KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Unity ile Işın İzleme Yönteminin Etkileşimli Görsel Eğitimi

MÜHENDİSLİK TASARIMI

M.Emin Altınışık
Ümit Şahan
Fahrettin Efe Okudan

2020-2021 BAHAR DÖNEMİ

KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Unity ile Işın İzleme Yönteminin Etkileşimli Görsel Eğitimi

MÜHENDİSLİK TASARIMI

Öğr. Gör. Ömer Çakır

2020-2021 BAHAR 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 Tasarım Projesi olarak hazırlanmıştır. Unity ve ışın izleme yönteminin kombinasyonu ile nasıl öğretici şekilde bir oyun ortaya çıkarabiliriz bunu görmeye çalıştık.

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 2021

İÇİNDEKİLER

ÖZET

Teknolojinin ilerlemesi ile birlikte oyun dünyası da bundan payını almış ve büyük bir hızla gelişim göstermiştir. Bu gelişimlerden birisi de Ray Tracing adlı yöntemdir.

Projemizde Ray Tracing yönteminin kullanım alanları olan yansıma, kırılma, saydamlık vb. gibi ışık kaynağının nesneler üzerindeki etkisini görsel ve etkileşimli bir şekilde, oyunu oynayan kişiye eğlenceli bir şekilde öğretmeyi hedefledik. Bunun için ücretsiz oyun motorlarından Unity ve Unity'nin desteklediği kodlama dilinden C# dilini kullandık.

1. GENEL BİLGİLER

1.1. GİRİŞ

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 yol açmıştır.

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. OYUN NEDİR

Oyun eğlence ve mutluluk amacıyla oluşturulan, kurallara sahip olan ve rekabetçi ruh aşılayan eğitici bir araçtır. Oyunlar bazen tamamıyla eğlence için bazense başarımlar ve ödüller için oynanırlar. Tek başına, takımlar halinde, online olarak amatörler ya da profesyoneller tarafından oynanabilirler. Oyuncular oyuncu olmayan kişiler tarafından izlenebilirler, izleyicilere de keyif verebilirler. Oyunların temel içerikleri; amaçlar, kurallar, rekabet ve interaksiyondur. Oyunlar genellikle zihinsel ya da fiziksel uyarıcılardır ve genellikle her ikisini de uyarır. Birçok oyun pratik becerileri geliştiren egzersizler gibi işlerler ve insanlar üzerinde eğitici ve psikolojik bir rol oynarlar.

Oyunlar genellikle oynandıkları araçlara göre sınıflandırılırlar (örneğin top, kart, masa ya da bilgisayar oyunları). Oyunlar araçlarla tanımlandığı gibi kurallarıyla da tanımlanabilirler. Kurallar oyundan oyuna değişikliler gösterirler. Genellikle zamana, oyuncuların haklarına ve sorumluluğuna ve oyuncuların amaçlarına bağlıdır. Rekabetçi oyunlarda mukaddes amaç kazanmaktır. (Örneğin satrançta amaç mat etmektir).

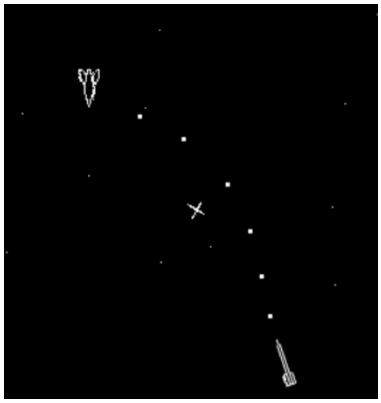
Genel zafer yolu olan yarışta birinci olmaktır gerek puan toplayarak gerekse rakibinden fazla araca sahip olarak vb. birçok yöntemle zafer kazanılabilir. Oyunlarda aynı zamanda kısa vadede amaçlar vardır. Bu amaçları zafere giden yoldaki küçük görevler olarak adlandırabiliriz. Örnek olarak futbolda amaç gol atmaktır ve gol atmak için bir organizasyon yapmanız gerekmektedir bu organizasyonları küçük görevler olarak değerlendirebiliriz. Oyunlarda amaç her ne kadar kazanmak olsa da bir oyuncu kazanmak zorunda değildir. Örneğin satrançta mat etmek kazanmaktır ancak bir oyuncu diğerini mat etmek zorunda değildir.
Amaçlara ulaşmak birtakım becerilere ihtiyaç duyar (bazense şansa).

Beceri strateji ve şans

Bir oyunun kuralları ve araçları haliyle beceriye, stratejiye veya şansa bazen de hepsine ihtiyaç duyar ve bunlara göre sınıflandırılabilirler. Örnek verecek olursak hedef vurmalı oyunlar, güreş, ip çekme, sek sek gibi oyunlar beceriye dayalı; satranç, dama, üçtaş gibi oyunlar stratejiye dayalı; blackjack, taş-kağıt-makas, zar oyunları gibi oyunlarda şansa dayalı oyunlardır.
Birden fazla alanda sınıfa tabi olan oyunlara örnek olarak beysbol, basketbol, monopoly ve bilgisayar oyunlarını verebiliriz.

Bilgisayar oyunları

1950'li yılların başında bilgisayar bilimcileri elektronik makineleri basit oyun sistemleri geliştirmek için kullanmaya başladılar. Bu sistemler elektronik ışık kullanmışlardır amaç bilgisayarların potansiyel gücünü göstermekti.

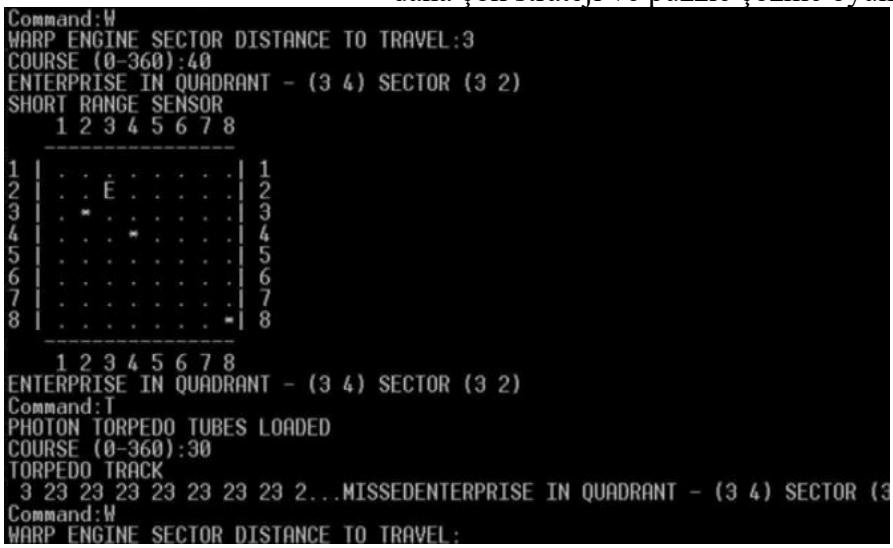


Şekil 1: Spacewar Oyunu

Spacewar, kabul gören ilk bilgisayar oyunu olmuştur.1961'de MIT deki bir grup öğrenci tarafından oluşturulmuştur. Bu oyun iki kişiyle oynanabilen bir uzay savaş oyunuydu. Oyunun kaynak kodu çevredeki diğer MIT öğrencileriyle paylaşılıp popülaritesi artırılmıştır.

1970'li yılların sonlarına doğru BASIC ve yüksek seviye programlara dili olan C bu yıllarda kabul gördü bu durum diğer teknik dillerden (FORTRAN ve COBOL) çok daha kolay erişilebilir olduğu için bilgisayar oyun geliştiriciliği yaygınlaştı.

ARCADE ve ev konsollarında en sık oynanan türler yarış ve hedef vurma oyunlarıydı. İşlemcilerin ve RAM in gelişmesiyle daha çok strateji ve puzzle çözme oyunları yaygınlaştı.



Şekil 2: Startrek Oyunu

Star Trek, Hunt the Wumpus ve Empire bu oyunlara örnek verilebilir. O dönemin en kaydadeğer oyunu ise Colossal Cave Adventure idi.

Çoğu oyun limitli grafik kapasitesi olan sistemlerde geliştirilirken Illinois üniversitesi PLATO adı verdikleri sistemle en etkileyici oyunları çıkarmaya başlamışlardı. 100 den fazla kullanıcıya ev sahipliği yapabilen uzaktan terminallere sahip olan bu sistem yüksek kalite plazma gösterimi kullanarak insanların birbiriyle etkileşime geçebilmesini sağladı. Bunun sonucunda ise multiplayer oyunlar yaygınlaşmaya başladı.

1977 Trinity: Commodore Pet ve Apple II nin gelişi ile ev bilgisayarları daha karşılanabilir oldu ve yaygınlaşmaya başladı. O dönemlerde zaten popüler olan oyunların kaynak kodları kitaplarda, dergilerde ve gazetelerde yayımlanmaya başlandı bu sayede kullanıcılar kendi kodlarını kendileri yazmaya başladılar.1983'te oyun sektöründe piyasadaki doygunluktan dolayı büyük bir çöküş yaşandı piyasa yeni cihazlar girene kadar bu durum böyle devam etti. Yeni gelen sistemler ev bilgisayarlarını ve oyun piyasasını katalize etti reklam çalışmaları başladı en dikkate değer olan ise BBC tarafından yapıldı. Bu zamana kadar "Bedroom Coders" olarak adlandırılan grup kendi yazılımlarını satmaya başladılar ve bu platformlar sayesinde küçük de olsa profesyonel yapılar oluşmaya başladı. Küçük yayıncı ve dağıtıcı gruplar bireylere oyunlarının kopyalarını satmasında yardımcı oldu. 1990'larda ise bilgisayar oyunlarında inovasyon dönemi olarak adlandırılabilir. Raster grafiklerden 3d grafiklere geçiş FPS, RTS ve MMO oyunlarda gerçekleşmeye başladı. Takip eden yıllarda da Playstation video oyun konsolu oluşturuldu.

1.3.Görsel Eğitimin Önemi

Günümüzden yaklaşık olarak 32000 yıl önce yaşayan yerli kabilelerin yaptığı ilkel resimler ve Fransa'da Lascaux mağarasında keşfedilen 20000 yıllık duvar resimleri insanlık tarihinin görsel anlatımdaki ilk eserleri olarak kabul edilir. Daha sonra birtakım anlamlar yüklenen görsel anlatım biçimler – resimler ve şekiller – uzun bir tarihsel süreçten geçerek alfabedeki harflere dönüşmüştür. Bu da sözel iletişim biçiminin kaynağını oluşturur.

Sözel anlatım ve görsel anlatım hiçbir zaman birbirinin yerini tutmaz ya da birbiri yerine kullanılmaz. Örneğin bir şair duygu ve düşüncelerini anlatırken şiir yazar ve bu da sözel iletişim biçimidir. Ama ressamlar duygu ve düşüncelerini anlatmak için resim yapmaya ihtiyaç duyarlar ve bu da görsel anlatım biçimidir. Bu durumda ikisi arasında bir karşılaştırma yapmak ve birini diğerinden üstün görmek mantıklı değildir. Bu iki iletişim yönteminin tek başlarına yeterli olmadığı yerler vardır ve bu durumda her ikisi birden kullanılır. Bazen resimler metin ile desteklenir, bazen ise metinler resim ile.

