Unity İçin Notlar

▼ *Componentler*

Transform: Bir objenin 3 boyutlu uzaydaki konumunun, eğiminin (`Rotation`) ve boyutunun (`Scale`) depolandığı, tüm GameObject'lerde ortak olan Component

RigidBody: Objenin fiziksel etkileşimlere girmesini sağlayan Component. (`yerçekimi`)

Collider: Bir objenin fiziksel olarak temas edilebilir olan kısımlarını ayarlayan Component. Bu component'teki "`Is Trigger`" seçeneği işaretlenirse, başka bir obje bu objeye temas edince durmaz, içinden geçer ve kodda "`OnCollision`" fonksiyonları yerine "`OnTrigger`" fonksiyonları çağrılır.

AudioListener: Bir sahnede sadece 1 tane olmalıdır. Oyun mekanındaki tüm ses dosyaları, bu objeye olan uzaklıklarına göre dinlenir. Bir başka deyişle AudioListener component'ine sahip olan obje, oyuncunun oyundaki kulağıdır.

Mesh Renderer: Objeyi sahnede çizdirmeye, yani onu görünür kılmaya yarayan Component. Eğer başındaki tik işareti kaldırılırsa obje görünmez olur.

Trail Renderer: Objenin hareketi esnasında arkasından bir şerit çıkmasını sağlar. Materyal olarak genelde Partikül materyalleri kullanılır.

AudioSource: Bir objeyi ses kaynağı olarak yetkilendirir. Objenin ses çıkarabilmesi için bu component'e sahip olması gereklidir `(aslında bu component'i kullanmadan da ses çalmaya yarayan bir fonksiyon var ancak genel anlamda ses çalmak için bu component'in kullanılmasını tavsiye ederim)`. AudioSource component'ine sahip bir obje birden çok ses çalabilir, kısıtlama yoktur.

Fixed Joint: İki objeyi birbirine bağlamaya yarar. Ancak iki obje arasında herhangi bir hareketli olay olmaz. Sadece ikisi bitişiktir. Objelerin ikisinin de

RigidBody'e sahip olması gereklidir.

Hinge Joint: İki objeyi, havada süzülen bir zincirin halkaları gibi veya bir kapının menteşeyle duvara bağlı olması gibi birbirine bağlar. Objelerin ikisinin de RigidBody'e sahip olması gereklidir.

NOT: Eğer bir obje için Mesh Collider kullanıyorsan, bu obje normal şartlarda başka Mesh Collider'a sahip objelerle temas etmez. Ancak eğer iki Mesh Collider'ın da birbiriyle temas etmesini istiyorsan, Mesh Collider'ların "Convex" değişkenini Inspector'dan işaretle.

▼ *DEĞİŞKENTİPLERİ*

`int`: Tamsayı

`float`: Virgüllü sayı

`double`: float'tan daha büyük üst ve alt limite sahip olan ve daha küsüratlı olabilen virgüllü sayı

`bool`: Sadece true veya false değerini alan, en basit değişken tipi

`string`: Yazı şeklinde değer alır

`GameObject`: Oyun sahnedeki herhangi bir obje

`AudioClip`: Ses Dosyası

`Light`: Sahne ışıkları için kullanılır.

▼ `Color`: Renk depolamaya yarar. Değeri şöyle ayarlanabilir:

```
Color renk = new Color(0.4f, 0.5f, 0.23f, 0.8f);
Color renk = Color.red;
```

İçerideki 4 adet parametrenin ilk üçü rengin RGB'si (yani kırmızı, yeşil ve mavi tonları; bilgisayarda renkler bu 3 rengin farklı oranlarda karışmasıyla oluşturulur) ve sonuncu parametre de rengin saydamlığı. Değeri 1 olursa tamamen opak, 0 olursa tamamen görünmez olur. Tüm değerler 0 ve 1 arasında olmalıdır.

`Vector3`: Uzaydaki bir noktanın koordinatlarını depolamaya yarar. Uzayda noktalar x,y ve z koordinatlarından oluşur ve bu yüzden bu değişken içinde 3 değer bulundurur: x, y ve z.

`Vector2`: 2 boyutlu düzlemde bir nokta (mesela ekranın üzerindeki bir nokta) belirtir; x ve y koordinatlarından oluşur.

`Transform`: Bir GameObject'in Transform componentini depolar.

`Texture2D`: İki boyutlu resim dosyalarını depolamak için kullanılır.

`GUISkin`: Arayüz elemanlarıyla uğraşırken (Özellikle interaktif menü yapımında) Unity'nin hazır arayüz stili yerine kendi arayüz stillerimizi kullanabilmemiz için kullanmamız gereken değişken türü. Bu değişkene uygun stili proje panelinden "`Create – GUI Skin`" yoluyla oluşturabilirsin. Ardından "`OnGUI()`" fonksiyonunda "`GUI.skin = arayuzDegiskeni;`" komutuyla arayüze kendi oluşturduğun stili atayabilirsin.

`KeyCode`: Bir klavye tuşu depolamaya yarar. Daha sonra `Input.GetKeyDown()` ve benzeri fonksiyonlara parametre olarak atanabilir.

`Touch`: Mobil platformlarda, ekrandaki bir parmağın özelliklerini depolamak için kullanılan bir veri türü. Depoladığı özellikler ise "`position`", "`deltaPosition`", "`phase`" ve "`fingerId`"dir.

`RaycastHit`: Raycast kullanımında, raycast'ın bir nesneye çarparsa geri döndüreceği bilginin depolanacağı değişken türü.

`Quaternion`: Rotasyonu depolamaya yarar. Sanıldığının aksine objelerde gerçekçi bir rotasyon elde etmek için 4 boyutlu bir değişken türü, Quaternion kullanılır ve gerçekten karmaşık bir şeydir kendisi. Neyse ki işimizi kolaylaştırmak ve rotasyonlarla içli dışlı olmak için kullanımı rahat çeşitli Quaternion fonksiyonları mevcuttur.

`Array`: Bir 'dizi' oluşturmaya yarar. Diziler, içlerinde birden çok değeri depo edebilen özel değişkenlerdir.

`List`: Array'den daha gelişmiş özelliklere sahip olan bir dizi oluşturmaya yarar. Bu dizilerin obyutu sabit değildir o yüzden içine dinamik olarak yeni değerler eklenip var olan değerler silinebilir.

`delegate`: Normal bir değişken, içerisinde bir veri tutmaya yararken; bir delegate ise içerisinde bir veya birden çok fonksiyon tutmaya yarar. İsmi çok karmaşık bir veri türüymüş gibi gösterebilir ama aslında kullanımı basittir ve doğru kullanıldığında çok faydalı olabilir.

