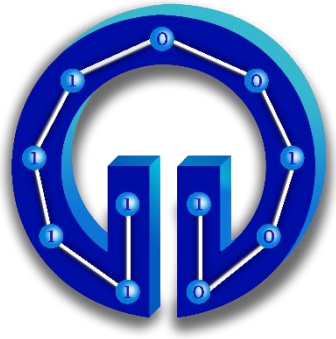
KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ



Unity ile Işın İzleme Yönteminin Etkileşimli Görsel Eğitimi

BİTİRME PROJESİ

M.Emin Altınışık
Ümit Şahan
Fahrettin Efe Okudan

2021-2022 GÜZ DÖNEMİ

KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Unity ile Işın İzleme Yönteminin Etkileşimli Görsel Eğitimi

BİTİRME PROJESİ

M.Emin Altınışık
Ümit Şahan
Fahrettin Efe Okudan

2021-2022 GÜZ DÖNEMİ

IEEE Etik Kuralları
IEEE Code of Ethics

Mesleğime karşı şahsi sorumluluğumu kabul ederek, hizmet ettiğim toplumlara ve üyelerine en yüksek etik ve mesleki davranışta bulunmaya söz verdiğimi ve aşağıdaki etik kurallarını kabul ettiğimi ifade ederim:

1. Kamu güvenliği, sağlığı ve refahı ile uyumlu kararlar vermenin sorumluluğunu kabul etmek ve kamu veya çevreyi tehdit edebilecek faktörleri derhal açıklamak;
2. Mümkün olabilecek çıkar çatışması, ister gerçekten var olması isterse sadece algı olması, durumlarından kaçınmak. Çıkar çatışması olması durumunda, etkilenen taraflara durumu bildirmek;
3. Mevcut verilere dayalı tahminlerde ve fikir beyan etmelerde gerçekçi ve dürüst olmak;
4. Her türlü rüşveti reddetmek;
5. Mütenasip uygulamalarını ve muhtemel sonuçlarını gözeterek teknoloji anlayışını geliştirmek;
6. Teknik yeterliliklerimizi sürdürmek ve geliştirmek, yeterli eğitim veya tecrübe olması veya işin zorluk sınırları ifade edilmesi durumunda ancak başkaları için teknolojik sorumlulukları üstlenmek;
7. Teknik bir çalışma hakkında yansız bir eleştiri için uğraşmak, eleştiriyi kabul etmek ve eleştiriyi yapmak; hatları kabul etmek ve düzeltmek; diğer katkı sunanların emeklerini ifade etmek;
8. Bütün kişilere adilane davranmak; ırk, din, cinsiyet, yaş, milliyet, cinsi tercih, cinsiyet kimliği, veya cinsiyet ifadesi üzerinden ayırımcılık yapma durumuna girişmemek;
9. Yanlış veya kötü amaçlı eylemler sonucu kimsenin yaralanması, mülklerinin zarar görmesi, itibarlarının veya istihdamlarının zedelenmesi durumlarının oluşmasından kaçınmak;
10. Meslektaşlara ve yardımcı personele mesleki gelişimlerinde yardımcı olmak ve onları desteklemek.

IEEE Yönetim Kurulu tarafından Ağustos 1990'da onaylanmıştır.

ÖNSÖZ

"Unity ile ışın izleme yönteminin görsel eğitimi" adlı bu çalışma Karadeniz Teknik Üniversitesi Bilgisayar Mühendisliği Anabilim Dalında Bitirme Projesi olarak hazırlanmıştır. Bu projenin amacı "Işın İzleme" yönteminin ana mantığını, uygulamayı/oyunu açan herkese öğretmeyi hedeflemektir ve araç olarak Unity oyun motorunu kullandık.

Bu projenin yapımında bizlere yardımcı olan gerekli her türlü desteği sağlayan Öğr.Gör. Ömer Çakır hocamıza ve bizleri yalnız bırakmayan ailelerimize ve arkadaşlarımıza teşekkürlerimizi sunarız.

M.Emin Altınışık
Ümit Şahan
Fahrettin Efe Okudan
Trabzon 2022

İÇİNDEKİLER

ÖZET

Geçmişten günümüze oyun yapımcıları veya meraklı insanlar, oyun dünyasındaki görüntüyü, oyunu oynayan kişilere daha gerçekçi nasıl gösterebiliriz diye düşünmüşlerdir. Bu düşünceler ışığında çeşitli algoritmalar ile oyun motorları kullanılarak çeşitli görsel geliştirmeler yapılmıştır.

Bunlardan birisi de Işın izleme yöntemidir. Işın izleme kullanılarak ışın-cisim kesişimlerindeki görüntülerin her birisinin(gölge, yansıma, kırılma vb.) cisimler üzerindeki etkisi bile gözlemlenebilir hale gelmiştir.

1. GENEL BİLGİLER

1.1. Giriş

Ray Tracing yöntemi ile ışığın düştüğü yerin, oluşturduğu gölgelerin, yansımaların ve kırılmaların çeşitli algoritmalar ve komponentler ile hesaplanıp desteklenmesi ile birlikte oyun dünyası ve ekran kartı dünyasında ciddi değişimlere neden olmuştur. Bunlar hali hazırda oyun motorlarının birçoğunda yer almaktadır.

Oyun motorları da bu dünyaya ayak uydurmak adına bizler için oldukça güvenilir bir üretim aracı olmuştur. Tabii ki oyun motorları bir yere kadar, çeşitli kodlamalar ve modellemelerin nasıl şekilde kullanılacağı bizim yaratıcılığımıza kalmıştır.

1.2.RAY TRACING NEDIR VE NASIL ÇALIŞIR?

Türkçesi ışın izleme olan bu yöntemde 3 adet birbirleriyle etkileşim halinde olan fiziksel nesneler kullanılmaktadır. Bunlar bakış noktası, görüntü düzlemi ve görüntü düzlemi ile 3d cisimlerin kesişim testleridir.
Gözlemcinin bakış doğrultusu ve konumu hesaplanır sonrasında görüş alanı diye tabir edilen sanal bir görüntü düzlemi oluşturulur. Bu görüntü düzleminden geçen hayali ışınlar ile 3d cisimler kesiştirilir. Bu kesiştirme sonucunda 1 piksel değeri elde edilir. Görüntü düzlemimiz ve cismimizin piksel boyutlarına bağlı olarak bütün piksel değerleri hesaplandıktan sonrasında en yakın cismin piksel değerleri (renk kodu vb.) çeşitli algoritmalara sahip boyama teknikleri ile ekrana basılır [1]. Bu işlemlerin hepsi tek bir piksel değeri için yapılır. Yapılması gereken, kalan piksel değerlerinin döngüler aracılığı ile tek tek hesaplanıp ekrana basılmasıdır.

Tabii ki cisimlerin fiziksel özellikleri, kaplamaları vs. farklılık göstermektedir. Bunlar yansıtıcı, kırıcı gibi bilinen ve çok kullanılan özellikler olmaktadır. Örneğin baktığımız cisim yansıtıcı bir malzeme ile kaplanmış olsun. Yukarıda bahsedilen [1] boyama teknikleri ve kesişim testleri kullanıldığı zaman şu şekilde bir senaryo oluşur;

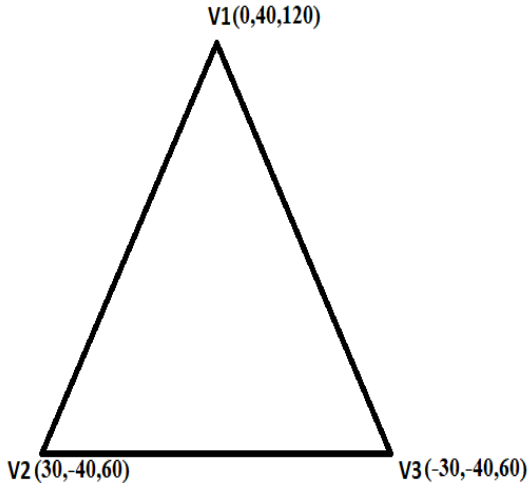
Görüntü düzleminden geçen ve yansıtıcı cisimle kesişen bir hayali ışınımız olsun. Bu ışın yansıtıcı cismin üzerine düşen, o cisim üzerine yansıyan diğer nesnelerin piksel değerlerini de alır ve buna göre ekrana basılır. Kırıcı bir nesne ile kaplanan yüzeyde de bu sefer kırılan görüntünün piksel değerleri ekrana basılır.

Sürekli ışın denen bir şeyden bahsediyoruz. Biraz detayına girelim sonrasında ray-tracing konusuna derinlemesine gireriz.
Işın vektörel bir madde olduğu için Ray Tracing'de vektörel çarpım, skaler çarpım normalizasyon gibi metotlardan yararlanacağız.

