ERCİYES ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

PROGRAMMING LABORATORY NOTLARI

8. HAFTA NOTU (Unity İle Oyun Programlama)

Bu not dersin 8. haftasında yapacağınız uygulamaya hazırlanmanızı sağlayacak olup temel kavramları anlatan bilgileri içermektedir.

DERSİN ÖĞRETİM ÜYESİ

Prof. Dr. Bahriye AKAY

DENEYİ YAPTIRAN ÖĞRETİM ELEMANI

Arş. Gör. Sezgin Güven Sarı

2026
KAYSERİ

1. DENEYDE KULLANILACAK KÜTÜPHANELER VE FONKSİYONLAR HAKKINDA KISA BİLGİ

1.1. UnityEngine

Unity ile geliştirilen tüm temel oyun mekanikleri büyük ölçüde `UnityEngine` isim alanı altında yer alan sınıflar aracılığıyla gerçekleştirilir. Bu kütüphane; oyun nesnelerinin oluşturulması, sahne üzerindeki konumlarının yönetilmesi, fizik etkileşimleri, çarpışmalar, materyaller, renk işlemleri ve kullanıcı girdileri gibi oyun geliştirme sürecinin temel bileşenlerini içerir. Bir Unity scripti yazılırken çoğu zaman en sık kullanılan yapıların tamamı bu kütüphane üzerinden erişilebilir durumdadır.

1.2. MonoBehaviour

Unity'de yazılan scriptler genellikle `MonoBehaviour` sınıfından türetilir. Bunun temel sebebi, Unity'nin oyun döngüsüne katılabilmek ve belirli olay fonksiyonlarını kullanabilmektir. Örneğin `Start()`, `Update()`, `OnCollisionEnter()` veya `OnTriggerEnter()` gibi fonksiyonlar ancak script bir `MonoBehaviour` sınıfı olarak tanımlandığında etkin biçimde çalışır. Bu yapı, yazılan kodun bir GameObject üzerine eklenmesini ve o nesneye davranış kazandırılmasını sağlar.

1.3. Input Sınıfı (Eski Input Sistemi)

Unity'nin klasik giriş sistemi olan `Input` sınıfı, klavye ve fare gibi donanımlardan gelen verilerin okunmasını sağlar. Özellikle başlangıç seviyesinde kullanıcıdan veri almak ve nesne hareketini kontrol etmek için sıkça kullanılır.
Bu yapı sayesinde:

- Belirli bir tuşun basılı olup olmadığı kontrol edilebilir,
- Tuşa yeni basılıp basılmadığı tespit edilebilir,
- Gerçek zamanlı hareket sistemleri geliştirilebilir.

Örneğin bir karakterin klavye ile sağa, sola, yukarı veya aşağı hareket ettirilmesi bu yapı ile mümkündür. Laboratuvar çalışmasında kullanılacak klavye kontrollü nesne mantığının temelini bu sınıf oluşturacaktır.

1.4. Transform Bileşeni

Unity'de her GameObject mutlaka bir `Transform` bileşenine sahiptir. Bu bileşen, nesnenin sahne üzerindeki konumunu (position), dönüşünü (rotation) ve ölçeğini (scale) tutar. Bir nesnenin hareket ettirilmesi, ekranda farklı bir noktaya taşınması ya da yönünün değiştirilmesi gibi işlemler doğrudan `Transform` üzerinden yapılır. Oyun geliştirme sürecinde en temel işlemlerden biri nesnenin konumunu değiştirmek olduğundan, bu bileşen merkezi öneme sahiptir.

1.5. Time.deltaTime

Unity'de hareket sistemleri geliştirilirken önemli kavramlardan biri de zaman bağımsızlıktır. Farklı bilgisayarlarda veya farklı FPS değerlerinde oyun nesnelerinin tutarlı hareket etmesi gerekir. `Time.deltaTime`, bir önceki frame ile mevcut frame arasında geçen süreyi temsil eder. Bu yapı, hareketin bilgisayardan bağımsız ve dengeli olmasını sağlar. Böylece nesne bir sistemde çok hızlı, başka bir sistemde ise çok yavaş hareket etmez.