`event`: delegate'lere çok benzer ve bu veri türü de içerisinde fonksiyon(lar) tutar. Tek fark, bir delegate'i tüm class'lar tetikleyebilirken (yani delegate'teki tüm fonksiyonların çalıştırılmasını sağlarken) bir event'i sadece içerisinde tanımlandığı script tetikleyebilir.

▼ *Fonksiyonlar*

▼ *ÖnemliFonksiyonlar*

`void Update():` Oyunun bir frame'inde (kare) gerçekleşen olayların yazıldığı fonksiyon. Saniyede defalarca çalısır.

`void FixedUpdate():` Fizik olayları için (rigidbody Component'i ile yapılan olaylar) bu fonksiyon kullanılmalıdır.

`void LateUpdate():` Tüm objelerin Update() fonksiyonları çalıştırıldıktan sonra `LateUpdate()` fonksiyonları çalıştırılır. Özellikle kameranın bir objeyi takip ettiği kodlar için kullanışlıdır (araba yarışları vb.) çünkü takip edilen obje Update()'in içerisinde hareket etmişse, LateUpdate'te kamera objenin son konumuna göre ayarlanır ve bu sayede daha istenilen tarzda bir görüntü elde edilir.

`void Awake():` Start() fonksiyonundan daha önce çalışan ve tıpkı onun gibi sadece bir kere çalıştırılan bir fonksiyon

`void Start():` Sadece script ilk çalıştırıldığında yapılacak olayların yazıldığı fonksiyon.

`void OnCollisionEnter():` Scriptin uygulandığı obje herhangi başka bir objeyle temas (collision) ederse gerçekleşir. Eğer '`void OnCollisionExit()`' yazılırsa, olaylar temas kesilince, '`void OnCollisionStay()`' yazılırsa da temasın olduğu her frame'de gerçekleşir.

`void OnTriggerEnter()`: Scriptin uygulandığı objenin Collider'inde "Is Trigger" seçeneği işaretliyse, Collider'ine bir obje temas edince bu komut çalışır. '`void OnTriggerExit()`' fonksiyonuyla temas kesilince, '`void OnTriggerStay()`' fonksiyonuyla da temas süresince yapılacakları ayarlayabilirsin. Bu fonksiyonun '`OnCollisionEnter()`'dan bir farkı, içerisine girilen argümanın türünün "Collision" değil de "Collider" olmasıdır!

▼ Örnek

```
void OnTriggerEnter(Collider temasEdilen)
{
  if(temasEdilen.gameObject.name == "Yukseltici Zemin")
  {
  GetComponent<Rigidbody>().AddForce(Vector3.up*10);
}
else if(temasEdilen.gameObject.name == "Zaman Yavaslatici Zemin")
{
  Time.timeScale = 0.5;
}
```

```
}
void OnTriggerExit(Collider cikisYapilan)
{
  if(cikisYapilan.gameObject.name == "Zaman Yavaslatici Zemin")
  {
    Time.timeScale = 1.0;
}
}
```

`void OnMouseDown():` Scriptin uygulandığı bir GUI objesine ya da bir objenin Collider'ine mouse ile tıklandığı anda gerçekleşir. Eğer '`void OnMouseDrag()`' yazılırsa mouse ile basılı tutulduğunda gerçekleşir. Veya '`void OnMouseUp()`' yazılırsa mouse ile tıklanma kesildiği anda olaylar gerçekleşir.

`void OnMouseEnter():` Scriptin uygulandığı bir GUI objesinin veya bir objenin Collider'inin üzerine mouse ile gelindiğinde gerçekleşir. Eğer '`void OnMouseExit()`' yazılırsa mouse ilgili objenin/GUI'nın üzerinden çekilince olaylar gerçekleşir. Veya '`void OnMouseOver()`' yazılırsa, mouse ilgili objenin/GUI'nin üzerinde olduğu her step'te ilgili olaylar gerçekleşir.

`void OnDrawGizmos():` Scene panelinde temsili şekiller çizmeye yarar. Örneğin Point Light'taki lamba sembolü, Collider'daki dikdörtgen gibi semboller ve şekiller bunların arasındadır. Oyuna herhangi bir etkisi yoktur, bu gizmo'lar sadece Scene panelinde gözükür ve oyun yapımcısına görsel anlamda kolaylık sağlar.

`void OnDrawGizmosSelected():` Üstteki fonksiyondan tek farkı, bu fonksiyonda çizdirilen gizmo'lar sadece obje Scene panelinde seçiliyse görünür ancak üstteki `OnDrawGizmos()` fonksiyonunda gizmo'lar obje seçili olmasa da görünür durumdadır.

`void OnBecameVisible():` Obje herhangi bir kamera tarafından görüldüğü zaman gerçekleşir. Bu kameralara Scene panelindeki gezmeye yarayan kamera da dahildir. Bu özellik performans açısından faydalı olabilmektedir. Genelde `OnBecameInvisible()` fonksiyonuyla beraber kullanılır.

`void OnBecameInvisible():` Obje hiçbir kamera tarafından görülmediği zaman gerçekleşir. Örneğin bir objedeki bir scripte sadece obje kameraların en az biri tarafından görüldüğü zaman ihtiyaç duyuluyorsa, o scriptte şöyle bir kod yazılarak performansta artış sağlanabilir:

`void OnEnable():` Script "`enabled = true;`" komutuyla aktif edildiğinde çalıştırılır. Ayrıca Start fonksiyonundan hemen önce de 1 kere çağrılır.

`void OnDisable()`: Script "`enabled = false;`" komutuyla deaktif edildiğinde çalıştırılır.

`void OnDestroy():` Script veya GameObject "`Destroy()`" komutuyla yok edildiği zaman, yok olma işlemi bitmeden önce gerçekleştirilir.

`void OnApplicationQuit()`: Oyun kapatılmadan hemen önce çalıştırılır.

`void Reset():` Bir component'in sağında yer alan dişli ikona tıklayıp Reset seçeneğini seçersen o component'teki Reset fonksiyonu çalışır.

`void OnApplicationFocus( bool focus ):` Çok kullanışlı bir fonksiyon. Eğer oyun açıkken kullanıcı Windows tuşuyla masaüstüne dönerse ya da bir uygulama açarsa, yani kısaca başka bir Windows penceresine odaklanırsa ve ardından tekrar oyuna geri dönerse çeşitli şeyler yapmak için birebirdir (Örneğin oyunu otomatik olarak pause etmek). Alttaki kodda, kullanıcı ne zaman oyundan başka bir işle meşgul olursa oyun otomatik olarak duraklatılır ve kullanıcı oyuna geri dönünce oyun eski hızıyla kaldığı yerden devam eder:

▼ $Transform Fonksiyonlar$ 1

`transform.Translate( float xDegisimi, float yDegisimi, float zDegisimi )`: Bir objenin konumunu (Inspector panelindeki Transform Componentinde yer alan Position değerlerini) değiştirmeye yarar.

