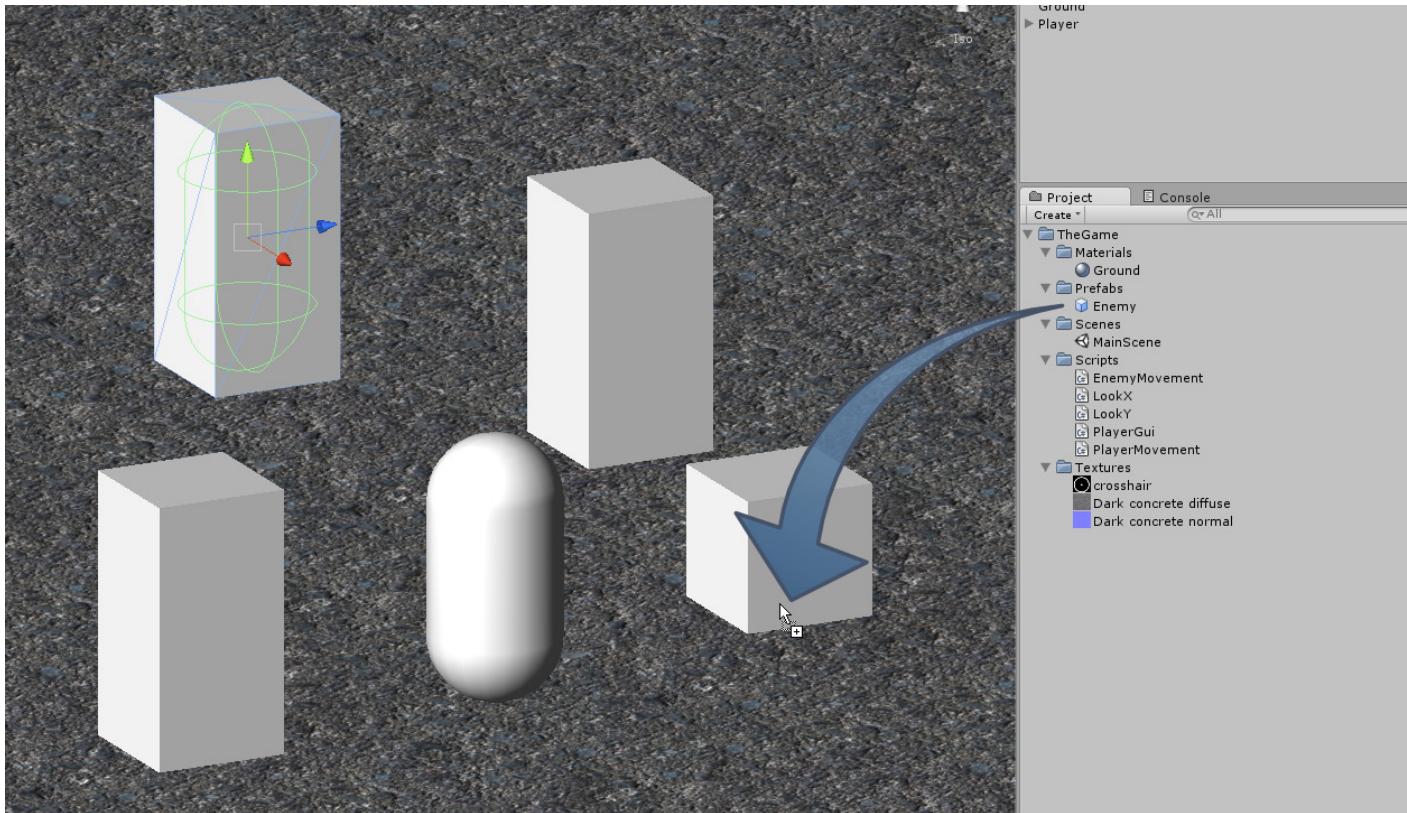


Görsel 7.2: Bir prefab ile bağlantılı olan bir objenin ikonunun ise mavi bir küp olduğunu dikkat edin.

Prefablar da birer game object olduğu için onlara component verebilir/silebilir, component'lerinde yer alan değerleri değiştirebilirsiniz.

Prefab'ı Kullanarak Klon (Instance) Oluşturmak

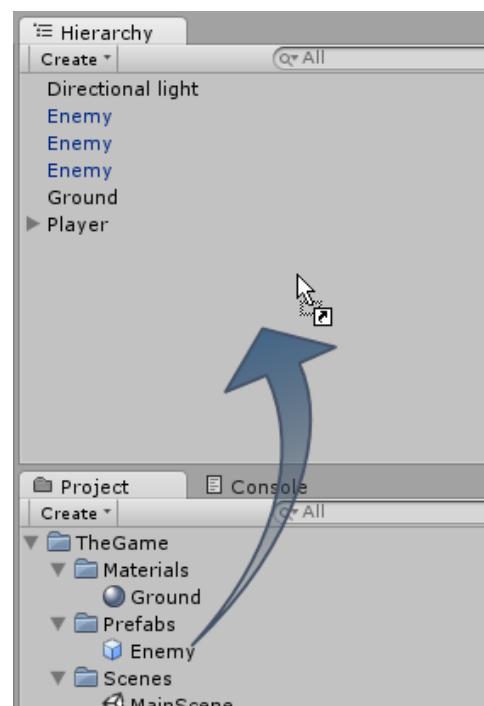
Bu işlemi yapmak sandığınızdan da kolay. Enemy prefab'ını Project panelinden tutup Scene panelinde istediğiniz yere sürükleyin. İşte bu kadar!



Alternatif olarak prefab'ı Scene paneline değil Hierarchy paneline sürükleyebilirsiniz.

Durmayın ve sahneye en azından iki düşman objesi daha ekleyin.

ÇEVİRMEN EKLEMESİ: Prefab klonu oluşturmanın bir başka yolu daha var: sahnedeki bir düşman objesini seçin ve **CTRL + D** kombinasyonu ile onu klonlayın. Yeni oluşan klon da prefab ile bağlantılı olacak.



Prefab Üzerinde Değişiklik Yapmak

Eğer ki birden çok düşman objesi istiyorsak neden hiçbir prefab oluşturmadan direkt CTRL+D ile düşmanı klonlamadık? Prefab kullanınca ne değişti?

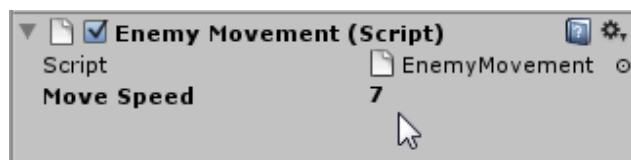
Prefab sisteminin çok önemli bir farkı var ve biz de bu yüzden prefab kullandık: Project panelindeki prefab üzerinde değişiklik yaparsanız yaptığınız değişiklikler sahnedeki prefab klonlarına aynen yansıyor. Yani diyelim düşmanın boyunu kısaltmak istediniz. Sahnedeki tüm düşman objelerinin Scale değeriyle oynamak yerine Project panelindeki prefab'in Scale değeriyle oynamak yeterli olacaktır.

Prefab'ın bir başka faydası da bir objenin oyun sırasında (runtime) script vasıtasiyla klonlanmasına yardımcı olmasıdır. Bu fonksiyon ile daha sonra uğraşacağız.

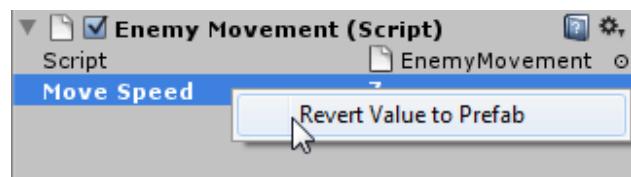
Şimdi prefab üzerinde değişiklik yapalım. Project panelinden Enemy prefab'ını seçin. Inspector paneli normal bir objede olduğu gibi prefab'ın sahip olduğu component'leri listeleyecek.

"Move Speed" değerini 5.25 yapın. Eğer sahnedeki herhangi bir Enemy objesini seçeceğ olursanız onun da Move Speed'inin otomatik olarak 5.25 olduğunu göreceksiniz. Harika!

Peki ya sadece bir düşman klonunun farklı bir hızda sahip olmasını isteseydik ne olacaktı? Her bir klonun Inspector panelindeki değerleri yine ayrı ayrı değiştirebilirsiniz. Bir değeri değiştirdiğinizde o değer kalın puentoyla vurgulanır. Bunun anlamı o değerin prefab'taki değerden farklı olduğunu gösterir.

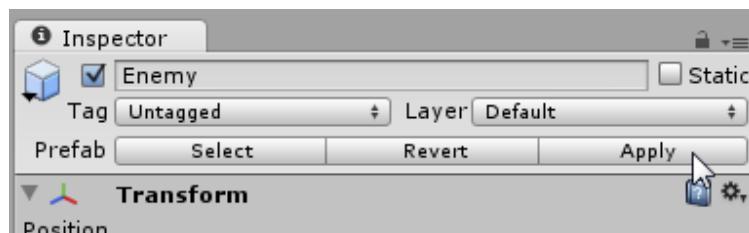


Eğer bir klonun bir değerini prefab'daki varsayılan değerle geri döndürmek isterseniz o değerle sağ tıklayıp "Revert Value to Prefab"ı seçmeniz yeterli.



Bir Klondaki Oynanmış Değeri Prefab'a Uygulamak

Eğer bir klondaki değerlerin prefab'taki varsayılan değerlerden daha iyi olduğunu düşünüyor ve tüm prefab klonlarının bu klondaki değerlere sahip olmasını istiyorsanız yapmanız gereken tek şey ilgili klonu seçip Inspector'un tepesindeki Apply butonuna tıklamak.

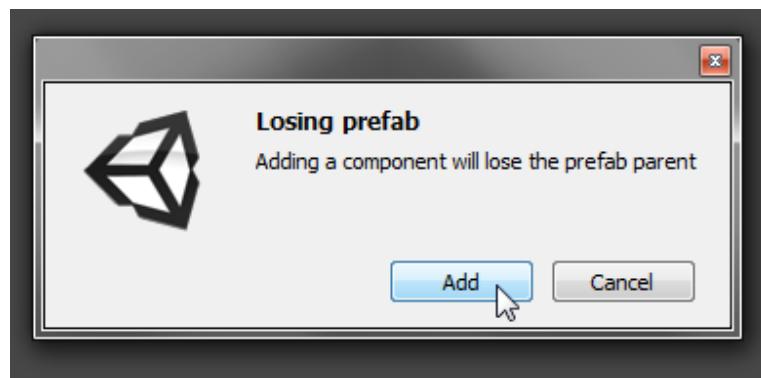


Burada iki buton daha var:

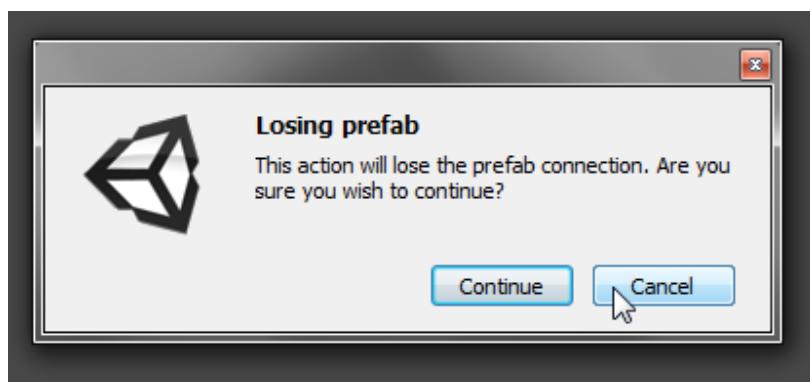
- “Select” klonun çıktıgı prefab objesini Project panelinde göstermeye yarar.
- “Revert” klondaki tüm değerleri prefab'daki varsayılan değerlere geri döndürmeye yarar.

Prefab'a Component Ekleme/Çıkarma

Eğer prefab'a yeni bir component verecek olursanız bir uyarı penceresi gelir (**ÇEVİRMEN EKLEMESİ**: Yeni Unity sürümlerinde bu uyarı(lar) gelmiyor):



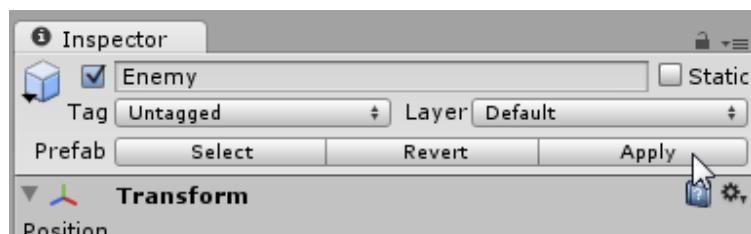
Ayrıca prefab'tan bir component silecek olursanız da benzer bir uyarı geliyor:



Eğer amacınız tüm prefab klonlarına o component'ten vermek/çıkarmak idi ise endişelenmeyin. İstediğiniz şeyi elde etmenin yolu buradan geçiyor.

Örneğin sahneden bir düşman objesi seçin. **Component > Audio > Audio Source** ile Audio Source ekleyin. Eğer uyarı mesajı gelirse Add butonuna tıklayın.

Klona Audio Source eklediğiniz için artık klon prefab ile bağlantılı değil (**ÇEVİRMEN EKLEMESİ**: Unity'nin yeni sürümlerinde bu işlemler prefab ile klonun bağının kopmasına sebep oluyor.). Şimdi Inspector'dan "Apply" butonuna tıklayın:



Artık prefab ve tüm klonları birer Audio Source component'ine sahip olacak.

Özet Geçecek Olursak...

Bu derste sadece prefablar üzerine odaklandık. Artık nasıl prefab oluşturup onu ve klonlarını eşzamanlı olarak düzenleyebileceğinizi biliyorsunuz.

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

Unity 3D İle Oyun Programlama

Bölüm 8: Düşmana Hasar Vermek



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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Artık bizi kovalayan zombiler var. Bu bölümde karakterin zombilere ateş etmesini sağlayacağız.

Can Eklemek

Düşmanlara hasar vermeden önce temel bir konuyu halletmeliyiz: onlara birer sağlık değeri vermeliyiz ki hasar uygulayabilelim. Sağlık bir tamsayı (integer) olacak ve örneğin ilk değeri 100 olacak. Düşman hasar alındığça bu değer azalacak. Eğer sağlık 50'ye inerse anlayacağımız ki düşman sağlığınıın yarısını kaybetmiş. Sağlık sıfıra inince ise düşman ölmüş olacak.

Sağlık (Health) için yeni bir script yazacağız. "Health" adında yeni bir C# scripti oluşturun:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     void Start()
12     {
13         _currentHealth = _maximumHealth;
14     }
15 }
```

"_maximumHealth" değişkeni maksimum sağlık değerini belirliyor. Sağlığı en başta bu değere eşitliyoruz. Örneğin boss vari düşmanlar için bu değişkenin değerini 100 değil de daha yüksek bir değer yapabilir ve düşmanın daha zor ölmesini sağlayabiliriz.

"_currentHealth" ise mevcut sağlığın değerinin depolandığı değişkenimiz.

Yeni bir fonksiyon yazalım:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     void Start()
12     {
13         _currentHealth = _maximumHealth;
14     }
15
16     public void Damage(int damageValue)
17     {
18         _currentHealth -= damageValue;
19     }
20 }
```

Damage (HasarVer) fonksiyonu sağlığı azaltmaya yarıyor. Düşmana hasar verirken bu fonksiyonu kullanacağınız.

Şimdi düşmanlara Health scriptini verelim.

Project panelinden Enemy prefabını seçin. Component > Scripts > Health yolunu izleyerek Health scriptini prefaba verin. Artık Health scripti tüm Enemy klonlarına otomatik olarak eklendi.

Raycast Sistemini Kullanarak Ateş Etmek

Şimdi sıra düşmana hasar vermeye yarayan scripti yazmaya geldi! Basit bir silah scripti yazacağınız. "RifleWeapon" adında yeni bir C# scripti oluşturun.

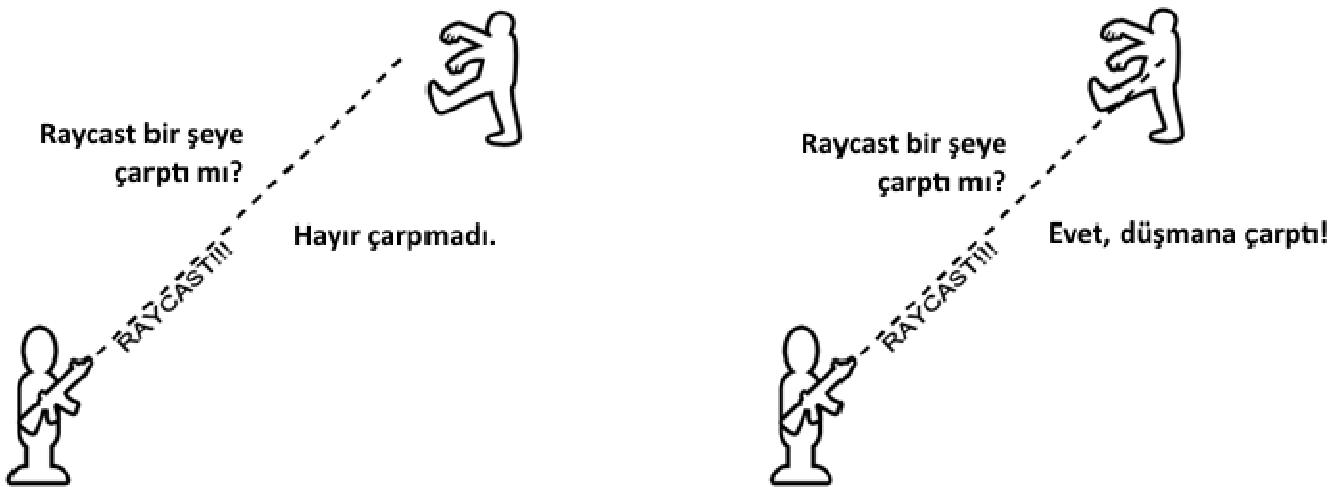
NOT: Scriptleri her zaman için "Scripts" klasöründe oluşturun ki Project paneli düzenli kalsın.

Health scriptindeki Damage fonksiyonunu çağırın script işte bu RifleWeapon scripti olacak.

Ateş ederken düşmanı vurup vurmadığımızı anlamak için "raycasting" denen bir teknolojiyi kullanacağınız.

Raycasting sistemi, 3 boyutlu uzayda görünmez bir çizgi oluşturup bu çizginin bir objeyle temas edip etmediğine bakmaya yarar.

Bizim çizgimiz karakterin üzerinden başlayacak ve ileri yönde olacak. Bu çizginin bir düşmana çarpıp çarpmadığına bakacağız. Eğer çarpmışsa o düşmanın Health scriptindeki Damage fonksiyonunu çağıracağız.



Aslına bakacak olursanız 3D uzayda kurşun objeleri oluşturmayacağız. Raycast ile zaten o kurşunun nereye isabet edeceğini aşağı-yukarı hesaplıyoruz. Raycast işlemi anlık gerçekleşir, yani eğer hedefinizde bir düşman varsa düşman anında hasar alır. Video oyun terminolojisinde buna "*hitscan*" denir (bkz. <http://en.wikipedia.org/wiki/Hitscan>)

Adım adım ilerleyelim. Önce RifleWeapon'u şu şekilde güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class RifleWeapon : MonoBehaviour
05 {
06     void Update()
07     {
08         Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
09         RaycastHit hitInfo;
10
11         if (Physics.Raycast(mouseRay, out hitInfo))
12         {
13             Debug.Log(hitInfo.transform.name);
14         }
15     }
16 }
```

Bu kod, ekranın ortasından dümdüz ileri doğru bir raycast oluşturur ve eğer raycast işini bir objeye temas ederse onun ismini konsola yazdırır.

8. satırda Ray (ışın) türünde bir değişken oluşturuyoruz. Bu işinin başlangıç noktasını ekranın tam ortası (kameranın verdiği görüntünün tam ortası) olarak ayarlıyoruz. ViewportPointToRay fonksiyonu sadece bir Vector3 parametre alır. Bu parametrenin X değeri 0 olursa ışın ekranın solundan, 1 olursa sağдан çıkar. Y değeri 0 olursa ışın ekranın alt kenarından, 1 olursa üst kenarından çıkar. Eğer iki değer de 0.5 olursa ışın ekranın ortasından çıkar. Vector3'ün Z değeri ise işinin başlangıç noktasının kameradan kaç birim uzakta olacağını belirler. 0 verdigimiz için ışın tam kameranın olduğu noktadan çıkar. -1 yazsaydık işinin başlangıç noktası kameranın 1 metre berisi, +1 yazsaydık da 1 metre ilerisi olurdu.

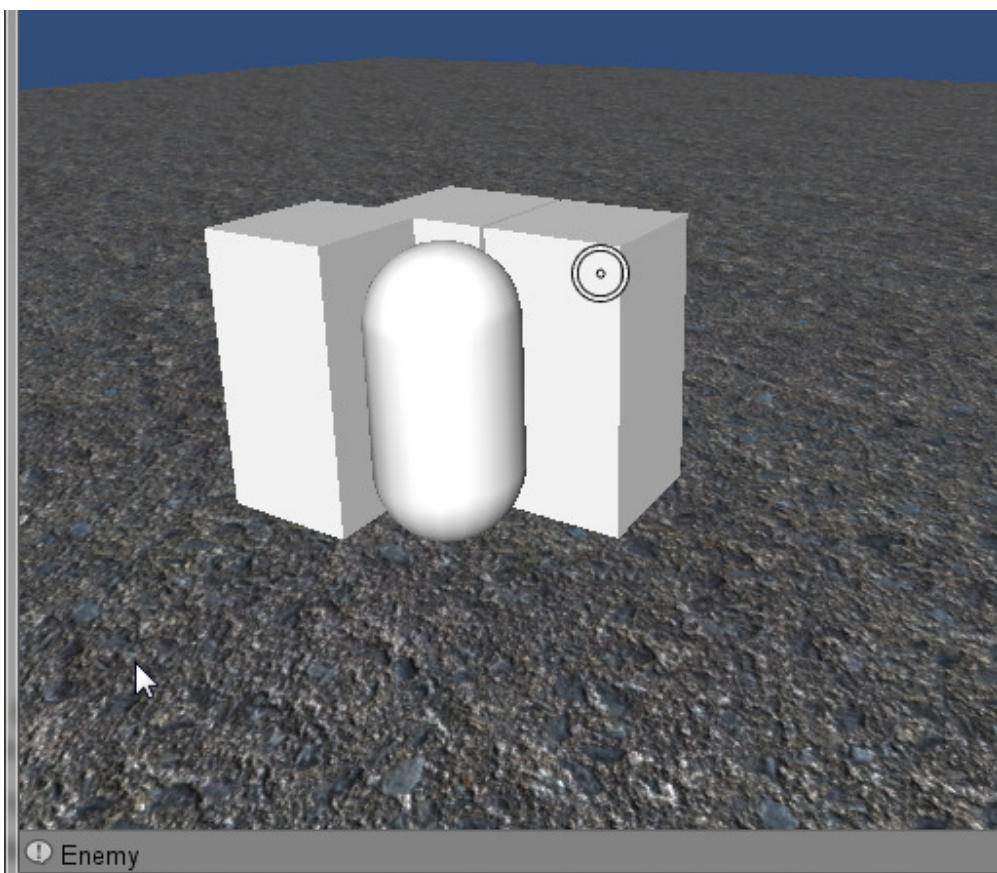
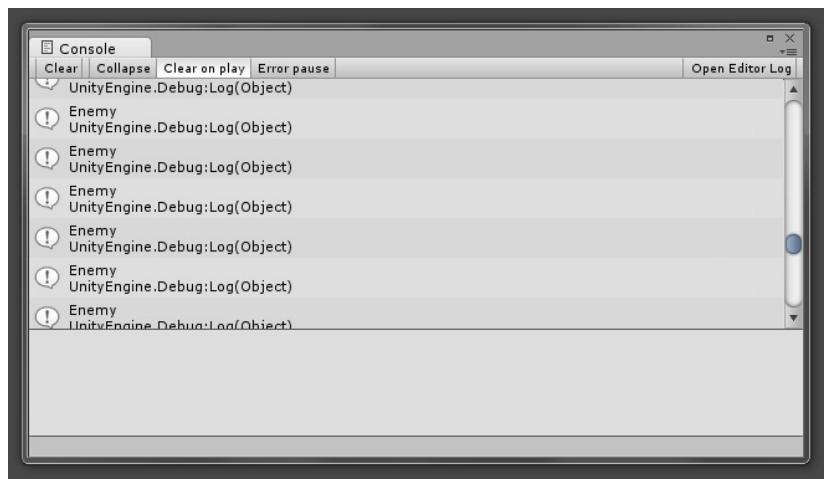
Raycast işlemi sonucu işnimiz bir objeyle temas ederse o objenin bilgilerini depolamak üzere bir değişken kullanılır ve bu değişkenin türü RaycastHit'tır. Kodun 9. satırında bu türde geçici bir değişken oluşturuyoruz.

11. satır asıl ışın yapıldığı satır. Burada Raycast fonksiyonunu çağırıyoruz. Eğer işnimiz bir objeye temas ederse Raycast fonksiyonu true döndürür yoksa false döndürür. Yani Raycast fonksiyonunu bir if'in içine koyarsak o if koşulu ancak ışın bir objeye temas edince gerçekleşir. İkinci parametrenin başında "out" diye bir anahtar kelime var. Bu kelime C#'ta kullanılır ve girilen parametrenin değerini fonksiyonun değiştirebileceğini belirler. Daha fazla bilgi için bkz. <http://msdn.microsoft.com/en-us/library/ee332485.aspx>

Eğer raycast işini bir objeye temas ederse 13. satırdaki kod çalışıyor ve Debug.Log fonksiyonu ile konsola, işinin temas ettiği objenin ismi yazılıyor. Debug.Log fonksiyonu scriptlerinizde hata ayıklama yaparken çok kullanışlıdır. Burada kullandığımız hitInfo.transform komutu temas edilen objenin Transform component'ine erişmeye ve hitInfo.transform.name ise bu objenin ismine erişmeye yarar.

Konsola verilen output'ları görmek için **Window > Console** yaparak konsol penceresini açın.

Şimdi scripti Player objesine verin ve oyunu çalıştırın. Konsoldaki mesajları dikkatlice kontrol edin. İmlecinizin ucunda hangi obje varsa onun ismi konsolda yazmalı:



Görsel 8.1: En son gelen konsol output'unu ayrıca Unity'nin sol altındaki ufak bölmeye de görebilirsiniz.

Raycast kodunu Update'in içine yazdığımız için her karede (frame) raycast işlemi yapılıyor. Bu hem gereksizdir hem de sistemi boş yere yormak demektir. Raycast işlemini sadece sol mouse tuşuna basınca, yani ateş edince yapalım. Bunun için scripti şu şekilde güncelleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class RifleWeapon : MonoBehaviour
05 {
06     void Update()
07     {
08         if (Input.GetButtonDown("Fire1"))
09         {
10             Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
11             RaycastHit hitInfo;
12
13             if (Physics.Raycast(mouseRay, out hitInfo))
14             {
15                 Debug.Log(hitInfo.transform.name);
16             }
17         }
18     }
19 }

```

Bir if koşulu oluşturup içinde GetButtonDown fonksiyonunu kullandık. GetButtonDown fonksiyonuna parametre olarak "Fire1" verirseniz sol mouse tuşuna tıklayınca fonksiyon true, öbür durumlarda false döndürür. **Böylece raycast işleminin sadece ateş edince gerçekleşmesini sağladık.**

ÇEVİRMEN EKLEMESİ: GetButtonDown("Fire1") komutu sadece **sol mouse tuşuna bastığınız ilk frame'de** true döndürür. Yani diyelim ki sol mouse tuşuna basılı tutuyorsunuz. GetButtonDown sadece tuşa basmaya başladığınız anda true döndürür. Ardından basılı tutsanız bile false döndürür. Ta ki tuştan elinizi çekip tekrar basana kadar.

Hasar Vermek

Raycast yapabildiğimize göre düşmana hasar verme kısmını artık halletmeye hazırız:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class RifleWeapon : MonoBehaviour
05 {
06     void Update()
07     {
08         if (Input.GetButtonDown("Fire1"))
09         {
10             Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
11             RaycastHit hitInfo;
12
13             if (Physics.Raycast(mouseRay, out hitInfo))
14             {
15                 Health enemyHealth = hitInfo.transform.GetComponent<Health>();
16                 if (enemyHealth != null)
17                 {
18                     enemyHealth.Damage(50);
19                 }
20             }
21         }
22     }
23 }

```

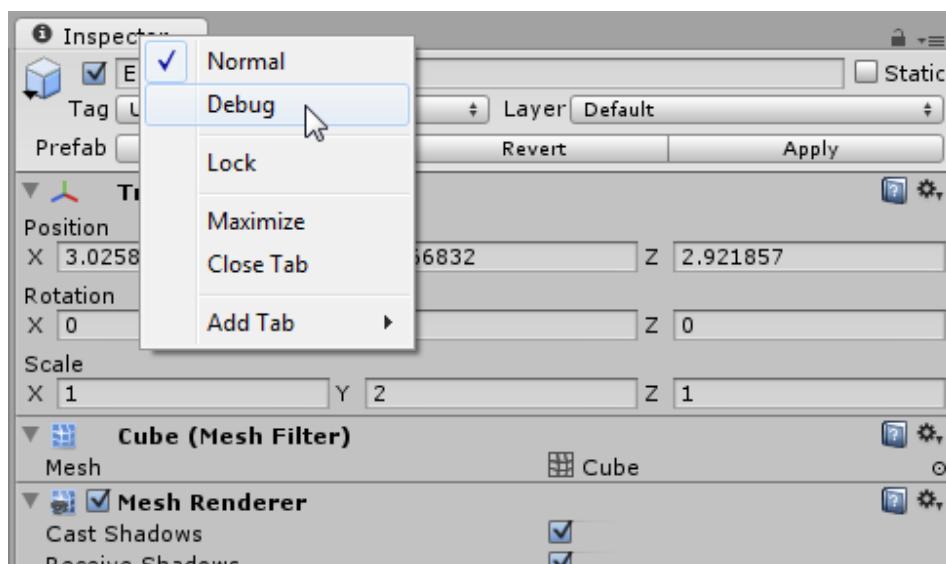
Artık raycast'in çarptığı objenin ismini print etmek yerine o objenin Health component'ine erişiyoruz. Eğer objenin Health componenti yoksa (raycast mesela zemine çarptıysa) "`enemyHealth != null`" ifadesi false döndürür ve if'in içine girilmez. Ama eğer Health componenti varsa o zaman if'in içine girilir ve Health scriptindeki Damage fonksiyonu çağrılarak objeye 50 hasar verilir.

Bu koddaki sıkıntısı verilen hasarın sabit (50) olması. Bu değeri bir değişken vasıtasyyla değiştirebilirsek daha güzel olur:

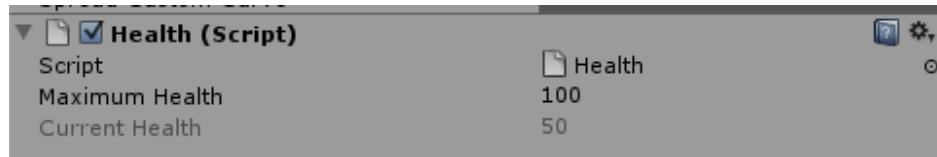
```

01 using UnityEngine;
02 using System.Collections;
03
04 public class RifleWeapon : MonoBehaviour
05 {
06     [SerializeField]
07     int _damageDealt = 50;
08
09     void Update()
10     {
11         if (Input.GetButtonDown("Fire1"))
12         {
13             Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
14             RaycastHit hitInfo;
15
16             if (Physics.Raycast(mouseRay, out hitInfo))
17             {
18                 Health enemyHealth = hitInfo.transform.GetComponent<Health>();
19                 if (enemyHealth != null)
20                 {
21                     enemyHealth.Damage(_damageDealt);
22                 }
23             }
24         }
25     }
26 }
```

Scripti henüz test etmek zor çünkü düşmanların ölmesini daha ayarlamadık. Şu anda test etmenin tek yolu bir düşmana birkaç kez ateş etmek ve ardından o düşmanı seçip Inspector'u da Debug moduna alarak düşmanın sağlığını (Current Health) kontrol etmek. Inspector'a sağ tıklayıp Debug deyin:



Artık Inspector private değişkenlerin değerlerini de gösteriyor ama bu değerler koyu gri renkte gözükmüyor ve elle değiştirilemiyor.



Şimdi oyunu çalıştırıp bir zombiye ateş edin ve zombinin canının azalıp azalmadığını kontrol edin. Azalmadıysa bir sıkıntı var demektir, bu bölümü baştan okuyup bir detay atlamağınızdan emin olun.

Ölünce Olacakları Ayarlamak

Bir düşman ölünce bizi takip etmeyi bırakıp yere yiğilmeli. Ama henüz yere yiğilacak bir karakter modelimiz yok; o yüzden şu safhada ölen düşmanları direkt sahneden sileceğiz.

Bunun için ise Health scriptinde ilgili değişiklikleri yapalım:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     void Start()
12     {
13         _currentHealth = _maximumHealth;
14     }
15
16     public void Damage(int damageValue)
17     {
18         _currentHealth -= damageValue;
19
20         if (_currentHealth <= 0)
21         {
22             Destroy(gameObject);
23         }
24     }
25 }
```

Artık sağlık sıfır ulaştığı vakit objeyi siliyoruz. Silme işlemini Destroy fonksiyonu ile yapıyoruz.

Oyunu çalıştırın ve bir zombiye saldırın. Yeterince vurduğunuzda zombi puf diye yok olacak. İleride zombi öldüğünde yok olmak yerine yere yiğilacak; şimdilik bununla idare edin.

Fare İmlecinin Görünmesini Engellemek

Ekranın ortasında bir nişangahımız var, fare imlecinin ortada dolaşıp kafamızı karıştırmasına gerek yok. Fare imlecinin hareket etmesini engellemek ve onu görünmez kılmak için bir kod bulunmakta.

RifleWeapon scriptini güncelleleyelim:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class RifleWeapon : MonoBehaviour
05 {
06     [SerializeField]
07     int _damageDealt = 10;
08
09     void Start()
10     {
11         Screen.lockCursor = true;
12     }
13
14     void Update()
15     {
16         if (Input.GetButtonDown("Fire1"))
17         {
18             Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
19             RaycastHit hitInfo;
20
21             if (Physics.Raycast(mouseRay, out hitInfo))
22             {
23                 Health enemyHealth = hitInfo.transform.GetComponent<Health>();
24                 if (enemyHealth != null)
25                 {
26                     enemyHealth.Damage(_damageDealt);
27                 }
28             }
29         }
30     }
31 }
32 }
```

Screen.lockCursor bir boolean'dır ve true yaparsanız imleç hem görünmez olur hem de hareket etmeyi keser.

Oyunu test ederseniz bunun işe yaradığını ama imlecin bir süre sonra tekrar belirdiğini göreceksiniz. Bu sorun sadece Unity editöründe böyle. Oyunu build ederseniz bu sorunu yaşamazsınız.

Escape tuşuna basınca bir menü çıktığını varsayıyalım. Bu durumda imleci geri görünür kılmamız gereklidir. Bunu başarmak için şu düzenlemeyi yapın:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class RifleWeapon : MonoBehaviour
05 {
06     [SerializeField]
07     int _damageDealt = 10;
08
09     void Start()
10     {
11         Screen.lockCursor = true;
12     }
13
14     void Update()
15     {
16         if (Input.GetKeyDown(KeyCode.Escape))
17         {
18             Screen.lockCursor = false;
19         }
20
21         if (Input.GetButtonDown("Fire1"))
22         {
23             Screen.lockCursor = true;
24             Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
25             RaycastHit hitInfo;
26
27             if (Physics.Raycast(mouseRay, out hitInfo))
28             {
29                 Health enemyHealth = hitInfo.transform.GetComponent<Health>();
30                 if (enemyHealth != null)
31                 {
32                     enemyHealth.Damage(_damageDealt);
33                 }
34             }
35         }
36     }
37 }
38

```

Artık ESC tuşuna basınca imleç belirecek, sol mouse tuşuna basıp ateş ettiğimizde tekrar kaybolacak.

Özet Geçecek Olursak...

Bu bölümde hatırlı sayılır derecede farklı şeyler gördük: raycast sistemi, objeyi yok etmek, konsola output vermek ve imleci gizlemek.

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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

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

Şimdi sahnedeki dandık 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 programcisından, 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 deðin farketmediğimiz hataları (bug) gün yüzüne çıkaracağız.

3D Modeller Hakkında

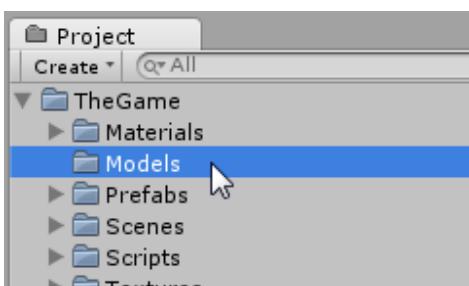
3D modeller, sahnelerimizde yer alan/alacak olan üç boyutlu objelerdir. Genellikle 3D modeller animasyonlara sahip olurlar. Örneğin bir zombi modelinden yürüme, dökilme, saldırıcı gibi animasyonlara sahip olması beklenir.

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 kullanacağımız. Buna 3D modeller ve animasyonlar da dahil.

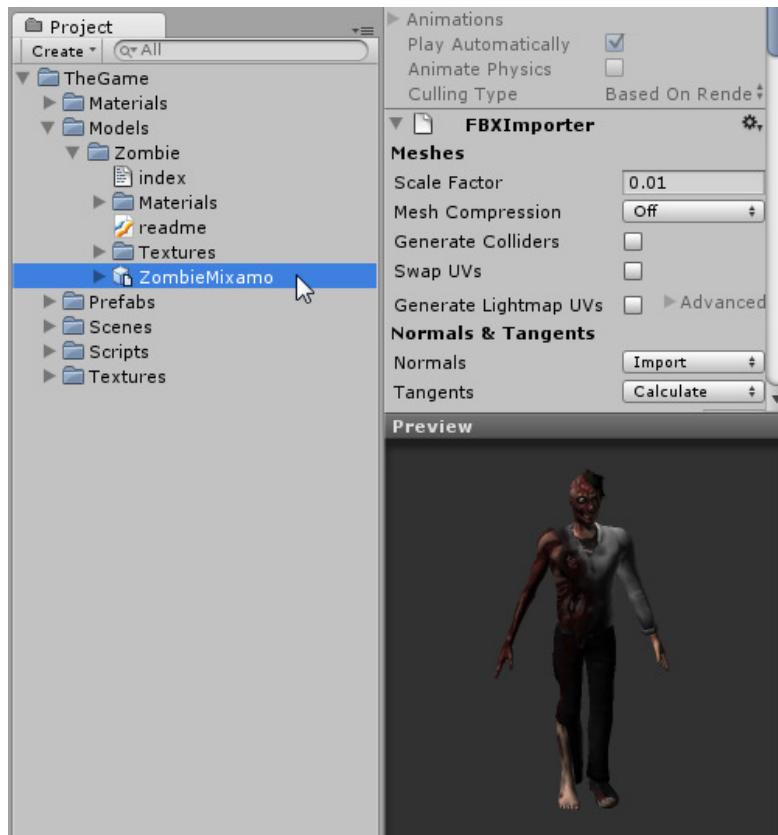
“TheGame” içinde “Models” klasörü oluşturun. Buraya 3D modelleri koyacağımız.



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 kameraları 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.

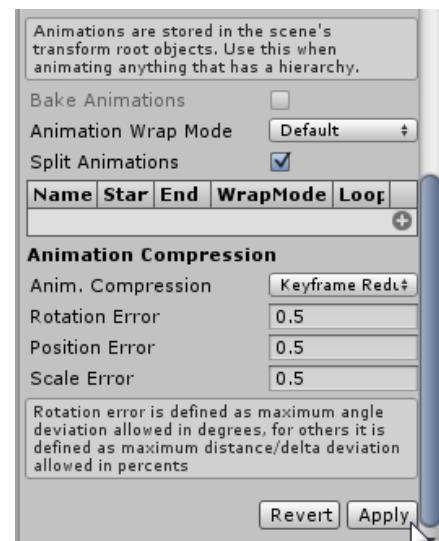
Bu değeri 0.58 olarak değiştirin ve Inspector'un aşağı kısımlarında yer alan “Apply” butonuna basın. Değişiklik modele uygulanacak.

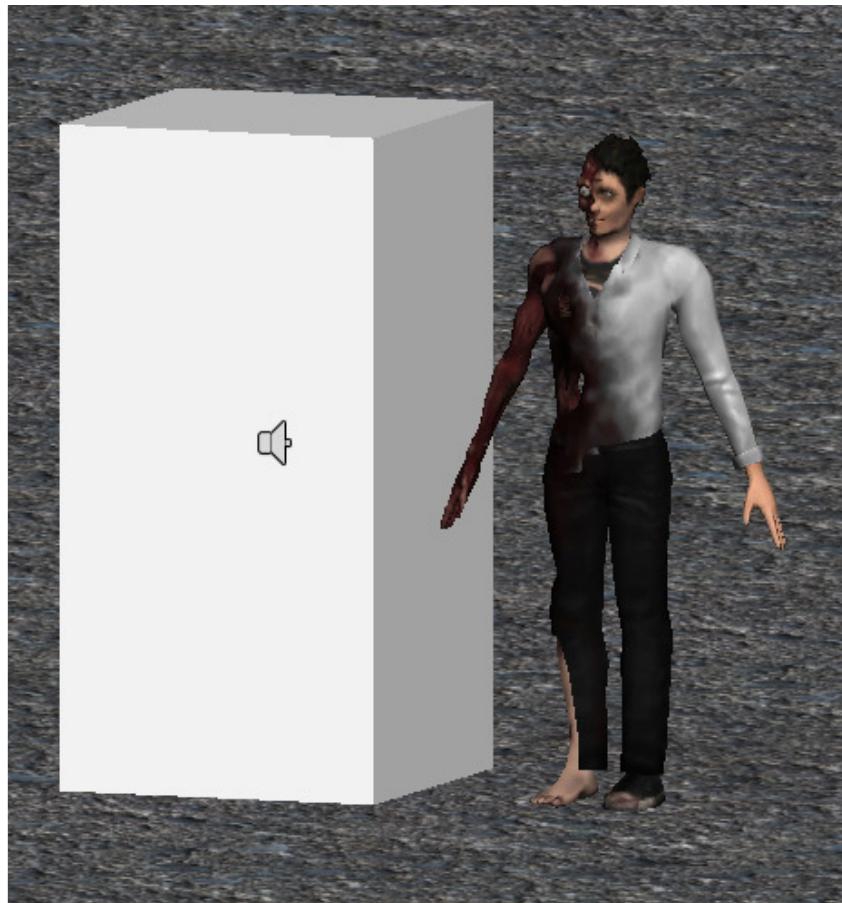
Scale Factor modelin büyüğünü belirler. Artık zombi, sahnedeki küp modelleri ile hemen hemen aynı boyutlarda.

ÇEVİRMEN EKLEMESİ: Neden boyutlandırma aracını (**R**) kullanmadık diyebilirsiniz. Eğer boyutlandırma aracını kullandığınız da sonuç kesinlikle birebir aynı olacaktır, ama tek ve önemli bir fark olacaktır: **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 yüksekliğindedir. 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 bırakırsak zombi 2 kat daha uzun olacak ve "garip" bir görüntü oluşacaktır.

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 seçeceğiz onun da pivot noktasının ayak hizasında olduğunu göreceksiniz. İki objenin pivot noktalarını üst üste koyacaksanız zombi tam yere deðiyor konumda olacak.

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



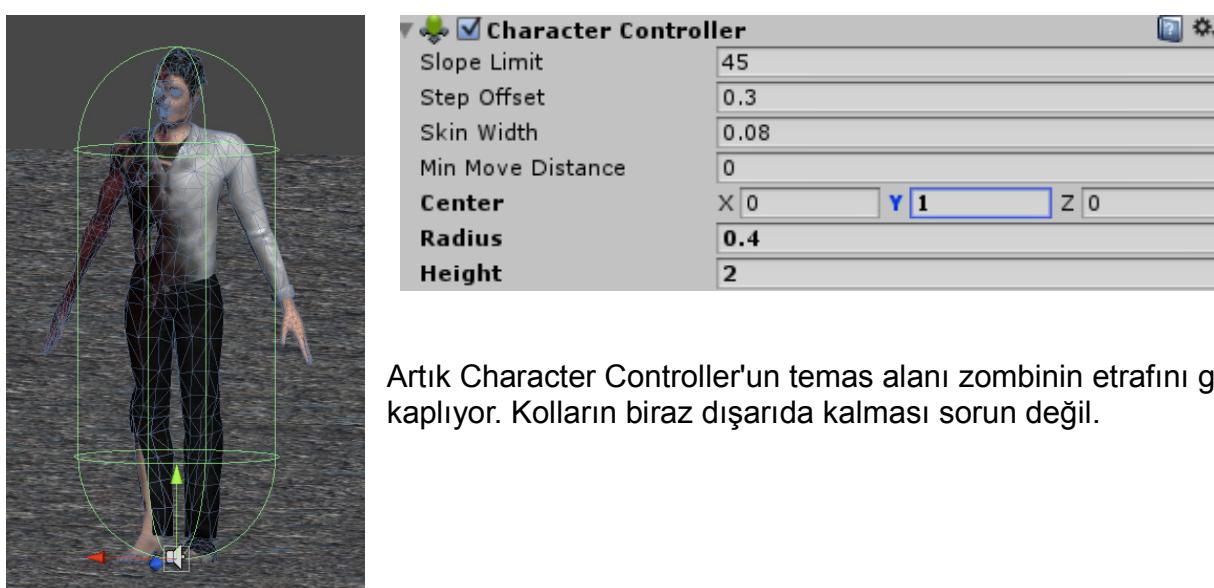
Şimdi zombi modelini Enemy objesinin içine atacağız. ZombieMixamo'yu Hierarchy'den tutup Enemy'nin üzerine sürükleyerek Enemy objesinin bir **child objesi** yapın.

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'ının 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:

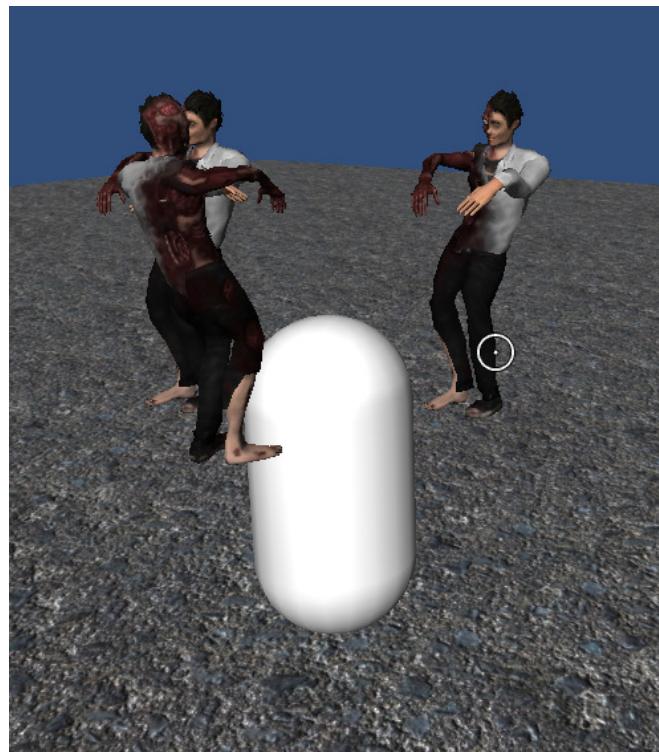
Height: 2
Radius: 0.4
Center: (0,1,0)



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

Zombi modelini Enemy objesine verdığimize 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 safhalarla 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 vasıtıyla düşman yere düşmeye başlıyor.

Oyunu çalıştırığınızda 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, tipki Euler açılar gibi eğimi temsil etmek için kullanılan bir veri türüdür.

Oyunu çalıştırın. Malesef zombiler yanımıza gelince sapılmaya 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=rrUCBOIJdt4>.

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. İşleyışı daha farklıdır yani.

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

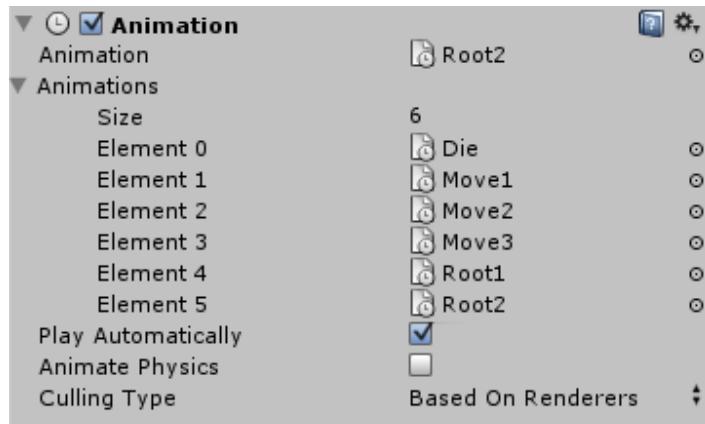
Artık zombiler bizi "normal" bir şekilde takip ediyorlar.



