**YAZILIM STAJ DEFTERİ**



**T.C.**

**KAHRAMANMARAŞ SÜTÇÜ İMAM ÜNİVERSİTESİ**

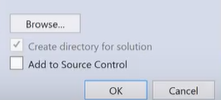
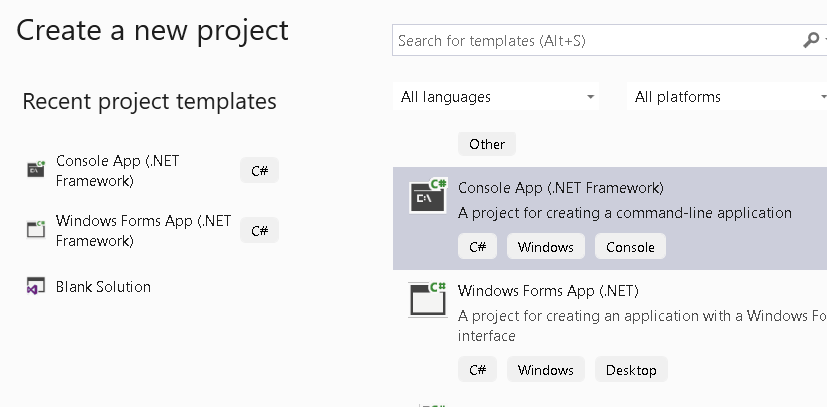
**MÜHENDİSLİK VE MİMARLIK FAKÜLTESİ**

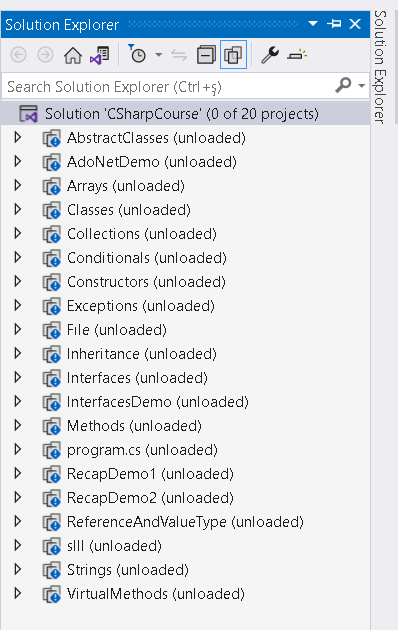
**BİLGİSAYAR MÜHENDİSLİĞİ**

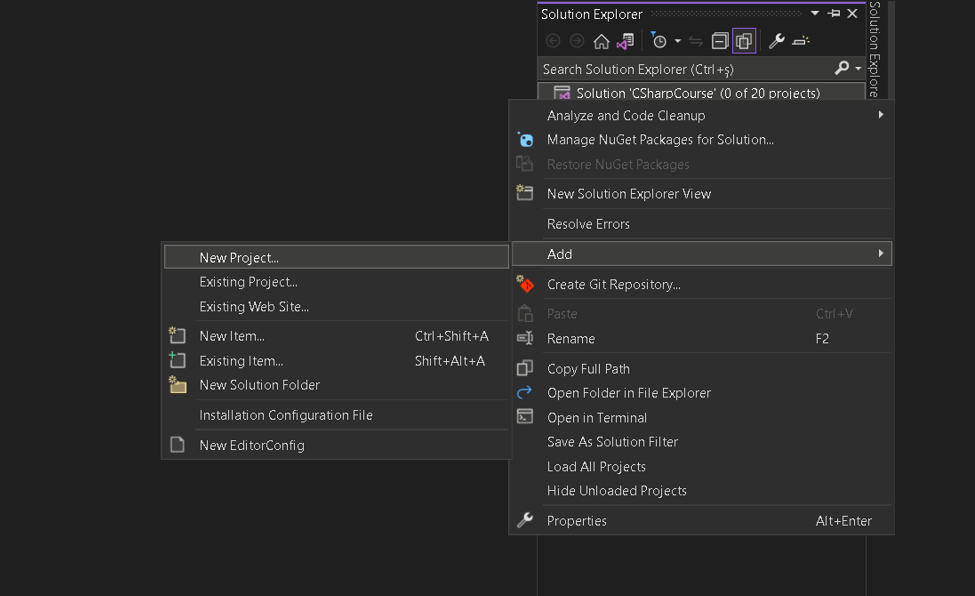
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STAJ YAPAN ÖĞRENCİNİN** | | | | FOTOĞRAF |
| **ADI SOYADI:** | | MERVE NİLGÜN YILMAZ | |
| **ÖĞRENCİ NUMARASI:** | | 19\*\*\*\*\*\*\*\* | |
| **YARIYILI:** | | Bahar | |
| **STAJ TÜRÜ:** | | Zorunlu Yazılım Stajı | |  |
| **STAJIN YAPILDIĞI ÖĞRETİM YILI:** | |  | |
| **STAJ BİLGİLERİ** | | | | |
| İŞYERİ ADI | Helikanon Yazılım Ve Bilişim Teknolojileri | | | |
| ADRESİ / TELEFON NUMARASI |  | | | |
| YAPILAN ÇALIŞMANIN NİTELİĞİ |  | | | |
| STAJ BAŞLANGIÇ TARİHİ |  | | | |
| STAJ BİTİŞ TARİHİ |  | | | |
| STAJ SÜRESİ (GÜN) |  | | | |
| **STAJ YERİ YETKİLİSİ** | | | **İMZASI** | |
|  | | |  | |

Stajımı hangi programlama dili üzerine yapmam gerektiği konusunda şirket çalışanlarından fikir alındı. Staj Hocam C# ve Unity ağırlıklı çalıştıklarını ve benim de bu alana yönelik çalışmalar yapmamı önermesiyle c# ve Unity ile karar kılındı. Unity’ e başlamadan önce temel c# bilgisinin olmasının önemi konuşuldu. Birinci gün genel programlama bilgimi ölçmek anlamında bir dizi sorular soruldu. Daha sonra C# programlama dilinin ne olduğu, nerelerde kullanıldığı ve diğer programlama dillerine göre avantaj ve dezavantajları konusunda bilgiler alındı. Araştırma ve kurulum yaparak ilk günümü tamamladım . Araştırma Sonucunda edindiğim bilgiler kısaca şöyledir;

Orta seviyeli bir programlama dili olan C# Microsoft tarafından geliştirilmiştir. Günlük kullanım diline de makine diline aynı yakınlıktadır. İlk Alfa versiyonu 2000'li yıllarda ortaya çıkmıştır. Java ve C++ ın güçlü yönlerinden esinlenmiştir. Tasarlanırken kullanım amacına uygun olması ve kolay adapte olunması göz önünde bulundurulmuştur. Nesne yönelimli bir dildir. Dolayısıyla C++ veya C kullanan insanlar kolayca adapte olabilmektedir. Genel olarak web site uygulamaları, masaüstü uygulamaları ve oyun oluşturma da yaygın olarak kullanılan bir programlama dilidir. Genellikle .NET platformunda ve açık kaynaklı olarak profesyonel ve dinamik web site geliştirmesinde kullanılmaktadır. Çok işlevli bir web site alt yapısı oluşturmak için C# kullanılmakta. Nesne odaklı bir programlama dili olduğundan kolay ve hızlı kod yazıp bakımını yapmak da kolaydır. Unity en yaygın olarak kullanılan oyun geliştirici motorlarından biridir. Dünyada pazar payının 3’te birine sahip Unity oyun motoru ile C#’ın çok iyi bir uyum sağladığı ve Xamarin gibi teknolojiler sayesinde mobil ve diğer  konsollarda kullanılabildiği öğrenildi. C# Microsoft tarafından Windows uygulamalarında ve .Net platformunda kullanılması için oluşturulmuş bir programlama dilidir. Windows masaüstü uygulamalarında ve .NET çerçevesine sahip bir web site alt yapısında c# yaygın bir kullanıma sahiptir. C# en iyi oyun oluşturma programlama dilleri arasında yer almakta ve Unity oyun motorunda bağımlılık yapan oyunların oluşturulmasında yaygın olarak kullanılmakta. Gün içinde Visual Studio kurulum yapıldı. Kurulum yapılırken Visual Studio Code ortamında Javascript geliştirme gibi ya da mobil geliştirme gibi işlemler yapılırken Visual Studio ile tamamen .Net dünyasına odaklanmış bir ide olduğundan bahsedildi. ide ile editör farkından kısaca bahsedildi örneğin metin dosyası txt bir editördür yazı yazılır fontu büyür Visual Studio ileri seviyede destek verir hata yakalama test teknikleri gibi her şeyi kendi içinde barındırıyor. Visual Studio Code ile de bir yerden sonra ide olarak kullanılabiliyor olsa da ben Visual Studio ile devam edeceğim.

 Proje oluşurken alt kısımda karşımıza çıkan Add to sourch Control css gibi sublogin gibi ortamlara proje atmak için kullanılıyor. Takım çalışmasının olduğu kodların paylaşılması ve tek bir yerde yönetilmesi amacıyla kullanıldığından da bahsedildi

İlk proje solution oluşturma => View -> solution Explorer



Unity ile mobil oyun yapabilmek için C# bilmemiz gerekiyor. Unity oyun motorunun kullandığı kendi fonksiyonları ve sınıfları yer alıyor ama bunları anlamadan önce temel bir C# eğitimi almam gerektiğinden dolayı C# çalışmalarıma devam ettim. Yan kısımda öğrenmek için çalıştığım C# projelerimin ekran resmi bulunmaktadır.

**Unity Kurulumu ve Unity Hub**

Unity indirme sayfasında karşıma2 hatta 3 adet seçenek çıkıyordu, bunlardan ilki Unity yi direkt olarak site üzerinden indirmemizi ve kurmamızı ikincisi ise Unity Hub indirmemizi ve daha sonra Unity Hub üzerinden Unity sürümlerini kurmamızı sağlıyordu. Üçüncü seçenek ise Unity nin daha kararlı olmayan sürümlerini indirebilmemizi sağlıyordu. Unity Hub ile istediğimiz sürümü indirebiliyor, güncelleyebiliyor ve istediğimiz modülleri otomatik olarak Unity Hub üzerinden ekleyebildiğimiz için ben Unity Hub üzerinden Unity nin güncel sürümünü indirerek indirme ve kurulum işlemimi tamamladım. Stajım boyunca kullandığım Unity sürümlerim bunlardır Unity Hub üzerinden kurulan modüller Unity için seçilmiş ve desteklenen sürümleri oldukları için herhangi bir hata ile karşılaşmadım. C# kodlarını yazabilmem için bize bir IDE yani editör programı gerekiyor Unity’i destekleyen programlardan biri olan Visual Studio ile çalışmalarıma devam ettim.

(**Fizik İşlemleri**) Unity e fizik işlemleri ile başlandı. Unity oyun motorunda fizik işlemleri oyunun temel yapılarından biri durumundadır. Çünkü her oyunda fizik işlemleri olmazsa olmazdır. Unity’ nin en önemli konularından birisidir. Birden fazla materyali bulunmaktadır. Rigidbody bileşeni fizik işlemlerinin gerçekleşmesini sağlamaktadır. Ayrıca collider, collision ve Trigger da ana bileşenlerindendir. Unity’nin çalışma mekaniği: Unity sahnelerle çalışıyor sahnenin içinde objeler bileşenlerindendir. Unity’nin çalışma mekaniği: Unity sahnelerle çalışıyor sahnenin içinde objeler bulunuyor. Hierarchy kısmından yeni bir obje ekleyebiliyoruz. Fizik sistemi componentlerden oluşuyor. Sahnemizin içinde gameobjelerimiz , gameobjelerimizde de componentler var.