`transform.Rotate( float xDegisimi, float yDegisimi, float zDegisimi ):` Bir objenin eğimini (Inspector panelindeki Transform Componentinde yer alan Rotation değerlerini) değiştirmeye yarar.

`transform.eulerAngles.x, transform.eulerAngles.y, transform.eulerAngles.z`: Bir objenin herhangi bir eksendeki rotation açısını verir. Eğer obje bir başka objenin child'ı ise genelde bu değerler Transform component'indeki değerlerle aynı çıkmaz çünkü bu değişkenler global uzaydaki eğimi vermektedir. Bu değerler kesinlikle teker teker değiştirilmemelidir. Ancak onun yerine bir objenin açısını şöyle değiştirebilirsin:
`transform.eulerAngles=new Vector3(x,y,z)`

`transform.localEulerAngles.x, transform.localEulerAngles.y, transform.localEulerAngles.z:` Bir objenin herhangi bir eksendeki rotation açısını verir. Bu değişkenler daima Transform componentindeki rotasyon değerlerini döndürür (Objenin hiyerarşisinden bağımsız bir şekilde).

`transform.TransformDirection( Vector3.right )`: Objenin kendine has (Local) X eksenindeki sağ yönü (bunu objeyi seçince Scene panelinde kırmızı ok olarak görebilirsin) depolamaya yarar.

`transform.LookAt( Transform hedefObje )`: Scriptin uygulandığı objenin, kodun içerisindeki ‘hedefobje’ isimli objeye bakacak şekilde rotation değerlerinin otomatik olarak ayarlanmasını sağlar. Örneğin bir spot lambanın sürekli oyuncuya doğru bakmasını ayarlamak için kullanılabilir. Ancak kullanım alanı oldukça geniştir. ÇOK ÖNEMLİ bir not: ‘Hedef Obje’nin bir ‘Transform’ olması lazım. Bunun için ‘Transform’ olarak tipi belirlenmiş bir değişken tanımlayıp onu ilgili oyun objesine eşitlemek yeterlidir.

`transform.SetParent( Transform parentObje, bool konumuSabitTut )`: Scriptin uygulandığı objenin parent objesini değiştirmeye yarar. Eğer konumuSabitTut değişkenine 'true' değeri verilirse, parent değişimi esnasında objenin konumu sabit kalır.

`Vector3 Input.mousePosition`: Fare imlecinin ekrandaki (ekran 2 boyutlu bir düzlemdir) konumunu verir, değeri sadece okunabilir. Eğer mouse ekranın en sol üstündeyse değeri ’ (0,0,0) ’, ekranın en sağ altındaysa değeri, ekranın genişliği ve yüksekliği olan ‘(Screen.width,Screen.height, 0)’dır. Görüldüğü gibi farenin Z konumu daima 0'dır.

▼ $Component'lerle İlgili Komutlar$

`GameObject.Find( "Obje Adı" )`: "Obje Adı" isimli objeyi bulmaya yarar. Bu kodu örneğin o objeyi bulup bir değişkende depolamak için kullanabilirsin.

`GameObject.FindWithTag("Tag Adı" )`: Girilen tag'a sahip bir objeyi bulmaya yarar.

`string gameObject.name:` Bir objenin ismini verir.

`*.enabled`: Bir componentin adı girilip yazılırsa, o componentin aktifliğini değiştirmeye yarar. Eğer değeri ‘true’ya eşitlenirse component aktif olur, ‘false’ye eşitlenirse component etkisiz hâle gelir.

`bool GetComponent<Renderer>().enabled`: Kodun uygulandığı objeyi, eğer değeri ‘false’ yapılırsa görünmez yapmaya; değeri ‘true’ yapılırsa tekrar görünür yapmaya yarar.

`Color GetComponent<Renderer>().material.color`: Kodun uygulandığı objenin materyalinin ana rengini, hâliyle objenin rengini değiştirmeye yarar. Değeri örneğin ‘Color.red’ veya ‘Color.green’ yapılabilir.

`Transform transform.parent`: İlgili objenin Parent objesini seçmeye yarar. Mesela bir evin kapısına bunu uygularsan büyük olasılıkla evin kendisini GameObject olarak elde etmiş olursun.

`Transform transform.root`: Kodun uygulandığı objenin parent'ının parent'ının parent'ını, yani en üst düzey parent'ını seçmeye yarar.

`SendMessage( "Fonksiyon Adı", Eğer varsa argümanlar )`: Bir objedeki, ismi girilen fonksiyonu; tercihe bağlı olarak değeri girilmiş argümanlarla (ilgili argümanların fonksiyonda tanımlı olması lazım) çağırmaya yarar.

`SendMessageUpwards( "Fonksiyon Adı", Eğer varsa argümanlar )`: SendMessage'dan tek farkı, bu komut kullanılınca scriptin yazıldığı objenin dışında, ayrıca o objenin tüm parent objeleri için de SendMessage komutu çalıştırılır, eğer onlarda da "Fonksiyon Adı" adında bir fonksiyon varsa bu fonksiyon onlarda da çalıştırılır.

`BroadcastMessage( "Fonksiyon Adı", Eğer varsa argümanlar )`: SendMessage'dan tek farkı, bu komut kullanılınca scriptin yazıldığı objenin dışında ayrıca o objenin tüm child objeleri için de SendMessage komutu çalıştırılır, eğer onlarda da "Fonksiyon Adı" adında bir fonksiyon varsa bu fonksiyon onlarda da çalıştırılır.

`GetComponent<ComponentTürü>()`: Bir objedeki, türü girilen component'i seçmeye yarar. Örneğin: GetComponent<Rigidbody>()

`GetComponentInChildren<ComponentTürü>()`: Bir objedeki, türü girilen component'i seçmeye yarar. Eğer objenin kendisinde component yoksa, hiyerarşik bir sıraya göre child objelerine de bakılır ve hiçbirinde bu component yoksa null döndürülür.

`AddComponent<ComponentTürü>()`: Bir objeye, türü girilen component'i eklemeye yarar. Rigidbody gibi bazı component'lerden bir objede sadece 1 tane olabilir; eğer o tarz bir component eklemeye çalışıyorsan ve o component'ten objede zaten varsa, objede hiçbir değişiklik olmaz.

`Instantiate( GameObject obje, Vecto3 pozisyon, Quaternion rotasyon)`: Girilen pozisyonda girilen rotasyona sahip bir 'obje' objesi oluşturur.