Günümüzde bilgi aktarmak için sözel iletişim biçimi kadar görsel iletişim biçimi de kullanılır ve aynı derecede önemlidir. Çünkü, günümüz iletişim sürecinde yazılı materyallerin görsel öğelerle bütünleştirilmiş olarak kullanılması yaygınlaşmıştır ve bu durum bilgisayarların yaygınlaşması ile doğru bir orantı içindedir. Bundan 500 yıl önce bilgisayar gibi büyük bilgi birikimlerini kaydetmek konusunda güçlü bir aygıt olmadığı için Leonardo da Vinci farklı bir yol denemiştir. Bu yöntem farklı görüntüler olarak var olan sözcükleri yalnızca bir sayfada ifade edilebilecek biçimde desenlemekti. Günümüzde de yeni araştırmalar ve buluşlarla bilgi hacmi geometrik olarak artmaktadır. Bu bilgilerin aktarılmasında da çoğu zaman tek bir iletişim yöntemi yeterli olmamaktadır. İyi bir bilgi aktarımı için hem sözel hem de görsel öğeler birlikte kullanılmalı, her iki dil arasında iyi bir denge oluşturulmalı ve her iki düşünce biçimi arasında karşılıklı dayanışma güçlendirilmelidir.

Görsel eğitiminin genel eğitimde çok önemli olmasını dört farklı başlık içerisinde inceleyebiliriz.

- Görsel iletişimde kullanılan materyaller beynin sağ yarı küresine hitap eder ve bu kısım tam insan gelişimi için oldukça önemlidir. Böylece düşünme sürecinde beynin her iki yarı küresi de kullanılır ve bu durum da bütünsel düşünmeyi geliştirir.

- Ayrıca beynin sol yarı küresinde olan soyut düşünceleri somut bir şekilde yansıtarak canlı, inandırıcı, bilindik ve yoğun yapar. Bu sayede soyut düşüncelerin daha iyi anlaşılmasını sağlar.

- Aynı düşünceyi farklı şekillerde işleme yeteneğinin kazanılmasını sağlar.

- Sonuncu olarak da yaşadığımız doğal ve doğal olmayan çevreden etkilenmektense insanların kendi kararlarını verebilmesi için görsel materyalleri okuyup anlayabilmeleri gerekmektedir.

1.4.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.

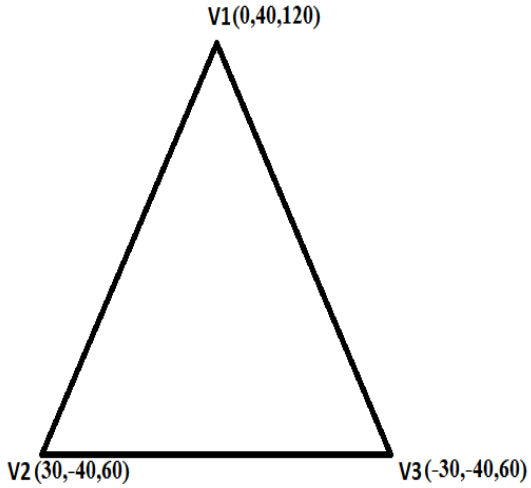
1)Normalizasyon:

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 3: 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.

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.

1.5.3D MODELLEME NEDİR?

Canlı veya cansız farketmeksizin herhangi bir varlığın görüntüsünü bilgisayar dünyasında üç boyutlu şekilde bütün yüzeyleri ile tasarlamak olarak kısaca özetleyebiliriz.

Gerçek dünyadaki bir nesneyi bilgisayarın anlayabileceği dünyaya geçişinin sağlanması, teknolojinin gelişmesi ile oldukça kolay ve kullanışlı hale gelmiştir. Örneğin bir nesneyi bilgisayar içerisinde çeşitli modelleme araçları ile modelledikten sonrasında 3d yazıcılar ile gerçek dünyaya aktarabiliyoruz. Çeşitli maketler, oyuncaklar ve ihtiyacımız olan

birçok şeyi modelleyebiliyor, bizce en güzel kullanım alanlarından olan sağlık sektöründe çeşitli protezler yapabiliyoruz. Tabii ki bu anlatılanlar gerçek dünyaya yansıyan olaylar.

Oyun sektöründe 3d modellemenin gerçek dünyaya etkisi oldukça sınırlı olmakla beraber çeşitli VR gözlükler gibi araçlar ile sağlanmaktadır. Oyun dünyasında, yaşadığımız dünyadaki görebildiğimiz neredeyse her şey gerçekmişçesine modellenmektedir. Öncelikle 2 boyutlu şekilde cisimler tasarlanır sonrasında bu taslaklar, hatalarından arındırılmış olarak son halini alıp 3d modelleri oluşturulur ve oyun içerisine eklenerek çeşitli ışıklandırılmalar ile gerçekçi görüntüler yakalanabilir.

Çoğu oyun motorunda dahili bir şekilde 3d modelleme arayüzü olmasına rağmen, çeşitli sınırlandırılmalar sebebi ile yetersiz kalabilmektedir. Bu yüzden harici modelleme uygulamalarına yönelmek kaçınılmaz olacaktır. Bu gibi uygulamalara Blender, Lightwave, Autocad 3d örnek olarak verilebilir.

1.6.OYUN MOTORU NEDİR?

Bireysel çalışmalar yapan insanların, genellikle de büyük oyun şirketlerinin, oyun geliştirmek amacıyla kullandığı ücretli veya ücretsiz olan programlara verilen isimdir. Bu oyun motoru dediğimiz araç, aslında bolca programlama dillerinin kütüphanelerinden oluşmuş bir yapıdır. Bu kütüphanelerin içinde fonksiyonlar ve sınıflar gibi veriler hali hazır bir şekilde bulunmaktadır.

Kişi oyun motorunu kullandığında, program üreticisi tarafından ve assets dediğimiz hazır 3d modelleri üreten kişiler tarafından yapılmış, çoğu zaman ücretli olan modelleri kullanarak istediği oyununu yapabilmektedir. Bu yapının kişilere veya kurumsal şirketlere sağladığı fayda;Oyun programcısı olan birisini bu verileri yeniden kodlama zahmetinden kurtarması yani daha az programlama bilgisi ile çok güzel işler çıkarabilmesini sağlamaktadır. Bunların yanında daha az karmaşıklık yaşanması ve zamandan tasarruf edilmesini sağlar.

C, C++, C#, Java ve C Programlama dili ile benzer tarihlerde çıkan Pascal programlama dili ile bile oyun motorları kodlamak mümkün olmuştur. Aşağıda oyun, oyun motoru ve bu oyun motorunun hangi dilde kodlandığı örnekleri ile verilmiştir.

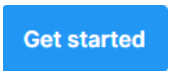
OYUN İSMİ	OYUN MOTORU	PROGRAMLAMA DİLİ
Star Trek 2	Genesis Device	Object Pascal
The Purge 2	Genesis3D Engine	C
Crysis	CryEngine 2	C++
Toontown Online	Panda 3D	C++
Fortnite	Unreal Engine 4	C++
League of Legends: Wild Rift	Unity	C#
Spacemen	jMonkeyEngine	Java

Tablo 1: OYUN İSMİ, MOTORU VE KULLANILAN PROGRAMLAMA DİLİNİN OLDUĞU BİR TABLO

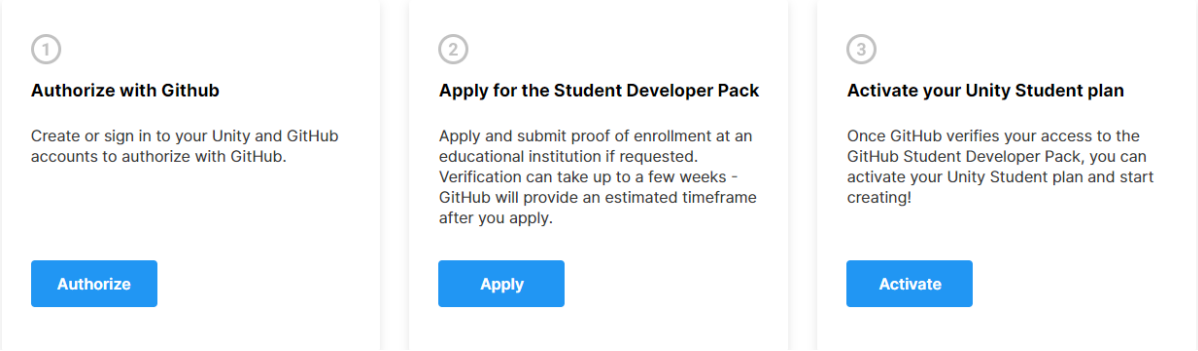
1.7.UNITY OYUN MOTORU VE KURULUMU

Unity şirketinin öncelikli hedefi olarak birçok platformda oyun tasarımı ve simülasyonlar yapmaktı. İlk yayınlanma tarihi olarak 8 Haziran 2005 yılında yalnızca Apple şirketinin bilgisayarlarında kullanmış olduğu OS X işletim sistemi ile uyumlu halde çıkmıştır. Sonrasında diğer işletim sistemleri ve konsollar için gelişme sürecine girmiştir. Yalnızca oyun için değil, çeşitli grafiksel mühendislik uygulamalarında, film ve otomotiv sektörlerinde kullanılmaktadır. 2006 yılında Apple şirketi Unity'yi en iyi Mac OS X Grafik kullanımı kategorisinde birinci olarak seçmiştir. Öğretici içeriklere sahip bir sitesinin olması ve özellikle ücretsiz bir oyun motoru olması, şirketin genç oyun geliştiricilerine verdiği önemi oldukça iyi bir şekilde göstermektedir.

1.7.1. UNITY KURULUM VE YAPILMASI GEREKEN AYARLAMALAR

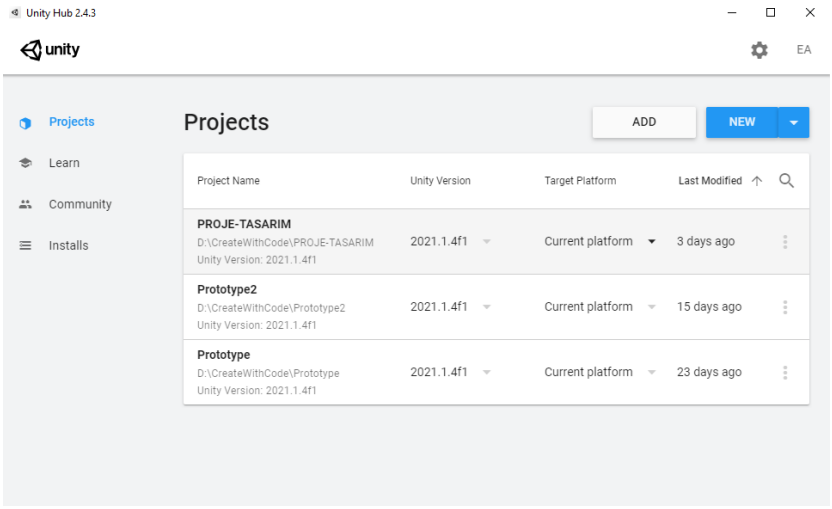
Unity'nin web sitesine girdikten sonrasında 

butonuna tıklanır. Bu buton bizi programı hangi amaçla kullanacağımıza dair seçim yapabileceğimiz bir web sayfasına yönlendirecek. Individual, Teams ve Industiral & Enterprise sekmelerinden Individual kısmı ücretsiz ve öğrenci üyeliği de mevcut. O yüzden Student kısmını seçiyoruz. Buradan üye olarak gerekli aşamaları tamamlayarak üyeliğimizi aktif ediyoruz. Gerekli aşamalar;