1)Normalizasyon:
Bir vektörün boyunu ne kadar büyüklükte olursa olsun 1 birim yapma işlemine denir. Örneğin V=(0,3,4) vektörümüz olsun. Vektörümüzün boyu ise |V| =sqrt(3^2+4^2+0^2) =5 bulunur. V/|V| yaparsak normalizasyon vektörümüzü hesaplamış oluruz. (0/5,3/5,4/5) yaparsak sonuç (0,0.6,0.8) çıkar bu vektörün boyunu hesapladığımızda ise sqrt((0.6)^2+(0.8)^2+0) =1 çıkar. Bu şekilde boyu 1 birim olmayan vektörleri 1 birim seviyesine indirebiliyoruz.

2)Vektörel Çarpım:
Herhangi iki tane V1 ve V2 vektörü olsun. Bu vektörlerin boyları ile arasındaki açının sinüsünün çarpımı ile V1 ve V2'nin bulunduğu düzleme dik bir vektör elde edilir. Bu vektöre Normal adı verilir. Üçgen normali hesabı küre normali hesabı gibi birçok nesnenin farklı şekillerde normal hesapları vardır, aynı değildir. 2d cisim için tek bir normal hesaplanırken, küre gibi bir cisim için her bir pikselin normali ayrı ayrı hesaplanmaktadır. Üçgenin normali hesaplanırken 1 tane normal hesabı yeterlidir. Normal hesabı için üçgenin köşe noktalarının sırası önemlidir. Bu sıraya göre üçgenin normali köşe noktalarının saat yönündeki yüzeyine dik şekilde olur.

Herhangi 3 adet vektör alalım. V1, V2, V3 olsun.



Şekil 1: Örnek Üçgen

V1=(0,40,120) V2=(30,-40,60) V3=(-30,-40,60) olsun.Vektörel çarpım ve saat yönündeki yüzey kural gereği (V2-V1),(V3-V2) vektörlerinin hesaplanması ve vektörel çarpım yapılması gerekmektedir.

V2-V1=(30,-80,-60) (V3-V2) =(-60,0,0) Olduğuna göre; ***(xy,yx;yz,zy;zx,xz)***

Normal=(-80*0-(-60*0), -60*-60-30*0, 30*0-(-80*-60)) = N=(0,3600,-4800) Normalize edildiğinde (0,0.6,-0.8) bulunur.

3)Skaler Çarpım:

Skaler çarpım yönteminde ise her 2 köşe koordinatları ve aralarındaki açının kosinüsü çarpılır ve toplanır. Skaler çarpımı [1]'de bahsettiğimiz boyama yöntemlerinde daha detaylı göreceğiz.

Başlangıç noktası ve doğrultusu olan ışınların üretilme adı verilen bir yöntem vardır. Bu yöntemde ışının o doğrultu üzerinde kaç birim ilerleyeceğini hesaplıyoruz. Ray Tracing için bu oldukça önemlidir çünkü ışınımız bir nesneye çarpacak ve nesne ile bakış açısı arasındaki uzaklığın hesaplanması gerekmektedir. Biz ancak bu uzaklık ve doğrultu bilgisi ile yansıyan veya cismin içerisinden kırılan ışının, görüntü düzlemindeki pikselini çizebilir, boyayabiliriz.

Tracing ne anlama gelmektedir?

Tracing(t): Rd doğrultusu boyunca kaç birim(t) ilerleyeceğini söyler. $R=R_o+t*R_d$ (R_d= ışının doğrultusu, t: ışının doğrultu boyunca kaç birim ilerleyeceği, R_o ışının başlangıç noktasıdır.)
Bu şekilde yolladığımız ışının hangi konumda olduğunu bilebilir daha sonrasında da bu konuma göre kesişim testleri ve gerekli boyama teknikleri ile görüntümüzü çizdirebiliriz.

Bu durumu şu şekilde anlatabiliriz;
1)Işınlar yollanır bu ışınlara birincil ışın adı verilir.
2)Cisimler ile kesişim testleri yapılır ve kesişimler ile o cisme olan t uzaklığı hesaplanır.
3)Yakınlık sıralaması yapılır t uzaklığına göre en yakın cismin pikselinin rengi ekrana bastırılır.

Kesişim Testi ne işe yarar?

Kesişim testi ile ışın nesne ile nasıl etkileşim içerisine giriyor, çeşitli yöntemler kullanılarak bunlar hesaplanır. Örneğin yansıyan bir cisimdeki kesişim testi ile transparan bir cisimdeki kesişim testi veya saydam olmayan bir cisim ile kesişim testi birbirinden farklı algoritmalar ile hesaplanmaktadır.

Bütün cisimleri küçük küçük üçgen ve kürelerden oluştuğunu varsayabiliriz. Boyutlarını arttırıp azaltma, döndürme gibi işlemler ile bütün nesneler oluşturulabilir. Biz de üçgen ve küre kesişim testleri ile bütün nesnelerin görüntü düzlemindeki görüntüsünü hesaplayabiliriz.

Üçgen kesişim testi:

İlk olarak üçgenin tanımladığı yüzey denklemini çıkarmamız gerekli daha sonrasında üçgenin herhangi bir köşe noktasının koordinatını yüzey denklemindeki gereken yerlere yazdığımızda ve 0'a eşitlediğimizde yüzey denklemini bulabiliriz. Yüzey denklemi de üçgen üzerindeki herhangi bir noktayı sağlamaktadır.
Yüzey denklemini de yukarıda Tracing kısmında bahsettiğimiz gibi t uzaklığına göre düzenlediğimizde ve t'yi yalnız bıraktığımızda
(-N*Ro+D) / N*Rd =t olmaktadır. Buna göre;

t>0 ise kesişim vardır.
t<0 ise kesişmiyor demektir.
t=sonsuz ise ışın yüzeye paralel geçiyor denmektedir.

Kesişiyor bile olsa bu uzaklık hesabı ile öncelik hangi nesnenin pikseli çizileceği belirlenir.İkinci olarak ışın üçgenin alanının içerisinde mi bu hesaplanır.Buna alan testi yöntemi adı verilir.Bu yöntemde hesaplanan kesişim noktasından üçgenin köşelerine doğrular çizilir ve bu doğrular üçgenler oluşturur eğer ki bu üçgenlerin alanları toplam alana eşitse alan kesişim testi başarılı bir şekilde tamamlanmış olur ve sonrasında piksel ekrana bastırılır.

Küre kesişim testi:

Üçgen kesişim testinden farkı olarak çok fazla kesişme durumu vardır çünkü üç boyutlu bir cisim üzerinde çalışıyoruz. Işının başlangıç noktası doğrultusu yanında ışının başlangıç noktası ile küre merkezine olan uzaklığının ışın doğrultusu üzerine izdüşümü gibi durumlar gözetilerek algoritmalardan yararlanarak hesaplamalar yapılır. Bu algoritmalar şöyle;

Başlangıç noktasının küre merkezine olan uzaklığına x diyelim. Bu uzaklığın ışın doğrultusu üzerine izdüşümüne d, kürenin yarıçapına r diyelim.
D<0 ise;
Eğer başlangıç noktası dışarıda ise kesişim var
Başlangıç noktası kürenin içerisinde ise her zaman kesişim var
D>0 olduğu durumda ise pisagordan m^2 hesaplanır;
m^2=x^2-d^2 hesaplanır. Eğer m^2>r^2 ise kesişim yoktur.
M^2<=r^2 ise kesişim vardır. X^2<r^2 ise içten kesişim, m^2>r^2 ise dıştan kesişim vardır.
R yarıçaptı ve m^2 'yi hesaplamıştık. R^2 -m^2 ile q adı verilen küre dışındaki nokta ve merkez izdüşümü bulunabilir buna göre ise;
Dıştan kesiştiğinde ışın küre uzaklığı t=d-q olur. İçten kesiştiğinde t=d+q olur.

Phong boyama modeli ve renk bileşenleri

Phong boyama modeli:

Kesişim testleri ile hesaplanan piksel değerleri direkt olarak ekrana basılmaz. Öncelikle renk bileşenleri hesaplanmalı ve en son piksel değerinin hali ekrana basılmalı.Projemizde bu oldukça önemli yer tutmakta.

Diffuse renk bileşeni:

Yüzey normali ile ışık kaynağına doğru olan vektör skaler çarpılarak diffuse katsayı hesaplanır.Bu katsayı ile yüzeyin rengi çarpılarak phong boyama modelinin diffusse renk bileşeni hesaplanır.