* **Rigidbody** : Gerçekte olduğu gibi fiziksel hareketlerin taklit edilmesini sağlar. Herhangi bir objeyi yere düşürmek fizik eklemek için kullanılır.
* **Mass** : Nesnenin kütlesi (varsayılan olarak kilogram cinsinden).
* **Drag** : Nesnenin hareket ederken hava direncinin ne kadar etki ettiği anlamına gelir.
* **Angular Drag**: Nesnenin dönüşte hava direncinin ne kadar etki ettiği anlamına gelir.
* **Use Gravity**: Etkinleştirilirse, nesne yer çekiminden etkilenir.
* **Is Kinematic**: Etkinleştirilirse, nesne fizik motoru tarafından çalıştırılmaz ve yalnızca Dönüşümü ile değiştirilebilir. Bu, hareketli platformlar için veya bir HingeJoint takılı bir
* **Discrete**: Diğer çarpışanlar, ona karşı çarpışmayı test ederken ayrık çarpışma algılaması kullanacaktır
* **Continuous**: Dinamik Çarpıştırıcılara (Rigidbody ile) ve süpürme tabanlı olarak ayrık çarpışma algılamasını kullanılır sürekli çarpışma tespiti Statik Çarpıştırıcılara karşı (Rigidbody olmadan). Sürekli Dinamik olarak ayarlanmış katı cisimler, bu katı cisme karşı çarpışmayı test ederken sürekli çarpışma algılamasını kullanacaktır.
* **Continuous Speculative**: spekülatif sürekli çarpışma tespitidir.
* **Freeze Rotation**: Rigidbody’nin X, Y ve Z eksenlerinde seçici olarak hareket etmesini durdurur.
* **Collider**: Bir nesnenin başka bir nesne ile çarpışıp çarpışmayacağını Collider (Çarpıştırıcı) belirler. Nesnenin boyut ve şekline göre çeşitli Collider türleri vardır.
* **Collision**: Sert çarpışmadır.
* **Trigger**: Çarpışmaları algılamak için kullanılır.
* **OnCollisionEnter**: OnCollisionEnter bir objenin çarpma durumunda 1 kere çalışır.
* **OnCollisionStay**: OnCollisionStay bir objenin çarpma durumu sürdüğü sürece çalışır.
* **OnCollisionExit**: OnCollisionExit bir objenin çarpma durumu bittiğinde 1 kere çalışır.
* **OnTriggerEnter**: OnTriggerEnter bir objenin çarpma durumunda 1 kere çalışır.
* **OnTriggerStay**: OnTriggerStay bir objenin çarpma durumu sürdüğü sürece çalışır.
* **OnTriggerExit**: OnTriggerExit bir objenin çarpma durumu bittiğinde 1 kere çalışır.

(**VEKTÖR İŞLEMLERİ**)Vector3 Unity programı üzerinde kullanılabilecek, içerisinde 3 boyuttaki eksenlere denk gelen x,y,z adlı 3 float tipli değişken bulunduran, yani vektörel bir büyüklüğü belirtmek için kullanılan değişken tipidir.

* **back:** Vector3(0,0,-1) yazmanın kısayolu
* **down:** Vector3(0, -1, 0) yazmanın kısayolu

Vector3 sınıfının “**left**” ve “**right**” özelliği, objeyi X ekseninde pozitif ve negatif alana doğru hareket ettirir.

Vector3 sınıfının “down” ve “up” özelliği, objeyi Y ekseninde pozitif ve negatif alana doğru hareket ettirir.

Vector3 sınıfının hazır özelliklerinden olan “**one**” ve “**zero**” için hazır tanımlı koordinat bilgilerini girer.

**Normalized Özelliği**: Normalized bir objenin pozisyon bilgilerini 1 uzunluğunda döndürür. X,Y ve Z koordinatlarıma yazılan değer kaç olursa olsun -1 ve 1 arasında bir sayı dönüşü yapar.

“**ToString**” metoduyla Vector3 sınıfında üretilmiş bir nesnenin pozisyon bilgilerini “**String**” türüne çevirebilirsiniz. Set metoduyla C# Script üzerinden X,Y,Z eksenleri için yeni pozisyonlar belirleyebiliriz.

“**Equals**” metodu ile iki farklı objenin pozisyon bilgilerinin aynı olup olmadığını kıyaslayabilirsiniz.

(Transform-Translate) “**Lerp**” metodu ile objenizin belirleyeceğiniz başlangıç pozisyonundan bitiş pozisyonuna hareket etmesini sağlarız. “**Scale**” metodu ile iki farklı objenin X,Y ve Z pozisyon değerlerini her koordinat için ayrı ayrı çarparak farklı bir “**Vector**” değer olarak dönüş yapar.

**Transform**: İçerisinde konum, rotasyon ve boyut bulunduran bileşendir. Nesnenin konumunu, dönüşünü ve ölçeğini saklamak ve değiştirmek için kullanılır.

**Position**: Unity ortamında objenin x , y , z eksenlerinde konumunu gösterir.

**Rotation**: Unity ortamında objenin x , y , z eksenlerinde rotasyonunu hamgi açıda olduğunu gösterir.

**Scale**: Unity ortamında objenin x , y , z eksenlerinde ölçeğini gösterir. Obje değerinin büyümesi o eksende daha büyük ölçekli görünmesini sağlayacaktır.

**Transform.localScale**: Nesnenin baglı bulundugu ebeveyne göre büyüklügünü belirtir.

**Transform.rotation**: Nesnenin 3D ortamdaki yönünü belirtir.

**Transform.root**: Nesnenin baglı bulundugu hiyerarşik dizinin en ustteki nesneyi (kök) verir.

**Transform.childCount**: Nesneye baglı bulunan çocuk (baglı nesne) sayısını verir.

**Transform.localRotation**: Nesnenin Baglı bulundugu evebeyn'e göre yönünü belirtir.

**Translate**: Transform bileşenin Position(x,y,z pozisyon bilgileri) değerlerine müdahale ederek çeşitli işlemler yapılmasını sağlayan bir fonksiyondur.

(**Diziler,İnstantiate Raycast**)

**Diziler**: Değişkenler içerisinde bir değer tutabilen yapılardır. Bir dizi içindeki bütün elemanlara aynı isimle ulaşılır. Sabit boyutludur.

**indeks**: Dizideki verilere verilen sıra numarasıdır.

**Listler**: Nesneleri bir liste sınıfı olarak gösterilmesi için özel olarak tasarlanmıştır. Listeler değişken boyutludur. Ekleme ve çıkarma yapılır.

**Listeye eleman ekleme**: Listeye elaman eklemek için ‘Add metodu’ kullanılır.

**Listeden eleman çıkarma**: Listeden eleman silmek için ‘Remove metodu’ kullanılır.

**Liste elemanlarına erişme**: Köşeli parantezler arasına erişmek istediğimiz index numarasını yazarak yapılır.

**İnstantiate:** mevcut bir objeyi sahneye yeniden oluşturma ya da klonlama işlemi yapılabilir. İstediğimiz bir noktada istediğimiz bir objeyi oluştururuz. 3 tane metod alır. Birinci üretilecek obe, ikinci üretilecek parametre üçüncüsü objenin doğrultusu.

**Raycast**: Sahneye bir ışını yansıtan ve bir hedef başarılı bir şekilde vurulursa bir boole değeri döndüren bir Fizik işlevidir. Bu olduğunda, mesafe, konum veya nesnenin Dönüşümüne bir referans gibi isabetle ilgili bilgiler, daha sonra kullanılmak üzere bir Raycast Hit değişkeninde saklanabilir.

**Orijin**: Işın'ın başlayacağı dünyadaki konumdur ve bir Vector 3 değeri alır. Raycast Hit değişkeni, Ray'in çarpışması hakkında bilgi depolayan bir veri yapısıdır. Çarpmanın dünyanın neresinde gerçekleştiği, Işın'ın hangi nesneyle çarpıştığı ve Işın'ın kaynağından ne kadar uzakta olduğu gibi bilgileri kaydeder. RaycastHit.point kullanarak isabetin dünyadaki konumu alınabilir veya RaycastHit.distance kullanarak Ray'in başlangıç ​​noktasından çarpma noktasına kadar olan mesafe alınır.

(**Dotween)** Dotween kullanarak nesnenin konumunu, rotasyonunu ve rengini değiştirebiliriz.

* **transform.DOScale**( Vector3 hedefLocalScale, float sure ): Transform objesinin localScale değerini (Inspector’daki Scale değeri) değiştirir. Sadece X, Y ve Z eksenleri için de ayrı versiyonları vardır.
* **materyal.DOColor**( Color hedefRenk, float sure ): Materyalin rengini değiştirir. Teknik olarak konuşmak gerekirse, materyalin shader’ındaki “\_Color” isimli değişkenin değerini değiştirir. Eğer başka bir shader değişkeninin değerini değiştirmek istiyorsanız, DOColor( Color hedefRenk, string degiskenAdi, float sure ) fonksiyonunu kullanabilirsiniz.
* **transform.DOMove**( Vector3 hedefPosition, float sure ): Transform objesinin position değerini değiştirir. DOLocalMove’a benzer şekilde, bu fonksiyonun da DOMoveX, DOMoveY ve DOMoveZ versiyonları vardır.
* **transform.DOLocalMoveX**( float hedefXLocalPosition, float sure ): Transform objesinin localPosition’ının X değerini değiştirir. Y ve Z eksenleri için de ayrı fonksiyonlar bulunmaktadır.
* **DOTween.Kill**( object hedefObje, bool tweeniTamamla = false ): hedefObje‘nin üzerinde çalışan tüm tween’leri elle sonlandırmaya yarar. Eğer tweeniTamamla‘nın değeri true ise, tween’ler sonlandırılmadan önce hedef değerleri hedefObje’ye uygulanır.

(**SOLİD VE DESİGN PATTERN**)

**Singleton Nedir**: Singleton desgin pattern çalışma zamanında yalnızca 1 object yaratılmasını garanti eden tasarım desenidir. Kullanımına ihtiyaç duyulan durum şudur; Birden çok sınıfın aynı instance’ı kullanması gerekmektedir. Tüm uygulama için yalnızca bir nesne olması gerekmektedir. Sadece bir nesne olduğu (unique) garanti edilmelidir.

Singleton pattern kullanmanın faydaları; Bir instance‘a kontrollü erişim sağlanır. Global değişkenler yaratmaktan kaçınılır. Singleton Pattern bize global bir erişim noktası sunar, global değişkenler gibi dezavantajı yoktur.

**Solid  Prensipleri**: Geliştirilen herhangi bir yazılımın esnek, yeniden kullanılabilir, sürdürülebilir ve anlaşılır olmasını sağlayan, ayrıca kod tekrarını önleyen prensiplerdir. Kodun esnek, sürdürülebilir ve geliştirilebilir tasarlanmaması kodu kırılganlaştırır ve yazılım ürününün gelişmesini etkiler. SOLID 5 farklı prensipten oluşur ve her birini baş harfini alır.

1)**Single Responsibility Principle**: Türkçe anlamı "Tek sorumluluk" anlamına gelen bu prensip her fonksiyonun ya da sınıfın sadece bir amaca hizmet etmesi gerektiğini savunur.

2)**Open/Closed Principle**: Open Sınıf için yeni davranışlar eklenebilmesini sağlar. Gereksinimler değiştiğinde, yeni gereksinimlerin karşılanabilmesi için bir sınıfa yeni veya farklı davranışlar eklenebilir olmasıdır.