`GameObject.CreatePrimitive( PrimitiveType sekil )`: Unity'nin hazır basit şekillerinden oluşturmaya (GameObject-3D menüsü altındaki geometrik şekiller) yarar. Oluşturulan şeklin türünü "sekil" parametresi belirler. Değeri PrimitiveType.Plane (2 boyutlu düzlem), PrimitiveType.Cube (küp), PrimitiveType.Sphere (küre), PrimitiveType.Capsule (uçları yumuşatılmış silindir vari olan kapsül şekli) ve PrimitiveType.Cylinder (silindir) olabilir. Mantıken, oluşturulan objenin değeri bir değişkene atılmalıdır ve daha sonra bu değişken vasıtasıyla konumu, rotasyonu ve boyutu ayarlanmalıdır.

`Destroy( GameObject obje )`: Bir objeyi yok etmeye yarar.

▼ $Fizik - Rigidbody ile İlgili Komutlar$

`GetComponent<Rigidbody>().AddForce( Vector3 gucMiktari )`: Scriptin uygulandığı objede rigidbody varsa, fizik kurallarını yok saymadan (Translate'in aksine) o objeye belirli bir yönde istediğimiz güçte bir kuvvet uygulamak için kullanılır. Örneğin "GetComponent<Rigidbody>().AddForce(Vector3.up * 100)" komutu objeye alttan 100 birimlik güç uygular. Bu komutla uygulanan gücün yönü global koordinat sistemine göredir. Yani objenin rotasyonuna bakmaksının Vector3.right sürekli Scene panelinin sağ üstündeki x eksenini (kırmızı eksen) ifade eder.

`GetComponent<Rigidbody>().AddRelativeForce( Vector3 gucMiktari )`: GetComponent<Rigidbody>().AddForce()'tan tek farkı, gücün uygulandığı objenin rotasyonunun gücün yönünde etkin rol oynamasıdır. Yani uygulanan gücün yönü global değil de local (yerel) koordinat sistemine göre belirlenir.

`Vector3 GetComponent<Rigidbody>().velocity`: Objenin sürat değerini öğrenmeye veya bu değeri değiştirmeye yarar. Mesela karakteri zıplatırken kullanılabilir.

`GetComponent<Rigidbody>().AddTorque( Vector3 torkMiktari )`: Objeyi fizik motoru vasıtasıyla döndürmeye (tork vererek) yarar. Uygulanan tork global koordinat sistemine göredir.

`GetComponent<Rigidbody>().AddRelativeTorque( Vector3 torkMiktari )`: Girilen vektör objenin local (yerel) koordinat sistemine göre yorumlanır, yani objenin rotasyonu torkun yönünde etkilidir.

`Physics.Raycast( Vector3 baslangicKonumu, Vector3 yon, RaycastHit raycastHitDegiskeni (İsteğe bağlı), float raycastUzunlugu )`: Başlangıç konumundan ilgili yöne ilgili uzunlukta bir raycast ışını yollar ve bu raycast ışını bir objenin collider'ine temas ederse, fonksiyon "true" değerini döndürür. Eğer raycast ışını bir şeye çarpmazsa fonksiyonun döndürdüğü değer "false" olur. Ayrıca isteğe bağlı olarak, RaycastHit türünde bir değişken oluşturup onu bu fonksiyona parametre olarak vererek, raycast ışınının temas ettiği obje hakkında çok detaylı bilgiler alabilirsin. Örneğin objeyle raycast'in başlangıç noktası arasındaki uzaklık, objenin adı vb. Ayrıca objenin "gameObject"i sayesinde onun üzerinde istediğin değişikliği yapabilirsin

`Physics.Linecast( Vector3 baslangicKonumu, Vector3 bitisKonumu, RaycastHit raycastHitDegiskeni )`: Physics.Raycast'e çok benzer. Ondan farkı ise temas olup olmadığına bakan ışının bu komutta baslangicKonumu ile bitisKonumu

arasında oluşturulmasıdır. Yani ışının uzunluğunu ve yönünü belirtmeye gerek yoktur.

`Physics.IgnoreCollision(Collider collider1, Collider collider2, bool durum)`: Fizik motorunun "collider1" değişkeninde depolanmış obje ile "collider2" değişkeninde depolanmış obje arasındaki temasları ihmal etmesini (Eğer "durum" true yapılırsa) sağlar, yani bu 2 obje arasında herhangi bir fiziksel olay gerçekleşmez. Üçüncü parametresi "false" yapılırsa bu 2 obje arasındaki olası etkileşimlerde fizik motoru tekrar devreye girer. Örneğin "Physics.IgnoreCollision(collider, kursunCollider, true);" komutu, bir kurşunun fırlatıldığı silahla temas edip istenmeyen görüntülere sebep olmasını engellemek için birebirdir.

`Physics.OverlapSphere( Vector3 kureKonum, float kureYaricap )`: Girilen Vector3 konumunu merkez kabul eden ve kureYaricap kadar yarıçapa sahip zahiri bir kürenin içinde kalan veya onunla temas eden, Collider componentine sahip tüm GameObjectleri Collider[] türünde bir array olarak döndürür. Patlama gibi fiziki unsurlar için oldukça kullanışlıdır.

`GetComponent<Rigidbody>().AddExplosionForce( float gucMiktari, Vector3 patlamaKonumu, float patlamaYaricapi, float dikeyEkstraGuc )`: Gerçekçi patlama olayları için kullanılır. Obje eğer girilen Vector3 şeklindeki patlamaKonumu'nda oluşan, patlamaYaricapi kadar yarıçapa sahip gucMiktari şiddetindeki bir patlamanın menzilindeyse, Unity ona uygun bir doğrultuda uygun miktarda bir güç (Force) uygular. dikeyEkstraGuc değişkeni ise ne kadar büyürse, patlama menzilindeki obje o kadar havaya fırlatılır. Eğer değeri 0 yapılırsa, dikey doğrultuda ekstra bir güç uygulanmaz.

`GetComponent<Rigidbody>().MovePosition( Vector3 yeniKonum )`: Rigidbody'e sahip objeyi yeniKonum'a hareket ettirir ve bu esnada fizik olaylarını dikkate alır. Örneğin eğer yol üzerinde bir başka fizik objesi varsa o obje ile fiziksel bir etkileşim gerçekleşir.

`GetComponent<Rigidbody>().MoveRotation( Quaternion yeniEgim )`: Rigidbody'e sahip objenin eğimini değiştirir ve bu esnada fizik olaylarını dikkate alır. Örneğin obje bir küreyse ve kürenin tam üzerinde rigidbody'e sahip bir başka obje dengede duruyor ise, küreyi bu fonksiyon ile döndürünce kürenin üzerindeki objenin dengesi bozulur ve obje aşağı düşer.