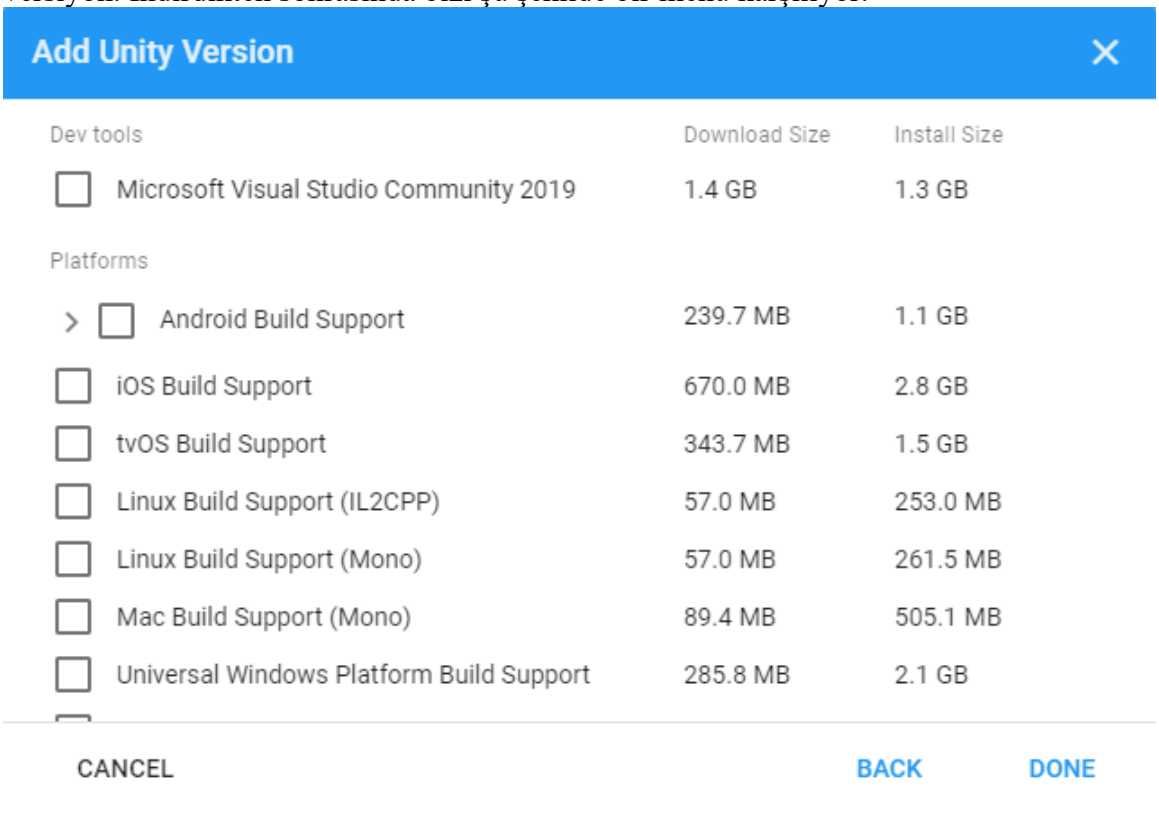
Şekil 4: Yazılımın indirilmesi.

Üyeliğimiz aktif olduğunda ise https://store.unity.com/download-nuo sayfasındaki yönlendirmeleri kullanarak indirmemizi başlatıyoruz. İndirilen dosya UnityHub diye adlandırılan başlangıç kullanıcıları için tavsiye edilen bir yazılımdır.
İndirme tamamlandığında UnityHubSetup dosyasını açıyoruz ve yüklemeye başlıyoruz. Yükleme sonlandığında programımızı açıyor ve aşağıdaki şekilde bir arayüzle karşılaşıyoruz.



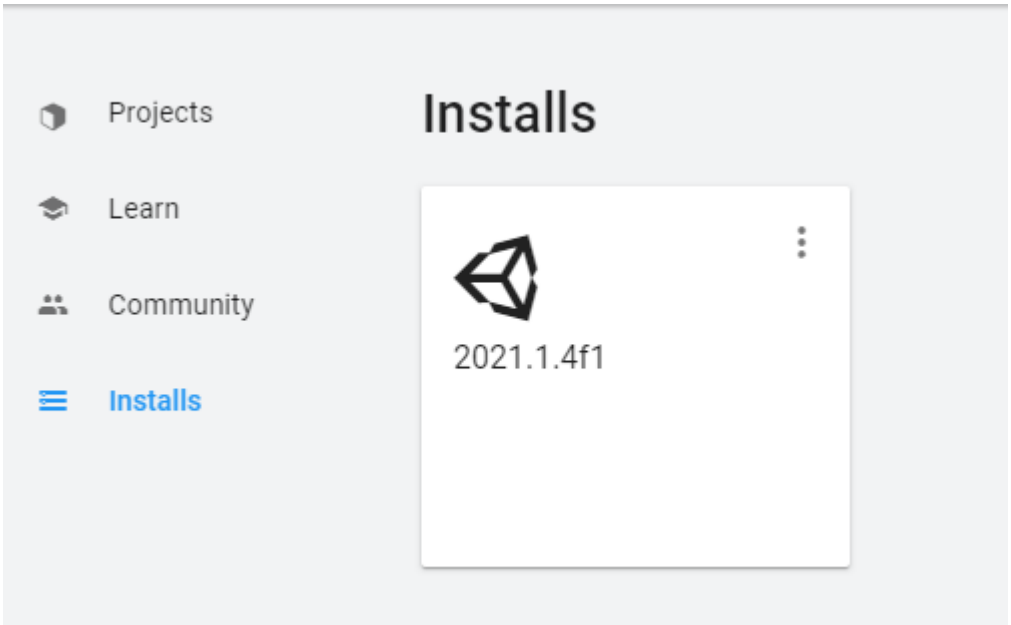
Şekil 5: Unity Hub ekranı.

Burada Projects menüsü altında projelerimiz, Learn kısmı altında Unity'nin öğretici hazır oyunları nasıl yapılıyor bunu görebiliyor, Community altında yeni başlayan bizler için sormak istediğimiz soruları vb. burada sorabiliyoruz. En son olarak Installs kısmında ise Unity'nin çeşitli versiyonlarını indirebileceğimiz bir bölüm bizleri karşılıyor. Buradan LTS olan bir versiyonu indiriyoruz çünkü Long Term Support yani uzun süreli destek içeren bir versiyon. İndirdikten sonrasında bizi şu şekilde bir menü karşılıyor.



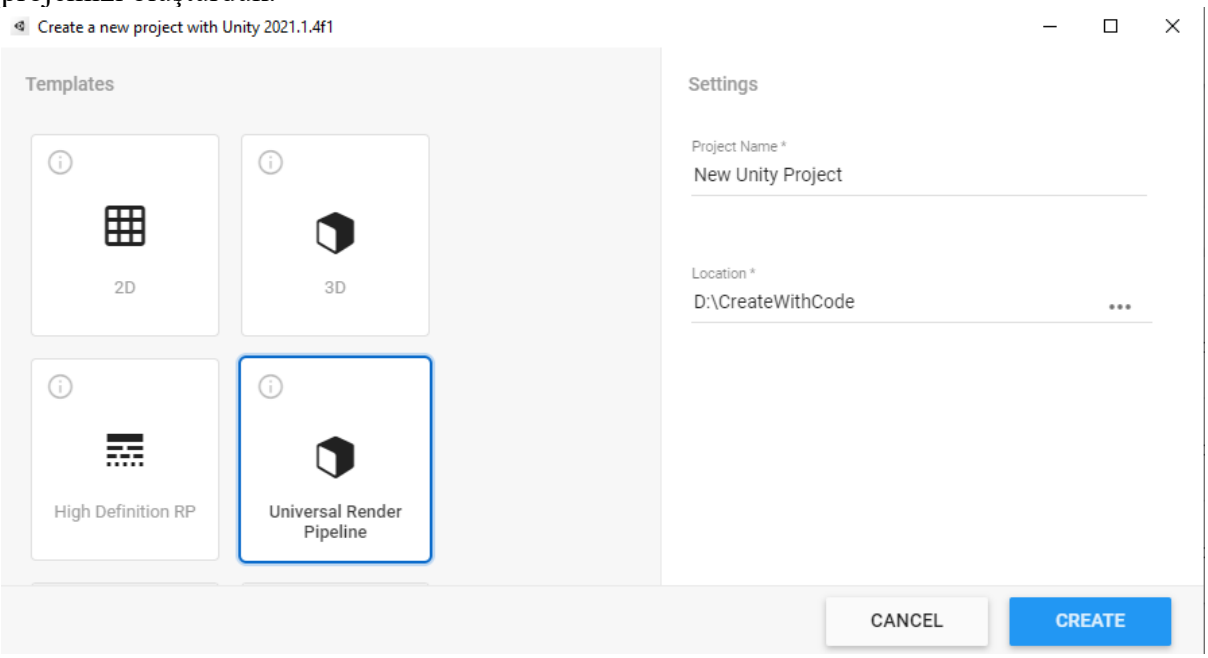
Şekil 6: Unity versiyon ve diğer yardımcı araçları seçme ve yükleme penceresi.

Biz burada Microsoft Visual Studio Community 2019'u önceden kurduğumuz için kurmaya gerek yok. C# kodlarımızı daha düzenli ve hızlı yazabilmek adına bu derleyiciyi kullanıyoruz.



Şekil 7: Kurulan Unity versiyonlarını gösteren bölüm.

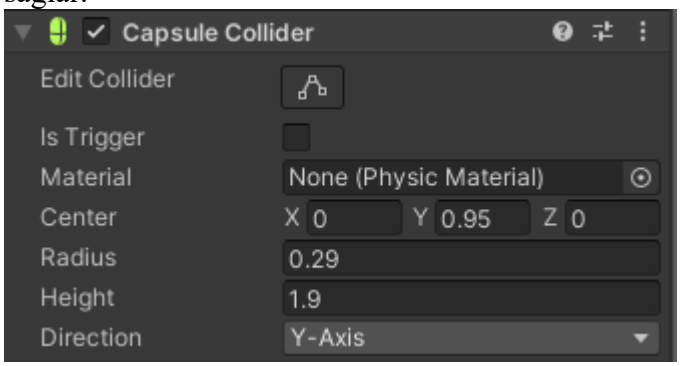
Son olarak Projects menüsünden 2D, 3D, veya High Definition RP, Universal Render Pipeline'dan ihtiyacımıza göre boş bir proje oluşturabilir, öğretici içeriklerden kendimizi geliştirebiliriz. Projemizde biz Universal Render Pipeline'ı kullanacağımız için bu altyapı ile projemizi oluşturduk.



Şekil 8: Proje altyapısı seçme bölümü.

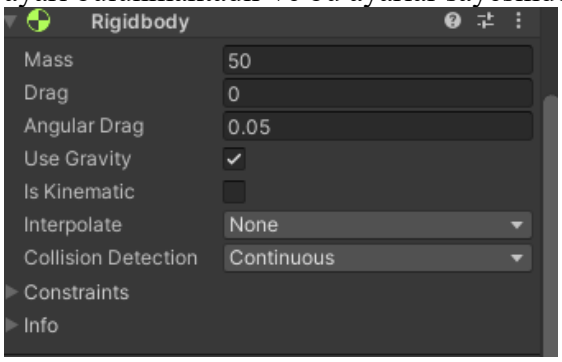
1.8.UNITY OYUN MOTORU İLE ALAKALI TERİMLER

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



Şekil 9: 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 10: 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.