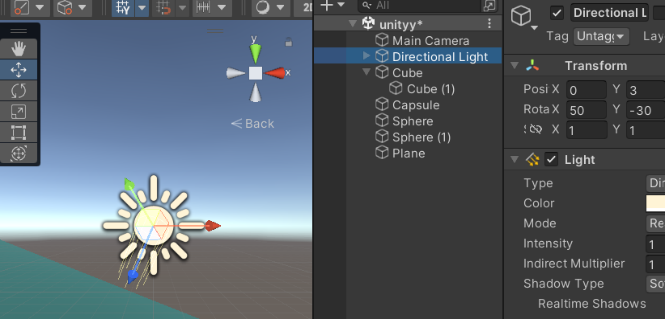
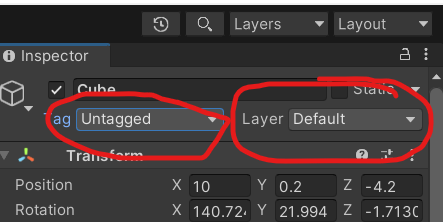
3)**Liskov Substitution Principle**: alıtım ve polimorfizm gibi nesne yönelimli tasarım soyutlama kavramlarıyla desteklenir. Bunlar genellikle, ortak davranışı tanımlayan arabirimler gibi soyut veri türleri kullanılarak üst sınıf nesnelerinin alt sınıflaması olarak gerçekleştirilir.

4)**Interface Segregation Principle**: “Arayüz Ayırımı” prensibinde; bir interface’e gerekenden fazla metot ve değişken eklemek yerine, daha çok özelleştirilmiş birden fazla interface oluşturulmalıdır. Nesneler, ihtiyacı olmayan özellik veya metotlar içeren interface’leri miras almaya zorlanmamalıdır.

5) **Dependency Inversion Principle**: Üst seviye sınıflar alt seviyedeki sınıflara bağımlı olmamalıdır. Yani yazdığımız projede sınıflar arası bağımlılıklar minimum seviyede olmalıdır. Bu prensibe göre de bir sınıf diğer bir sınıfa doğrudan bağımlı olmamalıdır. Aralarındaki bağ soyut bir kavram üzerinden sağlanmalıdır. Bu soyut kavram interface de olabilir abstract class da olabilir.

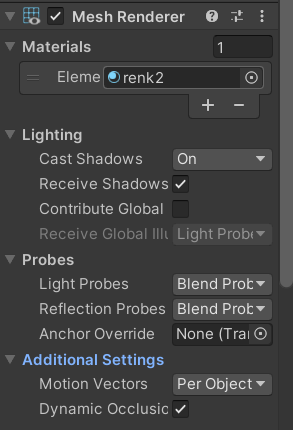
**EventSystem Nedir?** EventSystem, Unity sahnesindeki olayları işlemekten ve yönetmekten sorumludur. Bir sahne yalnızca bir EventSystem içermelidir. EventSystem, bir dizi modülle birlikte çalışır ve çoğunlukla yalnızca durumu tutar ve işlevselliği belirli, geçersiz kılınabilir bileşenlere devreder. Bir GameObject'e bir Event System bileşeni eklediğinizde, çok fazla işlevselliğe sahip olmadığı fark edilecek, bunun nedeni Event System'in kendisinin Event System modülleri arasında bir yönetici ve iletişim kolaylaştırıcı olarak tasarlanmış olmasıdır. Etkinlik Sisteminin birincil rolleri aşağıdaki gibidir;

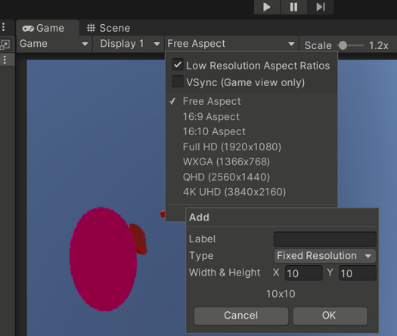
Hangi GameObject'in seçili olarak kabul edildiğini yönetin. Hangi Giriş Modülünün kullanımda olduğunu yönetin.

Inspector görünümün de etiket ve layer bulunuyor bunlar objelere daha kolay erişim sunmak ve aynı zamanda da objeleri farklı bir kategoride bulundurmak için kullanılan yapılar. Mesela oyunda düşman bir diye tag belirttik ve karakterimiz ateş ettiği zaman sadece düşmanları öldürsün diyebiliriz. Layerlersa bazı objelerimiz görünmez olucak ama çarpabilicez gibi veya bazı obje grubuna daha farklı bir şekilde muamele edeceksek sanki onlar farklı bir ışık kaynağının altındaymış gibiyse kullanılabilir

Directional light bit gameobject transformları var hareket ettirdiğimizde bir şey olmuyor çünkü bir yönlü ışık kaynağıdır bulunduğun yerin bir önemi yoktur yönlü ışık kaynaklarının sadece yönü önemlidir

Zemine bir mesh Collider ( çarpışma hesabını yapıyor ) ekledik bu katı cisimle çarpışabilmesini sağlayan bir componenttir bundan dolayı zeminimizle çarpışan objemiz çarpışmayı başardı ve zeminimizin içinden geçmedi.

Mesh renderer: mesh dediğimiz 3 boyutlu objeler noktalar ve yüzeylerden oluşur noktalar ve yüzeylerden oluşan 3 boyutlu objeye mesh denilir mesh renderer de bu meshi ışıkla beraber gölgelendiren bir componenttir. Cast Shadows; objemiz başka yüzeylere gölge göndersin mi?receive shadovs; Seçili objemizin başka yüzeylere gölge göndermesini istiyorsak seçeriz.



Game 🡪 Free Aspect(Serbest Oran)

Aspect Ratio oranını ayarlamamızı sağlıyor

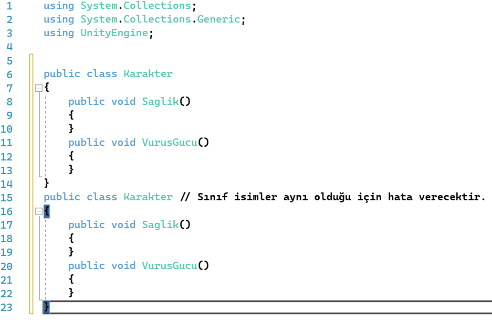
Fixed Resolution Çözünürlüğünü.

Unity Namespace ve Using Nedir?

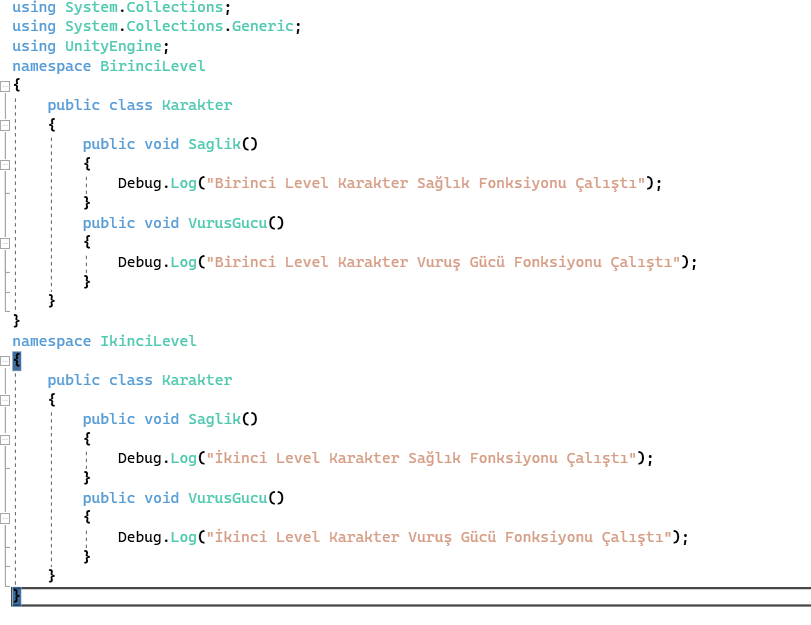
Tüm fonksiyonlarımız bir sınıfın içinde yer alıyordu yani sınıf fonksiyonları kapsıyordu. Namespace ise sınıfları tutan bir yapıdır. Hiyerarşi olarak **Namespace > Class > Function** olarak düşünebilirsiniz.

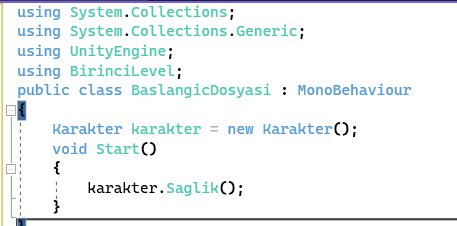
Unity Namespace’ler ne işe yarıyor?

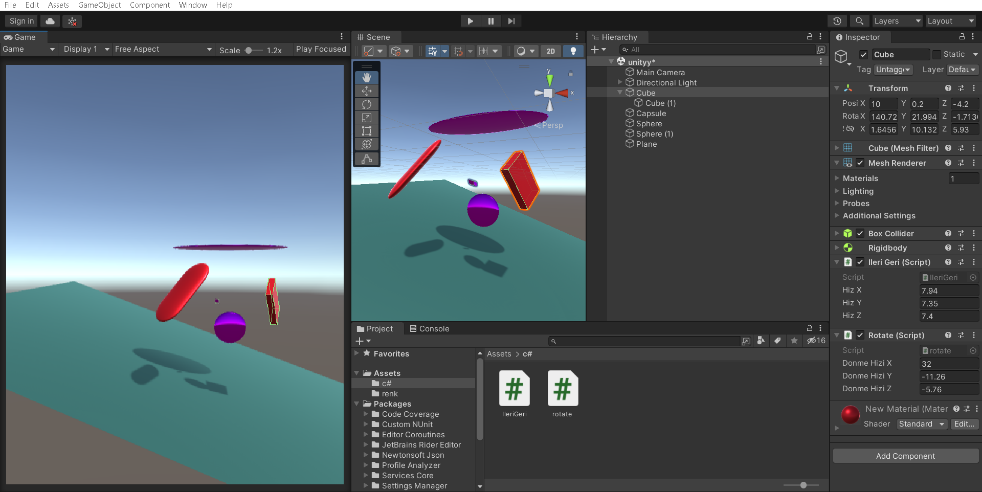
Projelerimiz büyüdükçe ve birden fazla kişinin kod yazması gereken durumlar olduğu zaman karışıklık oluşması çok muhtemeldir. Çünkü projelerimiz büyüdükçe sınıfların isimleri ya da fonksiyonların isimleri çakışabilmektedir. Bu durumlarda kod karışıklığının önüne geçmek ve kod okunabilirliğini arttırmak için Namespace’ler kullanılır.

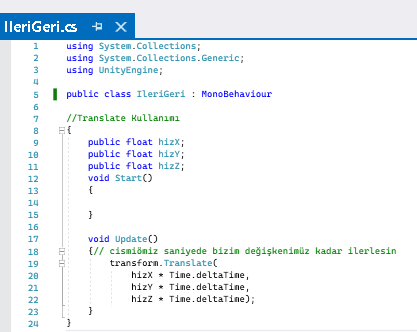


Öğrendiklerimi kısaca şu şekilde aktarabilirim;Diyelim ki oyunda bir karakterimiz var. Karakterimizin ise sağlık ve vuruş gücü özellikleri olsun. Karakterimiz için public class Karakter isminde bir sınıf oluşturduk. Özellikleri için ise public void Saglik() ve public void VurusGucu() isminde iki adet fonksiyon oluşturduk

 Şimdi Unity üzerinde yeni bir C# Script dosyası oluşturalım ve ismini “KarakterYonetimi” verelim. İçerisine ise aşağıdaki kodu yazalım. namespace anahtar kelimesi ile iki farklı namespace oluşturduk. Böylece aynı dosya içinde iki adet “Karakter” isminde sınıf olmasına rağmen bize hata vermeyecektir.