Zombilere Yürüme Animasyonu Vermek

Geldik oyunu renklendirecek bir detaya: bu bölümü bitirdiğimizde zombilerimiz yürüyecek! 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.

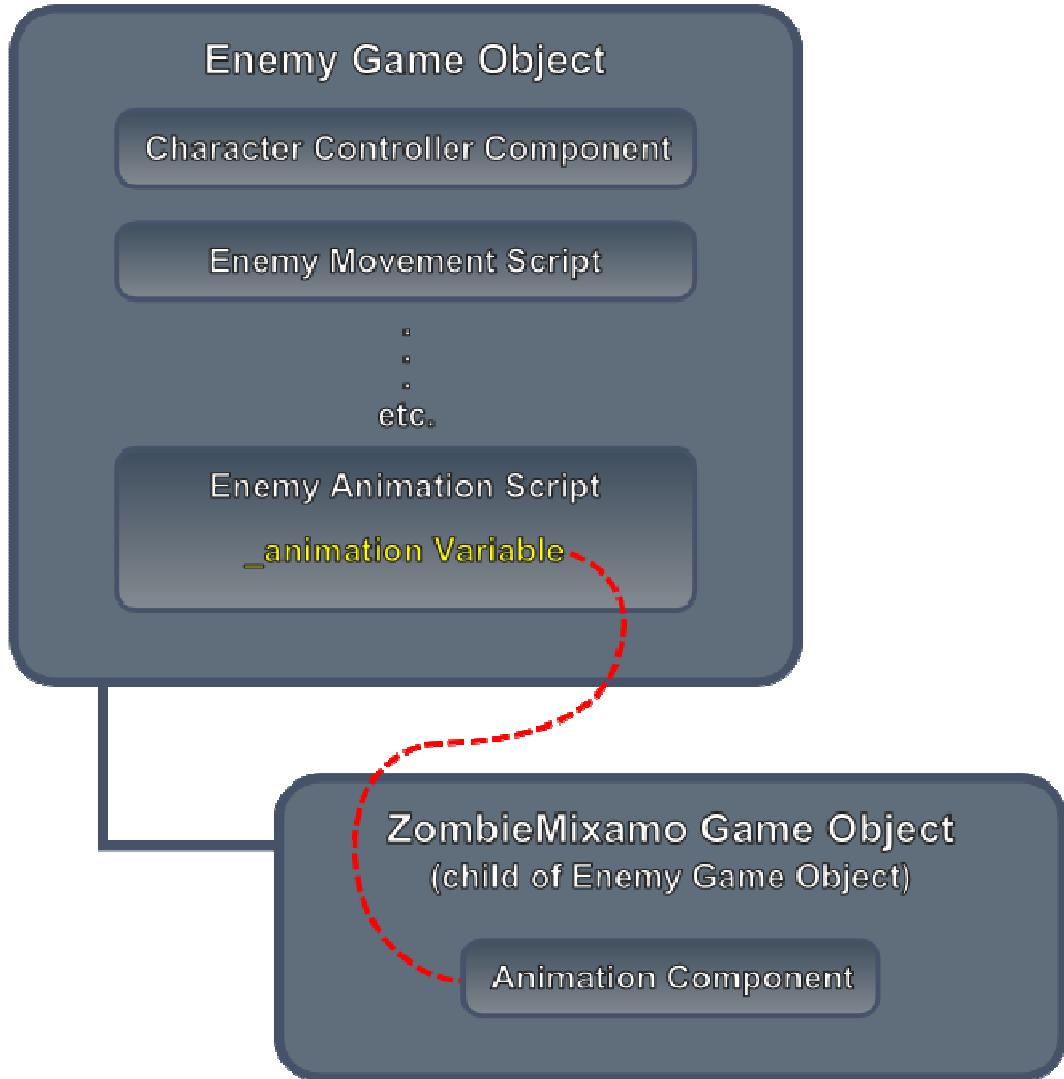
“EnemyAnimation” adında yeni bir C# scripti oluşturun:

```
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 çağrıncı olan bitenin şunu 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 = WrapModeLoop” 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 sağlayabiliyoruz:

```
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 verdigimiz için her zombinin animasyonu farklı bir kareden oynamaya başlayacak ve animasyonları eş zamanlı oynuyor izlenimi uyandırmayacak.

Farklı Yürüme Animasyonu Oynatmak

Bildığınız gibi zombinin üç farklı yürüme animasyonu var. Kodu biraz daha genişletecek her zombinin bu üç animasyondan rastgele birisini kullanmasını sağlayabiliyoruz:

```

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 çağrıryorduk: "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 yüreme animasyonundan birisinin ismini animationToPlay değişkeninde depoluyoruz.

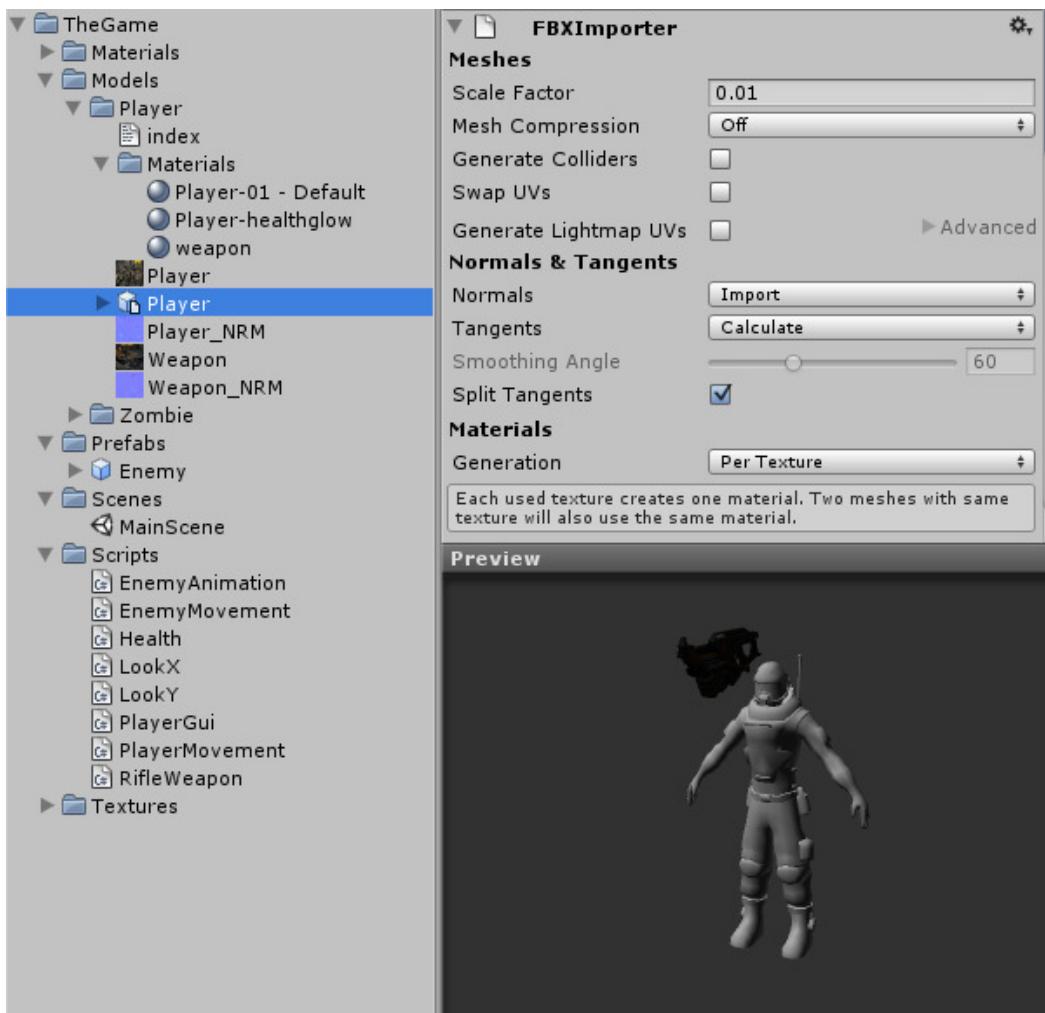
Kaydetmeyi Unutmayın!

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!

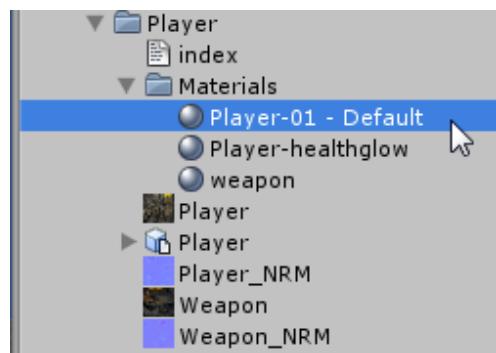
Winrar arşivinin oradaki "Player" klasörünü projenizin Models klasörünün içine kopyalayın. Klasördeki "index" dosyasını isterseniz silebilirsiniz.



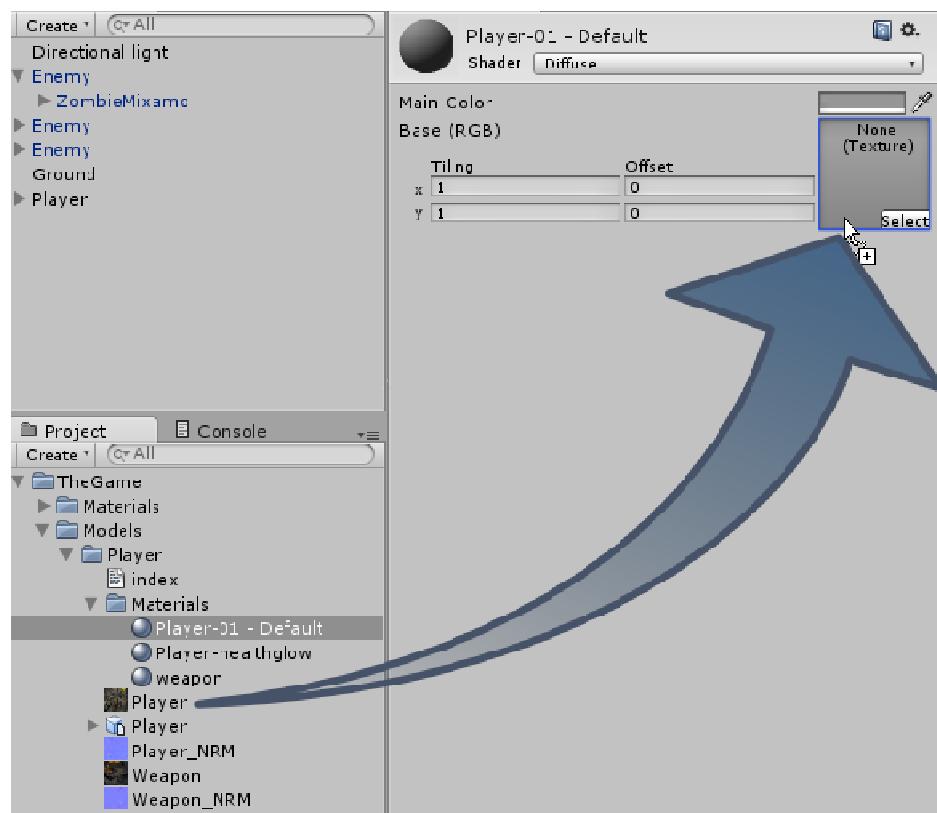
Modeli Project panelinden seçerseniz gri olduğunu göreceksiniz, çünkü henüz modele kaplaması (texture) atanmamış.

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 yazda "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.

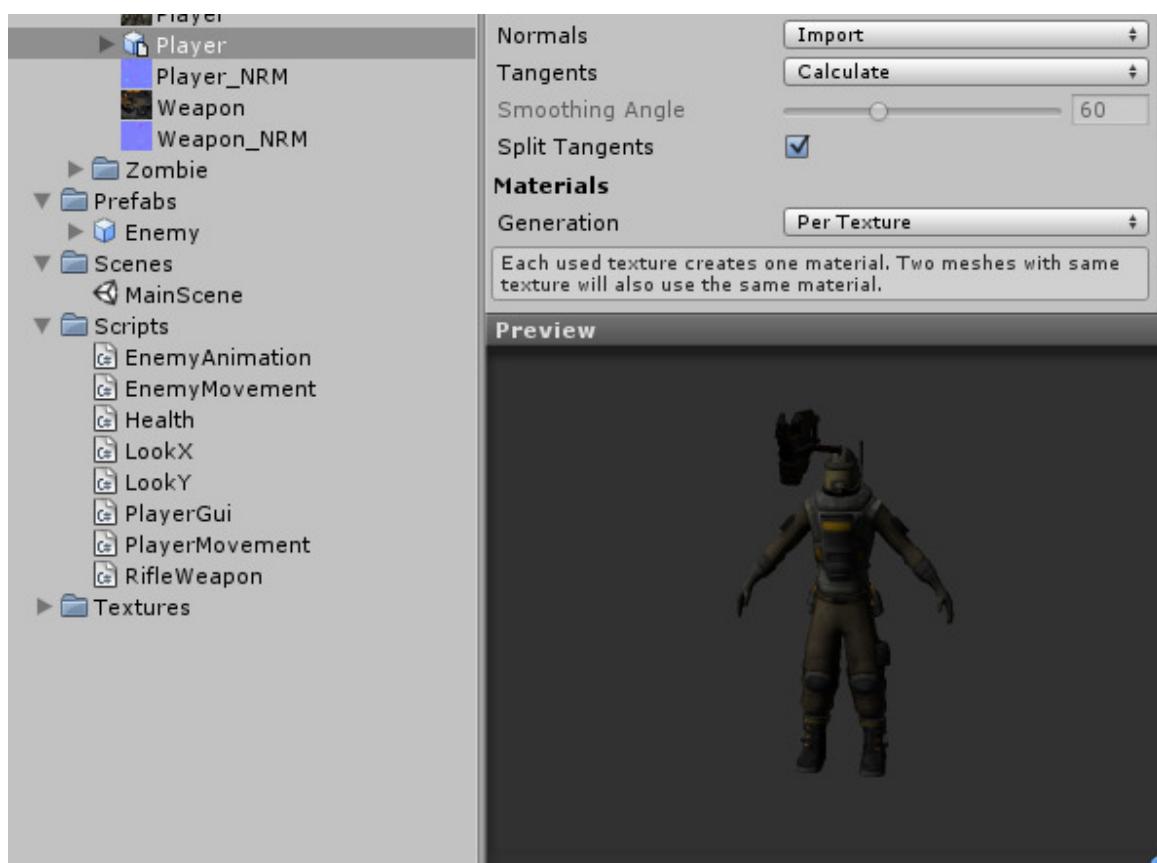


Aslında yapacağımız işlem çok basit. Project panelinden Player isimli kaplamayı (texture) sürükleyip "Player-01 – Default" materyalindeki ilgili kısma yerleştirin:



ÇEVİRMEN EKLEMESİ: "Player-01 – Default" materyalinin rengi (**Main Color**) gri. Bu, kaplamanın modelin üzerinde gereğinden daha koyu gözükmemesini sağlıyor. Büyüyük ihtimalle yazar bu detayı söylememeyi unuttu. Eğer **Shader'i** **Diffuse**'tan **Mobile-Diffuse**'a değiştirirseniz hem modeli render etmek daha hızlı gerçekleşir (**Mobile-Diffuse** shader'i daha basit işlemlere sahiptir) hem de kaplama, modelin üzerinde istediği gibi (daha aydınlatık) durur.

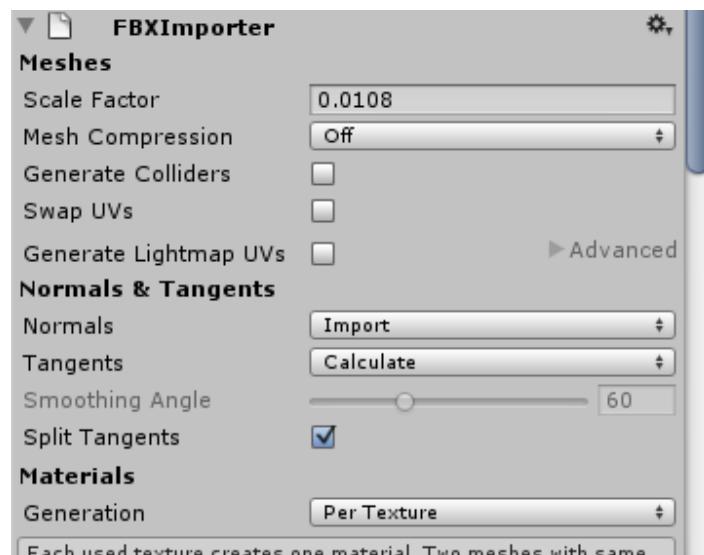
Artık Player modelini seçtiğinizde önizleme ekranında rengarenk bir model sizi bekliyor olacak:



Şimdi Player modelini Project panelinden Scene paneline sürükleyin.



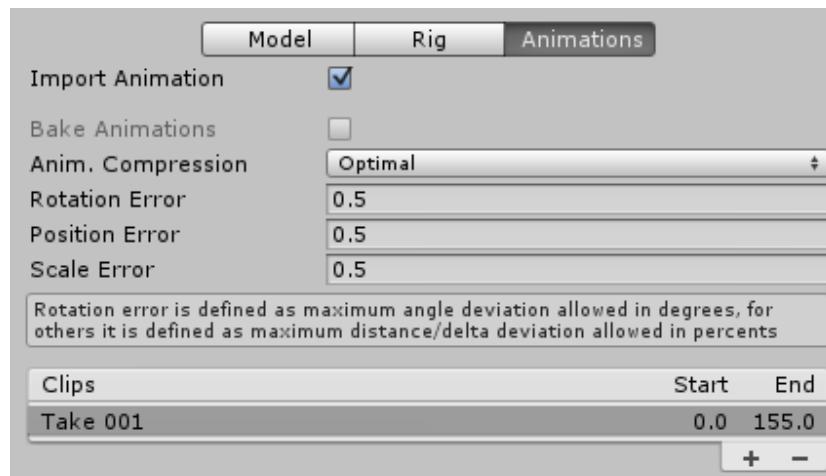
Scale Factor'ü 0.0108 yaparak modelin boyunun kapsülüne boyuyla eşdeğer düzeye gelmesini sağlayın. Scale Factor'ü değiştirdikten sonra Apply demeyi unutmayın.



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

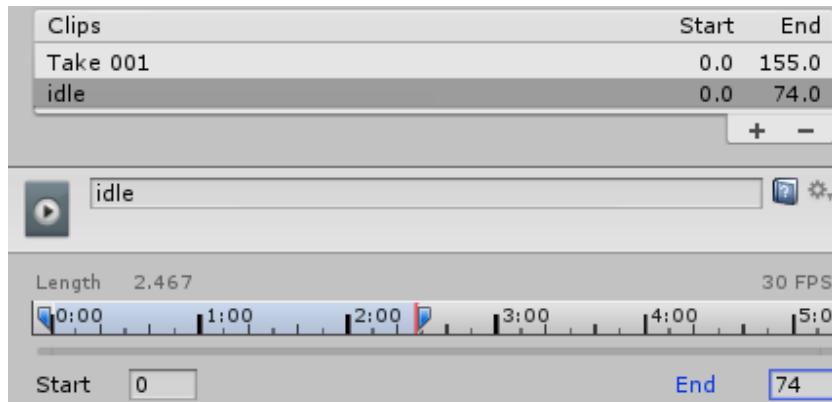
Nedeni ne olursa olsun bu uzun animasyonu parçalara bölmek zorundayız. Böylece karakterin istediğimiz animasyonunu oynatabileceğiz. Bu işlemi Inspector'dan yapıyoruz.

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



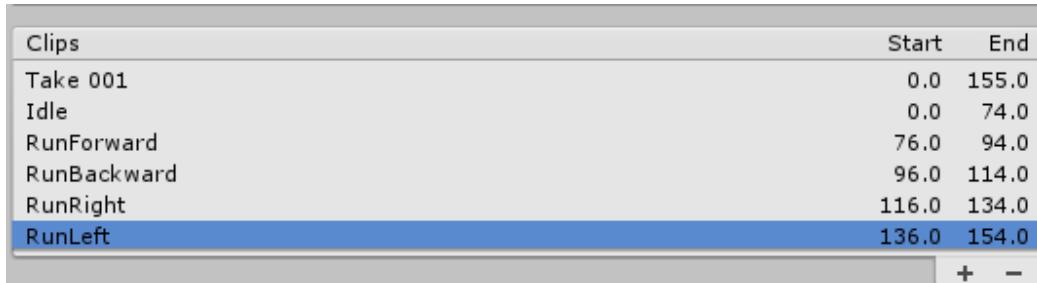
Klibi önizleme penceresinde oynatırsanız tüm animasyonların tek bir büyük animasyonda birleştirilmesinden neyi kastettiğimi görebilirsiniz.

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:

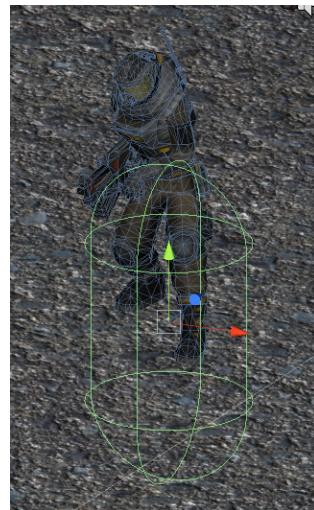


Tüm animasyonları resimdeki gibi ayırdıktan sonra "Take 001" animasyonunu seçin ve + işaretinin sağındaki - işaretine tıklayın. "Take 001" animasyonuna ihtiyacımız olmadığı için onu listeden bu yöntemle silelim. Ardından Apply butonuna basarak değişiklikleri modele uygulayın.

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:

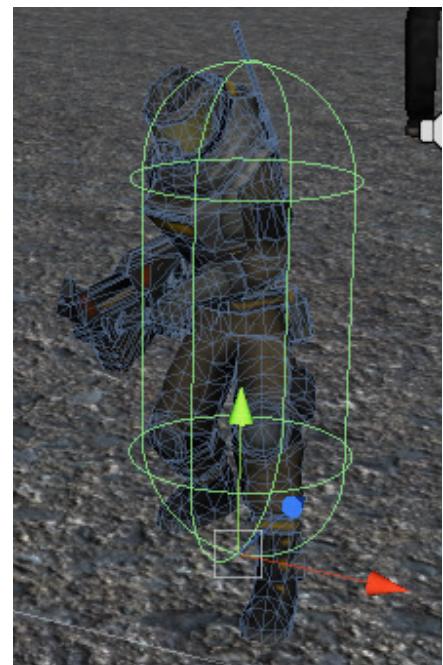
Height: 1.8

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.

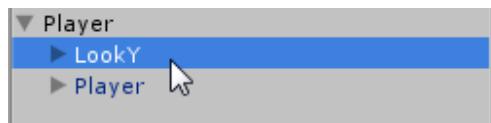


Nişangahın Konumunu Ayarlamak



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.

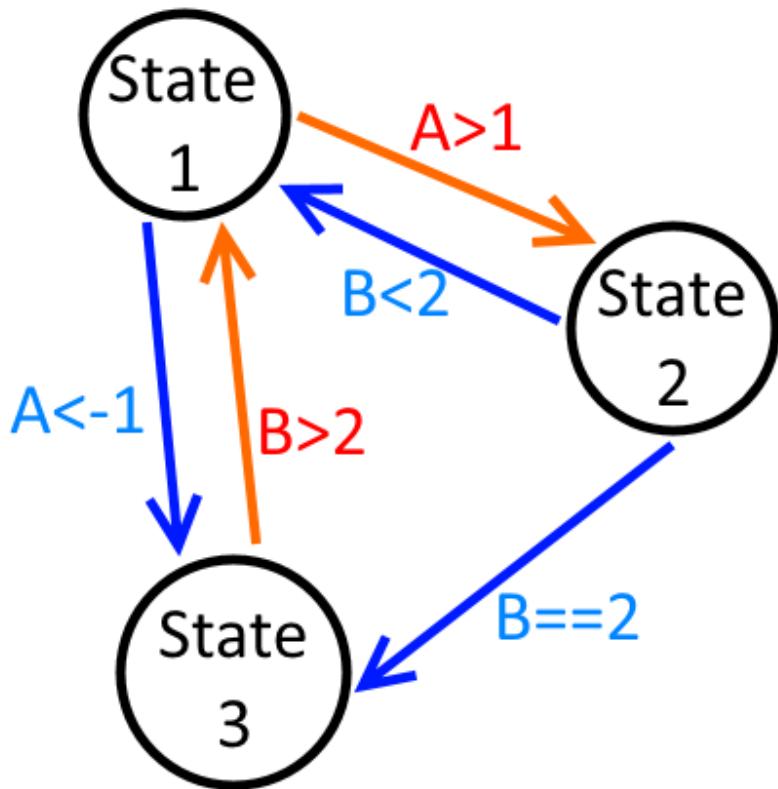


LookY objesini seçip pozisyonunu (1, 1.25, 1.5) yapın. Artık kamera daha yukarıda yer almaktır ve böylece imleç de daha yukarıda durmaktadır. Oyunu test edip farkı kendiniz de gözlemleyebilirsiniz.

Karaktere Yürüme Animasyonu Vermek

ÇEVİRMEN EKLEMESİ: Karaktere animasyon verdigimiz bu kısmı kendi başına 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 görmeyi istedim.

Mecanim sisteminin çalışma yapısı state-machine adında bir konsepte dayanıyor. Bu konsept üniversitede dijital devre tasarımlı 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?

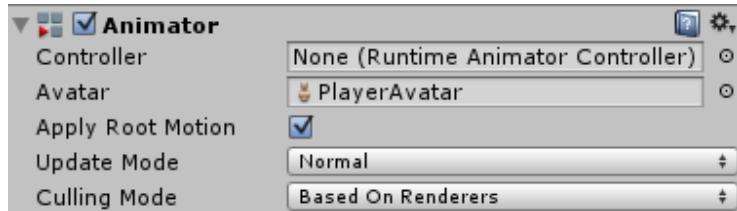
"State 1"den "State 2"ye çıkan okun üzerinde "A>1" yazıyor ve "State 1"den "State 3"e çıkan okun üzerinde "A<-1" yazıyor. Bu oklar birer "iki state arası geçiş" (**transition**)dır ve yanlarında yazan yazılar da bu transition'ın gerçekleşmesi için gerekli olan koşullardır (**condition**).

Şöyledi 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 hiçbirini 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.

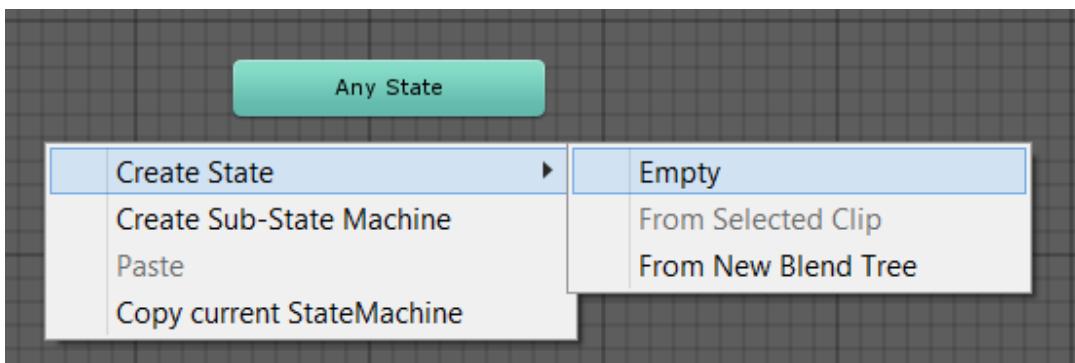
Peki bir state'teyken neler gerçekleşir? Dijital devre tasarılığında örneğin yeşil bir LED ışık yanabilir, bir yere bir işin yollanabilir vb. Unity'nin Mecanim sisteminde ise ne gerçekleşeceğini belli olur: animasyon oynatılır.

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.

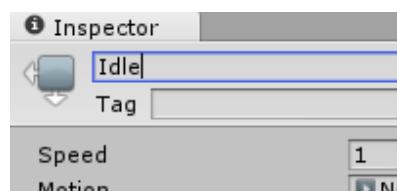


Bu component'te "Controller" adında bir değer yer almaktadır. Buraya bir Animator Controller (Mecanim state-machine) bağlamalıyız. Bunun için Project panelinde "**Create-Animator Controller**" yolunu izleyin ve oluşan asset'in ismini "PlayerAnimatorController" olarak değiştirin. Ardından bu controller'i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

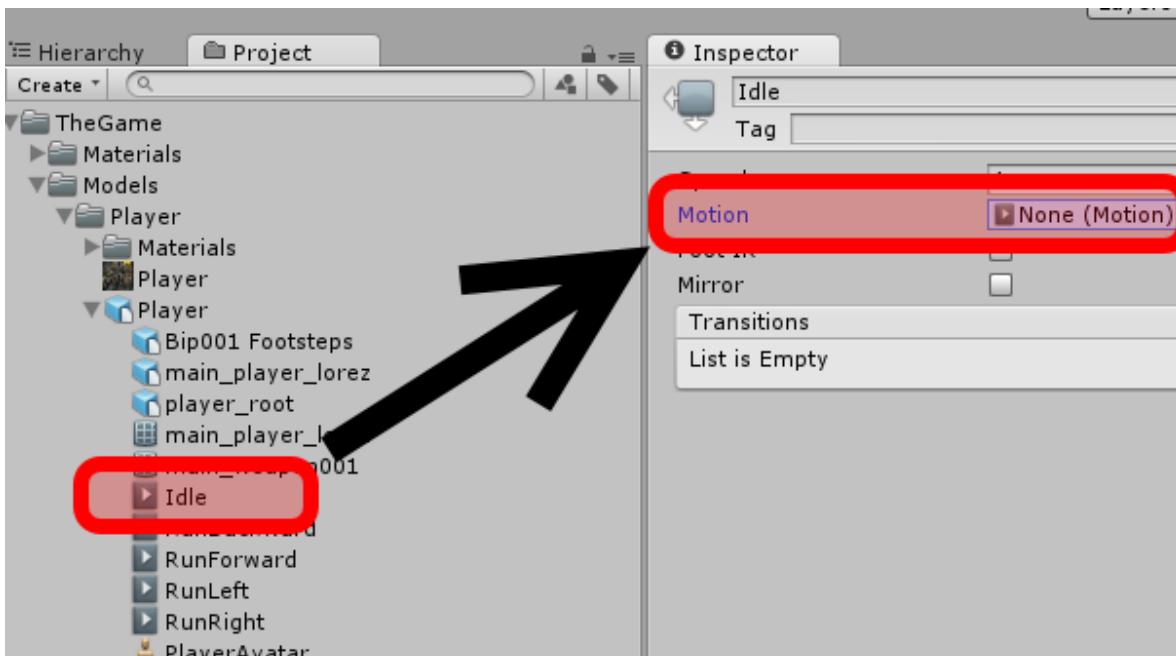
Şimdi "PlayerAnimatorController" a çift tıklayın. Editör arayüzünde **Animator** adında bir pencere belirecektir. İşte burada state-machine'imizi kuruyoruz. Şu anda ekranda sadece "Any State" var. Bu state'i kullanmayacağımız, nolduğuyla ilgili kafanızı yormayın. Ekranda boş bir yere sağ tıklayın ve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



Tıkladığınız yerde "New State" adında yeni bir state oluşturulacak. 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:



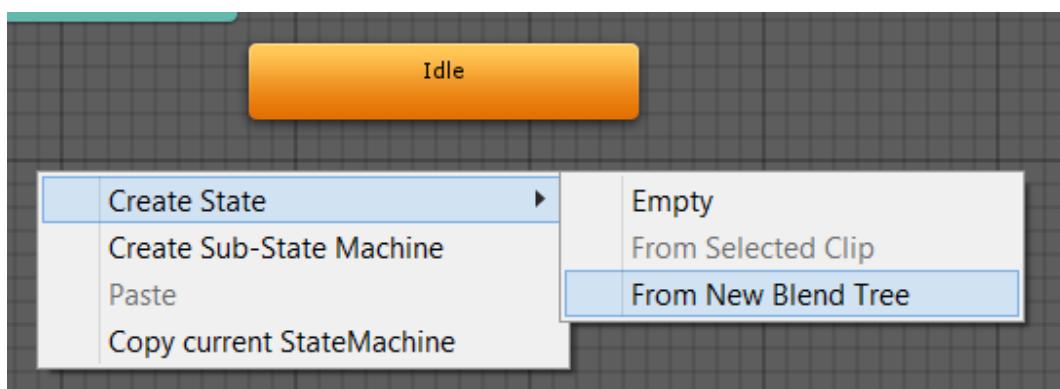
Bu state'te bulduğumuz zaman karakterin Idle, yani olduğu yerde sabit durma animasyonu oynayacak. Ama önce Idle animasyonunu Idle state'ine tanıtmalıyız. Bunun için ise Idle state'ini seçin ve **Motion** kısmına değer olarak Project panelinden Idle animasyonunu sürükleyin:



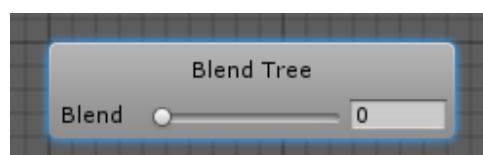
Oyunu şu halde çalıştırıcaz olursanız karakterin Idle animasyonunu loop halinde oynattığını göreceksiniz. İlk aşamayı tamamlamış olduk. Sırada hareket etme animasyonları var.

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-sağ gitme animasyonu oluyor. Bunu yapmak Mecanim ile mümkün.

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



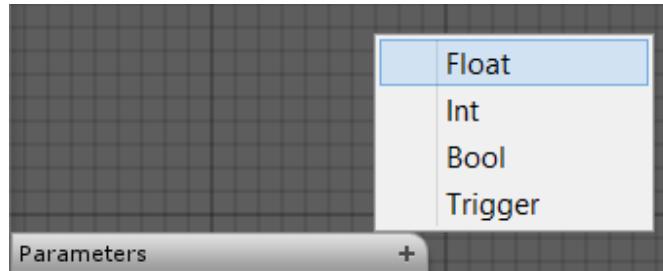
Oluşan state'in ismini "RunForward" olarak değiştirin. Şimdi RunForward state'ine çift tıklayın. Farklı bir state ekranı karşınıza gelecek:



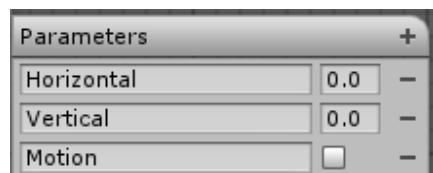
Blend Tree formatı birden çok animasyonu harmanlamak için idealdir. Idle'da Blend Tree kullanmadık çünkü Idle animasyonu tek bir animasyon. Ama ileri koşma animasyonunda harmanlama işlemi yapmamız zorunlu zira dediğim gibi, ileri giderken sağa veya sola da gitmek isteyebiliriz.

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

Animator penceresinin sol alt kısmında Parameters var. Burası vasıtıyla 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:

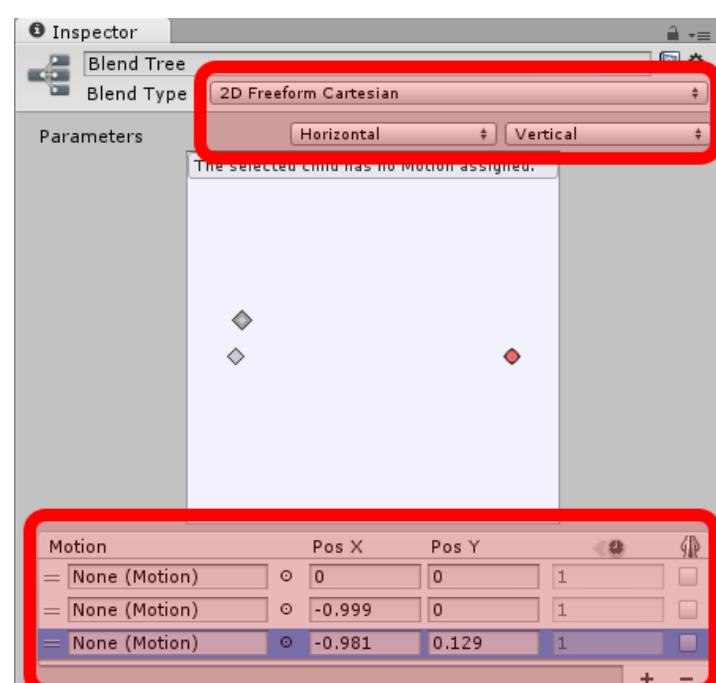


Bu değişkenlerin değerlerini ileride script ile dinamik olarak ayarlayacağız. "Horizontal" değişkeni klavyenin sağ-sol ok tuşlarından aldığımız input'u `"Input.GetAxis("Horizontal")"` değer olarak alacak ve "Vertical" da klavyenin ileri-geri ok tuşlarından aldığımız input'u `"Input.GetAxis("Vertical")"` değer olarak alacak. "Motion" iki state arası geçiş yaparken kolaylık sağlama için oluşturduğum bir değişken. Eğer klavyeden herhangi bir yön tuşuna basıysak o zaman karakter hareket ediyor olacak (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 ve **"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:

Motion	Pos X	Pos Y	
= RunLeft	○ -1	0	1
= RunForward	○ 0	1	1
= RunRight	○ 1	0	1

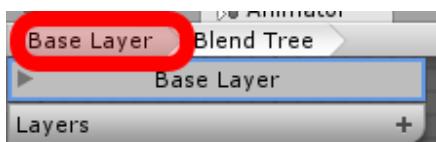
"Pos X" ve "Pos Y" sırasıyla parametre olarak girdiğimiz "Horizontal" ve "Vertical"ı temsil ediyor. Şöyle ki eğer Horizontal'ın değeri -1, Vertical'ın değeri 0 olursa (klavyeden sadece sol ok tuşuna basılıyor) sadece RunLeft animasyonu oynarken Horizontal 1, Vertical 0 iken (sadece sağ ok tuşuna basılıyor) 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 harmanlanması izleyin:

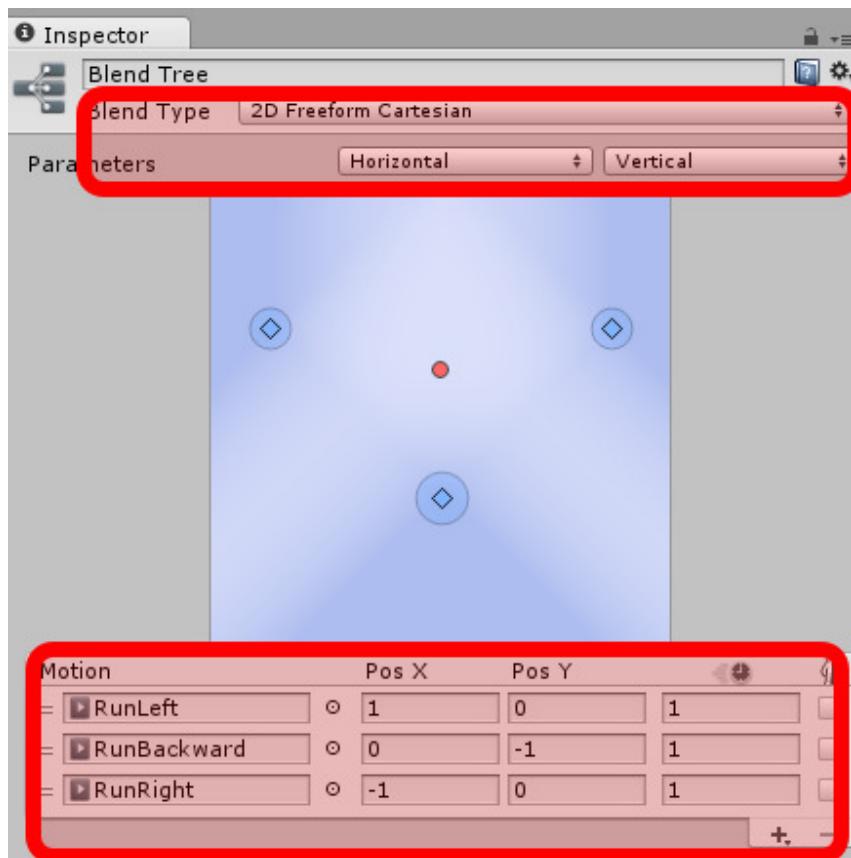


İleri koşma animasyonunu hayranlıkla izlemeniz bittiye şimdü geri koşma animasyonunu yapmaya başyalalım. Neden hem ileri hem de geri gitme animasyonunu tek bir state'te yapmadık diye haklı olarak sorabilirsiniz. Ben de bunu çok isterdim ama ilginç bir şekilde hem geri hem sağa giderken harmanlanması 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 içindeyseñiz sol üstteki "Base Layer" butonuna basarak ana ekran'a 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 basılıncı geri koşma ve sağa koşma animasyonlarını harmanlarken sağ-geri ok tuşlarına basılıncı 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 sayılırlı! 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 koyacağz. Bunun için + tuşuna basın. Koşulu şöyle değiştirin:



Eğer Vertical değişkeni -0.01'den küçükse (geri ok tuşuna basıysak) 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:

Transitions	Solo	Mute
= Base Layer.Idle -> Base Layer.RunBackward	<input type="checkbox"/>	<input type="checkbox"/>
= Base Layer.Idle -> Base Layer.RunForward	<input type="checkbox"/>	<input type="checkbox"/>

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-sağ, ileri-sola koşarken) kullanmayacağız, aynı zaman sadece sola koşarken veya sadece sağa koşarken de kullanacağız. Zaten bu yüzden state-machine'mızde "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 yaptıktı 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'lardan birindeyken tüm ok tuşlarını bıraklığımızda karakter Idle state'ine geçecek ve olduğu yerde durma animasyonunu (Idle) oynatmaya başlayacak.

Tek bir sorunumuz kaldı: diyalog sağ ok tuşuna basıp RunForward state'ine geçtiğim. Sağ ok tuşundan elimizi çekmeden geri tuşuna bastığımızda RunBackward state'ine geçiş yapıp geri-sağ 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 modelde 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($h \neq 0 \text{ || } v \neq 0$)" yazmadık diye sorabilirsiniz. Çünkü float sayıları bu şekilde karşılaştırırmak yanlıştır. Siz mesela bir float'u 0.5 diye ayarlırsınız 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 dökümanyonlara bakarak animasyonları daha iyi bir hale getirmeye çalışabilirsiniz.

Zıplama Animasyonu

Malesef model bir zıplama animasyonuyla birlikte gelmediği için ziplarken 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 insanlarınız. Çünkü Mecanim state-machine'ini belki daha önce hiç kullanmadınız ve gerçekten Mecanim karmaşık duran bir sistem. Bizim hiç ellememişiz 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.

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

Unity 3D İle Oyun Programlama

Bölüm 10: Ragdoll Sistemi



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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with
Creative Commons Attribution-NonCommercial 3.0 Unported

Karakterimiz ve zombiler, animasyonlu bir şekilde hareket edebilen birer gelişmiş canlı formuna girdiler ama öldükleri zaman *puf* diye sahneden yok oluyorlar. Onun yerine gerçekçi bir şekilde yere yiğilmaları daha iyi olmaz mıydı?

Bir ölmeye animasyonu ile bu işi halledebiliriz ama hayır, biz daha farklı bir sistem kullanacağız: Ragdoll sistemi! Bu sistem ile karaktere ve zombilere kemik ve eklemler vereceğiz ve öldükleri zaman fizik motorunun etkisiyle yere yiğilmalarını sağlayacağız.



Ragdoll Oluşturmak

Ragdoll'lar birden çok collider'in (kemik), *joint*'lerle (eklem) birbirine bağlanmasıyla meydana gelen sistemlerdir.

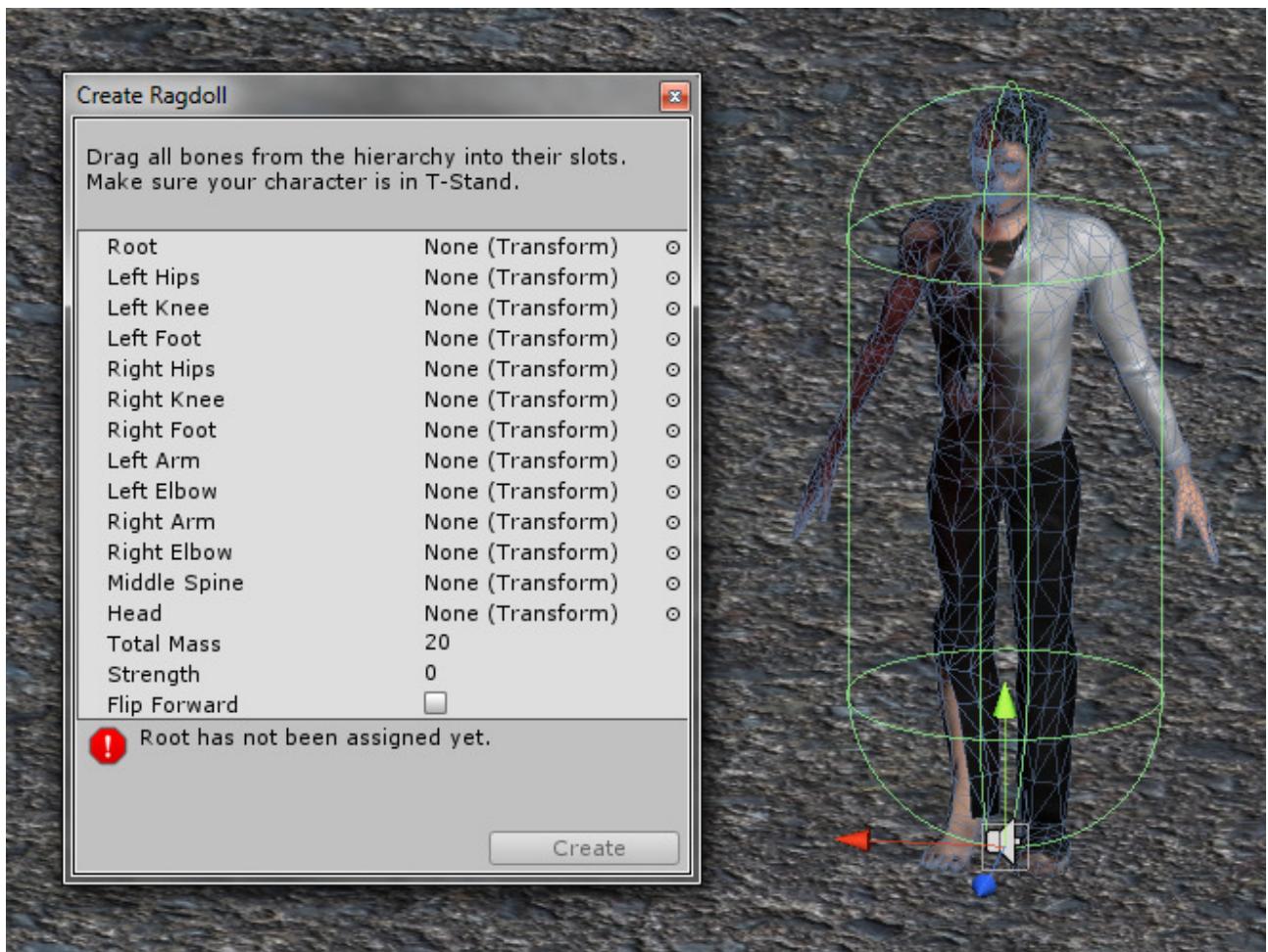


Görsel 10.1: Ragdoll'lar kürelerin, küplerin ve kapsüllerin birbirine bağlanmasıyla oluşur.

Unity'nin Ragdoll Wizard'ı ile ragdoll sistemi oluşturmak kolaydır. İhtiyacınız olan rigidbody'ler ve joint'ler sizin için otomatik olarak oluşturulur.

Bir zombi objesini seçip GameObject > Create Other > Ragdoll... yolunu izleyin.

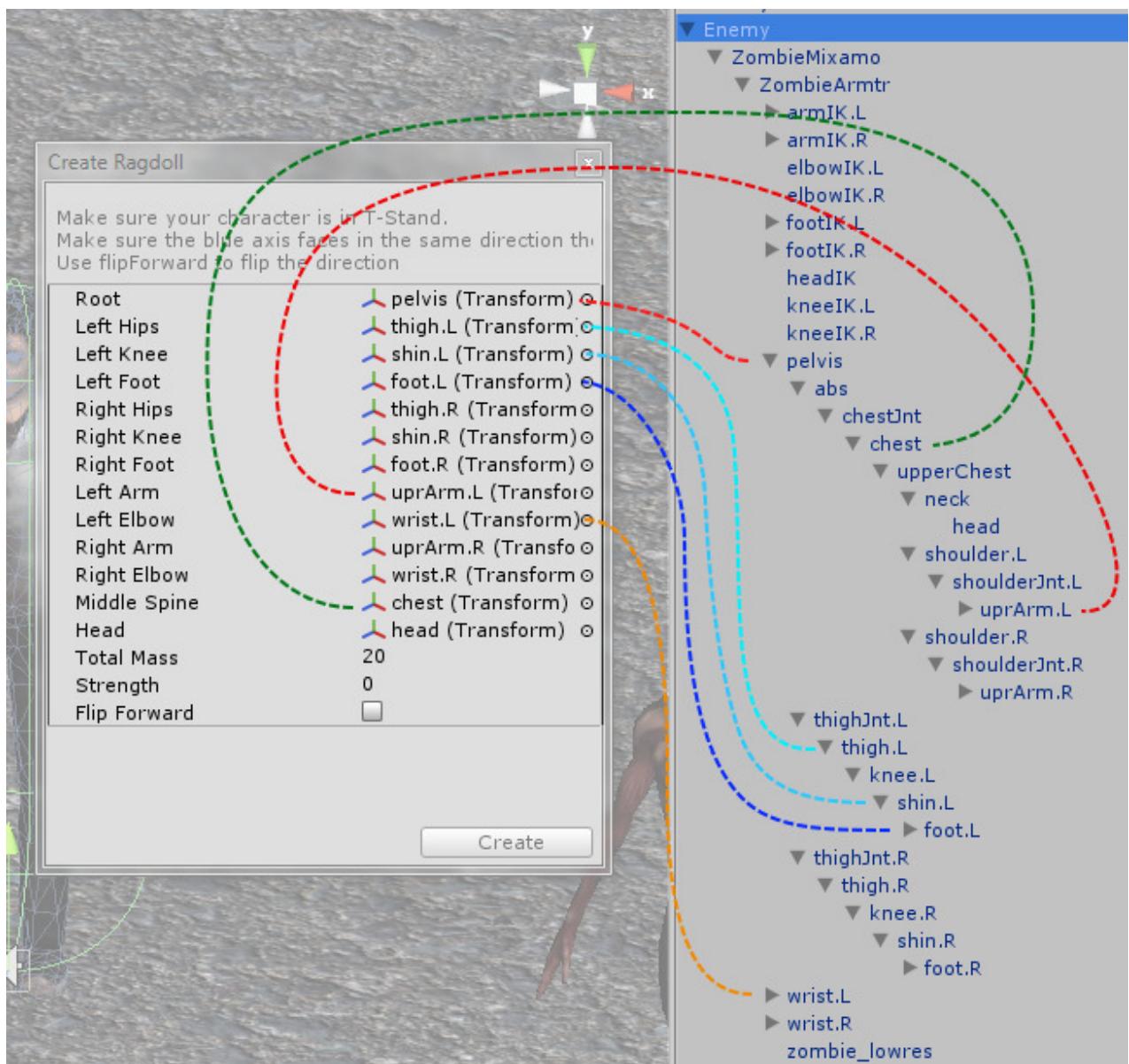
Create Ragdoll penceresi belirecek:



Pencerede gördüğünüz her değer bir kemiği temsil ediyor. Hierarchy panelinden uygun kemiği uygun kısma değer olarak vermemeliyiz.

ÇEVİRMEN EKLEMESİ: Ragdoll sistemi nedir? Bir karakterin yere yiğilirken (çoğunlukla ölüm sonucu) gerçeğe uygun şekilde yiğilmasını sağlayan bir sistem. Çalışma sistemi ise basit: Vücutunuzu, onları çevreleyen birer collider (ve birer rigidbody) veriliyor ve eklem noktalarındaki iki kemik (collider) birbirine Joint adı verilen bir component ile bağlanıyor. Her şey düzgün gitmişse karakter, rigidbody'deki yerçekiminin ve eklem noktalarındaki Joint'lerin etkisiyle yere gerçekçi bir şekilde yiğiliyor. Aslında ragdoll sadece yere yiğilmakla alakalı değil. Ragdoll daha çok bir insan modelinin insan gibi fiziksel tepki vermesini sağlayan bir sistem. Diyelim yerçekimi olmayan bir ortamda ragdoll'lu düşmanı kolundan vurup da kol kemiğine rigidbody.AddForce ile güç uyguladığınızda bu güç sadece kolun değil tüm vücudun gerçekçi bir şekilde tepki vermesine sebep olur (ragdoll vasıtası ile). Sizin bilmeniz gerekenler şunlar: ragdoll sistemi collider, rigidbody ve joint component'lerinden oluşur ve eğer rigidbody'de Is Kinematic açık değilse (yerçekimi etki ediyorsa) karakterin tüm bedeni gerçekçi bir şekilde yere yiğilir.

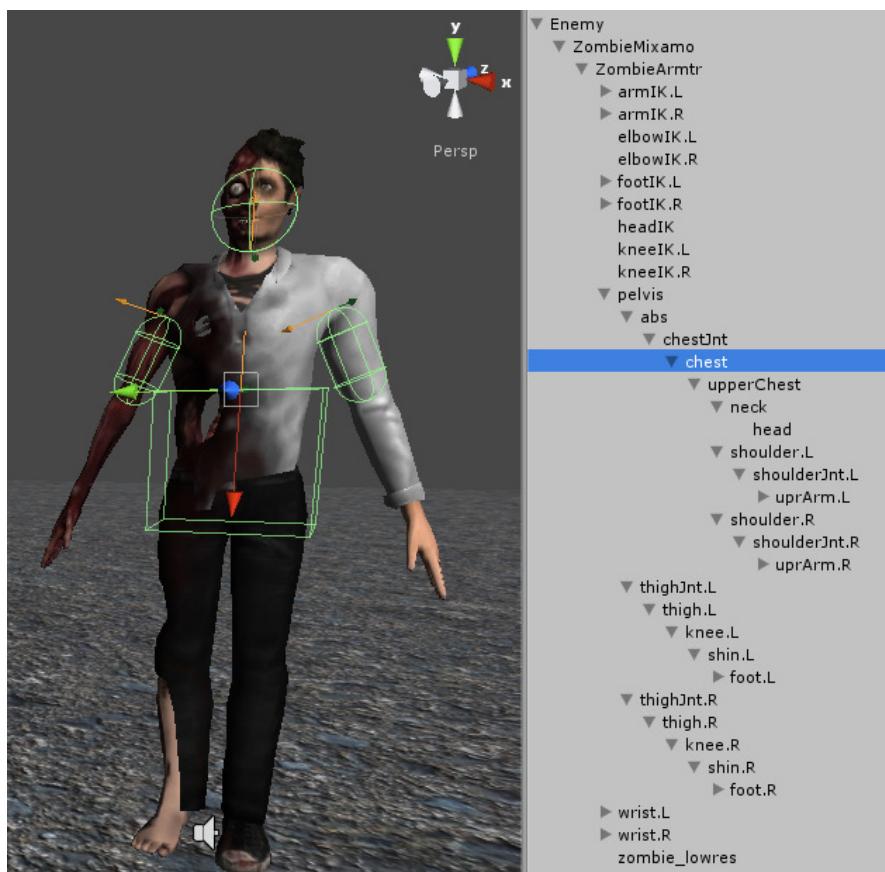
Altta referans resimden hangi kemiğe hangi objenin geleceğini görebilirsiniz:



Tüm değerleri uygun şekilde doldurduktan sonra Create butonuna basın. Gerisini Unity hallediyor ve ihtiyacınız olan collider'lar, rigidbody'ler ve joint'ler otomatik olarak oluşuyor.

Ragdoll'daki Bozuklukları Gidermek

Unity'nin ragdoll sihirbazı (ragdoll wizard) kusursuz değil. Bazen bir collider yanlış konumda oluşabilir. Zombinin "chest" child objesini seçin. Resimdeki gibi bir görüntü sizi karşılayacak:



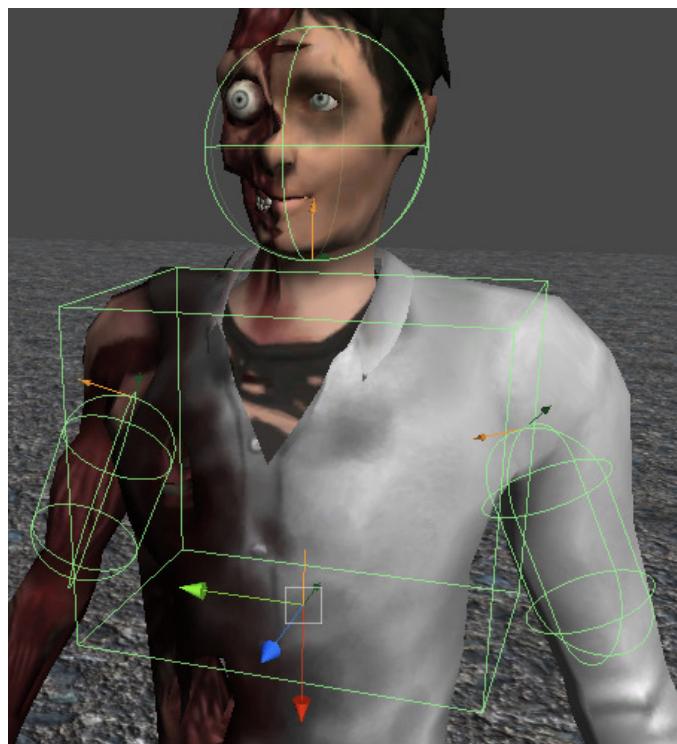
Göbek hizasında gördüğünüz collider'in göğüs (chest) hizasında olması gerekiyordu. Onu yeniden konumlandırmamız lazımdır.

ÇEVİRMEN NOTU: Nedense bende resimdeki hata oluşmadı. Yani chest collider'ı göbek hizasında değil gerçekten de göğüs hizasındaydı. Bu yüzden aşağıda anlatılan değişiklikleri benim yapmam gerekmeyecektir. Sizde de büyük olasılıkla bendeki gibi kusursuz olacak ragdoll.

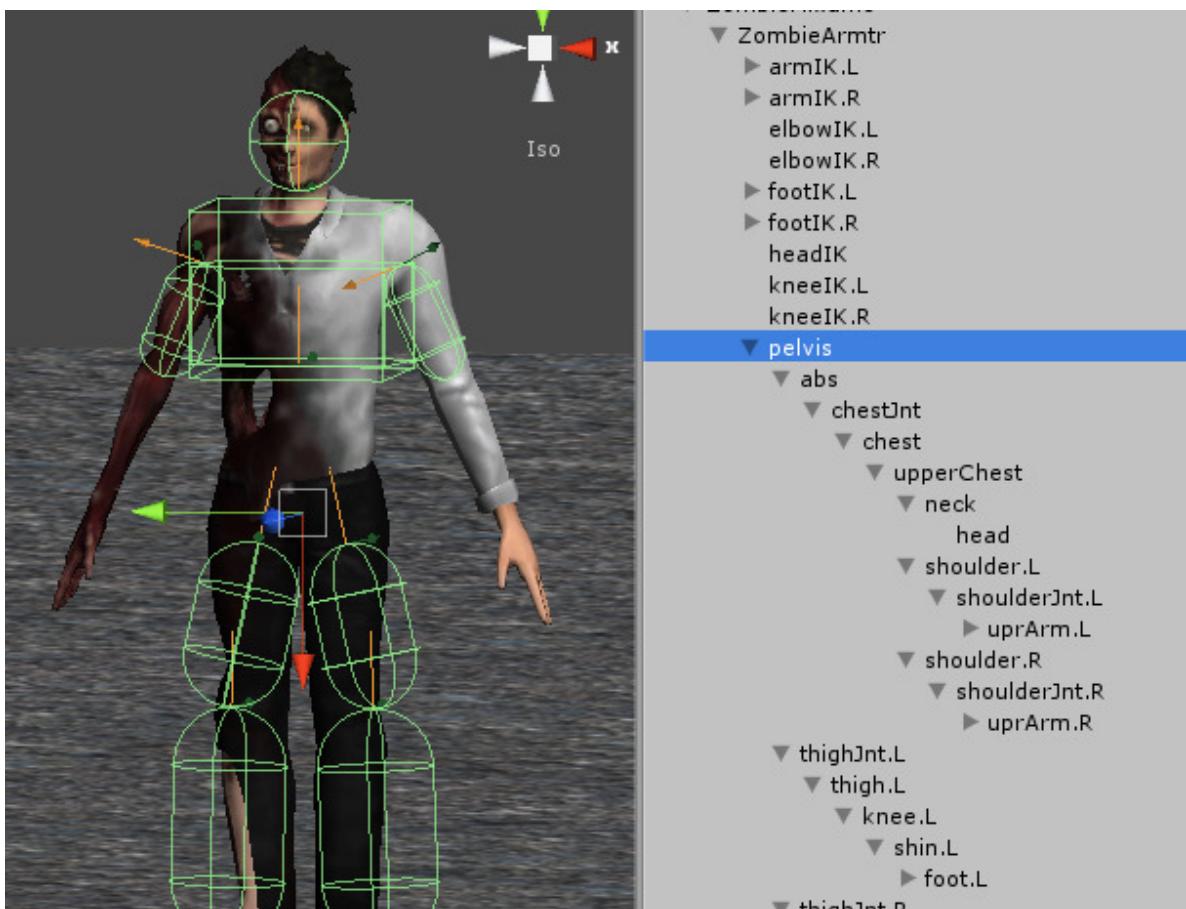
Shift'e basılı tutunca kutuyu çevreleyen noktalar belirecek. Bu noktalardan tutup sürükleyerek o collider'in boyutunu ayarlayabilirsiniz. Önce üst noktadan tutup yukarı çekip ardından da alt noktadan tutup yukarı çekerseniz de haliyle collider'ı yukarı taşımiş olursunuz.



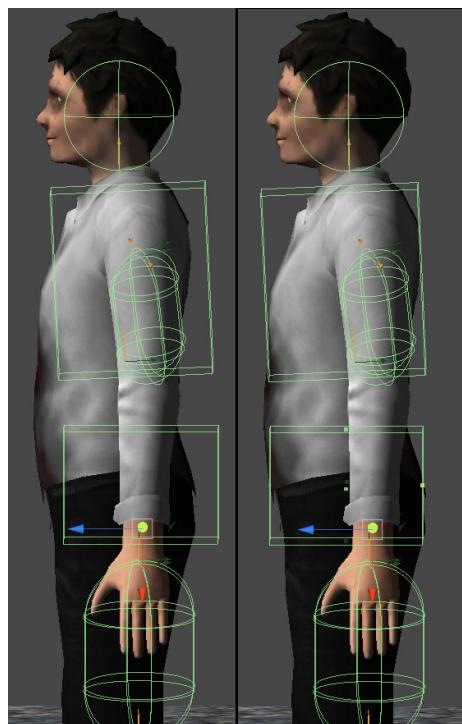
Bu yöntemi kullanarak göbek hizasındaki collider'i göğüs hizasına taşıyın.



"pelvis" objesini seçin. Büyük olasılıkla bu da yanlış konumda (**Bende doğru konumdaydı (göbek hizası)**). Bu collider'ı ise göbek hizasına çekin.



Collider'ların yandan görünüşüne de dikkat edin. Eğer collider gereğinden fazla genişse onu daraltın:

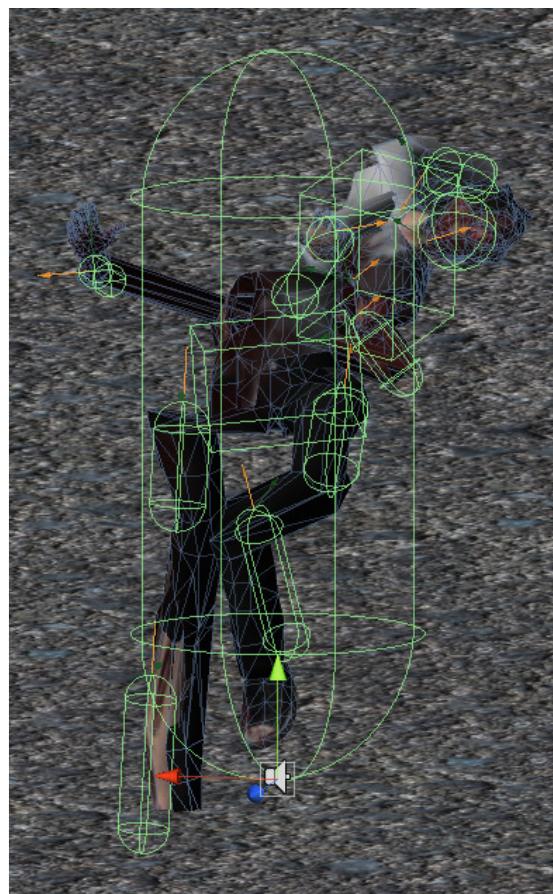


İşte bu kadar. Oyunu çalıştıracak olursanız zombinin "delirdiğini" göreceksiniz:



Görsel 10.2: Görünüşe göre zombi kendinden geçmiş!

Çünkü ragdoll'un collider'ları Character Controller component'inin collider'ı ile çakışıyor:



Görsel 10.3: Büyük kapsül şeklindeki collider diğer collider'lar ile çakışıyor.

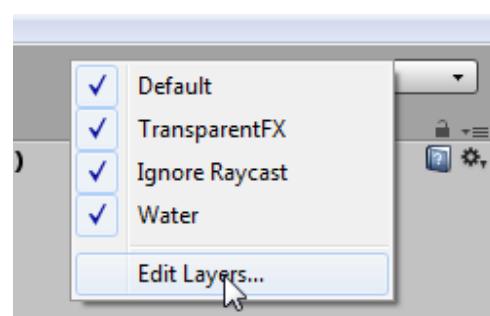
Ragdoll collider'larının Character Controller'ı nükleyle çakışmamasını Unity'e söylemeliyiz. Bunu yapmak için ragdoll collider'larını başka bir *Layer*'a (katman) taşıyabiliriz.

Yeni Bir Layer Oluşturmak

Unity'nin sağ üst kısmındaki Layer butonuna basın.

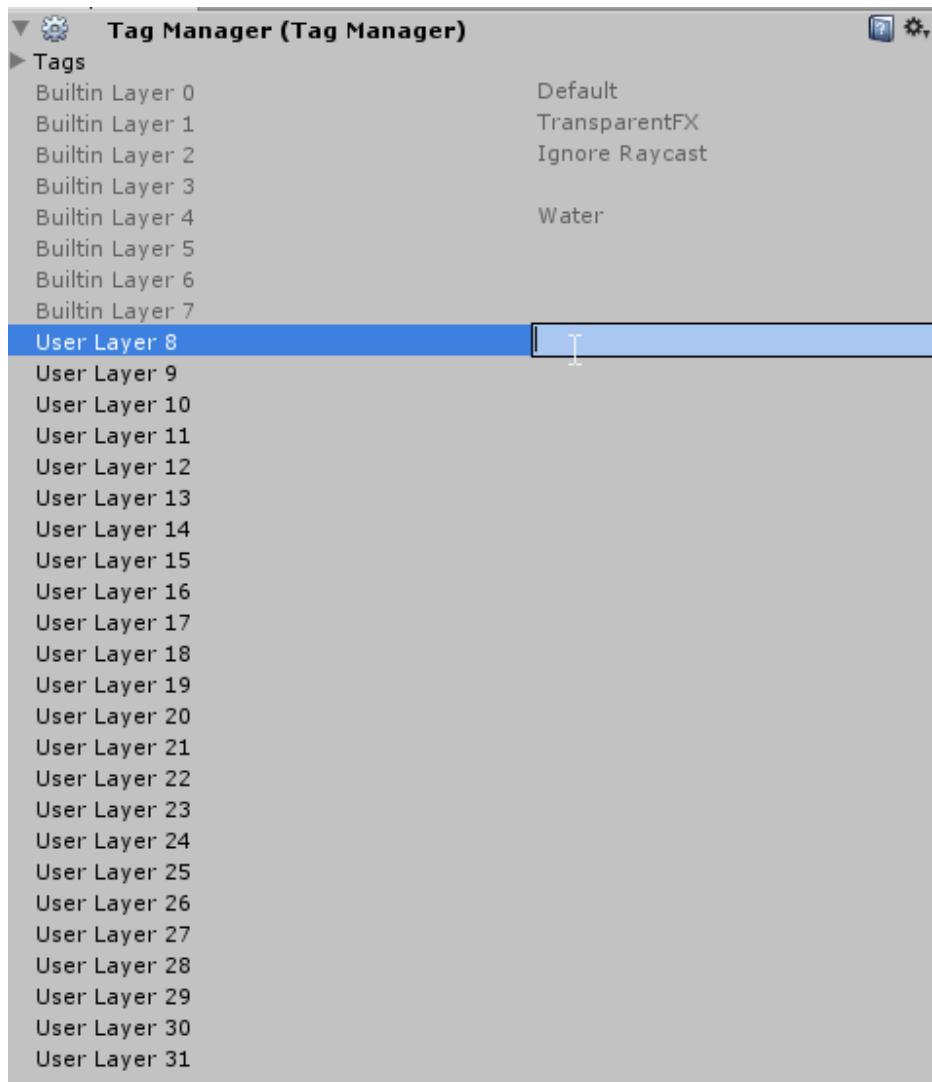


Gelen menüden "Edit Layers..."ı seçin:

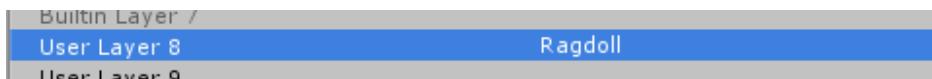


Inspector'da Layer Editor açılacak. Toplama 32 layer'ımız var. İlk 8'i Unity tarafından korunmaktadır ve değerleri kesinlikle değiştirilemez.

Kendi layer'ımızı "User Layer 8"de oluşturacağız. Bu layer'in yanındaki kutucuğa tıklayın.

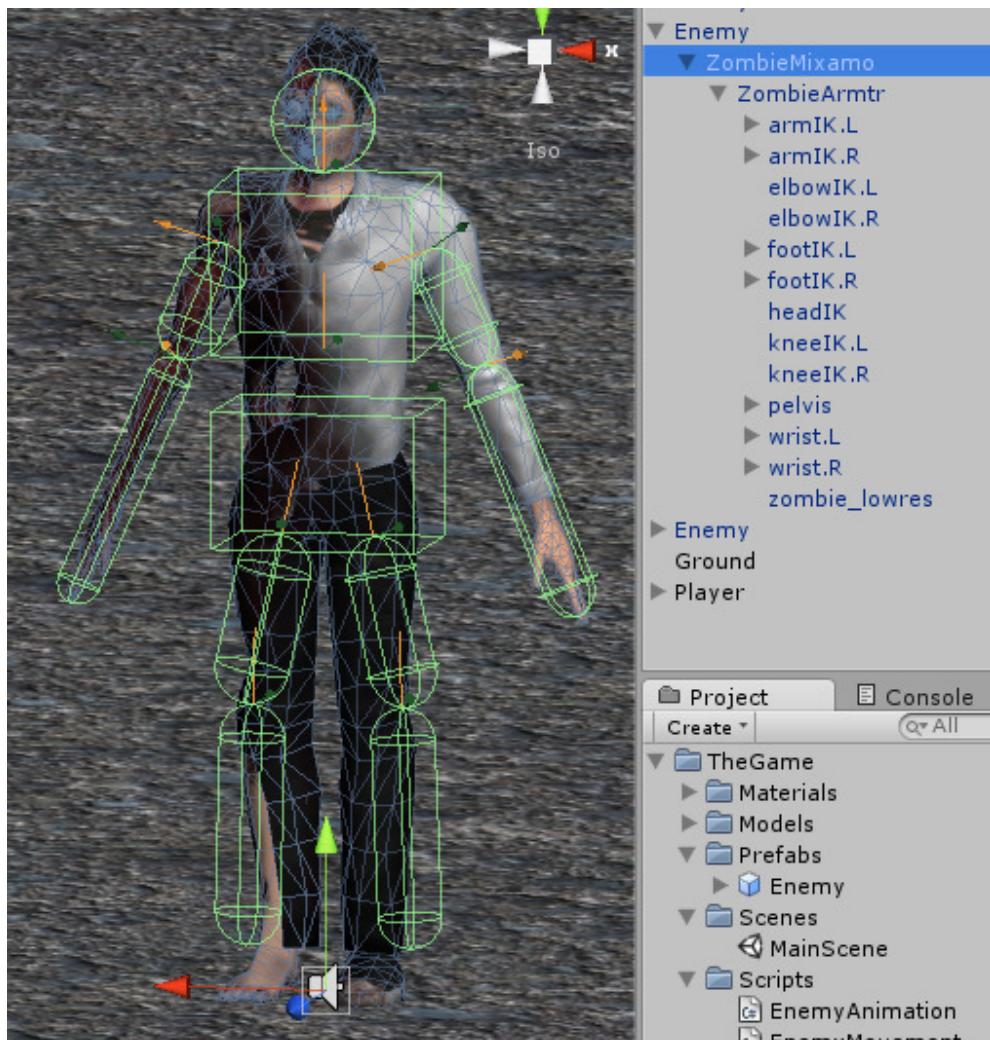


Layer'a isim olarak "Ragdoll" verin:

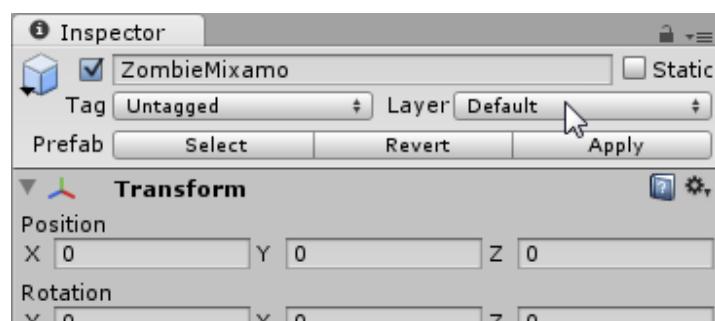


Objeleri Başka Bir Layer'a Taşımak

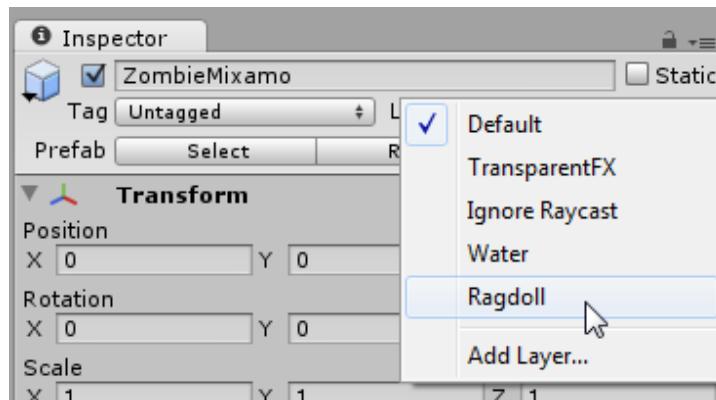
ZombieMixamo objesini seçin.



Inspector'daki "Layer" kısmına tıklayın:



Listeden “Ragdoll” layer'ını seçin.



Bir uyarı kutusu çıkacak ve ZombieMixamo'nun child objelerinin Layer'ını da "Ragdoll" olarak değiştirmeyi isteyip istemediğimiz soracak. Evet istiyoruz, böylece ragdoll'daki collider'ların layer'i değişmiş olacak. Bunun için "Yes, change children" seçeneğini seçin.



Tüm child objelerin de layer'ı değişmiş olmalı.



Kötü haber şu ki Character Controller da farklı bir layer'da yer almalı. Bunun için tekrar Layer Editor'ü açın.

“User Layer 9”a “Character” ismini verin.

Enemy objesini seçin. Character Controller componenti bu objedeydi hatırlarsanız.

Objenin layer'ını “Character” yapın ama bu sefer “No, this object only” seçeneğini seçin. Böylece sadece Enemy'nin layer'ı değişecek, child objelerin layer'ı aynı kalacak (“Ragdoll”).

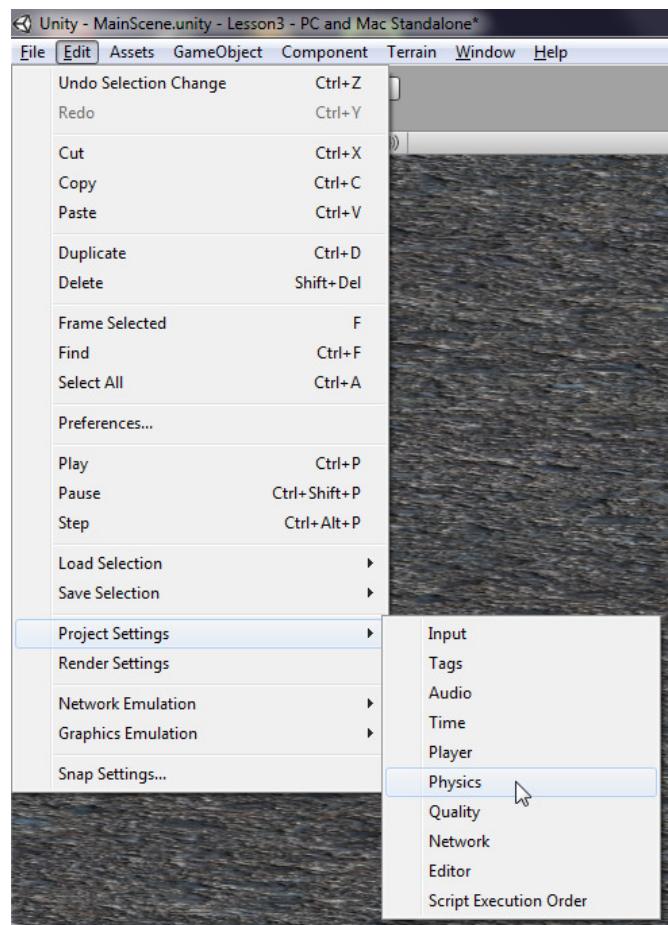
Artık Character Controller componenti “Character” layer'ında yer alırken ragdoll collider'ları da “Ragdoll” layer'ında yer alıyor.

Şimdi Inspector'dan “Apply” tuşuna basın ve yaptığınız tüm bu değişikliklerin sahnedeki diğer zombilere de otomatik olarak aktarıldığına şahit olun.

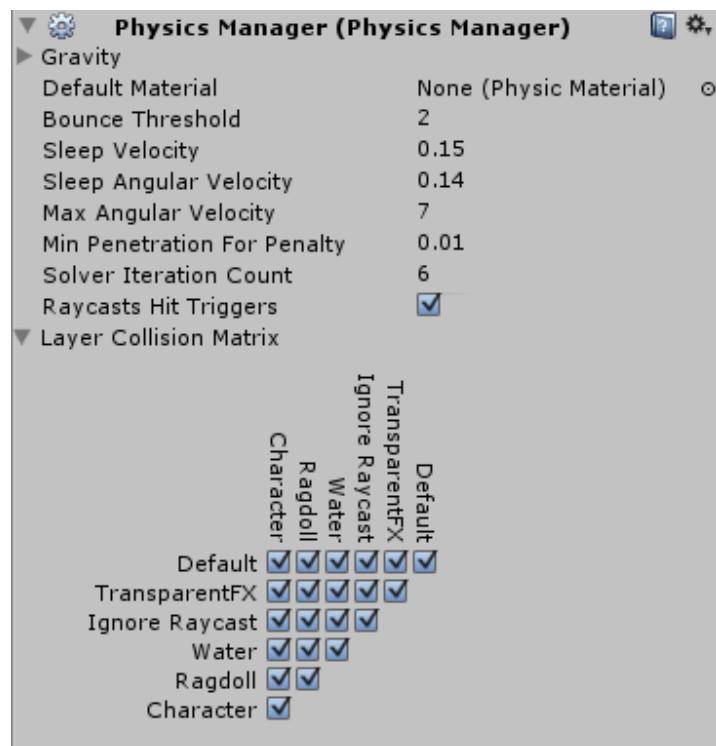
Bir Layer'daki Collider'ların Başka Bir Layer'daki İle Etkileşime Girmesini Önlemek

Şimdi “Character” layer'ındaki collider'ların “Ragdoll” layer'ındakilerle teması girmesini önleyelim.

Edit > Project Settings > Physics yolunu izleyin.

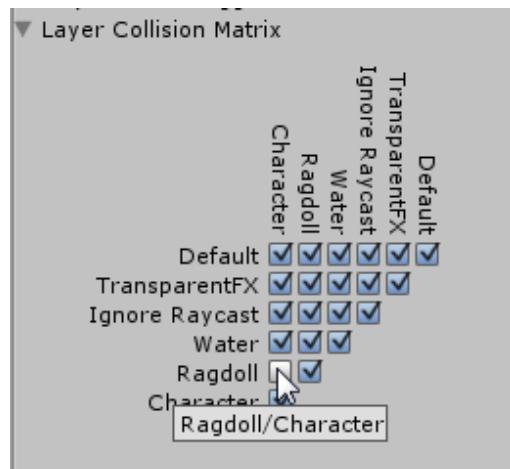


Karşınıza, üzerinde oynama yapabileceğiniz fizik değerleri gelecek. Bunların arasında, iki layer'ın etkileşimi önlemek için kullanılan bir matrix var: *Layer Collision Matrix*.



Burada çeşitli işaretli kutucuklar var. Her satır ve her sütun bir layer'ı temsil ediyor. İşaretli kutucuklar ise o iki layer'in etkileşime girip girmedğini belirliyor.

"Ragdoll" ile "Character" layer'larının kesiştiği kutucuğu bulun ve üzerindeki işaretü kaldırın:



Artık "Ragdoll" ve "Character" layer'larındaki collider'lar birbirlerinin işlerine karışmayacak!

Kaydetmeyi Unutmayın

Projeyi kaydetmek için şu an çok doğru bir an.

Ragdoll'u Devre Dışı Bırakmak

Oyunu test edince zombilerin hâlâ garip davranışını göreceksiniz.

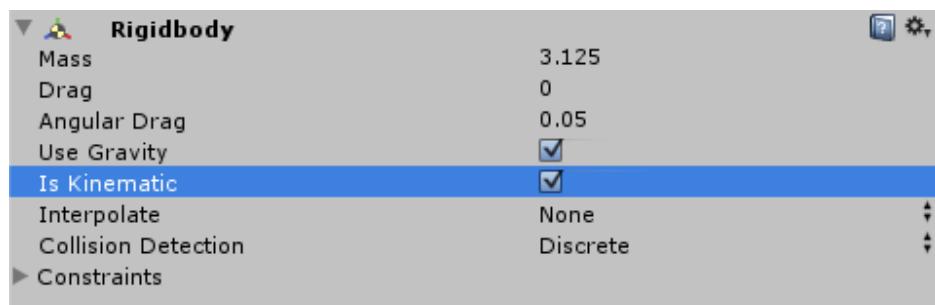


Çünkü şu anda zombilerin yürüme animasyonu ile Ragdoll'a fizik motorunun uyguladığı güçler birbiriyile savaş halinde.

Ragdoll sistemini sadece zombi öldüğünde devreye sokmamız gereklidir. Yani zombi ancak öldüğü zaman ragdoll'un etkisiyle yere yıgilmalıdır. O zamana kadar 3D modelin kontrolü yürüme animasyonunun elinde olacak.

“pelvis” objesini seçerseniz bir Rigidbody component'ine sahip olduğunu göreceksiniz. Sadece bu değil; zombideki tüm ragdoll bileşenlerinin birer Rigidbody'si var.

Ragdoll'u devre dışı bırakmanın bir yolu tüm bileşenlerdeki Rigidbody component'lerinin “Is Kinematic” seçeneğini devreye sokmaktır.



Tüm bileşenlerde Is Kinematic'i elle devreye sokup elle devreden çıkarmak akıl kârı değil. Bu iş için bir script kullanacağız.

“Ragdoll” adında bir C# scripti oluşturun, scripti bir tane Enemy objesine verin ve kodu şu şekilde değiştirin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Ragdoll : MonoBehaviour
05 {
06     void DisableRagdoll()
07     {
08         Rigidbody[] allRigidbodies = GetComponentsInChildren< Rigidbody >();
09
10         foreach (Rigidbody r in allRigidbodies)
11         {
12             r.isKinematic = true;
13         }
14     }
15 }
16