1.6. Collider Bileşeni

Bir nesnenin başka nesnelerle çarpışabilmesi ya da etkileşime girebilmesi için `Collider` bileşenine ihtiyaç vardır. Unity'de çarpışma sistemleri fizik tabanlı olarak çalışır ve sahnedeki nesnelerin birbirleriyle temasını algılamak için collider kullanılır. Küp, küre veya özel şekiller için farklı collider türleri

bulunmaktadır. Laboratuvar uygulamasında nesnenin başka objelerle temasını tespit edebilmek için bu yapı kullanılacaktır.

1.7. Rigidbody Bileşeni

`Rigidbody`, bir nesnenin Unity fizik motoru tarafından yönetilmesini sağlar. Çarpışmaların daha sağlıklı çalışabilmesi, nesnenin fizik kurallarına göre hareket edebilmesi veya çarpışma olaylarının tetiklenebilmesi için çoğu durumda bu bileşene ihtiyaç duyulur. Bu yapı nesneye ağırlık, hız, kuvvet gibi fiziksel özellikler kazandırır. Özellikle çarpışma algılamada collider ile birlikte düşünülmesi gereken temel bileşenlerden biridir.

1.8. Renderer ve Material Yapısı

Bir nesnenin ekranda görünmesini sağlayan bileşenlerden biri `Renderer` bileşenidir. Bu yapı sayesinde nesnenin hangi materyalle çizileceği belirlenir. Materyal üzerinden ise renk, parlaklık ve çeşitli görsel özellikler yönetilebilir. Bir nesnenin rengini çalışma anında değiştirmek istediğimizde genellikle renderer bileşenine erişilir ve nesnenin materyal rengi güncellenir. Bu özellik, oyun içindeki görsel geri bildirim mekanizmalarının en temel örneklerinden biridir.

1.9. Random Sınıfı

Unity'de rastgelelik gerektiren işlemler için `Random` sınıfı kullanılır. Özellikle belirli bir aralıktan sayı seçmek, rastgele konum belirlemek ya da rastgele görsel özellikler üretmek için tercih edilir. Oyunlarda çeşitlilik ve tekrar hissini azaltmak için bu yapı oldukça önemlidir. Laboratuvar uygulamasında nesne renginin belirli bir olaya bağlı olarak rastgele değişmesi gibi işlemler için bu sınıf yararlı olacaktır.

1.10. Destroy() Fonksiyonu

Unity'de bir GameObject'i sahneden kaldırmak için `Destroy()` fonksiyonu kullanılır. Bu işlem sayesinde nesne oyun sırasında yok edilebilir, görünmez hale gelebilir ve artık sahneyle etkileşime girmemesi sağlanır. Bu fonksiyon genellikle bir karakterin ölmesi, bir engelin ortadan kalkması ya da tek kullanımlık bir nesnenin görevini tamamlaması gibi durumlarda tercih edilir.

1.11. CompareTag() Fonksiyonu

Unity'de nesneleri gruplandırmanın en pratik yollarından biri `Tag` sistemidir. Her nesneye belirli bir etiket atanabilir ve bu etiket üzerinden nesnenin türü veya amacı anlaşılabilir. `CompareTag()` fonksiyonu, bir nesnenin belirli bir tag'e sahip olup olmadığını kontrol etmek için kullanılır. Bu yöntem, string karşılaştırmasına göre daha güvenli ve daha performanslıdır. Laboratuvar uygulamasında hangi nesneye temas edildiğinin anlaşılmasında bu yapı önemli rol oynayacaktır.

2. DENEYDE KULLANILACAK PROGRAMLAMA YAPILARI HAKKINDA KISA BİLGİ

2.1. if-else Karar Yapıları

Programlama sürecinde belirli durumlara göre farklı işlemler yapılması gerektiğinde `if-else` yapıları kullanılır. Oyun programlamada da bu yapılar oldukça yaygındır. Örneğin oyuncu belirli bir nesneye temas ettiğinde farklı, başka bir nesneye temas ettiğinde farklı bir işlemin gerçekleştirilmesi bu karar yapıları sayesinde mümkün olur. Böylece oyun mantığı koşullara göre şekillendirilmiş olur.