Örnek olarak Unity Oyun Motoru üzerinde yeni bir C# Script dosyası oluşturarak ismini “**BaslangicDosyasi**” yaptım.using BirinciLevel; şeklinde bir giriş yapıldı. Oluşturduğumuz namespace’leri bu şekilde using anahtar kelimesi ile kullanabiliyor olduğunu öğrendim. **UnityEngine**‘de bir namespace ve içerisinde **MonoBehaviour**isminde bir sınıf barındırıyor.  **using** BirinciLevel; namespace’ini ekledikten sonra Karakter karakter = new Karakter();koduyla bir nesne oluşturduk. Sonrasında ise karakter.Saglik();yazarak oluşturduğumuz karakter nesnesinden **saglik()**fonksiyonuna eriştik

Unity ara yüzüne giriş yaptığımda sahne görünümünde ne varsa hierarchy görünümünde de o var sağ tarafta denetleyici ekranında objemizin özelliklerini değiştirebildik. Hierarchy ekranında sağ tık yapıp objeleri ekledik ve yüzeyin konumuna göre objelerimizi yerleştirmiş olduk sağ tık duplicate ile objemizi çoğalttık main kameramızı game ekranına göre ayarladık Rigidbody ekleyerk objelerimize fizik hareketi ekledik zemine rigidbody eklemedik ekleseydik katı cisim gibi o da düşerdi ama mesh Collider ekledik objelerimizin içinden geçip gitmesini engellemiş olduk .Daha sonra objelere dönme ve hareket eklendi.



Rigidbody ekli olan objenin transformunu doğrudan değiştirmek yerine Rigidbody üzerinden değişiriz Hareket davranişi Scriptinde objelerimize dönme hareketini 3 eksende belirleyerek objelerimizin dönmesini sağlamış olduk. Translate objeyi belli bir yönde taşımak için kullanıldı.

Fonksiyonlar için; <https://docs.unity3d.com/ScriptReference/30_search.html?q=transform> unitynin kendi sitesinden yararlandım

Staj hocam bugün Unity MonoBehavior Sınıfı Temel Fonksiyonlarınından start(), awake() ve UpDate() fonksiyonlarının çok kullanıldığından bahsetti şirket tarafından yapılan birkaç proje üzerinden kısaca anlattı ve yaptıkları projeleri inceleme şansım oldu benden bugün fonksiyonları araştırmamı istendi . Fonksiyonların kullanıldığı örnek projelere bakılıp incelendi ve araştırmalar yapıldı.

MonoBehaviour, her Unity scriptinin türetildiği temel sınıftır. C# script kullandığınızda, MonoBehaviour’dan türetmemiz gerekiyor. Kısaca en çok kullanılan fonksiyonlar;

**Awake():** Bu fonksiyon daima Start fonksiyonundan önce çalıştırılır, ayrıca bir prefab Instantiate edildiği anda da çalıştırılır. (Eğer GameObject aktif (active) değilse bu fonksiyon obje aktif olana kadar ya da bu objedeki bir scriptte yer alan bir fonksiyon dışarıdan çağrılana kadar çalıştırılmaz.)

**Start():** Start fonksiyonu Awake fonksiyonundan sonra çalışan fonksiyondur. Start fonksiyonunun çalışması için bulunduğu objenin aktif olması gerekir. Obje aktif değilse sadece Awake fonksiyonu çalışır Start fonksiyonu çalışmaz.

**Update():**Update fonksiyonu bir kere çalışıp durmaz, her frame de çalışır.Bilgisayarın donanımına göre de çalışma sayısı değişiklik gösterir. Daha iyi bilgisayarlarda Update fonksiyonu saniyede 100 kere de çalışabilir. Ve önemli bir bilgi olarak da çalışma aralıkları aynı değildir.

**FixedUpdate ():** FixedUpdate fonksiyonu düzenli bir biçimde her zaman çalışır. Update fonsiyonundan farkı düzenli bir şekilde çalışmasıdır. Genelde 0.02 saniyede bir çalışır. Donanımdan donanıma bu süre değişmez. Düzenli çalıştığı için fizik olaylarında kullanılır. Fizik olaylarında normal Update fonksiyonu kullanılmaması gerekir. Çünkü düzenli bir şekilde çalışmadığı için sağlıklı olmaz.

**LateUpdate ():**LateUpdate Fonksiyonu diğer Update fonksiyonları bittikten sonra çalışır. Genelde en çok kamera konumlandırılmasında kullanılır.

**OnEnable ve OnDisable :** OnEnable fonksiyonu obje aktif hale getirildiğinde OnDisablefonksiyonu obje pasif hale getirildiğinde çalışır. Basit scriptin bulunduğu obje aktif ve pasif hale getirildiğinde çalışır **SetActive ():**SetActive fonksiyonu objenin görünürlüğünü kapatır. Tam tersini yapmak için yani objenin görünürlüğünü açmak için false yerine true yazılır.

**Destroy ():**Destroy’un içine yazdığımız obje yok edilir.  2. Değer ise kaç saniye sonra yok edileceğini belirler zorunlu değildir.

**OnApplicationQuit:**Oyundan çıkılmadan hemen önce gerçekleşir. Web player’da oyuncu web sayfasını kapatmak istediği anda çalıştırılır.

**OnDisable:**Script disabled yapıldığında ya da GameObject inactive yapıldığında çalıştırılır.

Özetleyecek olursak, işte scriptlerin çalıştırılma sıralaması:

-Tüm scriptlerdeki Awake fonksiyonları

-Tüm Start fonksiyonları

-Tüm FixedUpdate fonksiyonları

-Fizik hesaplamaları

-OnEnter/Exit/Stay Trigger fonksiyonları

-OnEnter/Exit/Stay Collision fonksiyonları

-Eğer Rigidbody’de interpolate özelliği açıksa bunun için gerekli hesaplamalar yapılır

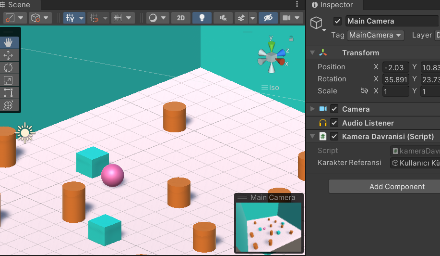
-OnMouseDown/Up vb. olaylar gerçekleşir

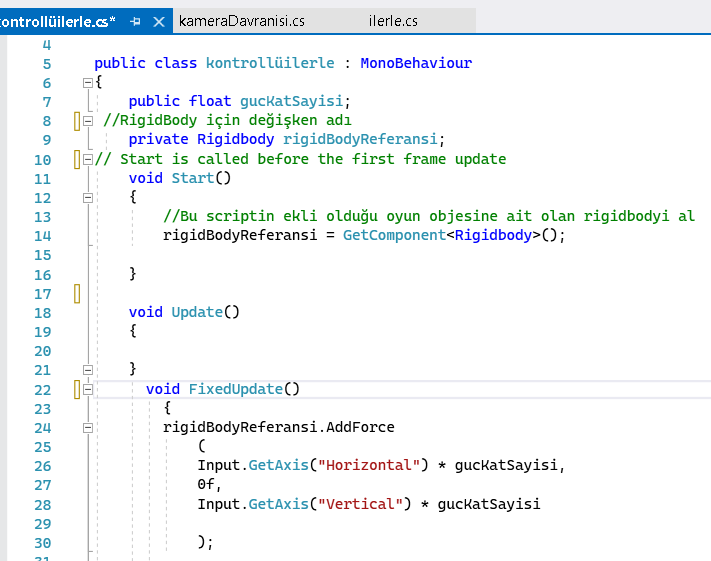
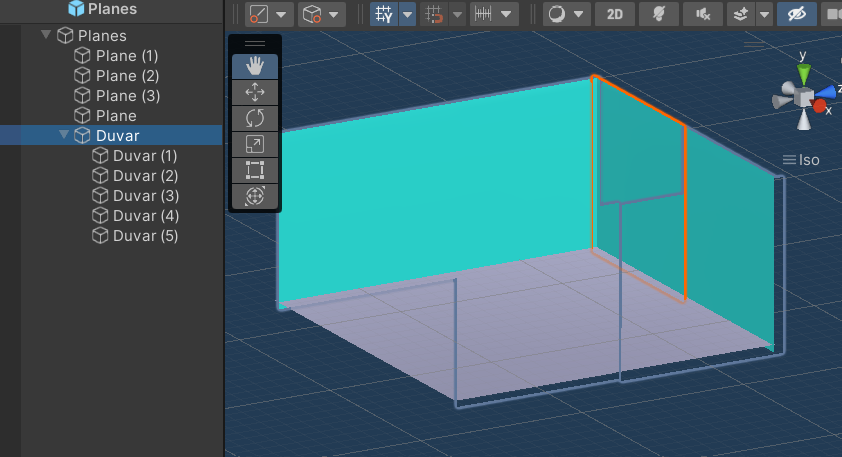
-Tüm Update fonksiyonları

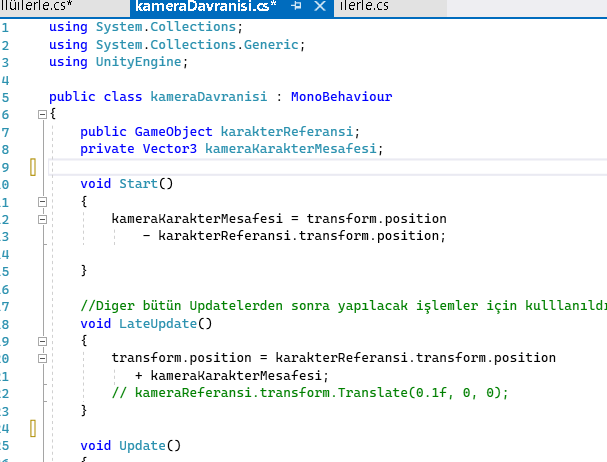
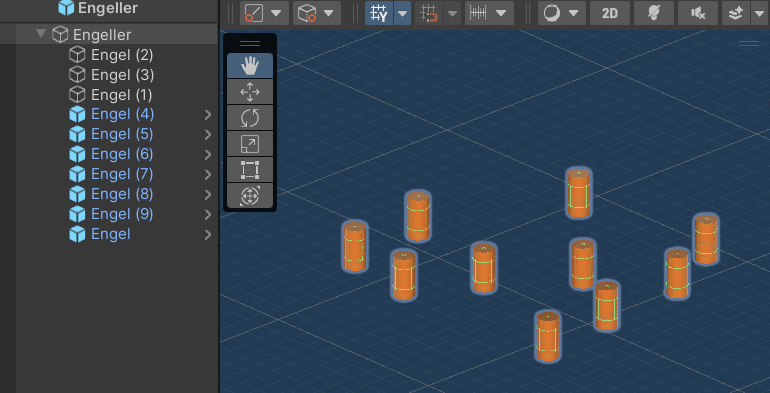
-Animasyon işleri gerçekleştirilir

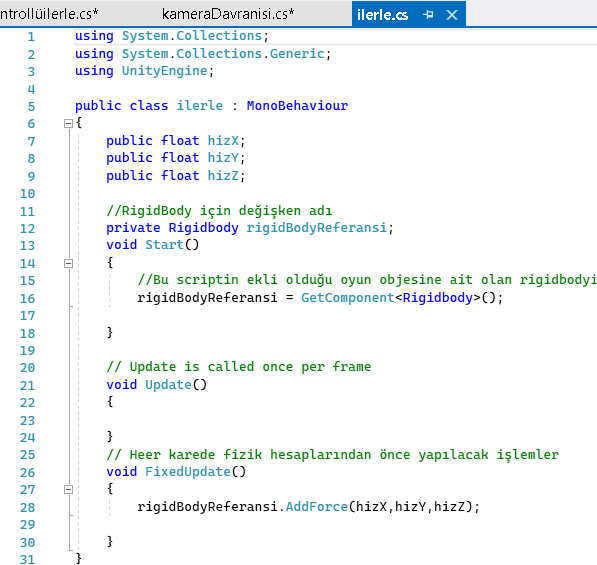
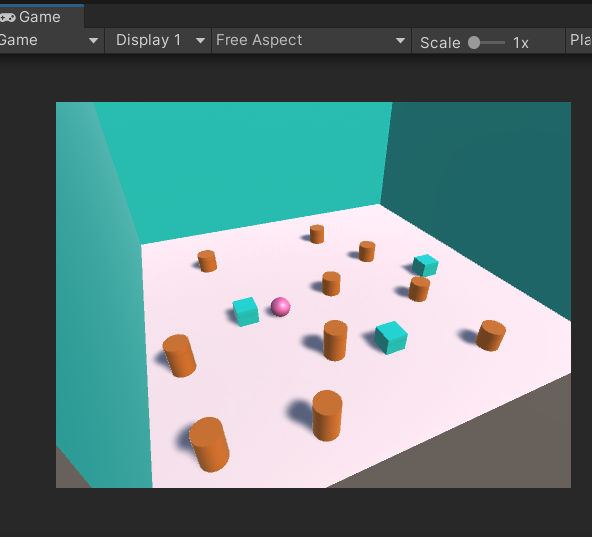
-Tüm LateUpdate fonksiyonları