`float raycastHitDeğişkeni.distance`: Raycast için eğer bir RaycastHit değişkeni belirlenmişse, onun ismi ve ardından ".distance" yazılır. Raycast eğer bir objeye temas etmişse, o objeyle raycast'ın başlangıç noktası arasındaki uzaklığı verir.

`Vector3 raycastHitDeğişkeni.point`: Raycast eğer bir objeye temas etmişse, bu temasın 3 boyutlu uzayda tam olarak hangi koordinatlarda gerçekleştiğini depolar.

`Collider raycastHitDeğişkeni.collider`: Raycast eğer bir objeye temas etmişse, o temas edilen objenin Collider component'ine erişmek için kullanılır.

$Eğimle İlgili Komutlar (QUATERNION)$

`Quaternion.FromToRotation( Vector3 birinciDogrultu, Vector3 ikinciDogrultu )`: birinciDogrultu ile ikinciDogrultu arasında yer alan rotasyonu döndürür. Örneğin bir objenin tepesinin (y ekseninin) sürekli Main Camera'nın baktığı yönde bakması için şu kod kullanılabilir:

`Quaternion.AngleAxis( float dereceMiktari, Vector3 yon )`: yonVektoru'nun etrafında dereceMiktari kadar döndürülmüş bir rotasyon döndürür. Örneğin "Quaternion.AngleAxis(30, Vector3.up)" komutu eulerAngles'ı (0,30,0) olan bir Quaternion (rotasyon) döndürür.

`Quaternion.LookRotation( Vector3 dogrultu )`: Bir doğrultu yönünde bakan bir rotasyon döndürür. Bir objenin sürekli başka bir objeye bakması için birebirdir.

$Klavye - Mouse Etkileşimi$

`Input.GetAxis( "Axis Adı" )`: İçerisine axis adı olarak genelde "Horizontal" ya da "Vertical" yazılır. Örneğin "Vertical" yazılmışsa, yukarı ok tuşuna basıldığında değeri 1, geri ok tuşuna basıldığında -1, hiçbirine basılmadığında 0 olur. Ancak sol veya sağ ok tuşuna basarsan bir şey olmaz. Sol ve sağ tuşları için "Horizontal" kullanmalısın. Hareket scriptlerinde kullanılabilecek çok pratik bir koddur.

`Input.GetAxisRaw( "Axis Adı" )`: Input.GetAxis()'ten tek farkı, bu komutta yumuşatma işlemi uygulanmamasıdır. Yani mesela "Input.GetAxisRaw("Vertical");" komutu, yukarı ok tuşuna basıldığı anda 1 döndürür.

`Input.GetButton( "Önceden Belirtilmiş Tuşun Özel Adı" )`: İlgili tuşa basılı tutulup tutulmadığını test eder. Eğer ‘Input.GetButtonDown’ kodu kullanılırsa, sadece tuşa basıldığı ilk anda olaylar gerçekleşir. ‘Input.GetButtonUp’ kodu kullanılırsa da olaylar tuştan elimizi çekince gerçekleşir. Örnek bir tuş adı: ‘Fire1’, mousenin sol tuşu veya CTRL tuşu yerine geçer. Buraya girilebilecek tuş isimleri ve onların hangi butonlara ayarlandığı ‘Edit – Project Settings – Input'tan görülebilir.

`Input.GetKey( KeyCode tusKodu )`**: Bunun ‘*Input.GetButton()*’dan farkı, bu scriptte kullanabileceğimiz tuşların ‘*Input*’ kısmında önceden tanımlanmış olan tuşlarla sınırlı olmaması. Bu script ile klavyedeki tüm tuşlar için işlem yapılabilir. İşlem, ilgili tuşa basılı tutulduğu sürece gerçekleşir. Eğer ‘*Input.GetKeyDown*’ kullansaydık tuşa bastığımız anda, ‘*Input.GetKeyUp*’ kullansaydık tuştan elimizi çektiğimiz anda gerçekleşirdi.**

`Input.GetMouseButton( int mouseTuşu )`: Mousenin bir tuşuna basılı tutulup tutulmadığını kontrol eder. Eğer ‘Input.GetMouseButtonDown’ kodu kullanılırsa sadece ilgili tuşa basıldığında, ‘Input.GetMouseButtonUp’ kullanılırsa da sadece o tuştan el çekilince ilgili olaylar gerçekleşir. 3 adet tanımlı tuş bulunur. Eğer değeri ‘0’ girilirse sol mouse tuşu, ‘1’ girilirse sağ mouse tuşu, ‘2’ girilirse de orta mouse tuşu ele alınır.

`bool Input.anyKeyDown`: Mevcut karede (frame) herhangi bir mouse ya da klavye tuşuna basılıp basılmadığını döndürür. Daha önceki bir kareden basılı tutulan bir tuş true döndürmez, tuşa mevcut frame'de basılması gereklidir.

`bool Input.anyKeyDown`: Mevcut karede (frame) herhangi bir mouse ya da klavye tuşuna basılıp basılmadığını döndürür. Daha önceki bir kareden basılı tutulan bir tuş true döndürmez, tuşa mevcut frame'de basılması gereklidir.

▼ $Ses - MüzikKomutları$

`GetComponent<AudioSource>().PlayOneShot( AudioClip sesDosyasi )`: Bir ses dosyasını, tek bir kere çalmayı sağlar.

`GetComponent<AudioSource>().Play()`: Eğer ki objenin ‘Audio Source’ componentine bir ses dosyası hâli hazırda atanmışsa, bu komut o ses dosyasını çalmaya yarar. Eğer Audio Source başka bir ses çalmakta ise, çalmakta olan ses otomatik olarak durdurulur (PlayOneShot'ta bu durum söz konusu değildir). Bu fonksiyon şöyle de kullanılabilir:

`GetComponent<AudioSource>().Pause()`: Scriptin yazıldığı objede çalınan ses dosyasını duraklatmaya yarar. Daha sonra ‘`GetComponent<AudioSource>().Play()`’ komutuyla çalmaya devam edilebilinir.

`GetComponent<AudioSource>().Stop()`: Scriptin yazıldığı objede çalınan ses dosyasını durdurmaya yarar.

`GetComponent<AudioSource>().PlayClipAtPoint( AudioClip sesDosyasi, Vector3 calinacagiKonum )`: Ses dosyasını, uzayda herhangi bir konumda çaldırmaya yarar. Bu kodun güzel yanı, kodun uygulandığı objenin "Audio Source" Component’ine sahip olmak zorunda olmamasıdır. Çünkü bu kod

gerçekleştirildiğinde sesi çaldıracak, geçici yeni bir obje, belirtilen konumlarda oluşturulur ve sesi çalıp bitirdikten sonra otomatikman kendisini yok eder.