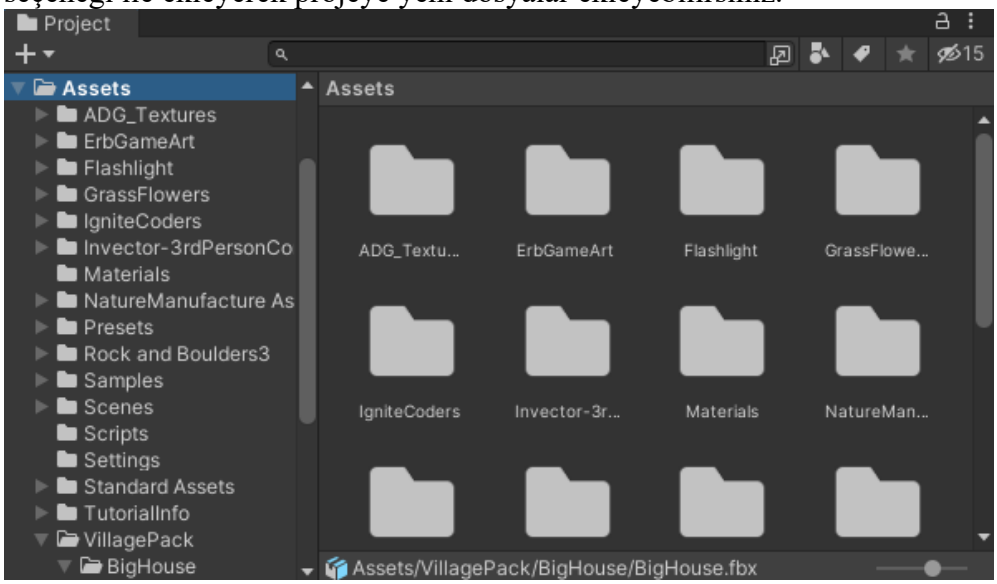
OnDestroy: Obje yok olmadan önceki son karenin tüm update fonksiyonları çalıştırıldıktan sonra gerçekleştirilir. Objenin yok olmasının sebebi destroy komutu kullanılması olduğu gibi başka bir sahneye geçme de olabilir.

OnApplicationQuit: Oyundan çıkılmadan önce gerçekleştirilen son komuttur.

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

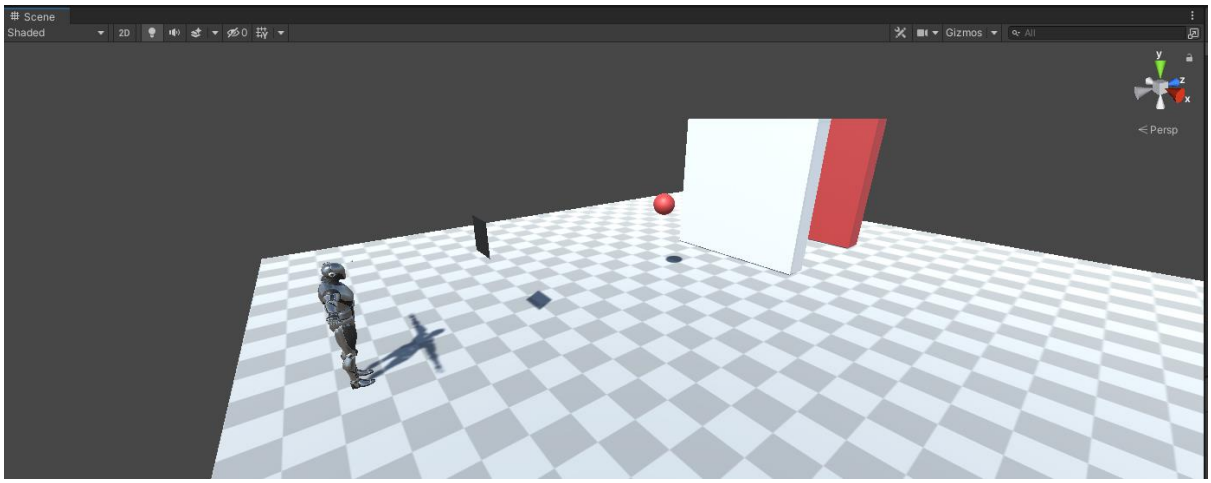
Varsayılan unity ekranında bulunanlar:

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



Şekil 11: Project penceresi (temsilidir).

Scene: O andaki pozisyonun sahnesi gösterilen panel.

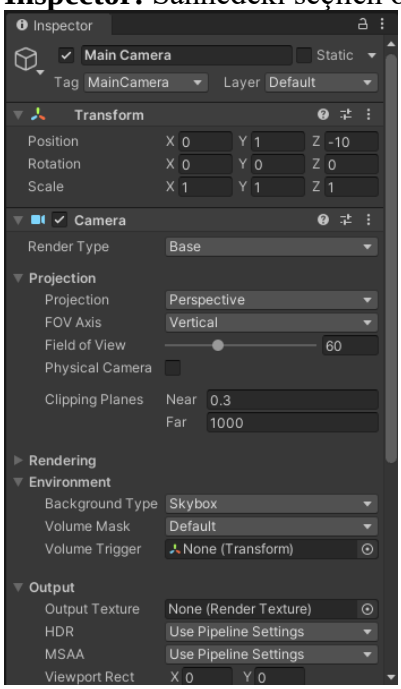


Şekil 12: Scene penceresi(temsilidir).

Hierarchy: Sahnede bulunan tüm objeler.

Şekil 13: Hierarchy penceresi (temsilidir).

Inspector: Sahnedeki seçilen objenin özelliklerini gösteren panel.



Şekil 14: 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 15: 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. Unity'de kendi class'ınızı oluşturduğunuzda onu serileştirmeniz gerekmektedir. Bu Unity'nin denetçide bakabileceği basit verilere dönüştürüleceği anlamına gelir. Bunu yaptığınızda, class'a sahip olduğunuzu görecek ve Inspector de görünecektir.

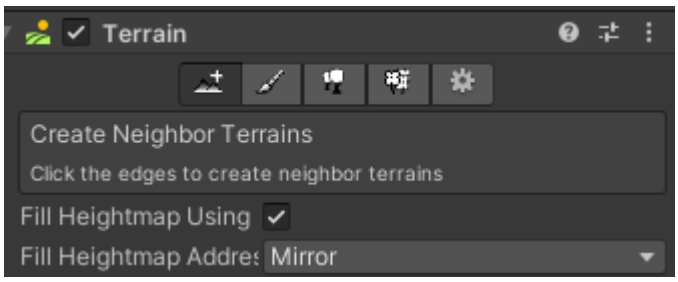
2.PROJE TASARIMI

2.1.ÇEVRE TASARIMI

Çevre tasarımında ilk önce yapmamız gereken bir terrain oluşturmaktı. Terrain oluşturduktan sonra dümdüz bir alan elde ettik. Bu alanda çeşitli dağların oluşacağı yüksek alanlar ve suların oluşturacağı alçak alanlar olması gerekiyordu. Biz de bunun için Terrain Tools denilen aracı kullanarak istediğimiz şekilde ve seviyelerde zemin oluşturabildik. Terrain Tools araçlarını tanırsak şu şekildedir;

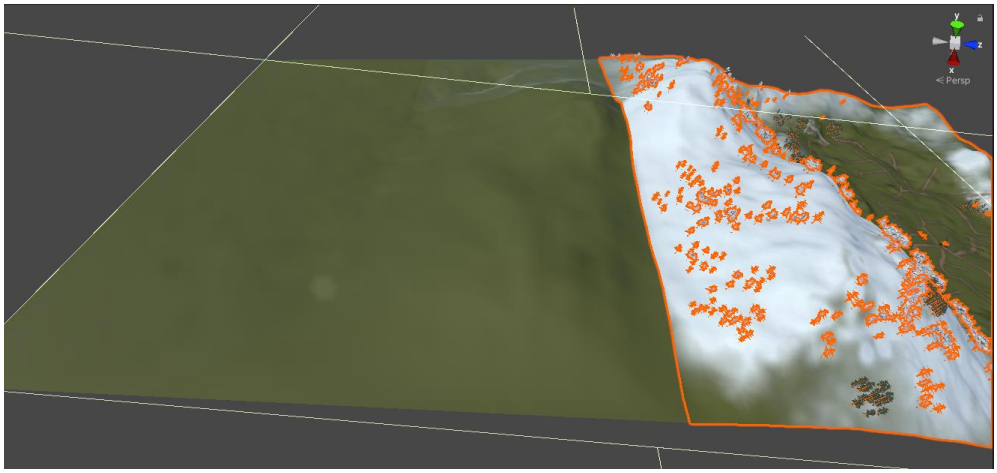
2.1.1.TERRAIN TOOLS

2.1.1.1.CREATE NEIGHBOR TERRAINS



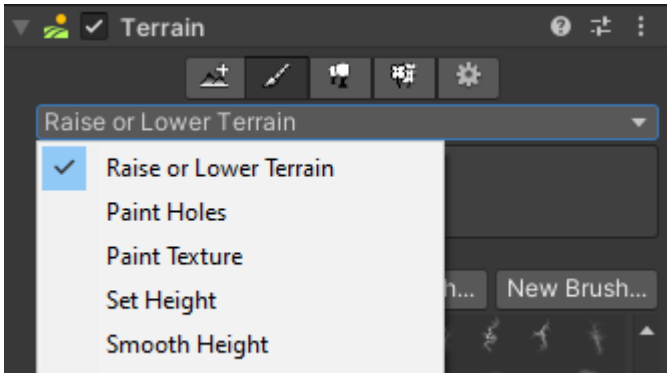
Bu araç hali hazırda mevcut olan terrainimizin yanına örneğin yukarıdaki mirror yani ayna görüntüsünü andıran bir başka terrain oluşturuyor. Kullanılması güzel bir özellik fakat bize 1000x1000 alan yeterli olduğu için bu özelliğe gerek duymadık.

Şekil 16: Create neighbor terrains bölümü.



Şekil 17: Create neighbor terrains kullanımı örneği.

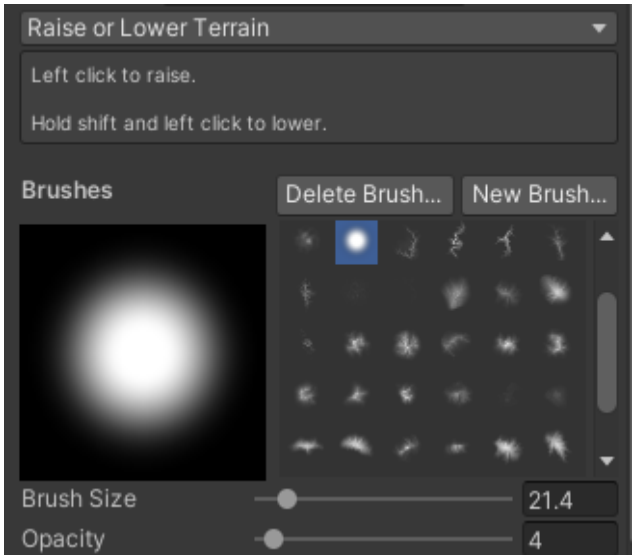
2.1.1.2.PAINT TERRAIN



Adından da anlaşılacağı üzere boyama olayımız var. Boyayarak yükseklik verebiliyor,diğer alanlardaki zemine göre otomatik bir şekilde yüksekliğe ayarlayabiliyoruz ya da boyayarak ağaç ve çimen ekebiliyoruz. Bu araç olmazsa olmazlarımızdan.

Şekil 18: Paint terrrain bölümü.