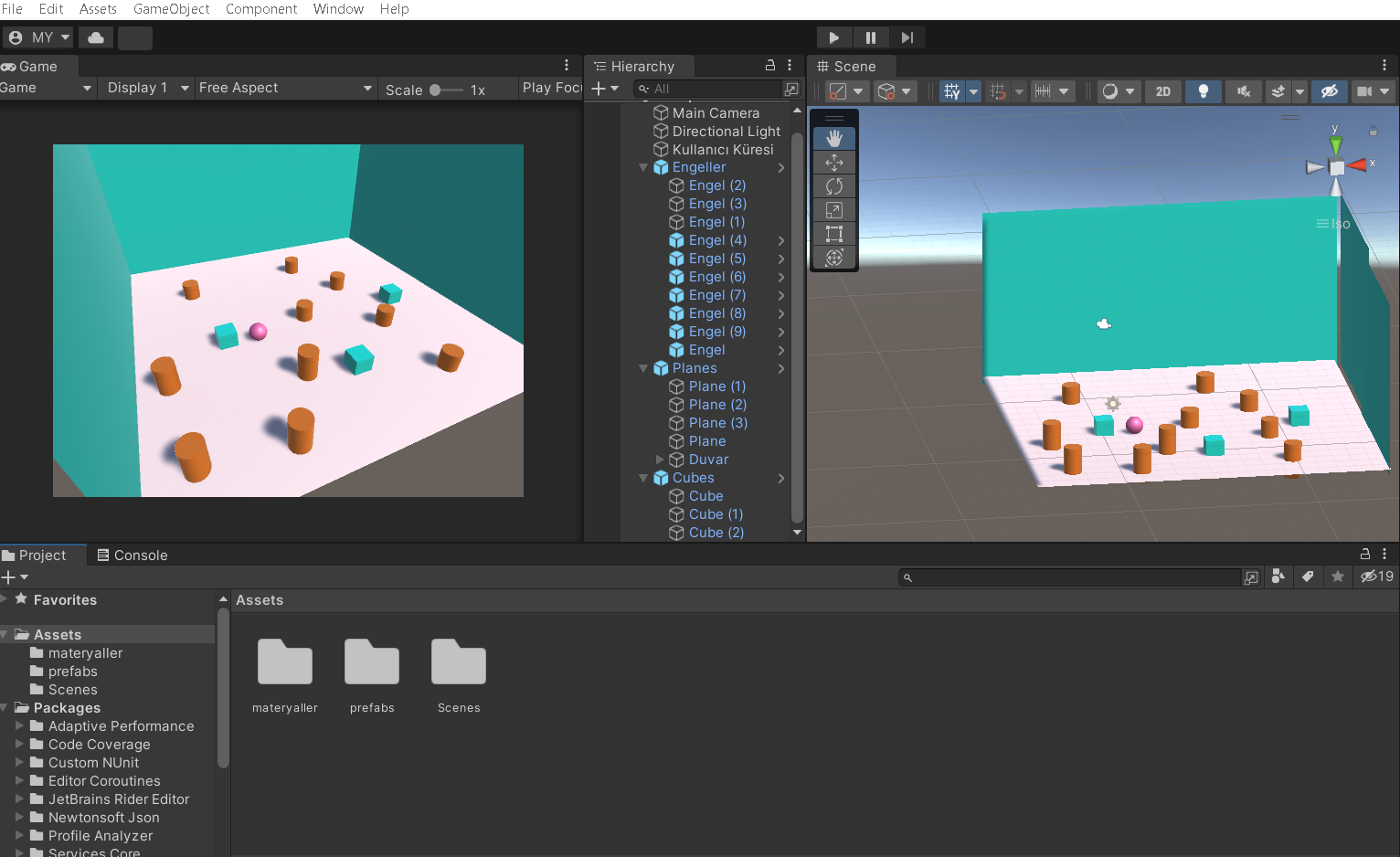
-Render (görüntü) alma işlemi

Staj hocam Unity ara yüzüne daha iyi hakim olabilmem adına benden bir oda yapmamı ve bir adet kullanıcı küresi oluşturmamı ve klavye ile hareketine yön vermemi , oda içerisinde engellerin olmasını ve kameranın kullanıcı küresini takip etmesini istedi bunun üzerine araştırma yaparak benden isteneni yapmaya çalıştım.

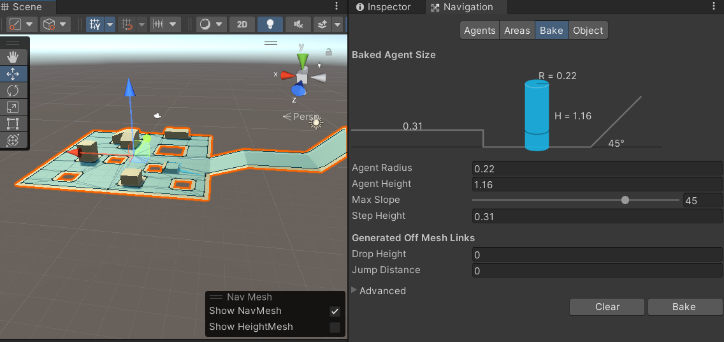


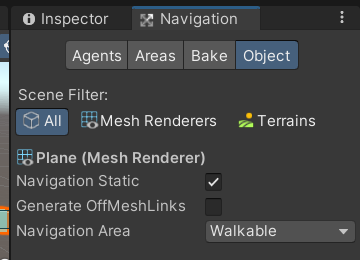
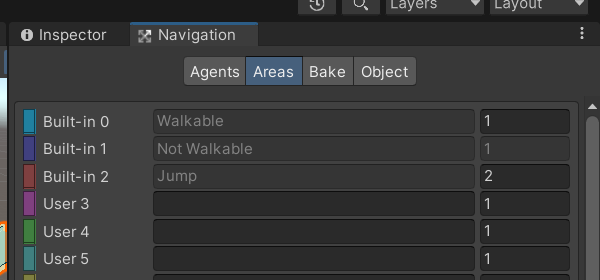


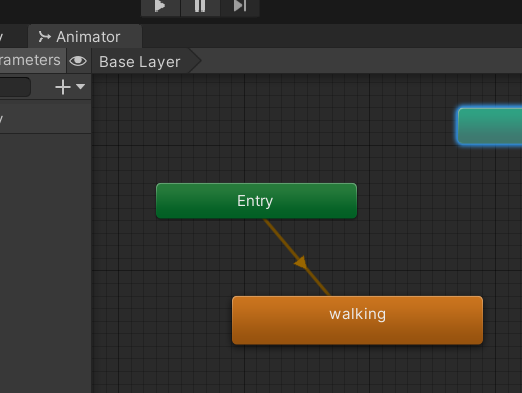
 

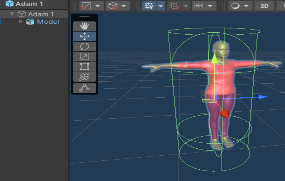
İlk olarak düzlemimi ve objelerimi ekleyip onları sahneme yerleştirdim. Küreme fiziksel temelli bir hareket yapmak istediğim için gameObjemin transformunu değiştirmek için kontrollü ilerle scriptini yazdım Rigidbody üzerinden objeye kuvvet ekledim .Rigidbodye ulaşmak birkaç farklı yolu gösterdi Bunlardan bir tanesi private bir değişkeni tutarak o objeye ait olan Rigidbody i get component diyerek bu bir Rigidbody olacak denildi ve bu şekilde rigidbodysini almış olduk ve aynı Rigidbody referansı üzerinden AddForce() ‘ i kullanarak objemize x,y,z yönlerinde belli bir kuvveti eklemiş olduk. Bu hareketi yaparken FixedUpdate() metodunun içerisinde kullandım çünkü kuvvet fiziksel hesaplamalardan önce eklemek istedik. FixedUpdate()’in içerisine yazılmış işlemler fiziksel hesaplamalar yapılmadan önce olan aşamayı gösterir daha sonra da Update ler falan kullanılıyor Updatelerden sonra LateUpdate i kameranın bulunduğu pozisyonu güncellemek için kullandım. Kamera ve karakter arasındaki mesafeyi korumak istediğimiz için kameraya KameraDavranisi scriptini ekledim scriptin içerisinde kameraya KarakterReferansı atadım bu referansla karakterin nerede olduğunu biliyoruz Karakterin transformundan pozisyonuna ulaşarak bu bilgiye erişmiş oldum .Aslında ilk olarak en başta karakterle kamera arasındaki bu mesafeyi vektör olarak atadık Start metodu bir defa yapıldığı için oyunun en başında defa bir defa bu vektör belirlendi ve LateUpdate içerisinde tüm hareketler hesaplandıktan sonra görüntüyü güncellemeden önce yapılan o anda kameranın pozisyonu güncellendi. Kameranın pozisyonunu ise karakterimizin bulunduğu nokta+ vektör(korumak istediğimiz mesafe) e eşitlendi. 

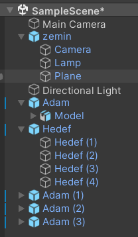
Süreç içerisinde takıldığım noktalarda staj hocamdan ve game developer olarak çalışan arkadaştan yardım aldım temel şeyleri anlatarak yardımcı oldular. Staj Hocam bu şekilde deneyerek ve temel şeyleri görebileceğim ödevler verilerek devam edileceğini söyledi

 Bugün Hocam belirli bir noktaya gitmesini istediğimiz zaman otomatik olarak o noktaya ulaşan bir rota çizen ve oraya doğru ilerleyen oyun objeleri oluşturmamı istedi ve takıldığım noktalarda yardımcı olundu.

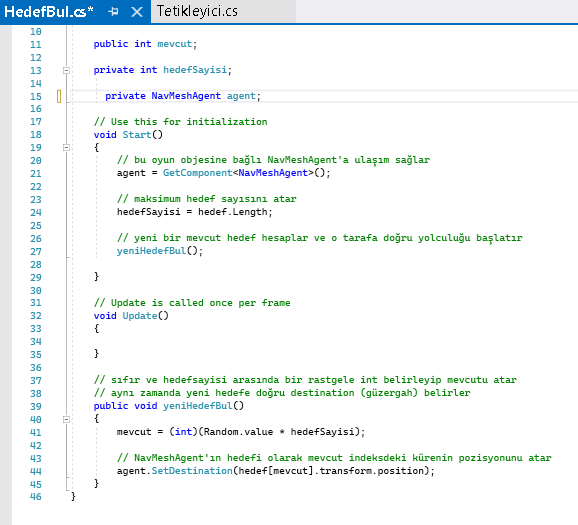
İlk olarak zeminimizi blender ile oluşturduk blender kullanımı konusunda zemin için hocamdan yardım aldım. Şeklin üzerinde hareket haritası çıkarmak için Unity nin navigation sekmesini açarak **Bake** (“Belli ayarları yüzeye kazımak için kullanılır.” Yürüme alanları daha sonradan hesaplamaya gerek kalmasın diye baştan oluşturup bizim yüzeyimize işlemeyi sağlıyor) Bake in altında **Agent Radius**: objemizin bir duvara veya zemine ne kadar yaklaşabilecğini ; **Agent Height**:objenin ulaşabileceği boşlukların ne kadar düşük olduğunu ; **Max Slope**: Yürüme haritasında yürünebilir alanlar hesaplanırken fazla bir eğim varsa bundan sonrasına agent(yürüyecek olan obje) miz ulaşamayacağını tanımlar **Step Height**: Adım atarak üzerinden geçebildiğimiz yüksekliktir burada değişiklik yapılarak zemin için güncelleme yapıldı.

