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

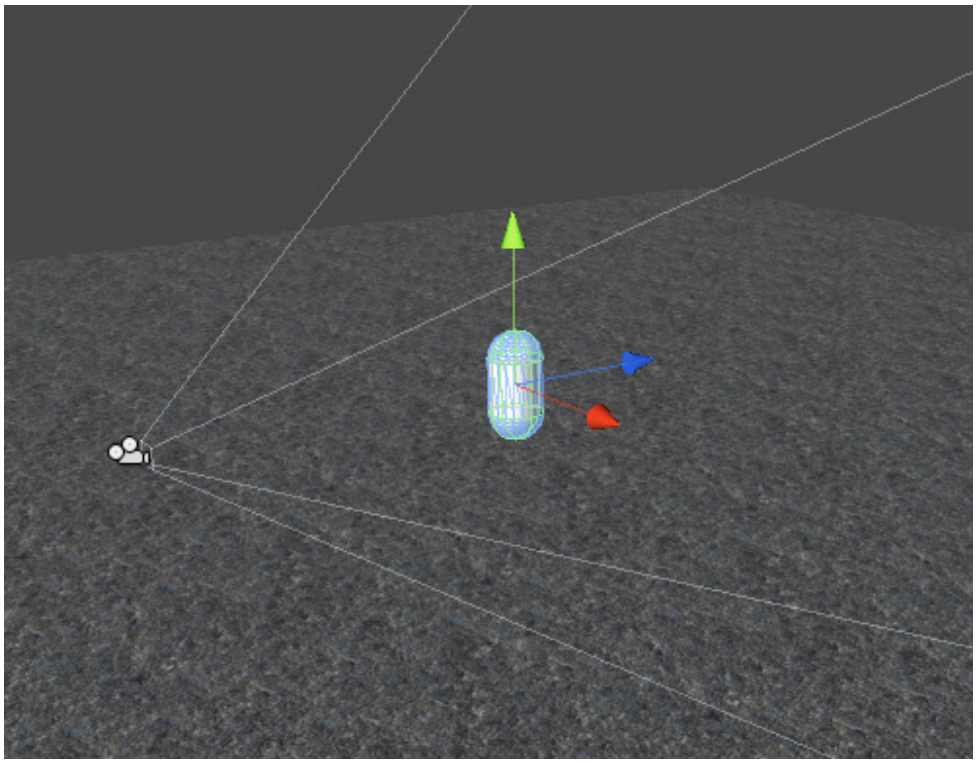
Karakteri hareket ettirebiliyoruz. Şimdi karakter ile etrafa bakabilmeyi (kamerayı döndürmeyi) yapalım.

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.

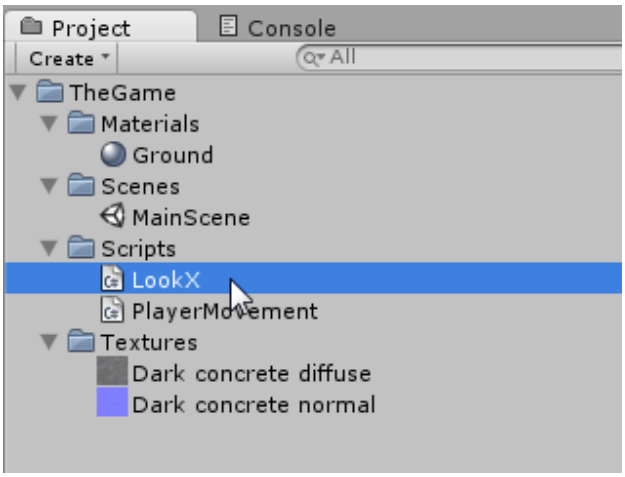
Şimdi kamera objesini karakterin arkasına doğru taşıyalım: kamerayı (0,0,-7) koordinatlarına taşıyın ve eğimini (0,0,0) yapın.



(Projeyi sık sık kaydetmeyi unutmayın.)

Sağa-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 ekseni) 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 kaldırdık.

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.