▼ $Android - İOS Komutları$

`Touch[] Input.touches`: Ekrandaki tüm parmakları Touch türünde depolayan bir değişken.

`Vector2 Touch.position`: Parmağın ekrandaki hangi koordinata dokunduğunu döndürür.

`Vector2 Touch.position:` Parmağın ekrandaki hangi koordinata dokunduğunu döndürür.

`TouchPhase Touch.phase`: Parmağın durumunu depolar. Daha açık konuşmak gerekirse, parmağın ekrana yeni mi dokunduğunu (`TouchPhase.Began`), yoksa ekranda basılı halde hareket etmekte olan bir parmak olduğunu mu (`TouchPhase.Moved)`, ekranda basılı halde duran ama hareket etmeyen bir parmak olduğunu mu (`TouchPhase.Stationary`) ya da ekrandan yeni kaldırılan bir parmak olduğunu mu (`TouchPhase.Ended`) depolar. Ekrana aynı anda beşten fazla parmağın dokunması gibi nadir olaylarda ise değeri `TouchPhase.Canceled` olur.

`int Touch.fingerId`: O parmağa has olan bir değer döndürür. Her parmağın fingerId'si farklıdır.

`DeviceOrientation Input.deviceOrientation`: Telefonun hangi pozisyonda tutulduğunu belirtir. Yani cihazın yere dik ve düzgün bir şekilde (HOME butonu aşağıda olacak şekilde) mi (`DeviceOrientation.Portrait`), yere dik ama tepetaklak bir şekilde (HOME butonu yukarıda olacak şekilde) mi (`DeviceOrientation.PortraitUpsideDown`), yere dik ve sola yatırılmış bir şekilde mi (`DeviceOrientation.LandscapeLeft`), yere dik ve sağa yatırılmış bir şekilde mi (`DeviceOrientation.LandscapeRight`), yere paralel ve ekranı yukarı bakacak şekilde mi (`DeviceOrientation.FaceUp`) yoksa yere paralel ve ekranı aşağı bakacak şekilde mi (`DeviceOrientation.FaceDown`) tutulduğunu depolar. Eğer ki cihaz abuk subuk bir eğimle tutuluyorsa, o zaman bu değişken "`DeviceOrientation.Unknown`" değerini alır.

`Vector3 Input.acceleration`: Telefonun hareket sensörünün telefonun mevcut rotasyonuyla ilgili hesapladığı değeri döndürür.

`Handheld.Vibrate()`: Telefonu titretmeye yarar.

`ScreenOrientation Screen.orientation`: Oyunun düz ekran mı yoksa yatay ekran mı oynandığını döndürür ve değeri değiştirilerek yatay ekranla düz ekran arasında geçiş yapılabilir. Varsayılan olarak değeri `ScreenOrientation.AutoRotation`'dır yani telefonu yan döndürdükçe oyun ekranı da otomatik olarak yatay ekrana ya da dikey ekrana geçiş yapar. Bu değer, `ScreenOrientation.Portrait` ile değiştirilerek oyun düz ekran olarak, `ScreenOrientation.LandscapeLeft` ile değiştirilerek sola yatık yatay ekran olarak, `ScreenOrientation.LandscapeRight` ile değiştirilerek de sağa yatık yatay ekran olarak oynanabilir. Veya `ScreenOrientation.PortraitUpsideDown` ile değiştirilerek düz ama tepetaklak ekran olarak oynanabilir.

`string SystemInfo.deviceModel`: Cihazın modelini döndürür

`bool Application.genuine`: Uygulamanın korsan olup olmadığını döndürür. AppStore'daki ücretli uygulamaları kırıp ücretsiz olarak sunan korsanlara karşı kullanılan bir tedbirdir. Eğer `Application.genuineCheckAvailable` değişkeni true döndürüyorsa ve `Application.genuine` komutu komutu da false döndürüyorsa o zaman oyununuzun kullanıcının o an kullanmakta olduğu versiyonu büyük olasılıkla korsanlar tarafından kırılmış versiyonudur. Bu durumda çeşitli tedbirler almak isteyebilirsiniz. Bu komutu kullanırken işlemciye çok yük biner ve bu yüzden `Update()` gibi bir fonksiyonda kullanılmamalıdır!

▼ $WebcamKomutları$

`WebCamDevice[] WebCamTexture.devices`: Donanıma bağlı olan tüm webcam'leri, kameraları döndürür.

`string webcam.name`: Girilen webcam cihazının okunabilir, özgün ismini verir. Buradaki "webcam" değişkeni WebCamDevice türündedir.

`bool webcam.isFrontFacing`: Girilen webcam cihazının ön kamera olup olmadığını depolar. Yani eğer kamera kullanıcıya bakıyorsa değeri true, arkaya bakıyorsa false'dir. Buradaki "webcam" değişkeni WebCamDevice türündedir.

`WebCamTexture( string cihazAdi )`: İsmi girilen webcamin görüntüsünü anlık depolayacak olan bir texture oluşturmaya yarar. Bu bir constructor'dır ve değeri `WebCamTexture` türünde bir değişkene aktarılmalıdır. Ardından bu `WebCamTexture` herhangi bir materyale atanarak o materyale sahip objelerin kameranın görüntüsünü render etmesi sağlanabilir. Eğer ki metodun içi boş bırakılırsa (`WebCamTexture()` şeklinde), cihaza bağlı ilk webcamin

görüntüsünü depolayan bir `WebCamTexture` döndürür. cihazAdi'na atanabilecek kamera isimleri için `WebCamTexture.devices` ve webcam.name komutları kullanılabilir.

`webcamGoruntusu.Play()`: Girilen webcamGoruntusu'nün bağlı olduğu kameranın anlık görüntü vermeye başlamasını sağlar, yani kamerayı çalıştırır. webcamGoruntusu değişkeni `WebCamTexture` türünde olmalıdır.

`webcamGoruntusu.Pause()`: Girilen webcamGoruntusu'nün bağlı olduğu kameranın anlık görüntü vermeyi duraklatmasını sağlar. Webcam sonradan `webcamGoruntusu.Play()` ile çalışmaya devam ettirilebilir. webcamGoruntusu değişkeni `WebCamTexture` türünde olmalıdır.

`webcamGoruntusu.Stop()`: Girilen webcamGoruntusu'nün bağlı olduğu kameranın anlık görüntü vermeyi kesmesini sağlar. Webcam sonradan `webcamGoruntusu.Play()` ile tekrar çalıştırılabilir. webcamGoruntusu değişkeni `WebCamTexture` türünde olmalıdır.

