ORİJİNAL KAYNAK: http://noobtuts.com/unity-tower-defense-game-step-1-introduction

AŞAMA 1 – GİRİŞ

Bu tutorialde birlikte 3 boyutlu olan ve özgün bir grafik tarzına sahip bir Tower Defense oyunu tasarlayacağız. Oyunu yapmak hiç de zor değil, endişelenmeyin!

Bu ilk aşamada neler gerektiğinden, oyun mekaniğinden ve oyunu yaparken kullanacağımız şeylerden (asset) bahsedeceğiz.

Bilmeniz Gerekenler

Tutoriale başlamadan önce Unity'nin temel özelliklerini biliyor olmalısınız. Bilmiyorsanız 2 boyutlu Pong oyunu yaptığımız önceki tutorialimize bakabilirsiniz:
http://www.yasirkula.com/2013/07/24/unityde-pong-oyunu-tasarlamak-ceviri-ders/

Ayrıca script yazmak konusunda da aşinalığınızın olması sizin için iyi olur. Eğer script yazmak konusunda bilginiz yoksa üstteki Pong tutorialine bakmadan önce *Çaylaklar İçin UnityScript Kılavuzu*'na da göz atmalısınız:
http://yasirkula.wordpress.com/2011/08/09/caylaklar-icin-unityscripte-javascript-baslangic-kilavuzu-hicbir-programlama-gecmisi-gerekmez/

Gerekli Unity versiyonu

Herşeyden önce elinizin altındaki Unity sürümü en azından 4.0 olsun çünkü Unity'e bu versiyonda gelen **Mecanim** animasyon sistemini kullanacağız.

Ayrıca Unity'nin **Pathfinding** (yol bulma) teknolojisini de kullanacağız. İstersek bunun yerine kendimiz kod yazarak da sahadaki düşmanları hareket ettirebilirdik ama böylesi daha kolay. Ne yazık ki Pathfinding teknolojisi sadece Unity Pro ile kullanılabiliyor (Çevirmen Eklemesi: Unity 4.2 ile beraber kullanacağımız pathfinding sistemi artık ücretsiz. Yani Pro'ya ihtiyacınız yok.). Yani önünüzde iki seçenek var:

- 30 günlük Unity Pro Trial'ini aktif edin
- Yaratıklar için kendi hareket etme scriptinizi yazın

Bunlara ek olarak bir de Unity'nin özel efektlerini (Image Effects) kullanacağız. Bunlar da sadece Unity Pro ile kullanılabiliyor ama bu efektler opsiyonel bir tercih ve olmasa da bir sorun teşkil etmez.

Oyun Mekaniği

Eğer Tower Defense nedir bilmiyorsanız açıklayayım:

- Yaratıkların birkaç saniyede bir spawn olduğu (oluştuğu) bir nokta vardır

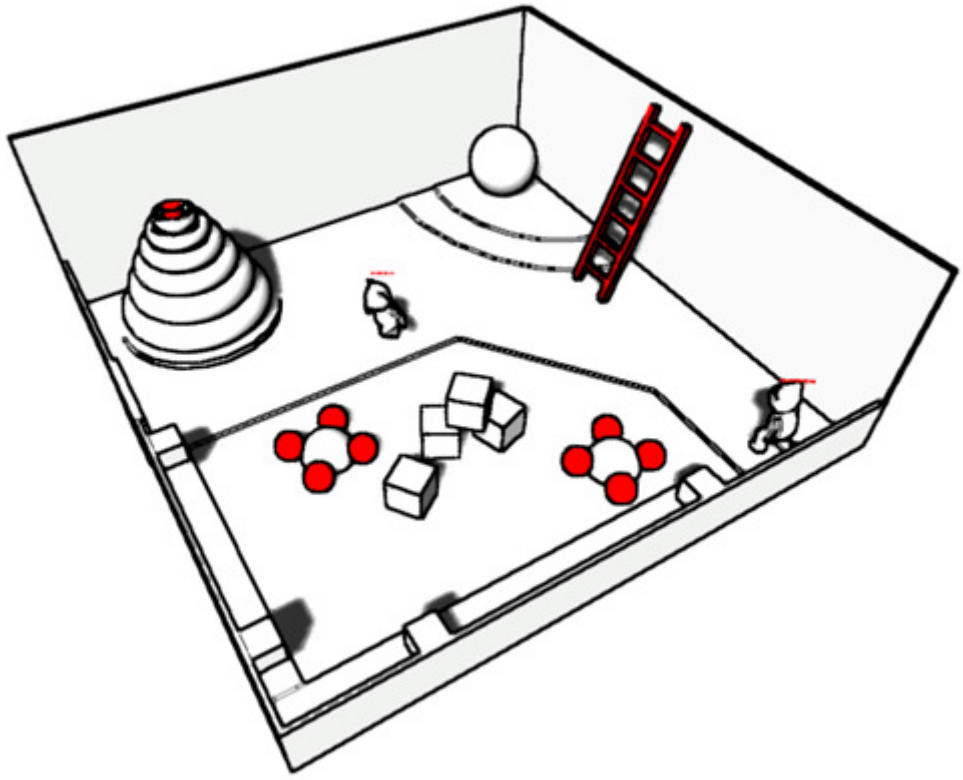
- Yaratıklar bir güzergah boyunca hareket ederek kalenize ulaşmaya çalışırlar
- Oyuncu olarak siz stratejik noktalara kuleler (tower) dikerek bu yaratıkları öldürmeye çalışırsınız
- Yaratık öldürdükçe yeni kuleler dikmek için kullanabileceğiniz para kazanırsınız
- Eğer yaratıklar kalenize ulaşıp onu yokederlerse oyunu kaybedersiniz
- Eğer tüm yaratık akınlarını savuşturabilirseniz oyunu kazanırsınız

Biz oyunumuzda kuleleri dikmek için belli başlı noktalar (Towerslot) belirleyeceğiz ve kuleler sadece bu noktalar üzerine dikilebilecek. Böylece büyük bir yükten de kurtulmuş olacağız (heryere kule dikebilecek olsaydık kodlama yapmak da çok daha zorlaşırdı).

Grafik Stili

İki seçeneğimiz var: ya gerçekçi grafiklerle bezeli olan ve yüzlerce yüksek kaliteli 3D model, kaplama gerektirecek bir stil seçeceğiz ya da çok daha basit ama yenilikçi ve özgün bir stil seçeceğiz.

Sizi bilmem ama ben ikinci seçeneği seçtim. Böylece hem yüksek kalitede modeller oluşturma zahmetinden kurtulmuş olacağım hem de orijinal bir stil yakalamış olacağım. Unity'nin hazır modellerini (küpler, küreler, silindirler) kullanacağız. Oyunumuzda sadece iki renk kullanacağız: beyaz ve kırmızı. Ve oyunun aşağıdaki gibi çizgi film tadında görünmesi için az önce bahsettiğim hazır Unity özel efektlerini (Unity Basic kullanıyorsanız sizin oyununuz çizgi film tadında görünmeyecek) kullanacağız:



Düşmanlarımız olan ayıcıklar sahanın sağından doğacak olup sol tarafa ulaşmaya çalışacaklar. Etrafında dört kırmızı küre bulunan cisimler ise kulelerimiz. Diğer herşey ise sadece dekoratif amaçlı.

Oda olabildiğince sade. Beyaz renklerin kombine olarak kullanılmasıyla ve kırmızı bir merdivenin dikkat çekmesi için yerleştirilmesiyle oluşmuş durumda. İşin güzel yanı herkes bir saatten az sürede, sadece Unity'nin hazır modellerini kullanarak buna benzer bir sahne oluşturabilir.

Assetler

Not: Oyunumuzda kullandığımız 3D modellerden tutun da seslere, kaplamalara kadar herşey birer asset olarak adlandırılır.

Harita

Dediğim gibi haritayı oluştururken Unity'nin modellerini kullanacağız. Bu modelleri **GameObject -> Create Other -> Cube/Sphere/Capsule/Cylinder/Plane** şeklinde oluşturabiliyoruz. Tek yapacağımız bunları sahneye ekleyip evirip çevirerek güzel bir sahne oluşturmak.

Kaplamalar

Oyunumuzda sadece beyaz ve kırmızı renkleri kullanacağız. Ama Unity'de her 3D modelin renklendirilmek için bir kaplamaya ihtiyacı vardır. Bu yüzden 4×4 pixel boyutlarında 2 minicik PNG dosyası kullanacağız: biri tamamen beyaz öteki ise tamamen kırmızı. (Bu 2 PNG dosyasını Paint'ten kendiniz oluşturun. Özellikler'den 4×4 bir tuval oluşturup beyaz PNG dosyasını *texcolor_white* olarak, kırmızı PNG dosyasını da *texcolor_red* olarak adlandırın.)

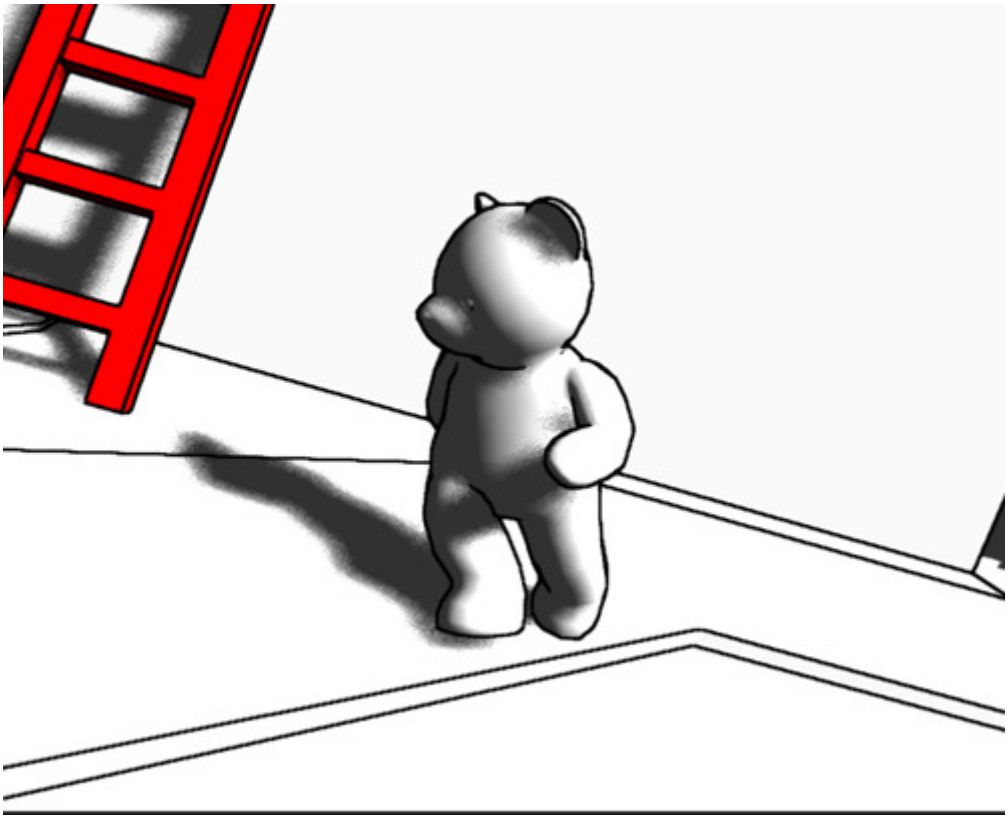
Shaderlar

Shader dediğimiz şey 3 boyutlu modelleri render ederken özel grafik efektleri uygulayan ufak scriptlerdir. Eğer az önce gördüğümüz resme geri dönecek olursak çoğu objenin etrafındaki siyah çizgileri görebiliriz. Bu çizgiler ne 3D modelin bir parçasıdır ne de kaplamanın (texture). Bu çizgiler bir shader tarafından her bir karede hesaplanmaktadır. Bunun için kullanacağımız shader **Edge Detect Effect Normals**'tır ve bu shader Unity Pro'daki Unity Image Effects paketi ile beraber gelmektedir. Bu paketi **Assets->Import Package** vasıtasıyla projenize import edebilirsiniz. (Eğer Unity Basic kullanıyorsanız bu aşamayı es geçin.)

Shaderı kullanmak için onu sahnedeki objelerin materyallerinin "Shader" kısmından seçmelisiniz (sahneyi texcolor_white ve texcolor_red resim dosyaları ile renklendirirken Unity otomatik olarak bu isimlerde materyaller oluşturacak. Bu materyallerin "Shader" özelliklerine tıklayıp menüden ilgili shader'ı seçmeniz yeterli olacaktır.).

Düşman Yaratıklar

Daha önceden yapılmış "Unity Mecanim Demo Project" adında bir proje bulunmakta (https://www.assetstore.unity3d.com/#/content/5328)(İndirmenize gerek yok) ve bu proje içerisinde animasyon ve Pathfinding desteğiyle gelen hazır bir ayıcık bulundurmakta. İşte biz de bu ayıcığı kullanarak hazıra konacağız. Hem grafik stilimize uyuyor bu ayıcık hem de kendimizin animasyon oluşturmamıza ya da Pathfinding ile uğraşmamıza gerek bırakmıyor. Üzerine beyaz kaplama atanmış olarak kullandığımızda ayıcığımız şekildeki gibi duracak:



Ayıcığı hareket ettirmek için şunun gibi bir kod kullanacağız:

```
1GetComponent().destination = pos;
```

Böylece ayıcık hedefe giden en uygun yolu bulacak ve yürüme esnasında içindeki hazır script sayesinde animasyon geçişlerini otomatik olarak halledecek. *Bu harika birşey*!