```

8. satırda kodun atandığı objenin tüm child objelerinde yer alan Rigidbody component'lerini bir array'de depoluyoruz.

10. satırda array'de yer alan tüm Rigidbody'lerin üzerinden tek tek geçiyor ve onların isKinematic değerini true yapıyoruz. Böylece o Rigidbody'e fizik motoru müdahale etmiyor.

Şimdi Start fonksiyonunda DisableRagdoll fonksiyonunu çağırarak zombinin ilk başta ragdoll'unun kapalı olmasını sağlayalım:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Ragdoll : MonoBehaviour
05 {
06     void Start()
07     {
08         DisableRagdoll();
09     }
10
11     void DisableRagdoll()
12     {
13         Rigidbody[] allRigidbodies = GetComponentsInChildren< Rigidbody >();
14
15         foreach (Rigidbody r in allRigidbodies)
16         {
17             r.isKinematic = true;
18         }
19     }
20 }
21

```

Oyunu çalıştırın. Scripti attığınız zombi garip hareket etmeyi kesecek ve normal yürekleme animasyonu ile sizi takip edecek. Zombiyi öldürdüğünzde ise zombi yine *puf* diye yok olacak. Bu yok olma olayını çözelim ve zombinin yok olmak yerine yere yığılmasını sağlayalım. Yapmamız gereken tek şey zombinin ragdoll'unu tekrar aktifleştirmek, bir başka deyişle ragdoll bileşenlerinin Rigidbody'lerindeki Is Kinematic seçeneğini kapatarak onları tekrar işlevselleştirmek.

Bu iş için EnableRagdoll fonksiyonu ekleyelim:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Ragdoll : MonoBehaviour
05 {
06     void Start()
07     {
08         DisableRagdoll();
09     }
10
11     void DisableRagdoll()
12     {
13         Rigidbody[] allRigidbodies = GetComponentsInChildren< Rigidbody >();
14
15         foreach (Rigidbody r in allRigidbodies)
16         {
17             r.isKinematic = true;
18         }
19     }
20
21     void EnableRagdoll()
22     {
23         Rigidbody[] allRigidbodies = GetComponentsInChildren< Rigidbody >();
24
25         foreach (Rigidbody r in allRigidbodies)
26         {
27             r.isKinematic = false;
28         }
29     }
30 }
31

```

Kod DisableRagdoll ile aynı. Tek bir fark var o da Is Kinematic'i bu sefer açmak yerine kapatıyoruz. EnableRagdoll fonksiyonunu dışarıdan çalıştırılabilme için bir fonksiyon tanımlayalım:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Ragdoll : MonoBehaviour
05 {
06     void Start()
07     {
08         DisableRagdoll();
09     }
10
11     void DisableRagdoll()
12     {
13         Rigidbody[] allRigidbodies = GetComponentsInChildren< Rigidbody >();
14
15         foreach (Rigidbody r in allRigidbodies)
16         {
17             r.isKinematic = true;
18         }
19     }
20
21     void EnableRagdoll()

```

```

22     {
23         Rigidbody[] allRigidbodies = GetComponentsInChildren< Rigidbody >();
24
25         foreach (Rigidbody r in allRigidbodies)
26         {
27             r.isKinematic = false;
28         }
29     }
30
31     public void OnDeath()
32     {
33         EnableRagdoll();
34     }
35 }
36

```

Bu scriptteki OnDeath fonksiyonunu çağrıdığımız vakit ragdoll devreye girecek. Şimdi de OnDeath fonksiyonunu zombi ölünce çağrıma işlemini yapalım.

Bunun için Health script'ini açıp güncelleyelim:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     void Start()
12     {
13         _currentHealth = _maximumHealth;
14     }
15
16     public void Damage(int damageValue)
17     {
18         _currentHealth -= damageValue;
19
20         if (_currentHealth <= 0)
21         {
22             Animation a = GetComponentInChildren< Animation >();
23             a.Stop();
24
25             Ragdoll r = GetComponent< Ragdoll >();
26             if (r != null)
27             {
28                 r.OnDeath();
29             }
30         }
31     }
32 }
33

```

Objeyi *puf* diye sahneden silmek yerine ilk önce objedeki yürüme animasyonunu durduruyoruz "a.Stop();". Böylece ölüp de devreye ragdoll girdiğinde yürüme animasyonu ragdoll'un işine karışmayacak.

Animasyonu durdurduktan sonra Ragdoll scriptindeki OnDeath fonksiyonunu çalıştırıyoruz. Böylece ragdoll aktifleştiriliyor.

Bu kadar! Ragdoll scriptini attığınız zombi objesinin Inspector'undan "Apply" butonuna basarak değişiklikleri prefab'a ve diğer klonlara da uygulayın. Sonrasında oyunu çalıştırın. Bir zombi öldürdüğünüzde zombi gerçekçi bir şekilde yere yiğilmeli.

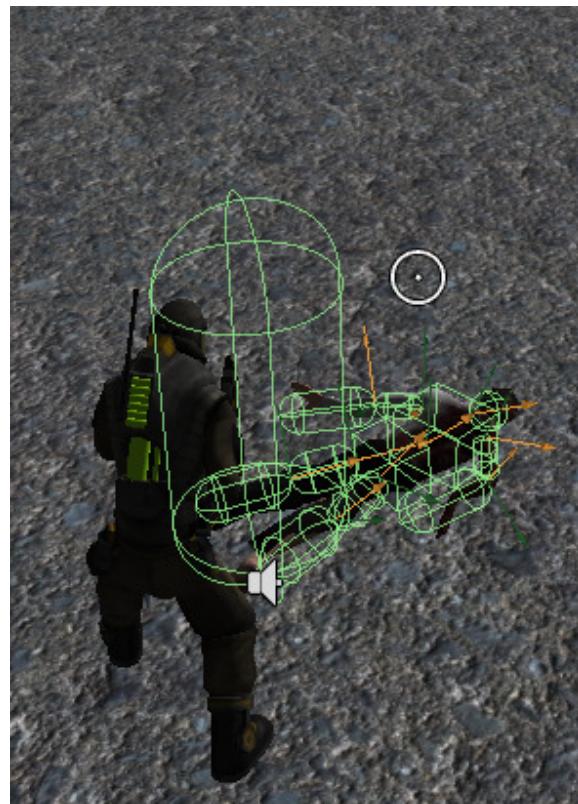
Oyunu test ederken komik bir hata ile karşılaşmaktayız: zombi ölüp yere yiğilince hâlâ bizi takip etmeyi bırakmıyor!

Çünkü zombideki EnemyMovement scripti hâlâ iş yapıyor ve zombiya bize doğru hareket etmeye zorluyor. Zombi öldüğünde bu scripti objeden atalım ve böylece bu sorunu da aşalım:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     void Start()
12     {
13         _currentHealth = _maximumHealth;
14     }
15
16     public void Damage(int damageValue)
17     {
18         _currentHealth -= damageValue;
19
20         if (_currentHealth <= 0)
21         {
22             Animation a = GetComponentInChildren<Animation>();
23             a.Stop();
24
25             Destroy(GetComponent<EnemyMovement>());
26
27             Ragdoll r = GetComponent<Ragdoll>();
28             if (r != null)
29             {
30                 r.OnDeath();
31             }
32         }
33     }
34 }
35
```

GetComponent fonksiyonu ile EnemyMovement scriptine erişiyoruz ve bu scripti Destroy fonksiyonuna parametre olarak vererek scriptin objeden atılmasını (objeden silinmesini) sağlıyoruz.

Zombiyi öldürdükten sonra üzerinden geçmek isterseniz "görünmez" bir duvarın yolunuzu tıkadığını göreceksiniz. Çünkü zombinin Character Controller component'ının collider'ı hâlâ oyun alanında duruyor:



Tıpkı EnemyMovement'a yaptığımız gibi, CharacterController component'ini de zombi öldüğünde objeden silersek sorun kalmayacak:

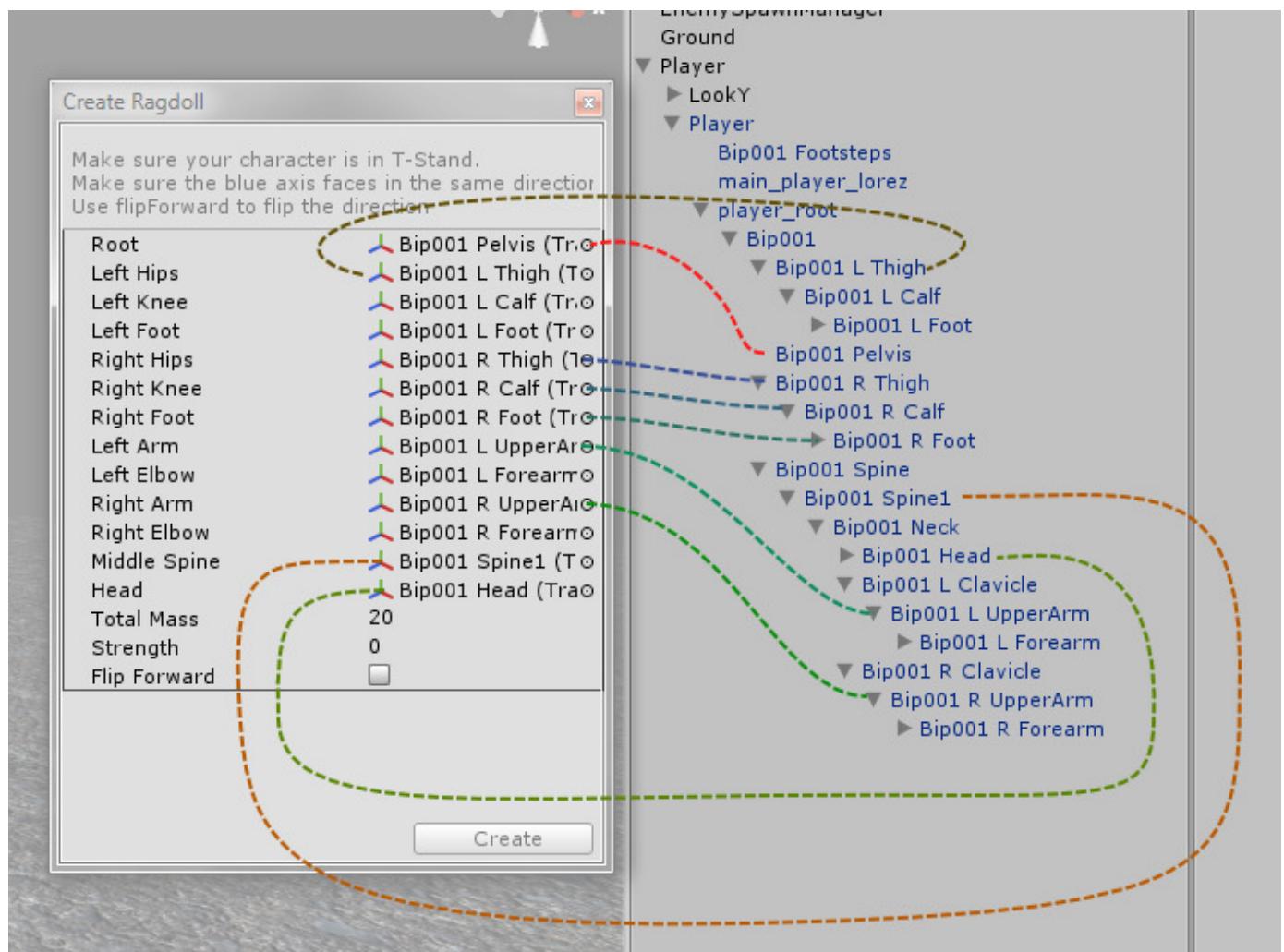
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     void Start()
12     {
13         _currentHealth = _maximumHealth;
14     }
15
16     public void Damage(int damageValue)
17     {
18         _currentHealth -= damageValue;
19
20         if (_currentHealth <= 0)
21         {
22             Animation a = GetComponentInChildren<Animation>();
23             a.Stop();
24
25             Destroy(GetComponent<EnemyMovement>());
26             Destroy(GetComponent<CharacterController>());
27
28             Ragdoll r = GetComponent<Ragdoll>();
29             if (r != null)
30             {
31                 r.OnDeath();
32             }
33         }
34     }
35 }
36
```

Player Objesine Ragdoll Vermek

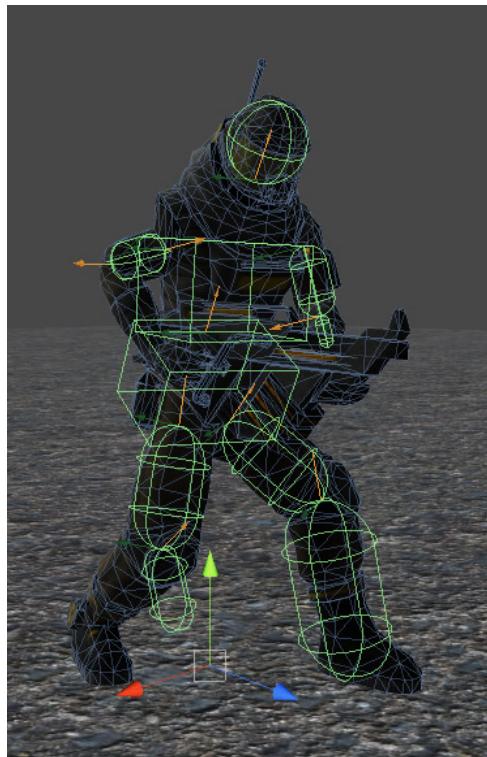
Zombilerle işimiz şu an bitti gibi. Ragdoll sistemini bu sefer player'a verelim ve biz de öldüğümüz zaman tipki zombiler gibi yere yiğilalım.

Player objesini seçip yine `GameObject > Create Other > Ragdoll...` yolunu izleyin.