2.2. Metot Kullanımı

Metotlar, belirli bir işi yerine getiren kod bloklarıdır. Oyun programlamada kodun düzenli, okunabilir ve tekrar kullanılabilir olması için işlemler metotlara ayrılır. Örneğin hareket işleminin bir metotta, çarpışma ile ilgili kontrolün başka bir metotta ele alınması kod organizasyonunu güçlendirir. Bu yaklaşım, programın daha anlaşılır olmasını ve gerektiğinde kolayca geliştirilmesini sağlar.

2.3. Bileşen Tabanlı Programlama Mantığı

Unity'de her nesne birçok bileşenden oluşur ve geliştirici ihtiyaç duyduğu bileşene erişerek işlem yapar. Örneğin bir nesnenin rengi değiştirilecekse `Renderer`, fiziksel durumu kontrol edilecekse `Rigidbody`, konumu değiştirilecekse `Transform` bileşenine erişilir. Bu yaklaşım, Unity'nin temel mimarisidir ve oyun programlamada nesnelerin modüler biçimde kontrol edilmesini sağlar.

2.4. GetComponent Yapısı

Bir GameObject üzerinde bulunan başka bir bileşene erişmek için `GetComponent<T>()` yapısı kullanılır. Bu yöntem, script içinde ihtiyaç duyulan bileşene ulaşmayı sağlar. Örneğin bir nesnenin materyalini değiştirmek için `Renderer` bileşenine, fizik hareketi uygulamak için `Rigidbody` bileşenine bu yöntemle erişilebilir. Ancak gereksiz tekrarlar performans sorunlarına yol açabileceğinden, bileşen erişimi dikkatli planlanmalıdır.

2.5. Olay Tabanlı Programlama

Unity'de bazı işlemler sürekli çalışan fonksiyonlar içerisinde, bazıları ise belirli olaylar gerçekleştiğinde çalışır. Örneğin `Update()` her frame'de çalışırken, `OnCollisionEnter()` yalnızca bir çarpışma gerçekleştiğinde tetiklenir. Bu yapı olay tabanlı programlama yaklaşımının bir örneğidir. Oyun geliştirme sürecinde kullanıcı girdileri, çarpışmalar, sahne değişimleri ve buton tıklamaları gibi pek çok mekanizma bu mantıkla yönetilir.

2.6. Nesne Karşılaştırma ve Durum Kontrolü

Oyun içinde hangi nesneyle etkileşim kurulduğunu anlamak çoğu zaman programın doğru davranması açısından kritiktir. Bu nedenle nesnelerin isimlerine, türlerine veya tag bilgilerine göre karşılaştırmalar yapılır. Ancak isim karşılaştırmaları değişebilir olduğundan, tag yapısı daha güvenilir bir yöntem sunar. Bu sayede program, sahnedeki nesneleri belirli rollerine göre ayırt edebilir.

2.7. Rastgelelik ve Çeşitlilik Mantığı

Oyunlarda her durumun tamamen sabit olması çoğu zaman tekdüzelik oluşturur. Bu nedenle rastgelelik mekanizmaları görsel veya davranışsal çeşitlilik sağlar. Bir nesnenin renginin her temas sonrası farklı olması, oyuncuya dinamik bir geri bildirim

sunar. Böylece oyun daha canlı ve etkileşimli bir hale gelir. Bu tür durumlar programlama açısından rastgele sayı üretimi ve bu sayıların uygun mantıkla kullanılmasını gerektirir.

3. UNİTY'DE GİRDİ, HAREKET VE NESNE ETKİLEŞİMİ HAKKINDA BİLGİ

3.1. Klavye Girdisinin Okunması

Bir oyun nesnesinin kullanıcı tarafından kontrol edilebilmesi için önce klavyeden gelen verilerin doğru şekilde okunması gerekir. Unity'nin eski input sisteminde bu işlem `Input` sınıfı ile gerçekleştirilir. Kullanıcının bir tuşa basması, oyunun o anda bir komut alması anlamına gelir. Bu komut bir hareket olabilir, bir nesnenin yönünün değişmesi olabilir veya özel bir davranışın tetiklenmesi olabilir. Özellikle başlangıç seviyesinde klavye girdisi okumak, oyun programlamaya girişte en temel konulardan biridir.