Specular renk bileşeni:

Işık kaynağından yüzeye doğru olan vektörün normalinden başlangıç noktasına doğru olan vektör arasındaki açının değişken skaler bir değer ile çarpılarak hesaplanabilir. Bu katsayı ile ışık kaynağının rengi çarpılarak phong boyama modelinin specular renk bileşeni hesaplanabilir.

Ambient renk bileşeni:

Doğrudan aydınlatılamayan yüzeyler mevcuttur ve bu yüzeylere gelen alternatif (diğer nesnelerden yansıyan ışınlar) ışınlar bu yüzeyleri aydınlatır. Bunun için dışarıdaki cismin rengi ve 0-1 arasındaki aydınlatma miktarını belirten sayı ile çarpılarak ambient renk bileşeni hesaplanabilir.

Yansıma, saydamlık ve kırılma olaylarının tanımı ve gölge testi

Yansıma:

Gönderilen ışını yansıtan bir cisim için yansıyan ışının doğrultusu hesaplanır. Sonrasında bu doğrultu kullanılarak önceki bahsettiğimiz kesişim testleri yapılır ve en yakın cisim belirlendikten sonrasında yansıyan cismin rengi hesaplanır ve ekrana basılır.

Saydamlık:

Saydam cisimlerde herhangi bir yansıma olmayacağı için doğrultu sabit kalır. Fakat içinden geçtiği için kesişim testleri buna göre yapılır ve görüntü ekrana basılır.

Kırılma:

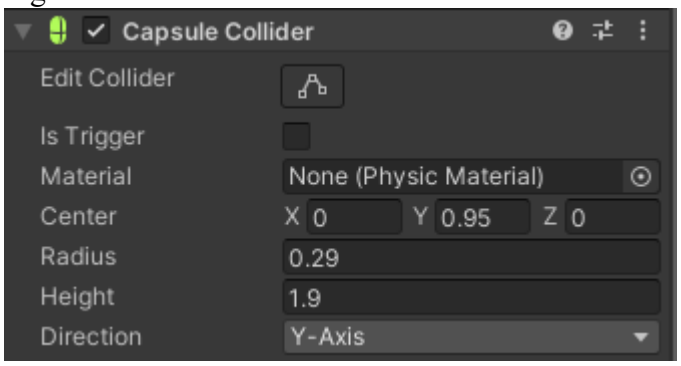
Işığın yoğunluk farkına göre cisimlerin içerisinden geçtiği zamanki doğrultu değiştirme olayıdır. Buna göre kırılma olayı gerçekleştiği andaki doğrultular tekrar hesaplanır ve kesişim testleri tekrar yapılır. En sonunda görüntü ekrana basılır.

Gölge Testi:

Herhangi bir cismin diğer bir cismin gölgesinde mi değil mi bunu belirleyebilmek için gölgenin düşeceği cisim üzerindeki kesişim noktasından ışık kaynağına doğrular çizilir. Bu ışın ile kesişim testleri yapılarak en yakın cisim belirlenir. Uzaklık değeri ışık kaynağına olan uzaklıktan küçük ise uzak olan cisim yakın olan cismin gölgesinde kalıyor demektir.

1.3.UNITY OYUN MOTORU İLE ALAKALI TERİMLER

Collider: Objelere eklenir ve fizik kullanılırken objelerin birbiri ile etkileşime geçmesini sağlar.



Şekil 2: Capsule Collider penceresi.

Rigidbody: Nesnelere fizik desteği sağlayan kod rigibody kodudur. Bir nesneye rigidbody kodu eklenirse bu nesne yerçekiminden etkilenir. Unity oyun motorunda nesnelerin fizikleri collider'lerine ile hesaplanır. Yani fizik nesnenin şekline göre değil içine atanan colider'e göre hareket eder, yerçekiminden etkilenir. Mass, drag, angular drag, use gravity gibi pek çok ayarı bulunmaktadır ve bu ayarlar sayesinde nesnelerin daha gerçekçi hareket etmesini sağlar.

Şekil 3: Rigidbody Penceresi

Start() Fonksiyonu: Script dosyası çalışınca update() fonksiyonundan önce awake() fonksiyonundan sonra devreye giren başlangıç fonksiyonudur.

Update() Fonksiyonu: Sürekli çalışan ve aslında oyundaki nesnelerin birbirleriyle iletişimde kalmasını sağlayan asıl fonksiyondur. Bu fonksiyon içine yazılan kodlar program sonlanana kadar veya scriptin bağlı olduğu nesne yok olana kadar çalışmaya devam eder.

FixedUpdate() Fonksiyonu: Görevi update fonksiyonu ile birebir aynıdır tek farkı fizik söz konusu olduğunda daha gerçekçi sonuçlar almamızı sağlar.

LateUpdate() Fonksiyonu: Diğer update metodlarından sonra çalışan bir fonksiyondur. Nesneyi oyun kamerası ile takip ederken kullanılır.

IsTrigger: Diğer collider'ları önemsemez. Bu da fizik olayların devreye girmesini engeller. Collison'ların çarpışıp çarpışmadığını kontrol etmek için kullanılır.

Physic Material: Nesnelerin herhangi bir çarpışma durumunda nasıl tepki vereceğini ayarlamamızı sağlar.

Fonksiyonların çalıştırma önceliği

Awake: Bu fonksiyon her zaman **Start** fonksiyonundan önce çalıştırılır. Ayrıca prefab Instantiate edildiği anda da çalıştırılır. Obje aktif değilse ya da objeyi etkileyen bir kod çağırılmadıkça bu kod çalıştırılmaz.

OnEnable: Bu fonksiyon script aktifleştirildiğinde çağırılır. Bu durum ise objeye yeni bir bileşen eklemesiyle, yeni bir sahne yüklenmesiyle ya da game objectin örneklendirilmesiyle gerçekleşir.

Start: Script aktifken update fonksiyonları hiç çalıştırılmamışken tek seferlik çalıştırılır.

OnApplicationPause: Ekrandaki mevcut işlemler bittikten sonra eğer durma durumuna geçilecekse bu fonksiyon çalıştırılır.

FixedUpdate: Oyunun kasma durumuna göre bazı kareler birden çok çalışabilir. Aşırı akıcı durumlar da hiç çalışmayabilir. Fizik hesaplamaları ve fiziksel olaylar bu update'den hemen sonra gerçekleşir.

Update: En sık kullanılan update çeşididir ve her karede sadece bir kez çalışır.

LateUpdate: Sahnedeki objeler için update metodu bittiğinde çalışır. Her bir karede yalnız bir kere çalışır.

OnGUI: Bir karede birden fazla kez çalıştırılabilir. Önce GUI elemanları ekrana dizilir ve çizdirilir. Ardından her girdi için bu işlem tekrarlanır.

OnDrawGizmos: Oyunu görsel olarak test ederken kullanılır. Scene paneline gizmo çizer.

yield: Corountine'i bir sonraki ekrandaki update fonksiyonuna kadar bekletir. Bu update fonksiyonu çalıştıktan sonra işleme sokulur.

yield WaitForSeconds(): Belli bir süre – saniye cinsinden- bekledikten sonra kodun çalışmasını sağlar.

yield WaitForFixedUpdate(): Tüm scriptlerde bulunan FixedUpdate fonksiyonlarının çalıştırılmasını bekler.

yield WWW: Bir WWW verisinin internetten indirilmesi tamamlandıktan sonra kodu çalıştırmaya devam eder.

yield StartCoroutine( ): İçine girilen coroutine fonksiyonu çalıştırır ve onun bitmesini bekler. Ardından mevcut kodu çalıştırmaya kaldığı yerden devam eder.