Navigation Static aktif hale getirilirek yürünebilir alan yaptık zeminimizin üzerinde mavi bir kaplama oluştu bu yürünebilir alan olduğunu gösteriyor. Areas; yürünebilir yürünemez ve zıplanabilir olarak ayarlandı ve daha farklı alanlar oluşturabilmemizi sağlıyor.

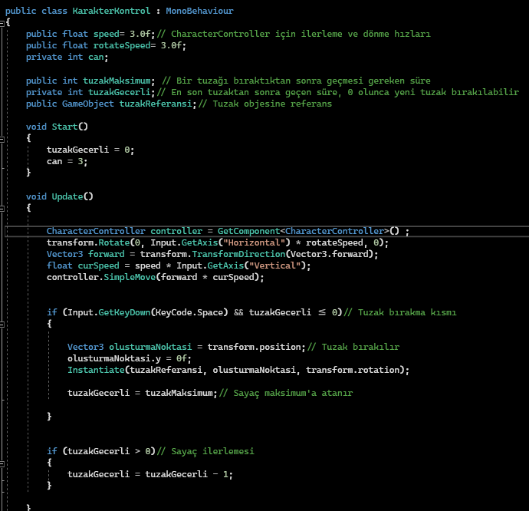
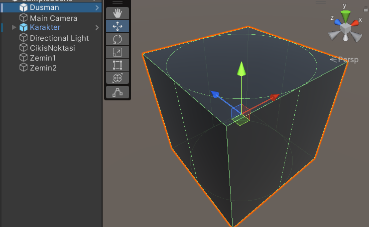
animatör kısmından varsayılan animasyonumuzu yürüme olarak ayarladım.

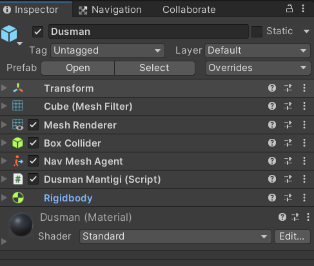
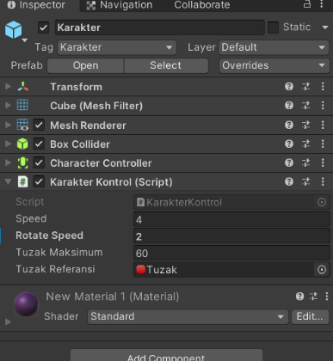


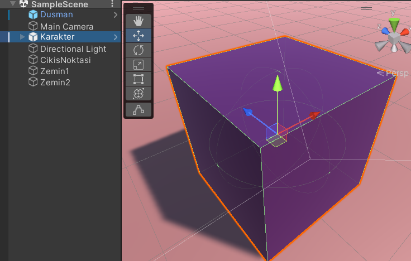
Adam objemi oluştururken izlediğim yolu anlatmam gerekirse önce silindir oluşturdum daha sonra silindiri görünmez yapıp modelimi ekledim. Modelim ise AdobeFuse ile oluşturuldu.

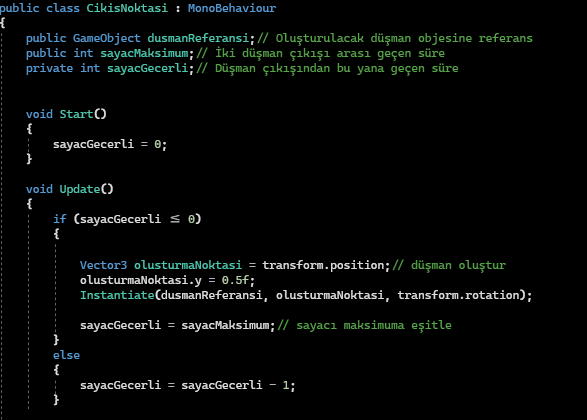
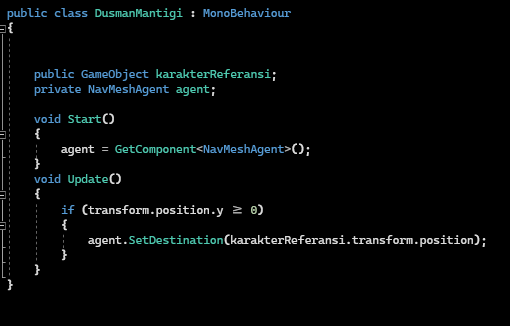
Hedef olarak göstereceğimiz nesnenin küre olmasını istiyoruz bunun için hedef nesnesinin bulunduğu pozisyona git diyoruz Adam objemizin hedef olarak gösterdiğimiz nesneye doğru gitmesini istiyoruz bunun için HedefBul scriptini yazdım. NavMeshAgent.SetDestination: Her bir NavMeshAgentın gitmeye çalıştığı bir nokta var başlangıç durumunda gitmeye çalıştığı bir yer yok ancak SetDestination diyerek bir noktaya gitmesini istediğim için navmeshagenttan oraya ulaşan bir rota çizerek biz yürü demeden o tarafa doğru yürümeye başlamasını sağlamış oldum.

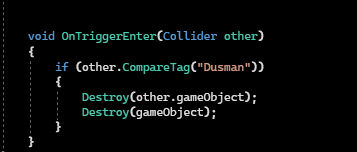
Adam objem hedefe ulaştığı zaman küremizin tetikleyicisini çalıştırarak objemizin içerisine girmesini sağladım. Tetikleyici scriptini içinde yenihedefbulu çağırdım. Tetikleyici Collider üzerinden çalışıyor Collider; objemizin fiziksel boyutta kapladığı alanı ifade ediyor. Zeminimizde mesh renderer seçili, bir de mesh Collider var mesh Collider 3 boyutlu şeklin bütün yüzeylerini hesaba katarak çarpışma hesabı sağlar. Bütün yüzeyleri hesaba kattığı için çok daha keskin bir çarpışma hesabı yapar. Colliderin içerisindeki belli durumda bir tetiklemeye sebep olan fonksiyonlardan OnTriggerEnter ı kullandık OnTriggerEnter bu tetikleyicinin içine başka bir obje girdiği zaman kullanılan bir metottur.

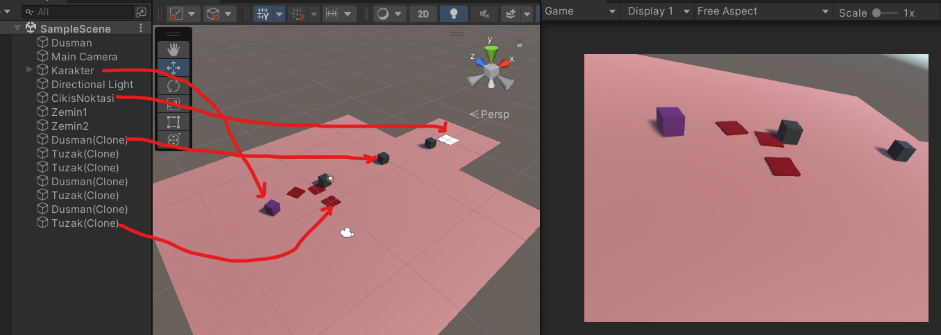
Sahnede bir küp karakteri oluşturulup karakterimizin hareketini kullanıcının klavyeden belirlemesi sağlandı. Küpü takip eden bir düşman objesi oluşturup bu düşmana karşı bir tuzak objesi oluşturduk. NavMesh sahnede kendi kendine bir noktaya gitmesini sağlıyor. Hareket için character controller adlı component kullanıldı **Character Controller** : oyun karakteri gibi davranacak olan bir yandan fiziksel bir hacmi olacak bir şeylere çarpabilecek aynı zamanda da tamamen fiziksel hareket edemeyecek nesneler için kullanılan componenttir. Kullanıcı doğrudan hareketi belirtecekse bu component kullanılıyor **getcomponent metodu**: bu objeye ekli olan herhangi bir componente ulaşmamızı sağlıyor. Hareketimizi sağlamak için KarakterKontrol scriptinde **simple Move metodunu** kullandık. **Simple Move**:Hareketin detaylarını sadeleştirir Karakterin içine yer çekimini de hesap edip öyle ekler. Bu objeye ekli olan karakter kontrolcüsünü aldık ve controller adlı değişkene atanmış durumda. Transform ve rotate fonksiyonu kullanarak Y ekseninde hareket etmesini istedik sağ veya sol 1 ve -1 dönderir ve dönme hızıyla çarparak biraz daha hızlı olması sağlanacak **forward** ise objenin öne bakan ileri yönü . Hareketi sağlamak için bir vektör oluşturuldu .dikey eksende yukarı yönde ilerliyorsa artı tersi yönde gidiyorsa – yönde ilerler. **Simple Move fonksiyonu** çağrıldı bu bir adet fonksiyon bekliyor hangi yöne gidilmesini istiyorsak o vektörü yazmamız gerektiğinden forward ile mevcut hızı çarptık bu şekilde objemizin hareketi sağlandı eğer space tuşuna basılırsa ve tuzakgeçerli değeri 0 dan küçük veya eşit olursa komut verilecek. **GetKey** bu tuşa basılı olduğu sürece sürekli ateş edilecekse **GetKeyDown** ise tuşa sadece basıldığı anda sadece bir defa tetikleniyor. Tuşa basıldığında tetiklensin istedik bu yüzden GetKeyDown kullanıldı.



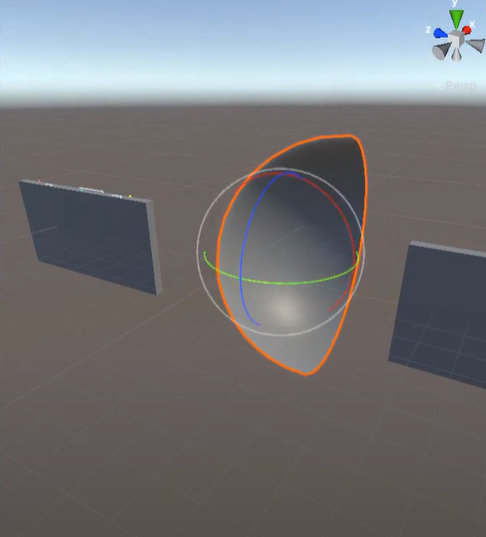
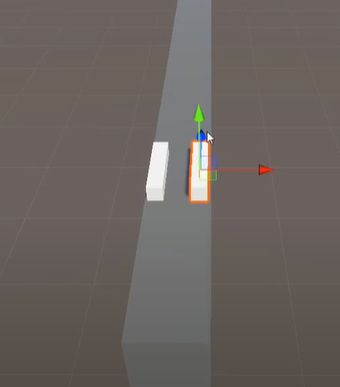


DusmanMantigi scriptimizde düşman objemize nav mesh agent eklendi belli bir hızı var dönme açısı var ama belli bir noktaya gitmesi için atama yapmamız gerekiyordu bunun için nav mesh agent kullanıldı. Update in içerisinde navmeshagentllarımızın belli bir noktaya gitmesi için **setDestination metodu** kullandık bu metot belli bir noktaya giden hedef bulmamızı sağlıyor . setdestination ile hedefimiz, adamımız neredeyse orada olacak. Karakteri takip etmek için ona bir karakterreferansı verildi Bu script'in ekli olduğu game object'e ait olan NavMeshAgent component Bu component'i GetComponent metoduyla alır Düşman zeminin civarındaysa Karakterin olduğu yere bir yol bulur ve gitmeye başlar

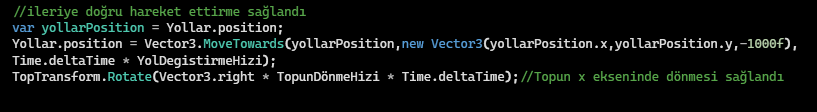
 Cıkıs noktası oluşturarak Duşmanların oluşmasını otomatikleştirmek istedik bunun için de çıkış notası kübü oluşturuldu ve scriptin içinde çıkış noktasında bir sayaca ihtiyacımız var fonksiyonumzda sayaçgeçerli sıfırdan küçük ve eşitse düşman oluşsun dedik . Düşmanımızı çıkış noktasının bulunduğu noktadan başlatmak istedik bunu Instantiate i kullanarak nerde bulunuyorsa oraya yerleştirmiştik tuzak bırakılmasıyla çıkış noktasının düşman oluşturulması aynı mantıkta çıkış noktası nereyse onun bulunduğu transforma karşılık gelecek.