3.2. Frame Mantığı ve Update() Fonksiyonu

Unity oyunları frame tabanlı çalışır. Yani oyun çalıştığı sürece ekranda arka arkaya kareler üretilir ve her karede belirli güncellemeler yapılır. `Update()` fonksiyonu her frame'de bir kez çalışır. Bu nedenle, kullanıcının bastığı tuşları okumak ve nesnenin konumunu sürekli güncellemek için en sık kullanılan fonksiyonlardan biridir. Hareket gibi gerçek zamanlı sistemler genellikle burada yönetilir.

3.3. Nesne Hareketinin Programlanması

Bir nesnenin hareket etmesi, temel olarak konum bilgisinin değiştirilmesi anlamına gelir. Bu işlem doğrudan `Transform` üzerinden yapılabilir. Nesnenin hangi yönde ve ne hızla ilerleyeceği ise kullanıcının girdisine bağlı olarak belirlenir. Burada dikkat edilmesi gereken temel konu, hareketin sabit ve kontrollü olmasıdır. Aksi halde nesne çok hızlı, dengesiz veya sistemden sisteme farklı davranış gösteren bir yapıya dönüşebilir.

3.4. Çarpışma Algılama Mantığı

Oyun dünyasında nesnelerin birbirleriyle etkileşime girmesi çoğu zaman çarpışma sistemi ile sağlanır. Bir oyuncu nesnesinin başka bir objeye değmesi, oyun mantığında önemli sonuçlar doğurabilir. Bu nedenle Unity'de çarpışma sistemi fizik motoru ile bütünleşik biçimde çalışır. Çarpışma algılamanın doğru çalışması için collider ve çoğu durumda rigidbody bileşenlerinin uygun şekilde kullanılması gerekir. Bu sayede nesnelerin birbirine temas edip etmediği algılanabilir.

3.5. Tag Kavramı ile Nesneleri Ayırt Etme

Bir oyunda sahne üzerinde birçok nesne olabilir ve her biri farklı anlam taşıyabilir. Bazı nesneler ödül niteliğinde olabilirken, bazıları tehlike oluşturabilir. Programın bu nesneleri ayırt edebilmesi için tag sistemi kullanılır. Örneğin aynı çarpışma fonksiyonu içinde, temas edilen nesnenin türüne göre farklı işlemler yapılabilir. Bu yaklaşım, sahne yönetimini ve etkileşim tasarımını oldukça kolaylaştırır.

3.6. CompareTag Kullanımının Önemi

Bir nesnenin tag bilgisini kontrol etmek için `CompareTag()` kullanılması önerilir. Bunun sebebi hem daha güvenli olması hem de Unity tarafından optimize edilmiş bir yöntem sunmasıdır. Oyun geliştirme sürecinde sıkça yapılan hatalardan biri tag adını doğrudan string karşılaştırmasıyla kontrol etmeye çalışmaktır. Ancak bu yaklaşım yazım hatalarına ve gereksiz risklere açıktır. `CompareTag()` bu problemi

azaltır ve daha temiz bir yapı kurulmasına yardımcı olur.

3.7. Renk Değişimi ve Görsel Geri Bildirim

Oyunlarda oyuncuya sistemin tepki verdiğini göstermek için görsel geri bildirim mekanizmaları kullanılır. Bir nesnenin renginin değişmesi de bunun en temel örneklerinden biridir. Özellikle belirli bir etkileşim sonrasında nesnenin farklı bir görünüme kavuşması, kullanıcıya "bir olay gerçekleşti" bilgisini anında verir. Bu uygulamalarda genellikle renderer ve materyal üzerinden çalışılır. Renk değerleri sabit olabileceği gibi rastgele de üretilebilir.

3.8. Rastgele Renk Mantığı

Bir nesnenin her temas sonrası farklı bir renge dönüşmesi için rastgele sayı üretimi kullanılır. Bu yaklaşım programlama mantığı açısından önemli bir örnektir; çünkü üretilen sayıların belirli bir görsel sonuca dönüştürülmesi gerekir. Rastgelelik, yalnızca teknik bir konu değil, aynı zamanda oyun tasarımında tekrar hissini azaltan önemli bir unsurdur. Küçük ölçekli laboratuvar çalışmalarında bile bu yapı, öğrenciye oyunların dinamik yapısını kavratır.