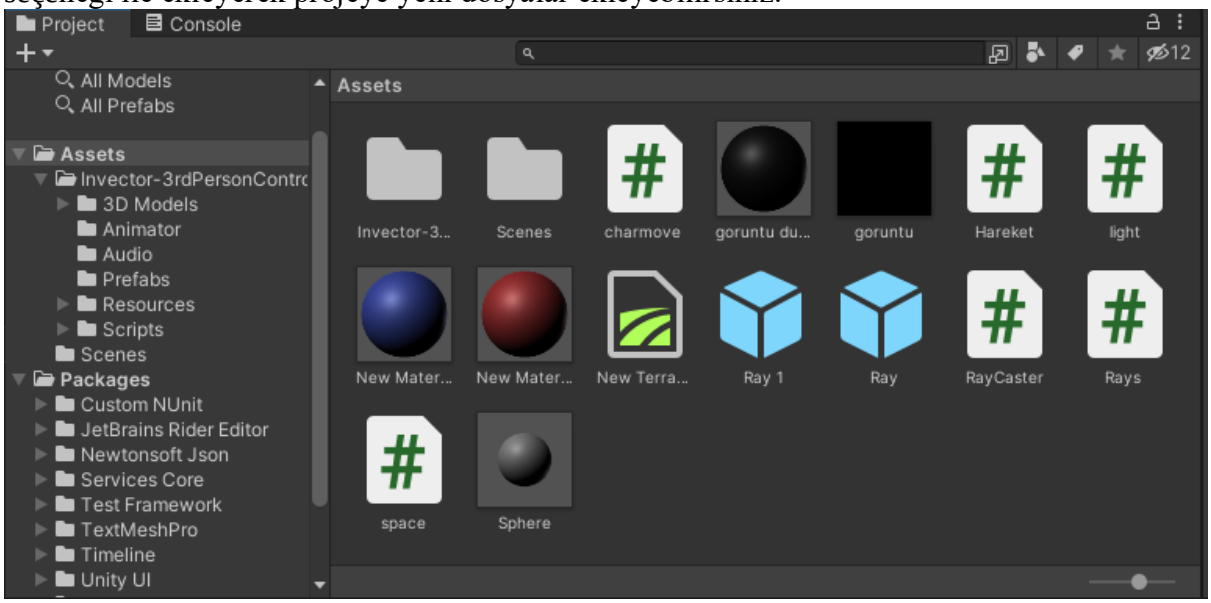
OnDestroy: Obje yok olmadan önceki son karenin tüm update fonksiyonları çalıştırıldıktan sonra gerçekleştirilir. Objenin yok olmasının sebebi destroy komutu kullanılması olduğu gibi başka bir sahneye geçme de olabilir.

OnApplicationQuit: Oyundan çıkılmadan önce gerçekleştirilen son komuttur.

OnDisable: Script disable yapıldığında ya da GameObject inactive yapıldığında çalıştırılır.

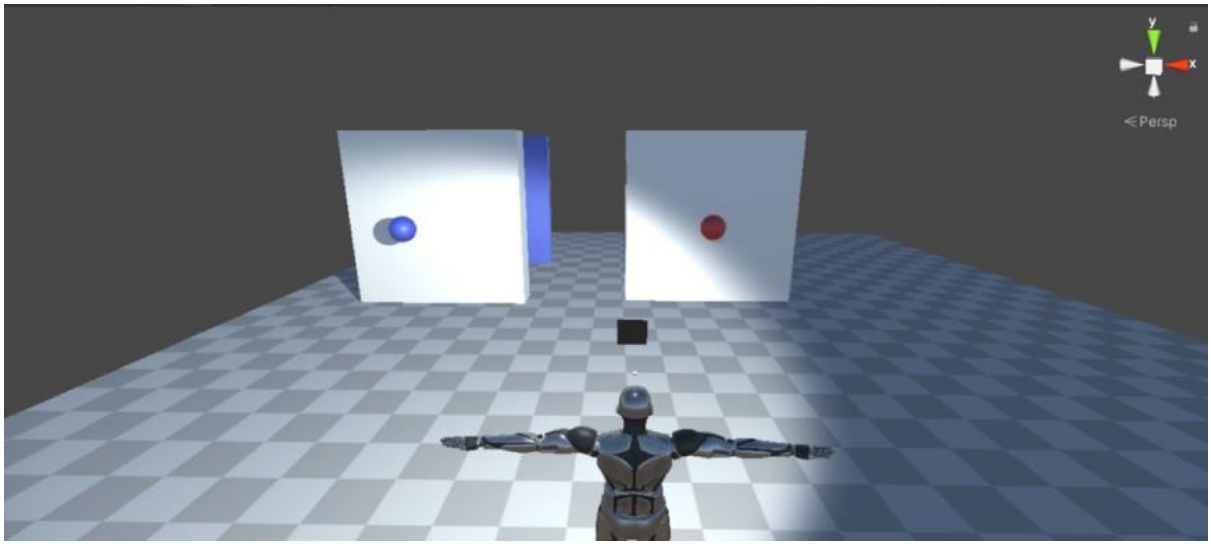
Varsayılan unity ekranında bulunanlar:

Project: Projede bulunan tüm dosyaların olduğu panel. Buraya dosya sürükleyerek ya da ekle seçeneği ile ekleyerek projeye yeni dosyalar ekleyebilirsiniz.



Şekil 4: Project penceresi (temsilidir).

Scene: O andaki pozisyonun sahnesi gösterilen panel.

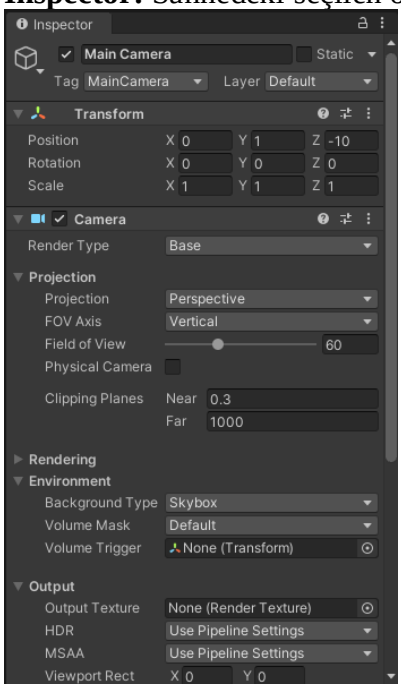


Şekil 5: Scene penceresi.

Hierarchy: Sahnede bulunan tüm objeler.

Şekil 6: Hierarchy penceresi (temsilidir).

Inspector: Sahnedeki seçilen objenin özelliklerini gösteren panel.



Şekil 7: Inspector penceresi.

Toolbar: Solda Pan (kaydırmak için), Move (hareket ettirmek için), Rotate (döndürmek için), Scale (ölçeklemek için) bulunur. Ortasında Play, Pause, Advance Frame vardır. Play tuşu ayrı derleme yapmak zorunda kalmadan oyunu başlatır. Pause ise oyunu duraklatır. Advance frame ise oyunu kare kare oynatır. Genelde oyundaki küçük hataların düzeltilmesi için kullanılır.

Console: Bu pencere gizlenebilir. Derlemeleri, hataları, uyarıları, çıktıları ve benzeri durumları gösterir. Ayrıca debug mesajları da burada gözükür.

1.9.C# PROGRAMLAMA DİLİNİN UNITY İLE KULLANIMI

Unity' de normal bildiğimiz c# yüksek düzey programlama dili(HLL) kullanılmaktadır. C# çoğu yüksek seviye programlama dilinden oyun için daha kullanışlı olanıdır çünkü makine dilinden çok uzaklaşıp insan diline yaklaşmıştır okuması ve yazması diğer dillere nazaran daha kolaydır. Unity bize bu konuda da destek olup öğrenmemiz için platform oluşturmuşlardır (bkz. https://learn.unity.com/). C# kullanılırken kütüphaneden kod çağırmak gerekir. Using UnityEngine; kullanılarak Unity kütüphanesindeki kodlar çağırılır ve kodlama için kullanılmaya hazır hale getirilir. C# kodunu yazmamız için bize bir text editör gerekmektedir, Unity Visual studio, notepad ya da süblime text kullanmamıza izin vermektedir. Unity oyununuzu test etmenize herhangi bir IDE kullanmadan izin vermektedir.

Metinlerin Unity' de çağırılabilmesi için sahnedeki GameObjecte bağlanması gerekmektedir. Metinler Unitynin anlayabileceği özel bir dilde yazılmış olması gerekir işte bu dil c#'tır. Unity işleyebildiği tüm diller nesne yönelimli programlama dilleridir. Bütün diller gibi bu dillerinde yazım kuralları ve temel parçaları vardır. Değişkenler, fonksiyonlar ve classları inceleyelim.

Değişkenler

Değişkenler değer tutan ve nesnelere referans olan yapılardır. Değişkenler her zaman küçük harfle başlar. Değişkenleri bizim için değer saklayan kutular olarak düşünebiliriz. Değişkenler public ve private olarak ikiye ayrılırlar.

```
5 public class DemoScript: MonoBehaviour {
6
7     public Light myLight;
8     private Light myOtherLight;
9
10 }
```

Şekil 8: Public ve private değişken tipleri.

Yukarıdaki kodu GameObjecte atarsanız public olarak gözüken light değişkenine erişebildiğinizi fakat private olanı göremeyeceğinizi görürsünüz bunun sebebi "private" olarak tanımlanan bir değişkene yalnızca belirli komut dosyasından bu belirli classın içinden erişilebilmesidir. Public'e çevirdiğimizde diğer classlara tekrardan açılır bunu Unityde Inspector olarak düzenleyebiliriz. Private değişkenler kodunuzun daha temiz olmasını sağlar bunu söylememizin sebebi bu değişkenlerin yalnızca o sınıf içerisinde değiştirilebildiğini bildiğimizden dolayı hata ayıklamamız ve bakım yapmamız kolaylaşmış olur. Public olarak kullandığımızda kodun izini sürmemiz zorlaşır ancak objelerin birbirleriyle iletişim halinde olması için ve bazı değişkenlere ihtiyacınız olabileceğinden public değişkenler kullanmak zorunda kalabilirsiniz. Değişkenlerin bir başka önemli özelliği de Type'dır. Typelar sayı, metin ya da daha karmaşık olabilirler. Unity sizin nesnenizi kullanmak için hangi type'da olduğunu bilmek zorundadır. Adlandırmalara gelecek olursak; adlandırmalara sayıyla başlayamazsınız ve boşluk bulunduramazsınız.