2.1.1.3.RAISE OR LOWER TERRAIN



Bu araç ile opacity ve fırça büyüklüğü ayarı yaparak istediğimiz şekilde yüksekliği veya alçaklığı, oluşturacağımız zemine kazandırabiliyoruz.

Şekil 19: Raise or lower terrain bölümü.

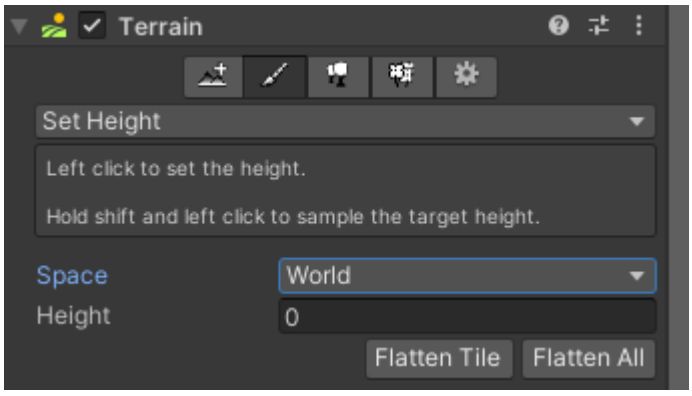
2.1.1.4.PAINT HOLES

Bu özellik haritamızda boşluk oluşturuyor ve bu yüzden, karakterimizin erişemeyeceği yerlerde kullanılması daha doğru olur.

2.1.1.5.PAINT TEXTURE

Bu özellik belirlediğimiz bir materyali veya katmanı kullanarak haritamızda bir bölümü kaplamamızı sağlıyor.

2.1.1.6.SET HEIGHT



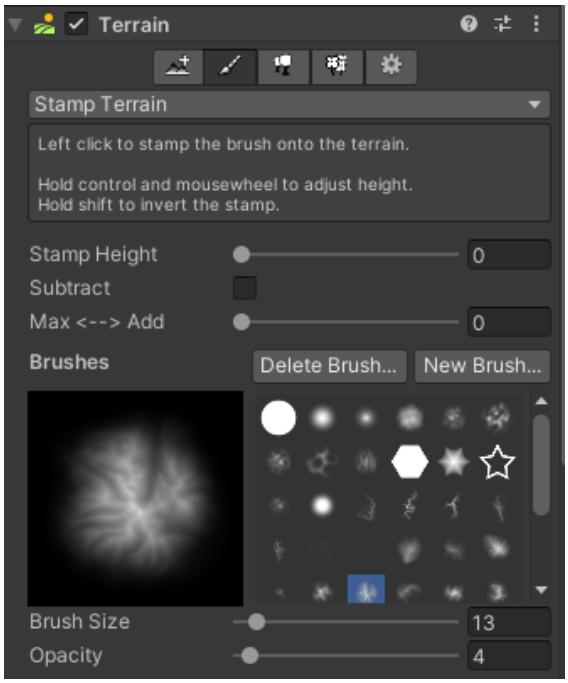
Haritada bir bölgeyi sabit bir yüksekliğe ayarlamamızı sağlar.

Şekil 20: Set height

2.1.1.7.SMOOTH HEIGHT

Bu araç kullanacağımız alan için en yakın yüzeye göre yükseklik ortalaması alır. Görünüşü güzelleştirir.

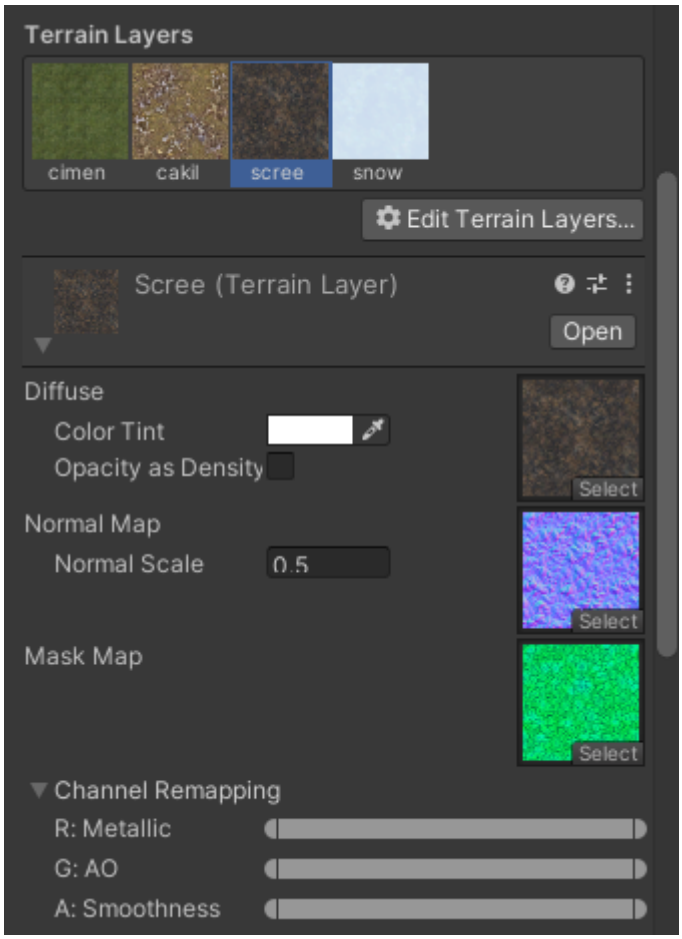
2.1.1.8.STAMP TERRAIN



Bu araç belirlediğimiz yüzeye fırça şekillerini verebilmemiz için kullanışlıdır. Raise or Lower Terrain aracına benzemektedir.

Şekil 21: Stamp terrain bölümü.

2.1.2.PAINT TEXTURE ARACINI KULLANARAK ZEMINI KAPLAMAK

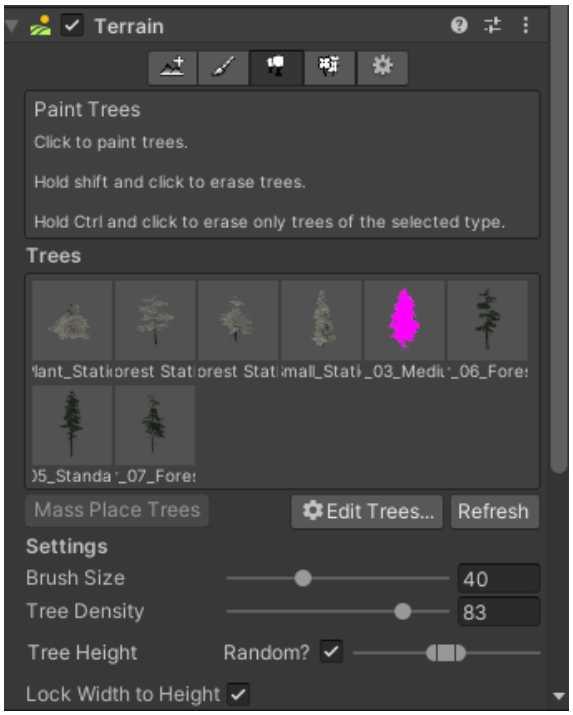


Öncelikle Terrain'i Hiearchy'den seçtik ve Paint Texture aracına Terrain Katmanı yani çimen,çakıl,kar gibi malzemeleri ekledik. İlk seçilen katman çimen ile her yer kaplandı. Bundan sonra sırayla yüksek olan yerlere kar, yol olan yerlere çakıl ve yolun kenarlarını da çamur ile kapladık. Örneğin çamur katmanını seçtiğimizde görüldüğü üzere diffuse, normal ve mask denilen texture'lar var. İlki yani Diffuse bileşen Temel dokuyu oluşturmakta iken, Normal bileşeni aydınlık hesabı yapmakta ve Mask ise dokular arasında oluşan bozulmalardaki maskelemeyi ifade eder. Net ve güzel bir görüntü alınabilmesi için gereklidir.

Diffuse bileşenin altındaki;
Color Tint: Renk Tonunu ayarlamamızı,
Opacity as Density: Dokunun göstermek istediğimiz alanlarını maskeleme yöntemi ile göstermemize olanak sağlayan özelliktir.

Şekil 22: Paint texture bölümü.

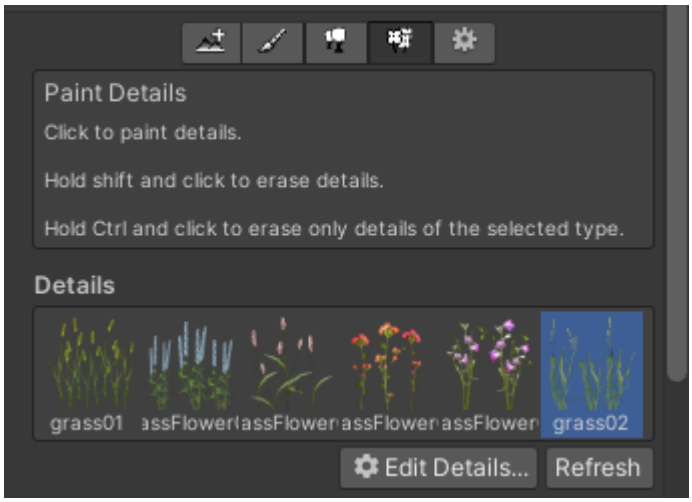
Normal Map bileşenin altındaki;
Normal Scale: Kullandığımız normal bileşenin yoğunluğunu göstermektedir.
Mask Map ve Channel Remapping altındaki;
R: Metalik katsayıyı
G: Ambient maskeleme katsayısını
A: Pürüzsüzlük katsayısını ifade etmektedir
Buna göre çeşitli göze hoş görünecek şekilde ayarlamalar yaparak zeminimizi kapladık.



Şekil 23: Paint trees aracı

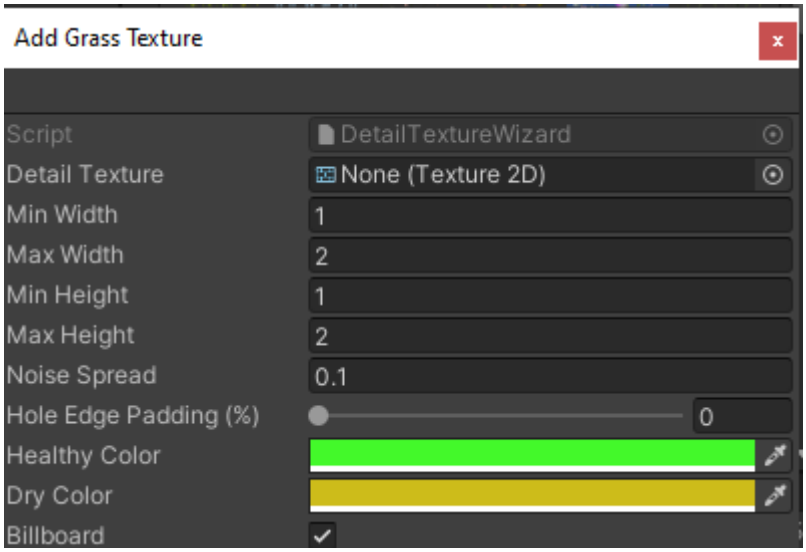
2.1.3.PAINT TREES ARACINI KULLANARAK HARİTAYI AĞAÇLANDIRMAK:

Öncelikle Edit Trees' aracına tıklanır. Buradan kullanacağımız ağaç seçilir ve add düğmesine tıklanarak ağacımız hazır hale gelir. Bunun gibi birden çok ağacı ekleyebilir, kullanıma hazır hale getirebiliriz.
Ağaçların özelliklerini ilgili ağacın Prefab'ına gidilerek değiştirebilir, Terrain Tools aracında da kullanılabilir hale getirebiliriz.
Ağaç seçildikten sonra Brush Size ile tek tıklamada ne kadar büyüklükte bir alanı ağaçlandıracağımız ayarlanır.
Tree Density ile ağaçlandıracağımız alandaki ağaç yoğunluğını belirleyebiliriz.
Tree Height ile rastgele oluşacak ağaçlarımızın değişken uzunluklarını ayarlayabiliriz.