Resmi referans alarak gerekli yerleri doldurun:

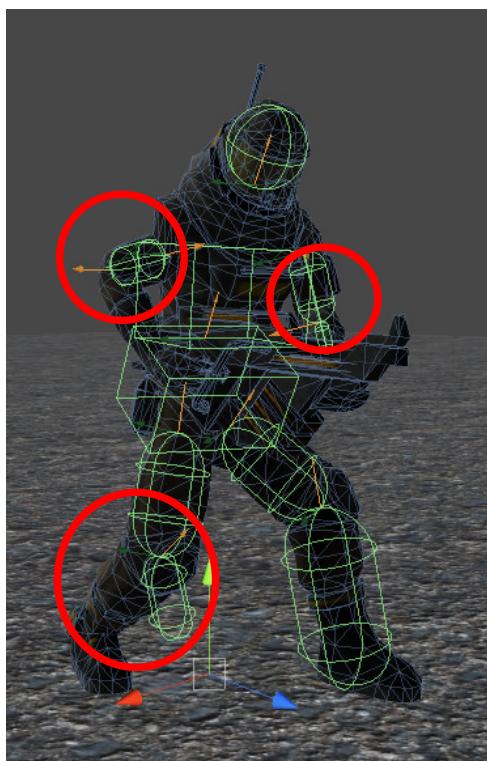


İşlem tamamlandığında karakterin görünümü şöyle olacak:

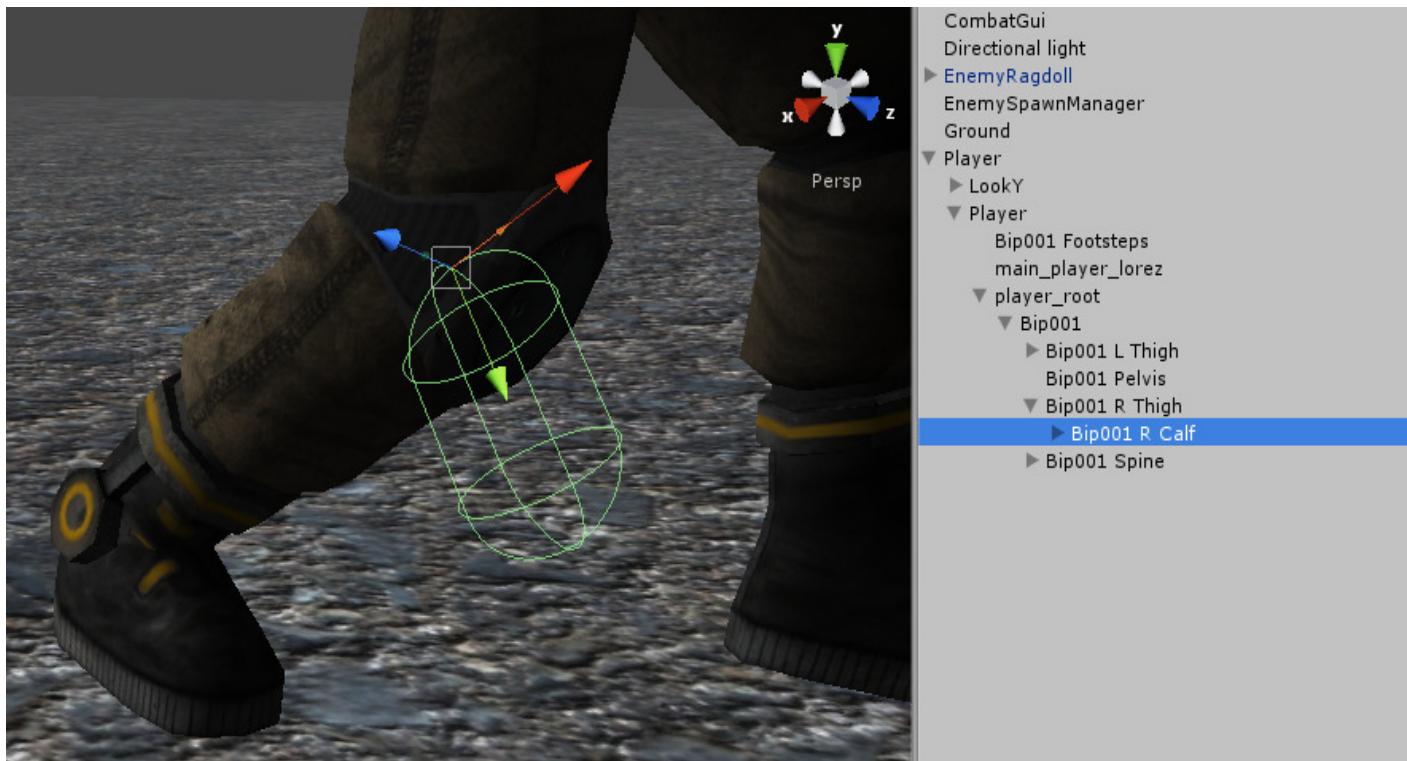


Kimi kemiklerin collider'larının düzgün oluşturulmadığını göreceksiniz. Onları düzeltmeliyiz.

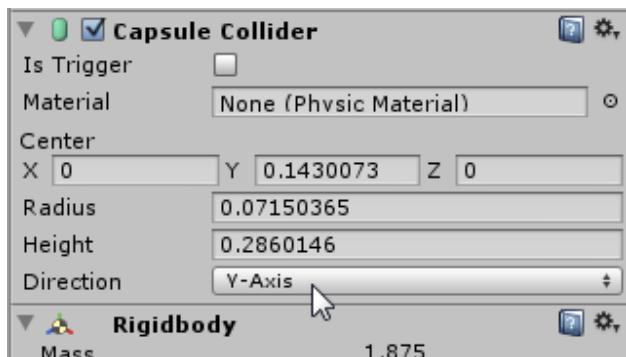
ÇEVİRİMEN EKLEMESİ: Büyük olasılıkla siz collider'ları resimdeki gibi net görmüyorsunuz çünkü bazı collider'lar objenin içinde kalmıştır. Bunu çözmek ve resimdeki görüntüyü elde etmek için Scene panelinin üstündeki "Gizmos" butonuna tıklayıp "3D Gizmos" seçeneğini kapatın.



Baldırda yer alan kaval kemiğiyle başlayalım. "Bip001 R Calf" objesini seçin:



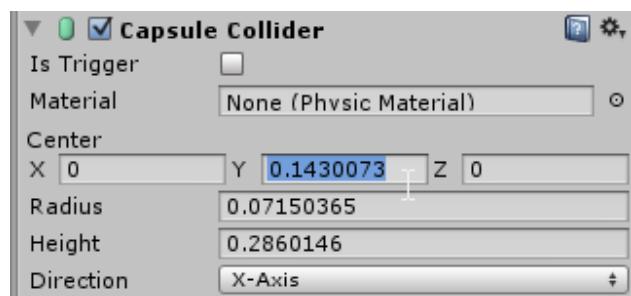
Kapsül şeklindeki collider'ımız ters yöne bakıyor. Inspector'a göz atalım:



Direction olarak Y eksenini verilmiştir. Bunu X eksenine (X-Axis) değiştirin.



Artık kemik doğru yöne bakıyor. Tek yapmamız gereken kemiği yerine oturtmak. Inspector'da Center adında bir vektör var:

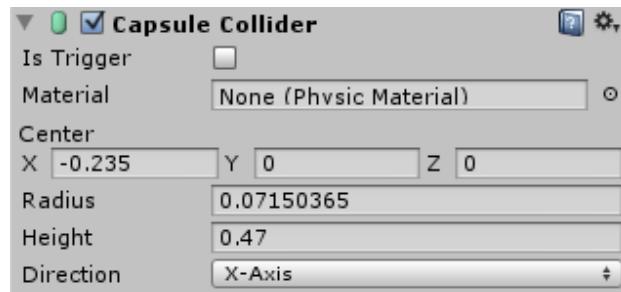


Bunun değerini resetlersek (X, Y ve Z'yi sıfırlarsak) kemik yerine oturacak:

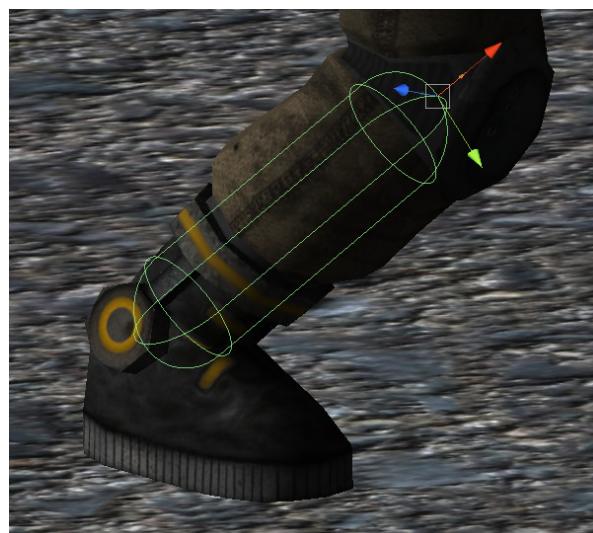


Ama hâlâ kemiğin boyu kısa ve bu sefer kemik biraz yukarıya kayık vaziyette.

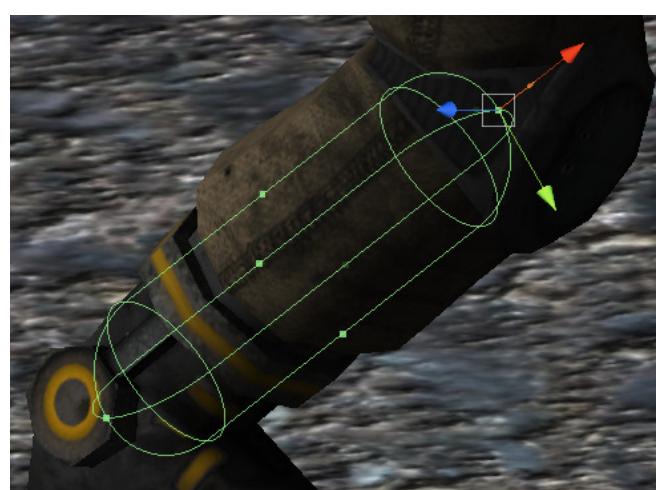
Inspector'da Height'ı 0.47 ve Center X'i -0.235 yapın:



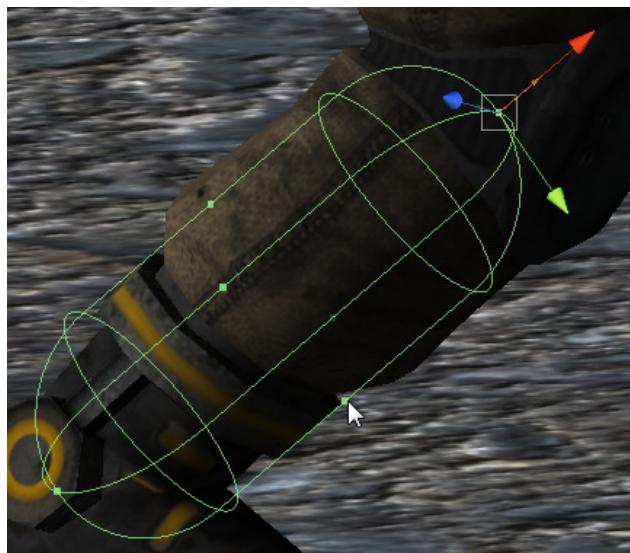
Collider'ı aşağı kaydırıp biraz büyütük ve artık collider tam yerine oturdu!



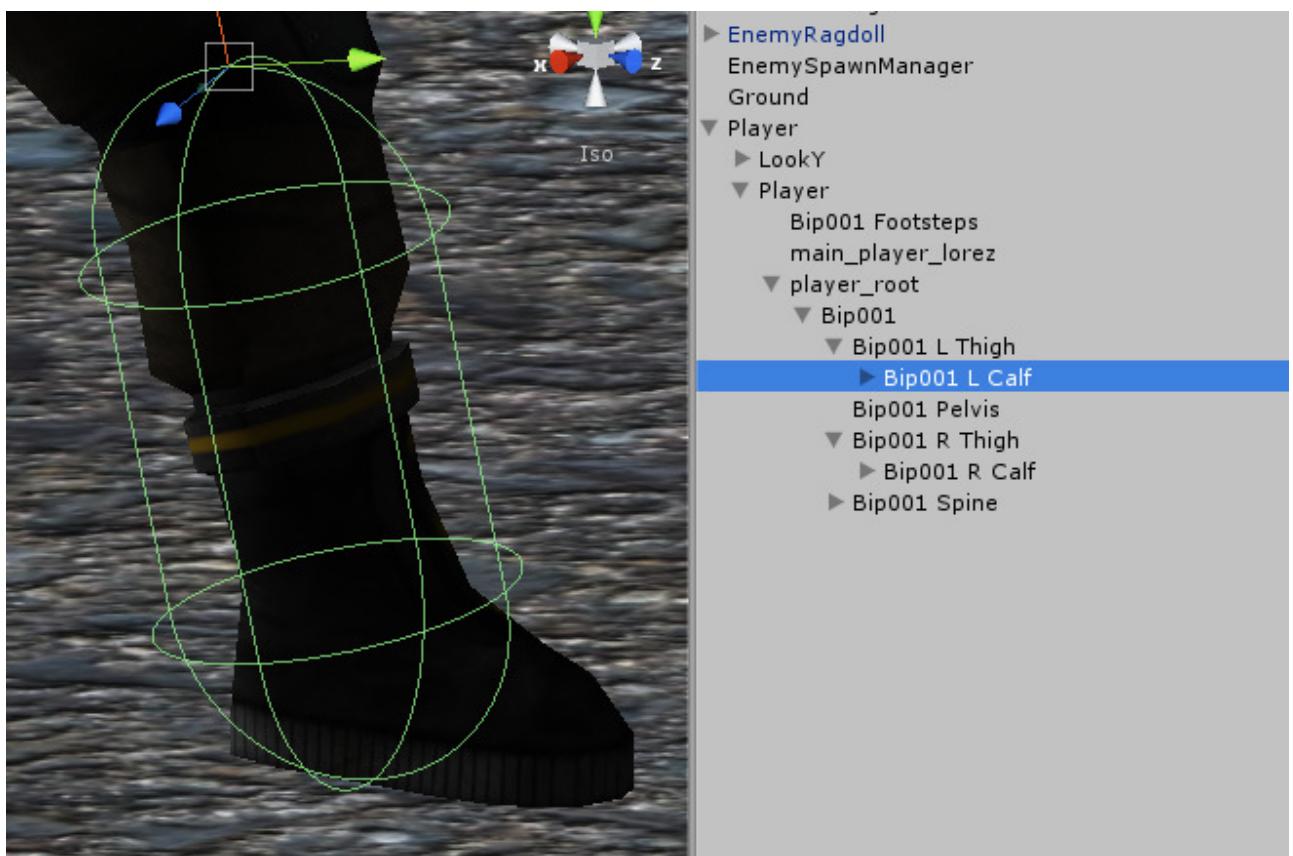
Tabi kapsülü biraz daha genişletip bacağın genişliğine eşdeğer yaparsak daha güzel olacak.
Shift' basılı tutunca kapsülün etrafında noktalar belirecek.



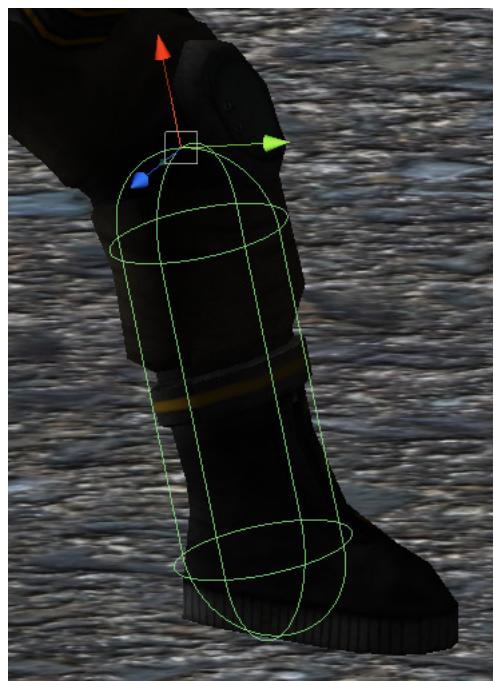
Kenarlardaki noktalar ile kapsül genişletilebilmekte. Üç noktalardaki iki noktası vasıtası ile de kapsülün yüksekliği değişmekte. Kapsülü çevreleyen noktalardan birini tutarak kapsülü genişletin (yüksekliği artırmayın, genişletin sadece!). Kapsül hemen hemen bacağın genişliği kadar geniş olsun:



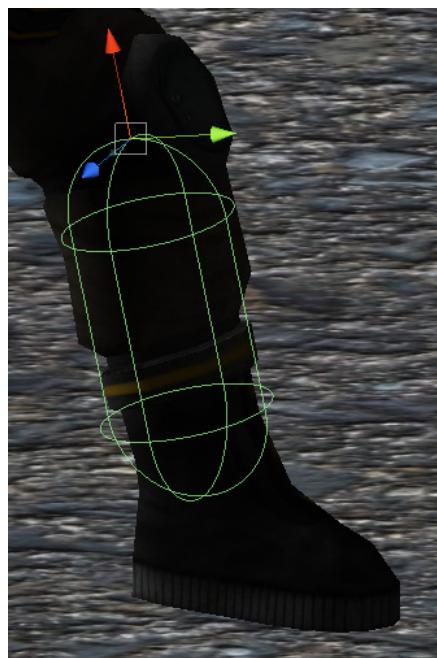
Şimdi de sol kaval kemiğini (Bip001 L Calf) seçin:



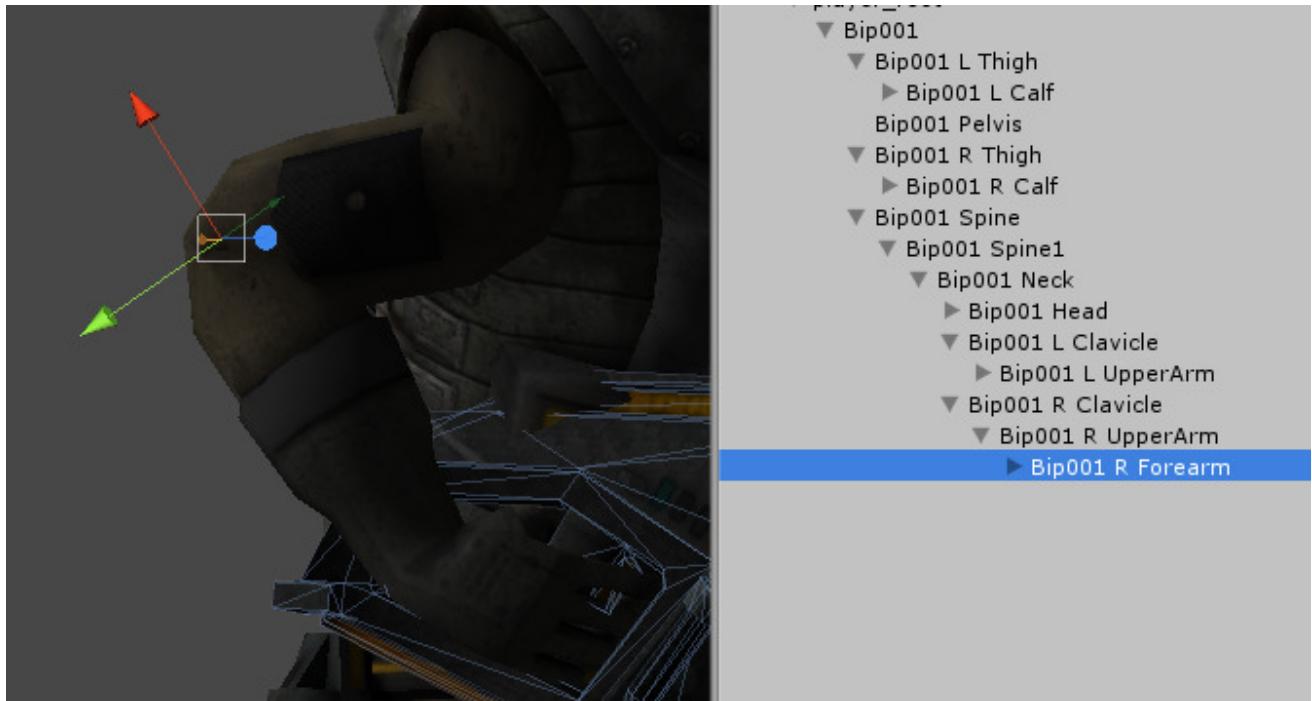
Kemiğin yarıçapı çok büyük. Shift tuşuna basarak kemiğin yarıçapını, bacağın etrafını saracak kadar azaltın:



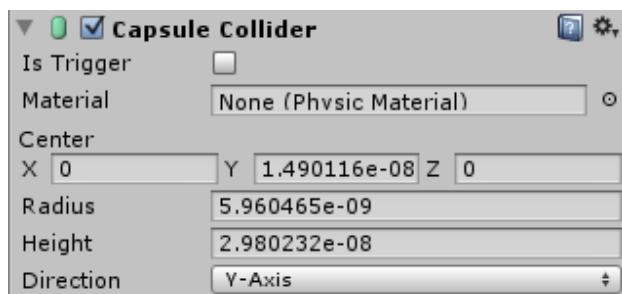
Kemiğin yüksekliği de fazla duruyor. Bence Height 0.47 ve Center X de -0.235 iken kemik çok iyi duruyor.



Sıra geldi sağ dirsek kemiğine. "Bip001 R Forearm" objesini seçin.

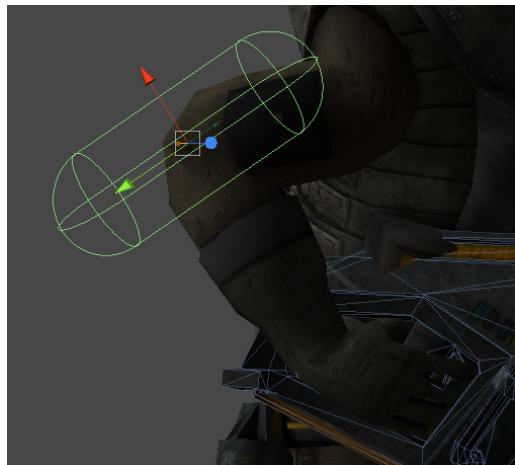


Sanki kemik (collider) yerinde yok gibi bir görüntü var ama Inspector'da collider'ı görebilmekteyiz:

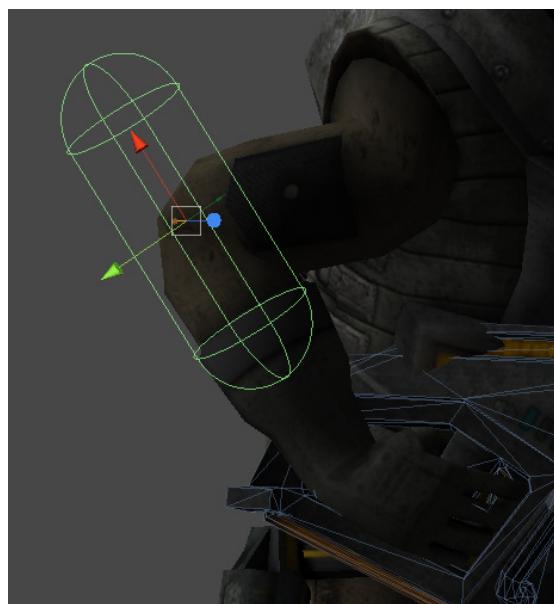


Ne yazık ki Ragdoll Wizard kemiğin yarıçapını ve yüksekliğini o kadar küçük vermiş ki mikroskop kullanmadan kemiği görmemiz mümkün değil.

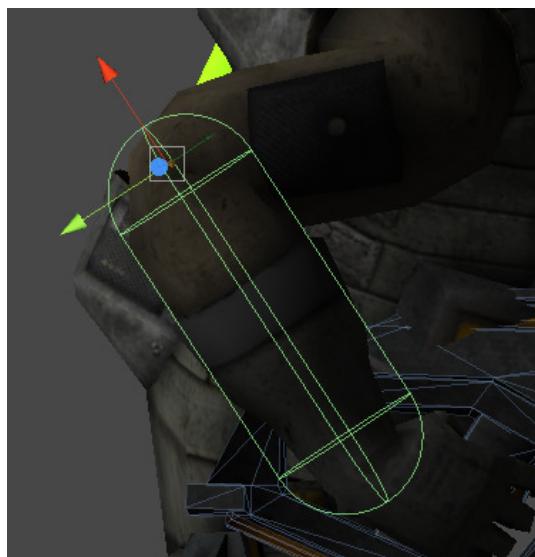
Radius'u 0.07 ve Height'ı 0.4 yapın.



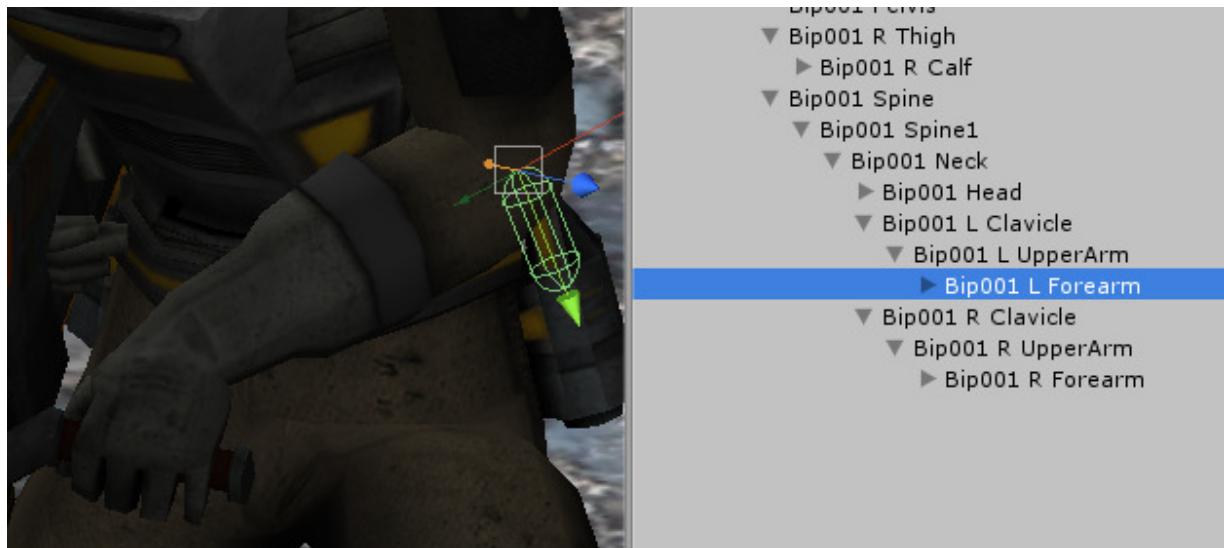
Daha iyi. Şimdi Direction'ı X-Axis olarak değiştirin:



Son olarak da Center X değerini -0.16 yapın.



Geldik sol dirsek kemiğine. Bu kemiğin ismi ise “Bip001 L Forearm”:



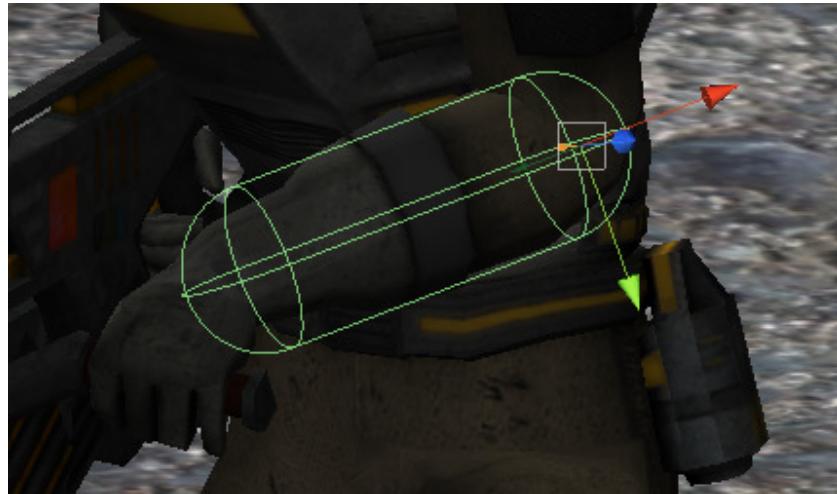
Değerleri şu şekilde değiştirin:

Center: (-0.16, 0, 0)

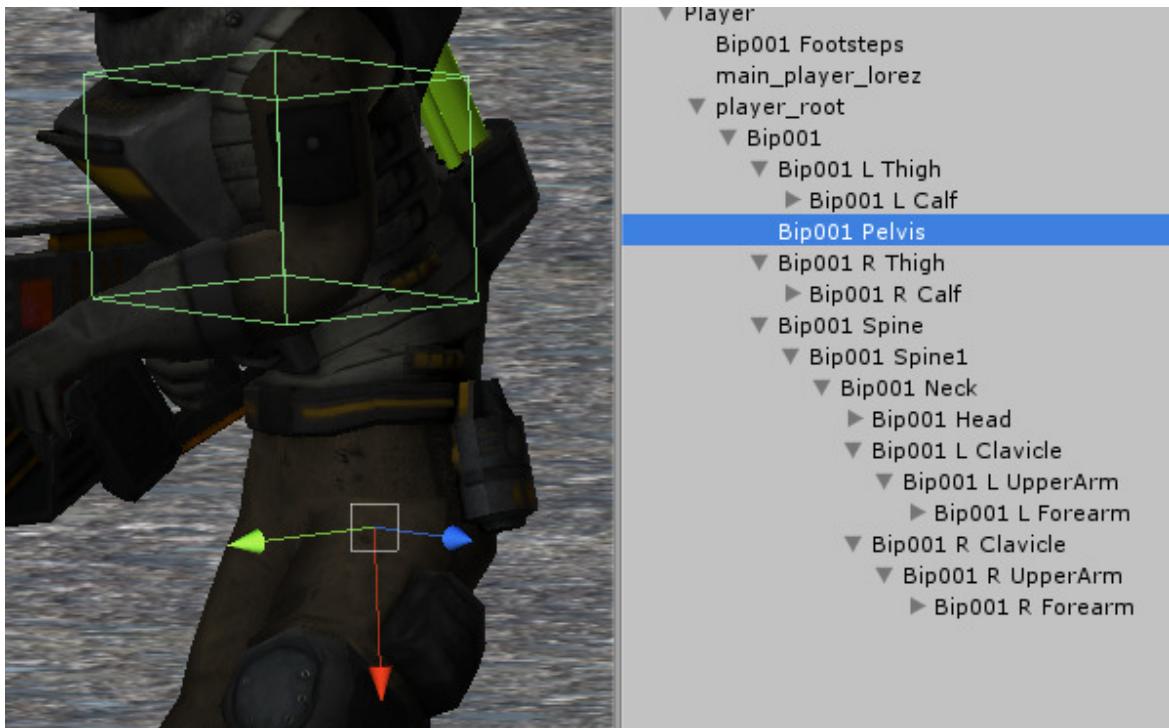
Radius: 0.07

Height: 0.4

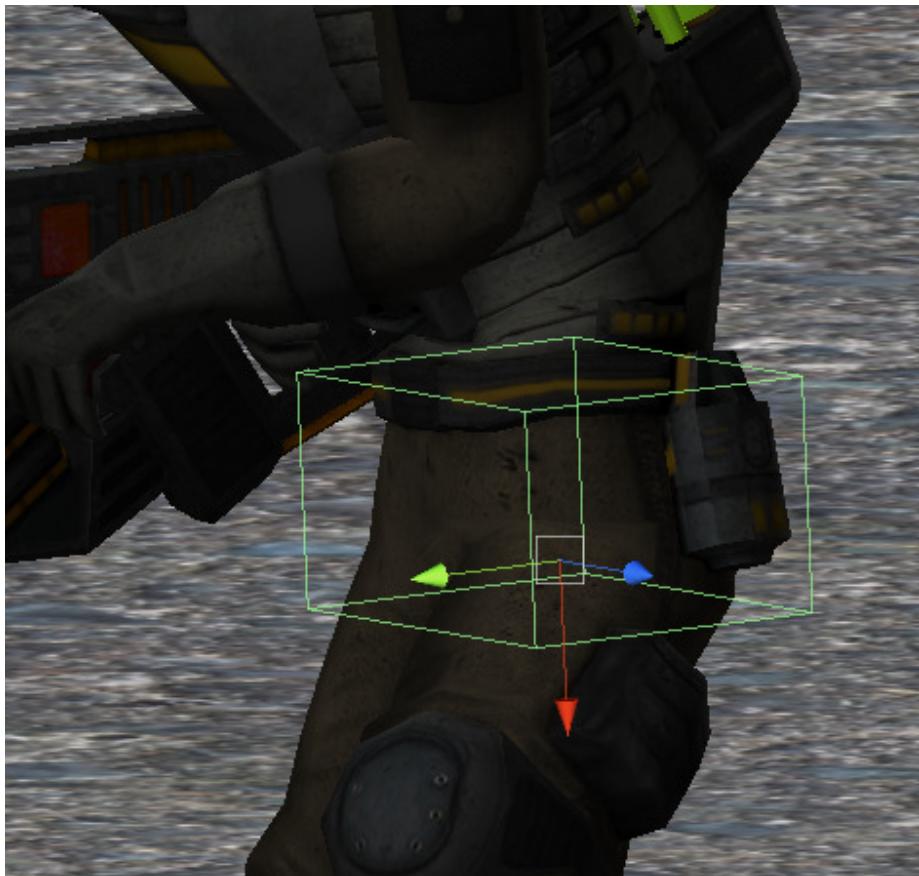
Direction: X-Axis



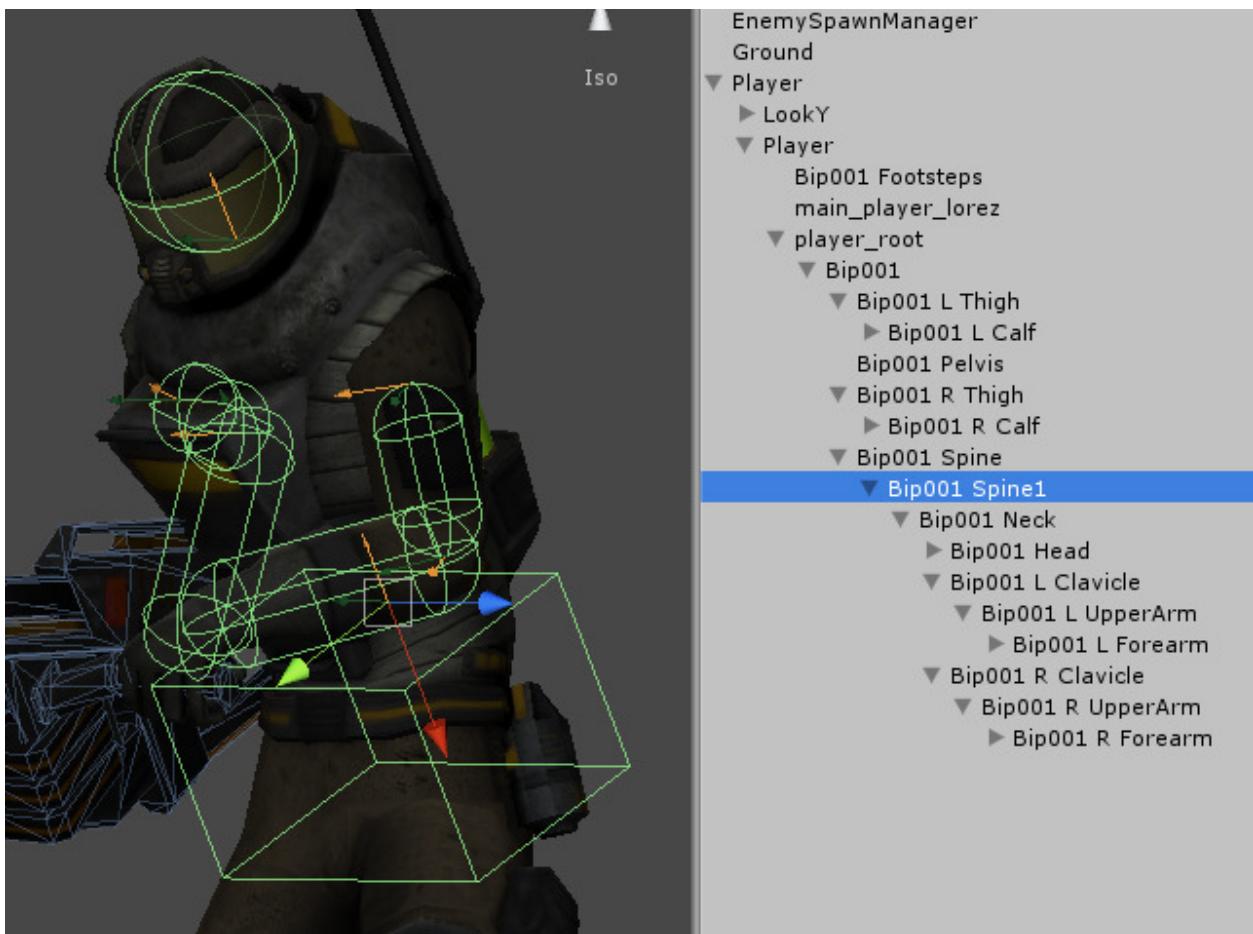
Hemen hemen bitti. Şimdi “Bip001 Pelvis”i seçin:



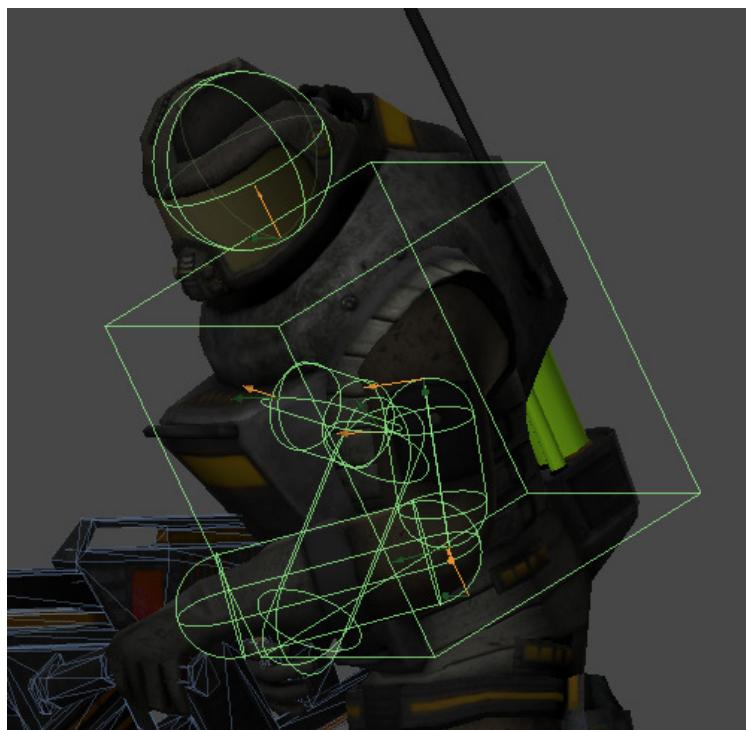
Collider göbek hizasında olmalı. Shift tuşu basılıken noktalardan çekerek bunu ayarlayın.



Son olarak “Bip001 Spine1”ı seçin:



Kutu şeklinde collider'ın göğüs hizasında olup gövdeyi kaplamasını sağlayın:



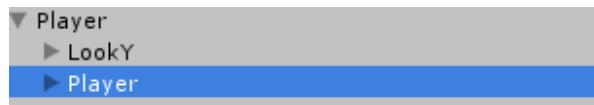
Artık karakterin ragdoll'u katbekat daha güzel duruyor:



Player'ın Ragdoll'una Son Rötuşları Yapmak

Zombinin ragdoll'unda olduğu gibi karakterimizin ragdoll'undaki collider'lar da “Ragdoll” layer'ında olmalı ve ardından Ragdoll scriptini Player objesine vermeliyiz.

Child vaziyettedeki Player objesini seçin (parent olanı değil!):



Objenin layer'ını “Ragdoll” olarak değiştirin ve gelen soruya Yes diye cevap vererek child objelerin de layer'larının değişmesini sağlayın.

Şimdi ise parent olan Player objesini seçin ve layer'ını “Character” yapın. Gelen soruya No deyin ki sadece seçtiğimiz objenin layer'i değişsin, child'larının değil!

Parent olan Player objesi hâlâ seçiliyken ona Ragdoll scriptini atayın.

Player'ın Ragdoll'unu Test Etmek

Oyunun en başında player'ı öldürerek ragdoll'unu test etmeye ne dersiniz?

Player'a Health Scripti Vermek

Player'ı öldürebilmek için onun bir sağlığa sahip olması lazım. Zombilerde sağlık için Health scriptini kullanmıştık, ayınısını Player'da da kullanabiliriz.

O halde durmayın ve Health scriptini parent olan Player objesine verin.

Sonrasında PlayerMovement scriptini güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerMovement : MonoBehaviour
05 {
06     CharacterController _controller;
07
08     [SerializeField]
09     float _moveSpeed = 5.0f;
10
11     [SerializeField]
12     float _jumpSpeed = 5.0f;
13
14     [SerializeField]
15     float _gravity = 2.0f;
16
17     float _yVelocity = 0.0f;
18 }
```

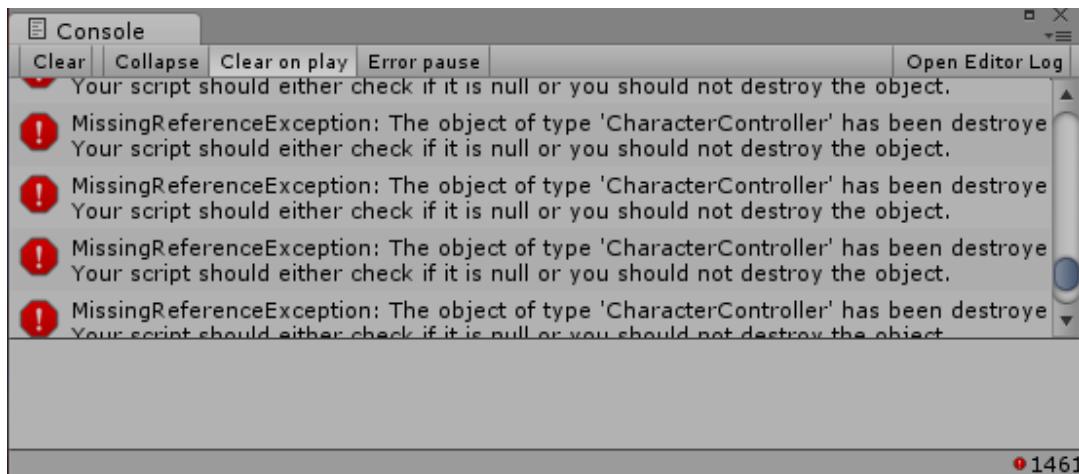
```

19 // Use this for initialization
20 void Start()
21 {
22     _controller = GetComponent<CharacterController>();
23     GetComponent<Health>().Damage(999);
24 }
25
26 // Update is called once per frame
27 void Update()
28 {
29     Vector3 direction = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
30     Vector3 velocity = direction * _moveSpeed;
31
32     if (_controller.isGrounded)
33     {
34         if (Input.GetButtonDown("Jump"))
35         {
36             _yVelocity = _jumpSpeed;
37         }
38     }
39     else
40     {
41         _yVelocity -= _gravity;
42     }
43
44     velocity.y = _yVelocity;
45
46     velocity = transform.TransformDirection(velocity);
47
48     _controller.Move(velocity * Time.deltaTime);
49 }
50 }
51

```

Artık oyunun başında Player'a çok yüksek bir hasar vererek onun yanında ölmeyi sağlıyoruz. Böylece ragdoll devreye girecek ve biz de onu test edebileceğiz.

Oyunu çalıştırın. Muhtemelen bir düzine hata alacaksınız.



Karakter ölünce onun CharacterController'unu silerek collider'inin artık aktif olmamasını sağlıyorduk hatırlarsanız. Ama bu işlemden sonra bile (karakter öldüğü halde) PlayerMovement hâlâ CharacterController'a erişmeye çalışıyor.

Aslına bakarsanız öldükten sonra karakteri hareket ettiremememiz lazım. Bu sebepten ötürü de PlayerMovement'a zaten ihtiyacımız yok ölünce. O halde karakterimiz ölünce onu silelim gitsin!

Health script'ini açıp şu değişikliği yapın:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     void Start()
12     {
13         _currentHealth = _maximumHealth;
14     }
15
16     public void Damage(int damageValue)
17     {
18         _currentHealth -= damageValue;
19
20         if (_currentHealth <= 0)
21         {
22             Animation a = GetComponentInChildren<Animation>();
23             a.Stop();
24
25             Destroy(GetComponent<PlayerMovement>());
26
27             Destroy(GetComponent<EnemyMovement>());
28             Destroy(GetComponent<CharacterController>());
29
30             Ragdoll r = GetComponent<Ragdoll>();
31             if (r != null)
32             {
33                 r.OnDeath();
34             }
35         }
36     }
37 }
38 }
```

Oyunu çalıştırınca artık hata almadığınızı göreceksiniz.