Fonksiyonlar

Fonksiyonlar değişkenleri karşılaştırıp üzerlerinde oynama yapan kod satırlarıdır. Fonksiyonlar her zaman büyük harfle başlar. Kodları fonksiyonlarla organize etmemizin sebebi birden çok kez tekrardan kullanımını kolaylaştırmasıdır.

Awake, bu bileşene sahip GameObject in somutlaştırılması durumunda yalnızca bir kez çağırılır. Bir GameObject aktif değilse tekrar aktif olana kadar çağırılmaz. Fakat, GameObject aktif olsa bileşen aktif olmasa bile awake çağırılır (isminin yanındaki küçük kutu ile). Bir değer atamanız gereken tüm değişkenleri başlatmak için awake fonksiyonunu kullanabilirsiniz.

Start, awake gibi start da bir GameObject aktifken çağırılabilir ancak bileşeninde erişilebilir olması şartıyla.

Update, frame başına bir kez çağırılır. Animasyonlar, AI ve oyunun sürekli güncellenmesi gereken diğer bölümleri gibi sürekli çalışan mantığı tanımlamak için kod koyduğumuz yer burasıdır.

FixedUpdate, fizikle alakalı işler yapmak istediğinizde kullandığınız fonksiyon.

LateUpdate,Update benzer ancak frame in sonunda çağırılır. Unity her işlemi tamamladıktan sonra LateUpdate i arayacaktır. Oyununuzda bir karakteri hareket ettirmek istediğinizi varsayalım ve bu karakter başka bir karakterler çarpışırsa başka bir pozisyonda olacağını düşünelim kamera ve karakteri aynı anda hareket ettirirsek titreme olur ve kamera olması gerektiği yerde olmaz Lateupdate bu durumun çözümüdür ve oldukça kullanışlıdır.

Classlar

Classlar nesnenin özelliklerini tanımlayan bir şablon oluşturmak için değişken ve fonksiyonları bir araya getiren yapılardır.
Classın adı her zaman C# dosyasının adıyla eşleşmek zorundadır. Classlar'da değişkenler gibi public ve private olabilirler.

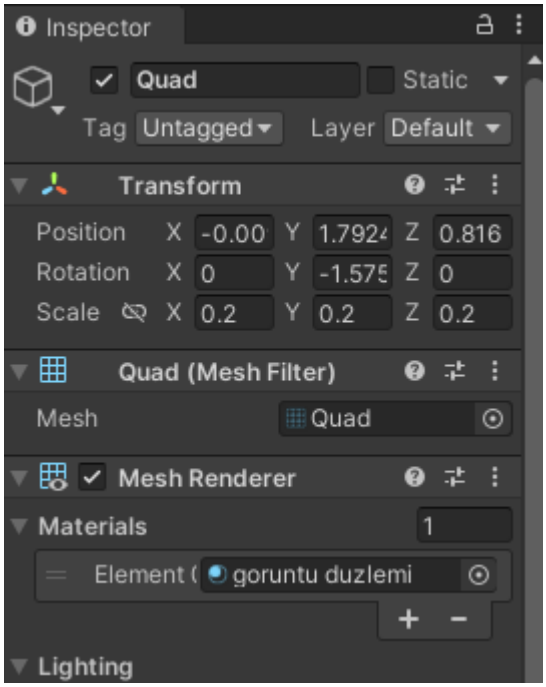
2.YAPILAN ÇALIŞMALAR

2.1. GÖRÜNTÜ DÜZLEMİ İLE IŞINLARIN BAĞLANTISI

Görüntü düzlemi adını verdiğimiz görüş alanı diye tabir edilen sanal bir görüntü düzlemi, karakterimizin gözlerinden çıkan ışınların değdiği yüzeylerin görüntüsünün piksellerini tek tek alıp kendi üstüne çizen bir düzlemdir. Bu görüntü düzleminden geçen hayali ışınlar ile 3d cisimler kesiştirilir. Kesiştirme işlemi esnasında kesişen ışınlar o yüzeyin üstünde kalarak arka tarafa geçmezler. Bu sayede öndeki cismin arkasında kalan bir cismin görüntüsü de görüntü düzlemine çizilmez.

Projemizde görüntü düzlemi ile ışınlar hayali yani gözle görülemeyen varsayılmakta ancak öğretici olması açısından bu ışınlar ve görüntü düzlemi gözle görülür hale getirilmeye çalışılmıştır. Işınlarımızın hareketi dağılımı gibi olaylar sayesinde gerçekçi bir görüntü sağlanmaya çalışılmıştır.

Görüntü düzlemini Unity'de hazır olan *Plane* yani 2 boyutlu bir düzlem olarak ekledik (Quad ismi ile) ve dokusuna da Ray temel ışınının gördüğü görüntüyü atadık. Ray clone ışınları ise yine aynı yöntem ile görüntü düzlemine atadık. Pikselleri bu şekilde görüntü düzlemine sırasıyla çizmiş olduk.



2.2. IŞINLARIN DAĞILIMI

Işınların dağılımı için çoğaltma yönteminden yararlandık. 2.1. Bölümünde bahsettiğimiz temel Ray(Oyun nesnesi) 'ni ve temel kamerayı kullanarak bir kod *"RayCaster.cs"* dosyası aracılığı ile bu oyun nesnelerini çoğaltmayı ve eşit aralıklarla dağıtmayı başardık. Kod içerisinde iki adet iç içe for döngüsü kullandık ilk döngü x eksenindeki kopyalamayı sağlıyor içerideki for döngüsü de y eksenindeki kopyalamayı sağlıyor.

I ve j değerleri ***(0,0)*** olduğu zaman yani (temel prefab ve kameramızın merkez konumu) için herhangi bir kopyalama yapmamıza gerek yoktur, bu yüzden ***(0,0)*** için for döngüsüne girmiyor ve kısaca;

(1,1)-(1,sizey) kadar y ekseninde,

(1,1)-(sizex,1) kadar x ekseninde yani toplam ***(1,1)*** ve ***(sizex,sizey)*** kadar bir matris dizisi oluşturduğumuzu ve her bir koordinat için benzer kopyalama işlemleri yaptığımızı söyleyebiliriz.

Ray oyun nesnesi çoğaltıldığında *rayI* ve *spherel* adındaki iki nesneyi referans alarak döngüye girip kod çalışmaya başlayacak. *Instantiate* adlı fonksiyona(yeni obje üreten yani kopyalayan bir fonksiyon) parametre olarak; *ray* veya *sphere* temel nesnelerimizi veriyoruz (çoğaltılacak nesne), sonrasında iki boyutlu koordinatları alıyor bunları *i* ve *j* ile çarparak kopyalanmış nesnelerin güncel konumları(eşit aralıklı konumları) hesaplanıyor ve setleniyor.

```
rayI = Instantiate(ray, new Vector3(i * 0.01f, j * 0.01f, 0f), Quaternion.identity);
spherel = Instantiate(sphare, new Vector3(i * 0.01f, j * 0.01f, 0f), Quaternion.identity);
```

rayI kopyalanmış nesnesi için çalışan *"Rays.cs"* kodu için *num* değişkenlerine koordinatlar setlenerek kameraların konumları değişiyor.

2.3. IŞINLARIN HAREKETİ

Enabled komutu o kod için bir bool değişkendir. False ise o nesne için script çalışmayacak anlamına gelmektedir. Bunu yapmamızın sebebi, kullanıcı space tuşuna bastığında bu kodun çalışmasını sağlamak. Yani *ray* ve *sphere* nesneleri ve klonları yerlerinde sabit duracaklar. Space tuşuna bastığımızda ayrı bir kod çalışacak ve hareket etmeye başlayacaklar.

```
rayI.GetComponent<Rays>().num = new Vector2(i, j);
rayI.GetComponent<Rays>().enabled = false;
rayI.GetComponent<Hareket>().enabled = false;
```

RayI ve *sphereI* kopyalanmış nesnelerin referans alacağı *ray* ve *sphere* nesnelerinin konumları setleniyor. "*.start"* ile de *ray* ve *sphere* nesneleri hareket etmeye başladığı zaman kopyalanmış nesnelerin de hareket etmeye başlayacağı tanımlanmış.

```
rayI.transform.parent = gameObject.transform;
spherel.transform.parent = gameObject.transform;
rayI.GetComponent<Hareket>().start = gameObject;
spherel.GetComponent<Hareket>().start = gameObject;
```

Kodun birçok yerinde değişkenler değişerek kodun tekrar etmesi söz konusu olmakta çünkü ışınlarımız sadece düz hareket etmiyor, ***(+x,-x)*** ve ***(-y, +y)*** yönlerinde de hareket etmektedir. Yönler için ise tekrar kod satırları işin içerisine giriyor.