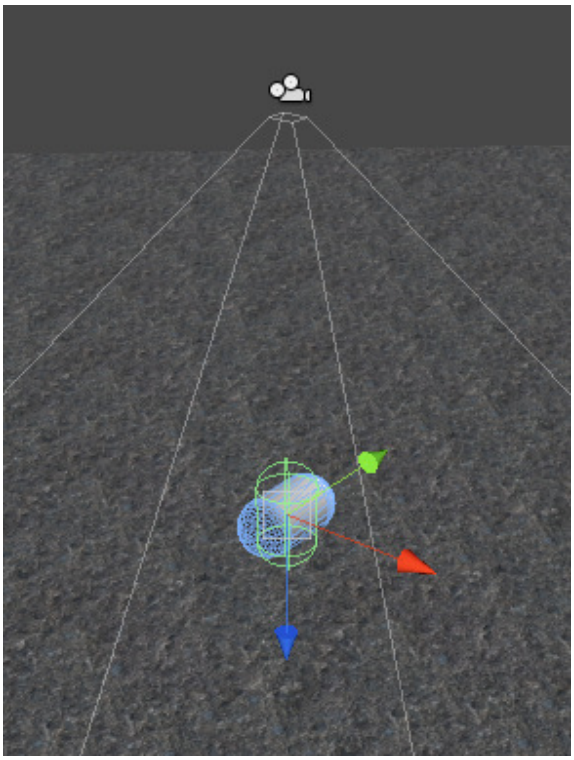
Kamerayı döndürürken Time.deltaTime ile çarpmadığımıza dikkat edin. Oyun ne kadar yavaşlasa da fare yavaşlamayacaktır. Belki imlecin konumunun güncellenmesi yavaşlayacaktır ama bizim işimiz imlecin konumuyla alakalı değil.

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 döndürürken aşağı oynattıkça negatif değer döndürür. Ama biz kamerayı yukarı oynatmak için fareyi aşağı hareket ettirmeliyiz. Döndürülen değeri -1 ile çarptığımız için artık istediğimiz şey gerçekleşiyor.



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

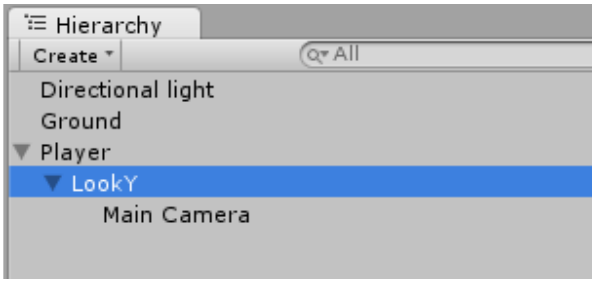
Objenin x eksenindeki eğimini değiştirdiğimiz için obje devriliyor. Scripti hangi objeye verirsek o obje fareyi yukarı-aşağı oynattıkça devrilecek-düzelecek. Ama biz sadece kameranın yukarı-aşağı oynamasını istiyoruz.

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 verdiğimiz 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 Yönde 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 çağırılmışsa 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” demeye 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. Şimdilik ekranın ortasında bir resim çizdireceğiz.

“PlayerGui” adında yeni bir C# scripti oluşturun:

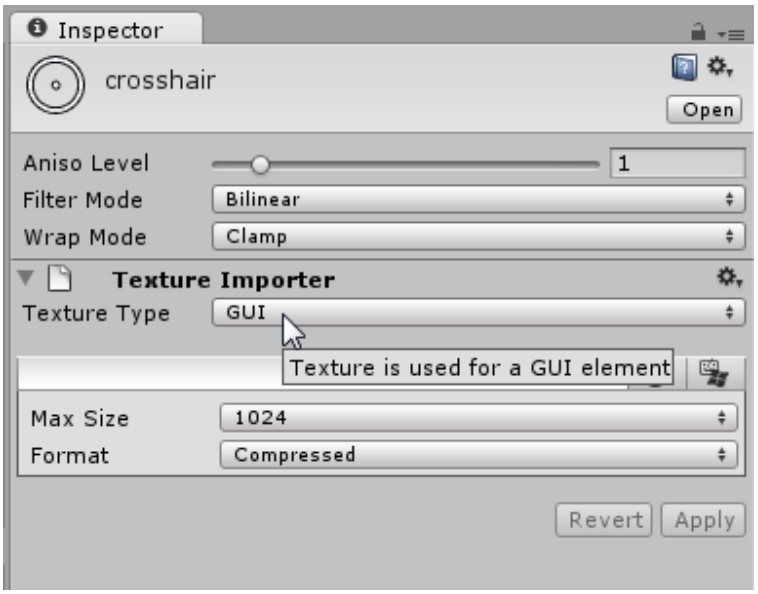
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _crosshair;
08
09     void OnGUI()
10     {
11         GUI.DrawTexture(new Rect(0, 0, 30, 30), _crosshair);
12     }
13 }
```

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'inin 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:

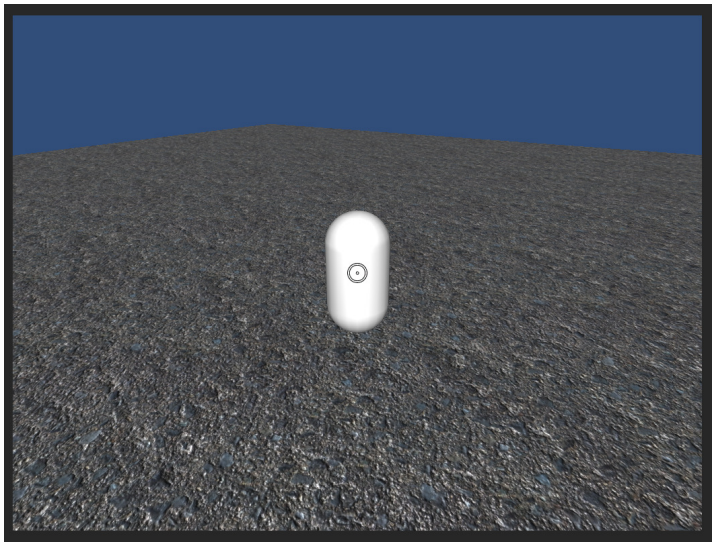
```
11              float  x = (Screen.width - _crosshair.width) / 2;
12              GUI.DrawTexture(new  Rect(x, 0, _crosshair.width, _crosshair.height), _crosshair);
```

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

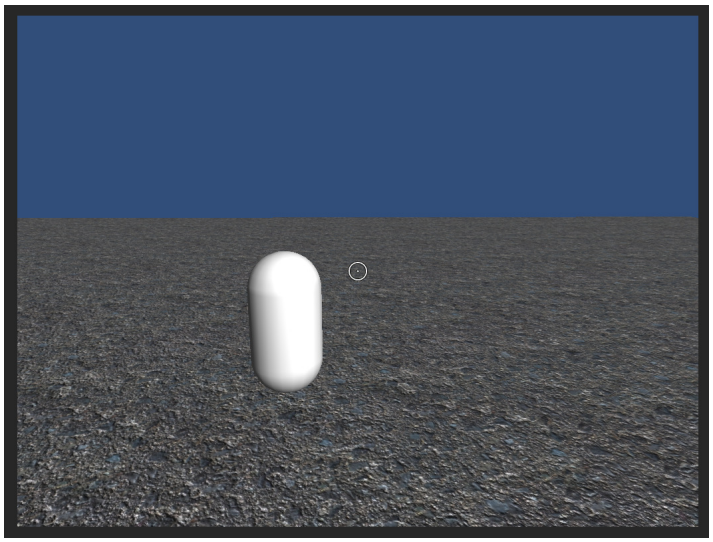
```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  PlayerGui : MonoBehaviour
05 {
06      [SerializeField]
07      Texture2D _crosshair;
08
09      void  OnGUI()
10      {
11          float  x = (Screen.width - _crosshair.width) / 2;
12          float  y = (Screen.height - _crosshair.height)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