Projede kullanacağımız ayıcık modelini depolayan paketi şu linke tıklayarak indirin ve projenize import edin (Böylece **TeddyPrefab** adında bir dosya gelecek):
http://noobtuts.com/content/unity/tower-defense-game-step-1-introduction/teddy.unitypackage

AŞAMA 2 – ANA MENÜ

Bu aşamada bir menü oluşturacağız ve oyuncu oyuna başlayınca önce bu menüyü görecek. Bu aşamadan önce Unity'nin GUI sistemiyle ilgili bilgi edinmeniz yararınıza olacaktır.

Projeyi Oluşturmak ve Menü Sahnesi (Scene)

Eğer hâlâ bir proje oluşturmadıysanız hemen bir yane oluşturun. Ardından az önce oluşturduğumuz kırmızı ve beyaz PNG dosyaları ile ayıcığı barındıran paketi projenize import edin. Şimdi **File->Save Scene** ile sahneyi *scene_menu* adıyla kaydedin.

Menü scriptini yazmak

Menu.cs adında yeni bir C# script oluşturun. Start ve Update metodlarına ihtiyacımız olmayacağından onları silin ve GUI elemanlarıyla çalışmaya yarayan OnGUI fonksiyonunu scriptinize ekleyin:

```
using UnityEngine;
using System.Collections;

public class Menu : MonoBehaviour {

    void OnGUI() {

    }
}
```

Buraya kadar bilmediğiniz birşey görmedik.

GUILayout alanı oluşturmak

Menümüz için yakında **Play** (Oyna) ve **Quit** (Çık) butonları oluşturacağız. Unity'nin **GUILayout** class'ını kullacağız ve böyle GUI class'ının aksine butonların pixel cinsinden koordinatları ve boyutları ile uğraşmak zorunda kalmayacağız; Unity bu işi bizim için hallediyor.

Öncelikle GUILayout elemanlarının dizileceği bir alan (Area) tahsis edelim. Eğer bir alan tahsis etmezsek GUILayout elemanları tüm ekranı alan olarak kullanırlar. Kullanacağımız bu alan ekranın belli bir yüzdesini kaplayacak ve ekranın boyutu değişirse alanın boyutu da değişerek hep bu yüzdeye sadık kalacak. Bize gerekenler birkaç public (değeri Inspector'da gözüken ve başka scriptler tarafından değiştirilebilen) değişken ve biraz da matematik:

```
using UnityEngine;
using System.Collections;

public class Menu : MonoBehaviour {

    public float widthPercent = 0.3f; // % olarak genişlik
    public float heightPercent = 0.3f; // % olarak yükseklik

    void OnGUI() {
        // GUILayout'ları içine sokacağımız alanın koordinatlarını ve
ebatlarını belirle
        Rect r = new Rect(Screen.width * (1 - widthPercent) / 2,
                          Screen.height * (1 - heightPercent) / 2,
                          Screen.width * widthPercent,
                          Screen.height * heightPercent);

        // Menüyü belirlediğimiz bu alana çiz
        GUILayout.BeginArea(r);
            GUILayout.BeginVertical("box");

                // YAPILACAK: butonları oluştur

            GUILayout.EndVertical();
        GUILayout.EndArea();
```

```
22      }
23  }
24
25
```

Yukarıdaki kod ile ekranda ortalanmış olan ve ekranın genişliğinin %30′ü genişlikte, ekranın yüksekliğinin %30′u yükseklikte olan bir dikdörtgensel alan (**Rect**) oluşturduk oluşturduk ve bu dikdörtgensel alanı **GUILayout.BeginArea** ile bir alana (Area) atadık. Böylece artık çizeceğimiz **GUILayout** elemanları bu alanın içerisinde oluşacak. Ardından **GUILayout.EndArea** komutu ile bu alanı kapattık.

GUILayout.BeginVertical("box"); komutuyla GUILayout elemanlarına yukarıdan aşağıya doğru dizilmelerini söyledik ve içine **"box"** (kutu) parametresi ekleyerek butonları çizeceğimiz bu alanının etrafının basit bir kutu ile çevrilmesini sağladık (menü daha güzel dursun diye).

Play butonunu oluşturmak

En önemli parçayı, yani **Play** butonunu oluşturalım. Bunun için **GUILayout.Button** komutunu kullanmalıyız. Eğer bu butona basılırsa "**scene_main**" adındaki oyun sahnemizi (scene) yükleyeceğiz. Bu sahneyi ise bir sonraki aşamada oluşturacağız.

İşte Play butonu için gerekli kodumuz:

```
1
2  using UnityEngine;
3  using System.Collections;
4
5  public class Menu : MonoBehaviour {
6
7      public float widthPercent = 0.3f; // % olarak genişlik
8      public float heightPercent = 0.3f; // % olarak yükseklik
9
10     void OnGUI() {
11         // GUILayout'ları içine sokacağımız alanın koordinatlarını ve
ebatlarını belirle
12         Rect r = new Rect(Screen.width * (1 - widthPercent) / 2,
13                           Screen.height * (1 - heightPercent) / 2,
14                           Screen.width * widthPercent,
15                           Screen.height * heightPercent);
16
17         // Menüyü belirlediğimiz bu alana çiz
18         GUILayout.BeginArea(r);
19             GUILayout.BeginVertical("box");
20
21                 if (GUILayout.Button("Play"))
22                     Application.LoadLevel("scene_main");
23
24             GUILayout.EndVertical();
25         GUILayout.EndArea();
26     }
27 }
28
```

Gördüğünüz gibi GUILayout.Button komutu ile yeni bir buton oluşturduk ve başına **if** koyarak eğer bu butona basılırsa yapılacak şeyleri yazdık: Application.LoadLevel("scene_main"). Bu komut sayesinde "scene_main" isimli sahneyi yükleyip ona geçiş yapıyoruz.

Quit butonunu oluşturmak

Tekrar bir buton oluşturacağız ama bu sefer butona tıklanırsa **Application.Quit();** komutu vasıtasıyla oyundan çıkacağız.
Not: oyundan çıkma komutu Unity editöründeyken işe yaramaz.

İşte güncellenmiş kodumuz:

```
1
2
3  using UnityEngine;
   using System.Collections;
4
5  public class Menu : MonoBehaviour {
6
7      public float widthPercent = 0.3f; // % olarak genişlik
8      public float heightPercent = 0.3f; // % olarak yükseklik
9
10     void OnGUI() {
           // GUILayout'ları içine sokacağımız alanın koordinatlarını ve
11 ebatlarını belirle
12         Rect r = new Rect(Screen.width * (1 - widthPercent) / 2,
13                           Screen.height * (1 - heightPercent) / 2,
14                           Screen.width * widthPercent,
15                           Screen.height * heightPercent);
16
17         // Menüyü belirlediğimiz bu alana çiz
18         GUILayout.BeginArea(r);
19             GUILayout.BeginVertical("box");
20
21                 if (GUILayout.Button("Play"))
22                     Application.LoadLevel("scene_main");
23
24                 if (GUILayout.Button("Quit"))
25                     Application.Quit();
26
27             GUILayout.EndVertical();
28         GUILayout.EndArea();
29     }
   }
```

Herşey apaçık ortada sanırım.

Logo oluşturmak

İki butonumuz orada yalnız başlarına kalmasınlar diye onların yanına bir de ufak bir logo ekleyelim. Logo eklemek için kullanacağımız komut ise **GUI.DrawTexture**. Logonun butonların üstüne değil de altına çizilmesi için ise logo çizme kodunu buton çizme

kodlarından önce yazacağız. Böylece önce logo çizilecek ve ardından da üstüne butonlar çizilecek.

Logonun menümüzün sağ üst köşesine hizalanmasının güzel duracağını düşünüyorum. Bunun için yine matematiksel işlemler yapıyorum:

```
Rect l = new Rect(r.x + r.width - logo.width / 2,
                  r.y - logo.height / 2,
                  logo.width,
                  logo.height);
```

Eğer bu kodu anlamadıysanız boşverin gitsin. Zaten çok önemli bir kod değil.

Artık **Texture2D** türünde değişkene depolanmış logomuzu göstermeye de yarayan menü scriptimizin son hali emrinize âmâde:

```
using UnityEngine;
using System.Collections;

public class Menu : MonoBehaviour {

    public float widthPercent = 0.3f; // % olarak genişlik
    public float heightPercent = 0.3f; // % olarak yükseklik

    public Texture2D logo = null;

    void OnGUI() {
        // GUILayout'ları içine sokacağımız alanın koordinatlarını ve
ebatlarını belirle
        Rect r = new Rect(Screen.width * (1 - widthPercent) / 2,
                          Screen.height * (1 - heightPercent) / 2,
                          Screen.width * widthPercent,
                          Screen.height * heightPercent);

        // menünün sağ üstüne logoyu çiz (eğer değeri atanmışsa)
        if (logo != null) {
            Rect l = new Rect(r.x + r.width - logo.width / 2,
                              r.y - logo.height / 2,
                              logo.width,
                              logo.height);
            GUI.DrawTexture(l, logo);
        }

        // Menüyü belirlediğimiz bu alana çiz
        GUILayout.BeginArea(r);
            GUILayout.BeginVertical("box");

                if (GUILayout.Button("Play"))
                    Application.LoadLevel("scene_main");

                if (GUILayout.Button("Quit"))
                    Application.Quit();

            GUILayout.EndVertical();
        GUILayout.EndArea();
    }
```

```
35}
36
37
38
39
40
```

Şimdi kodu **Main Camera**'ya ekleyin ve kendi seçtiğiniz bir logoyu dilerseniz **logo** değişkeninin olduğu yere sürükleyin. Main Camera'nın arkaplan rengini beyaz yapın. Artık menümüz hazır:

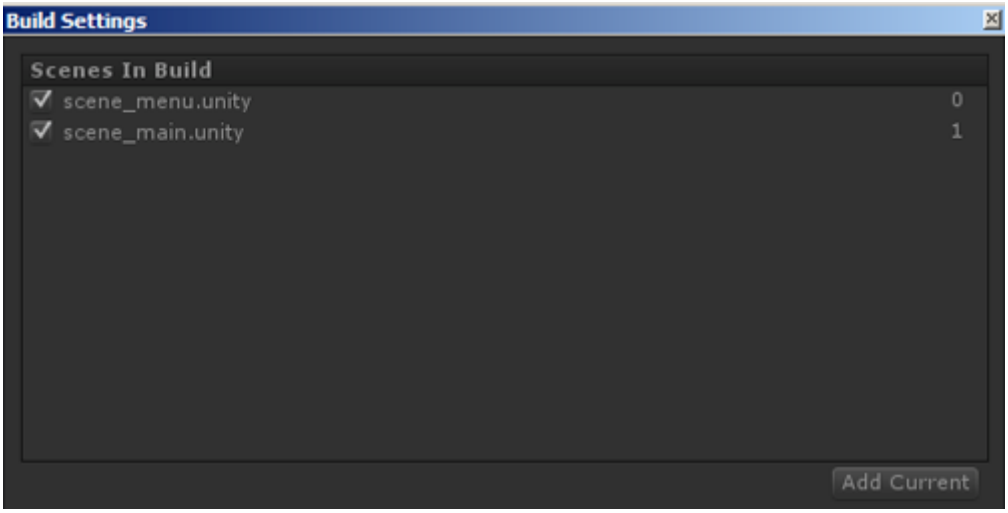
AŞAMA 3 – OYUN ALANI

Bu aşamada oyunun oynanacağı bölümü tasarlayacağız. Hedefimiz küçük ama özgün bir tarza sahip bir bölüm oluşturup bir sonraki aşamada bu bölüme oyun mekaniği ekleyerek canlılık katmak.

Yeni bir sahne

Menü sahnenizi kaydedin ve **File->New Scene** yoluyla yeni bir sahne oluşturup bu sahneyi *scene_main* adıyla kaydedin. İşte oyun sahamızı oluşturacağımız sahne bu.

Hatırlarsanız ana menüde Play butonuna basınca şimdi oluşturduğumuz scene_main sahnesine geçiş yapıyorduk. Eğer menüye geri dönüp Play'e basarsanız muhtemelen hata alırsınız çünkü Application.LoadLevel komutunun çalışması için gidilecek bölümün Build Setting'e ekli olması lazım. Bunun için **File->Build Settings…** yolunu izleyin:



Oyun açılınca ilk açılmasını istediğiniz bölümü (scene_menu) ilk önce sürükleyip listeye bırakın ve ardından sonraki bölümleri (bizim durumumuzda sadece scene_main) tek tek ekleyin.

Işıklandırma

Güneşi taklit eden ve objelerin gölgelerinin oluşmasını sağlayan en popüler ışık türü Directional Light ve biz de bunu kullanacağız. Hemen **GameObject->Create Other->Directional Light** yolunu izleyerek yeni bir tane oluşturun. Eğer Unity Pro ile çalışıyorsanız gölgeleri aktif etmek için ışık objesinin Inspector'undan **Shadow Type**'ı (gölge türü) **Soft Shadows** (yumuşak gölge) yapın.