X eksenindeki ışınların hareketi ve dağılım işlemleri için küçük farklılıkların gösterildiği ***"RayCaster.cs"*** kodunun belirtilen kısmı aşağıdaki gibidir.

```
for(int i = 0; i<= sizex; i++)
{
    for (int j = 0; j <= sizey; j++)
    {
        if (!(i == 0 && j == 0))
        {
            if (i != 0)
            {
                rayI = Instantiate(ray, new Vector3(i * 0.01f, j * 0.01f, 0f), Quaternion.identity);
                spherel = Instantiate(sphare, new Vector3(i * 0.01f, j * 0.01f, 0f), Quaternion.identity);
                rayI.GetComponent<Rays>().num = new Vector2(i, j);
                rayI.GetComponent<Rays>().enabled = false;
                rayI.GetComponent<Hareket>().enabled = false;
                rayI.transform.parent = gameObject.transform;
                spherel.transform.parent = gameObject.transform;
                rayI.GetComponent<Hareket>().start = gameObject;
                spherel.GetComponent<Hareket>().start = gameObject;
                rayI = Instantiate(ray, new Vector3(-i * 0.01f, -j * 0.01f, 0f), Quaternion.identity);
                spherel = Instantiate(sphare, new Vector3(-i * 0.01f, -j * 0.01f, 0f), Quaternion.identity);
                rayI.GetComponent<Rays>().num = new Vector2(-i, -j);
                rayI.GetComponent<Rays>().enabled = false;
                rayI.GetComponent<Hareket>().enabled = false;
                rayI.transform.parent = gameObject.transform;
                spherel.transform.parent = gameObject.transform;
                rayI.GetComponent<Hareket>().start = gameObject;
                spherel.GetComponent<Hareket>().start = gameObject;
            }
```

Y eksenindeki ışınların hareketi ve dağılım işlemleri için küçük farklılıkların gösterildiği ***"RayCaster.cs"*** kodunun belirtilen kısmı aşağıdaki gibidir.

```
if (j != 0)
{
    rayI = Instantiate(ray, new Vector3(i * 0.01f, -j * 0.01f, 0f), Quaternion.identity);
    spherel = Instantiate(sphare, new Vector3(i * 0.01f, -j * 0.01f, 0f), Quaternion.identity);
    rayI.GetComponent<Rays>().num = new Vector2(i, -j);
    rayI.GetComponent<Rays>().enabled = false;
    rayI.GetComponent<Hareket>().enabled = false;
    rayI.transform.parent = gameObject.transform;
    spherel.transform.parent = gameObject.transform;
    rayI.GetComponent<Hareket>().start = gameObject;
    spherel.GetComponent<Hareket>().start = gameObject;
    rayI = Instantiate(ray, new Vector3(-i * 0.01f, j * 0.01f, 0f), Quaternion.identity);
    spherel = Instantiate(sphare, new Vector3(-i * 0.01f, j * 0.01f, 0f), Quaternion.identity);
    rayI.GetComponent<Rays>().num = new Vector2(-i, j);
    rayI.GetComponent<Rays>().enabled = false;
    rayI.GetComponent<Hareket>().enabled = false;
    rayI.transform.parent = gameObject.transform;
    spherel.transform.parent = gameObject.transform;
    rayI.GetComponent<Hareket>().start = gameObject;
    spherel.GetComponent<Hareket>().start = gameObject;
```

"Hareket.cs" kodunun ışınların hareketindeki etkisi ise, space tuşuna basıldığında ışınların tanımlanan *start* nesnesinin konumu kullanılarak yani belirli bir uzaklıktaki mesafeye setlenmiş konum kullanılarak bu sabit konumdan çıkıp dağılmasını sağlamaktır. Hız ve açılma değişkenleri ise ışınlarımızın z yönündeki hızını, x ve y yönündeki açılmaların miktarını belirlemekte.

```
public class Hareket : MonoBehaviour
{

    public float hiz;
    public float acilma;
    public GameObject start;

    // Start is called before the first frame update
    Unity iletisi | 0 başvuru
    void Start()
    {

    }

    // Update is called once per frame
    Unity iletisi | 0 başvuru
    void Update()
    {
        GetComponent<Rigidbody>().AddForce(new Vector3(((transform.position.x - start.transform.position.x) * acilma - GetComponent<Rigidbody>().velocity.x),
            ((transform.position.y - start.transform.position.y) * acilma - GetComponent<Rigidbody>().velocity.y),
            10f * (hiz - GetComponent<Rigidbody>().velocity.z)));
    }
}
```

"Rays.cs" kodunda ise kameraya sabit bir konum veriyoruz. Kameranın biçimi de dikdörtgen prizma şeklinde verilmiştir tek bir yönde bir piksel görüntüsünü alması için bu gereklidir.

"Raycaster.cs" kodundaki num değişkenini kullanarak doğrusal şekilde görüntüyü alan kameraların görüş alanlarını belirleyen kod satırları ve kameraların bir nesneye çarptığında durmasını sağlayan kod satırları verilmiştir. Kameralarımız ***"Finish"*** tagına çarptığında durması ve kamerayı açması gerekmektedir. Bu işi yapan fonksiyon ise ***OnTriggerEnter()*** fonksiyonudur.

```
public class Rays : MonoBehaviour
{
    static public float dif;
    public Camera kamera;
    public Vector2 num;

    // Start is called before the first frame update
    Unity İletisi | 0 başvuru
    void Start()
    {
        kamera.rect = new Rect(0.4f + num.x * 0.1f / 2f, 0.4f + num.y * 0.1f / 2f, 0.05f, 0.05f);
    }

    // Update is called once per frame
    Unity İletisi | 0 başvuru
    void Update()
    {
        if (num.x==0 && num.y == 1)
        {
            dif = transform.position.y;
        }
        kamera.orthographicSize = 0.035f + dif / 10f;
    }

    Unity İletisi | 0 başvuru
    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Finish")
        {
            kamera.enabled = true;
            GetComponent<Rigidbody>().isKinematic = true;
        }
    }
}
```

Bahsettiğimiz space tuşuna basıldığında ışınların hareketi için ***"Rays, RayCaster ve Hareket.cs"*** kodlarının aktif hale getirilmesini sağlayan kod dosyası aşağıdaki gibidir. Bool değişkenlerini true yaptığımızda kodlarımız algılanır ve çalışır hale gelmektedir. Hatırlarsak ***"RayCaster.cs"*** kod dosyasında da bu değişkenler false olarak ayarlanmıştı.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5
6   public class space : MonoBehaviour
7   {
8       // Start is called before the first frame update
9       void Start()
10      {
11
12      }
13
14      // Update is called once per frame
15      void Update()
16      {
17          if(Input.GetKey(KeyCode.Space))
18          {
19              GetComponent<Hareket>().enabled = true;
20              GetComponent<Rays>().enabled = true;
21              GetComponent<RayCaster>().enabled = true;
22          }
23      }
24  }
```

2.4.KARAKTER HAREKETİ

Karakterin hareketi için çok basit bir kod dosyası hazırladık. Bunun sebebi ise karakterin hali hazırdaki hareket kodları ile ışınlarımızın çakışması ve sorun oluşması idi. Kod dosyasında sağ ok, sol ok, üst ok, aşağı ok, d ve a tuşları ile çeşitli ilerletme işlemleri yapılmakta.

Sağ ok sağa doğru ilerletme, sol ok sola doğru ilerletme, üst ok ileri doğru ilerletme ve aşağı ok geriye doğru ilerletme iken d tuşu yukarı doğru kaldırma ve a tuşu ise aşağı doğru indirme işlemini yapmakta.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class charmove : MonoBehaviour
{
    private float speed = 2.0f;
    public GameObject character;

    void Update()
    {

        if (Input.GetKey(KeyCode.RightArrow))
        {
            transform.position += Vector3.right * speed * Time.deltaTime;
        }
        if (Input.GetKey(KeyCode.LeftArrow))
        {
            transform.position += Vector3.left * speed * Time.deltaTime;
        }
        if (Input.GetKey(KeyCode.UpArrow))
        {
            transform.position += Vector3.forward * speed * Time.deltaTime;
        }
        if (Input.GetKey(KeyCode.DownArrow))
        {
            transform.position += Vector3.back * speed * Time.deltaTime;
        }
        if (Input.GetKey(KeyCode.D))
        {
            transform.Rotate(Vector3.up * Time.deltaTime);
        }
        if (Input.GetKey(KeyCode.A))
        {
            transform.Rotate(Vector3.down * Time.deltaTime);
        }
    }
}