Şekil 24: Paint details aracı.

Paint Details aracında ise haritamıza yerleştireceğimiz çimen ve çiçeklerin detaylarını ekleyebiliriz.
Edit Details kısmından istediğimiz bitkinin veya detay katacak nesnelerin texture veya meshini seçebiliriz. Texture daha az gerçekçi dururken Mesh gerçekçi bir yapıda durmaktadır.



Texture eklemeye çalıştığımızda bu menü ile karşılaşmaktayız. Rastgele oluşturulacak çiçek veya çimenlerimizin minimum ve maksimum yükseklik veya genişliklerini, renk yoğunlukları gibi değerleri ayarlayabilmekteyiz.
Mesh ekleme kısmında da aynı özellikler mevcuttur fakat render edilme açısından texturelara göre farklılıklar göstermektedir.

Şekil 25: Grass texture aracı.

Bu gibi ayarlamaları yaptıktan sonra boyama yöntemi ile zeminimize detayları ekleyebiliriz.

Şekil 26: Zeminin kaplanmış hali (temsili).

2.2.Işınların Objelerle Etkileşimi :

2.2.1.IŞINLARIN VE KAMERALARIN HAREKETİNİN SAĞLANMASI

Işınlarımız ve kameralarımız Şekil 27'deki kod sayesinde hareketini ve dağılımını sağlamaktadır.

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

public class Hareket : MonoBehaviour
{

    public float hiz;
    public float acilma;

    // Start is called before the first frame update
    void Start()
    {

    }

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

Şekil 27: Işınların ve kameraların hareketinin sağlanması.

2.2.2. IŞINLARIN VE KAMERALARIN KLONLANMASININ SAĞLANMASI

Şekil 28'deki kod sayesinde kameralarımız ve ışınlarımız kopyalanıp çoğalmaya başlıyor

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

public class RayCaster : MonoBehaviour
{
    public GameObject ray;
    public GameObject sphare;
    public int sizex;
    public int sizey;

    private GameObject rayI;

    // Start is called before the first frame update
    void Start()
    {
        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);
                        Instantiate(sphare, new Vector3(i * 0.01f, j * 0.01f, 0f), Quaternion.identity);
                        rayI.GetComponent<Rays>().num = new Vector2(i, j);
                        rayI = Instantiate(ray, new Vector3(-i * 0.01f, -j * 0.01f, 0f), Quaternion.identity);
                        Instantiate(sphare, new Vector3(-i * 0.01f, -j * 0.01f, 0f), Quaternion.identity);
                        rayI.GetComponent<Rays>().num = new Vector2(-i, -j);
                    }
                    if (j != 0)
                    {
                        rayI = Instantiate(ray, new Vector3(i * 0.01f, -j * 0.01f, 0f), Quaternion.identity);
                        Instantiate(sphare, new Vector3(i * 0.01f, -j * 0.01f, 0f), Quaternion.identity);
                        rayI.GetComponent<Rays>().num = new Vector2(i, -j);
                        rayI = Instantiate(ray, new Vector3(-i * 0.01f, j * 0.01f, 0f), Quaternion.identity);
                        Instantiate(sphare, new Vector3(-i * 0.01f, j * 0.01f, 0f), Quaternion.identity);
                        rayI.GetComponent<Rays>().num = new Vector2(-i, j);
                    }
                }
            }
        }
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Şekil 28: Işınların ve kameraların klonlanmasının sağlanması

2.2.3.IŞINLARIN YAPMASI GEREKEN HAREKETLERİ YAPMASINI SAĞLAMAK

Şekil 29'daki verilen kod sayesinde ışınlarımız "finish" tagine sahip olan objelere değdiği zaman hem kamera kayda başlıyor hem de temas eden kameralar hareket etmeyi bırakıyor. Bu sayede Görüntü düzleminden hemen sonraki cisimleri renderlayıp arkalarındaki cisimleri renderlamayı durdurmuş oluyoruz.

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

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

    // Start is called before the first frame update
    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
    void Update()
    {
        if (num.x==0&& num.y == 1)
        {
            dif = transform.position.y;
        }
        kamera.orthographicSize = 0.035f + dif / 10f;
    }

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

Şekil 29: Işınların yapması gereken hareketleri yapmasının sağlanması.

2.3.GEREKSİNİM ANALİZİ

Projenin gereksinimleri dört alt başlıkta incelenebilir.

Donanımsal gereksinimler:

Windows için;

Windows 7 (SP1+) ve Windows 10, 64-bit versionları, 64bit mimarili SSE2 destekli işlemci, DX10, DX11, ve DX12 destekli ekran kartı

macOs için;

High Sierra 10.13+, 64bit mimarili SSE2 destekli işlemci, metal destekli Intel veya AMD ekran kartı

Linux için;

Ubuntu 20.04, Ubuntu 18.04 ve CentOS 7, 64bit mimarili SSE2 destekli işlemci, OpenGL 3.2+ veya Vulkan destekli Nvidia veya Amd ekran kartları

Görsel ve işitsel gereksinimler

- Haritada saydamlık, gölgelendirme, kırılma, yansıma gibi ışık izleme türlerini destekleyecek materyaller ile ışık izleme prensibini daha iyi anlamak.
- Bu materyalleri gözlemleyebilecek ve kullanıcı tarafından yönlendirilebilecek bir karakter.

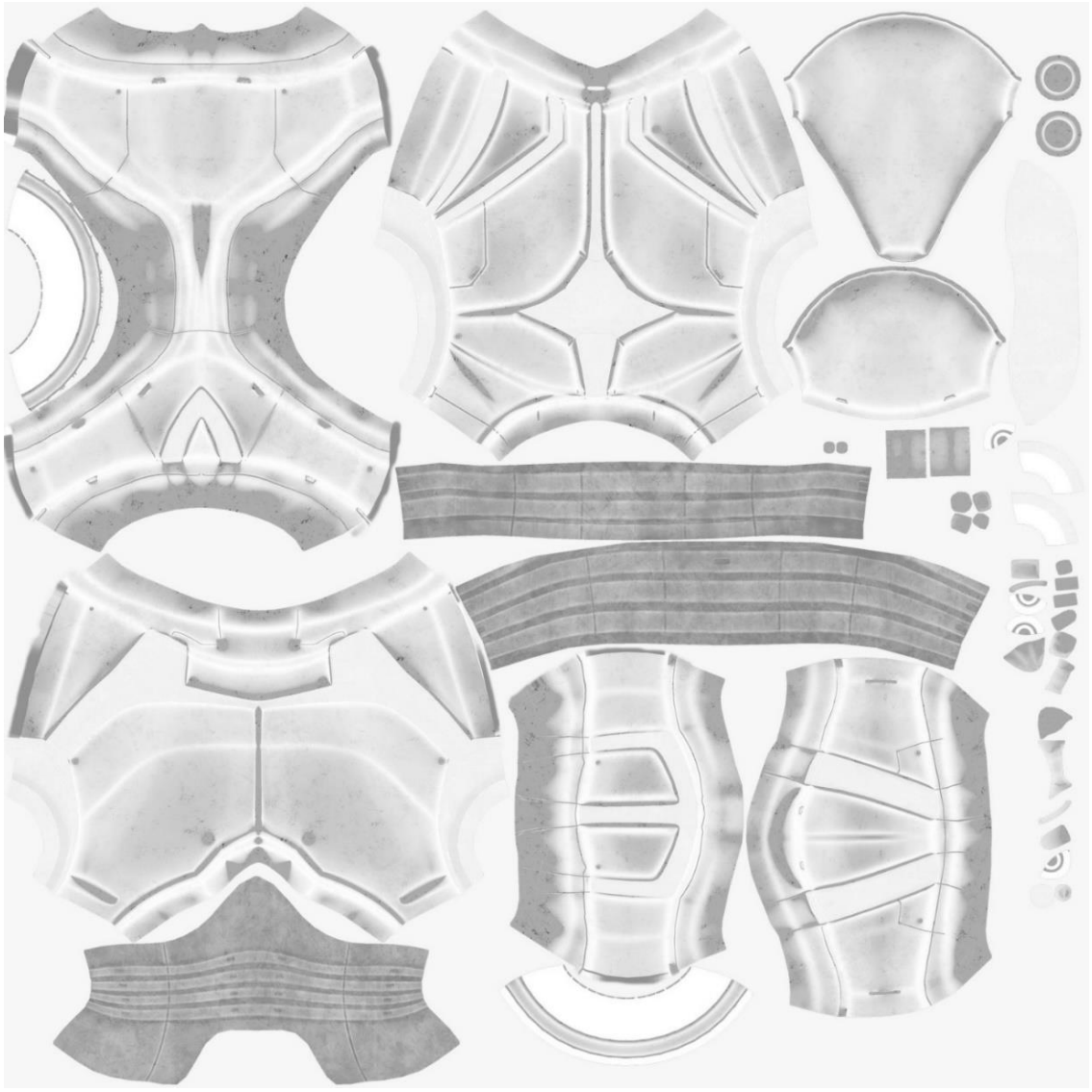
Arka plan gereksinimleri

- Karakterin hareketlerine göre değişen ve doğada gözlemlenebilir bir şekilde bulunan ışın izleme türleri.
- Kameranın ışın izleme yöntemlerini gözlemlemesi için uygun bir şekilde konumlandırılması.