*Çevirmen Notu: Eğer Unity Basic'in 4.2 veya üstü bir sürümüne sahipseniz artık siz de sahnede gerçek zamanlı gölgeler kullanabilirsiniz. Ama Soft Shadows yerine yumuşatma işleminden geçmeyen ve daha kaba duran **Hard Shadows** kullanabilirsiniz.*

Not: ayrıca ışığın rengini beyaz yapın.

Modeller

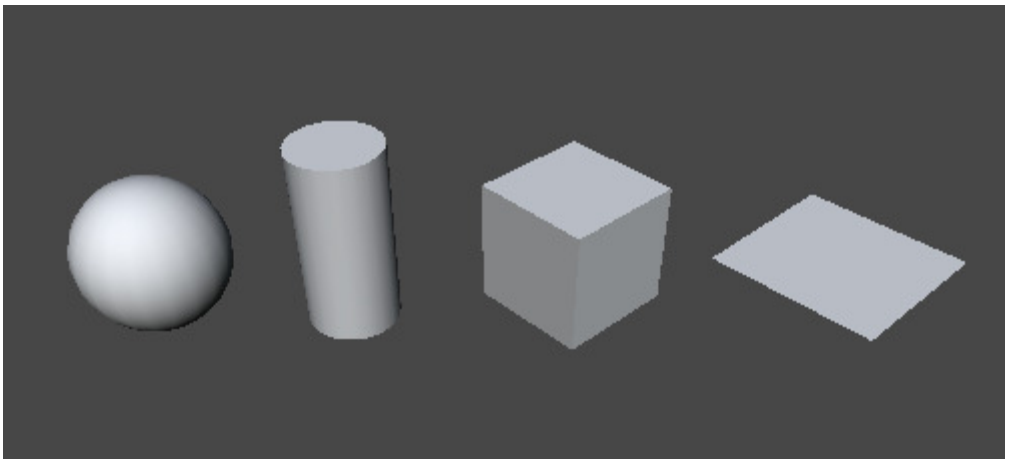
Sahamıza modeller eklemek için iki seçeneğimiz var:

Seçenek 1: CAD

İlk seçenek 3DS Max, Maya veya Blender gibi bir modelleme aracı kullanmaktır. Bu programlar çok güçlüdür ancak kullanması da kolay değildir.

Seçenek 2: Hazır Unity Objeleri

Unity'nin **GameObject->Create Other** menüsü altında küreler (sphere), küpler (cube), düz zeminler (plane) veya silindirler (cylinder) oluşturabiliriz:

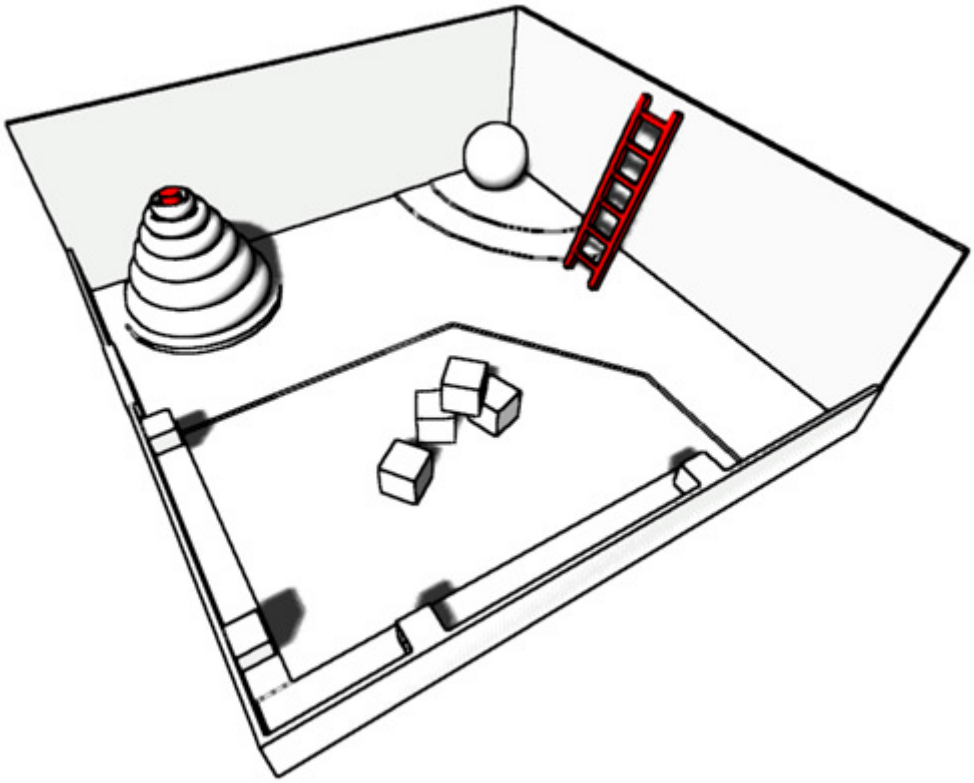


İkinci seçenek daha basit olduğu için onunla yola devam edeceğiz.

Temel şekillerden karmaşık şekillere

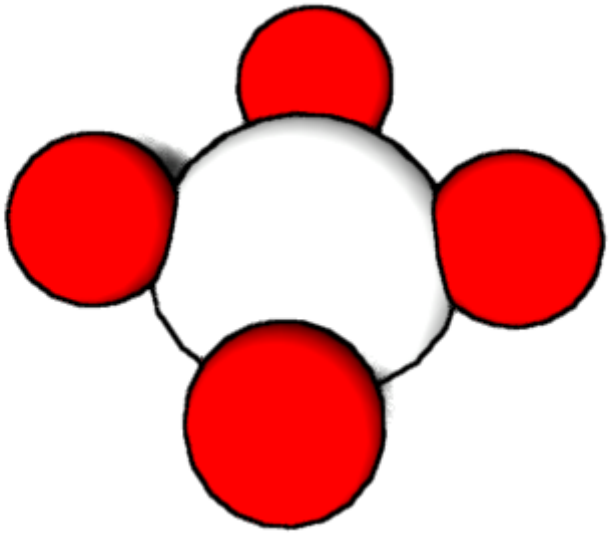
Bir merdiven yapmak istediğimizi varsayalım. Tek yapmamız gereken birkaç küp oluşturup bunları konumlandırmak, döndürmek ve boyutlandırmak. Ta ki sonuç aşağıdaki gibi görünene kadar (merdivene kırmızı renk vermek için texcolor_red texturesini Assets panelinden sürükleyerek Scene panelinde merdiven parçalarının üzerine bırakın):

Tutorialimiz için ben aşağıdaki gibi bir sahne oluşturmayı tercih ettim. Ama siz istediğiniz gibi bir sahne yapabilirsiniz:



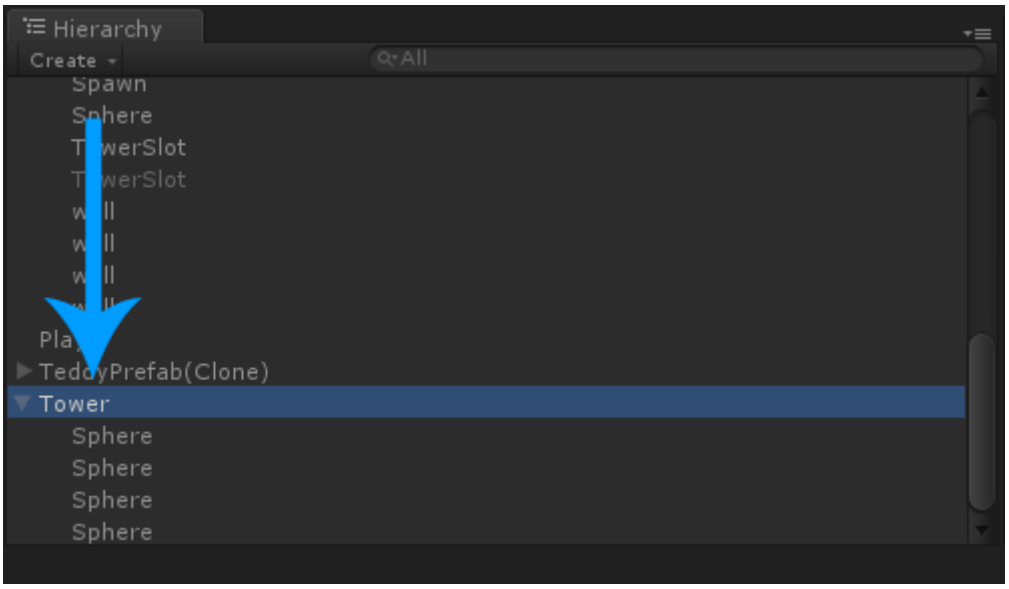
Kule

Kulemizi oluşturma zamanı. Küreler oluşturup onları şekildeki gibi dizerek kule oluşturabilirsiniz:

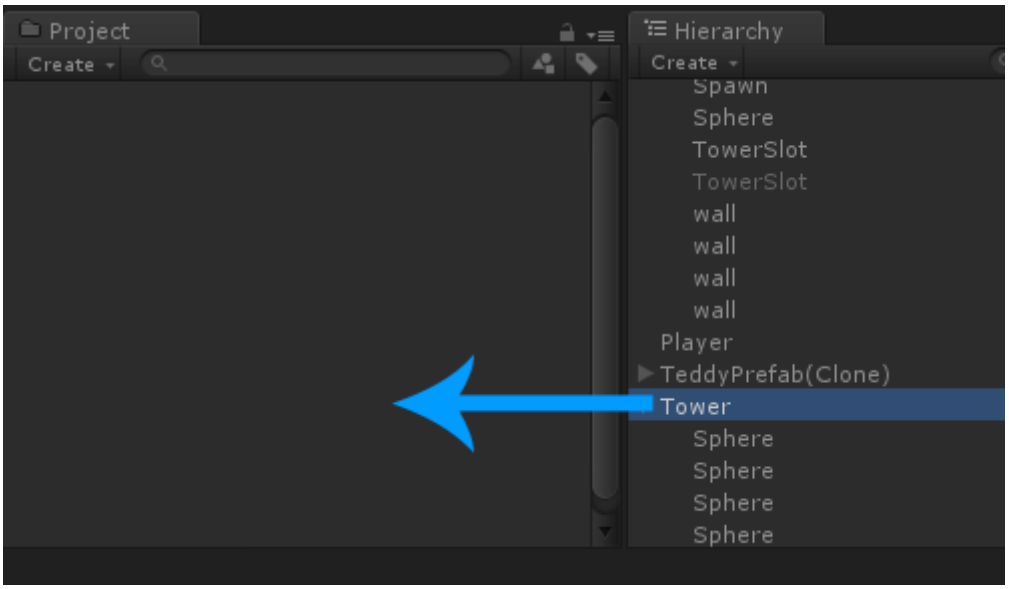


Kulenin henüz oyun sahnemizde olmasını istemiyoruz. Sadece oyuncu tarafından dikildiklerinde oraya gelmelerini istiyoruz. Bize bir **Prefab** lazım şimdi (Objeyi Project panelinde depolamaya yarayan ve sonradan dilediğimizce oluşturmaya yarayan bir şey).

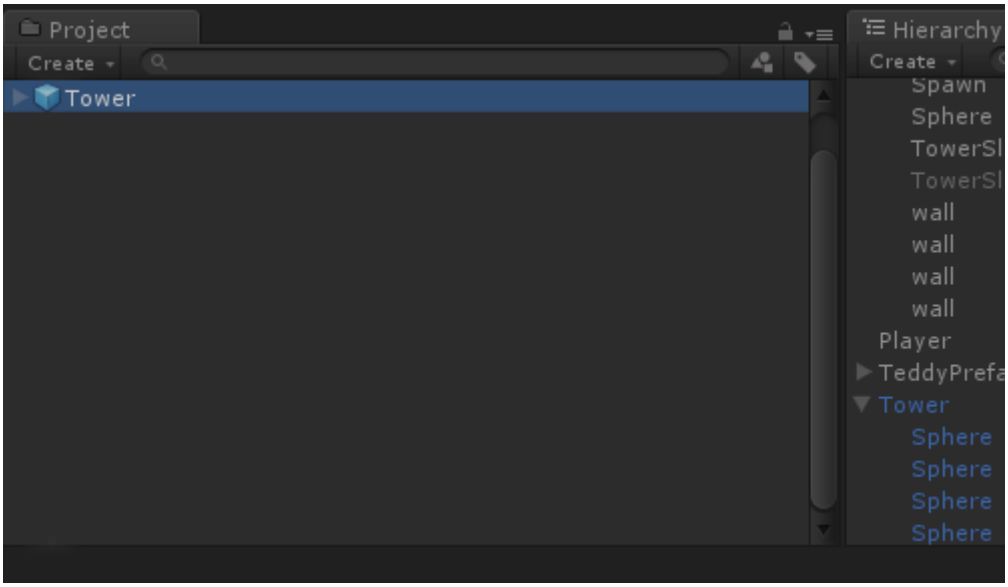
Önce kulenizi oluşturmak için kullandığınız tüm objeleri boş bir obje altında birleştirin. Bunun için önce boş bir GameObject (**Empty GameObject**) oluşturun ve buna “Tower” (kule) adını verin. Ardından Hierarchy panelinden kuleyi oluşturan tüm parçaları bu boş objenin üzerine sürükleyip bırakarak onun child objesi yapın. İşte sonuç:



Son olarak Tower’ı Hierarchy panelinden Project paneline sürükleyin:



Artık Tower objesi bir Prefab’a dönüştü:



Şimdi Tower'ı **Hierarchy** panelinden silebilirsiniz. Bundan böyle Project panelinden Tower prefab'ını sahneye sürükleyerek ya da script yazarak sahnede Tower oluşturabiliriz (birazdan yapacağız da).