```

2.5. IŞIK KAYNAĞI VE HAREKETİ

Işık kaynağını eklememizin sebebi hoş bir görüntü oluşturmasının yanında ışınlarımız bir gölge yakaladığı zaman o görüntüyü de gölgeli şekilde puslu bir görüntü halinde almasını ve görüntü düzlemine o şekilde çizdirilmesini sağlamaktır. Yani daha gerçekçi bir görüntü yakalamak istediğimizden dolayı ışık kaynağını kullandık.
Kodun açıklamalara geçersek eğer;

IsikKaynagi adlı oyun nesnesinin içerisine ışık kaynağımızı yerleştiriyoruz ve bu sayede koordinatlarını alıyoruz. Bu koordinatları almamızın sebebi döndürme esnasında sürekli güncel bir koordinata ihtiyaç duymamız.

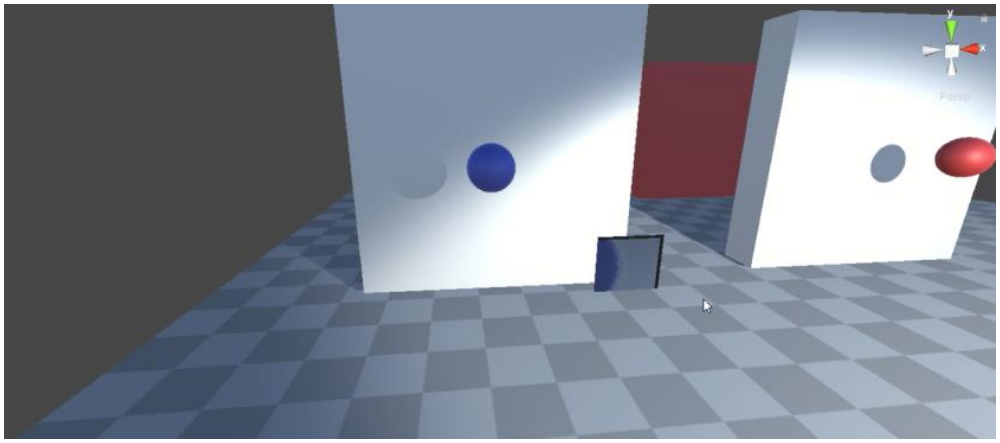
RotateAround() fonksiyonu aracılığı ile de ışık kaynağının hareketine hoş bir görüntü vermiş oluyoruz.