Tuzağımızı Trigger yaptıktan sonra other referansı olarak **OnTriggerEnter** ı tetikler tagi karşılaştırmak için ontrigger bir şey bu alana girdiği zaman tagini kontrol edeceksek other yani Collider çarpışan objenin tagine bakıp yok edebiliyoruz. Tuzak Mantigi scripti oluşturduk. Dusman tuzağa girdiyse düşmanı yok et. Colliderler oyun hareket ederken oyun başladıktan sonra colliderler sabit cisimse yeniden hesaplanmıyor sabit cisim gibi davranmaması için objemizle Rigidbody ekledik ıskinematiği de açarak objemizin dış kuvvetlerden etkilenmesinin önüne geçmiş olduk SetActive ile destroy etmek yerine doğrudan kaldırmak yerine sahnede kalkmayıp pasif hale geliyor.

Proje Başlangıcı : Color Balls

 Color Balls projesinin amacı, oyuncu topumuz oyuna başladıktan sonra sahnemizdeki yolda ilerleyerek bitiş çizgisine ulaşmaya çalışacaktır. Yol boyunca oyuncu topumuzun karşısına farklı renkte toplar çıkacaktır; değdiği top hangi renkte ise o renkteki rengini alırken aynı zamanda da skoruna puan eklenecek. Yolumuzda engelller de var eğer oyuncu topumuz engellerden birine çarparsa oyuna yeniden başlaması gerekir. Topumuz yolun sonunda bitiş çizgisine ulaştığında oyun sonlanmaktadır ve yeniden başlat butonuyla yeni skorlar denenebilmektedir.

Projeme sahne tasarımıyla başladım . Sahnemde yolumu oluşturup üzerine engel objelerimi yapıp yerleştirdim . Oyuncu topumu da oluşturduktan sonra topu sol ve sağ pozisyonda hareket ettirmemiz gerekiyordu bu sol maus düğmesini koruyarak ve konumunu değiştirerek gerçekleşecek startTouchPos ile İlk tıklanma noktası alındı hareket ettirme var mı diye kontrol edildi hareket ettirme 0 landı **Math.Abs** parametresi ile sağ sol miktarı oranlandı Math.Abs Verilen sayının mutlak değerini elde etmemizi sağlıyor -1 ve 1 değeri de aynı uzunluğa sahiptir. Abs ile bu yön bilgisinden kurtulup yalnızca büyüklüğünü elde etmiş olduk. **Math.Clamp** fonksiyonu herhangi bir değeri istenilen iki değer arasında tutmaya ve bunun dışına çıkmamasına yardımcı olur. Bu sayede karakter sağa veya sola giderken yoldan dışarıya çıkmamış olur.. Bu fonksiyona verdiğimiz değer eğer belirttiğimiz sınırdan az ise, minimum değeri, sınırdan fazla ise de, belirttiğimiz maksimum değeri bize döndürür bu şekilde Sağ sol yapılabilecek sınırlar belirlendi topumuz sınırını bilerek yolumuzda sağa ve sola kaydığında kontrol edilmiş oldu . İnputlar alınarak sağ sol yapıldığındaki bilgiler alındı ve uygulatıldı. Topumuzu ileri doğru hareket ettirmek için **[MoveTowards](https://docs.unity3d.com/ScriptReference/Vector3.MoveTowards.html)**  kullanıldı . Bu fonksiyon tarafından hesaplanan konum kullanılarak her karede topumuzun konumu güncellendi ve hedefe doğru düzgün bir şekilde hareket etmiş oldu.

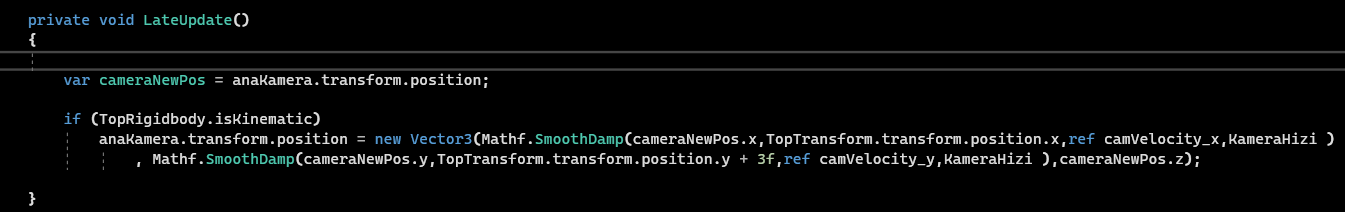


Camera Controlu ile devam edilecek olursa, karakterimizin anlık konumunu alabilmek ve kameraya o konumu takip ettirebilmek açısından camerNewPos adında bir transform değişkeni tanımlanıyor. Ve sonrasında kameranın pozisyonu ile karakterin pozisyonu arasındaki mesafeyi tutabileceğimiz bir değişken üretiyoruz. SmoothDamp yönteminde bir referans parametresi olarak geçirilir; bu, bu değişkene SmoothDamp yöntemi içinde bir değer atandığı anlamına gelir. Bu nedenle, SmoothDamp yöntemini çağırdıktan sonra, hız değişkeni, gerektiğinde kullanabileceğiniz yeni atanan değeri tutacaktır. Bu da kameranın karakteri takip etmesini sağlıyor.

LERP yöntemine kıyasla daha iyi pürüzsüzlük

sağladığından Düzgün Kamera hareketi için **SmoothDamp yöntemi** kullanıldı.

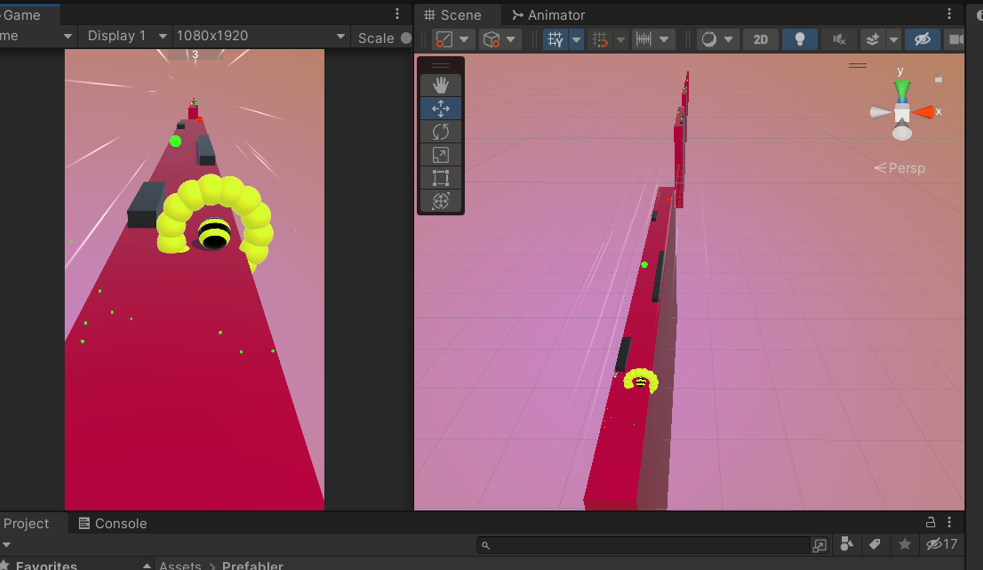
Sağa ve sola her vardiyadan sonra kameranın topla birlikte hareket etmesi sağlandı

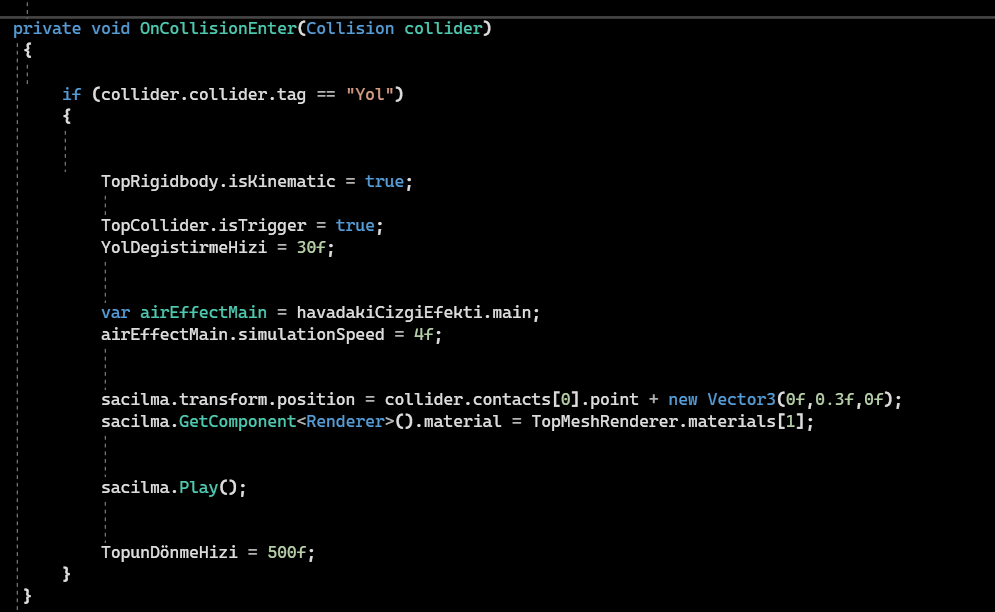
****

**SmoothDamp**(): Verilen vektor değerine aniden erişmek yerine, yumuşatarak erişmemizi sağlayan bir fonksiyondur. En basit kullanım örneği ise bir objenin başka bir objeyi yumuşatılmış, anında kendini ışınlamadan takip etmesi işlemidir.

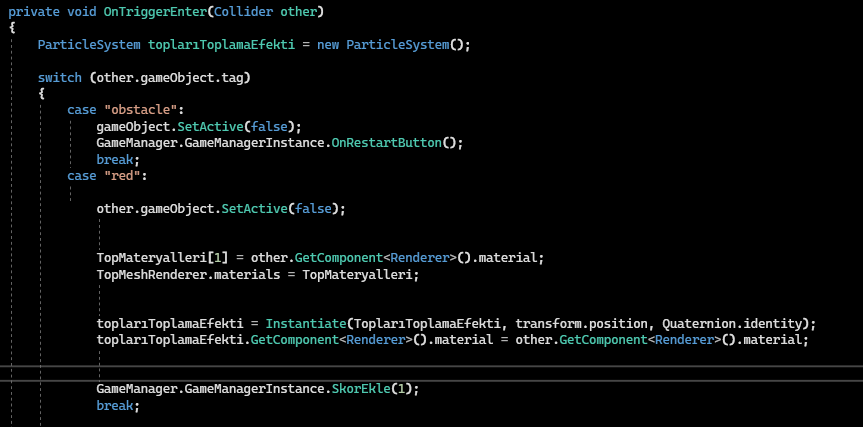
Objenin pozisyonunu hedef pozisyona yumuşatarak erişerek güncelliyoruz.