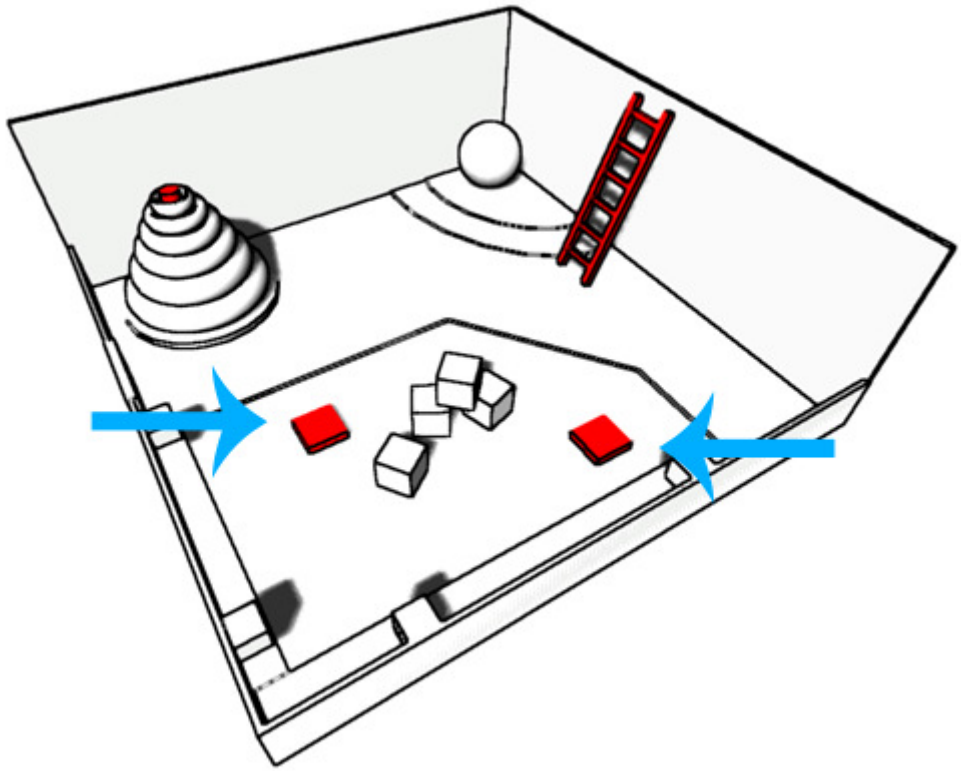
Kurşun

Kulelerimizin ateşleyeceği kurşunlar lazım bize. Bunun için basit bir küre (sphere) yeterli olacaktır. Yeni bir küre oluşturup bu kurşunu uygun boyuta getirin ve ona "Bullet" adını verin. Daha sonra tıpkı Tower'a yaptığımız gibi kurşunu da Hierarchy panelinden Project paneline sürükleyerek onu bir Prefab haline getirin ve ardından sahnedeki kurşunu silin.

Kule Noktaları

Daha önceden bahsettiğim gibi, kuleleri üzerinde oluşturabileceğimiz kule noktaları kullanarak işimizi kolaylaştıracağız (böylece oyuncu sahada istediği her yere kule dikemeyecek). Aslında oyuncu istediği yere kule dikebilse daha iyi olurdu ama o zaman collision konularıyla alakalı bir dolu şeyle uğraşmak zorunda kalırdık.

Kule noktaları için kendi 3D modelinizi kullanabilirsiniz ama ben basitlik açısından kırmızı bir küp kullanmayı tercih ettim. Kendi kule noktalarınızı da sahaya şekildeki gibi dikin:



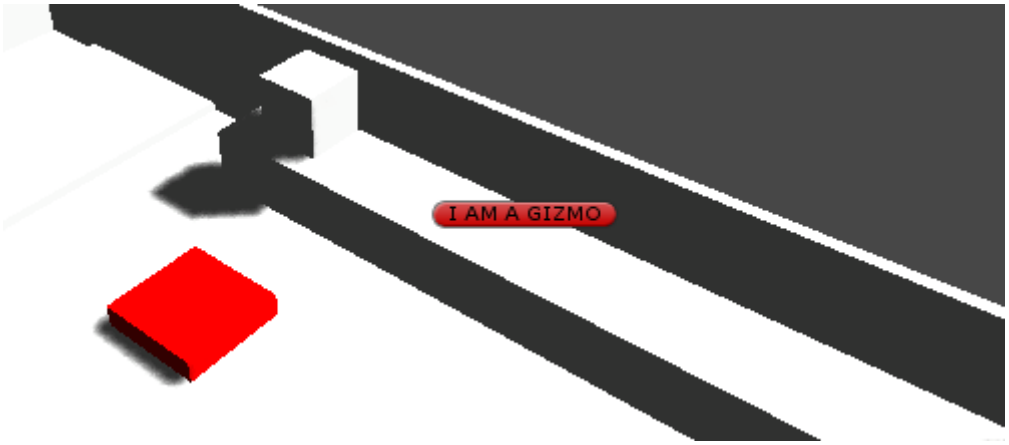
İlerleyen safhalarda bu kule noktaları için bir script yazacak ve oyuncunun üzerlerine tıklayarak bir kule inşa edebilmesini sağlayacağız.

Yaratık doğma noktası ve kale

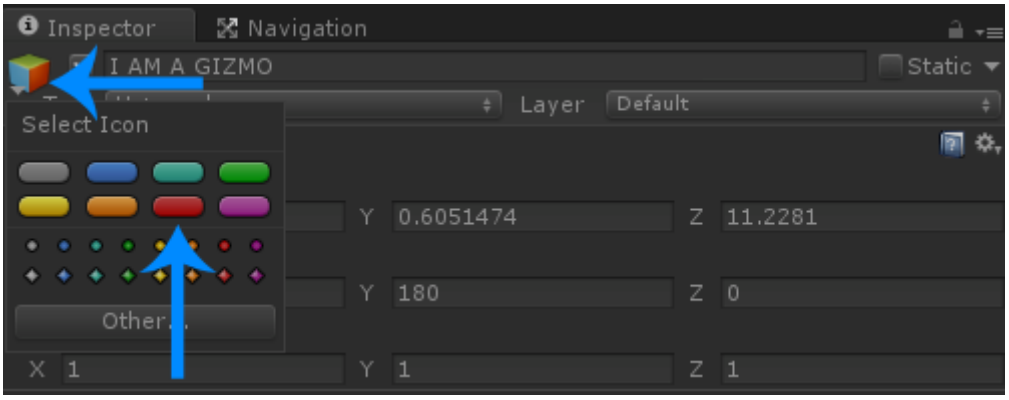
Bir Tower Defense oyunu yapıyoruz ve dolayısıyla yaratıkların doğacağı bir spawn noktasına ve yok etmek için koşacakları bir hedefe (kale) ihtiyacımız var. Sahnemiz gayet küçük olduğu ve içine bir kalenin sığmasının zor olduğu için kale yerine ufak bir küp objesi ile yetineceğiz.

Spawn noktası

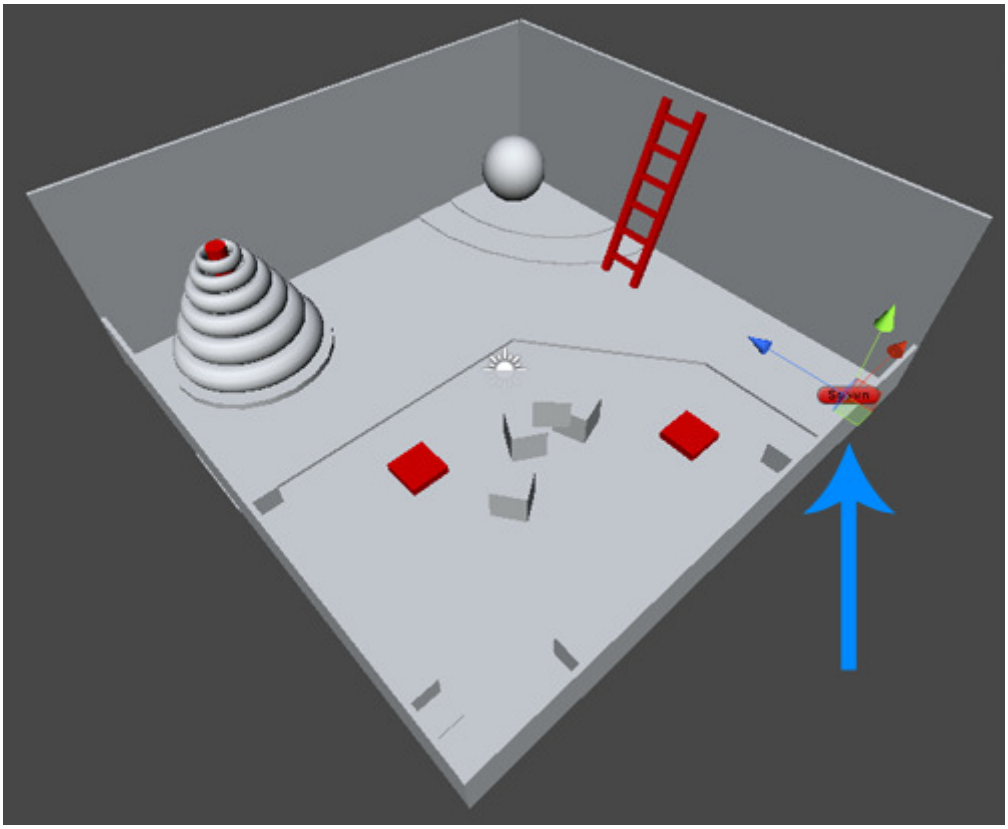
Yaratıkların doğacağı noktanın özel bir cisim olmasına gerek yok. Bu yüzden boş bir obje (**Empty GameObject**) kullanacağız spawn noktası için ve bu noktayı uygun bir yere taşıyacağız. **GameObject->Create Empty** yolunu izleyerek boş bir obje oluşturun ve ismini "Spawn" olarak değiştirin. Ardından obje için basit bir gizmo seçin; böylece objenin yerini daha iyi görebilirsiniz. Gizmo dediğimiz şeyler sadece editörde (Scene panelinde) görebildiğimiz ufak işaretçilerdir ve tamamen opsiyoneldirler. Sadece bazen gerçekten işinizi kolaylaştırabilirler. İşte bir örnek:



Bir objeye gizmo atamak için resimdeki talimatları izleyebilirsiniz:



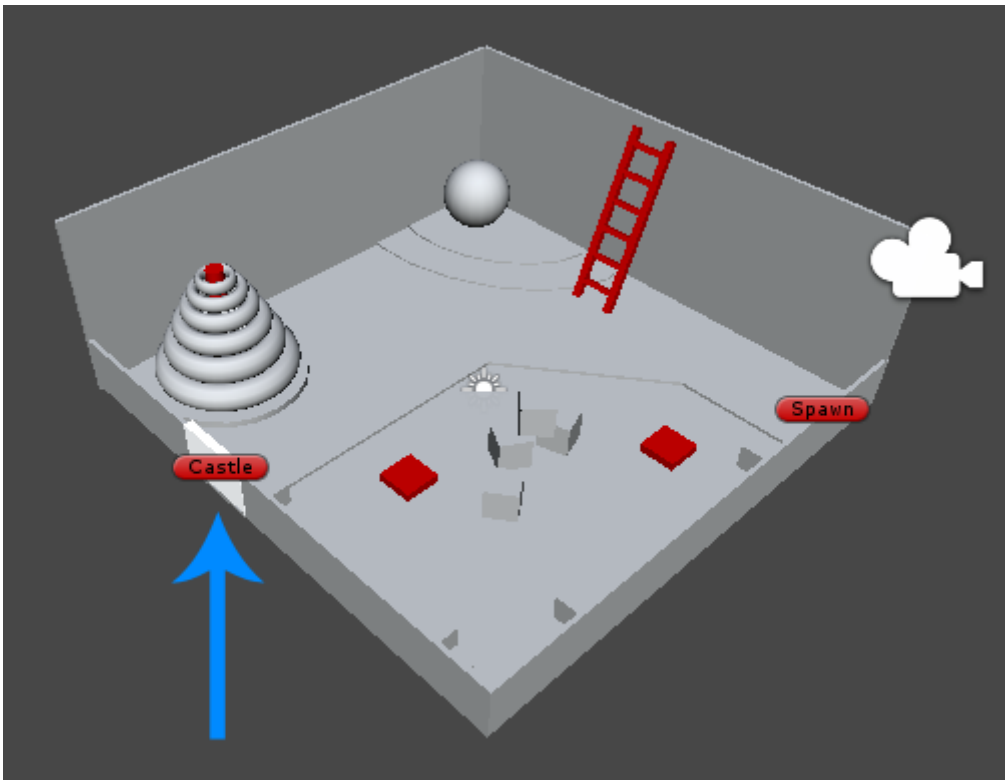
Objemiz için bir gizmo seçtikten sonra artık onun yerini Scene panelinde daha rahat görebilir ve onu kolaylıkla istediğimiz yere taşıyabiliriz. Siz de spawn noktasını şekildeki noktaya taşıyın:



Not: spawn noktasını yerden bir miktar yukarıda konumlandırın ve böylece yaratıkların düzgün spawn olması için onlara yeterli boşlığı sağlayın.