ÇEVİRMEN EKLEMESİ: Aslında hata alacaksınız çünkü biz karaktere animasyon verirken **Animation** değil **Animator** component'ini kullandık. Ama **Health** scripti **Player**'da yer almayan **Animation** component'ine erişmeye çalışıyoruz ve bu yüzden hata alıyoruz. Peki ne yapmalı? Eğer **Health** scriptinin atıldığı objenin tag'ı **Player** ise **Animator** component'ıyla işlem yapmalı, değilse (obje zombi ise) **Animation** component'ıyla işlem yapmalı. **Health** scriptinin 22. ve 23. satırlarını şöyle değiştirin (dersin ilerleyen kısımlarında **Health** scriptinin 22. ve 23. satırları eskisi gibi gözükmüyor, onları güncellemedim çünkü her birini tek tek güncellemek çok zahmetli birşey):

```

if( tag == "Player" )
{
    Animator a = GetComponentInChildren<Animator>();
    Destroy(a);
}
else
{
    Animation a = GetComponentInChildren<Animation>();
    a.Stop();
}

```

Son olarak da PlayerAnimation component'ini silelim çünkü bu component, aldığı input'a göre her frame'de objelerimize animasyon emirleri yağıdırıp duruyor. Kodu güncelleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     void Start()
12     {
13         _currentHealth = _maximumHealth;
14     }
15
16     public void Damage(int damageValue)
17     {
18         _currentHealth -= damageValue;
19
20         if (_currentHealth <= 0)
21         {
22             Animation a = GetComponentInChildren<Animation>();
23             a.Stop();
24
25             Destroy(GetComponent<PlayerMovement>());
26             Destroy(GetComponent<PlayerAnimation>());
27
28             Destroy(GetComponent<EnemyMovement>());
29             Destroy(GetComponent<CharacterController>());
30
31             Ragdoll r = GetComponent<Ragdoll>();
32             if (r != null)
33             {
34                 r.OnDeath();
35             }
36         }
37     }
38 }
39

```

Oyunu çalıştırın. En sonunda ragdoll'u çalıştırmayı başardık.

Ama o da ne! Ragdoll çalışmada karakter çok garip bir görünüm alıyor:

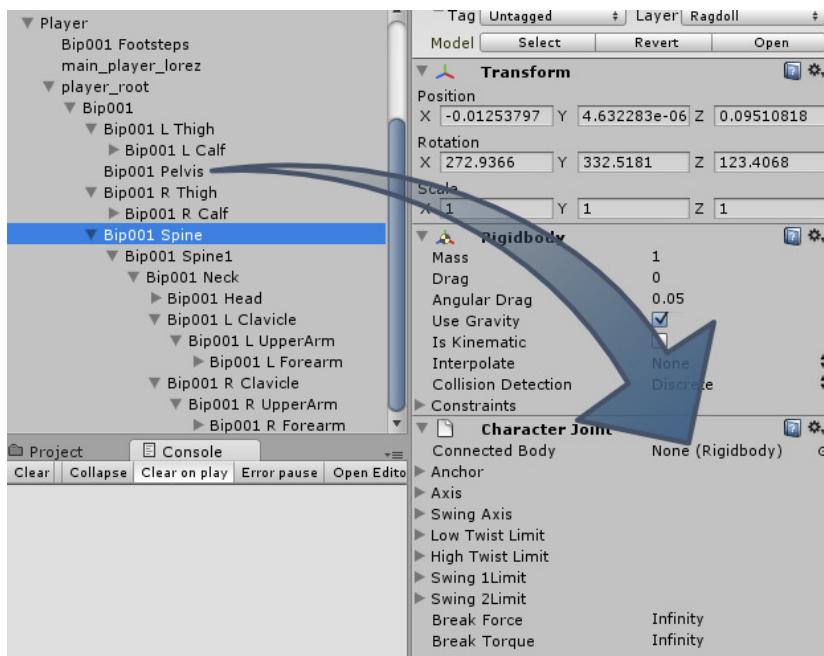


Ragdoll Wizard'ın yaptığı otomatik ayarlamalar bu model için düzgün olmamış. Ragdoll'un çalışması için tüm kemiklerin birer rigidbody'si olmalı. Ayrıca iki kemik arasında bir joint (eklem) yer almalı.

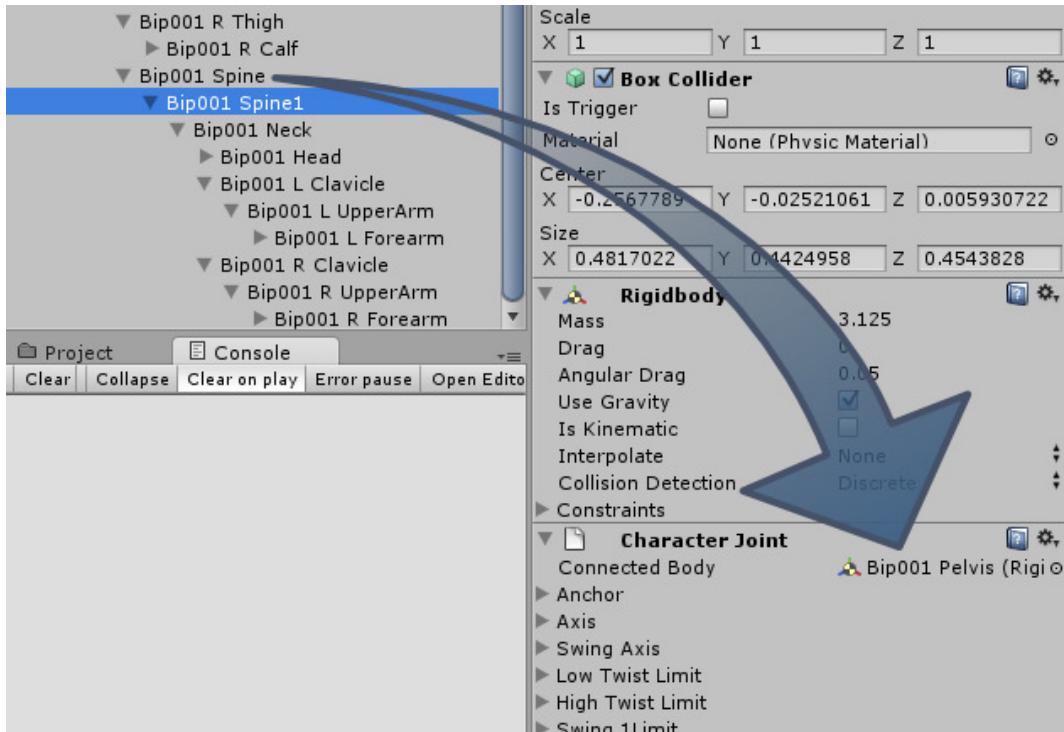
Player'daki "Bip001 Spine" objesini bulun (Kendisi "Bip001 Spine1" isimli objenin parent objesidir). Bu objenin bir rigidbody'e ihtiyacı var çünkü gövde ile göbek kısmındaki collider'ları birbirine bağlıyor.

"Bip001 Spine" seçili iken objeye bir Rigidbody ekleyin. Ayrıca (Component > Physics > Character Joint) yolunu izleyerek bir de Character Joint ekleyin.

Character Joint component'inde Connected Body adında bir değer var. Buraya "Bip001 Pelvis" objesini değer olarak verin:



Şimdi "Bip001 Spine1" objesini seçin ve Character Joint component'indeki Connected Body değerini "Bip001 Spine" olarak değiştirin:



Artık karakterin ragdoll'u sorunsuz çalışmalı.



Test işlemini başarıyla tamamladığımıza göre PlayerMovement'in Start fonksiyonundaki Damage kodunu silerek karakterin oyunun başında ölmesini engelleyin.

Özet Geçecek Olursak...

ÇEVİRMEN EKLEMESİ: Karakterin ragdoll'undaki en son ortaya çıkan sıkıntıları çözdüğümüz kısmı ezbere yapmış gibi oldu ama yapacak birşey yok, ragdoll gerçekten karmaşık bir sistem. Neyse ki FPS oyun yapmadığınız sürece ragdoll'lar ile uğraşmaya ihtiyaç duymamanız olasılığı yüksek.

Bu bölümde ragdoll'ları işledik. Ayrıca yeni layer oluşturup objeleri bu layer'lara vermeyi ve iki layer arasında fiziksel etkileşim olmamasını sağlamayı öğrendik.

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

Unity 3D İle Oyun Programlama

Bölüm 11: Düşmanın Hasar Vermesi



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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Biz zombilere hasar verebiliyoruz ama zombiler bize hasar veremiyor. Bu bölümde, zombilerin menzilimize girince bize hasar vermesini sağlayacağız.

Sağlığı Ekranda Göstermek

Hasar alıpmadığımızı kolayca anlamak için sağlığını ekranda göstermeliyiz.

Health scriptini güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     void Start()
17     {
18         _currentHealth = _maximumHealth;
19     }
20
21     public void Damage(int damageValue)
22     {
23         _currentHealth -= damageValue;
24
25         if (_currentHealth <= 0)
26         {
27             Animation a = GetComponentInChildren<Animation>();
28             a.Stop();
29
30             Destroy(GetComponent<PlayerMovement>());
31             Destroy(GetComponent<PlayerAnimation>());
32
33             Destroy(GetComponent<EnemyMovement>());
34             Destroy(GetComponent<CharacterController>());
35
36             Ragdoll r = GetComponent<Ragdoll>();
37             if (r != null)
38             {
39                 r.OnDeath();
40             }
41         }
42     }
43 }
44 }
```

Her scriptte otomatik olarak yer alan (ama scriptte gözükmeyen) `ToString` fonksiyonunu kendimiz yeniden yazıyoruz (override). Script bize mevcut canımızın maksimum canımıza olan oranını söylüyor; "94 / 100" gibi...

`ToString`'in döndürdüğü bu string'i ekranда göstermemiz lazım. Bunun için PlayerGui kodunu değiştirelim:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _crosshair;
08
09     Health _playerHealth;
10
11     void Start()
12     {
13         _playerHealth = GetComponent<Health>();
14     }
15
16     void OnGUI()
17     {
18         GUI.Label(new Rect(5,5,100,100), "Health: " + _playerHealth.ToString());
19
20         float x = (Screen.width - _crosshair.width) / 2;
21         float y = (Screen.height - _crosshair.height) / 2;
22         GUI.DrawTexture(new Rect(x, y, _crosshair.width, _crosshair.height), _crosshair);
23     }
24 }
25
```

Player'ın Health scriptini `_playerHealth` değişkeninde tutuyoruz ve OnGUI fonksiyonunda bu scriptteki `ToString()` fonksiyonunu kullanıyoruz.

Aslına bakarsanız `ToString` fonksiyonu özel bir fonksiyon. Kendisini çağırmak için o class'a erişmeniz yeterli. Yani class'a eriştiğten sonra elle `ToString()` fonksiyonunu çağırmak zorunda değilsiniz, bu işlem otomatik yapılır (`ToString`'e has bir durum):

```
18     GUI.Label(new Rect(5,5,100,100), "Health: " + _playerHealth);
```

Oyunu çalıştırınca ekranın sol üstünde sağlığınıza görebilirsiniz.



Düşmanın Bize Saldırmasını Sağlamak

Zombi yakınımızda gelince bize saldırın ve hasar versin.

Yapacağımız şey şu: zombiye küre şeklinde bir collider (sphere collider) vereceğiz ve player bu collider'ın içinde olduğu süre boyunca hasar alacak.

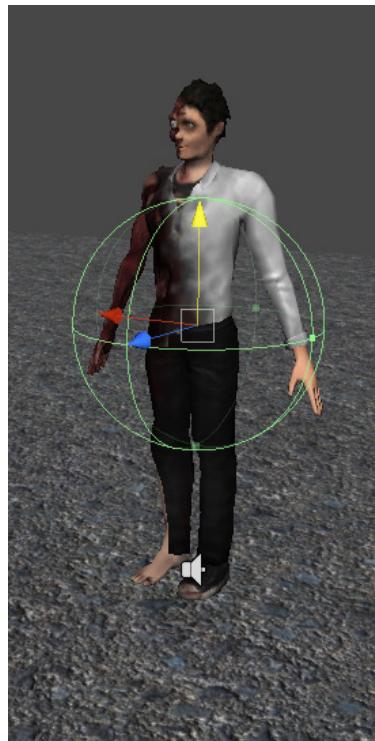
GameObject > Create Empty ile "EnemyAttack" adında yeni bir obje oluşturup objeye Component > Physics > Sphere Collider ile bir Sphere Collider ekleyin.

EnemyAttack objesini zombilerden tekinin child objesi yapın.

Sonrasında EnemyAttack'ın pozisyonunu (0, 0, 0), eğimini (rotation) da (0, 0, 0) ve boyutunu (scale) ise (1, 1, 1) yapın. ([Kısa yolu: Transform'un sağındaki dişli ikona tıklayıp Reset deyin.](#))



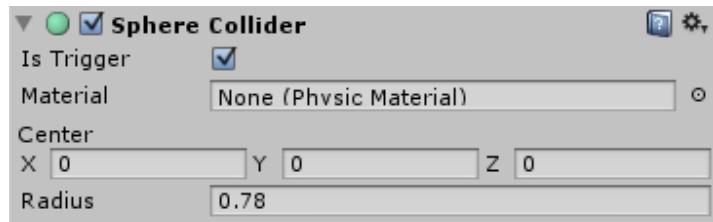
Şimdi kürenin merkezini göbek hizasına çıkarın:



Bu küre zombinin saldırısı menzilini temsil ediyor. Şu an çok küçük olduğu için zombinin bize saldırması çok zor. Küreyi büyütelim ki zombinin menzili artsin, bize daha rahat saldırısın. Bunun için Inspector'da Sphere Collider'ın Radius (yarıçap) değerini 0.78 yapın.



Inspector'da bir de "Is Trigger" adında bir değer göreceksiniz. Onu işaretleyin. Bu seçeneği işaretlediğiniz zaman collider duvar etkisi yapmaz ve collider'ın içinden rahatça geçilebilir.



EnemyAttack objesinin seçili olduğundan emin olun ve objenin layer'ını "Ignore Raycast" olarak değiştirin. Player ateş ederken raycast sistemi kullanıyor ve biz bu raycast'i Enemy objesinin almasını istiyoruz, EnemyAttack objesinin değil. EnemyAttack'ın layer'ını "Ignore Raycast" yaptığımız için her ne kadar objede sphere collider olsa da raycast işinleri bu collider'ı yok sayacak ve böylece raycast yanlışlıkla EnemyAttack objesini vurmayacak.

Şimdi düşmanın hasar vermesini kodlayalım.

"EnemyAttack" adında yeni bir C# scripti oluşturun:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAttack : MonoBehaviour
05 {
06     void OnTriggerEnter(Collider other)
07     {
08         if (other.tag == "Player")
09         {
10             Debug.Log("we're colliding with player!");
11         }
12     }
13 }
14 }
```

OnTriggerStay fonksiyonu bir collider, "Is Trigger"ı işaretli başka bir collider ile temas halinde olduğu süre boyunca her frame'de çalıştırılır.

Yani Sphere Collider'ımıza bir başka collider temas ettiği müddetçe fonksiyon çalıştırılacak. Fonksiyonun içinde ise ilk önce temas eden objenin tag'ını kontrol ediyoruz ve eğer tag'ı "Player" ise Debug.Log fonksiyonu ile konsola yazı yazdırıyoruz.

Şimdi EnemyAttack scriptini EnemyAttack objesine verin. Sonrasında EnemyAttack'ın parent'ı olan Enemy objesini seçip Inspector'dan "Apply" butonuna basın ve böylece yaptığımız bu değişikliklerin tüm zombi prefab klonlarına uygulanmasını sağlamış olun.

Sıra geldi test etmeye! Bir zombi ile temas halinde olduğunuzda konsola mesaj yazdırılacak.

Player'a Hasar Vermek

Test başarılı olduğuna göre Debug.Log komutunu gerçek bir hasar verme kodu ile değiştirelim:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAttack : MonoBehaviour
05 {
06     void OnTriggerStay(Collider other)
07     {
08         if (other.tag == "Player")
09         {
10             Health playerHealth = other.GetComponent<Health>();
11             playerHealth.Damage(1);
12         }
13     }
14 }
15
```

Oyunu çalıştırıp zombiler tarafından hasar alın. Sol üstteki sağlık azalacak ve öldüğünüz zaman ragdoll devreye girecek.

Herşey güzel duruyor ama player çok çabuk hasar alıyor ve ölüyor. Bunu çözmemiz lazım.

Hızlı Hasar Alma Sorununu Çözmek

OnTriggerStay'in collider ile temasta olduğumuz her frame'de çağrıldığından bahsettim. Eğer oyun 60 FPS'de (frames per second) oynuyorsa bunun anlamı fonksiyonun saniyede 60 kez çağrılması ve haliyle saniyede 60 hasar almamızdır!

Her frame'de 1 hasar almak yerine örneğin her 1 saniyede 5 hasar almak çok daha mantıklı:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAttack : MonoBehaviour
05 {
06     float _nextTimeAttackIsAllowed = -1.0f;
07
08     void OnTriggerStay(Collider other)
09     {
10         if (other.tag == "Player" && Time.time >= _nextTimeAttackIsAllowed)
11         {
12             Health playerHealth = other.GetComponent<Health>();
13             playerHealth.Damage(5);
14             _nextTimeAttackIsAllowed = Time.time + 1.0f;
15         }
16     }
17 }
18
```

10. satırda Time.time kullanıyoruz. Bu değişkenin değeri oyun başladığında 0'dır ve her saniyede değeri 1 artar. “_nextTimeAttackIsAllowed” değişkeni zombinin hasar verebilmesi için Time.time'ın olması gereken minimum değeri depolar. Eğer Time.time bu değerden büyükse hasar verme kodlarını çalıştırıyoruz.

14. satırda, hasar aldıktan sonra “_nextTimeAttackIsAllowed” değişkeninin değerini Time.time değişkeninin değerinin 1 fazlası olarak ayarlıyoruz. Bunun anlamı da zombi şu andan (Time.time) bir saniye sonra bize tekrar saldırabilecek.

1 saniye ve 5 hasar değerlerini birer değişkenden çeksek daha güzel olur. Böylece bu değerleri kolayca istediğimiz gibi değiştirebiliriz:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAttack : MonoBehaviour
05 {
06     float _nextTimeAttackIsAllowed = -1.0f;
07
08     [SerializeField]
09     float _attackDelay = 1.0f;
10
11     [SerializeField]
12     int _damageDealt = 5;
13
14     void OnTriggerEnter(Collider other)
15     {
16         if (other.tag == "Player" && Time.time >= _nextTimeAttackIsAllowed)
17         {
18             Health playerHealth = other.GetComponent<Health>();
19             playerHealth.Damage(_damageDealt);
20             _nextTimeAttackIsAllowed = Time.time + _attackDelay;
21         }
22     }
23 }
24 }
```

Player'ın Ölümü İle İlgili Ufak Bir Sorun

Öldüğümüz zaman hareket etmeyi kesiyor ve animasyonu durduruyoruz ama hâlâ ateş edebiliyoruz. Bu sorunu çözmek için Health scriptini düzenleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     void Start()
17     {
18         _currentHealth = _maximumHealth;
19     }
20
21     public void Damage(int damageValue)
22     {
23         _currentHealth -= damageValue;
24
25         if (_currentHealth <= 0)
26         {
27             Animation a = GetComponentInChildren<Animation>();
28             a.Stop();
29
30             Destroy(GetComponent<PlayerMovement>());
31             Destroy(GetComponent<PlayerAnimation>());
32             Destroy(GetComponent<RifleWeapon>());
33
34             Destroy(GetComponent<EnemyMovement>());
35
36             Destroy(GetComponent<CharacterController>());
37
38             Ragdoll r = GetComponent<Ragdoll>();
39             if (r != null)
40             {
41                 r.OnDeath();
42             }
43         }
44     }
45 }
46

```

Ölünce RifleWeapon scriptini de objeden atıyoruz ve artık ateş etme imkanımız olmuyor.

Sağlığın Negatif Değer Almasını Önlemek

Farkedeceğiniz üzere sağlığımız negatif değer de alabiliyor ama bu biraz saçma duruyor sanki, değil mi? Çözümü ise basit:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     void Start()
17     {
18         _currentHealth = _maximumHealth;
19     }
20
21     public void Damage(int damageValue)
22     {
23         _currentHealth -= damageValue;
24
25         if (_currentHealth < 0)
26         {
27             _currentHealth = 0;
28         }
29
30         if (_currentHealth == 0)
31         {
32             Animation a = GetComponentInChildren<Animation>();
33             a.Stop();
34
35             Destroy(GetComponent<PlayerMovement>());
36             Destroy(GetComponent<PlayerAnimation>());
37             Destroy(GetComponent<RifleWeapon>());
38
39             Destroy(GetComponent<EnemyMovement>());
40
41             Destroy(GetComponent<CharacterController>());
42
43             Ragdoll r = GetComponent<Ragdoll>();
44             if (r != null)
45             {
46                 r.OnDeath();
47             }
48         }
49     }
50 }
```

Düşmanın Ölse De Hâlâ Hasar Verebilmesini Durdurmak

Bir zombi öldüğünde ondan EnemyAttack scriptini silersek artık bize hasar veremeyecektir:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     void Start()
17     {
18         _currentHealth = _maximumHealth;
19     }
20
21     public void Damage(int damageValue)
22     {
23         _currentHealth -= damageValue;
24
25         if (_currentHealth < 0)
26         {
27             _currentHealth = 0;
28         }
29
30         if (_currentHealth == 0)
31         {
32             Animation a = GetComponentInChildren<Animation>();
33             a.Stop();
34
35             Destroy(GetComponent<PlayerMovement>());
36             Destroy(GetComponent<PlayerAnimation>());
37             Destroy(GetComponent<RifleWeapon>());
38
39             Destroy(GetComponent<EnemyMovement>());
40             Destroy(GetComponentInChildren<EnemyAttack>());
41
42             Destroy.GetComponent<CharacterController>();
43
44             Ragdoll r = GetComponent<Ragdoll>();
45             if (r != null)
46             {
47                 r.OnDeath();
48             }
49         }
50     }
51 }
52 }
```

EnemyAttack scriptinin bir child objede (EnemyAttack objesi) olduğunu hatırlayın. Bu sebepten ötürü scripte GetComponentInChildren ile erişiyoruz.

Ölünce Ekranda Mesaj Göstermek

Öldüğümüzde ekranda bunu belirten bir mesaj belirse daha güzel durur bence.

Project panelinde "TheGame" klasörünün içinde "Gui" adında yeni bir klasör oluşturun.

Şimdi Winrar arşivinin olduğu yerdeki "Gui" klasöründe yer alan "GameOverScreen.png" dosyasını projenize import edip Project panelinden "Gui" klasörünüzü içine atın.

Import ettiğiniz GameOverScreen asset'ini seçin ve Inspector'dan Texture Type'ı "GUI (Editor \ Legacy)" olarak değiştirin. Sonrasında "Apply" butonuna basın. Böyle yaptığımız zaman resmin arkaplanı görünmez oluyor (zaten önizleme penceresinden siz de görebilirsiniz).

Sırada Player öldüğünde ekranda bu resmi göstermek var. Ama önce Player'ın ölüp ölmeyeğini bilmemiz lazım. Health scriptini şöyle güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     public bool IsDead { get{ return _currentHealth <= 0; } }
17
18     void Start()
19     {
20         _currentHealth = _maximumHealth;
21     }
22
23     public void Damage(int damageValue)
24     {
25         _currentHealth -= damageValue;
26
27         if (_currentHealth < 0)
28         {
29             _currentHealth = 0;
30         }
31
32         if (_currentHealth == 0)
33         {
34             Animation a = GetComponentInChildren<Animation>();
35             a.Stop();
36         }
37     }
38 }
```

```

37         Destroy.GetComponent<PlayerMovement>());
38         Destroy.GetComponent<PlayerAnimation>());
39         Destroy.GetComponent<RifleWeapon>());
40
41         Destroy.GetComponent<EnemyMovement>());
42         Destroy.GetComponentInChildren<EnemyAttack>());
43
44         Destroy.GetComponent<CharacterController>());
45
46         Ragdoll r = GetComponent<Ragdoll>();
47         if (r != null)
48         {
49             r.OnDeath();
50         }
51     }
52 }
53 }
54 }
```

IsDead ne bir değişken ne de bir fonksiyondur. Arada kalmış bir property'dir. (Bu property kavramını öğrenmek için Unity'nin resmî video tutorialine bakmanızı şiddetle tavsiye ederim: <http://unity3d.com/learn/tutorials/modules/intermediate/scripting/properties>)

Şimdi de "CombatGui" adında yeni bir C# scripti oluşturun:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class CombatGui : MonoBehaviour
05 {
06     Health _playerHealth;
07
08     void Start()
09     {
10         GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
11         _playerHealth = playerGameObject.GetComponent<Health>();
12     }
13
14     void OnGUI()
15     {
16         if (_playerHealth.IsDead)
17         {
18             // game over resmini burada ekrana çizdireceğiz
19         }
20     }
21 }
22 }
```

Player objesinin Health scriptini bir değişkende tutuyoruz. Ama önce Player objesine erişmemiz lazım (CombatGui scriptini Player'a değil başka bir objeye vereceğiz). Bunun için ise önce FindGameObjectWithTag ile "Player" tag'lı objeye, ardından onun Health scriptine erişiyoruz. OnGUI fonksiyonunda ise Player'in ölüp olmadığını sorguluyoruz.

Eğer Player ölmüşse ekranada az önce import ettiğimiz resmi gösterelim:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class CombatGui : MonoBehaviour
05 {
06     Health _playerHealth;
07
08     [SerializeField]
09     Texture2D _gameOverImage;
10
11     void Start()
12     {
13         GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
14         _playerHealth = playerGameObject.GetComponent<Health>();
15     }
16
17     void OnGUI()
18     {
19         if (_playerHealth.IsDead)
20         {
21             GUI.DrawTexture(new Rect(0, 0, _gameOverImage.width, _gameOverImage.height),
22                             _gameOverImage);
23         }
24     }
25 }
26

```

PlayerGui kodunda ekrana crosshair çizdirirken de benzer bir kod kullandığımız için açıklama yapmama gerek yok.

Artık scripti test edelim. "CombatGui" adında yeni bir empty gameobject oluşturun. CombatGui scriptini objeye verin. "Game Over Image" kısmına değer olarak "GameOverScreen" resmini verin.



Oyunu test edince game over yazısının ekranın ortasında değil sol üst noktasında gösterildiğini göreceksiniz. Çünkü resmin çizdirileceği Rect'in X ve Y koordinatları (0, 0). Bunu çözmek için crosshair'de kullandığımız "resmi ortalama" metodunu kullanalım:

```

17     void OnGUI()
18     {
19         if (_playerHealth.IsDead)
20         {
21             float x = (Screen.width - _gameOverImage.width) / 2;
22             float y = (Screen.height - _gameOverImage.height) / 2;
23             GUI.DrawTexture(new Rect(x, y, _gameOverImage.width, _gameOverImage.height),
24                             _gameOverImage);
25         }
26     }
27 }
28

```

Ve işte sonuç karşınızda:



Özet Geçecek Olursak...

Bu bölümde düşmanların bize hasar verebilmesini sağladık. Bunun için Is Trigger'ı işaretli sphere collider kullandık. Ayrıca Time.time değişkenini de ilk defa kullanmış olduk. Çok iyi gidiyoruz!

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

Unity 3D İle Oyun Programlama

Bölüm 12: Düşmanların Spawn Olması



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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Şu anda oyunda sadece üç zombi var. Onları öldürünce tüm olay bitiyor. Peki ya zamanla yeni zombiler spawn edersek (oluşturursak) nasıl olur?

Düşmanın Spawn Olması

Basit bir script yazarak işe başlayalım. Her 10 saniyede bir sahnedeki bir spawn objesinden bir zombi spawn olsun. “EnemySpawnManager” adında yeni bir script oluşturun:

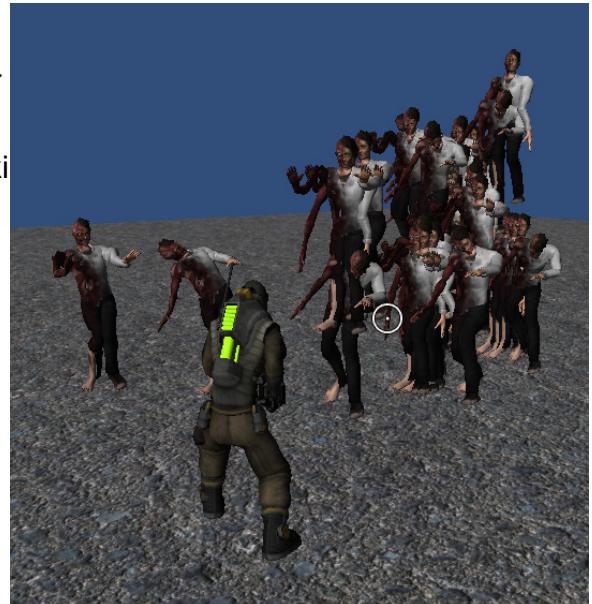
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     void Update()
10     {
11         Instantiate(_enemyToSpawn);
12     }
13 }
14
```

Şimdi de sahnede “EnemySpawnManager” adında yeni bir empty gameobject oluşturun.

EnemySpawnManager scriptini objeye verin. Inspector'daki “Enemy To Spawn”a değer olarak Project panelinden Enemy prefab'ını verin. Oyunu çalıştırın (**bence** çalıştmayın, yoksa Unity birkaç saniye içinde takılmaya başlayacak).

Düşmanların çok hızlı bir şekilde spawn olduğunu göreceksiniz.

Nasıl düşmanın iki saldırısı arasına Time.time kullanarak bir gecikme koyduysak spawn sistemine de ayınsını yapalım.



Görsel 12.1: Bu kadarı da çok fazla!

Kodu şöyle güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     void Update()
15     {
16         if (Time.time >= _nextSpawnTime)
17         {
18             Instantiate(_enemyToSpawn);
19             _nextSpawnTime = Time.time + _spawnDelay;
20         }
21     }
22 }
23
```

Artık iki spawn arasında _spawnDelay kadar saniye geçecek.

Spawn Sisteminin Geliştirmek

Şu anda zombilerin spawn olduğu yer belli ve hep oradan spawn oluyorlar. Bu çok güzel görünmüyör.

Sistemi biraz daha geliştirerek zombilerin bulunduğu konumun yakınında rastgele bir yerde spawn olmasını sağlayalım.

Aşama aşama gideceğiz. Önce kodu şöyle değiştirin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     void Update()
15     {
16         if (Time.time >= _nextSpawnTime)
17         {
18             Vector3 placeToSpawn = new Vector3(0, 0.75f, 0);
19             Quaternion directionToFace = Quaternion.identity;
```

```

20         Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
21     }
22 }
23 }
24 }
25 }
```

Instantiate fonksiyonuna iki yeni parametre ekledik: placeToSpawn ile düşmanın 3 boyutlu uzayda hangi konumda (position) spawn olacağını belirlerken directionToFace ile de düşmanın spawn olduğu anda ki eğimini (rotation) belirledik.

Şu anda düşman (0, 0.75, 0) konumunda spawn oluyor. Bu değeri değiştirerek düşmanın rastgele bir yerde spawn olmasını sağlayacağız.

Eğim (rotation) olarak ise Quaternion.identity değerini kullanıyoruz. Bunun anlamı düşmanın doğduğu anda Euler açı cinsinden (0,0,0) rotasyonuna sahip olacağıdır.

Kodu güncellemeye devam edelim:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     void Update()
15     {
16         if (Time.time >= _nextSpawnTime)
17         {
18             Vector3 edgeOfScreen = new Vector3(1.0f, 0.5f, 8.0f);
19             Vector3 placeToSpawn = Camera.main.ViewportToWorldPoint(edgeOfScreen);
20             Quaternion directionToFace = Quaternion.identity;
21             Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
22             _nextSpawnTime = Time.time + _spawnDelay;
23         }
24     }
25 }
26 }
```

Oyunu çalıştırın. Zombiler ekranın sağ kenarının tam ortasından spawn olacaklar ve kamerayı döndürseniz bile zombiler yine ekranın sağından spawn olmaya devam edecekler.

Bu güzel sistemin nasıl işlediğini anlamak için önce viewport space'i anlamalıyız.

Viewport Space

Daha önce local space (yerel uzay) ile world space'i (global uzay) gördük. Bir başka space türü ise viewport space. Diğer iki uzayın aksine viewport space 3D düzlemde değil 2D düzlemde çalışır.

Viewport space, ekranınızı temsil eden bir uzaydır. Bu space'in X ve Y koordinatları vardır. Bu uzaydaki bir vektörün X koordinatı 0.0 olunca ekranın en solu, 0.5 olunca ekranın ortası, 1.0 olunca ekranın en sağı kastedilir. Benzer şekilde, vektörün Y koordinatı 0.0 olunca ekranın en altı, 0.5 olunca ekranın ortası, 1.0 olunca ekranın en üstü kastedilir.

Yani Viewport Space'te (0.0, 0.0) ekranın sol alt köşesini, (0.5, 0.5) vektörü ekranın tam ortasını, (1.0, 1.0) vektörü ise ekranın sağ üst köşesini temsil eder. (1.0, 0.5) ise ekranın yatay eksende en sağını, dikey eksende de orta noktasını temsil eder. Yani ekranın sağ-orta noktasını temsil eder.

placeToSpawn'a değer verirken artık Camera class'ının ViewportToWorldPoint fonksiyonunu kullanıyoruz. Bu fonksiyon viewport space'indeki bir noktayı world space'indeki bir noktaya çevirir.

Peki ama 2D uzaydan 3D uzaya nasıl olur? Sonuçta 3D uzayda bir derinlik var. Bu derinlik için de fonksiyonun aldığı Vector3 parametresinin (edgeOfScreen) Z değeri kullanılıyor.

Vector3'ün X ve Y değerleri viewport space'teki bir noktayı temsil ederken Z değeri ise bu noktayı world space'e çevirirken elde ettiğimiz noktanın kameradan kaç birim uzakta olacağını belirliyor.

Fonksiyona parametre olarak verdığımız edgeOfScreen vektörünün değeri (1.0, 0.5, 8.0). Dediğim gibi, X ve Y koordinatları viewport space'teki bir noktayı temsil ediyor: (1.0, 0.5). Yani ekranın sağ-orta noktası. Z koordinatına değer olarak 8.0 verdik, bunun anlamı da elimizdeki noktayı viewport space'ten world space'e çevirince elde ettiğimiz noktanın kameraya uzaklığı 8.0 birim olacak.

Instantiate fonksiyonu düşmanı placeToSpawn noktasında oluşturduğu için de kamerası ne kadar çevirirsek çevirelim yeni düşmanlar hep kamerasının sağ-orta noktasında, kamerasından 8.0 birim uzakta spawn ediliyor.

Spawn Noktasını Geliştirmek

edgeOfScreen'in X koordinatı 1.0 olduğu için zombiler ekranın sağ kenarında spawn oluyor. Ancak böyle olunca onların orada *puf* diye spawn olduklarını kolayca görebiliyoruz. Bu X koordinatını öyle bir değiştirmeliyiz ki zombilerin *puf* diye spawn olduklarını görmemeliyiz. Yani zombiler ekranın dışında bir yerde spawn olmalıdır.

Neyse ki bu çok basit. Viewport space'te X koordinatı 0.0 ile 1.0 olacak diye bir kural yok. Eğer X koordinatı 2.0 olursa bu ekranın sağ kenarından, ekranın genişliği kadar daha sağda olan görünmez bir noktayı temsil eder.

Eğer edgeOfScreen'in X koordinatını 1.25 olarak değiştirirsek bu zaman da zombilerin spawn olacağı nokta ekranın sağ kenarından, ekranın genişliğinin 1/4'ü kadar daha uzakta olacak ve onların *puf* diye aniden spawn olduğunu görmeyeceğiz:

```
18     Vector3 edgeOfScreen = new Vector3(1.25f, 0.5f, 8.0f);
```

Artık zombileri spawn olurken değil, spawn olduktan sonra bize doğru hareket ederken göreceğiz.

Böylesi çok daha iyi. Ama bununla yetinmek bize yakışmaz. edgeOfScreen'in Y koordinatı ile de biraz oynayalım:

```
18     Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
```

Hatırlayın, Random.value değeri 0.0 ile 1.0 arasında rastgele bir sayı döndürür. Bu sayıyı viewport space'in Y eksenine değer olarak veriyoruz ve böylece zombiler ekranın sadece sağ-orta noktasından değil ama sağ-alt ve sağ-üst noktalarından da (sağ kenardaki herhangi bir noktadan) spawn olabiliyor.

Bazen birkaç saniye beklediğiniz halde bir zombinin spawn olmadığını göreceksiniz. Çünkü kamera açısına bağlı olarak zombilerin spawn noktası bazen yerin altında oluyor ve zombiler spawn oldukları gibi aşağı düşmeye başlıyorlar. Veya zeminin bir kenarına gelip boşluğu ekranın sağ tarafına alırsanız da zombilerin boşlukta spawn olup aşağı düştüğünü görebilirsiniz. Bu gayet ciddi bir sorun. Zombilerin her zaman için zeminin üzerinde spawn olduklarından emin olmalıyız.

Bu işlem için raycast sistemine başvuracağız:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     void Update()
15     {
16         if (Time.time >= _nextSpawnTime)
17         {
18             Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
19             Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
20             RaycastHit hit;
21             if (Physics.Raycast(ray, out hit))
22             {
23                 Vector3 placeToSpawn = hit.point;
24                 Quaternion directionToFace = Quaternion.identity;
25                 Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
26                 _nextSpawnTime = Time.time + _spawnDelay;
27             }
28         }
29     }
30 }
31
```

Viewport space'teki bir noktayı direkt world space'teki bir noktaya çevirmek yerine artık o noktadan çıkan bir raycast işini (ray) oluşturuyoruz. Eğer bu raycast işini bir yere temas ederse (if koşulunu içine girersek) zombiyi bu temas edilen noktada (hit.point) oluşturuyoruz. Yok eğer raycast işini bir yere çarpmazsa düşman spawn etmiyoruz.

ÇEVİRMEN EKLEMESİ: Artık edgeOfScreen'in Z koordinatı olan 8.0 değeri önelsiz oldu. Çünkü bir Raycast işininin "derinliği" gibi birşey söz konusu değil.

Düşmanlar hep ekranın sağ kenarından spawn oluyor. Bazen de ekranın sol kenarından spawn olmaları güzel olurdu. Çok ufak bir değişiklik ile bunu yapmak mümkün:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     void Update()
15     {
16         if (Time.time >= _nextSpawnTime)
17         {
18             Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
19             if (Random.value > 0.5f)
20             {
21                 edgeOfScreen.x = -0.25f;
22             }
23
24             Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
25             RaycastHit hit;
26             if (Physics.Raycast(ray, out hit))
27             {
28                 Vector3 placeToSpawn = hit.point;
29                 Quaternion directionToFace = Quaternion.identity;
30                 Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
31                 _nextSpawnTime = Time.time + _spawnDelay;
32             }
33         }
34     }
35 }
36

```

Random.value değerini tekrar çağrıyoruz ve eğer bu değer 0.5'ten büyükse (%50 olasılık) edgeOfScreen'in X koordinatını -0.25 yapıyoruz. Bu da viewport uzayda ekranın sol kenarından ekranın genişliğinin 1/4'ü kadar soldaki noktayı temsil ediyor.

Zombilerin Birbirinin Üzerinde Spawn Olmasını Önlemek

Bazen zombiler birbiri üzerinde spawn olabilmekte:



Eğer raycast işini bir zombinin kafasına denk gelirse böyle istenmeyen bir sonuç oluşuyor. Bizim istediğimiz şey raycast işininin zombileri es geçmesi; yani onlara temas etse bile onların içinden geçip yoluna devam etmesi.

Bunun için birkaç ufak değişiklik lazım:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     [SerializeField]
15     LayerMask _spawnLayer;
16
17     void Update()
18     {
19         if (Time.time >= _nextSpawnTime)
20         {
21             Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
22             if (Random.value > 0.5f)
23             {
24                 edgeOfScreen.x = -0.25f;
25             }
26         }
27     }
28 }
```

```

26
27     Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
28     RaycastHit hit;
29     if (Physics.Raycast(ray, out hit, Mathf.Infinity, _spawnLayer.value))
30     {
31         Vector3 placeToSpawn = hit.point;
32         Quaternion directionToFace = Quaternion.identity;
33         Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
34         _nextSpawnTime = Time.time + _spawnDelay;
35     }
36 }
37 }
38 }
39

```

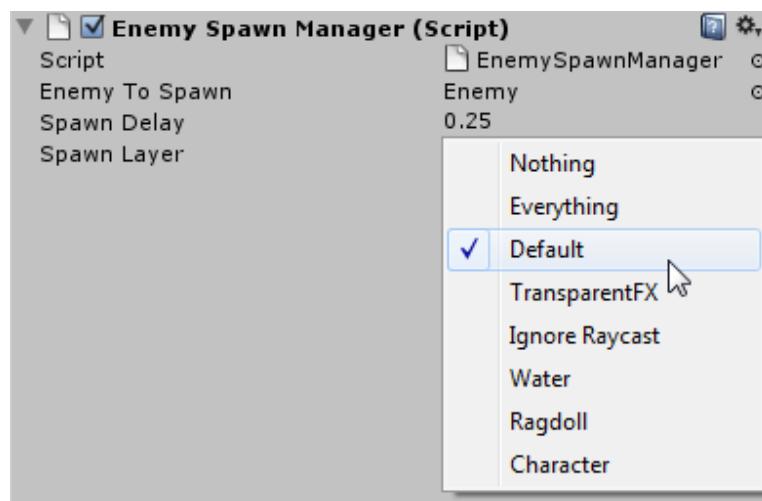
Unity'de *layer mask* olarak bilinen bir değişken türünden faydalıyoruz. Bu değişkeni kullanarak raycast işininin hangi layer'daki objeleri görmezden gelip hangilerine temas edebileceğini belirleyebiliyoruz.

15. satırda LayerMask değişkenimizi oluşturuyoruz.

29. satırda ise bu değişkeni kullanıyoruz. Maalesef Unity'de Raycast fonksiyonuna LayerMask verecekseniz öncesinde o raycast işininin uzunluğunu parametre olarak girmelisiniz. Biz işinin 2-3 metre olmasını istemiyor, sonsuza kadar gitmesini istiyoruz. Bunun için işinin uzunluğunu Mathf.Infinity (sonsuz) olarak belirliyoruz.

EnemySpawnManager'ın Inspector'unda Spawn Layer adında bir değişken göreceksiniz. Bu bizim Layer Mask'ımız. Varsayılan değeri "Nothing" (hiçbir şey). Yani raycast işini hiçbir layer ile temas geçmiyor.

Bizim istediğimiz ise sadece zemin ile temasa geçmesi. Zemin objemiz (Ground) "Default" layer'ında yer alırken zombi objesi "Character" layer'ında yer alıyor. Yani Layer Mask'e değer olarak sadece "Default" verirsek raycast işini sadece zemin ile temas edecek. Bu tam da istediğimiz şey!



Zombi Sayısını Sınırılamak

Oyunu biraz oynadıktan sonra ekranда çok fazla zombi birikmekte ve bu da oyunun takılmasına yol açmaktadır. Oyunda aynı anda maksimum kaç tane zombinin yer alabileceğini sınırlamalıyız.



Görsel 12.2: Çok fazla zombi var, çok!

Örneğin zombi sayısı 30 ise daha fazla zombi spawn olmasın, ta ki bu 30 zombiden en azından birini öldürrene kadar.

Elbette ki bunu başarmak için henüz ölmemiş olan zombilerin sayısını bir değişkende tutmalıyız.

EnemySpawnManager scriptini güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     static int _livingZombies = 0;
07
08     [SerializeField]
09     GameObject _enemyToSpawn;
10
11     [SerializeField]
12     float _spawnDelay = 1.0f;
```

```

13
14     float _nextSpawnTime = -1.0f;
15
16     [SerializeField]
17     LayerMask _spawnLayer;
18
19     void Update()
20     {
21         if (Time.time >= _nextSpawnTime && _livingZombies < 30)
22         {
23             Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
24             if (Random.value > 0.5f)
25             {
26                 edgeOfScreen.x = -0.25f;
27             }
28
29             Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
30             RaycastHit hit;
31             if (Physics.Raycast(ray, out hit, Mathf.Infinity, _spawnLayer.value))
32             {
33                 Vector3 placeToSpawn = hit.point;
34                 Quaternion directionToFace = Quaternion.identity;
35                 Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
36                 _nextSpawnTime = Time.time + _spawnDelay;
37                 ++_livingZombies;
38             }
39         }
40     }
41 }
42

```

`_livingZombies` adında static bir değişkenimiz var ve yeni zombi oluşturdukça bu değişkenin değerini 1 artırıyoruz: `++_livingZombies`. Eğer bu değişken 30'dan küçükse ancak o zaman spawn işlemini gerçekleştiriyoruz.

Şimdi bir zombi ölünce `_livingZombies` değişkeninin değerini 1 azaltmalıyız:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     static int _livingZombies = 0;
07     static public void OnEnemyDeath()
08     {
09         --_livingZombies;
10     }
11
12     [SerializeField]
13     GameObject _enemyToSpawn;
14
15     [SerializeField]
16     float _spawnDelay = 1.0f;
17
18     float _nextSpawnTime = -1.0f;
19
20     [SerializeField]
21     LayerMask _spawnLayer;

```

```

22
23     void Update()
24     {
25         if (Time.time >= _nextSpawnTime && _livingZombies < 30)
26         {
27             Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
28             if (Random.value > 0.5f)
29             {
30                 edgeOfScreen.x = -0.25f;
31             }
32
33             Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
34             RaycastHit hit;
35             if (Physics.Raycast(ray, out hit, Mathf.Infinity, _spawnLayer.value))
36             {
37                 Vector3 placeToSpawn = hit.point;
38                 Quaternion directionToFace = Quaternion.identity;
39                 Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
40                 _nextSpawnTime = Time.time + _spawnDelay;
41                 ++_livingZombies;
42             }
43         }
44     }
45 }
46

```

OnEnemyDeath fonksiyonu static bir fonksiyon ve çağrıldığı zaman _livingZombies değişkenini 1 azaltmaya yarıyor. Artık yapmamız gereken bir zombi ölünce bu fonksiyonu çağrırmak.

Health scriptini açıp şu şekilde güncelleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     public bool IsDead { get{ return _currentHealth <= 0; } }
17
18     void Start()
19     {
20         _currentHealth = _maximumHealth;
21     }
22
23     public void Damage(int damageValue)
24     {
25         _currentHealth -= damageValue;
26
27         if (_currentHealth < 0)

```

```

28     {
29         _currentHealth = 0;
30     }
31
32     if (_currentHealth == 0)
33     {
34         Animation a = GetComponentInChildren<Animation>();
35         a.Stop();
36
37         if (tag == "Player")
38         {
39             Destroy(GetComponent<PlayerMovement>());
40             Destroy(GetComponent<PlayerAnimation>());
41             Destroy(GetComponent<RifleWeapon>());
42         }
43         else // its an enemy
44         {
45             EnemySpawnManager.OnEnemyDeath();
46             Destroy(GetComponent<EnemyMovement>());
47             Destroy(GetComponentInChildren<EnemyAttack>());
48         }
49
50         Destroy(GetComponent<CharacterController>());
51
52         Ragdoll r = GetComponent<Ragdoll>();
53         if (r != null)
54         {
55             r.OnDeath();
56         }
57     }
58 }
59 }
60

```

Player objemize daha önceden tag olarak "Player" vermiştık. Scriptte bu tag'ı kontrol ediyoruz ve eğer Health scriptinin atıldığı objenin Tag'ı "Player" ise Player'in ilgili component'lerini siliyoruz; değilse (o halde bu bir zombi objesidir) zombinin ilgili component'lerini siliyoruz ve EnemySpawnManager scriptinin static olan OnEnemyDeath fonksiyonunu çağırıyoruz.

Hazır elimiz geçmişen maksimum zombi sayısını direkt 30 olarak belirlemeyelim ama bir değişken ile kontrol edelim. EnemySpawnManager'a şu eklemeleri yapın:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     static int _livingZombies = 0;
07     static public void OnEnemyDeath()
08     {
09         --_livingZombies;
10     }
11
12     [SerializeField]

```

```

13     GameObject _enemyToSpawn;
14
15     [SerializeField]
16     float _spawnDelay = 1.0f;
17
18     [SerializeField]
19     int _enemyLimit = 30;
20
21     float _nextSpawnTime = -1.0f;
22
23     [SerializeField]
24     LayerMask _spawnLayer;
25
26     void Update()
27     {
28         if (Time.time >= _nextSpawnTime && _livingZombies < _enemyLimit)
29         {
30             Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
31             if (Random.value > 0.5f)
32             {
33                 edgeOfScreen.x = -0.25f;
34             }
35
36             Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
37             RaycastHit hit;
38             if (Physics.Raycast(ray, out hit, Mathf.Infinity, _spawnLayer.value))
39             {
40                 Vector3 placeToSpawn = hit.point;
41                 Quaternion directionToFace = Quaternion.identity;
42                 Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
43                 _nextSpawnTime = Time.time + _spawnDelay;
44                 ++_livingZombies;
45             }
46         }
47     }
48 }
49

```

Zombi Cesetlerini Ortadan Kaldırmak

Yaşayan zombi sayısını sınırladık ama ölen zombilerin cesetleri yerde birikiyor ve zamanla sahnede çok fazla 3D obje olduğu için oyun yine takılmaya başlıyor. Ölen zombilerin cesetlerini sahneden kaldırmalıyız.



Görsel 12.3: Cesetler oyunun performansını hiç de iyi etkilemiyor.

Health scriptini şöyle güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     public bool IsDead { get{ return _currentHealth <= 0; } }
17
18     void Start()
19     {
20         _currentHealth = _maximumHealth;
21     }
22
23     public void Damage(int damageValue)
24     {
25         _currentHealth -= damageValue;
26
27         if (_currentHealth < 0)
28         {
29             _currentHealth = 0;
30         }
31     }
32 }
```

```

31     if (_currentHealth == 0)
32     {
33         Animation a = GetComponentInChildren<Animation>();
34         a.Stop();
35
36         if (tag == "Player")
37         {
38             Destroy(GetComponent<PlayerMovement>());
39             Destroy(GetComponent<PlayerAnimation>());
40             Destroy(GetComponent<RifleWeapon>());
41         }
42     }
43     else // its an enemy
44     {
45         EnemySpawnManager.OnEnemyDeath();
46         Destroy(GetComponent<EnemyMovement>());
47         Destroy(GetComponentInChildren<EnemyAttack>());
48
49         Destroy(gameObject, 8.0f);
50     }
51
52     Destroy.GetComponent<CharacterController>();
53
54     Ragdoll r = GetComponent<Ragdoll>();
55     if (r != null)
56     {
57         r.OnDeath();
58     }
59 }
60 }
61 }
62 }
```

Burada `Destroy` fonksiyonunun iki argümanlı versiyonunu kullanıyoruz. İlk argümana `gameObject`'i veriyoruz, bunun anlamı ölen zombi objesinin yok olacağı. İkinci argümana ise 8.0 değerini veriyoruz, bunun anlamı da bu satır çalıştırıldıkten 8.0 saniye sonra objenin yok olacağıdır. 8 saniye, zombinin ragdoll'unun devreye girip zombinin yerde bir süre kalması için yeterli bir zaman dilimi.

Cesetlerden Kurtulmanın Alternatif Yolu

Cesedin 8 saniye sonra *puf* diye ekrandan kaybolması garip durabilir. Onun yerine ceset kamerasının görüş alanından tamamen çıktığında onu yoketmek daha sağlıklı bir çözüm.

Az önce eklediğimiz `Destroy` satırını silin ve onun yerine scriptte şu değişiklikleri yapın:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
```

```

11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     public bool IsDead { get{ return _currentHealth <= 0; } }
17
18     Renderer _renderer;
19
20     void Start()
21     {
22         _renderer = GetComponentInChildren<Renderer>();
23         _currentHealth = _maximumHealth;
24     }
25
26     public void Damage(int damageValue)
27     {
28         _currentHealth -= damageValue;
29
30         if (_currentHealth < 0)
31         {
32             _currentHealth = 0;
33         }
34
35         if (_currentHealth == 0)
36         {
37             Animation a = GetComponentInChildren<Animation>();
38             a.Stop();
39
40             if (tag == "Player")
41             {
42                 Destroy(GetComponent<PlayerMovement>());
43                 Destroy(GetComponent<PlayerAnimation>());
44                 Destroy(GetComponent<RifleWeapon>());
45             }
46             else // its an enemy
47             {
48                 EnemySpawnManager.OnEnemyDeath();
49                 Destroy(GetComponent<EnemyMovement>());
50                 Destroy(GetComponentInChildren<EnemyAttack>());
51             }
52
53             Destroy(GetComponent<CharacterController>());
54
55             Ragdoll r = GetComponent<Ragdoll>();
56             if (r != null)
57             {
58                 r.OnDeath();
59             }
60         }
61     }
62
63     void Update()
64     {
65         if (IsDead && !_renderer.isVisible)
66         {
67             Destroy(gameObject);

```

```
68     }
69 }
70 }
71 }
```

Daha önce bahsettiğim gibi, Renderer component'i 3D objeyi ekranda çizdirmeye yarar. Bu component'in "isVisible" adında bir değişkeni vardır. Bu değişken, objenin herhangi bir kamera tarafından görünüp görünmediğini depolar. Eğer değeri true ise obje bir kameranın görüş açısındandır, false ise obje herhangi bir kameranın görüş açısından değildir.