Takip için araya belirli bir mesafe koyarak hedef konumumuzu belirliyoruz

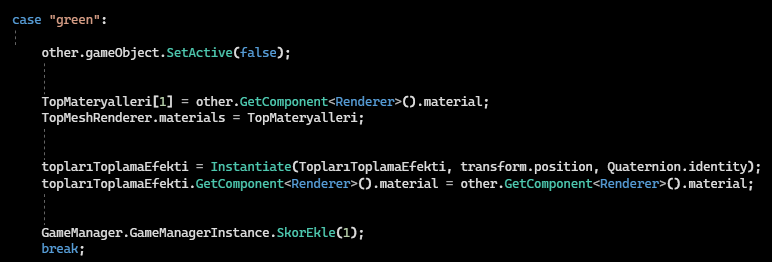




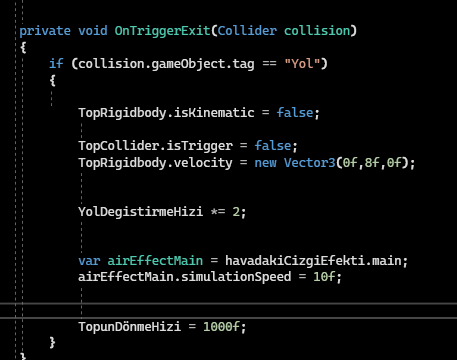
Oncollisionenter objelerin bir birbirinde değdiğinde ilk değdikleri anda 1 kere çalışır collision , unitynin hazır methodlarından biri Collisionun çalışması için Colider componentinin ve rigidbody  ekli olması gerekir. Yoldan yola atlarken birinden çıkıp diğerine girdiğinde çalışan method olarak Collider.collider.tag (isKinematic = true) yaparak topun fizik etkilerinden kurtulması sağlanır (isTrigger = true) yaparak topun objelerin içinden geçmesi açılır. aireffectMain satırında havadaki çizgi efektinin hızının azaltılması sağlandı yerde iken hızı 10 du 4 de düşürüldü. Sacims.transform.position satırında saçılma efektinin renk ve yeri ayarlandı. Saçılma efektinin oynatılması sağlandı Topun dönme hızı ise 500 olarak ayarlandı. Her yolun sonunda topun zıplamaya başlamasını istiyorum, topun Rigidbody bileşenine erişmemiz ve ayarlarında bazı değişiklikler yapmamız gerekiyor. RigidBody'yi kinematik moddan çıkardıktan sonra Hızın Y ekseni miktarını değiştirerek, top yerden uzaklaşacak ve yukarı çıkacak ve yerçekimi kuvveti sayesinde top düşecektir. Topun çarpıştırıcısının Tetikleyici seçeneği etkin olduğunda topun diğer yola geçtiğinde istediğimiz gibi ilerlemedi bu nedenle Is Trigger seçeneği devre dışı bırakıldı ."Trigger" ve "Is kinematic mode'u aktif hale getirildi.

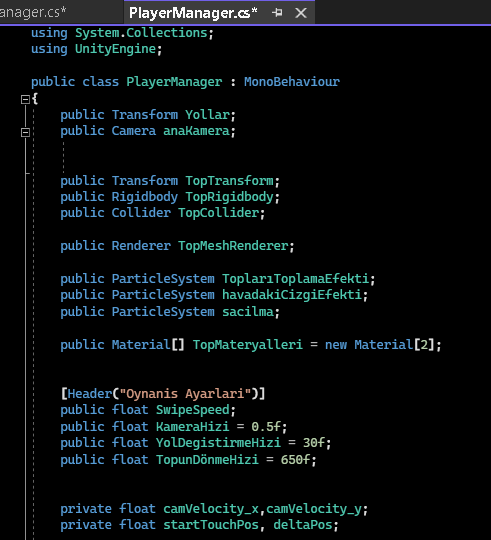
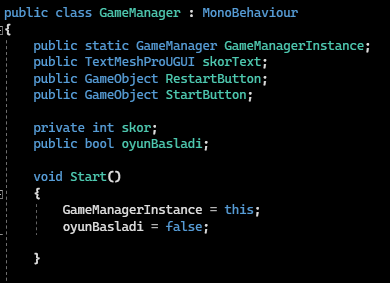


Siyah top (Oyuncu topumuz) rastgele bir topla çarpıştığında, çarpışan topun rengini emecek, bu nedenle ikinci topun Renderer bileşenine erişerek onu Oyuncu topuna atama işleminin yapılması sağlandı. SetActive(False) ile değdiği objenin kapatılması sağlandı Topumuz çarpışacağı renkli top objenin rengini alsın Toplanınca efekt oluşup renginin değiştiği yer ancak pozisyonu aynı olsun rotasyonu değişmesin istediğimiz için Kodumuzda transform.position diyerek bağlı olduğu obje ile aynı pozisyonda olmasını ve Quaternion.identity diyerek rotasyon bilgisinin değişmemesini belirtmiş olduk. Topumuz istenilen özellikleri yaptıysa skorumuza 1 puan eklenmesi sağlandı.

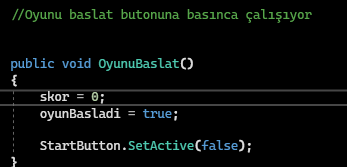


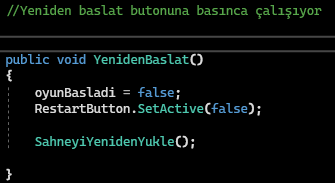
Tüm renkler için switch case içinde aynı işlem yapıldı.

Topun hızının arttığını göstermek için top zıplarken hava etkisinin daha hızlı olmasını istiyoruz. OnTriggerEnter objelerin bir birbirinde değdiğinde ilk değdikleri anda 1 kere çalışır , unitynin hazır methodlarından biri Collision ile arasındaki fark collision obje içine girmesine izin vermez iken triggerda içine girip çıkabilir topun (isKinematic = false) yaparak fizikten etkilenmesi sağlandı topun (isTrigger = false) yaparak objelerin içinden geçmesi kapatılır ileriye gitme hızı 2 katına çıkarılır havadaki çizgi efektinin hızının arttırılması sağlandı .havada iken hızı 4 dü yere inince 10 a çıkarıldı Topun dönme hızı değiştirildi.

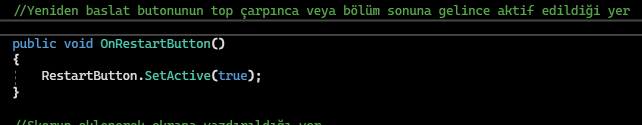
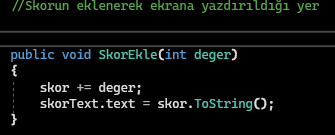


Staj hocamın gözetiminde projeye devam edilip kontrolleri yapıldı. Sorunlara çözüm yolu bulundu. Takıldığım noktalarda destek alarak sorularım cevaplandı.

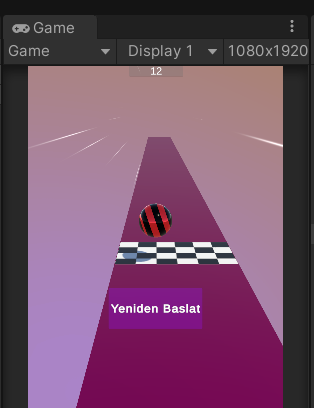
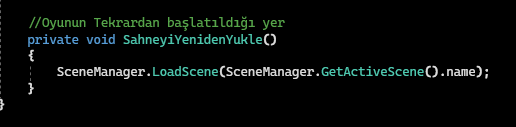


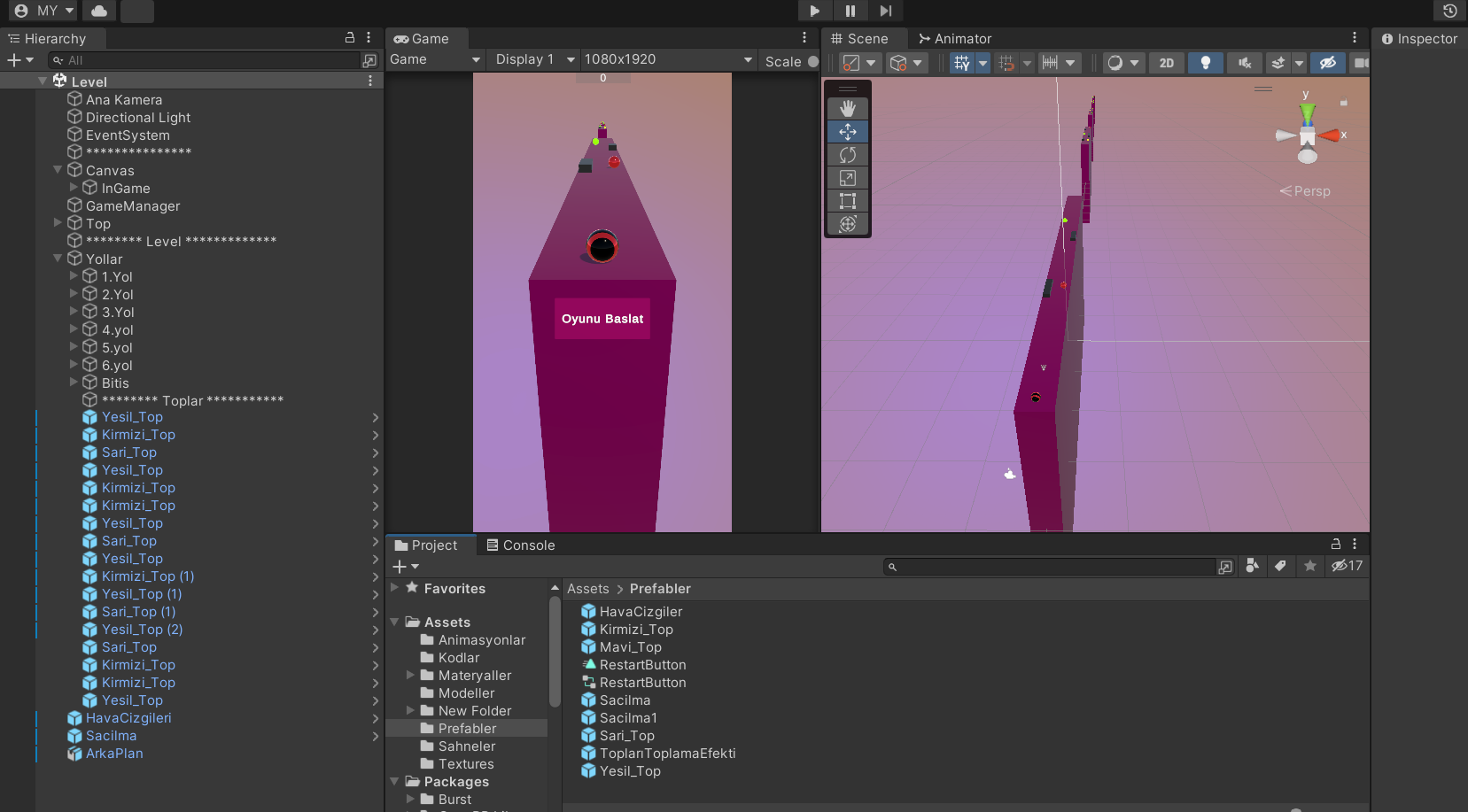
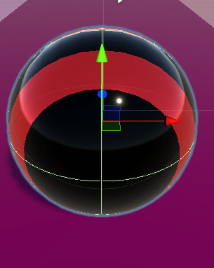












Scriptlerde mekanikleri uyguladıktan sonra her scripti kendine özel objelere yerleştirerek sahne prototipinde denemesi yapılmıştır. Daha sonrasında ise oyunu prototip olarak oynanabilecek halde olduğundan dolayı UI düzenlemesi yapılmaya başlanmıştır. Oyunun UI ve modellemelerini yaptıktan sonra oyunun test ve hata aşamalarına da bakılarak proje sonlanmıştır.