Kale

Bunu yapması oldukça kolay. Bir küp oluşturun, adını “Castle” yapın ve onu sahanın soluna taşıyın. Bu obje için de gizmo kullanarak işinizi kolaylaştırabilirsiniz:



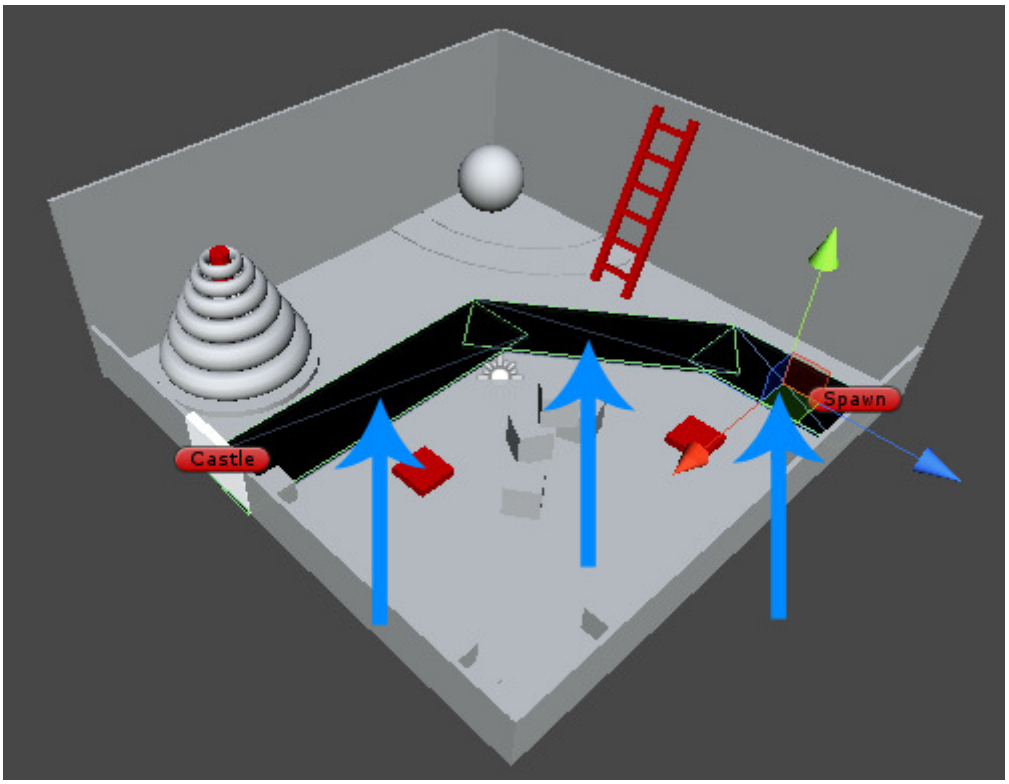
Düşman Güzergahı

Çevirmen Notu: Eğer Unity Basic kullanıyorsanız bu işlemi yapabilmek için Unity sürümünüz en az 4.2 olmak zorunda!

Spawn noktasını ve kalemizi oluşturduk. Eğer şimdi yaratıkları oluşturuyor olsaydık bir işe yaramazdı çünkü hâlâ spawn noktasından kaleye hangi güzergahı kullanarak gidecekleri konusunda fikirleri yok. Bu yüzden bir güzergaha (**Path**) ihtiyacımız var.

Kutular

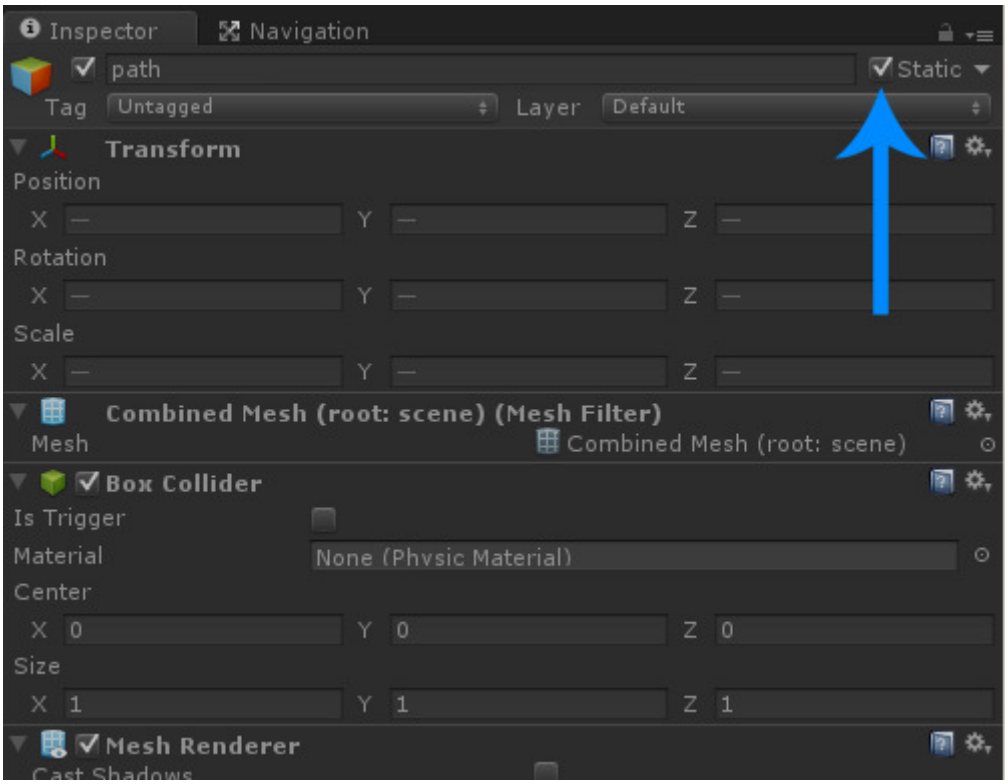
Unity'nin Pathfinding sistemini kullanmak için Unity'e hangi yolların güzergah için uygun olup hangi yerlerin uygun olmadığını söylememiz gerekiyor. Bunun için spawn noktasından kaleye doğru giden üç küp oluşturup şekildeki gibi dizin:



Not: yol siyah renkte çünkü böylece onu daha rahat görebilirsiniz. Ama ilerleyen resimler de yol da beyaz renkte ve gözle görülmesi oldukça zor.

Static

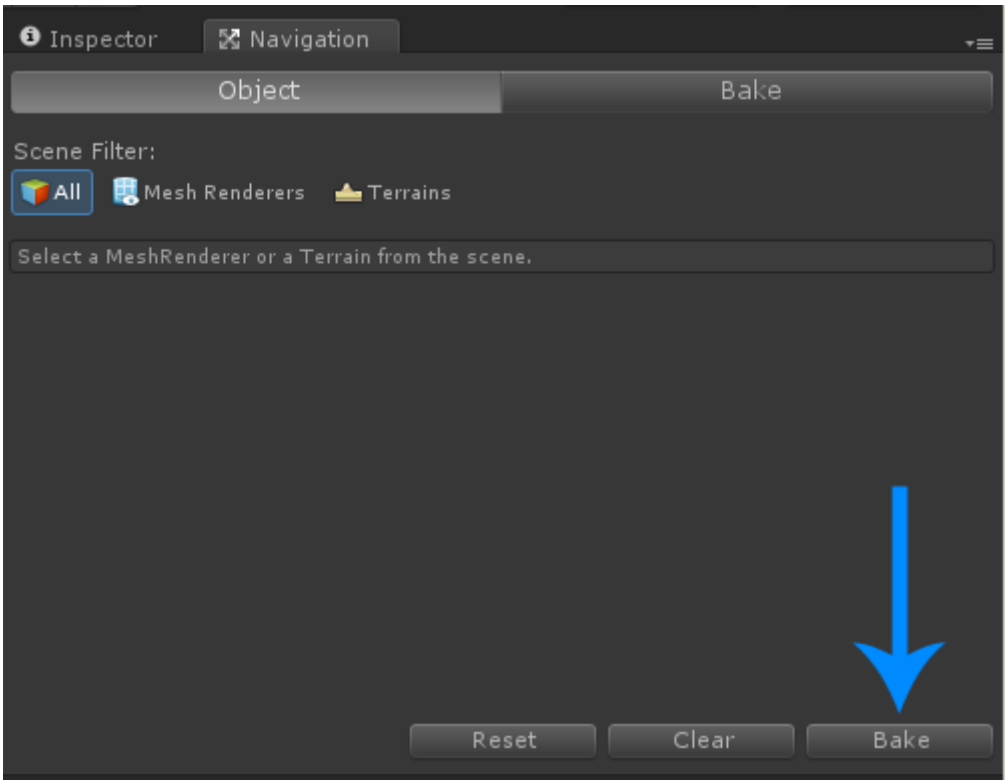
Yol için küpleri oluşturduktan sonra tekini seçin ve Inspector'a bir göz atın. Sağ üst köşede ufak bir "**Static**" seçeneği var. İşte size ipucu: Unity'e bir objenin Pathfinding sistemi için kullanılmasını söylemek için o objeyi **Static** yapmak yeterlidir (bu işlemi yol için kullandığımız üç küp objesine de uygulayın):



Yolu Oluşturmak (Bake Navigation)

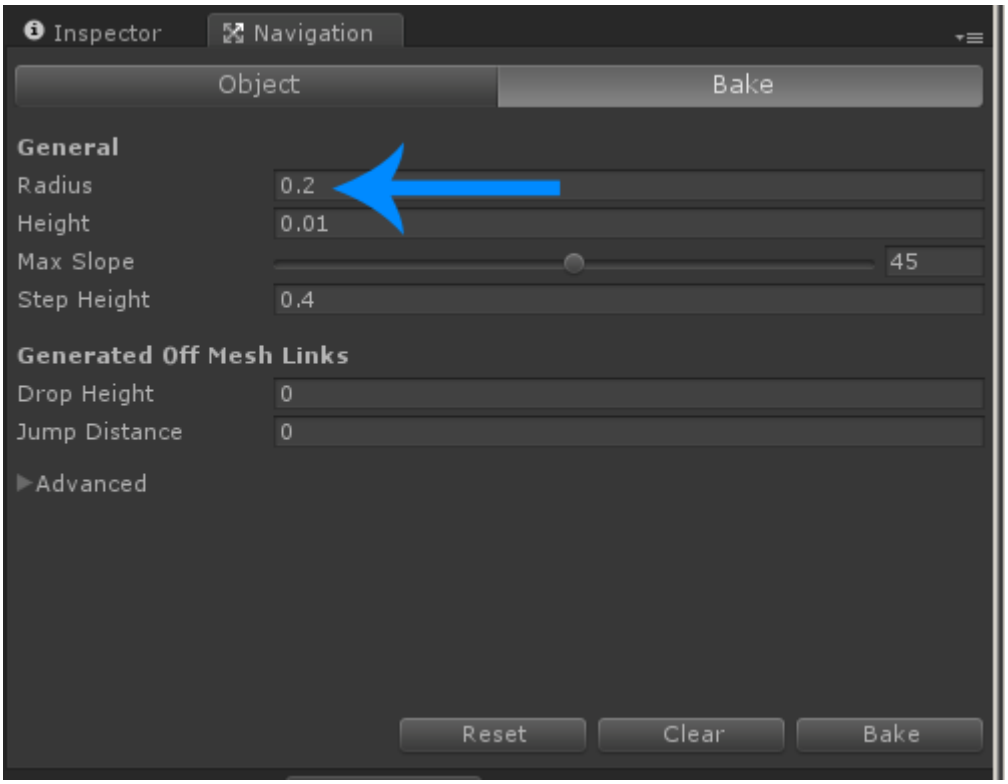
Yapmamız gereken son şey Unity'nin Pathfinding sistemi için kullanabileceği yolları hesaplayacağı tek seferlik bir işlemi uygulamak (yolu her değiştirdiğinizde bu işlemi tekrarlayın)(bu tutorial için bir daha tekrarlamanız gerekmiyor).

Bunu yapmak için yukarıdan **Window**->**Navigation** yolunu izleyin ve **Bake** butonuna basın:



Sonuç Scene panelinde görünür durumda olmalı. Eğer herşey düzgün olduysa Pathfinding sisteminin kullanacağı yol(lar)ı mavi şekilde görebilirsiniz:

Eğer mavi alanı göremiyorsanız Navigation penceresindeki ayarlardan **Radius** ile oynamayı deneyin (muhtemelen değerini düşürmeniz faydalı olur)(ayrıca Height değerini de 0.01 yapmak isteyebilirsiniz):



Unity'nin pathfinding sisteminin çalışması için yaptığımız bu işlemler ilk başta biraz garip gelebilir ama aslında oldukça mantıklı bir çalışma şekli var sistemin. Ve kullanımı da oldukça kolay.