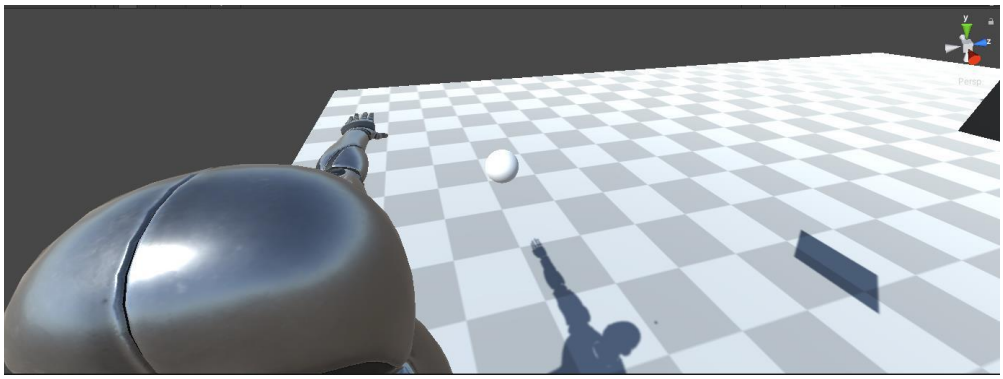
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Betiği | 0 başvuru
public class light : MonoBehaviour
{
    public GameObject IsikKaynagi;//Konum elde etmek için
    public Vector3 eksen;//hangi eksende dönecek? x,y veya z.Program içinden y ekseni etrafı olarak 0.25 ayarlandı.
    public float aci; //dönüş hızı.
    // Update is called once per frame
    Unity İletisi | 0 başvuru
    void Update()
    {
        //Işığı x,y,z ekseninde döndüren rotate kısmı.
        transform.RotateAround(IsikKaynagi.transform.position, eksen, aci);
    }
}
```

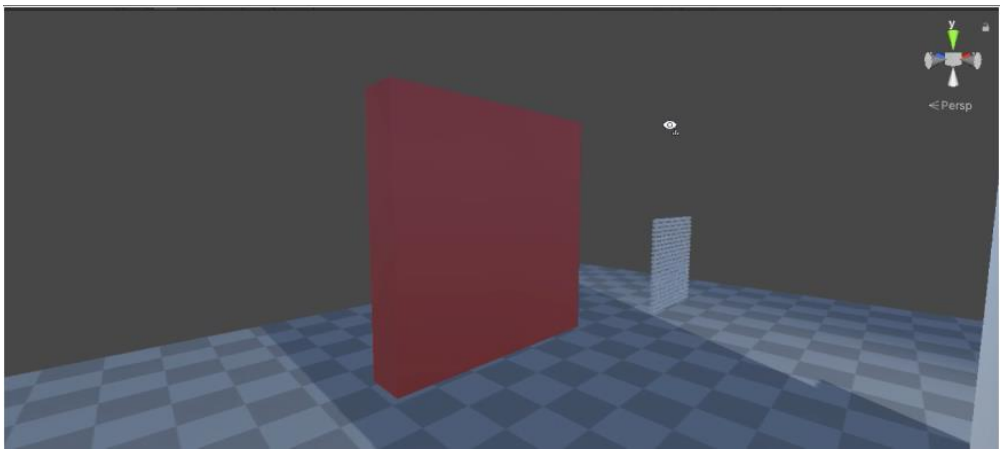
3.SONUÇLAR

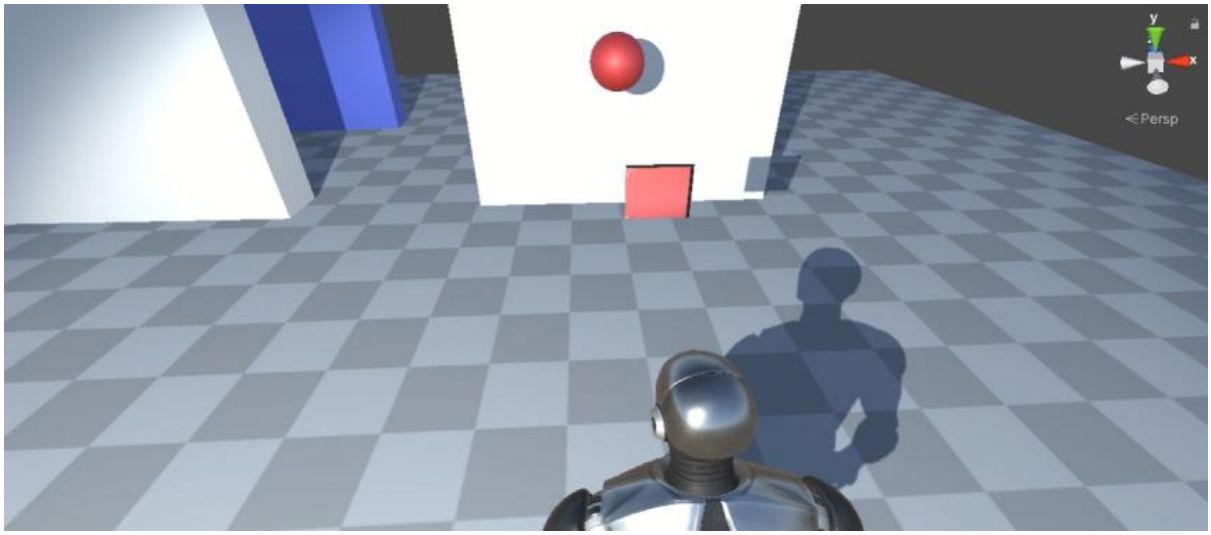
Gerekli kodlar yazıldıktan sonra proje tamamlanarak çalıştırılır. Bitirme projesi böylece tamamlandı. Bir proje yapılmak istendiğinde neler yapılması gerektiği öğrenildi. Projeye hemen başlamak yerine öncelikle planlama yapılmalı, proje için hangi dilin kullanılacağı ve nasıl bir algoritma kurulacağı belirlenmeli, bir projenin kullanıcı için en optimize halinin oluşturulması gibi önemli kazanımlar elde edildi. Projenin yazımında iyi bir yazılım bilgisi ve kod ile oyun nesneleri arasındaki bağlantıyı çözmemiz gerekliydi.

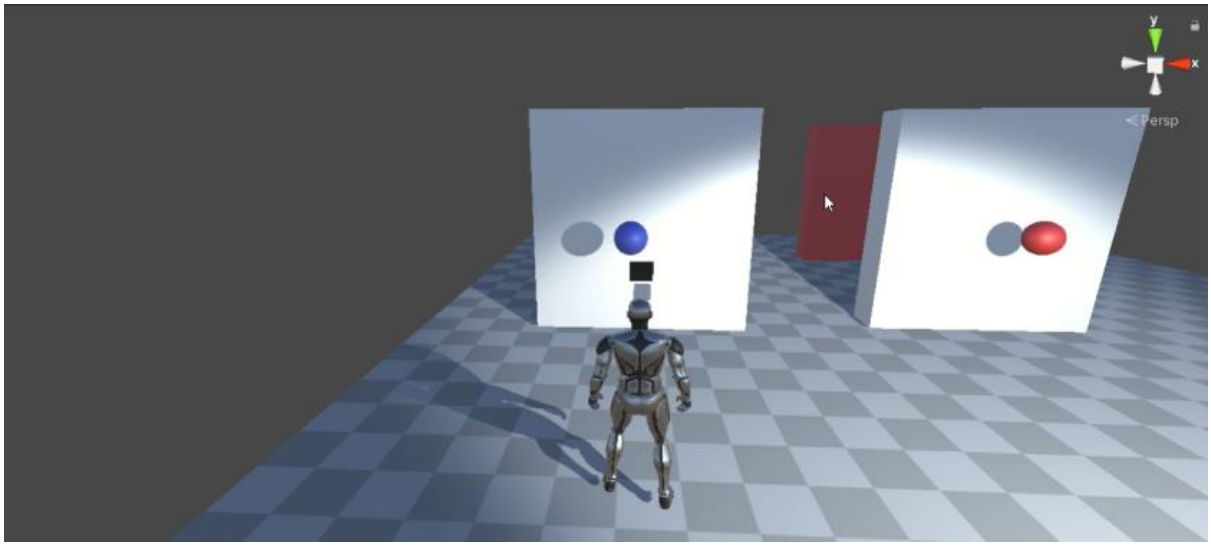
Okulda öğrenilen bazı teorik bilgilerin ve danışman hocamız Öğr. Gör. Ömer Çakır' ın yönlendirmeleri ile projenin son hali aşağıdaki şekillerdeki gibidir.

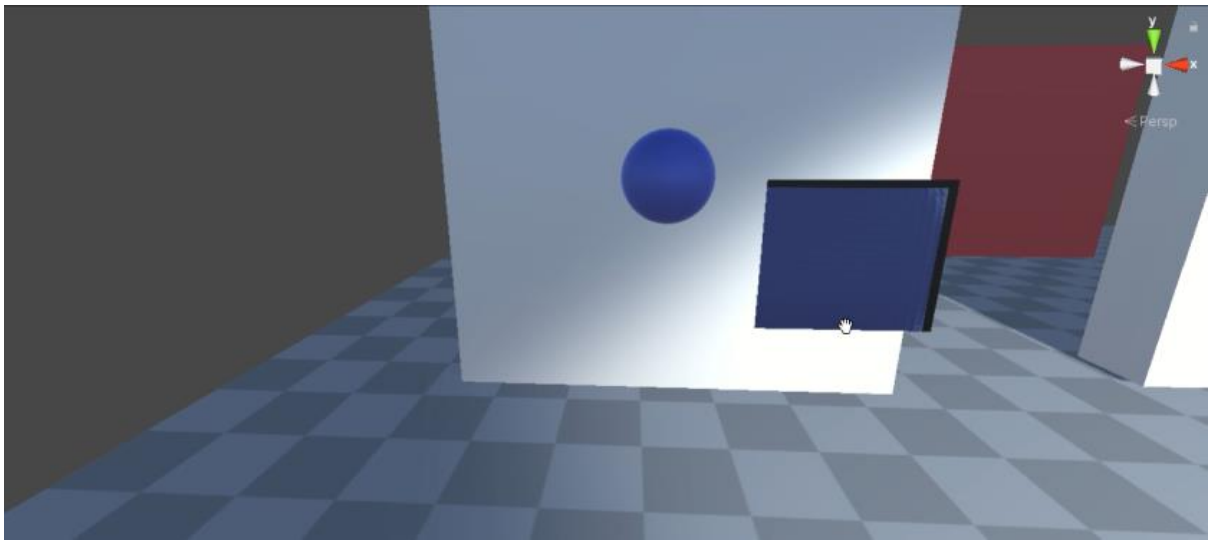







y
x
Persp


y
x
Persp


y
x
Persp

5.KAYNAKLAR

https://tr.wikipedia.org
https://learn.unity.com
https://www.raywenderlich.com/unity/paths/learn
https://www.tutorialspoint.com/unity/index.htm
https://www.gamedesigning.org/learn/unity/
https://www.ceng2.ktu.edu.tr/~cakir/grafikler-1.html
https://www.udemy.com

STANDARTLAR ve KISITLAR FORMU

Projenin hazırlanmasında uyulan standart ve kısıtlarla ilgili olarak, aşağıdaki soruları cevaplayınız.

1. Projenizin tasarım boyutu nedir? (Yeni bir proje midir? Var olan bir projenin tekrarı mıdır? Bir projenin parçası mıdır? Sizin tasarımınız proje toplamının yüzde olarak ne kadarını oluşturmaktadır?)

Yeni bir projedir. Işın izleme yöntemini, oyunu oynayan kişilere görsel ve eğitici bir şekilde öğretmesi hedeflenmektedir. Bu proje, Doç. Dr. Ömer Çakır ve Prof. Dr. Rıfat Yazıcı Hocalarımızın 2009 yılında yapmış oldukları "IŞIN İZLEME YÖNTEMİNİN GÖRSEL EĞİTİMİ" isimli projeden esinlenilerek yapılmıştır.

2. Projenizde bir mühendislik problemini kendiniz formüle edip, çözdünüz mü? Açıklayınız.

Hayır.

3. Önceki derslerde edindiğiniz hangi bilgi ve becerileri kullandınız?

Elbette, Doç. Dr. Ömer Çakır hocamızın Bilgisayar Grafikleri ve Bilgisayar Grafikleri Laboratuvarı derslerinde edindiğimiz modelleme ve animasyon bilgi, becerilerini kullandık. Bu bize hem teşvik edici oldu hem de projenin kalanında fikir edinmemiz açısından faydalı oldu.

4. Kullandığınız veya dikkate aldığınız mühendislik standartları nelerdir? (Proje konunuzla ilgili olarak kullandığınız ve kullanılması gereken standartları burada kod ve isimleri ile sıralayınız).

Oyun içerisindeki nesnelerin etkileşimi ve fiziği konusunda çeşitli C# algoritmalarından destek alınmıştır.

5. Kullandığınız veya dikkate aldığınız gerçekçi kısıtlar nelerdir? Lütfen boşlukları uygun yanıtlarla doldurunuz.

a) Ekonomi

Unity ücretsiz bir oyun motoru olduğu için, ekonomik bir problem ile karşılaşmadık.

b) Çevre sorunları:

Çevreye herhangi bir zararı olmayan bir projedir.

c) Sürdürülebilirlik:

Projemiz geliştirmeye açıktır. Uzun vadede, daha uzun geliştirme süresi, finansal araçlar ve kullanıcı geri bildirimleri ile daha başarılı, daha ilginç ve daha ilginç olabilir.

d) Üretilebilirlik:

Projemiz öğretici oyun projesi olduğu için herhangi bir üretilebilirlik sorunu söz konusu değildir.

e) Etik:

Etiksel bir problem yoktur. Projemizde kullanılan assetler Unity Market'ten alınan assetlerdir. Kodları kendimiz yazdık.

f) Sağlık:

Fazla oyun oynanması hem gözler açısından hem de zihinsel açıdan pek sağlıklı değildir.

g) Güvenlik:

Unity tabanlı proje olduğu için güvenlik sorunu olduğu takdirde, gerekli güncellemeler ile güvenlik sorunu ortadan kaldırılacaktır.

h) Sosyal ve politik sorunlar:

Bu proje böyle bir sorun teşkil etmemektedir.