İlk önce 18. satırda, düşmanın Renderer component'ini depolayacak bir değişken oluşturuyoruz.

Zombie objesini inceleyeceğiz olursanız "zombie_lowres" adındaki child objesinin Skinned Mesh Renderer component'ine sahip olduğunu görebilirsiniz. Bu, zombimizi ekrana çizdirmeye yarayan component. Component, Zombie'nin child objesinde olduğu için ona 22. satırda GetComponentInChildren fonksiyonu ile erişiyoruz.

Son olarak da Update fonksiyonunun içinde zombinin ölüp ölmemiğini (IsDead) ve bir kamera tarafından görünüp görünmediğini (_renderer.isVisible) kontrol ediyoruz. Eğer zombi ölmüşse ve kameranın görüş alanının dışındaysa onu yokediyoruz.

ÇEVİRMEN EKLEMESİ: Bazen zombi öldüğü ve kameranın dışında olduğu halde silinmeyebilir. Bunun sebebi Unity editöründe çalışırken eğer zombi cesedi Scene panelinde görünür vaziyette ise o zaman _renderer.isVisible'ın true döndürmesidir. Ama eğer oyunu build ederseniz ya da tam ekran olarak test ederseniz (Game panelinin sağ üstündeki "Maximize on Play" seçeneğini seçerek) bu sorunu yaşamazsınız.

Özet Geçecek Olursak...

Oyunumuz gerçekten birşeye benzemeye başladı. Artık bizi sadece üç zombi kovalamıyor ve zombiler öldükçe yenileri gelmeye devam ediyor. Bu bölümde viewport space'i, layer mask değişkenini ve bir objenin kameraların görüş alanında olup olmadığını kontrol etmeyi gördük.

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

Unity 3D İle Oyun Programlama

Bölüm 13: Oyunu Kazanmak



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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Bu bölümde oyuna bir amaç vereceğiz; kendimizi sadece zombi öldürmekle sınırlamayacağız.

Bir kurtarma helikopteri diyelim ki 3 dakika sonra bulunduğuuz bölge olacak. Bu süre zarfında bize saldıran zombileri püskürtmek zorundayız. Helikoptere bindiğimiz zaman ise oyunu kazanmış olacağız.

İlk Rötuşlar

Önce basit bir script ile başlayalım. Oyun başladıkten 3 dakika sonra oyunu direkt kazanmış olalım; henüz helikopterle uğraşmayağım.

“GameManager” diye bir C# scripti oluşturun:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class GameManager : MonoBehaviour
05 {
06     [SerializeField]
07     float _secondsToWin = 180.0f;
08
09     void Start()
10    {
11    }
12 }
13
```

_secondsToWin değişkeni oyun başladıkten kaç saniye sonra oyunu kazanacağımızı depolayan değişkendir.

Şimdi Unity'nin Invoke fonksiyonunu kullanacağız. Invoke, scriptte yer alan bir fonksiyonu belli bir miktar saniye sonra çalıştırılmaya (yani geciktirmeli çalıştırılmaya) yarar.

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class GameManager : MonoBehaviour
05 {
06     [SerializeField]
07     float _secondsToWin = 180.0f;
08
09     void Start()
10    {
11        Invoke("CheckWin", _secondsToWin);
12        Debug.Log("Time started: " + Time.time);
13    }
14
15     void CheckWin()
16    {
17        Debug.Log("Its " + Time.time + ". Checking winning conditions now...");
18    }
19 }
20
```

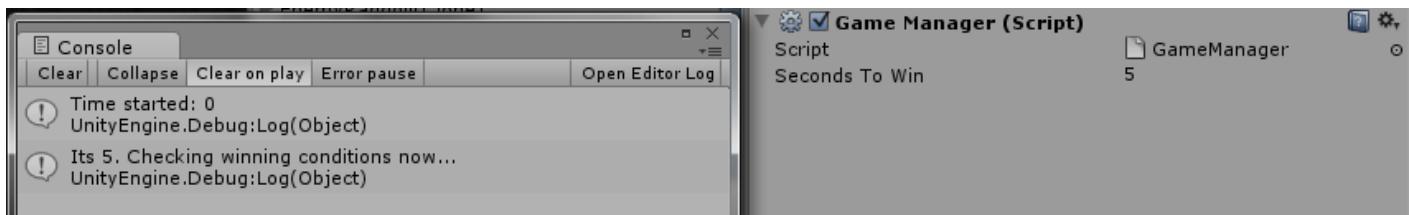
Start'ın içinde Invoke fonksiyonunu kullanıyoruz. İlk parametreye geciktirmeli çalıştırırmak istediğimiz fonksiyonun ismini, ikinci parametreye de bu fonksiyonu kaç saniye sonra çalıştırırmak istediğimizi giriyoruz.

Invoke'a girdiğiniz fonksiyonun public olmasına gerek yok. Örneğin bizim örneğimizdeki CheckWin fonksiyonu bir private fonksiyon.

Kodu test etme zamanı. Yeni bir empty gameobject oluşturun ve ismini "GameManager" olarak değiştirin. Ardından GameManager scriptini bu objeye verin.

"Seconds To Win" değişkeninin değerini Inspector'dan 5 saniyeye düşürün. Test ederken gidip de 3 dakika boş boş beklemeyelim. Sonra bu değeri tekrar 180 yapabilirsiniz.

Oyunu çalıştırın. 5 saniye sonra konsolda bir mesaj belirecek:



Konsola Debug.Log fonksiyonu ile test amaçlı yazılar yazdırıldı. Artık onları gerçek bir kodla değiştirebiliriz:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class GameManager : MonoBehaviour
05 {
06     [SerializeField]
07     float _secondsToWin = 180.0f;
08
09     static bool _hasPlayerWon = false;
10
11     void Start()
12     {
13         Invoke("CheckWin", _secondsToWin);
14     }
15
16     void CheckWin()
17     {
18         _hasPlayerWon = true;
19     }
20 }
21
```

Oyunu kazanıp kazanmadığımızı depolayan static bir değişken oluşturduk: _hasPlayerWon. Bu değişken static olduğu için ona rahatça erişebileceğiz.

Diğer scriptlerden bu scriptteki private bir değişkene ulaşamayız. O halde private olan _hasPlayerWon'a diğer scriptlerden erişebilmek için public bir "property" oluşturalım (**property** ile ilgili detaylı bilgi için bkz: <http://unity3d.com/learn/tutorials/modules/intermediate/scripting/properties>):

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class GameManager : MonoBehaviour
05 {
06     [SerializeField]
07     float _secondsToWin = 180.0f;
08
09     static bool _hasPlayerWon = false;
10
11     public static bool HasPlayerWon { get{ return _hasPlayerWon; } }
12
13     void Start()
14     {
15         Invoke("CheckWin", _secondsToWin);
16     }
17
18     void CheckWin()
19     {
20         _hasPlayerWon = true;
21     }
22 }
23

```

Oyun Bitince Zombilerin Spawn Olmasını Durdurmak

EnemySpawnManager scriptini düzenleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     static int _livingZombies = 0;
07     static public void OnEnemyDeath()
08     {
09         --_livingZombies;
10     }
11
12     [SerializeField]
13     GameObject _enemyToSpawn;
14
15     [SerializeField]
16     float _spawnDelay = 1.0f;
17
18     [SerializeField]
19     int _enemyLimit = 30;
20
21     float _nextSpawnTime = -1.0f;
22
23     [SerializeField]
24     LayerMask _spawnLayer;
25
26     void Update()

```

```

27    {
28        if (GameManager.HasPlayerWon)
29        {
30            Destroy(this);
31            return;
32        }
33
34        if (Time.time >= _nextSpawnTime && _livingZombies < _enemyLimit)
35        {
36            Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
37            if (Random.value > 0.5f)
38            {
39                edgeOfScreen.x = -0.25f;
40            }
41
42            Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
43            RaycastHit hit;
44            if (Physics.Raycast(ray, out hit, Mathf.Infinity, _spawnLayer.value))
45            {
46                Vector3 placeToSpawn = hit.point;
47                Quaternion directionToFace = Quaternion.identity;
48                Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
49                _nextSpawnTime = Time.time + _spawnDelay;
50                ++_livingZombies;
51            }
52        }
53    }
54 }
55

```

Oyunu kazandığımızda Destroy fonksiyonu ile EnemySpawnManager scriptini siliyoruz. Böylece yeni zombilerin spawn olmasının önüne geçiyoruz.

Ekranda Oyun Bitti Mesajı Göstermek

Oyun kazanıldığı zaman oyuncuya bunu ekrandaki bir mesaj ile bildirelim, típkı oyunu kaybedince ekranda game over yazısı yazdırduğumuz gibi...

CombatGui scriptini açıp güncelleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class CombatGui : MonoBehaviour
05 {
06     Health _playerHealth;
07
08     [SerializeField]
09     Texture2D _gameOverImage;
10
11     [SerializeField]
12     Texture2D _winImage;
13
14     void Start()
15     {

```

```

16     GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
17     _playerHealth = playerGameObject.GetComponent<Health>();
18 }
19
20 void OnGUI()
21 {
22     if (_playerHealth.IsDead)
23     {
24         float x = (Screen.width - _gameOverImage.width) / 2;
25         float y = (Screen.height - _gameOverImage.height) / 2;
26         GUI.DrawTexture(new Rect(x, y, _gameOverImage.width, _gameOverImage.height),
27                         _gameOverImage);
28     }
29     else if (GameManager.HasPlayerWon)
30     {
31         float x = (Screen.width - _winImage.width) / 2;
32         float y = (Screen.height - _winImage.height) / 2;
33         GUI.DrawTexture(new Rect(x, y, _winImage.width, _winImage.height), _winImage);
34     }
35 }
36 }
37

```

Game over yazısını ekrana yazdırırken kullandığımız kodun çok benzerini kullandık. Anlatacak fazladan birşey yok.

Winrar arşivinin olduğu yerdeki "Gui" klasörünün içinde "WinScreen.png" adında bir resim dosyası var. Onu projenizdeki "Gui" klasörüne import edin. Ardından dosyayı Unity'de seçin ve Texture Type'i "GUI (Editor \ Legacy)" olarak değiştirin (Apply butonuna basmayı unutmayın).

Şimdi CombatGui objesindeki "Win Image"a değer olarak "WinScreen"i verin ve oyunu çalıştırın:



Belki henüz dikkat etmediiniz ama aslında GameManager scriptimizde ufak bir bug var: eğer oyuncu _secondsToWin kadar saniye geçmeden önce ölüse yine de CheckWin fonksiyonu çalıştırıldığı zaman sanki oyunu kazanmışız gibi _hasPlayerWon'un değeri true oluyor.

CheckWin'de önce player'ın hayatı olup olmadığını kontrol etmeliyiz:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class GameManager : MonoBehaviour
05 {
06     [SerializeField]
07     float _secondsToWin = 180.0f;
08
09     static bool _hasPlayerWon = false;
10
11     public static bool HasPlayerWon { get{ return _hasPlayerWon; } }
12
13     Health _playerHealth;
14
15     void Start()
16     {
17         Invoke("CheckWin", _secondsToWin);
18
19         GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
20         _playerHealth = playerGameObject.GetComponent<Health>();
21     }
22
23     void CheckWin()
24     {
25         if (!_playerHealth.IsDead)
26         {
27             _hasPlayerWon = true;
28         }
29     }
30 }
31
```

Player'ın Health component'ini _playerHealth değişkeninde tutuyoruz ve CheckWin fonksiyonu çağrılmınca önce player'ın yaşayıp yaşamadığını kontrol ediyoruz. Sadece eğer player hayattaysa _hasPlayerWon'un değerini true yapıyoruz.

Skor Yapmak

Öldürdüğümüz zombi sayısını oyun bitince ekranda göstermeye ne dersiniz?

Ölen zombi sayısını tutan bir script gereklili bize. Bunun için "PlayerStats" adında yeni bir C# script oluşturun. **Scripti Player objesine verin** ve ardından kodu şöyle düzenleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerStats : MonoBehaviour
05 {
06     int _zombiesKilled = 0;
07     public int ZombiesKilled { get{ return _zombiesKilled; } set { _zombiesKilled = value; } }
08 }
09

```

Her zombi öldürüşümüzde `_zombiesKilled`'in değerini 1 artırmalıyız. Bunu yapmak için Health scriptini güncelleleyelim:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     public bool IsDead { get{ return _currentHealth <= 0; } }
17
18     Renderer _renderer;
19
20     PlayerStats _playerStats;
21
22     void Start()
23     {
24         _renderer = GetComponentInChildren<Renderer>();
25         _currentHealth = _maximumHealth;
26
27         GameObject player = GameObject.FindGameObjectWithTag("Player");
28         _playerStats = player.GetComponent<PlayerStats>();
29     }
30
31     public void Damage(int damageValue)
32     {
33         _currentHealth -= damageValue;
34
35         if (_currentHealth < 0)
36         {
37             _currentHealth = 0;
38         }
39
40         if (_currentHealth == 0)
41         {
42             Animation a = GetComponentInChildren<Animation>();
43             a.Stop();
44         }

```

```

45         if (tag == "Player")
46     {
47         Destroy.GetComponent<PlayerMovement>());
48         Destroy.GetComponent<PlayerAnimation>());
49         Destroy.GetComponent<RifleWeapon>());
50     }
51     else // its an enemy
52     {
53         _playerStats.ZombiesKilled++;
54         EnemySpawnManager.OnEnemyDeath();
55         Destroy.GetComponent<EnemyMovement>());
56         Destroy.GetComponentInChildren<EnemyAttack>());
57     }
58
59         Destroy.GetComponent<CharacterController>());
60
61         Ragdoll r = GetComponent<Ragdoll>();
62         if (r != null)
63     {
64             r.OnDeath();
65         }
66     }
67 }
68
69 void Update()
70 {
71     if (IsDead && !_renderer.isVisible)
72     {
73         Destroy(gameObject);
74     }
75 }
76 }
77

```

Bir zombi ölünce yapılacak şeyleri Health scriptinde yazmıştık. Bu yüzden bu kodu da Health scriptinin içine yazıyoruz. Yaptığımız şey basit: eğer ölen kişi bir zombi ise (player değilse) PlayerStats scriptindeki _zombiesKilled değişkenini 1 artırıyoruz (bunu yaparken ZombiesKilled adındaki property'i kullanıyoruz).

Artık öldürülen zombi sayısını ekrana yazdırabiliriz.

CombatGui scriptini güncelleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class CombatGui : MonoBehaviour
05 {
06     Health _playerHealth;
07     PlayerStats _playerStats;
08
09     [SerializeField]
10     Texture2D _gameOverImage;
11
12     [SerializeField]
13     Texture2D _winImage;
14

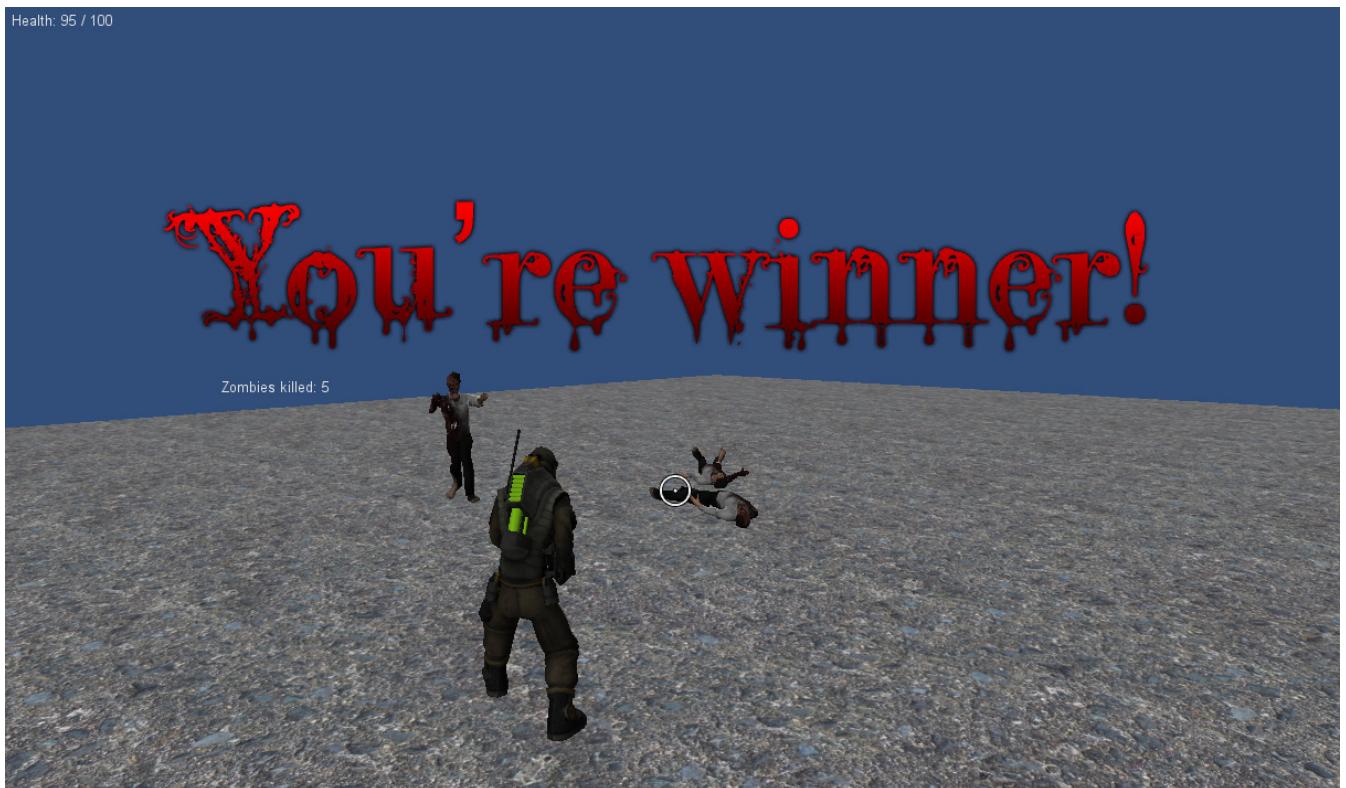
```

```

15 void Start()
16 {
17     GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
18     _playerHealth = playerGameObject.GetComponent<Health>();
19     _playerStats = playerGameObject.GetComponent<PlayerStats>();
20 }
21
22 void OnGUI()
23 {
24     if (_playerHealth.IsDead)
25     {
26         float x = (Screen.width - _gameOverImage.width) / 2;
27         float y = (Screen.height - _gameOverImage.height) / 2;
28         GUI.DrawTexture(new Rect(x, y, _gameOverImage.width, _gameOverImage.height),
29                         _gameOverImage);
30         GUI.Label(new Rect(x + 100, y + 280, 100, 100), "Zombies killed: " + _playerStats.
31                   ZombiesKilled);
32     }
33     else if (GameManager.HasPlayerWon)
34     {
35         float x = (Screen.width - _winImage.width) / 2;
36         float y = (Screen.height - _winImage.height) / 2;
37         GUI.DrawTexture(new Rect(x, y, _winImage.width, _winImage.height), _winImage);
38         GUI.Label(new Rect(x + 100, y + 280, 100, 100), "Zombies killed: " + _playerStats.
39                   ZombiesKilled);
40     }
41 }
42 }
43

```

PlayerGui scriptinde nasıl ekrana sağlığını yazdırıyorsak burada da benzer şekilde GUI.Label kullanarak öldürdüğümüz zombi sayısını ekrana yazdırıyoruz.



Helikopteri Oyuna Ekleme

Belli bir süre geçince oyunu otomatik olarak kazanmanın devri geçti artık. 3 dakikalık süre oyunu kazanmak için tetikleyici olmayacak ama kurtarma helikopterinin yere iniş yapması için tetikleyici olacak. Helikopter yere inip de kapılarını açtığında ona binebileceğiz ve işte o zaman oyunu kazanmış olacağız.

İlk önce 3D helikopter modelini projemize import edelim. Bunun için Winrar arşivinin olduğu yerdeki Helicopter klasörünü kopyalayın ve projenizin olduğu yerdeki "Models" klasörünün içine yapıştırın.

Project panelinden Helicopter modelini tutup sahnenize sürükleyn. Helikopter, karakterin yanında çok küçük kalabilir. Bunu çözmek için modelin Scale Factor'ünü 0.06 yapmayı deneyin.

ÇEVİRMEN EKLEMESİ: Helicopter modelini seçin. Inspector'dan "Rig" sekmesine geçiş yapın ve "Animation Type"ı "Generic"ten "Legacy"e çevirin.



Helikoptere Collider Vermek

Şu anda helikopterin içinden geçebiliyoruz. Bunu birlikte çözelim.

"Helicopter Collider" adında yeni bir empty gameobject oluşturun ve bu objeyi sahnedeki Helicopter objesinin bir child'ı yapın (Bunu yapmanın kısa yolu Hierarchy panelinden **Helicopter** objesini **seçmek** ve menü barından **GameObject-Create Empty Child** yolunu izlemektir). Sonrasında "Helicopter Collider"ın pozisyonunu (0,0,0) yapın.

Ardından Component > Physics > Box Collider yolunu izleyerek "Helicopter Collider'a box collider verin.

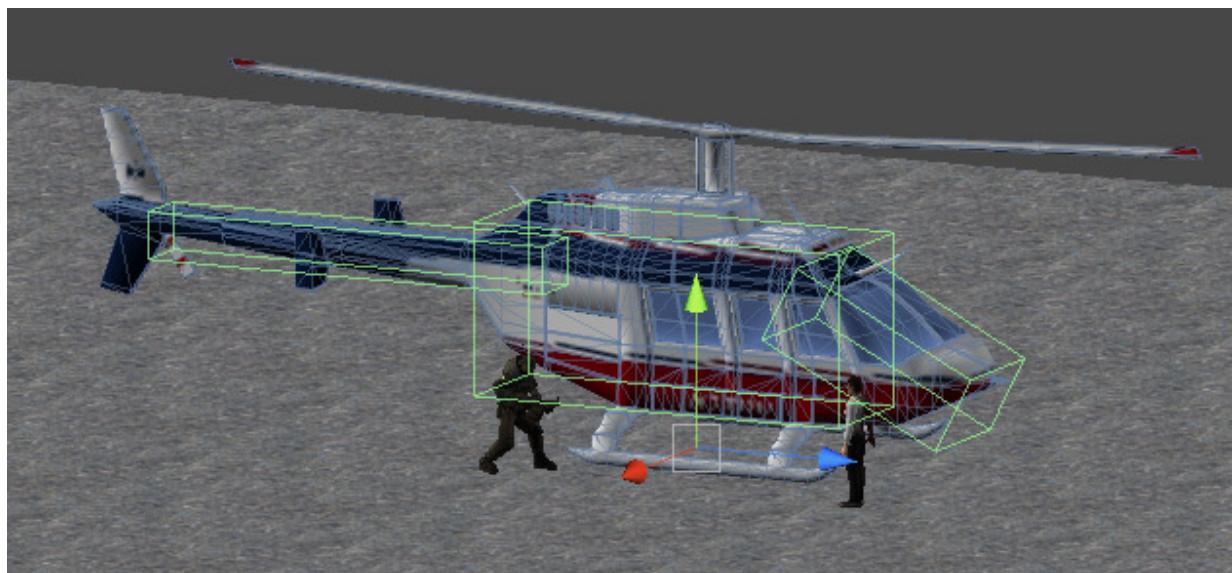
Shift tuşuna basılı tutun ve collider'in kenarlarında beliren noktalardan tutup sürükleyerek onu, helikopterin gövdesini kaplayacak şekilde boyutlandırın (ben Collider'ın Center'ını (0, 2.3, -0.3) ve Size'ını (2, 3, 6.5) yapınca güzel durdu):



Şimdi "Helicopter Collider" objesini CTRL + D ile klonlayın ve klonu, collider'ını kuyruğu kapatacak şekilde ayarlayın (Center'ı (0, 2.7, -6.5) ve Size'ı (1, 1, 6.5) yapınca güzel oturuyor):



"Helicopter Collider"ı bir kere daha klonlayın ve bu sefer collider'ı helikopterin burnunu kapatacak şekilde boyutlandırın (objeyi döndürmeniz gerekebilir)(ben objenin X ekseninde rotation'ını 36 derece yaptım ve collider'ın Center'ını (0, 3.5, 1.48), Size'ını (2, 1, 3) yaptım):



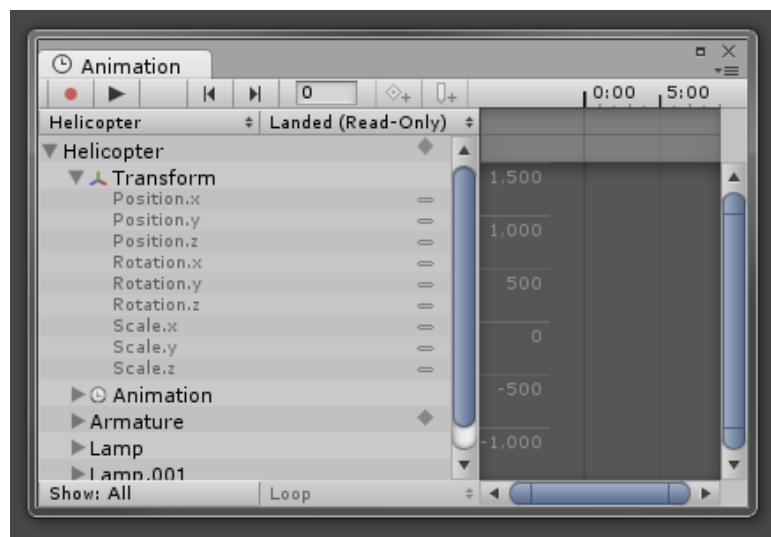
Oyunu çalıştırın. Artık helikopterin içinden geçemememiz lazım.

Helikopterin İniş Yapmasını Ayarlamak

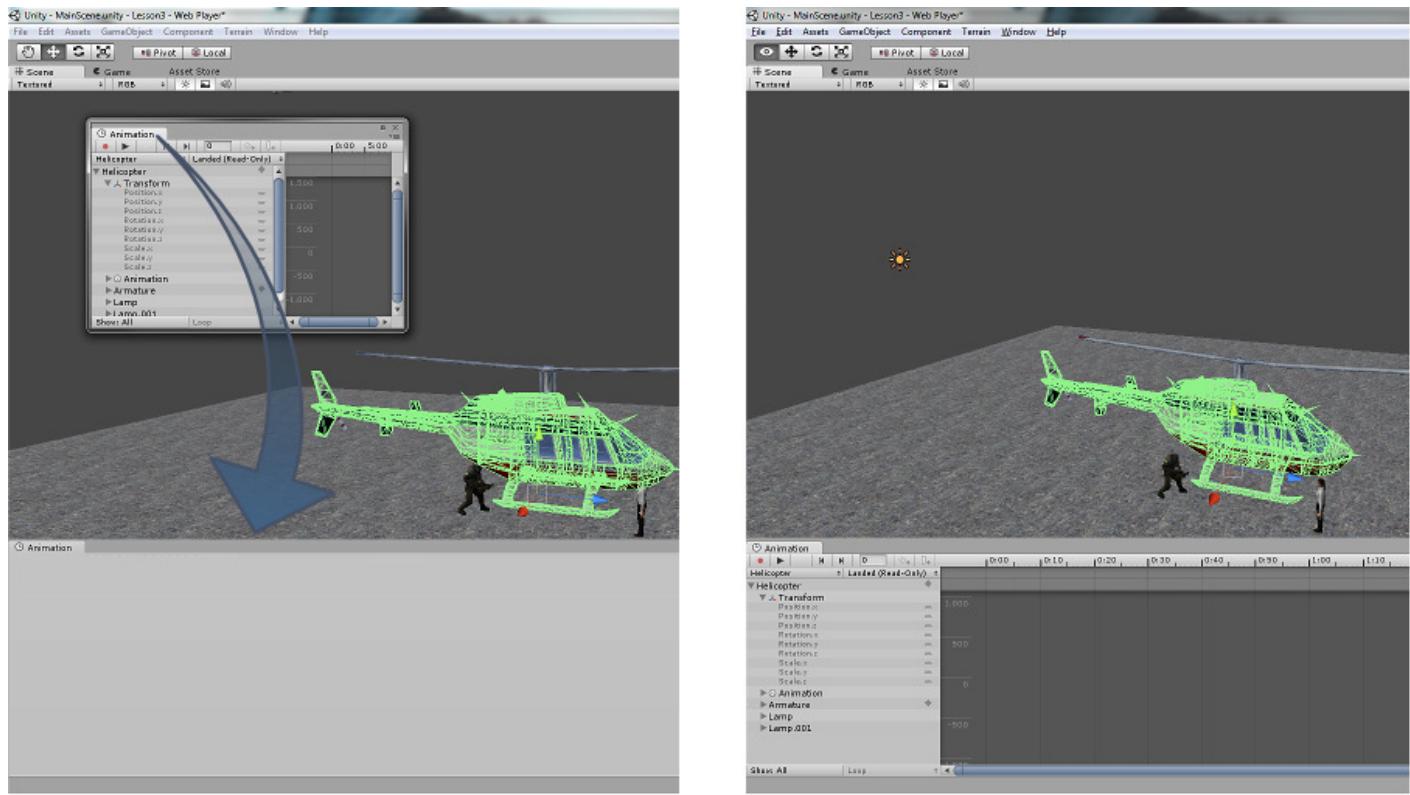
Bunun için kendimiz basit bir animasyon yapacağız.

Animation View'ı Kullanmak

Unity'de animasyon oluşturmak için Animation View kullanılır. **Window > Animation** ile ya da **Ctrl+6** ile bu pencereyi açın:



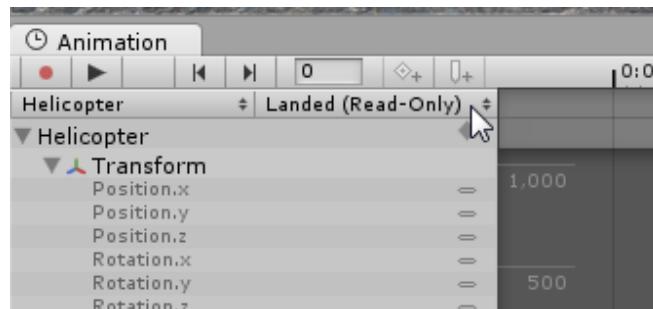
Animation penceresini başlığından (Animation yazan sekme) tutun ve ekranın alt kısmına sürükleyn. Böylece Animation penceresi ekranın alt kısmına sabitlenecek:



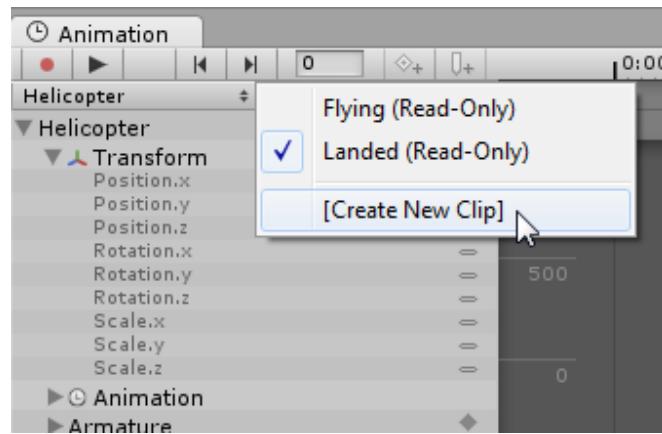
Yeni Bir Animasyon Oluşturmak

Helicopter objesini Hierarchy panelinden seçin. Parent objeyi seçtiğinizden emin olun, yanlışlıkla bir child objeyi seçmeyin.

Animation penceresinde "Landed (Read-Only)" ya da "Flying (Read-Only)" yazan bir yer var. Bu, seçili olan animasyonu temsil eder.



Yeni bir animasyon oluşturmak için onun üzerine tıklayıp “[Create New Clip]”ı seçin:



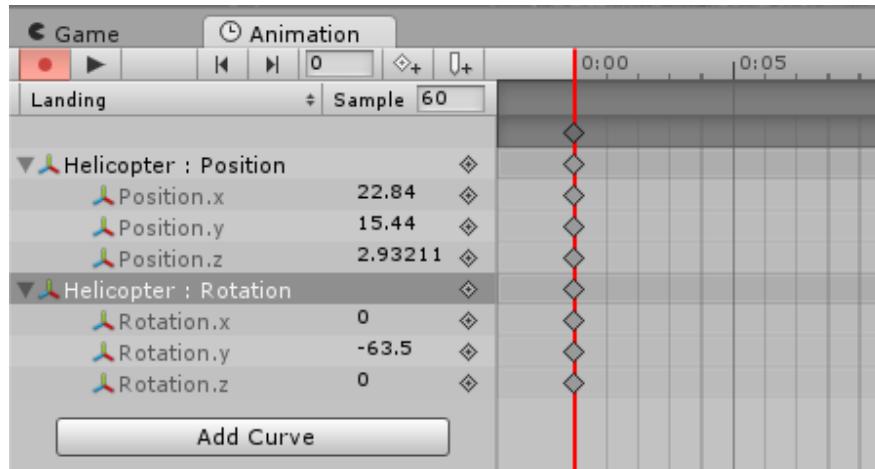
Animasyonu TheGame/Models/Helicopter/ klasörünün içine "Landing.anim" adıyla kaydedin.

ÇEVİRMEN EKLEMESİ: Bende nedense "[Create New Clip]" seçeneği yoktu. Bu yüzden işleri elle hallettim. Eğer sizde de yoksa Project panelinden Helicopter klasörünü seçin ve Create-Animation yolunu izleyin. Animasyona "Landing" adını verin. Son olarak Hierarchy'den Helicopter objesini seçin. Animation component'indeki Animations'in Size'ını 3 yapın ve "Element 2"ye değer olarak "Landing" animasyonunu verin. Artık Animation penceresinde Landing animasyonu da gözükecek.

Animasyonumuzu oluşturmaya hazırız. Kırmızı daire ikonuna sahip butona tıklayın. Buton kırmızı renge bürünecek (ayrıca Play butonu da kırmızı olacak).

Helikopteri havada bir yere taşıyın. Oyunun başında helikopter burada yer alacak.

Animation penceresinde gri noktalar belirecek. Bu noktalar keyframe oluyor. Objenin o özelliklerinin animasyona dahil edildiğini belirtiyorlar:



Animation penceresinin tepesindeki zaman çizelgesini gördünüz mü? Onda istediğiniz yere tıklayın. Kırmızı çizgi oraya zıplayacak. Bu kırmızı çizgi o an animasyonun hangi karesinde olduğunu belirlemekte.



Eğer mouse'nin orta tekerleğini çevirirseniz bu zaman çizelgesine zoom yaparsınız. "5:00" zamanı çizelgede belirene kadar zoom-out yapın. "5:00'a tıkladığımızda animasyonun 5. saniyesine gitmiş oluyoruz.

5. saniyedeyken helikopteri zeminde istediğiniz bir yere indirin.

Helikopterinizin iniş yapması toplam 5 saniye sürüyor. Zaman çizelgesinde bir yere tıklayarak mevcut frame'i değiştirdikçe helikopterin animasyonunun gerçekleştiğini göreceksiniz. İsterseniz daha çok keyframe oluşturarak daha detaylı bir animasyon yaparsınız. Benim için 2 keyframe yeterli.

Kırmızı daire ikonlu butona tekrar tıklayarak animasyon modundan çıkmak.

ÇEVİRMEN EKLEMESİ: Helikoptere animasyon verirken **Animation** component'ini kullanıyoruz. Bu, eski (**Legacy**) animasyon sistemi; artık sizin de bildiğiniz üzere **Mecanim** sistemi kullanılıyor. Animation component'inde kullanılan animasyonların "**Legacy**" animasyon olarak belirtilmesi lazım yoksa Unity "*The AnimationClip 'Landing' used by the Animation component 'Helicopter' must be marked as Legacy.*" uyarısı verir. Bunu yapmanın yolu basit: **Project** panelinden **Landing** animasyonunu seçin. **Inspector**'a sağ tıklayıp "**Debug**" moduna geçiş yapın ve "**Animation Type**"'ın değerini 2'den 1'e çevirin. Inspector'a tekrar sağ tıklayıp "**Normal**" moda geri geçiş yapın. Bu kadar!

Helikopter Scripti

Helikopterin sahip olduğu animasyonları kontrol etmek için bir script yazalım.

Yeni bir C# scripti oluşturup ismini "Helicopter" yapın. Scripti Helicopter objesine component olarak verin. Ardından kodu şöyle düzenleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Helicopter : MonoBehaviour
05 {
06     void Start()
07     {
08         animation["Flying"].wrapMode = WrapMode.Loop;
09         animation["Landed"].wrapMode = WrapMode.ClampForever;
10         animation["Landing"].wrapMode = WrapMode.ClampForever;
11     }
12 }
```

Helikopterin üç animasyonu var: "Flying" animasyonu pervaneyi çevirmeye yararken "Landed" animasyonu helikopterin kapılarını açmaya yarar. Az önce oluşturduğumuz "Landing" animasyonu ise helikopteri yere indirmeye yarıyor.

"Flying" animasyonunun wrapMode'unu Loop yapıyoruz. Bunun anlamı bu animasyon bittiği zaman başa saracak ve oynamaya devam edecek (loop yapacak). Diğer iki animasyonun wrapMode'unu ise ClampForever yaptık. Bunun anlamı ise o animasyonlar bitikten sonra animasyonun son frame'inde kalacaklar, tekrar başa sarmayacaklar. Helikopterin yere indikten sonra tekrar havaya işinlanması garip dururdu herhalde.

Oyunun başında, pervanelerin dönmelerini sağlayan "Flying" animasyonunun oynamasını sağlayalım:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Helicopter : MonoBehaviour
05 {
06     void Start()
```

```

07    {
08        animation["Flying"].wrapMode = WrapMode.Loop;
09        animation["Landed"].wrapMode = WrapMode.ClampForever;
10        animation["Landing"].wrapMode = WrapMode.ClampForever;
11
12        animation.Blend("Flying", 1.0f, 0.01f);
13    }
14 }
```

Animation component'inin **Blend()** fonksiyonunu kullandık. Bu fonksiyon iki animasyonun bir arada oynamasını sağlar (tipki karakterin ileri-sağda koşarken ileri koşma ve sağa koşma animasyonlarının beraber oynaması gibi). Eğer Animation component'inin **Play()** fonksiyonunu kullanırsak diğer animasyonlar otomatik olarak durdurulur ve sadece "Flying" animasyonu oynardı. Biz de ne "Landed" animasyonunu ne de "Landing" animasyonunu oynatamazdık.

Blend fonksiyonunda iki parametre daha var. İlk parametre olan 1.0f değeri animasyonun diğer animasyonlar ile ne kadar harmanlanacağını belirler. 1.0f değeri verirsek animasyon olduğu gibi harmanlanır. İkinci parametre olan 0.01f ise animasyonun kaç saniye içinde harmanlanacağını belirtir. 0.01f verdigimiz için animasyon anında diğer animasyonlar ile harmanlanır ama eğer 5.0f gibi bir değer verseydik yavaş yavaş harmanlanacaktı (**Ben bu parametreleri değiştirince bir fark gözlemleyemedim açıkçası**).

ÇEVİRMEN EKLEMESİ: Oyunu test ederseniz helikopterin kapılarının açıldığını göreceksiniz ama biz öyle bir komut vermedik. Bunun sebebi **Animation** component'inde "**Play Automatically**" seçeneğinin seçili olması. Onun başındaki işaret kaldırın.

Şimdi "Flying" animasyonu ile "Landing" animasyonunu harmanlayalım ve böylece helikopterin pervaneleri dönerken aynı zaman yere iniş yapmasını sağlayalım:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Helicopter : MonoBehaviour
05 {
06     void Start()
07     {
08         animation["Flying"].wrapMode = WrapMode.Loop;
09         animation["Landed"].wrapMode = WrapMode.ClampForever;
10         animation["Landing"].wrapMode = WrapMode.ClampForever;
11
12         animation.Blend("Landing", 1.0f, 0.01f);
13         animation.Blend("Flying", 1.0f, 0.01f);
14     }
15 }
```

Oyunu test edin. İki animasyonun birlikte oynaması lazım.

Helikopter yere indiğinde ise kapılarının açılmasını, yani "Landed" animasyonunun devreye girmesini istiyoruz.

"Landed" animasyonunun sadece "Landing" animasyonu son frame'ine ulaştığında (helikopter yere indiğinde) oynatılmasını sağlayalım:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Helicopter : MonoBehaviour
```

```

05 {
06     void Start()
07     {
08         animation["Flying"].wrapMode = WrapMode.Loop;
09         animation["Landed"].wrapMode = WrapMode.ClampForever;
10         animation["Landing"].wrapMode = WrapMode.ClampForever;
11
12         animation.Blend("Landing", 1.0f, 0.01f);
13         animation.Blend("Flying", 1.0f, 0.01f);
14     }
15
16     void Update()
17     {
18         if (animation["Landing"].normalizedTime >= 1.0f)
19         {
20             animation.Blend("Landed", 1.0f, 0.01f);
21         }
22     }
23 }
```

18. satırda "Landing" animasyonunun normalizedTime'ının 1.0'dan büyük olup olmadığına bakıyoruz. normalizedTime, animasyonun hangi safhasında olduğumuzu belirler. Değeri 0.0 ise animasyon ilk frame'de, 1.0 ise son frame'de, 0.5 ise tam ortadaki frame'dedir.

Eğer "Landing" animasyonunun sonuna gelmişsek 20. satırdaki kod ile "Landed" animasyonunu diğer animasyonlarla harmanlıyoruz (Blend).

Oyunu çalıştırın. Helikopter yere inince kapıları açılacak.

Helikopterin İstediğimiz Zaman İnmesini Sağlamak

Şu anda oyun başlayınca helikopter yere inmeye başlıyor. "Landing" animasyonunu oyunun başında durdurmalı (pause) ve yeterince zaman geçtikten sonra oynatmalıyız:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Helicopter : MonoBehaviour
05 {
06     void Start()
07     {
08         animation["Flying"].wrapMode = WrapMode.Loop;
09         animation["Landed"].wrapMode = WrapMode.ClampForever;
10         animation["Landing"].wrapMode = WrapMode.ClampForever;
11
12         animation.Blend("Landing", 1.0f, 0.01f);
13         animation.Blend("Flying", 1.0f, 0.01f);
14
15         animation["Landing"].speed = 0;
16     }
17
18     public void Land()
19     {
20         animation["Landing"].speed = 1;
21     }
22
23     void Update()
```

```

24     {
25         if (animation["Landing"].normalizedTime >= 1.0f)
26         {
27             animation.Blend("Landed", 1.0f, 0.01f);
28         }
29     }
30 }
```

Animasyonun speed'ini (hız) 0.0 yaptığımız zaman animasyon duruyor ve geri 1.0 yaptığımız zaman oynamaya devam ediyor. Eğer Land() fonksiyonunda hızı 2.0 yapsaydık animasyon iki kat hızlı oynar, -1.0 yapsaydık baştan sona değil sondan başa oynardı.

Land fonksiyonu public bir fonksiyon. Helikopterin yere ne zaman inmesini istiyorsak bu public fonksiyonu o zaman çağrırmamız yeterli.

GameManager scriptini açın ve içeriğinin büyük kısmını silin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class GameManager : MonoBehaviour
05 {
06     static bool _hasPlayerWon = false;
07
08     public static bool HasPlayerWon { get{ return _hasPlayerWon; } }
09 }
```

Geriye sadece _hasPlayerWon değişkeni ve HasPlayerWon property'si kalsın.

Helicopter scriptini açıp şöyle güncelleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Helicopter : MonoBehaviour
05 {
06     [SerializeField]
07     float _secondsToLand = 180.0f;
08
09     void Start()
10     {
11         animation["Flying"].wrapMode = WrapMode.Loop;
12         animation["Landed"].wrapMode = WrapMode.ClampForever;
13         animation["Landing"].wrapMode = WrapMode.ClampForever;
14
15         animation.Blend("Landing", 1.0f, 0.01f);
16         animation.Blend("Flying", 1.0f, 0.01f);
17
18         animation["Landing"].speed = 0;
19
20         Invoke("Land", _secondsToLand);
21     }
22
23     public void Land()
24     {
25         animation["Landing"].speed = 1;
26     }
27
28     void Update()
```

```

29     {
30         if (animation["Landing"].normalizedTime >= 1.0f)
31     {
32         animation.Blend("Landed", 1.0f, 0.01f);
33     }
34 }
35 }
```

Invoke() fonksiyonu ile _secondsToLand saniye sonra Land fonksiyonunu çağrıyoruz, yani helikoptere inmesini söylüyoruz.

Helikoptere Ulaşınca Oyunu Kazanmak

Geriye oyuncunun helikoptere ulaşınca oyunu kazanması kaldı. Bunun için “DoorsRight” adında yeni bir empty gameobject oluşturup bunu Helicopter objesinin child objesi yapın.

DoorsRight'ın pozisyonunu (0,0,0) yapın. Ardından DoorsRight'a box collider verin (Component > Physics > Box Collider).



Box Collider'ı helikopterin sağ taraftaki kapılarını dışarıdan kaplayacak şekilde konumlandırın/boyutlandırın (**Center (1, 2, 1)** ve **Size (1, 2, 4)** güzel duruyor). Oyuncu bu collider ile temas edince oyunu kazanacak.

Şimdi Inspector'dan Box Collider'daki “Is Trigger” seçeneğini işaretleyin ki karakter bu collider ile temas edince içinden geçebilsin.

Oyuncunun kapıdaki bu collider ile temas edip etmediğini anlamak için yeni bir C# scripti oluşturun. İsmini “HelicopterDoors” yapın ve kodu düzenleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class HelicopterDoors : MonoBehaviour
05 {
06     void OnTriggerEnter(Collider c)
07     {
08         if (c.transform.root.tag == "Player")
09         {
10         }
11     }
12 }
```

Yaptığımız şey basit: "Is Trigger"ı işaretli olan collider'ımız ile temas eden objenin tag'ının "Player" olup olmadığına bakıyoruz. Eğer Player ise oyunu kazanacağız.

Kazanma işlemi için önce GameManager scriptinde şu değişikliği yapın:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class GameManager : MonoBehaviour
05 {
06     static bool _hasPlayerWon = false;
07
08     public static bool HasPlayerWon { get{ return _hasPlayerWon; } }
09
10    public static void WinPlayer()
11    {
12        _hasPlayerWon = true;
13    }
14 }
```

HelicopterDoors scriptine geri dönün ve eğer Player ile temas etmişsek WinPlayer fonksiyonunu çağırın:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class HelicopterDoors : MonoBehaviour
05 {
06     void OnTriggerEnter(Collider c)
07     {
08         if (c.transform.root.tag == "Player")
09         {
10             GameManager.WinPlayer();
11         }
12     }
13 }
```

HelicopterDoors scriptini DoorsRight objesine verip oyunu test edin. Sağ kapıdaki collider ile temas edince oyunu kazanacaksınız. Sol kapı için yeni bir obje oluşturup (DoorsLeft) ona da "Is Trigger"lı Collider ile HelicopterDoors scriptini vermeyi size bırakıyorum.



Görsel 13.1: Oyuncu helikopterin kapısındaki collider ile temas edince oyunu kazanıyor.

Özet Geçecek Olursak...

Bu bölümde bir fonksiyonu Invoke ile istediğimiz kadar saniye sonra çalıştırmayı ve Unity'de basit animasyonlar oluşturmayı gördük.

Oyunumuz aşağı yukarı tamamlandı. Sonraki derslerde oyuna bazı cerez eklemeler yapacağız; ana menü gibi...

ÇEVİRMEN EKLEMESİ: Bence artık sahnede yer alan üç tane zombi klonunu silin. Zaten oyun ilerledikçe spawn oluyorlar, o üçünün orada durması gereksiz.

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

Unity 3D İle Oyun Programlama

Bölüm 14: Ana Menü Oluşturmak



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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Hemen her oyunda oyun açıldığı zaman karşımıza bir ana menü gelir. Buradan oyunu çalıştırabilir veya oyunun ses, görsel gibi ayarlarını değiştirebiliriz. Bizim ana menümüz şöyle duracak:



Görsel 14.1: Ana menüden bir görüntü

Gerekli Dosyaları Import Etmek

File-New Scene ile yeni bir sahne oluşturun. Sahneyi Scenes klasörünün içine “MainMenu” adıyla kaydedin.