AŞAMA 4 – SCRIPTLER

Bu son aşamada oyun mekaniğini uyarlamak için bazı scriptler yazacağız.

Spawn Scripti

Scripti oluşturmak

Daha önce sahanın sağ tarafında düşmanların her birkaç saniyede bir doğması için bir Spawn noktası oluşturmuştuk. Şimdi düşmanların gerçekten spawn olması için script yazma zamanı. "Spawn.cs" adında yeni bir script oluşturun ve ihtiyacımız olmayacağı için Start fonksiyonunu silin:

```
1
2  using UnityEngine;
3  using System.Collections;
4
5  public class Spawn : MonoBehaviour {
6
7      // Update is called once per frame
8      void Update () {
9
10     }
11 }
```

Zamanlayıcıyı (kronometre) kurmak

Eğer yaratıkların her birkaç saniyede bir doğmasını istiyorsak bir zamanlayıcıya ihtiyacımız olacak. Bir yaratık doğmadan önce geçmesi gereken zamanı (**interval**) depolayan bir değişken oluşturalım ve ardından Update metodunun içinde geçen zaman miktarını sayalım. Eğer yeteri kadar zaman geçmişse yeni bir yaratık oluşturalım:

```
1
2
3  using UnityEngine;
   using System.Collections;
4
5  public class Spawn : MonoBehaviour {
6
7      // yaratık oluşması için gereken süre
8      public float interval = 3.0f;
9      float timeLeft = 0.0f;
10
11     // Update is called once per frame
12     void Update () {
13         // yeni yaratık oluşturmanın vakti gelmiş mi?
14         timeLeft -= Time.deltaTime;
15         if (timeLeft <= 0.0f) {
16             // Yapılacak: yaratık oluştur
17
18             // yaratık oluşması için gerekli süreyi başa sar
19             timeLeft = interval;
20         }
21     }
   }
```

Yaratık oluşturmak

Yaratık objesini depolamak için GameObject türünde yeni bir public değişken oluşturacağız ve daha sonra bu değişkene Inspector'dan ayıcığımızı sürükleyerek değer atayacağız. Ardından Unity'nin **Instantiate** metodunu (komut) kullanarak bu yaratıktan bir tane oluşturacağız. Bu metod parametre olarak üç değer alır: oluşturulacak GameObject (ayıcık prefabımız), oluşturulacak konum (spawn noktamız) ve oluşturulacak objenin eğimi (bizim durumumuzda Quaternion.identity; yani 0,0,0 rotasyonu).

İşte yazmamız gereken kod:

```
1  using UnityEngine;
   using System.Collections;
2
3  public class Spawn : MonoBehaviour {
4
5      // yaratık oluşması için gereken süre
6      public float interval = 3.0f;
7      float timeLeft = 0.0f;
8
9      // spawn edilecek objeyi depolayan değişken
10     public GameObject teddy = null;
```

```
    // Update is called once per frame
    void Update () {
        // yeni yaratık oluşturmanın vakti gelmiş mi?
        timeLeft -= Time.deltaTime;
        if (timeLeft <= 0.0f) {
            // yaratık oluştur
            GameObject g = (GameObject)Instantiate(teddy,
transform.position, Quaternion.identity);

            // yaratık oluşması için gerekli süreyi başa sar
            timeLeft = interval;
        }
    }
}
```

Not: Instantiate metodunun ilk parametresi bir Prefab olabileceği gibi sahnede yer alan herhangi bir GameObject de olabilir.

Yaratığa güzergah belirlemek

Eğer sahnedeki Spawn objemize Spawn.cs scriptimizi component olarak atarsak ve "teddy" değişkenine değer olarak TeddyPrefab'ını atıp oyunu çalıştırırsak spawn noktasında her üç saniyede bir ayıcık doğduğunu olduğunu görebiliriz.

Yine de hâlâ ayıcıklar ne yapacaklarını bilmiyorlar ve boş boş duruyorlar. Şimdi bu ayıcıklara gidecekleri bir hedef nokta vermeliyiz.

Öncelikle hedef noktayı depolamak için bir Transform değişken oluşturuyoruz. Böylece Inspector'dan hedef nokta için kolaylıkla değer atayabiliriz:

```
using UnityEngine;
using System.Collections;

public class Spawn : MonoBehaviour {

    // yaratık oluşması için gereken süre
    public float interval = 3.0f;
    float timeLeft = 0.0f;

    // spawn edilecek objeyi depolayan değişken
    public GameObject teddy = null;

    // tüm yaratıkların koşacağı hedef nokta
    public Transform destination = null;

    // Update is called once per frame
    void Update () {
        // yeni yaratık oluşturmanın vakti gelmiş mi?
        timeLeft -= Time.deltaTime;
        if (timeLeft <= 0.0f) {
            // yaratık oluştur
            GameObject g = (GameObject)Instantiate(teddy,
```

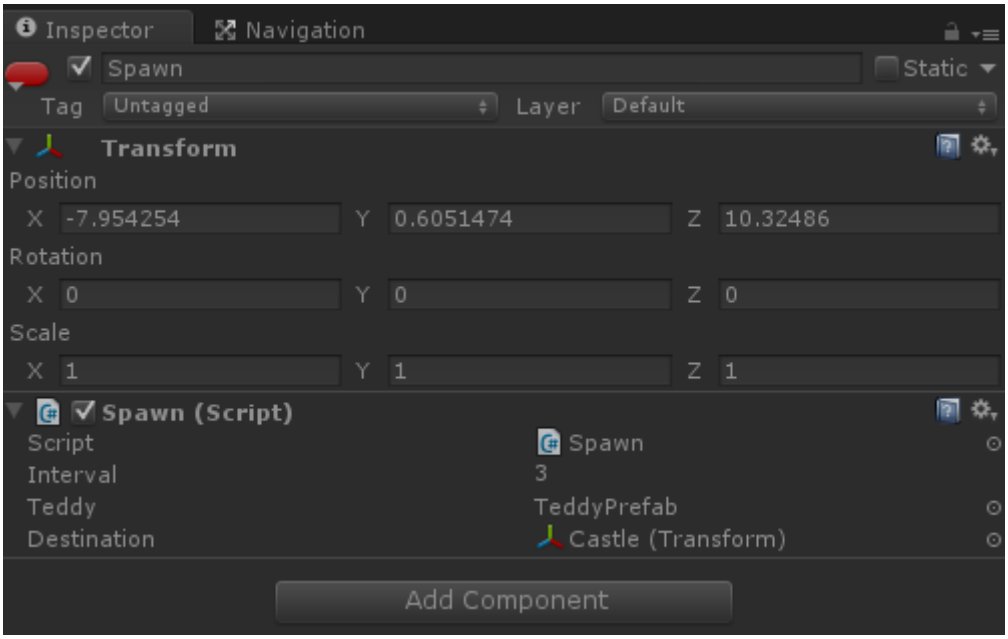
```
transform.position, Quaternion.identity);

            // yaratık oluşması için gerekli süreyi başa sar
            timeLeft = interval;
        }
    }
}
```

Şimdi yapmamız gereken son şey ayıcıklara gidecekleri noktayı söylemek. TeddyPrefab objesi **NavMeshAgent** component'ine sahiptir ve bu component pathfinding kullanarak ayıcığı otomatik olarak hedefine hareket ettirmeye yarar. Tek yapmamız gereken bu component'e bir hedef göstermek, gerisini Unity kendi halledecek.

İşte son haliyle kodumuz:

```


using UnityEngine;
using System.Collections;

public class Spawn : MonoBehaviour {

    // yaratık oluşması için gereken süre
    public float interval = 3.0f;
    float timeLeft = 0.0f;

    // spawn edilecek objeyi depolayan değişken
    public GameObject teddy = null;

    // tüm yaratıkların koşacağı hedef nokta
    public Transform destination = null;

    // Update is called once per frame
    void Update () {
        // yeni yaratık oluşturmanın vakti gelmiş mi?
        timeLeft -= Time.deltaTime;
        if (timeLeft <= 0.0f) {
            // yaratık oluştur
            GameObject g = (GameObject)Instantiate(teddy,
transform.position, Quaternion.identity);

            // NavMeshAgent component'ine ulaşarak ona bir hedef göster
            NavMeshAgent n = g.GetComponent<NavMeshAgent>();
            n.destination = destination.position;

            // yaratık oluşması için gerekli süreyi başa sar
            timeLeft = interval;
        }
    }
}


```

Artık scriptimiz bitti. Eğer scripte Inspector'dan gerekli tüm değişken değerlerini halen atamadıysanız o değerleri şekildeki gibi doldurun:



Player (Oyuncu) scripti

Oyuncunun kule satın almak için paraya ihtiyacı olacak. Bunu halletmek için Player.cs adında bir script oluşturacağız:

```
1
2 using UnityEngine;
3 using System.Collections;
4
5 public class Player : MonoBehaviour {
6     public static int gold = 3; // başlangıçta elimizdeki altın miktarı
7
8     void OnGUI() {
9         // altın miktarını ekrana çiz
10        GUI.Label(new Rect(0, 0, 400, 200), "Gold: " + gold);
11    }
}
```

Buradaki gold (altın) değişkeninin **static** olduğuna dikkat edin. "static" değişkenlerin faydası onlara herhangi bir scriptten ulaşmanın çok kolay olmasıdır, tıpkı şöyle:

```
1Player.gold = 42;
```

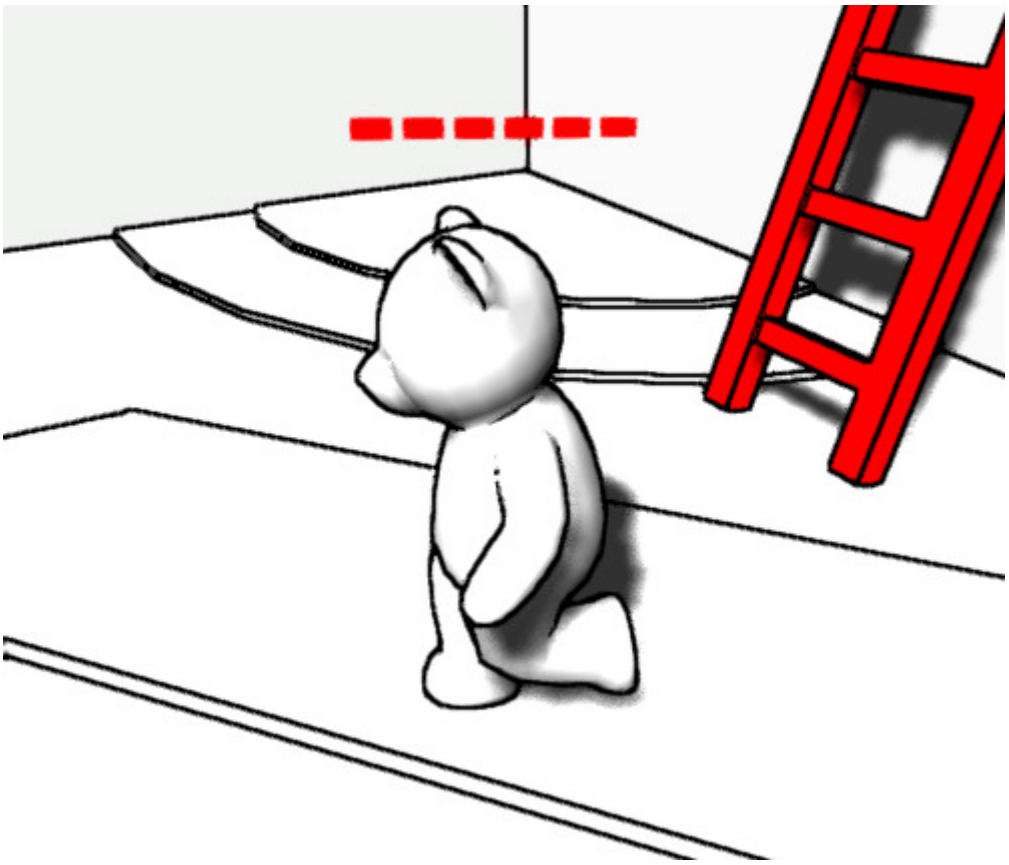
Bu script sadece altın miktarını depoladığı için scripti atayacağımız yeni bir obje oluşturmamıza gerek yok. Main Camera'ya atalım gitsin, değil mi? O halde scripti Main Camera'ya component olarak ekleyin.

Teddy (Ayıcık) Scripti

Ayıcıklarımız şu halleriyle de oldukça işlevli durumdalar ama bir script vasıtasıyla onlara bir sağlık miktarı atamamız ve ayıcık ölünce yapılacak şeyleri belirlememiz iyi olur.