`bool webcamGoruntusu.isPlaying`: Girilen webcamGoruntusu'nün bağlı olduğu kameranın o anda çalışıp çalışmadığını kontrol eder. webcamGoruntusu değişkeni WebCamTexture türünde olmalıdır.

`int webcamGoruntusu.videoRotationAngle`: Her kameranın kendine has bir çekim açısı olabilir. Yani çektiği görüntüyü düzleştirmek için kendi etrafında 90, 270 veya başka bir derece döndürüyor olabilir. Bu değişken de bu çekim açısını depoluyor. Bunun kullanım alanı ise çok önemli. Çünkü bu değişkeni kullanarak, kameranın çekim açısına bakmaksızın görüntünün ekranda düzgün gözükmesi sağlanabilir.

▼ $Save - LoadKomutlar_1$

`PlayerPrefs.SetInt( "Anahtar İçin İsim", int deger )`: Bir int değişkenin değerini depolamaya yarar. Buradaki "Anahtar İçin İsim", değişkenin ismiyle aynı olmak zorunda değildir. "Anahtar İçin İsim" ismi, kaydettiğimiz bu int değere sonradan ulaşmak için gereklidir.

`PlayerPrefs.GetInt( "Anahtar İçin İsim" )`: Daha önce kaydedilmiş bir int değere ulaşmak için kullanılır. Eğer girilen "Anahtar İçin İsim" isminde bir anahtar yoksa varsayılan olarak 0 döndürülür.

`PlayerPrefs.SetFloat( "Anahtar İçin İsim", float deger ):` Bir float değişkenin değerini depolamaya yarar.

`PlayerPrefs.GetFloat( "Anahtar İçin İsim" )`: Daha önce kaydedilmiş bir float değere ulaşmak için kullanılır. Eğer girilen "Anahtar İçin İsim" isminde bir

anahtar yoksa varsayılan olarak 0.0f döndürülür.

`PlayerPrefs.SetString( "Anahtar İçin İsim", "Değer" )`: Bir string değişkenin değerini depolamaya yarar.

`PlayerPrefs.GetString( "Anahtar İçin İsim" )`: Daha önce kaydedilmiş bir string değere ulaşmak için kullanılır. Eğer girilen "Anahtar İçin İsim" isminde bir anahtar yoksa varsayılan olarak "" döndürülür.

`PlayerPrefs.HasKey( "Anahtar İsmi" )`: Girilen "Anahtar İsmi"ne sahip bir anahtarın olup olmadığını, bool cinsinden döndürür.

`PlayerPrefs.DeleteKey( "Anahtar İsmi" )`: Girilen "Anahtar İsmi"ne sahip anahtarı ve değerini silmeye yarar.

`PlayerPrefs.DeleteAll()`: Şimdiye kadar oluşturulmuş tüm anahtarları ve değerlerini silmeye yarar.

▼ $GUI (Arayüz) Komutları$

`GUISkin GUI.skin:` Eğer Proje panelinde "Create – GUI Skin" yoluyla veya dışarıdan import ederek kişiselleştirilmiş bir GUI Skin oluşturmuşsanız ve bunu ‘GUISkin’ tipinde bir değişkene aktarmışsanız, o kişiselleştirilmiş arayüzü ilgili ‘OnGUI()’ fonksiyonunda kullanmanıza yarar:

```
public GUISkin arayuz;
void OnGUI()
{
GUI.skin = arayuz;
}
```

`Color GUI.color:` Arayüz elemanlarının temel rengini belirler (Butonlardaki yazı renkleri, arkaplan renkleri vb.).

`bool GUI.enabled`: Eğer değeri false yapılırsa, bu koddan sonraki tüm GUI elemanları, bunun (GUI.enabled) değeri tekrar true yapılana kadar tıklanamaz hâle gelir (eğer tıklanabilir bir GUI elementiyse) ve yarı saydamlaşır (eğer GUI elementini çevreleyen bir çerçeve varmışsa.).

`GUILayout.BeginArea ( new Rect( float baslangicXDeğeri, float baslangicYDegeri, float genislik, float yukseklik ) )`: Normal şartlarda GUI elemanları sen bu scripti yazmazsan (0,0) koordinatlarında oluşmaya başlar, yani ekranın en sol üstünde. Ancak bu kötü görünüme engel olmak için, mesela GUI elemanlarının (butonlar, kaydırılabilir çubuklar vb.) ekranın orta kısmı ile sol

kısmının tam ortasından başlamasını istiyorsan bu script ile bu GUI elemanlarını bir dikdörtgen alan içinde oluşturulmaya zorlarsın. İlk iki parametre ile dikdörtgenin ekrandaki başlangıç X ve Y değerlerini belirlerken sonraki parametrelerle dikdörtgenin genişliğini ve yüksekliğini belirlersin.

`GUILayout.EndArea()`: `GUILayout.BeginArea` ile açılmış bir GUILayout alanını kapatmaya yarar. Böylece kullanılacak yeni GUILayout komutları artık bu Layout'un içerisine yerleştirilmez.

`GUI.BeginGroup( new Rect( float x, float y, float genislik, float yukseklik ) )`: Bu komut GUI.EndGroup() ile kapatılmalıdır. X ve Y değerleri grubun ekrandaki başlangıç konumunu ifade ederken genislik ve yukseklik değerleri de grubun kapsadığı genişlik ve yükseklik değerlerini ifade eder. Bu komut ile GUI.EndGroup() arasında yer alan tüm GUI elemanları (GUILayout elemanları değil! GUI ve GUILayout farklı işleyen 2 ayrı sistem.) için artık (0,0) olan başlangıç koordinatı oyun ekranının en sol üstünü temsil etmez de GUI.BeginGroup ile açılmış grupta yer alan (X,Y) değerini temsil eder. Yani mesela GUI.BeginGroup(new Rect(100, 200, 500, 500)) ile bir grup açtıktan sonra GUI.Box(new Rect(0,0,100,100), "Merhaba") komutunu kullanırsak, üzerinde "Merhaba" yazan ilgili kutucuk ekranın (0,0) koordinatlarında değil de (100,200) koordinatlarında çizilir.

`GUILayout.BeginHorizontal():` GUILayout fonksiyonlarıyla oluşturulacak arayüz elemanlarının soldan sağa doğru dizilmesini sağlar. İşi bitince `GUILayout.EndHorizontal()` komutuyla kapatılmalıdır.

`GUILayout.BeginHorizontal():` GUILayout fonksiyonlarıyla oluşturulacak arayüz elemanlarının soldan sağa doğru dizilmesini sağlar. İşi bitince GUILayout.EndHorizontal() komutuyla kapatılmalıdır.