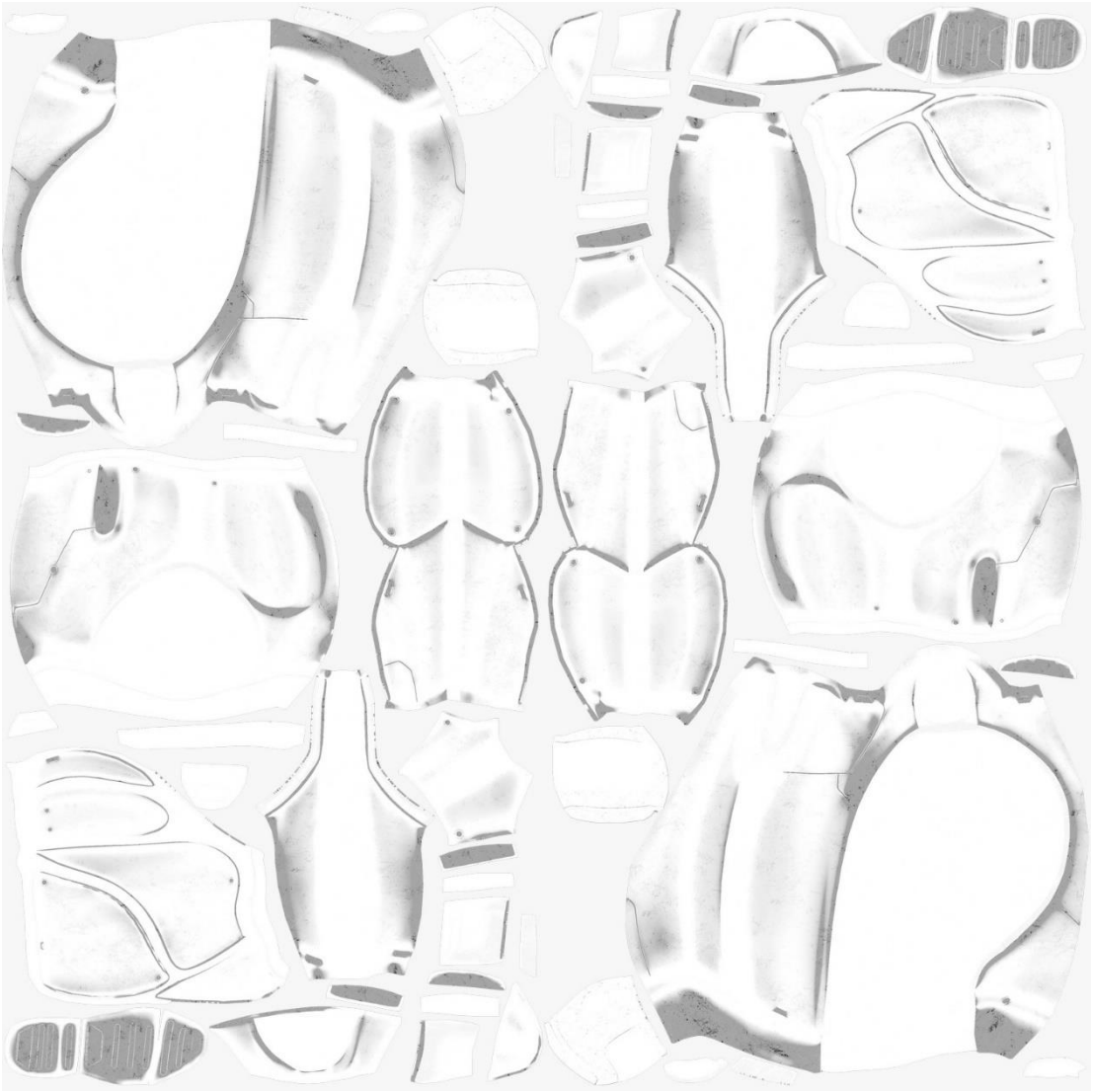
Proje yapımında ve kullanımındaki dış gereksinimler

- Sürüme uygun işletim sistemi kullanmak. (Windows 10)
- C# ve UnityEngine kütüphanesine hakim, Unity Game Engine kullanmasını bilen programcılar, ışın izleme teknolojisine uygun görsel tasarımlar için dizaynırlar ve senkronize çalışan geliştirici ekip

2.4.MİMARİ TASARIM



Şekil 30: Karakterimizin ön ve arka yüzlerinin texture'leri

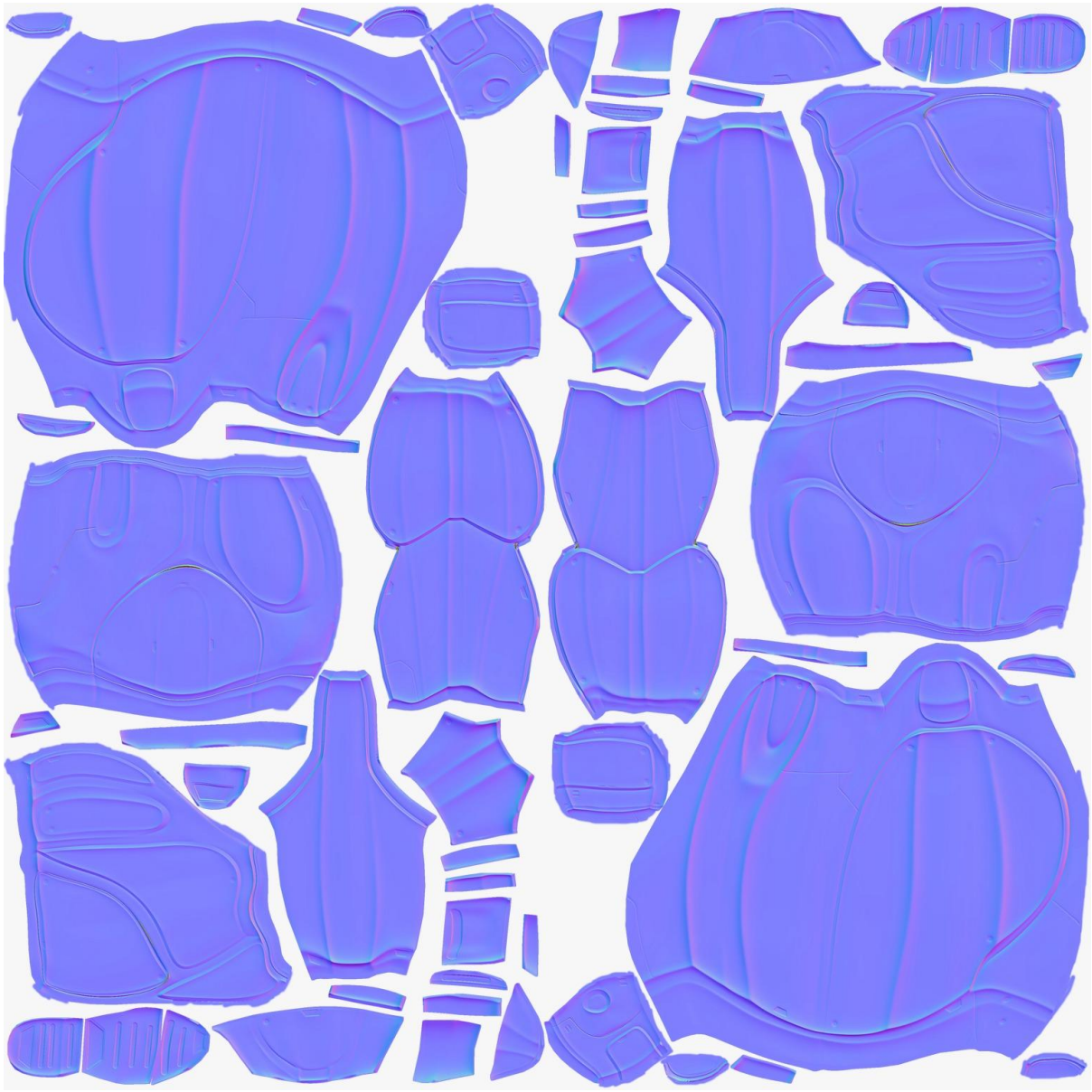


Şekil 31:Karakterimizin Yan Yüzlerinin texture'leri

Şekil 32:Karakterin Birleşim Yerlerinin texture'leri-1

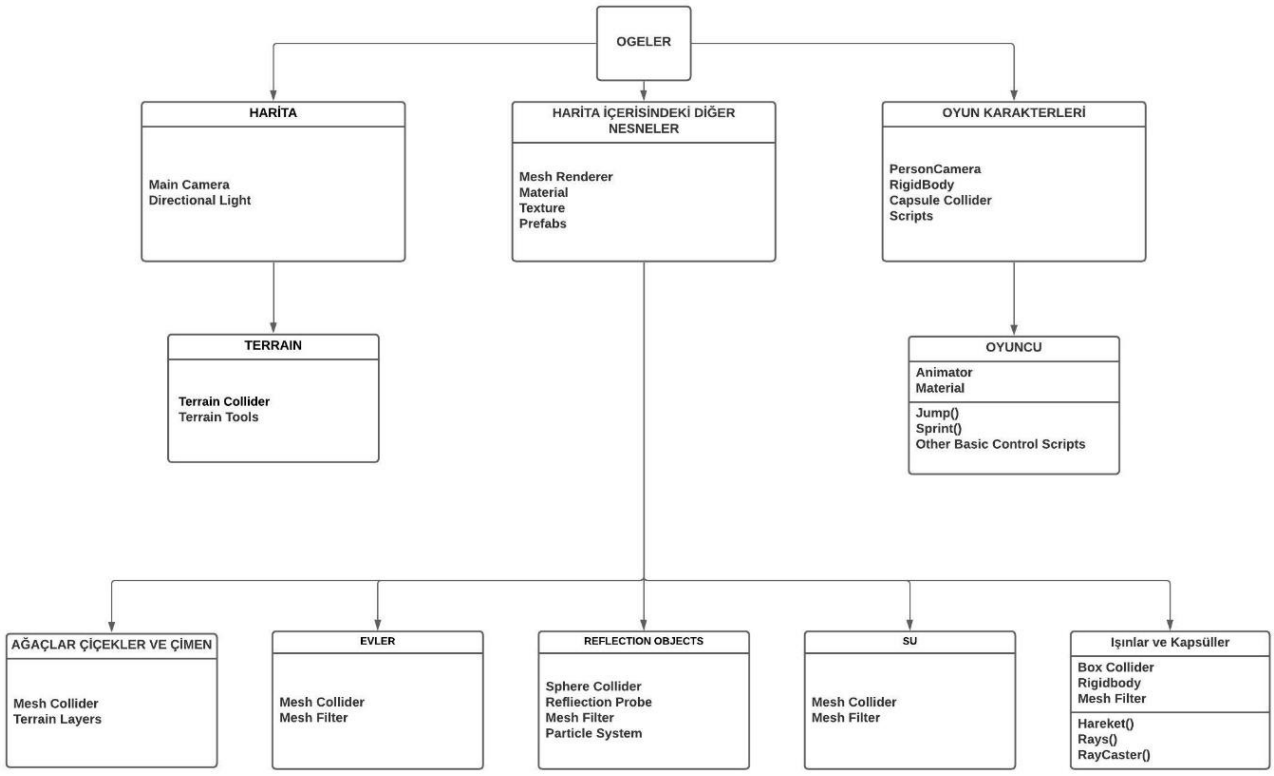
Şekil 33:Karakterin birleşim yerlerinin texture'leri-2