Sağlık

Ayıcığın canını Unity'nin TextMesh component'i vasıtasıyla ayıcığın kafasının üstünde göstereceğiz:



Burada kullandığımız şey sadece "-" harflerinden oluşan basit bir TextMesh (3 boyutlu uzaydaki bir yazı). Eğer ayıcığın 1 canı varsa bu yazı "-" olurken 3 canı varsa "—" olacak. Başlangıçta ise 6 canı ("——") olacak.

Bize gerekli olan TextMesh componenti TeddyPrefab'ın içinde zaten mevcut, bu konuda endişelenmemize hiç gerek yok.

Not: TextMesh 3 boyutlu bir objedir ve Unity'nin OnGUI komutlarıyla uzaktan yakından alakası yoktur.

Ayıcığın canını basit bir int değeri olarak depolayacağız. Bunun için Teddy.cs adında yeni bir script oluşturun:

```
1public class Teddy : MonoBehaviour {
2    public int health = 6;
3}
```

Update fonksiyonunun içinde ise ayıcığın TextMesh component'inin ayıcığın canı kadar "-" işaretine sahip olmasını sağlayalım:

```
1
2 // Update is called once per frame
3 void Update () {
4     // TextMesh'in yazısını düzenle
5     TextMesh tm = GetComponentInChildren<TextMesh>();
6     tm.text = new string('-', health);
7
8     // TextMesh'in yazı rengini kırmızı yap
9     tm.renderer.material.color = Color.red;
10
11    // TextMesh'i hep kameraya doğru bakacak şekilde döndür
12    tm.transform.forward = Camera.main.transform.forward;
13}
14
```

Buradaki **tm.text = new string…** kısmı basit anlamda TextMesh'te göstereceğimiz ve "-" işaretlerinden oluşan String'imizi uygun uzunlukta oluşturmaya yarıyor.

Not: sağlığı göstermek için daha yüzlerce yol bulabiliriz ama muhtemelen bizim kullandığımız yol en kolayı.

Ölüm

Eğer bir kuleden çıkan kurşun ayıcığı öldürürse ayıcığı sahneden çıkarmalı ve oyuncunun altın miktarını artırmalıyız. Ölüm anında yapılacak bu şeyleri kendi oluşturduğumuz **onDeath** adında bir fonksiyona yazacağız:

```
1
2public void onDeath() {
3    // oyuncunun altınını 1 artır
4    Player.gold = Player.gold + 1;
5
6    // ayıcığı yoket
7    Destroy(gameObject);
8}
```

Gördüğünüz gibi onDeath fonksiyonumuz public türde. Yani dışarıdan herhangi bir script bu fonksiyona ulaşabilir. İşte Teddy.cs scriptimizin son hâli:

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class Teddy : MonoBehaviour {
5     public int health = 6;
6
7     // Update is called once per frame
8     void Update () {
9         // TextMesh'in yazısını düzenle
10        TextMesh tm = GetComponentInChildren<TextMesh>();
11        tm.text = new string('-', health);
12
13        // TextMesh'in yazı rengini kırmızı yap
```

```
        tm.renderer.material.color = Color.red;

        // TextMesh'i hep kameraya doğru bakacak şekilde döndür
        tm.transform.forward = Camera.main.transform.forward;
    }

    public void onDeath() {
        // oyuncunun altınını 1 artır
        Player.gold = Player.gold + 1;

        // ayıcığı yoket
        Destroy(gameObject);
    }
}
```

Yapmanız gereken son şey bu scripti Project panelinden TeddyPrefab'ımıza component olarak atamak.

Castle (Kale) Scripti

Bu script sadece iki şey yapacak:

- Korumakla görevli olduğumuz kallemizin canını **OnGUI** ile gösterecek
- Bir ayıcığın kaleye ulaşıp ulaşmadığını **Update** ile kontrol edecek

OnGUI Fonksiyonu

Castle.cs adında yeni bir script oluşturup açın ve içinde yeni bir OnGUI fonksiyonu oluşturun:

```
using UnityEngine;
using System.Collections;

public class Castle : MonoBehaviour {

    void OnGUI() {

    }
}
```

Şimdi kalenin canını depolamak için bir değişken oluşturalım ve bunu OnGUI'de gösterelim. Bir ayıcık kaleye ulaşmayı başardığı zaman kalenin sağlığı 1 azalacak. Eğer can 0′a düşürse oyun kaybedilmiş olur:

```
using UnityEngine;
using System.Collections;

public class Castle : MonoBehaviour {
    public static int health = 100;

```

```
    void OnGUI() {
        // kalenin canını ekranda göster
        GUI.Label(new Rect(0, 40, 400, 200), "Castle Health: " + health);
    }
}
```

Burada daha önce görmediğimiz hiçbir şey yok.

Update Fonksiyonu

Bu fonksiyonda Unity'nin **FindObjectsOfType** metodunu kullanarak sahnedeki tüm ayıcıkları bulacak ve eğer bir ayıcığın kaleye çok yakın olduğunu farkedersek kalenin canını 1 azaltacağız. İşte böyle:

```
void Update() {
    // sahnedeki tüm ayıcıkları bul
    Teddy[] teddys = (Teddy[])FindObjectsOfType(typeof(Teddy));
    if (teddys != null) {
        // kaleye yakın olan ayıcıkları bul
        for (int i = 0; i < teddys.Length; ++i) {
            float range = Mathf.Max(collider.bounds.size.x,
collider.bounds.size.z);
            if (Vector3.Distance(transform.position,
teddys[i].transform.position) <= range) {
                // kale canını 1 azalt
                health = health - 1;

                // ayıcığı yoket
                Destroy(teddys[i].gameObject);
            }
        }
    }
}
```

Not: kalenin veya oyun sahasının boyutlarını bilmediğimizden ayıcıkların kaleye olan mesafesini sabit bir değerle kıyaslamıyoruz ama kalenin kendisinin ne kadar büyük olduğuyla kıyaslıyoruz (Mathf.Max kısmı)

Castle.cs scriptini Hierarchy panelinden Castle objesine atayarak bu kısmı da bitiriyoruz.

Bullet (Kurşun) Scripti

Bu script de iki şey yapacak:

- Update fonksiyonunda bir ayıcığa doğru yol alacak
- Eğer ayıcığa ulaşırsa canını azaltacak

Hedefe doğru yol almak

Bu işi yapmak için Unity'nin **Vector3.MoveTowards** komutunu kullanacağız:

```
1
2 using UnityEngine;
3 using System.Collections;
4
5 public class Bullet : MonoBehaviour {
6
7     // kurşunun hareket hızı
8     public float speed = 10.0f;
9
10    // kurşunun hedefi (daha sonra kule tarafından belirlenecek)
11    Transform destination;
12
13    // Update is called once per frame
14    void Update () {
15        // hedefe doğru yol al
16        float stepSize = Time.deltaTime * speed;
17        transform.position = Vector3.MoveTowards(transform.position,
18destination.position, stepSize);
19
20        // hedefe vardık mı?
21        if (transform.position.Equals(destination.position)) {
22            // Yapılacaklar...
23        }
24    }
25
26    public void setDestination(Transform v) {
27        destination = v;
28    }
29}
30
31
```

Not: setDestination metodunu daha sonra Tower (kule) tarafından çağıracağız.

Bu scripte biraz iyileştirme yapmamız gerekli. Zira kurşun ayıcığa doğru yol alırken ayıcık başka bir kurşun tarafından öldürülürse **destination** değişkenimizin değeri **null** olacak ve bu durumda null bir değişkenle işlem yapmaya çalıştığımız için hata alacağız. Bunu önlemek için Update fonksiyonunda ufak bir değişikliğe gidiyoruz:

```
1
2void Update () {
3    // eğer hedef obje yok olmuşsa kurşunu da yoket
4    if (destination == null) {
5        Destroy(gameObject);
6        return;
7    }
8
9    // kodun devamı...
10}
11
```

Ayıcığa ulaşınca canını azaltmak

Eğer kurşun ayıcığa ulaşırsa canını azaltıp kendini yoketmeli. Bunu yapmak için scriptimizin "*Yapılacaklar...*" kısmını şekildeki gibi güncelleyin:

```
// ayıcığın canını 1 azalt
Teddy t = destination.GetComponent<Teddy>();
t.health = t.health - 1;

// eğer ayıcığın canı kalmadıysa onu öldür (onDeath metodunu çağır)
if (t.health <= 0)
    t.onDeath();

// kurşunu yoket
Destroy(gameObject);
```

Hepsini bir araya toplarsak işte scriptimizin son hâli:

```
using UnityEngine;
using System.Collections;

public class Bullet : MonoBehaviour {

    // kurşunun hareket hızı
    public float speed = 10.0f;

    // kurşunun hedefi (daha sonra kule tarafından belirlenecek)
    Transform destination;

    // Update is called once per frame
    void Update () {
        // eğer hedef obje yok olmuşsa kurşunu da yoket
        if (destination == null) {
            Destroy(gameObject);
            return;
        }

        // hedefe doğru yol al
        float stepSize = Time.deltaTime * speed;
        transform.position = Vector3.MoveTowards(transform.position,
destination.position, stepSize);

        // hedefe vardık mı?
        if (transform.position.Equals(destination.position)) {
            // ayıcığın canını 1 azalt
            Teddy t = destination.GetComponent<Teddy>();
            t.health = t.health - 1;

            // eğer ayıcığın canı kalmadıysa onu öldür
            if (t.health <= 0)
                t.onDeath();

            // kurşunu yoket
            Destroy(gameObject);
        }
    }

    public void setDestination(Transform v) {
        destination = v;
    }
}
```

Şimdi scripti Project panelinden kurşun (Bullet) prefabına atayın.

Tower (Kule) Scripti

İşte yapılacaklar:

- Her birkaç saniyede bir yeni bir hedef (Target) bul
- Hedefe yeni bir kurşun yolla
- Kuleyi inşa etmek için gerekli altın miktarını depola (daha sonra TowerSlots scriptinde kullanacağız)
- Kuleyi kendi etrafında sürekli döndür, böylece daha yakışıklı dursun

Hedef bulmak

Hedef bulurken Castle scriptimizde tüm ayıcıkları bulmak için kullandığımız yöntemle benzer bir işlem kullanacağız. Ama bu sefer tüm ayıcıkları değil, sadece kuleye en yakın olan ayıcığı bulacağız. Bunun için Tower.cs scripti oluşturup şekildeki gibi düzenleyin:

```
Teddy findClosestTarget() {
    Teddy closest = null;
    Vector3 pos = transform.position;

    // tüm ayıcıkların içinden en yakın olanını bul
    Teddy[] teddys = (Teddy[])FindObjectsOfType(typeof(Teddy));
    if (teddys != null) {
        if (teddys.Length > 0) {
            closest = teddys[0];
            for (int i = 1; i < teddys.Length; ++i) {
                float cur = Vector3.Distance(pos,
teddys[i].transform.position);
                float old = Vector3.Distance(pos,
closest.transform.position);

                if (cur < old) {
                    closest = teddys[i];
                }
            }
        }
    }

    return closest;
}

```

Burası çok da karmaşık değil. Bir listeden en yakın objeyi bulmak için kullanılan genel bir algoritmadan faydalandık sadece.

Not: eğer sahnede hiç ayıcık yoksa null döndürülür.

Kulenin ayılara her iki saniyede bir saldırmasını istediğimizi varsayalım. Bunu yaparken Spawn scriptinde ayıcıkları doğurmak için kullandığımız yöntemi kullanacağız. Yani böyle:

```
using UnityEngine;
using System.Collections;

public class Tower : MonoBehaviour {
    // saldırı yapma aralığı
    public float interval = 2.0f;
    float timeLeft = 0.0f;

    // saldırı menzili
    public float range = 10.0f;

    Teddy findClosestTarget() {
        Teddy closest = null;
        Vector3 pos = transform.position;

        // tüm ayıcıkların içinden en yakın olanını bul
        Teddy[] teddys = (Teddy[])FindObjectsOfType(typeof(Teddy));
        if (teddys != null) {
            if (teddys.Length > 0) {
                closest = teddys[0];
                for (int i = 1; i < teddys.Length; ++i) {
                    float cur = Vector3.Distance(pos,
teddys[i].transform.position);
                    float old = Vector3.Distance(pos,
closest.transform.position);

                    if (cur < old) {
                        closest = teddys[i];
                    }
                }
            }
        }

        return closest;
    }

    void Update() {
        // yeni kurşun ateşlemenin vakti geldi mi?
        timeLeft -= Time.deltaTime;
        if (timeLeft <= 0.0f) {
            // en yakın ayıcığı bul (eğer varsa)
            Teddy target = findClosestTarget();
            if (target != null) {
                // ayıcık yeterince yakın mı?
                if (Vector3.Distance(transform.position,
target.transform.position) <= range) {
                    // Yapılacak: kurşun ateşle

                    // zamanlayıcıyı resetle
                    timeLeft = interval;
                }
            }
        }
```

```
    }
}
```

Not: kuleye bir menzil de ekledik. Böylece menzil dışındaki ayıcıklara saldırmayacak. Eğer kule hiçbir zaman ateş etmezse menzili artırmayı deneyin.

Hedefe ateş etmek

Kulenin bulunduğu yerde yeni bir kurşun oluşturmak için Unity'nin **Instantiate** metodunu kullanacağız. Ardından kurşuna hedef olarak en yakın ayıcığı vereceğiz:

```
// kurşun oluştur
GameObject g = (GameObject)Instantiate(bulletPrefab.gameObject,
transform.position, Quaternion.identity);

// kurşundaki Bullet scriptine eriş
Bullet b = g.GetComponent<Bullet>();

// kurşuna hedef ata
b.setDestination(target.transform);
```

Kule oluşturma ücreti

Kuleleri oluşturmaktan TowerSlot scripti sorumlu olacak. Ama kule oluşturmak için bir ücret ödememiz lazım. Bu ücreti de Tower scriptinde bir public int değişken vasıtasıyla depoluyoruz:

```
// kuleyi dikmek için gerekli altın miktarı
public int buildPrice = 1;
```

Mevcut durumda bir kule dikmek için 1 altın harcamamız gerekli.

Kuleyi döndürmek

Kulenin tüm oyun mekaniğini kodladık ama bununla kalmayıp ona güzel bir animasyon vererek ortamı renklendirelim. Teknik anlamda verdiğimiz şey gerçek bir animasyon bile değil aslında, sadece kuleyi kendi etrafında döndürüyoruz:

```
// kuleyi kendi etrafında döndür
Vector3 rot = transform.eulerAngles;
transform.rotation = Quaternion.Euler(rot.x, rot.y + Time.deltaTime *
rotationSpeed, rot.z);
```

Tower Scriptinin Son Hâli

Yazdığımız herşeyi bir araya koyarsak işte Tower scriptimizin son hâli:

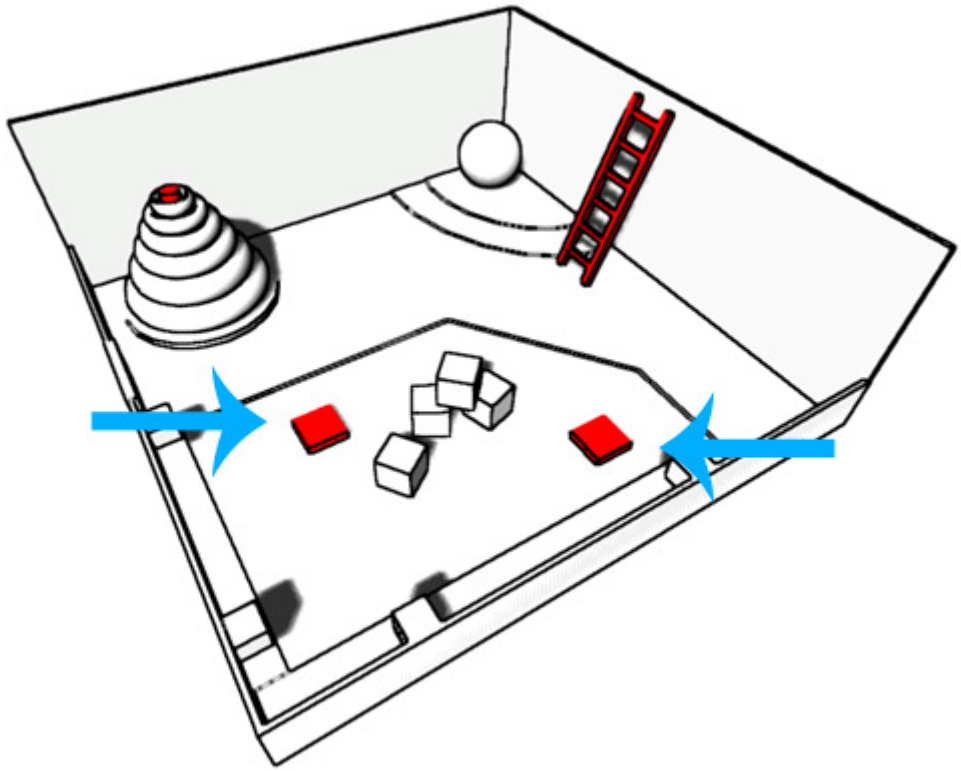
```
using UnityEngine;
using System.Collections;

public class Tower : MonoBehaviour {
    // kurşun objesi
    public Bullet bulletPrefab = null;

    // saldırı yapma aralığı
    public float interval = 2.0f;
    float timeLeft = 0.0f;

    // saldırı menzili
    public float range = 10.0f;

    // kuleyi dikmek için gerekli altın miktarı
    public int buildPrice = 1;

    // kulenin kendi etrafında dönme hızı
    public float rotationSpeed = 2.0f;

    Teddy findClosestTarget() {
        Teddy closest = null;
        Vector3 pos = transform.position;

        // tüm ayıcıkların içinden en yakın olanını bul
        Teddy[] teddys = (Teddy[])FindObjectsOfType(typeof(Teddy));
        if (teddys != null) {
            if (teddys.Length > 0) {
                closest = teddys[0];
                for (int i = 1; i < teddys.Length; ++i) {
                    float cur = Vector3.Distance(pos,
teddys[i].transform.position);
                    float old = Vector3.Distance(pos,
closest.transform.position);

                    if (cur < old) {
                        closest = teddys[i];
                    }
                }
            }
        }

        return closest;
    }

    void Update() {
        // yeni kurşun ateşlemenin vakti geldi mi?
        timeLeft -= Time.deltaTime;
        if (timeLeft <= 0.0f) {
            // en yakın ayıcığı bul (eğer varsa)
            Teddy target = findClosestTarget();
            if (target != null) {
                // ayıcık yeterince yakın mı?
                if (Vector3.Distance(transform.position,
target.transform.position) <= range) {
                    // kurşun oluştur
                    GameObject g =
(GameObject)Instantiate(bulletPrefab.gameObject, transform.position,
Quaternion.identity);
```

```
51
52                      // kurşundaki Bullet scriptine eriş
53                      Bullet b = g.GetComponent<Bullet>();
54
55                      // kurşuna hedef ata
56                      b.setDestination(target.transform);
57
58                      // zamanlayıcıyı resetle
59                      timeLeft = interval;
60                  }
61              }
62          }
63
64          // kuleyi kendi etrafında döndür
65          Vector3 rot = transform.eulerAngles;
66          transform.rotation = Quaternion.Euler(rot.x, rot.y +
Time.deltaTime * rotationSpeed, rot.z);
67      }
68  }
69
70
71
72
```

Şimdi bu scripti Project panelinden Tower prefab'ına ekleyin. Ardından Tower prefab'ına Inspector panelinden bulletPrefab değişkeni değeri olarak Bullet (kurşun) prefab'ını atayın.

TowerSlot (KuleDikmeNoktası) Scripti

Önceki aşamada kule dikme noktalarını zaten oluşturmuştuk:



Şimdi bu noktalara kule dikebilmeyi sağlamak için TowerSlots.cs adında yeni bir script oluşturalım. Eğer oyuncu bir kule dikme noktasına mouse ile tıklarsa ona “Build Tower (1 Gold)” şeklinde bir menü ["Kule Dik (1 Altın)"] göstereceğiz. Bir objeye 3 boyutlu uzayda mouse ile tıklanıp tıklanmadığını test etmek için Unity’nin **OnMouseDown** (MouseİleTıklandı) ve **OnMouseUp** (MouseButonuKaldırıldı) fonksiyonlarından faydalanabiliriz.

Fareyle bir kule dikme noktasına tıkladığımızda menümüz açılacak. Ardından ise iki şey olabilir:

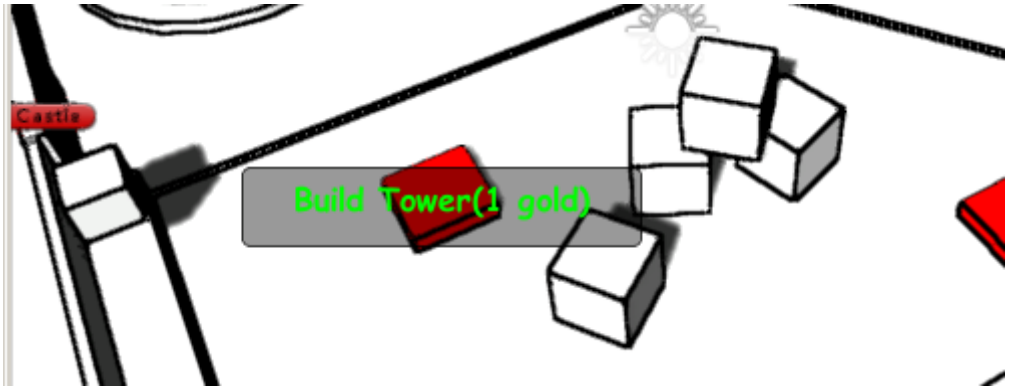
- Fare butonunu menünün üzerindeyken bırakmak: o zaman bir kule inşa et ve menüyü kapat
- Fare butonunu menünün dışındayken bırakmak: o zaman sadece menüyü kapat

Yazacağımız script çoğunlukla OnGUI fonksiyonunda yer alacak ve mouse tıklamalarıyla ilgili komutlar içerecek. Ayrıca oyuncunun elindeki altın miktarına göre menünün rengi de değişecek:

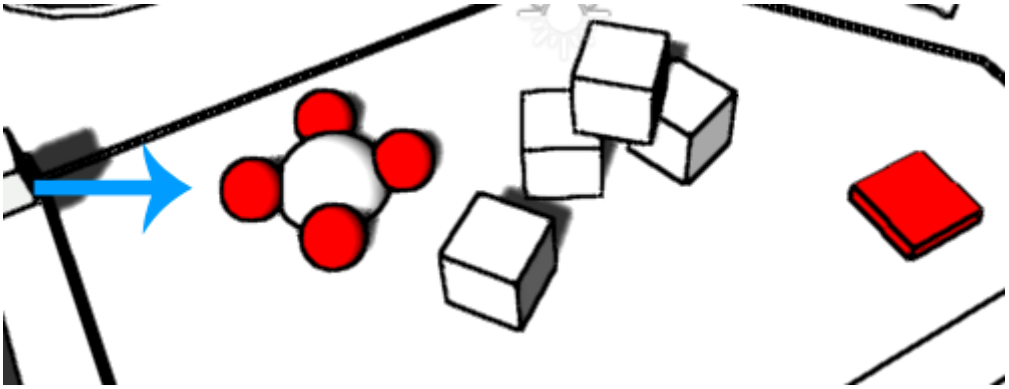
```
using UnityEngine;
using System.Collections;

public class TowerSlot : MonoBehaviour {
    bool gui = false;

    // Tower (kule) prefabı
    public Tower towerPrefab = null;

    void OnGUI() {
        if (gui) {
```

```
            // bu kule dikme noktasının ekrandaki yerini bul
            Vector3 v =
Camera.main.WorldToScreenPoint(transform.position);

            // bu ekran koordinatını gui koordinatına çevir
            v = new Vector2(v.x, Screen.height - v.y);

            // kule dikmek için menü oluştur
            int width = 200;
            int height = 40;
            Rect r = new Rect(v.x - width / 2, v.y - height / 2, width,
height);
            GUI.contentColor = (Player.gold >= towerPrefab.buildPrice ?
Color.green : Color.red);
            GUI.Box(r, "Build " + towerPrefab.name + "(" +
towerPrefab.buildPrice + " gold)");

            // mouse butonu artık tıklanmıyorsa ve mouse menünün
üzerindeyse kule oluştur
            if (Event.current.type == EventType.MouseUp &&
                r.Contains(Event.current.mousePosition) &&
                Player.gold >= towerPrefab.buildPrice) {
                // oyuncunun elindeki altını azalt
                Player.gold -= towerPrefab.buildPrice;

                // kuleyi oluştur
                Instantiate(towerPrefab, transform.position,
Quaternion.identity);

                // kule dikme noktasını kapat (deaktif et)
                gameObject.SetActive(false);
            }
        }
    }

    public void OnMouseDown() {
        gui = true;
    }

    public void OnMouseUp() {
        gui = false;
    }
}
```

Şimdi TowerSlots.cs scriptini sahnedeki tüm kule dikme noktalarına atayın ve **towerPrefab** değişkenine değer olarak TowerPrefab'ı atayın. Şimdi Play'e basarak oyunu test edin. Kule dikme noktasına tıklayınca menümüz şekildeki gibi belirecek:



Ve eğer mouse butonunu menü üzerindeyken bırakırsak kule dikilecek:



Kule dikme noktası yok oldu ve yerine kulemiz geldi. Muhteşem!

Oyunun son hâli

Artık oyun mekaniklerini kodladığımıza göre oyunun son hâline bakmak için "scene_menu" bölümüne geçiş yapın ve Play butonuna tıklayın. İşte benim oyunumdan bir video:

http://www.youtube.com/watch?v=CQoVpUZ1weM

Özet

Beraber tek bölümlük, hızlı oynanışlı ve orijinal bir tarza sahip olan ufak bir Tower Defense oyunu yaptık. Şimdi oyunu geliştirmek için sizin sıranız. İşte bazı ipuçları:

- Sesler ekleyin
- Yeni kuleler ekleyin
- Yeni bölümler ekleyin
- Yeni düşmanlar ekleyin
- Oyunu daha da zorlaştırın
- Oyuna kazanma/kaybetme ekranı ekleyin
- Kule geliştirme (upgrade) sistemi ekleyin

İşte bir tutorialin daha sonu. Her zaman olduğu gibi, başka tutoriallerde görüşmek üzere!