Winrar arşivinin olduğu yerdeki "Gui" klasöründe yer alan şu dosyaları kendi projenizdeki "Gui" klasörüne import edin:

1. ButtonBg.png
2. ButtonBgPressed.png
3. ButtonNewGame.png
4. ButtonOptions.png
5. ButtonQuit.png
6. MainMenuBg.png
7. Title.png



Bu resim dosyalarını ana menüde kullanacağınız (sizin Gui klasörünüzde henüz "GUISkin" asset'i yer almayacaktır, dert etmeyin. Dersin ilerleyen kısımlarında oluşturuyoruz onu). **Import ettiğiniz texture'lerin her birinin "Texture Type"ını “GUI (Editor \ Legacy)” yapın.**

“MainMenuGui” adında yeni bir C# scripti oluşturun:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _mainMenuBg;
08
09     void OnGUI()
10     {
11         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
12     }
13 }
```

Tek yaptığımız şey arkaplan resmini ekranın genişliğini (Screen.width) ve yüksekliğini (Screen.height) kaplayacak şekilde ekrana çizdirmek.

MainMenuGui scriptini "Main Camera" objesine verin. Inspector'dan "Main Menu Bg"ye değer olarak MainMenuBg texture'sini verin.

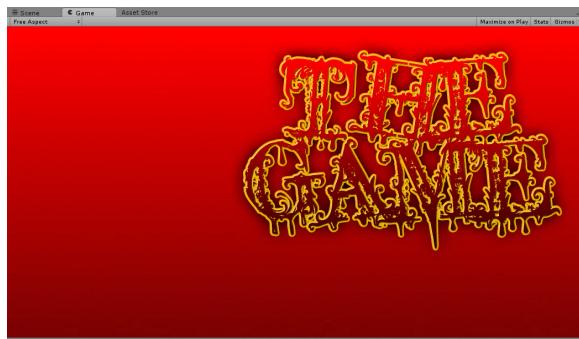
Oyunu çalıştırın. Basit bir kırmızı arkaplanın tüm ekranı kapladığını göreceksiniz.

Scripti biraz daha genişletelim:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _mainMenuBg;
08
09     [SerializeField]
10     Texture2D _title;
11
12     void OnGUI()
13     {
14         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
15
16         GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
17                         _title);
18     }
19 }
```

Yeni eklediğimiz kod ile ekranın sağ üstünde oyunun ismini barındıran texture'yi (title) çizdiriyoruz. Texture'yi çizdirdiğimiz Rect'in aldığı ilk parametreye dikkat edin: Screen.width - _title.width - 20. Bunun anlamı, texture ekranın sağ kenarından 20 pixel uzakta çizilecektir. İkinci parametrede ise direkt 20 değerini veriyoruz. Bunun anlamı da texture ekranın üst kenarından 20 pixel uzakta çizilecektir.

Şimdi Inspector'dan "Title"a değer olarak Gui klasöründeki "Title" asset'ini verip oyunu çalıştırın:

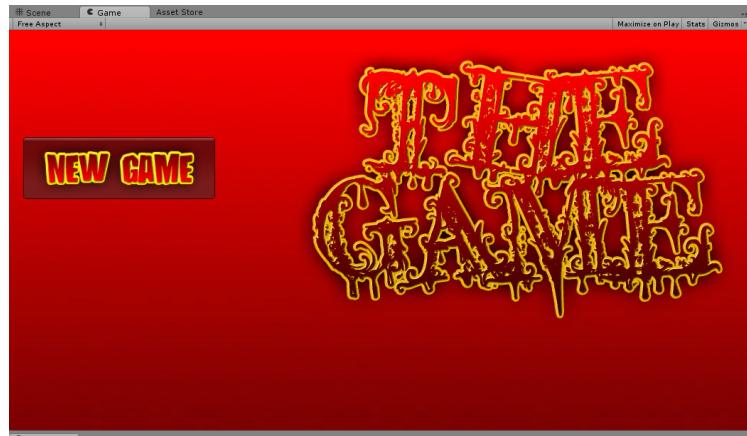


Sıra geldi butonları menüye eklemeye:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _mainMenuBg;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _newGameButton;
14
15     void OnGUI()
16     {
17         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
18
19         GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
20                         _title);
21
22         if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height), _newGameButton))
23         {
24         }
25     }
26 }
```

GUI.Button komutu sadece OnGUI fonksiyonunun içindeyken çalışır ve ekrana tıklanabilir bir buton çizdirmeye yarar. Eğer butona tıklanırsa fonksiyon true döndürür. Bu yüzden GUI.Button'u bir "if" koşulunun içine yazıyoruz; eğer butona tıklanırsa if'in içine yazdığımız kodlar çalışacak.

"ButtonNewGame" asset'ini "Main Menu Gui" component'indeki "New Game Button"a değer olarak verip oyunu çalıştırın. Ekranda tıklanabilir bir buton görünecek:



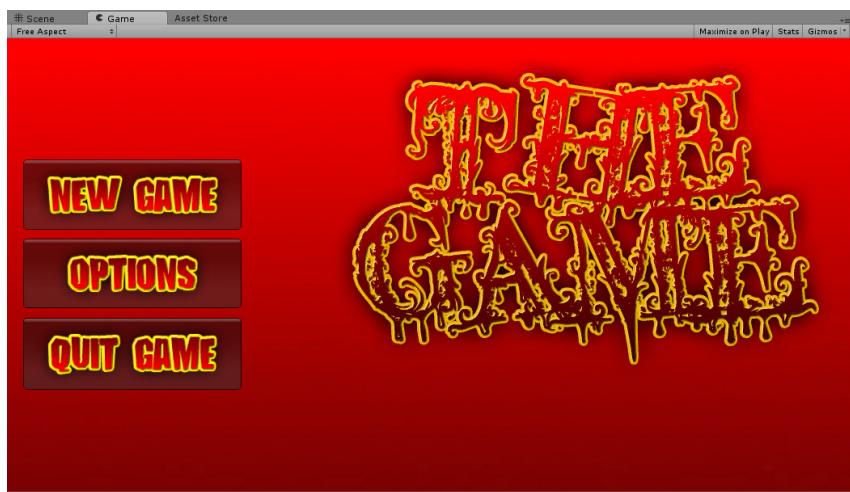
Hazır elimiz değişmişken diğer butonları da menüye ekleyelim:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _mainMenuBg;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _newGameButton;
14
15     [SerializeField]
16     Texture2D _optionsButton;
17
18     [SerializeField]
19     Texture2D _quitButton;
20
21     void OnGUI()
22     {
23         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
24
25         GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
26                         _title);
27
28         if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height), _newGameButton))
29         {
30         }
31
32         if (GUI.Button(new Rect(20, 150 + 88 + 10, 270, _optionsButton.height), _optionsButton))
33         {
34         }
35
36         if (GUI.Button(new Rect(20, 150 + (2*(88 + 10)), 270, _quitButton.height), _quitButton))
37         {
38         }
39     }
40 }
```

Rect'in ikinci parametresiyle oynayarak butonların farklı yüksekliklerde oluşturulmasını sağlıyoruz.

ÇEVİRMEN EKLEMESİ: $150 + 88 + 10$ 'daki 150, 88 ve 10'un ne anlama geldiğini merak edebilirsiniz. 150, butonlar ile ekranın üst kenarı arasındaki boşluğun kaç pixel olduğunu belirliyor. İki buton arasındaki dikey boşluk ise (bir butonun yüksekliği) + (üstteki butonun alt kenarı ile alttaki butonun üst kenarı arasındaki boşluk) olarak hesaplanmaktadır. Bizim butonlarda kullandığımız texture'lerin yüksekliklerine Inspector'dan bakacak olursanız 87, 86 ve 89 pixel olduklarını göreceksiniz. Yazar bu sayıları ortalayıp "bir butonun yüksekliği"ni 88 pixel olarak ele almış. "Üstteki butonun alt kenarı ile alttaki butonun üst kenarı arasındaki boşluk"un ise 10 pixel olmasını uygun görmüş.

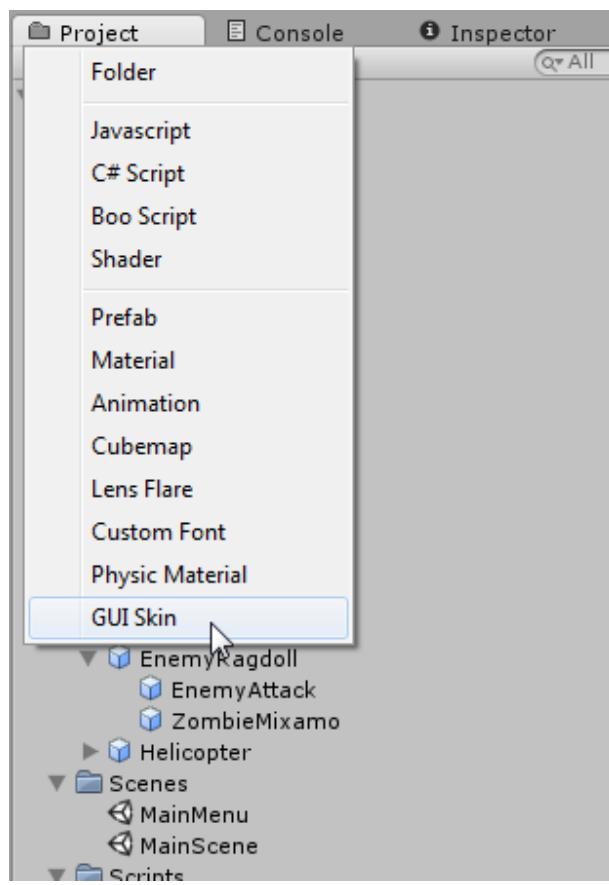
Inspector'daki "Options Button" ve "Quit Button"a uygun texture'leri değer olarak verip oyunu test edin:



Butonların Arkaplanını Değiştirmek

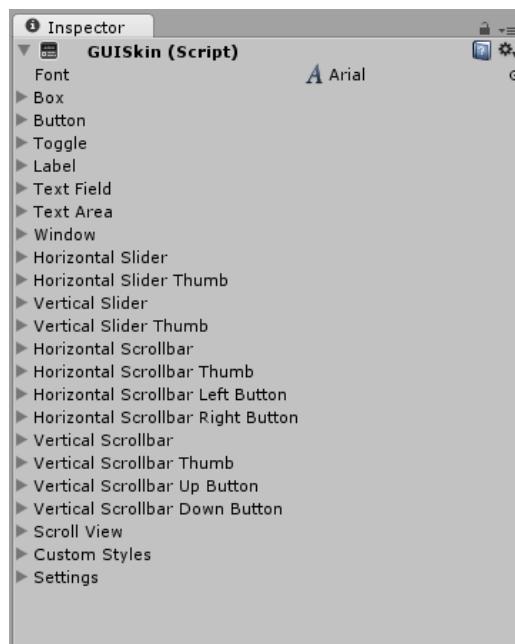
Butonların siyah arkaplanı menünün geri kalanıyla tarz olarak uyuşmuyor. Aslına bakarsanız bu siyah arkaplanın stilini değiştirmek bizim elimizde. Bunu yapmak için *GUI Skin* türünde bir asset'ten faydalanağınız. Bir GUI Skin ekranda çizdirilen GUI elemanlarının nasıl duracağını, kullanacakları fontu ve bunun gibi şeyleri belirler. Yani arayüzün stilini belirler.

Project panelinde "Gui" klasörünün içinde "Create-GUI Skin" yolunu izleyerek yeni bir GUI Skin oluşturun:



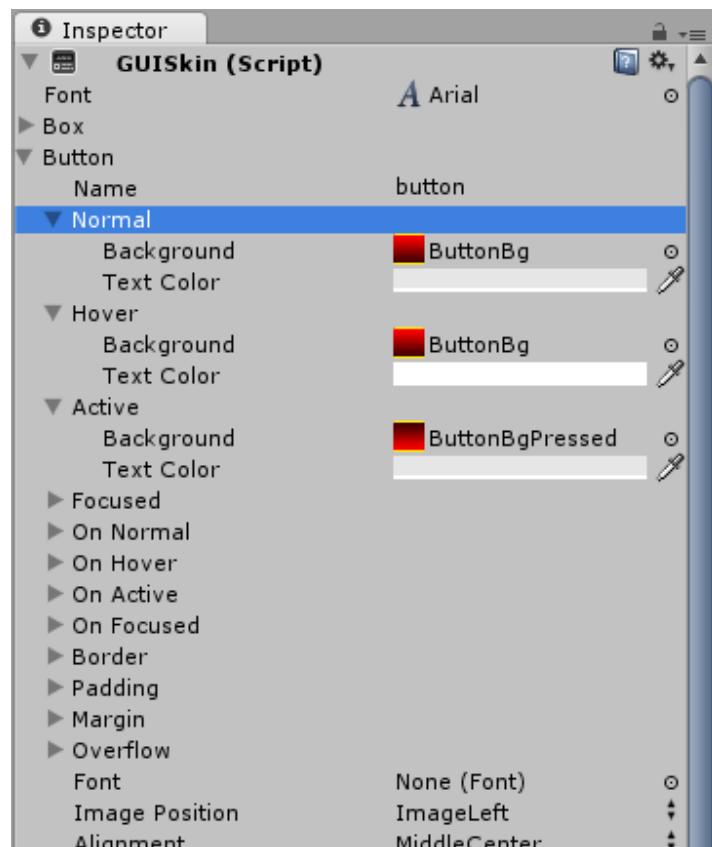
Oluşan dosyanın adını "GuiSkin" olarak değiştirin.

Oluşturduğunuz GUI Skin dosyasını seçin. Inspector'da çeşitli GUI elemanları için çeşitli ayarlar gözükecek. Buradaki her bir elemana bir "GUI Style" adı verilir.



Biz butonların stilini değiştirmek istiyoruz. Bunun için "Button"un yanındaki üçgene tıklayarak buton stilinin ayarlarının gözükmesini sağlayın.

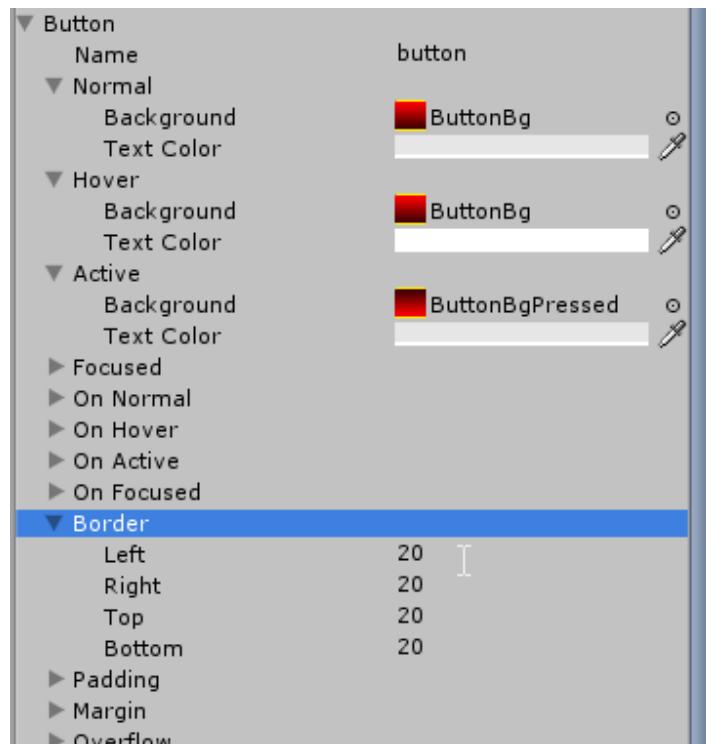
Buradaki "Normal", "Hover" ve "Active"in "Background"una değer olarak şu texture'leri verin:



"Normal" butonun normal stilini, "Hover" fare imlecini butonun üzerine getirince butonun sahip olacağı stili ve "Active" de butona tıklayınca butonun sahip olacağı stili belirler. Bu stillerde yer alan "Background" değeri butonun o anda arkaplanında çizilecek resim dosyasını temsil eder.

Biz, buton normal dururken veya fare butonun üzerindeyken arkaplana "ButtonBg" texture'sini, butona tıklayınca ise arkaplana "ButtonBgPressed" texture'sini çizdireceğiz.

Şimdi "Border"a gelin. Butonun ebatları arkaplan texture'sinin ebatlarından büyükse arkaplanın nasıl genişletileceği burada belirlenir. Buradaki tüm değerleri 20 yapın:



Artık bu GUI Skin'i kendi menümüzde kullanabiliriz. Bunun için MainMenuGui scriptini düzenleyelim:

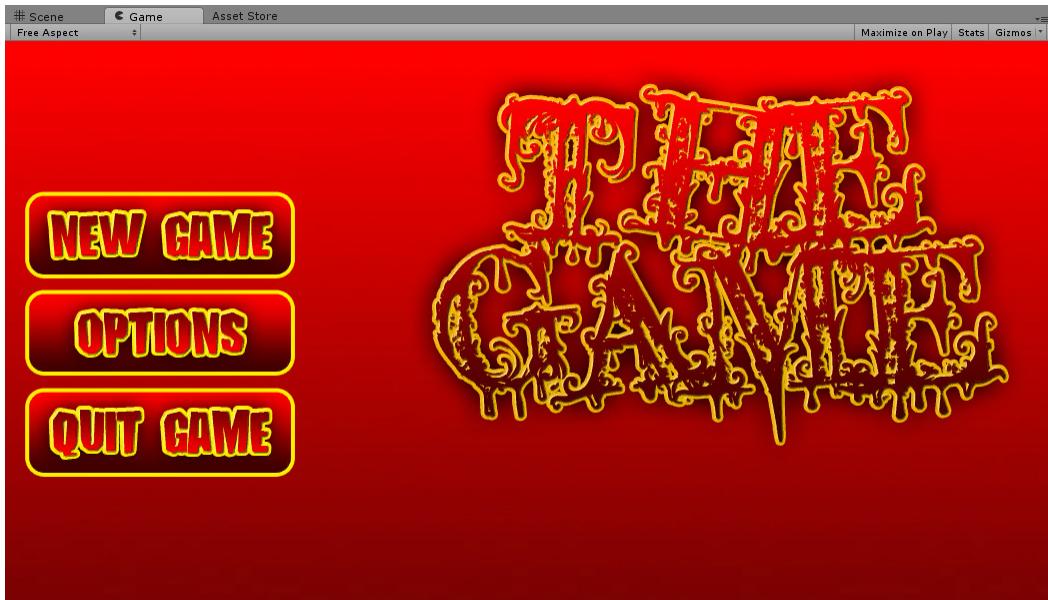
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _mainMenuBg;
11
12     [SerializeField]
13     Texture2D _title;
14
15     [SerializeField]
16     Texture2D _newGameButton;
17
18     [SerializeField]
19     Texture2D _optionsButton;
20
21     [SerializeField]
22     Texture2D _quitButton;
23
24     void OnGUI()
25     {
26         GUI.skin = _guiSkin;
27
28         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
29
30         GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
31                         _title);
```

```

32
33     if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height), _newGameButton))
34     {
35     }
36
37     if (GUI.Button(new Rect(20, 150 + 88 + 10, 270, _optionsButton.height), _optionsButton))
38     {
39     }
40
41     if (GUI.Button(new Rect(20, 150 + (2*(88 + 10)), 270, _quitButton.height), _quitButton))
42     {
43     }
44 }
45 }
```

"GUI.skin = _guiSkin;" diyerek OnGUI fonksiyonunda oluşturduğumuz arayüz (GUI) elemanlarının _guiSkin değişkeninde tutulan GUI Skin'deki stilleri kullanmasını sağlıyoruz.

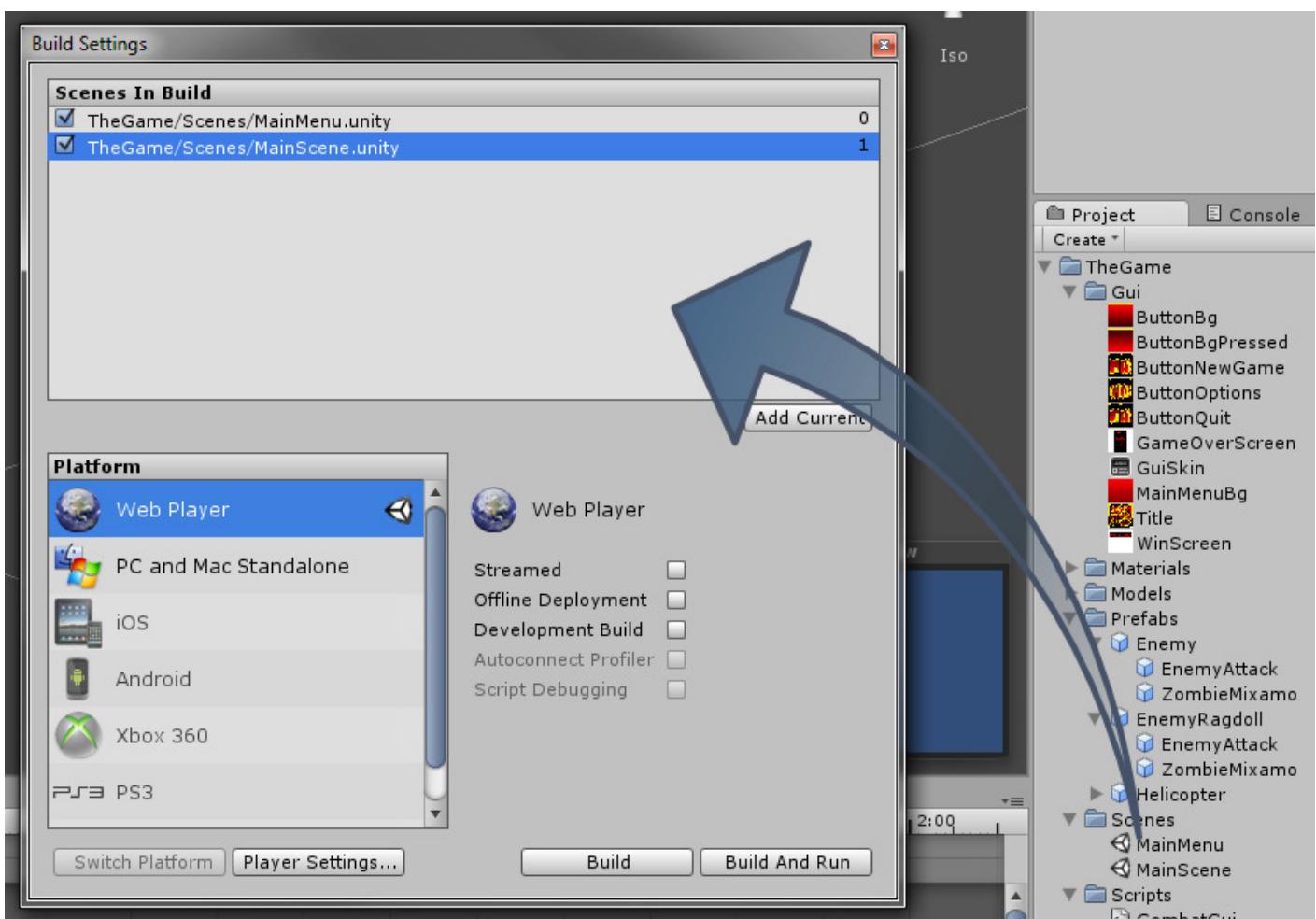
Inspector'dan "Gui Skin"e oluşturduğunuz GUI Skin asset'ini değer olarak verip oyunu çalıştırın:



New Game Butonuna Basınca Yapılacaklar

Oyuncu New Game butonuna basınca oyunun olduğu sahneye (scene)(level) geçiş yapmamız lazım. İlk önce Unity'e, oluşturduğumuz sahnelerden hangilerinin oyuna gerçekten dahil olduğunu söylememiz lazım. **File > Build Settings...** yolunu izleyin.

Build Settings penceresi gelecek. Burada “Scenes In Build” adında boş bir liste var. Yapmanız gereken Scenes klasöründe yer alan MainMenu ile MainScene'i bu listeye sürüklemek (Listedeki ilk scene, build edilen oyun başladığında açılacak olan scene'dir. Bu yüzden eğer listenin başında MainMenu değil de MainScene varsa MainMenu'yu sürükle-bırak yaparak listenin başına alın):



MainMenuGui scriptini açıp şu eklemeyi yapın:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _mainMenuBg;
11
12     [SerializeField]
13     Texture2D _title;
14
15     [SerializeField]
16     Texture2D _newGameButton;
17
18     [SerializeField]
19     Texture2D _optionsButton;
20
21     [SerializeField]
22     Texture2D _quitButton;
```

```

23
24     void OnGUI()
25     {
26         GUI.skin = _guiSkin;
27
28         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
29
30         GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
31                         _title);
32
33         if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height), _newGameButton))
34         {
35             Application.LoadLevel("MainScene");
36         }
37
38         if (GUI.Button(new Rect(20, 150 + 88 + 10, 270, _optionsButton.height), _optionsButton))
39         {
40         }
41
42         if (GUI.Button(new Rect(20, 150 + (2*(88 + 10)), 270, _quitButton.height), _quitButton))
43         {
44         }
45     }
46 }
```

Application.LoadLevel fonksiyonu başka bir scene'e geçiş yapmaya yarar. Mevcut scene'deki objeler bu esnada silinirler; yani diğer bölüme geçmezler. Bu fonksiyonun çalışması için fonksiyona parametre olarak girilen scene'in Build Settings'teki "Scenes In Build" listesinde yer olması lazım. Biz de zaten bu yüzden her iki scene'i de listeye ekledik.

Oyunu çalıştırıp "New Game"e basın ve mevcut scene'in değiştiğine kendi gözlerinizle tanık olun!

Quit Butonuna Basınca Yapılacaklar

Oyunu sonlandıracağız. Bunu yapmak çok basit:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _mainMenuBg;
11
12     [SerializeField]
13     Texture2D _title;
14
15     [SerializeField]
16     Texture2D _newGameButton;
17
18     [SerializeField]
19     Texture2D _optionsButton;
```

```

20
21 [SerializeField]
22 Texture2D _quitButton;
23
24 void OnGUI()
25 {
26     GUI.skin = _guiSkin;
27
28     GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
29
30     GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
31                     _title);
32
33     if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height), _newGameButton))
34     {
35         Application.LoadLevel("MainScene");
36     }
37
38     if (GUI.Button(new Rect(20, 150 + 88 + 10, 270, _optionsButton.height), _optionsButton))
39     {
40     }
41
42     if (GUI.Button(new Rect(20, 150 + (2*(88 + 10)), 270, _quitButton.height), _quitButton))
43     {
44         Application.Quit();
45     }
46 }
47 }
```

Bu kod editörde bir işe yaramaz ama oyunu örneğin .EXE olarak build ederseniz işte o zaman işe yarar.

Ayarlar (Options) Menüsünü Oluşturmak

Geriye sadece Options menüsünü oluşturmak kaldı. Bu menüden oyunun zorluğunu (difficulty) ve ses düzeyini (volume) değiştirebileceğiz. Yapacağımız Options menüsünden bir görüntü:



"OptionsGui" adında yeni bir C# scripti oluşturun:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     void OnGUI()
10    {
11        GUI.skin = _guiSkin;
12    }
13 }
```

MainMenuGui'de yaptığımız gibi, GUI elemanlarının oluşturduğumuz GUI Skin'deki stilleri kullanmasını sağlıyoruz.

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     void OnGUI()
10    {
11        GUI.skin = _guiSkin;
12
13        GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
14    }
15 }
```

Ardından ekrana, kenarlardan 15 pixel boşluk kalacak şekilde, basit bir kutucuk çizdiriyoruz.

Scripti "Main Camera"ya verin, "Gui Skin"e değerini atayın ve oyunu test edin.

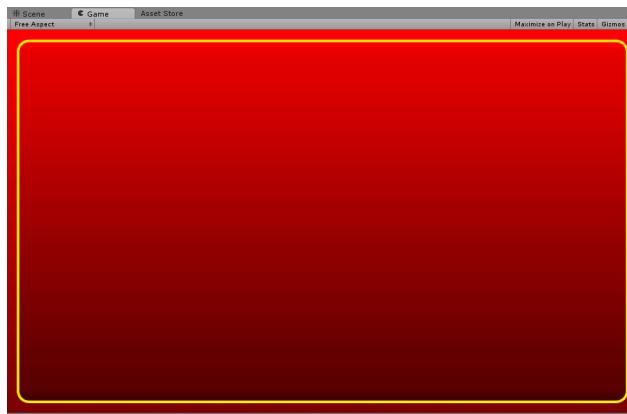
Kutucuğu siyah yarı-saydam bir arkaplana sahip olarak göreceksiniz; bizim bu siyah arkaplanı değiştirmemiz lazım:



Project panelinden GuiSkin asset'ını seçin. Inspector'dan "Box"ı genişletin. Buradaki "Normal" isimli stil bizi ilgilendiren şey. "Hover" ve "Active" önemli değil çünkü bir kutucuğa tıklamak gibi birsey mümkün değil.

ButtonBg texture'sini, "Normal" stilinin "Background"una atayın. İşlem tamamlanınca "Border"daki değerlerin hepsini 20 yapın; tıpkı "Button"da yaptığımız gibi...

Oyunu çalıştırın. Kutucuk çok daha güzel görünecek:



Şimdi Winrar arşivinin ordaki "Gui" klasöründen projenizdeki "Gui" klasörüne şu dosyaları import edin:

1. ButtonOk.png
2. CheckBoxChecked.png
3. CheckBoxUnchecked.png
4. SliderBg.png
5. SliderThumb.png
6. TextEasy.png
7. TextHard.png
8. TextNormal.png
9. TitleDifficulty.png
10. TitleOptions.png
11. TitleSound.png

Tüm bu texture'lerin de "Texture Type"ını "GUI (Legacy \ Editor)" yapmayı unutmayın.

Ardından şu kodu OptionsGui scriptine ekleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     void OnGUI()
13     {
14         GUI.skin = _guiSkin;
15 }
```

```

16     GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
17
18     GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
19 }
20 }
```

Inspector'dan "Title" a değer olarak "TitleOptions" resmini verin ve oyunu test edin:



Şimdi kodu şöyle güncelleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     void OnGUI()
16     {
17         GUI.skin = _guiSkin;
18
19         GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
20
21         GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
22
23         if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.height - 30 - _ok.height
24             - 10, _ok.width, _ok.height), _ok))
25         {
26
27     }
28 }
```

Options menüsünden geri Main Menu'ye dönmek için OK yazan bir buton oluşturuyoruz. Rect koordinatlarını öyle veriyoruz ki ekran çözünürlüğü ne olursa olsun OK butonu hep ekranın sağ altında yer alıyor.

Inspector'dan yeni değişkene değer olarak "ButtonOk" texture'sini verin:



Şimdi Options menüsünden gerçekten de Main Menu'ye dönmek için gerekli kodu yazalım:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     static bool _isOpen = true;
16
17     void OnGUI()
18     {
19         if (!_isOpen)
20         {
21             return;
22         }
23
24         GUI.skin = _guiSkin;
25
26         GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
27
28         GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
29
30         if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.height - 30 - _ok.height
31                             - 10, _ok.width, _ok.height), _ok))
32         {
33             Close();
34         }
35     }
36
37     void Close()
38     {
```

```
39     _isOpen = false;
40 }
41 }
```

Options menüsünde olup olmadığımizi belirleyen `_isOpen` adında bir değişken oluşturduk. OK butonuna basınca değerini false yapıyoruz. Değişkenin değeri false ise 21. satırdaki "return;" komutu çalıştırılıyor ve OnGUI'nin 21. satırdan sonraki satırları çalıştırılmıyor (Options menüsü ekrana çizdirilmiyor).

Oyunu çalıştırıp OK butonuna basın. Options menüsünün kapanması lazım.

Options Menüsünü Ana Menüden Açımak

Şu anda oyunun başında Options menüsü aktif durumda. Ama asıl olması gereken ana menünün ekranda gözükmesi ve buradaki Options butonuna basınca Options menüsünün belirmesi.

Bunu düzeltelim. OptionsGui scriptinde şu ufak değişikliği yapın:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     static bool _isOpen = false;
16
17     void OnGUI()
18     {
19         if (!_isOpen)
20         {
21             return;
22         }
23
24         GUI.skin = _guiSkin;
25
26         GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
27
28         GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
29
30         if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.height - 30 - _ok.height
31                         - 10, _ok.width, _ok.height), _ok))
32         {
33             Close();
34         }
35     }
36
37     void Close()
38     {
39         _isOpen = false;
```

```
40    }
41 }
```

Artık oyunun başında Options menüsü aktif olmayacağı (çünkü `_isOpen`'ın başlangıç değeri `false`).

Ana menüdeyken Options butonuna basınca Options menüsünün gelmesini sağlayalım.
OptionsGui'ye şu eklemeyi yapın:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     static bool _isOpen = false;
16
17     static public void Open()
18     {
19         _isOpen = true;
20     }
21
22     void OnGUI()
23     {
24         if (!_isOpen)
25         {
26             return;
27         }
28
29         GUI.skin = _guiSkin;
30
31         GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
32
33         GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
34
35         if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.height - 30 - _ok.height
36                         - 10, _ok.width, _ok.height), _ok))
37         {
38             Close();
39         }
40     }
41
42     void Close()
43     {
44         _isOpen = false;
45     }
46 }
```

Open fonksiyonu ile Options menüsünü aktif hale getirebiliyoruz. Tek yapmamız gereken Options butonuna tıklayınca bu fonksiyonu çağırırmak. MainMenuGui scriptini güncelleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _mainMenuBg;
11
12     [SerializeField]
13     Texture2D _title;
14
15     [SerializeField]
16     Texture2D _newGameButton;
17
18     [SerializeField]
19     Texture2D _optionsButton;
20
21     [SerializeField]
22     Texture2D _quitButton;
23
24     void OnGUI()
25     {
26         GUI.skin = _guiSkin;
27
28         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
29
30         GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
31                         _title);
32
33         if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height), _newGameButton))
34         {
35             Application.LoadLevel("MainScene");
36         }
37
38         if (GUI.Button(new Rect(20, 150 + 88 + 10, 270, _optionsButton.height), _optionsButton))
39         {
40             OptionsGui.Open();
41         }
42
43         if (GUI.Button(new Rect(20, 150 + (2*(88 + 10)), 270, _quitButton.height), _quitButton))
44         {
45             Application.Quit();
46         }
47     }
48 }

```

Oyunu çalıştırın. Options butonunun istediği gibi çalışması lazım.

Options Menüsünün Arkasında Kalan Butonlara İstemsiz Tıklamak

Oyunu çalıştırın ve Options menüsündeyken, ana menüde yer alan New Game butonunun olduğu yere tıklayın. Her ne kadar New Game butonu ekranda gözükmese de ona tıklayabiliyoruz çünkü bu butonun önünü GUI.Box ile kapatmamız onu tıklamamıza bir engel teşkil etmiyor.

Yapmamız gereken şu: ana menüdeki butonlara tıklayınca sadece ve sadece eğer ki options menüsü kapalıysa o butonun yapması gerekenleri yerine getirmeliyiz. Bunun için de `_isOpen`'ın false olup olmadığına bakmalıyız.

Maalesef `_isOpen` private bir değişken. Ona başka scriptlerden erişmek için bir property oluşturalım:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     static bool _isOpen = false;
16     static public bool IsOpen { get{ return _isOpen; } }
17
18     static public void Open()
19     {
20         _isOpen = true;
21     }
22
23     void OnGUI()
24     {
25         if (!_isOpen)
26         {
27             return;
28         }
29
30         GUI.skin = _guiSkin;
31
32         GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
33
34         GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
35
36         if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.height - 30 - _ok.height
37                             - 10, _ok.width, _ok.height), _ok))
38         {
39             Close();
40         }
41     }
42
43     void Close()
44     {
45         _isOpen = false;
46     }
47 }
```

Ardından MainMenuGui'yi güncelleyelim:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _mainMenuBg;
11
12     [SerializeField]
13     Texture2D _title;
14
15     [SerializeField]
16     Texture2D _newGameButton;
17
18     [SerializeField]
19     Texture2D _optionsButton;
20
21     [SerializeField]
22     Texture2D _quitButton;
23
24     void OnGUI()
25     {
26         GUI.skin = _guiSkin;
27
28         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height),
29                         _mainMenuBg);
30
31         GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.
32                         width, _title.height), _title);
33
34         if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height),
35                         _newGameButton) && !OptionsGui.isOpen)
36         {
37             Application.LoadLevel("MainScene");
38         }
39
40         if (GUI.Button(new Rect(20, 150 + 88 + 10, 270, _optionsButton.height),
41                         _optionsButton) && !OptionsGui.isOpen)
42         {
43             OptionsGui.Open();
44         }
45
46         if (GUI.Button(new Rect(20, 150 + (2*(88 + 10)), 270, _quitButton.
47                         height), _quitButton) && !OptionsGui.isOpen)
48         {
49             Application.Quit();
50         }
51     }
52 }
```

Zorluk (Difficulty) Ayarı Ekleme

Bunu checkbox (işaret kutucuğu) kullanarak yapacağız.

OptionsGui scriptini düzenleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     [SerializeField]
16     Texture2D _titleDifficulty;
17
18     [SerializeField]
19     Texture2D[] _difficulties;
20
21     static int _selectedDifficulty = 0;
22
23     static bool _isOpen = false;
24     static public bool IsOpen { get{ return _isOpen; } }
25
26     static public void Open()
27     {
28         _isOpen = true;
29     }
30
31     void OnGUI()
32     {
33         if (!_isOpen)
34         {
35             return;
36         }
37
38         GUI.skin = _guiSkin;
39
40         GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
41
42         GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
43
44         GUI.DrawTexture(new Rect(50, 150, _titleDifficulty.width,
45                               _titleDifficulty.height), _titleDifficulty);
46
47         _selectedDifficulty = GUI.SelectionGrid(new Rect(100, 250, 280, 400),
48                                              _selectedDifficulty, _difficulties, 1);
49
50         if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.
51                           height - 30 - _ok.height - 10, _ok.width, _ok.height), _ok))
52     {
```

```

53         Close();
54     }
55 }
56
57 void Close()
58 {
59     _isOpen = false;
60 }
61 }
```

44. satırda, üzerinde "Difficulty:" yazan bir texture'yi ekrana çizdiriyoruz.

47. satırda ise Selection Grids adında yeni bir GUI elemanından faydalıyoruz. Bu GUI elemanı bir dizi checkbox'tan oluşur ve bu checkbox'lar arasında aynı anda sadece birisi seçili olabilir. Bu da zorluk seçmek için ideal birşey.

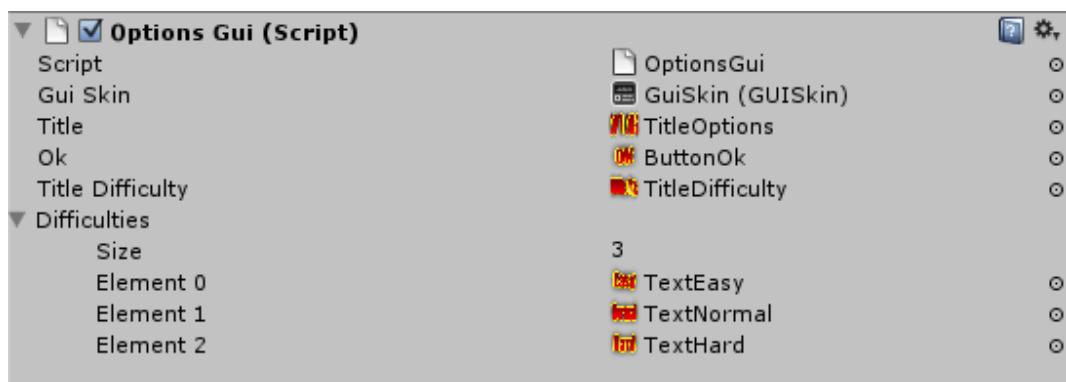
GUI.SelectionGrid fonksiyonu seçili olan checkbox'ın sırasını döndürür. Eğer ilk checkbox seçili ise 0, ikinci seçili ise 1, üçüncü seçili ise 2 döndürür ve bu böyle devam eder. Biz bu döndürülen değeri _selectedDifficulty değişkenine veriyoruz.

GUI.SelectionGrid'in ikinci parametresine de _selectedDifficulty yazıyoruz. Fonksiyon bu parametre sayesinde ekranındaki checkbox'lardan hangisine tık işaretini koyacağını anlar.

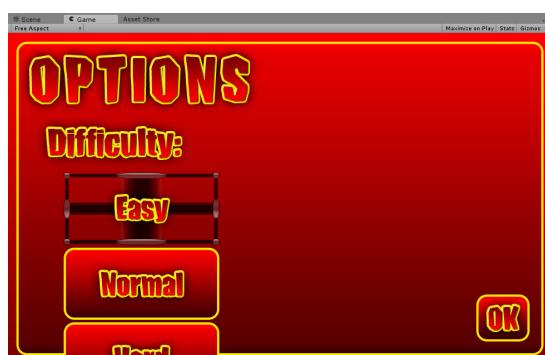
Üçüncü parametre olarak checkbox'ların yanında yer alacak olan Texture'leri barındıran bir array (dizi) giriyoruz. Eğer istersek checkbox'ların yanında texture göstermez, sadece bir string yazdırırız. Bunu yapmak isterseniz 3. parametreyi bir string array'i ile değiştirmelisiniz.

Son parametremiz ise checkbox listesinin kaç sütundan oluşacağını belirler. Değerini 1 verdigimiz için 1 sütundan oluşur ve checkbox'lar alt alta dizilirler. Ama diyelim ki GUI.SelectionGrid'e 4 elemanlı bir array'i parametre olarak verip sütun sayısını 2 yapsaydık o zaman bir satırda 2 checkbox gözüktü (yani checkbox'lar 2x2'lik bir matris şeklinde gözüktü).

Options Gui component'indeki değişkenlere resimdeki gibi değerlerini verin:



Oyunu test edin:



Checkbox'larımız pek bir çirkin duruyorlar.

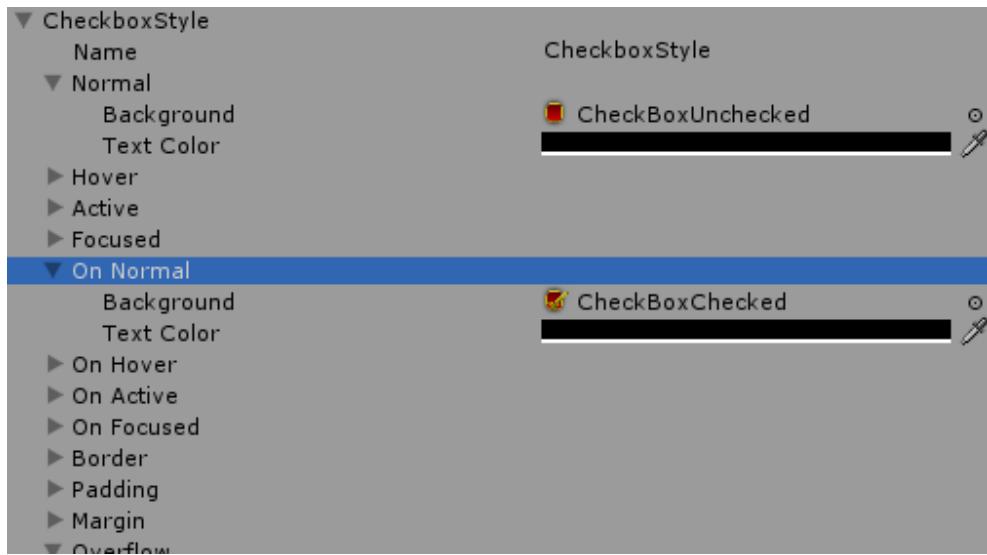
Unity'nin, checkbox'ları ekrana çizdirirken kullanacağı arkaplan texture'lerini biz belirlemeliyiz ve bunun için de yeni bir stile ihtiyacımız var.

Project panelinden GuiSkin'i seçin.

"Custom Styles"ın içindeki "Element 0"ı genişletin. Bu stilin ismini (Name) "CheckboxStyle" yapın (alttaki resimde Checkbox diye duruyor ama siz "CheckboxStyle" yapın).



Şimdi "Normal" ve "On Normal" kısımlarını genişletin ve "Background"larına şöyle değer atayın:



"Normal" stili checkbox seçili değilken kullanılırken "On Normal" stili checkbox seçiliyken kullanılır.

Artık GUI.SelectionGrid'in, oluşturduğumuz bu stili kullanmasını sağlayalım. OptionsGui scriptini açıp şu şekilde güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     [SerializeField]
16     Texture2D _titleDifficulty;
17
18     [SerializeField]
19     Texture2D[] _difficulties;
20
21     static int _selectedDifficulty = 0;
22
23     static bool _isOpen = false;
24     static public bool IsOpen { get{ return _isOpen; } }
25
26     static public void Open()
27     {
28         _isOpen = true;
29     }
30
31     void OnGUI()
32     {
33         if (!isOpen)
```

```

34     {
35         return;
36     }
37
38     GUI.skin = _guiSkin;
39
40     GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
41
42     GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
43
44     GUI.DrawTexture(new Rect(50, 150, _titleDifficulty.width,
45                             _titleDifficulty.height), _titleDifficulty);
46
47     _selectedDifficulty = GUI.SelectionGrid(new Rect(100, 250, 280, 400),
48                                             _selectedDifficulty, _difficulties, 1,
49                                             "CheckboxStyle");
50
51     if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.
52                     height - 30 - _ok.height - 10, _ok.width, _ok.height), _ok))
53     {
54         Close();
55     }
56 }
57
58 void Close()
59 {
60     _isOpen = false;
61 }
62 }
```

GUI.SelectionGrid'in en sonuna, kullanmak istediğimiz stilin ismini parametre olarak giriyoruz. Bu kadar basit!

Oyunu test ederseniz sonucun tam da istediğimiz gibi olmadığını görebilirsiniz:



Checkbox'lar halen tam düzgün gözükmüyor. GUI stilinde bazı ayarlar yapmamız lazım.

Bilgilendirme

GUI Skin'deki ayarları değiştirirken oyunu durdurmak zorunda değilsiniz. Yaptığınız değişikliklerin sonuçlarını anında gözlemlleyebilirsiniz!

GUI Skin asset'inin "CheckboxStyle"ındaki "Overflow" kısmını genişletin ve oradaki değerleri şöyle değiştirin:

- Left: 0
- Right: -72
- Top: 0
- Bottom: 0

"Fixed Width" ve "Fixed Height" değerlerini ise şu şekilde değiştirin:

- Fixed Width: 192
- Fixed Height: 97

Checkbox'ların boyutu düzeyecek. Geriye yazıyı checkbox'un üzerinden taşımak kaldı:



"Content Offset"ı bulun ve değerini şöyle güncelleyin:

- X: 77
- Y: 0

Son olarak "Alignment"ı "MiddleLeft" yapın:



Sonuç tek kelimeyle harika!

Ses Düzeyini (Volume) Ayarlamak

Henüz oyunumuzda ses yok ama ileride olacak. Bu seslerin şiddetini ise şimdiden ayarlayabileceğiz; ne kadar da şanslıyız!

OptionsGui scriptini düzenleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     [SerializeField]
16     Texture2D _titleDifficulty;
17
18     [SerializeField]
19     Texture2D[] _difficulties;
20
21     [SerializeField]
22     Texture2D _titleSound;
23
24     static int _selectedDifficulty = 0;
25
26     static float _soundVolume = 1.0f;
27
28     static bool _isOpen = false;
29     static public bool IsOpen { get{ return _isOpen; } }
30
31     static public void Open()
32     {
33         _isOpen = true;
34     }
35
36     void OnGUI()
37     {
38         if (!_isOpen)
39         {
40             return;
41         }
42
43         GUI.skin = _guiSkin;
44
45         GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
46
47         GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
48 }
```

```

49
50     GUI.DrawTexture(new Rect(50, 150, _titleDifficulty.width,
51                             _titleDifficulty.height), _titleDifficulty);
52
53     _selectedDifficulty = GUI.SelectionGrid(new Rect(100, 250, 280, 400),
54                                             _selectedDifficulty, _difficulties, 1,
55                                             "CheckboxStyle");
56
57
58     GUI.DrawTexture(new Rect(440, 150, _titleSound.width, _titleSound.
59                           height), _titleSound);
60
61     _soundVolume = GUI.HorizontalSlider(new Rect(490, 250, Screen.width-490-
62                                         50, 20), _soundVolume, 0.0f, 1.0f);
63
64     if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.
65                     height - 30 - _ok.height - 10, _ok.width, _ok.height), _ok))
66     {
67         Close();
68     }
69 }
70
71 void Close()
72 {
73     _isOpen = false;
74 }
75 }
```

GUI.HorizontalSlider fonksiyonu, yatay eksende (horizontal) kaydırılabilir bir slider oluşturmaya yarar. Ses düzeyini _soundVolume değişkeninde tutuyoruz. Eğer değeri 0.0 ise hiç ses yokken 1.0 ise sesler %100'lük bir şiddetle çalışıyorlar. 0.5 ise de sesler %50 şiddette çalışıyorlar.

GUI.HorizontalSlider'ın ilk parametresi ekrandaki konumu (Rect). Bu Rect'in genişliği (üçüncü parametresi) Screen.width'ten faydalıyor ve böylece slider ekranın çözünürlüğü ne olursa olsun ekranın sağ kenarına kadar uzanıyor.

GUI.HorizontalSlider'ın 2. parametresi mevcut ses şiddetini değer olarak alıyor ve ona göre slider'daki çubuğu ekranda çizdiriyor.

3. ve 4. parametreler de slider'ın döndürdüğü minimum ve maksimum değerleri belirliyor. Eğer çubuk slider'ın en solundaysa 0.0, en sağındaysa 1.0 döndürülmesini sağlıyoruz.