Şekil 34:Karakterimizin ön ve arka yüzlerinin normalleri



Şekil 35:Karakterimizin yan yüzlerinin normalleri

2.5.UML NESNE MODELİ



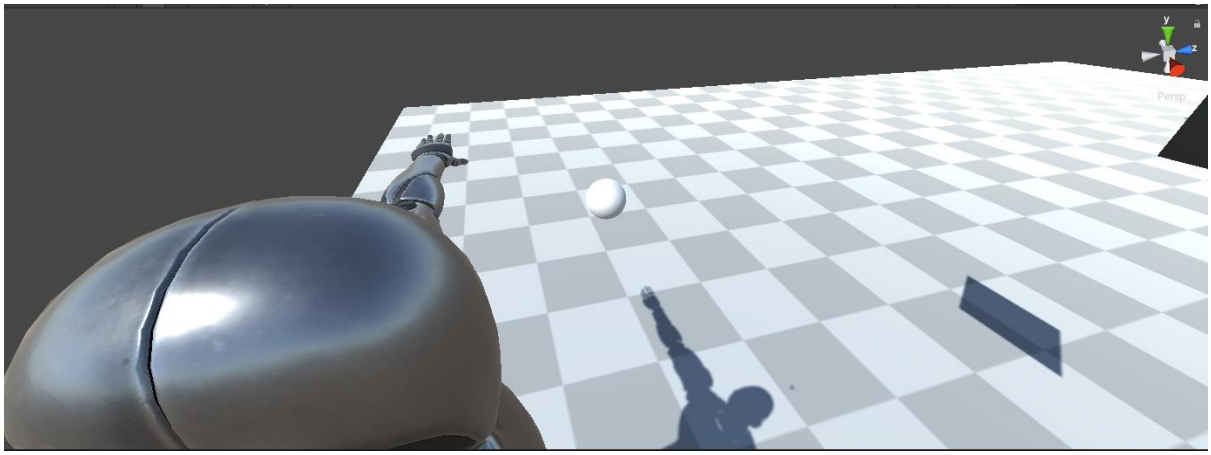
2.6.YAPILAN ÇALIŞMALAR

2.6.1.YAPILAN DEMO OYUNUNU TASARLAMA ÇALIŞMASI

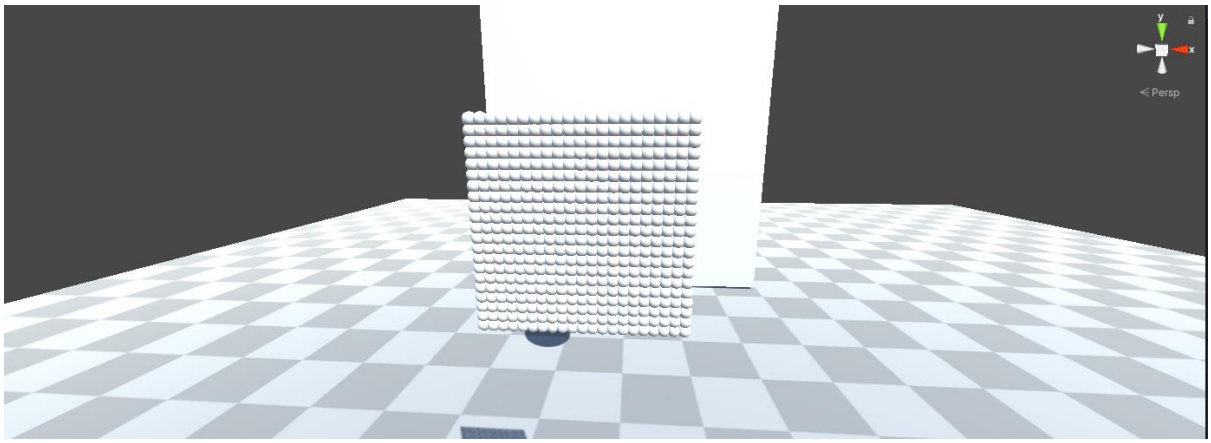
Işınları nasıl yolladık?

Işınları ilk olarak küre gibi temsili herhangi bir cisim olarak düşündük ve içerisine kamerayı yerleştirdikten sonra, kameranın özelliklerinden direkt olarak önündeki görüntüyü çekmesini sağlayan özelliği aktif ettik. Kameranın aktif edildiğinde, önündeki görüntüyü çekmesini ve kamera hareket ederken "Finish" Tag'ine sahip olan bir nesneye çarptığında hareket etmeyi durdurmasını Şekil 29'da yazılan script ile sağladık. Bu sayede ilk kontaktın sağlanmadığı herhangi bir cismin görüntüsünün görüntü düzlemine bastırılmasını engellemiş oluruz. Kameraların ve ışınların hareketini de Şekil 27'de yazdığımız script ile sağladık.

Normalde kamera ve temsili ışınımız birer adet, bu bir piksel anlamını taşımakta. Bu yüzden klonlama yapılmalı ve piksel sayımızı arttırmalıyız bu işlemi de Şekil 28'de yazdığımız script ile yaptık.



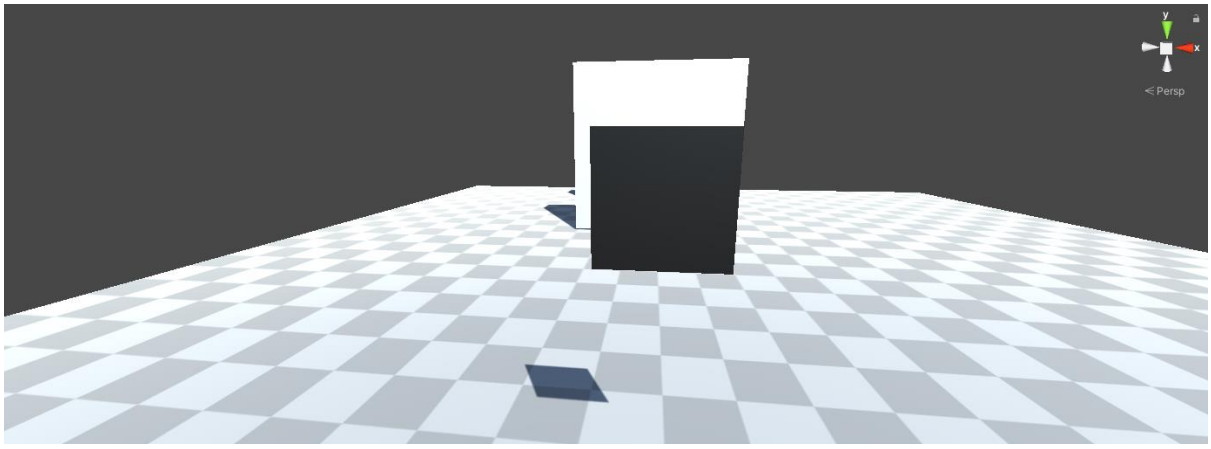
Şekil 36: Klonlanacak ışınımız.



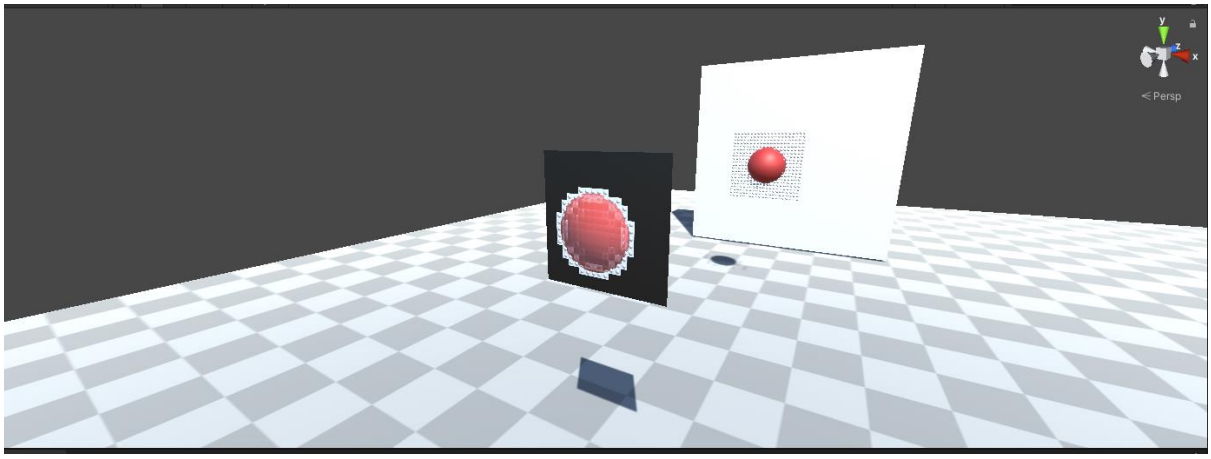
Şekil 37: Klonlanmış ışınlarımız ve kameralarımız.

Görüntüyü görüntü düzlemine nasıl uygun şekilde bastırdık?

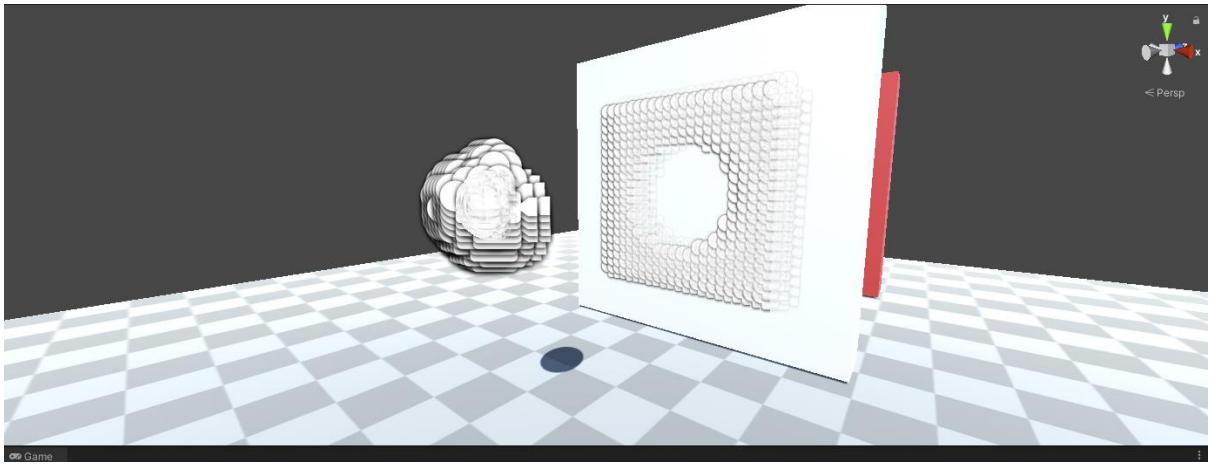
Yeni bir Render Texture ve Material oluşturduk. Sonra yeni oluşturduğumuz materyalin "Albedo" kısmına yine oluşturduğumuz Render Texture'u tanımladık. En son olarak kameranın "Target Texture'na oluşturduğumuz Render texture'unu tanımlıyoruz. Bu işlemleri klon mantığı kullandığımız için "Ray" adlı prefab'ın üzerindeki kamerada gerçekledik. Şekil 28'de yazdığımız script ile de görüntü düzlemine piksel piksel her bir kameradan aldığımız görüntüleri görüntü düzlemine bastırabildik.



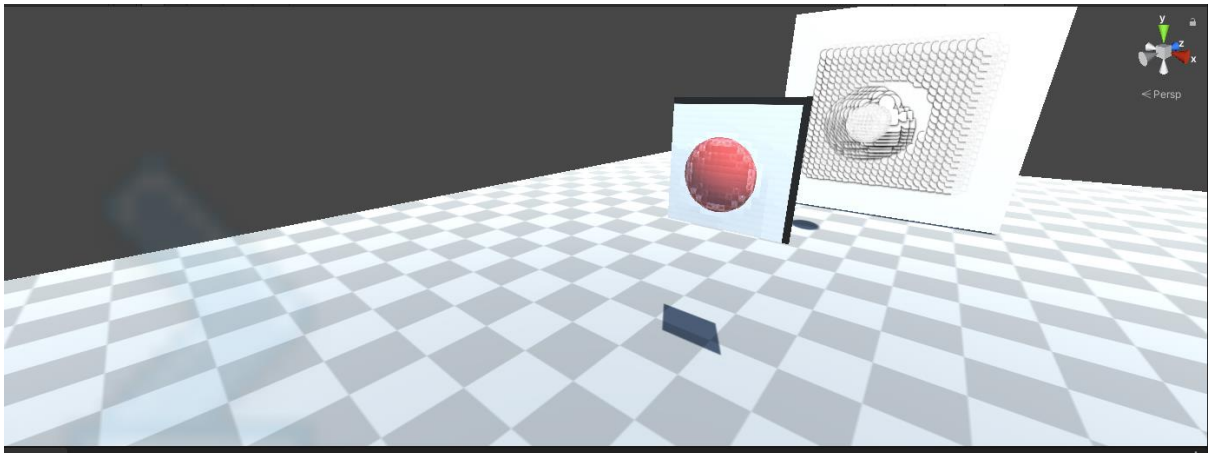
Şekil 38: Görüntü düzlemi.



Şekil 39: "Finish" Tagına gelmeden önce.



Şekil 40: Finish Tagına geldikten sonraki kameraların durma olayı.



Şekil 41: Görüntü düzlemindeki son görüntü.

3.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 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.

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.