3.9. Destroy ile Nesnenin Yok Edilmesi

Bir nesnenin belirli bir etkileşim sonucunda sahneden kaldırılması, oyun mantığında oldukça sık kullanılan bir işlemdir. Özellikle tehlikeli bir engele temas edildiğinde oyuncu nesnesinin yok olması, oyun sonu veya başarısızlık durumlarını modellemek için yaygın bir örnektir. `Destroy()` fonksiyonu burada temel araçtır. Bu yapı sayesinde öğrenci, oyun dünyasında nesne yaşam döngüsünün nasıl kontrol edildiğini anlamaya başlar.

3.10. Oyun Mantığında Durumlara Göre Davranış Geliştirme

Bu laboratuvarın en önemli yönlerinden biri, tek bir nesnenin farklı durumlara göre farklı tepkiler vermesidir. Örneğin aynı hareket eden küp, bir nesneye temas ettiğinde rengini değiştirebilir; başka bir nesneye temas ettiğinde ise tamamen yok olabilir. Bu yaklaşım, oyun programlamanın temel mantığını yansıtır: aynı sistem, farklı koşullara göre farklı sonuçlar üretir. Bu nedenle öğrencinin yalnızca kod yazmayı değil, aynı zamanda oyun mantığı kurmayı da öğrenmesi hedeflenmektedir.

4. DERS KATKILARI

Bu uygulama kapsamında kullanılan yapılar, öğrencinin aşağıdaki teknik kazanımları edinmesini sağlar:

4.1. Algoritmik Düşünme ve Mantıksal Kurgu

Öğrenci, kullanıcı girdisini okuyarak buna uygun hareket davranışı geliştirmeyi öğrenir. Ayrıca nesneler arası etkileşimlerde hangi durumda hangi işlemin yapılacağını planlayarak koşullu düşünme becerisi kazanır.

4.2. Gerçek Zamanlı Girdi Yönetimi

Klavye üzerinden alınan verilerin frame tabanlı olarak okunması, öğrencinin gerçek zamanlı sistem mantığını kavramasına katkı sağlar. Böylece oyun programlamada kullanıcı ile sistem arasındaki doğrudan ilişki anlaşılmış olur.

4.3. Çarpışma ve Etkileşim Mekanikleri

Öğrenci, oyun nesneleri arasındaki fiziksel veya mantıksal temasın nasıl algılandığını öğrenir. Bu sayede oyunlarda ödül, tehlike, hedef veya engel gibi kavramların nasıl kodlandığını temel düzeyde deneyimler.

4.4. Bileşen Tabanlı Programlama Bilinci

Unity'nin bileşen tabanlı yapısı sayesinde öğrenci, bir nesnenin tek parça bir yapı olmadığını; farklı görevler üstlenen birçok bileşenden oluştuğunu fark eder. Bu durum, oyun motoru mantığının daha sistemli şekilde anlaşılmasını sağlar.

4.5. Modüler ve Okunabilir Kod Yazımı

Hareket, çarpışma kontrolü ve görsel değişim gibi işlemlerin düzenli biçimde ayrıştırılması, öğrencinin daha temiz ve sürdürülebilir kod yazma alışkanlığı kazanmasına yardımcı olur. Bu yaklaşım, ilerleyen haftalarda geliştirilecek daha büyük projeler için temel oluşturur.

4.6. Görsel Geri Bildirim ve Oyun Tasarımı İlişkisi

Bir nesnenin renginin değişmesi veya sahneden kaldırılması gibi işlemler, yalnızca teknik değil aynı zamanda tasarımsal sonuçlar da doğurur. Böylece öğrenci, oyun programlamada teknik yapı ile kullanıcı deneyimi arasındaki ilişkiyi fark etmeye başlar.

4.7. Problem Çözme ve Hata Ayıklama Yetisi

Bu tür uygulamalarda öğrenciler çoğu zaman giriş okumama, çarpışmanın çalışmaması, tag eşleşmemesi veya rengin değişmemesi gibi sorunlarla karşılaşabilir. Bu süreç, hata ayıklama alışkanlığı kazandırır ve öğrencinin sistematik düşünme becerisini geliştirir.