Inspector'dan "Title Sound'a değer olarak "TitleSound" texture'sini verin ve oyunu çalıştırın:



Siyah slider stili öteki kırmızı stillerin yanında garip duruyor. Bunun için slider'ın stilini de değiştirelim. GUI Skin'deki "Horizontal Slider" stili slider'ın arkaplanını değiştirmeye yararken "Horizontal Slider Thumb" ise slider'ın üzerindeki çubuğu stilini değiştirmeye yarar.

"Horizontal Slider"ın "Normal"indeki "Background"ı "SliderBg" olarak değiştirin.

"Horizontal Slider Thumb"ın "Normal", "Hover" ve "Active"indeki "Background" değerlerini ise "SliderThumb" olarak değiştirin.

Sonrasında "Horizontal Slider"da şu değişiklikleri yapın:

- Overflow
 - Left: 0
 - Right: 0
 - Top: 25
 - Bottom: -25
- Fixed Height: 55

Overflow'u bu şekilde değiştirince slider biraz yukarı kayıyor. Fixed Height'ı 55 yapınca ise slider'ın hep 55 pixel yükseliğinde olmasını sağlıyoruz.

"Horizontal Slider Thumb" stilinde ise şu değişiklikleri yapın:

- Overflow
 - Left: 0
 - Right: 0
 - Top: 28
 - Bottom: -28
- Fixed Width: 62
- Fixed Height: 62

Artık çok sık bir slider'ımız var:



Options Menüsünde Yaptığınız Değişiklikleri Kaydetmek

Şu anda oyunu kapatıp tekrar açınca ayarlar menüsündeki ayarlarınız resetleniyor. Bu ayarları (zorluk ve ses düzeyi) cihaza kaydetmeli ve oyundan çıkışın yerinden girince ayarların en son halinde kalmasını

sağlamalıyız. Bunu, Unity'nin PlayerPrefs class'ı ile başarabiliriz.

Değerleri Kaydetmek

OptionsGui scriptine şu kodu ekleyin:

```
36 void OnGUI()
37 {
38     if (!_isOpen)
39     {
40         return;
41     }
42
43     GUI.skin = _guiSkin;
44
45     GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
46
47     GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
48
49
50     GUI.DrawTexture(new Rect(50, 150, _titleDifficulty.width,
51                             _titleDifficulty.height), _titleDifficulty);
52
53     _selectedDifficulty = GUI.SelectionGrid(new Rect(100, 250, 280, 400),
54                                             _selectedDifficulty, _difficulties, 1,
55                                             "CheckboxStyle");
56     PlayerPrefs.SetInt("Difficulty", _selectedDifficulty);
57
58
59     GUI.DrawTexture(new Rect(440, 150, _titleSound.width, _titleSound.
60                           height), _titleSound);
61
62     _soundVolume = GUI.HorizontalSlider(new Rect(490, 250, Screen.width-490-
63                                         50, 20), _soundVolume, 0.0f, 1.0f);
64     PlayerPrefs.SetFloat("SoundVolume", _soundVolume);
65
66
67     if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.
68                     height - 30 - _ok.height - 10, _ok.width, _ok.height), _ok))
69     {
70         Close();
71     }
72 }
73
74 void Close()
75 {
76     _isOpen = false;
77 }
78 }
```

56. satırda PlayerPrefs.SetInt fonksiyonunu kullanıyoruz. Bu fonksiyon cihaza bir tamsayı (int) değerini kaydetmeye yarar. İlk parametresi kaydedilecek değere sonradan erişmek için kullanacağımız bir isimdir: "Difficulty".

Bu değer cihazın harddiskinde kaydedilmekte ve bu işlem Unity tarafından otomatik olarak yapılmakta. Yani sizin harddiskteki dosyalara erişmekle, onları düzenlemekle vb. hiç uğraşmanız gerekmıyor.

64. satırda da PlayerPrefs.SetFloat ile float türündeki ses şiddetini "SoundVolume" adı altında cihaza kaydediyoruz.

Oyun Başlayınca Değerleri Cihazdan Okumak

Oyun başlayınca ayarlar menüsündeki değerleri cihazdan okumalıyız ki ayarlar son bıraktığımız halde dursun.

OptionsGui scriptini güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     [SerializeField]
16     Texture2D _titleDifficulty;
17
18     [SerializeField]
19     Texture2D[] _difficulties;
20
21     [SerializeField]
22     Texture2D _titleSound;
23
24     static int _selectedDifficulty = 0;
25
26     static float _soundVolume = 1.0f;
27
28     static bool _isOpen = false;
29     static public bool IsOpen { get{ return _isOpen; } }
30
31     static public void Open()
32     {
33         _isOpen = true;
34     }
35
36     void Start()
37     {
38         _selectedDifficulty = PlayerPrefs.GetInt("Difficulty", 1);
39         _soundVolume = PlayerPrefs.GetFloat("SoundVolume", 1.0f);
40     }
41
42     void OnGUI()
43     {
```

```
44     if (!isOpen)
45     {
46         return;
47     }
48
49     GUI.skin = _guiSkin;
```

.

.

Nasıl SetInt ve SetFloat cihaza bir değeri kaydetmeye yarıyorsa GetInt ve GetFloat da cihazda kayıtlı bir değeri okumaya yarar. Fonksiyona ilk parametre olarak cihazda kaydedilen değerin ismi girilir. İkinci parametre olarak da eğer ki cihazda bu isimde bir değer kaydedilmemişse fonksiyonun döndüreceği değer girilir.

Özet Geçecek Olursak...

Artık bir ana menümüz var. Bu bölümde nasıl GUI Skin'den ve stillerden faydalanaileceğinizi gördünüz. Ayrıca cihaza bir değeri kaydedip sonradan bu değeri geri okumayı da gördünüz.

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

Unity 3D İle Oyun Programlama

Bölüm 15: GörSEL ve Ses Efektleri 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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Bu bölümde oyunumuza partikül efektleri ve ses efektleri ekleyeceğiz.



Zombiyi Vurunca Kan Efekti Çıkması

Winrar arşivinin olduğu yerdeki "Packages" klasörünün içinde "BloodHit.unitypackage" adında bir Unitypackage var. Bu package'in içinde kullanıma hazır bir kan efekti bulunmaktadır. Biz oyunumuzda bunu kullanacağız.

Unity'de `Assets > Import Package > Custom Package...` yolunu izleyin ve BloodHit paketini projenize import edin. TheGame/Prefabs/ klasörüne "BloodHit" adında bir prefab gelecek.

RifleWeapon scriptini açın. Düşmana vurup vurmadığımızı bu scriptte kontrol ediyoruz. Kodu güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class RifleWeapon : MonoBehaviour
05 {
06     [SerializeField]
07     int _damageDealt = 10;
08
09     [SerializeField]
10     GameObject _hitEffectPrefab;
11
12     void Start()
13     {
14         Screen.lockCursor = true;
15     }
16
17     void Update()
18     {
```

```

19     if (Input.GetKeyDown(KeyCode.Escape))
20     {
21         Screen.lockCursor = false;
22     }
23
24     if (Input.GetButtonDown("Fire1"))
25     {
26         Screen.lockCursor = true;
27         Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.
28                                         5f, 0));
29         RaycastHit hitInfo;
30
31         if (Physics.Raycast(mouseRay, out hitInfo))
32         {
33             Health enemyHealth = hitInfo.transform.GetComponent<Health>();
34             if (enemyHealth != null)
35             {
36                 enemyHealth.Damage(_damageDealt);
37
38                 Vector3 hitEffectPosition = hitInfo.point;
39                 Quaternion hitEffectRotation = Quaternion.identity;
40                 Instantiate(_hitEffectPrefab, hitEffectPosition,
41                             hitEffectRotation);
42             }
43         }
44     }
45 }
46 }
```

Yaptığımız şey basit: Düşmana her vurduğumuzda, ona vurdugumuz noktada (hitInfo.point) BloodHit prefab'ının bir klonunu oluşturuyoruz (Instantiate).

Inspector'dan Player'ın RifleWeapon scriptindeki "Hit Effect Prefab'a "BloodHit" prefabını verin.

Oyunu çalıştırın. Zombiye vurunca zombiden kan fışkıracak.

Kan Efektinin Hep Aynı Yöne Doğru (Rotation) Çıkmasını Değiştirmek

Zombiye vurunca kan, her zaman vurdugumuz noktadan dışarı doğru fışkırmıyor. Bazen tam tersi yönde bazense garip bir yönde fışkıriyor. Kodu biraz daha güncelleyelim:

```

33         if (enemyHealth != null)
34         {
35             enemyHealth.Damage(_damageDealt);
36
37             Vector3 hitEffectPosition = hitInfo.point;
38             Quaternion hitEffectRotation = Quaternion.FromToRotation(Vector3.forward,
39                             hitInfo.normal);
40             Instantiate(_hitEffectPrefab, hitEffectPosition, hitEffectRotation);
41         }
42     }
43 }
44 }
```

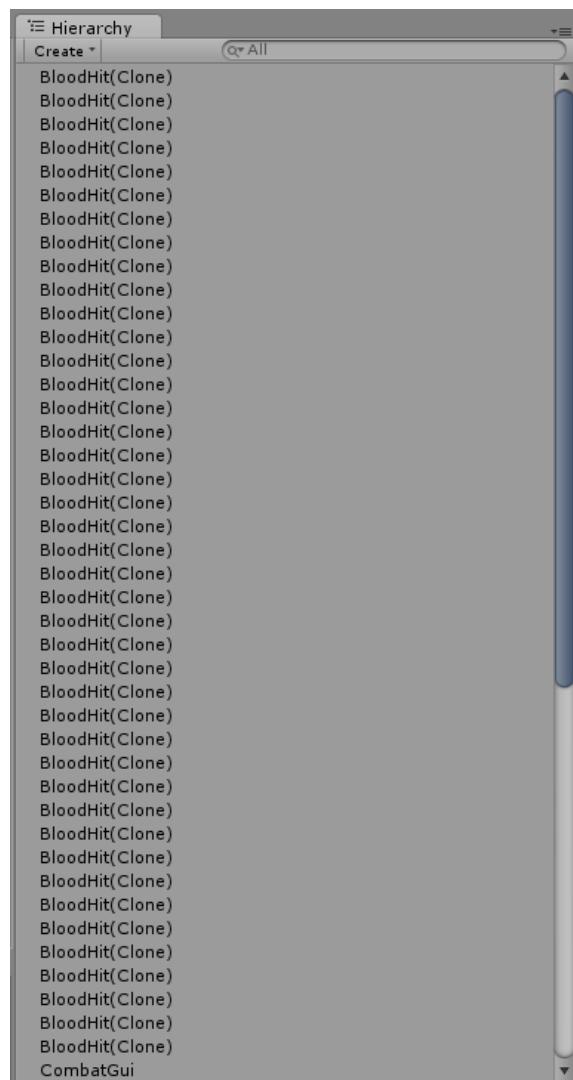
Artık kan efekti doğru yöne doğru fışkıracak.

"hitInfo.normal" komutu, raycast zombinin hangi noktasına isabet etmişse o noktada zombinin yüzeyine dik doğrultuda bir vektör (normal) döndürür.

Bildiğiniz üzere Unity'de bir objenin eğimi Quaternion türünde depolanıyor. Quaternion class'ının ise FromToRotation adında hazır bir fonksiyonu var. Bu fonksiyon bize öyle bir eğim (rotation)(quaternion) döndürür ki bu eğime sahip olan bir objenin birinci parametrede girdiğimiz eksen, ikinci parametrede girdiğimiz vektör yönünde olur. Bizim örneğimizde birinci parametre Vector3.forward ve ikinci parametre ise "hitInfo.normal". Bunun anlamı Instantiate ettiğimiz kan efektinin rotation'ı öyle olacak ki kan efektinin Z eksen (forward), "hitInfo.normal" yönünde, yani kurşunun zombiye temas ettiği noktadan dışarı doğru olacak ve böylece kan dışarı yönde fışkıracak.

Kan Efekti Klonlarını Temizlemek

Oyunu çalıştırıp da zombileri vurmaya başlayınca Hierarchy panelinde kan efekti klonlarının "BloodHit(Clone)" birliğini göreceksiniz:



Çünkü BloodHit prefab'ı iki child objeden oluşuyor. Bu iki child obje de birer partikül sistemi. Bu partikül sistemleri işleri bitince otomatik olarak yok oluyorlar ama parent obje olan BloodHit(Clone) sahnede kalmaya devam ediyor.

Yapmamız gereken parent objenin de silinmesini sağlamak. Peki nasıl? Child obje olan partikül sistemlerinin çalışmayı bitip bitmediğini kodla kontrol edip çalışmaları bittiğinde parent objeyi de elle silerek mi? Hayır, bunu yapmak zor olur. Daha basit bir çözüm yolu var: sahnede kan efekti klonu oluşturduğumuz anda bu klonun iki child objesini de child obje olmaktan çıkaralım (artık kan efekti klonu onların parent'ı olmasın) ve ardından tek başına kalan parent objeyi elle silelim.

“EffectsCleanup” adında yeni bir C# scripti oluşturun:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EffectsCleanup : MonoBehaviour
05 {
06     void Start()
07     {
08         foreach(Transform child in transform)
09         {
10             child.parent = null;
11         }
12         Destroy(gameObject);
13     }
14 }
```

Bir for döngüsü yardımıyla scriptin yazıldığı objenin tüm child objeleri üzerinden tek tek geçiyoruz ve bu child objenin artık child obje olmamasını sağlıyoruz (child.parent = null;). Ardından scriptin yazıldığı objeyi (parent objeyi) Destroy fonksiyonunu kullanarak siliyoruz.

Project panelinden BloodHit prefab'ını seçin ve EffectsCleanUp scriptini bu prefaba verin.

Oyunu test ettiğinizde artık Hierarchy'de klonların birikmediğini göreceksiniz.

Player Hasar Alınca Kan Efekti Oluşması

EnemyAttack scriptinde biraz düzenleme yaparak bir zombi bize hasar verirse player objesinden kan fışkırmasını sağlayacağız

Zombiler saldırırken raycast kullanmadığı için bu sefer hasarın tam olarak hangi noktada olduğunu bilmiyoruz. Bu yüzden kanın fışkıracağı noktayı farklı bir yöntemle hesaplayacağız.

EnemyAttack scriptini düzenleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAttack : MonoBehaviour
05 {
06     float _nextTimeAttackIsAllowed = -1.0f;
07
08     [SerializeField]
09     float _attackDelay = 1.0f;
10
11     [SerializeField]
```

```

12     int _damageDealt = 5;
13
14     [SerializeField]
15     GameObject _hitEffectPrefab;
16
17     void OnTriggerStay(Collider other)
18     {
19         if (other.tag == "Player" && Time.time >= _nextTimeAttackIsAllowed)
20         {
21             Health playerHealth = other.GetComponent<Health>();
22             playerHealth.Damage(_damageDealt);
23
24             Vector3 hitDirection = (transform.root.position - other.transform.position).normalized;
25             Vector3 hitEffectPosition = other.transform.position + (hitDirection * 0.01f) + (Vector3.up * 1.5f);
26             Quaternion hitEffectRotation = Quaternion.FromToRotation(Vector3.forward, hitDirection);
27             Instantiate(_hitEffectPrefab, hitEffectPosition, hitEffectRotation);
28
29             _nextTimeAttackIsAllowed = Time.time + _attackDelay;
30         }
31     }
32 }
```

Öncelikle 24. satırda, yönü player'dan zombiye doğru olan bir vektör oluşturup bunu hitDirection değişkeninde tutuyoruz.

Hatırlarsanız player objesinin pivot noktası ayak hizasında. Eğer kan efektini oradan çıkarırsak ayağımız kanıyor gibi garip bir görüntü oluşur. 25. satırda kan efektinin çıkacağı pozisyonu hesaplarken player'ın pozisyonunun (other.transform.position) 1.5 birim yukarıından (Vector3.up * 1.5f) zombinin olduğu yöne doğru 0.01 birim dışında (hitDirection * 0.01f) bir nokta alıyoruz.

26. satırda ise kan efektinin eğimini hesaplarken Quaternion.FromToRotation yardımıyla kan efektinin player'dan zombinin olduğu yöne doğru fışkırmamasını sağlıyoruz.

Inspector'dan, Enemy prefab'ının EnemyAttack child objesindeki "Enemy Attack" component'inde yer alan "Hit Effect Prefab"ı "BloodHit"ı verip oyunu test edebilirsiniz. Düşman size hasar verince player objesinden düşmana doğru kan fışkıracak.

Ses Efektleri

Player veya bir zombi hasar aldıça bir ses efekti oynatacağız.

Health scriptine şu kodu ekleyin:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
```

```
13     return _currentHealth + " / " + _maximumHealth;
14 }
15
16 public bool IsDead { get{ return _currentHealth <= 0; } }
17
18 Renderer _renderer;
19
20 PlayerStats _playerStats;
21
22 [SerializeField]
23 AudioClip _hitSound;
24
25 void Start()
26 {
27     _renderer = GetComponentInChildren<Renderer>();
28     _currentHealth = _maximumHealth;
29
30     GameObject player = GameObject.FindGameObjectWithTag("Player");
31     _playerStats = player.GetComponent<PlayerStats>();
32 }
33
34 public void Damage(int damageValue)
35 {
36     _currentHealth -= damageValue;
37
38     if (_currentHealth < 0)
39     {
40         _currentHealth = 0;
41     }
42     else
43     {
44         if (_hitSound != null)
45         {
46             audio.clip = _hitSound;
47             audio.Play();
48         }
49     }
50
51     if (_currentHealth == 0)
52     {
```

Audio Source componenti ses dosyaları çalmaya yarar. Audio Source component'inin Play() fonksiyonunu kullanarak _hitSound değişkeninde depoladığımız ses dosyasını oynatıyoruz. Tabi öncesinde Audio Source component'inin çalışacağı ses klibini (audio.clip) belirliyoruz.

Scriptin çalışması için Health scriptinin atandığı obje(ler)de (player ve enemy) Audio Source komponenti olması lazım. Eğer yoksa Component > Audio > Audio Source ile komponenti bu objelere ekleyin.

Project panelini kullanarak "TheGame" klasörünün içinde "Sounds" adında bir klasör oluşturun. Bu klasörde oyunda kullanacağımız ses dosyalarını depolayacağız.

Şimdi ise Winrar arşivini çıkardığınız yerdeki "Sounds" klasöründe yer alan "player_killed_1.wav" ve "ZombieDeath.wav" ses dosyalarını projenizdeki "Sounds" klasörüne import edin.

Ardından Zombie prefab'ındaki Health scriptinde yer alan "Hit Sound'a değer olarak "ZombieDeath.wav" asset'ini, Player'daki "Hit Sound'a da "player_killed_1.wav" asset'ini değer olarak verin ve oyunu çalıştırın. Hasar aldığınızda "player_killed_1" sesi, zombiye hasar verdığınızda "ZombieDeath" sesi çalacak.

Player Hasar Alınca Rastgele Bir Hasar Alma Sesi Çalmak

Winrar arşivinin oradaki Sounds klasöründe birden çok "player_killed_x.wav" ses dosyası olduğunu farketmişsinizdir. Bence player hasar alınca hep aynı ses dosyasını çalmak yerine Sounds klasöründeki ses dosyaları içinden rastgele bir tanesini çalalım.

Bunun için diğer "player_killed_x.wav" ses dosyalarını da projenizdeki Sounds klasörüne import edin. Sonrasında Health scriptini şöyle değiştirin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     public bool IsDead { get{ return _currentHealth <= 0; } }
17
18     Renderer _renderer;
19
20     PlayerStats _playerStats;
21
22     [SerializeField]
23     AudioClip[] _hitSounds;
24
25     void Start()
26     {
27         _renderer = GetComponentInChildren<Renderer>();
28         _currentHealth = _maximumHealth;
29
30         GameObject player = GameObject.FindGameObjectWithTag("Player");
31         _playerStats = player.GetComponent<PlayerStats>();
32     }
33
34     public void Damage(int damageValue)
35     {
36         _currentHealth -= damageValue;
37
38         if (_currentHealth < 0)
39         {
```

```

40         _currentHealth = 0;
41     }
42     else
43     {
44         if (_hitSounds != null && _hitSounds.Length > 0)
45         {
46             AudioClip soundToUse = _hitSounds[Random.Range(0, _hitSounds.Length)];
47             audio.clip = soundToUse;
48             audio.Play();
49         }
50     }
51
52     if (_currentHealth == 0)
53     {
54         :
55     }

```

Yaptığımız şey _hitSounds adında bir AudioClip array'i bulundurmak ve player ya da enemy hasar alınca bu array'in içindeki ses kliplerinden rastgele bir tanesini oynatmak. Array'den rastgele bir eleman çekmek için Random.Range fonksiyonunu kullanıyoruz.

Player objesini seçin ve "Hit Sounds"un Size'ını artırıp import ettiğiniz "player_killed_x.wav" ses dosyalarını array'e değer olarak ekleyin. Ardından Enemy prefab'ını seçin ve onun "Hit Sounds"unun Size'ını 1 yapıp "Element 0"ına değer olarak "ZombieDeath.wav" asset'ini verin.

Player Ölünce Bir Müziğin Çalması

Hasar alma sesiyle aynı mantığı kullanıyoruz. Bu sefer ses dosyasını sağlığımız dibe vurunca çaldırıyoruz:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     public bool IsDead { get{ return _currentHealth <= 0; } }
17
18     Renderer _renderer;
19
20     PlayerStats _playerStats;
21
22     [SerializeField]

```

```

23     AudioClip[] _hitSounds;
24
25     [SerializeField]
26     AudioClip _deathSound;
27
28     void Start()
29     {
30         _renderer = GetComponentInChildren<Renderer>();
31         _currentHealth = _maximumHealth;
32
33         GameObject player = GameObject.FindGameObjectWithTag("Player");
34         _playerStats = player.GetComponent<PlayerStats>();
35     }
36
37     public void Damage(int damageValue)
38     {
39         _currentHealth -= damageValue;
40
41         if (_currentHealth < 0)
42         {
43             _currentHealth = 0;
44         }
45         else
46         {
47             if (_hitSounds != null && _hitSounds.Length > 0)
48             {
49                 AudioClip soundToUse = _hitSounds[Random.Range(0, _hitSounds.Length)];
50                 audio.clip = soundToUse;
51                 audio.Play();
52             }
53         }
54
55         if (_currentHealth == 0)
56         {
57             if (_deathSound != null)
58             {
59                 audio.clip = _deathSound;
60                 audio.Play();
61             }
62
63             Animation a = GetComponentInChildren<Animation>();
64             a.Stop();
65
66             .
67             .
68             .

```

Winrar arşivindeki Sounds klasöründe yer alan "you are dead dead dead 2.wav" ses dosyasını projenizdeki Sounds klasörüne import edin ve ardından Player'daki "Death Sound"a değer olarak verin. Bu müziğin sadece Player ölünce çalmasını istiyoruz. O yüzden Enemy'de bir değişiklik yapmayacağız.

İpucu

Winrar arşivinde "23289__progic35__resocopter-fastE.wav" adında bir ses dosyası var. Bu, helikopterin pervanesi için ideal bir ses. Kendiniz uğraşarak helikopterin bu ses dosyasını sürekli olarak (loop halinde) çalmasını sağlayın.

ÇEVİRMEN EKLEMESİ: Bunu script yazmadan da yapabilirsiniz. Dikkat etmeniz gereken şey helikoptere vereceğiniz Audio Source component'inin "Audio Clip", "Play On Awake" ve "Loop" değerleri.

Özet Geçecek Olursak...

Oyuna bazı görsel efektler ve ses efektleri ekledik. Bunlar oyun mekaniklerini doğrudan değiştirmiyor; sadece oyunun daha hoş durmasını sağlıyor. Oyunlarınıza efektler eklemeniz, oyuncunun dikkatini çekmenize çok yardımcı olabilir.

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

Unity 3D İle Oyun Programlama

Bölüm 16: Düşmanın Sağlık Paketi Düşürmesi



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



All code snippets are licensed under CC0 (public domain)



This document except code snippets is licensed with Creative Commons Attribution-NonCommercial 3.0 Unported

Bazı zombilerin ölünce yere sağlık paketi (medkit) düşürmesi sizce nasıl olurdu? Bence güzel bir detay olurdu. Bu medkit'lerin üstünden geçersek sağlığımızı artacak.

Medkit'i Oluşturmak

Medkit'imiz basit bir küp objesinden ibaret olacak. Bu objeyi bir prefab yapacağımız ve böylece rahatlıkla Instantiate edebileceğiz (run-time olarak klonlayabileceğiz).

Eğer Player küp ile temas ederse sağlığı artacak.

“Medkit” adında C# scripti oluşturun:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Medkit : MonoBehaviour
05 {
06     void OnTriggerEnter(Collider c)
07     {
08         if (c.transform.root.tag == "Player")
09         {
10             Debug.Log("collided with the player!");
11         }
12     }
13 }
```

Şu anda sadece player'ın Medkit'e temas edip etmediğini kontrol ediyoruz. Bunu yaparken temas eden objenin (c) root objesinin tag'ını kontrol ediyoruz çünkü Player bir ragdoll'a sahip; ragdoll'lar ise collider'lara. Ragdoll'ların tag'ı "Player" değil, yani medkit örneğin player'ın bacağı ile temas ederse ve biz "c.transform.tag == "Player"" yazmış olsaydık if'in içindeki kod çalışmazdı. Ama şu anda bacak ile bile temas etsek root objenin (Player'ın kendisi) tag'ına ("Player") baktığımız için if'in içindeki kod çalışacak.

MainScene.unity sahnesinin açık olduğundan emin olun. GameObject > Create Other > Cube yolunu izleyip bir küp oluşturun ve bunu player'ın yakınına yerleştirin. Küp'ün "Box Collider"ındaki "Is Trigger"ı işaretleyin. Sonrasında Medkit scriptini küp objesine verin. Ayrıca bir de Rigidbody de verin (bence "Is Kinematic"ı işaretleyin, böylece medkit yerçekiminden etkilenmez).

Oyunu çalıştırın ve medkit'in içinden geçin. Konsola mesaj yazdırılacak.

Şimdi player'ın sağlığını tazeleyen bir fonksiyon lazım bize. Nasıl hasar almak için bir fonksiyon varsa can tazelemek için de bir fonksiyonumuz olsun.

Health scriptini şu şekilde güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
```

```
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     public bool IsDead { get{ return _currentHealth <= 0; } }
17
18     Renderer _renderer;
19
20     PlayerStats _playerStats;
21
22     [SerializeField]
23     AudioClip[] _hitSounds;
24
25     [SerializeField]
26     AudioClip _deathSound;
27
28     void Start()
29     {
30         _renderer = GetComponentInChildren<Renderer>();
31         _currentHealth = _maximumHealth;
32
33         GameObject player = GameObject.FindGameObjectWithTag("Player");
34         _playerStats = player.GetComponent<PlayerStats>();
35     }
36
37     public void Heal(int healAmount)
38     {
39         _currentHealth += healAmount;
40
41         if (_currentHealth > _maximumHealth)
42         {
43             _currentHealth = _maximumHealth;
44         }
45     }
46
47     public void Damage(int damageValue)
48     {
49         _currentHealth -= damageValue;
50
51         if (_currentHealth < 0)
52         {
53             _currentHealth = 0;
54         }
55         else
56         {
57             if (_hitSounds != null && _hitSounds.Length > 0)
58             {
59                 AudioClip soundToUse = _hitSounds[Random.Range(0, _hitSounds.Length)];
60                 audio.clip = soundToUse;
61                 audio.Play();
62             }
63         }
64
65         if (_currentHealth == 0)
66         {
67             if (_deathSound != null)
68             {
```

```

69             audio.clip = _deathSound;
70             audio.Play();
71         }
72
73         Animation a = GetComponentInChildren<Animation>();
74         a.Stop();
75
76         if (tag == "Player")
77     {
78             Destroy(GetComponent<PlayerMovement>());
79             Destroy(GetComponent<PlayerAnimation>());
80             Destroy(GetComponent<RifleWeapon>());
81         }
82         else // its an enemy
83     {
84             _playerStats.ZombiesKilled++;
85             EnemySpawnManager.OnEnemyDeath();
86             Destroy(GetComponent<EnemyMovement>());
87             Destroy(GetComponentInChildren<EnemyAttack>());
88         }
89
90         Destroy(GetComponent<CharacterController>());
91
92         Ragdoll r = GetComponent<Ragdoll>();
93         if (r != null)
94     {
95             r.OnDeath();
96         }
97     }
98 }
99
100 void Update()
101 {
102     if (IsDead && !_renderer.isVisible)
103     {
104         Destroy(gameObject);
105     }
106 }
107 }
```

Fonksiyonun yaptığı iş basit: sağlığımızı biraz artırıyor ve eğer sağlığımız maksimum değerini aştıysa onu maksimum değere çekiyor.

Şimdi Medkit scriptine geri dönüp şu değişiklikleri yapın:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Medkit : MonoBehaviour
05 {
06     [SerializeField]
07     int _healAmount = 50;
08
09     void OnTriggerEnter(Collider c)
10     {
11         if (c.transform.root.tag == "Player")
12         {
13             Health playerHealth = c.transform.root.GetComponent<Health>();
```

```
14         playerHealth.Heal(_healAmount);
15         Destroy(gameObject);
16     }
17 }
18 }
```

Oyuncunun sağlığını tazeliyor ve ardından medkit'i yok ediyoruz. Çok güzel!

Ölen Düşmanın Medkit Düşürmesi

Önce medkit'i bir prefab yapmalıyız. Küp objesini "Medkit" şeklinde yeniden adlandırın. Ardından Hierarchy'den tutup Project panelindeki "Prefabs" klasörüne sürüklestin. Böylece "Medkit" bir prefaba dönüşecektir.

"EnemyDrops" adında yeni bir C# scripti oluşturun. Bu script, zombilerin ölünce medkit düşürmesini sağlayacak:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyDrops : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _dropItemPrefab;
08
09     public void OnDeath()
10     {
11         Instantiate(_dropItemPrefab, transform.position, transform.rotation);
12     }
13 }
```

OnDeath adında public bir fonksiyonumuz var. Bu fonksiyon çağrılmışça _dropItemPrefab'ı, scriptin yazıldığı objenin olduğu yerde oluşturmaya (Instantiate) yarıyor.

Health scriptini açın. Burada düşman ölünce onun EnemyDrops class'ındaki OnDeath fonksiyonunu çağıracağız:

```
47     public void Damage(int damageValue)
48     {
49         _currentHealth -= damageValue;
50
51         if (_currentHealth < 0)
52         {
53             _currentHealth = 0;
54         }
55         else
56         {
57             if (_hitSounds != null && _hitSounds.Length > 0)
58             {
59                 AudioClip soundToUse = _hitSounds[Random.Range(0, _hitSounds.Length)];
60                 audio.clip = soundToUse;
61                 audio.Play();
62             }
63         }
64     }
```

```

62         }
63     }
64
65     if (_currentHealth == 0)
66     {
67         if (_deathSound != null)
68         {
69             audio.clip = _deathSound;
70             audio.Play();
71         }
72
73     Animation a = GetComponentInChildren<Animation>();
74     a.Stop();
75
76     if (tag == "Player")
77     {
78         Destroy(GetComponent<PlayerMovement>());
79         Destroy(GetComponent<PlayerAnimation>());
80         Destroy(GetComponent<RifleWeapon>());
81     }
82     else // its an enemy
83     {
84         _playerStats.ZombiesKilled++;
85         EnemySpawnManager.OnEnemyDeath();
86         Destroy(GetComponent<EnemyMovement>());
87         Destroy(GetComponentInChildren<EnemyAttack>());
88
89         EnemyDrops d = GetComponent<EnemyDrops>();
90         d.OnDeath();
91     }
92
93     Destroy(GetComponent<CharacterController>());
94
95     Ragdoll r = GetComponent<Ragdoll>();
96     if (r != null)
97     {
98         r.OnDeath();
99     }
100    }
101 }
102
103 void Update()
104 {
105     if (IsDead && !_renderer.isVisible)
106     {
107         Destroy(gameObject);
108     }
109 }
110 }
```

EnemyDrops scriptini Enemy prefabına atayın. Medkit prefabını da Inspector'u scriptteki “Drop Item Prefab” değişkenine değer olarak atayın.

Düşman ölünce yere medkit düşürecek. Geriye medkit'in her zaman değil de sadece bazı düşmanlar ölünce düşmesini ayarlamak kaldı.

EnemyDrops scriptini güncelleyin:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyDrops : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _dropItemPrefab;
08
09     [SerializeField]
10     float _chanceToDrop = 50.0f;
11
12     public void OnDeath()
13     {
14         if (Random.Range(0.0f, 100.0f) <= _chanceToDrop)
15         {
16             Instantiate(_dropItemPrefab, transform.position, transform.rotation);
17         }
18     }
19 }
```

Artık _chanceToDrop kadarlık bir % olasılıkla yere medkit düşüyor. Bunu başarmak için Instantiate fonksiyonunu Random.Range'i kullanan bir "if" koşulunun içine aldık. Random.Range 0 ile 100 arasında bir float döndürüyor ve eğer bu sayı _chanceToDrop'tan küçükse medkit oluşturuluyor.

Medkit Alınca Ses Çalması

Health scriptini güncelleyelim:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     public bool IsDead { get{ return _currentHealth <= 0; } }
17
18     Renderer _renderer;
19
20     PlayerStats _playerStats;
21
22     [SerializeField]
23     AudioClip[] _hitSounds;
24
25     [SerializeField]
```

```

26     AudioClip _deathSound;
27
28     [SerializeField]
29     AudioClip _healSound;
30
31     void Start()
32     {
33         _renderer = GetComponentInChildren<Renderer>();
34         _currentHealth = _maximumHealth;
35
36         GameObject player = GameObject.FindGameObjectWithTag("Player");
37         _playerStats = player.GetComponent<PlayerStats>();
38     }
39
40     public void Heal(int healAmount)
41     {
42         _currentHealth += healAmount;
43
44         if (_currentHealth > _maximumHealth)
45         {
46             _currentHealth = _maximumHealth;
47         }
48
49         if (_healSound != null)
50         {
51             audio.clip = _healSound;
52             audio.Play();
53         }
54     }
55
56     public void Damage(int damageValue)
57     {
58         _currentHealth -= damageValue;
59
60         if (_currentHealth < 0)
61         {
62             _currentHealth = 0;
63         }
64         else
65         {
66             if (_hitSounds != null && _hitSounds.Length > 0)
67             {
68                 AudioClip soundToUse = _hitSounds[Random.Range(0, _hitSounds.Length)];
69                 audio.clip = soundToUse;
70                 audio.Play();
71             }
72         }
73
74         if (_currentHealth == 0)
75         {
76             if (_deathSound != null)
77             {
78                 audio.clip = _deathSound;
79                 audio.Play();
80             }
81
82         Animation a = GetComponentInChildren<Animation>();

```

```

83         a.Stop();
84
85         if (tag == "Player")
86     {
87             Destroy.GetComponent<PlayerMovement>();
88             Destroy.GetComponent<PlayerAnimation>();
89             Destroy.GetComponent<RifleWeapon>();
90         }
91     else // its an enemy
92     {
93         _playerStats.ZombiesKilled++;
94         EnemySpawnManager.OnEnemyDeath();
95         Destroy.GetComponent<EnemyMovement>();
96         Destroy.GetComponentInChildren<EnemyAttack>();
97
98         EnemyDrops d = GetComponent<EnemyDrops>();
99         d.OnDeath();
100    }
101
102    Destroy.GetComponent<CharacterController>();
103
104    Ragdoll r = GetComponent<Ragdoll>();
105    if (r != null)
106    {
107        r.OnDeath();
108    }
109 }
110 }
111
112 void Update()
113 {
114     if (IsDead && !_renderer.isVisible)
115     {
116         Destroy(gameObject);
117     }
118 }
119 }
```

Burada yazdığımız kod, ölüm sesi çaldırmırken kullandığımız koda çok benziyor. Önce `_healSound`'un bir değerinin olup olmadığına (null olup olmadığına) bakıyoruz. Eğer bu değişkene bir ses dosyası atanmışsa onu Audio Source component'indeki "Audio Clip"e değer olarak veriyoruz ve ardından bu ses dosyasını çaldırıyoruz.

Winrar arşivinin oradaki "Sounds" klasöründe "health1.wav" adında bir dosya var. Player'ın "Health" component'indeki "Heal Sound"a değer olarak bunu verebilirsiniz. Zombilerin medpack alma durumu olmadığı için onlarda "Heal Sound'u boş bırakabilirsiniz.

Medkit'e Kaplama Vermek

Winrar arşivindeki "Images" klasöründe yer alan "MedikitTexture.png"yi projenize import edin. Sonrasında "Medkit" adında yeni bir materyal oluşturup "MedikitTexture"yi materyale texture olarak verin.

Materyalin shader'ını "Mobile-Diffuse" olarak değiştirin.

Son olarak Medkit prefab'ının "Mesh Renderer" component'indeki "Materials" adı altında yer alan "Element 0"ına değer olarak "Medkit" materyalini verin.

Medkit'imiz şimdi çok daha güzel duruyor:



Özet Geçecek Olursak...

Bu bölümde pek fazla birşey yapmadık. Önceden gördüğümüz konseptleri (instantiate, rastgele sayı üretmek) kullanarak oyuna medkit objesi ekledik sadece.

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

Unity 3D İle Oyun Programlama

Bölüm 17: Oyuna Roketatar Ekleme



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



All code snippets are licensed under CC0 (public domain)

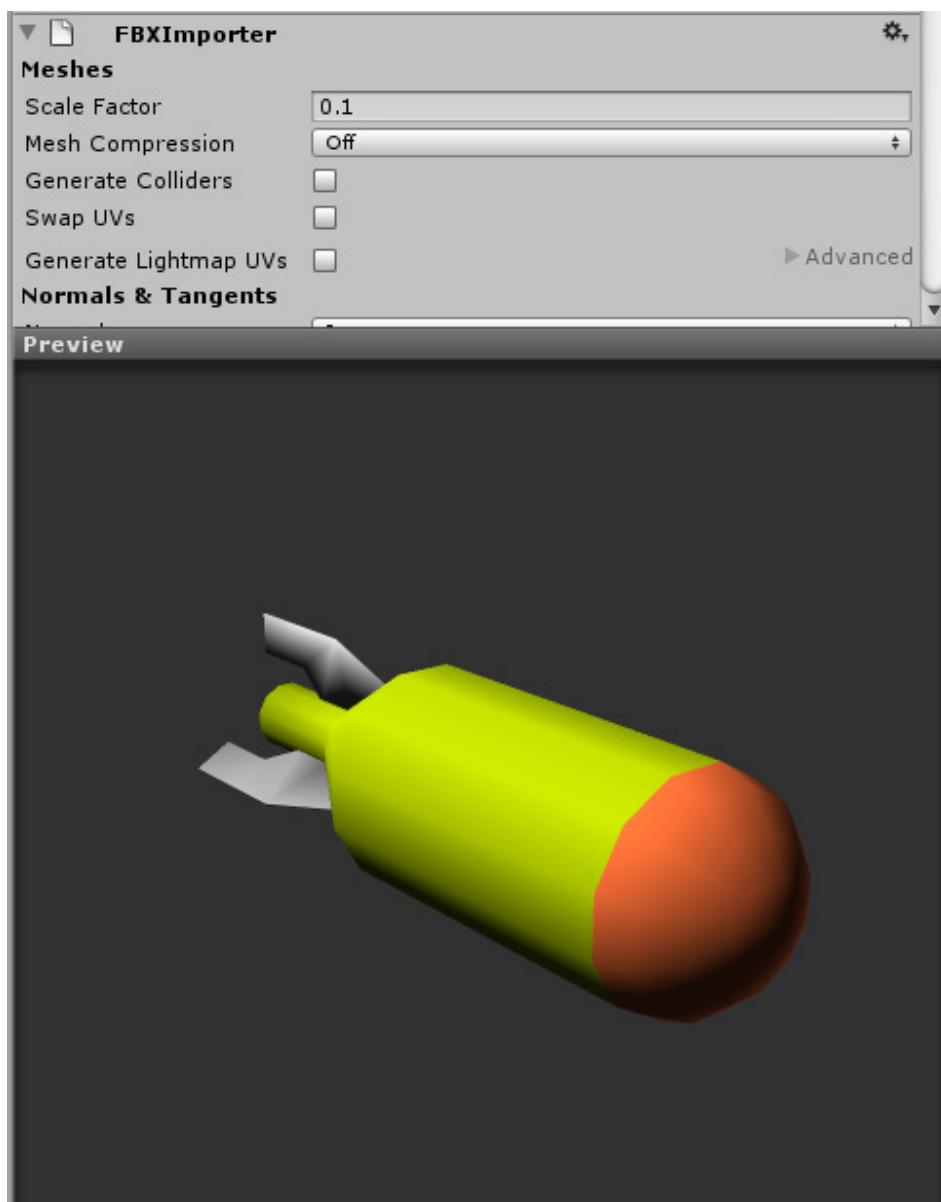


This document except code snippets is licensed with
Creative Commons Attribution-NonCommercial 3.0 Unported

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 sürükleyn. "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.GetKeyDown(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. Şimdiye "_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 yapacağz.



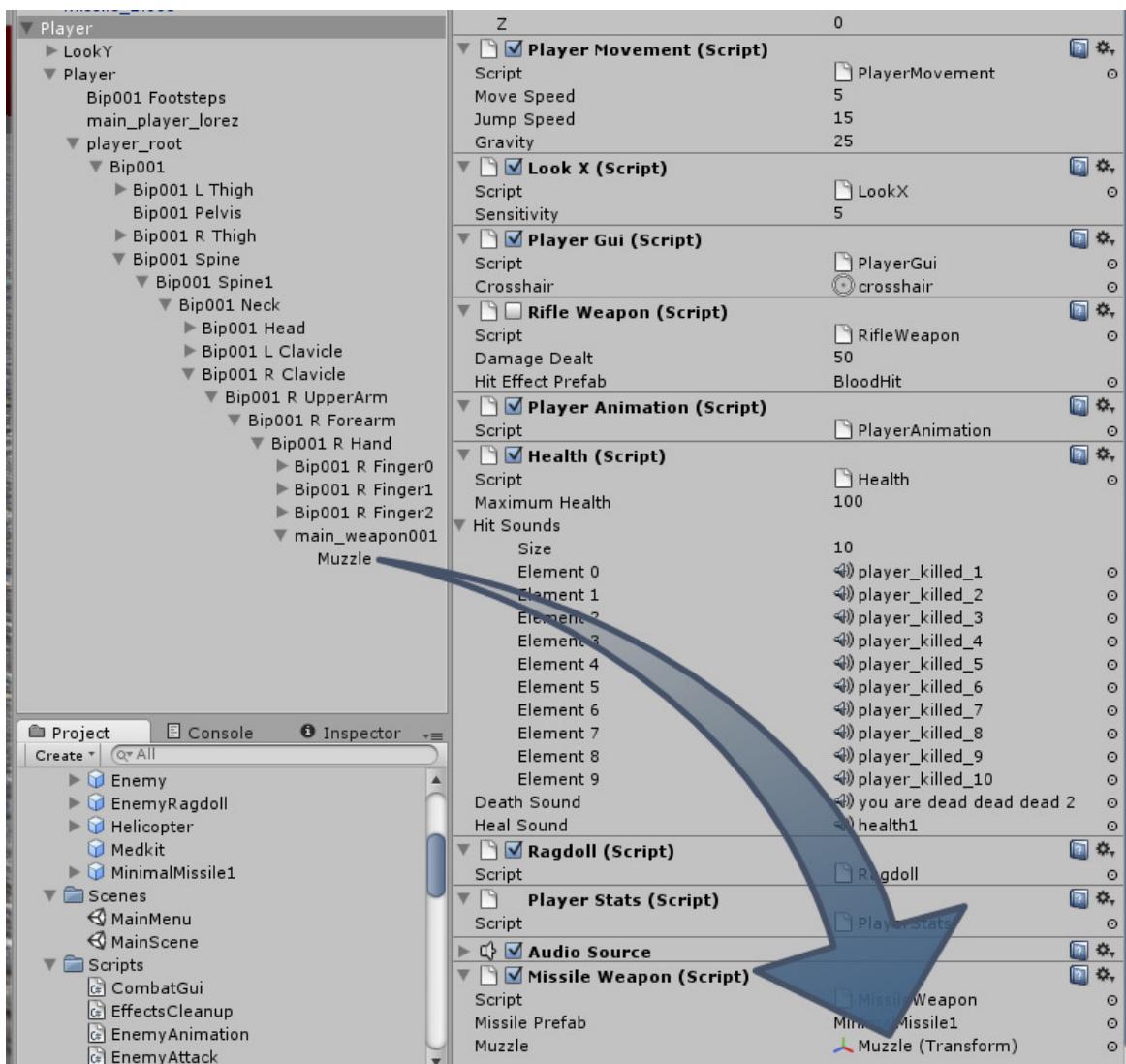
Görsel 17.2: Player'ın silahı seçili halde

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 ateslenecek. Muzzle'ın rotation'ını (270, 0, 0) yapmak bende iş gördüm.

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.GetKeyDown(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 füzeeye "Box Collider" verin (bunun için sahneye bir tane roket klonu koyn, 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 çağırıralım:

```

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 silahlarınızdan çıkan roket ateşlenir ateşlenmez sizi öldürüyor. Çünkü füzenin collider'i bazen player'ın bir collider'ı ile temas edebiliyor.

Missile scriptini şu şekilde güncelle�erek 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 yazabilirimiz.

"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. ([Konuya 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 içeriği 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.GetKeyDown(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. satırı 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     {
```

```

11     GameObject m = Instantiate(_missilePrefab, _muzzle.position, _muzzle.rotation) as GameObject;
12     m.transform.forward = direction;
13 }
14 }
```

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

Unutmayın, bir script (class) başka bir class'ın child'ı ise sanki parent class'taki kodlar child class'ta yazılmış 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 davranışını sağlıyoruz. Böylece ateş edince Weapon scriptindeki boş Shoot fonksiyonu değil MissileWeapon scriptindeki doldolu Shoot fonksiyonu çağrılıyor.

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

```

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(RaycastHit hitInfo, Vector3 direction)
13     {
14         if (hitInfo.transform == null)
15         {
16             return;
17         }
18
19         Health enemyHealth = hitInfo.transform.GetComponent<Health>();
20         if (enemyHealth != null)
21         {
22             enemyHealth.Damage(_damageDealt);
23
24             Vector3 hitEffectPosition = hitInfo.point;
25             Quaternion hitEffectRotation = Quaternion.FromToRotation(Vector3.forward, hitInfo.normal);
26             Instantiate(_hitEffectPrefab, hitEffectPosition, hitEffectRotation);
27         }
28     }
29 }
```

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 çalıştırılmıyor. 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 yağıdırıyor. Farkettiyseniz "inheritance" sayesinde RifleWeapon ile MissileWeapon scriptleri oldukça rahatladı.

Bundan böyle her iki silahda 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 verdığınız 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.

Yaptamamız 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.GetKeyDown(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 >=
32                         Camera.main.WorldToScreenPoint(_muzzle.position).z))
33             {
34                 foundTarget = true;
35                 direction = hitInfo.point - _muzzle.position;
36                 direction.Normalize();
37             }
38             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 önumüzde 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. Döndürulen 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 önumüzde 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 önumüzde de bir düşman varsa önumüzdeki 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 }
```

```

19     Health enemyHealth = hitInfo.transform.GetComponent<Health>();
20     if (enemyHealth != null)
21     {
22         enemyHealth.Damage(_damageDealt);
23
24         Vector3 hitEffectPosition = hitInfo.point;
25         Quaternion hitEffectRotation = Quaternion.FromToRotation(Vector3.forward, hitInfo.normal);
26         Instantiate(_hitEffectPrefab, hitEffectPosition, hitEffectRotation);
27     }
28 }
29 }
```

Çok basit: eğer namlumuzun ucunda düşman yoksa ateş etmiyoruz.

MissileWeapon:

```

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(bool foundTarget, RaycastHit hitInfo, Vector3 direction)
10    {
11        GameObject m = Instantiate(_missilePrefab, _muzzle.position, _muzzle.rotation) as GameObject;
12        m.transform.forward = direction;
13    }
14 }
```

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:

```

01 using UnityEngine;
02 using System.Collections;
03
04 public class Weapon : MonoBehaviour
05 {
06     [SerializeField]
07     protected Transform _muzzle;
08
09     [SerializeField]
10     float _fireDelay = 0.3f;
11
12     float _nextFireTimeAllowed = -1.0f;
13
14     void Start()
15     {
16         Screen.lockCursor = true;
```

```

17    }
18
19    void Update()
20    {
21        if (Input.GetKeyDown(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 >=
39                Camera.main.WorldToScreenPoint(_muzzle.position).z))
40            {
41                foundTarget = true;
42                direction = hitInfo.point - _muzzle.position;
43                direction.Normalize();
44            }
45            else
46            {
47                direction = Camera.main.ViewportToWorldPoint(new Vector3(0.5f, 0.5f, 50)) - _muzzle.position;
48                direction.Normalize();
49            }
50
51            RaycastHit gunHitInfo;
52            if (Physics.Raycast(_muzzle.position, direction, out gunHitInfo))
53            {
54                foundTarget = true;
55            }
56
57            Shoot(foundTarget, gunHitInfo, direction);
58        }
59    }
60
61    virtual protected void Shoot(bool foundTarget, RaycastHit hitInfo, Vector3 direction)
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.