`GUILayout.BeginVertical():` GUILayout fonksiyonlarıyla oluşturulacak arayüz elemanlarının yukarıdan aşağıya doğru dizilmesini sağlar. İşi bitince GUILayout.EndVertical() komutuyla kapatılmalıdır.

`GUILayout.Width( float genislik )`: İçine yazılan arayüz elemanının genişliğini elle belirlemeye yarar.

`GUILayout.Height( float yukseklik ):` İçine yazılan arayüz elemanının yüksekliğini elle belirlemeye yarar:

```
 void OnGUI()
{
  GUILayout.Box("Merhaba", GUILayout.Width(150), GUILayout.Height(70));
}
```

`GUILayout.Space( float boslukMiktari ):` İki GUI elemanı arasında boslukMiktari kadar piksel boşluk bırakır.

`GUILayout.Button( "Butonun Adı" )`: Üzerinde "Butonun Adı" yazan tıklanabilir bir GUI butonu oluşturur.

`GUILayout.RepeatButton( "Butonun Adı" )`: Bu fonksiyonun GUILayout.Button'dan tek farkı, GUILayout.Button'da butona her bastığımızda olaylar sadece 1 kez gerçekleşirken bu fonksiyonda butona basılı tuttuğumuz sürece olay gerçekleşir.

`GUI.Button( new Rect( 0, 0, 200, 100 ), "Butonun Adı" ):` Ekranın en sol üstünde (0,0 koordinatlarında) 200 pixel genişliğinde ve 100 pixel yüksekliğinde bir buton çizer. Bu komutta GUILayout.Button'dan farklı olarak butonun konumunu ve boyutlarını tamamen kendimiz belirliyoruz.

`GUILayout.Box( "Kutunun adı" ):` Üzerinde "Kutunun adı" yazan bir GUI kutusu oluşturur. Kutuların butonlardan farkı; kutulara tıklanamaz ancak butonlara tıklanabilir. Ayrıca kutular görsel olarak da butonlardan biraz farklıdır. GUI.Box fonksiyonu da ayrıca mevcuttur.

`GUILayout.Label( "Yazı" ):` Ekrana "Yazı" yazdırır. Bu GUI elemanının buton ya da kutu gibi bir görseli yoktur, tıpkı GUIText gibi sadece ekrana girdiğin yazıyı yazar. Ya da GUILayout.Label(gorsel) ile Texture2D tipinde bir görseli (gorsel) ekrana çizdirebilirsin. Bu görsel çizdirme işlemi diğer GUI elemanları için de aynen geçerlidir. GUI.Label fonksiyonu da ayrıca mevcuttur.

`GUILayout.TextField( string duzenlenecekString ):` Bir String türünden değişkenin değerini arayüz elemanı vasıtasıyla, girdi olarak düzenlememizi/girmemizi sağlar. Çok kullanışlı özellik, mesela online bir oyunda kullanıcı adı girmek için kullanılabilir. Dilenirse 2. bir argüman vasıtasıyla girilen yazının olabilecek maksimum uzunluğu belirlenebilir. Örnek bir kullanımı için GUILayout.PasswordField fonksiyonunun tanıtıldığı kısma bakabilirsin. GUI.TextField fonksiyonu da ayrıca mevcuttur.

`GUILayout.TextArea( string duzenlenecekString ):` Bu fonksiyonun GUILayout.TextField'dan farkı; GUILayout.TextField'da string sadece tek bir satırdan oluşabilirken TextArea'da ise Enter tuşuyla yeni bir satıra geçebiliyoruz. GUI.TextArea fonksiyonu da ayrıca mevcuttur.

`GUI.DrawTexture( new Rect( float x, float y, float genislik, float yukseklik ), Texture textureDosyasi ):` Girilen X ve Y koordinatlarında, girilen genişlik ve

yükseklik değerlerine sahip bir dikdörtgen alanın içerisine bir textureDosyası (resim dosyası) çizer.

`GUILayout.PasswordField( string duzenlenecekParola, '*' )`: string türündeki duzenlenecekParola değişkenini, bir arayüz elemanı vasıtasıyla klavyeden girdi olarak düzenlememizi/girmemizi sağlar. Parola * şeklinde görünür. Bunu değiştirmek için, fonksiyondaki * işaretini başka bir işaretle değiştirmek yeterlidir.

`GUILayout.FlexibleSpace():` Normal şartlarda GUILayout elemanları belirli bir alanın içerisine otomatik olarak yerleştirilirler, yani her bir GUILayout elemanının ekrandaki konumunu tasarımcı tek tek girmez (Zaten üstteki GUILayout fonksiyonlarında da hiç konum bilgisi bulunmamakta.). `GUILayout.FlexibleSpace()` komutu ise, geçerli GUILayout için öngörülmüş, belirli bir yerdeki tüm alanı `(GUILayout.BeginArea()` fonksiyonuyla belirlenmiş bölgeyi. Zaten eğer `GUILayout.BeginArea()` fonksiyonunu kullanmamışsanız `GUILayout.FlexibleSpace()` komutu işe yaramaz.) doldurmaya yarar.

`GUILayout.BeginScrollView():` Bunun pek çok zaman gördüğümüz en yaygın örneği, bir oyunu veya programı yüklerken karşılaştığımız çok uzun olan ve bizim direkt "Kabul ediyorum (I accept/agree …)" diyerek geçiştirdiğimiz Kullanıcı Sözleşmeleri'dir. Tıpkı onun gibi, içinde bir yazı ve yazının sağında da onu aşağı yukarı kaydırmaya yarıyan bir scrollbar içeren bir sistemdir bu

`GUIUtility.RotateAroundPivot( float dereceMiktari, Vector2 referansNokta )`: Bu komuttan sonra yazılan GUI elemanları, Referans Nokta'ya göre 'Derece' miktarında döndürülür. Eğer bir GUI elemanını tam olduğu yerde döndürmek istiyorsan Referans Nokta olarak o GUI elemanının tam merkezindeki noktasının koordinatlarını Vector2 olarak vermelisin.

▼ $Matematiksel Komutlar$

`Random.Range( float minimumDeger, float maksimumDeger )`: İçine girilen minimum ve maksimum değerlerin arasında rasgele bir değer elde eder. Pek çok yerde kullanılabilir. Ancak önemli husus, eğer girdiğiniz değerlerin biri 'float' ise diğeri de 'float' olmalı, 'int' ise diğeri de 'int' olmalı ve minimum asla maksimumdan büyük olmamalı. Ayrıca eğer girilen değerler 'int' ise döndürülecek değer minimum değere eşit olabilirken maksimum değere eşit olmaz. Yani minimum değer bu rasgele işlemine dahilken maksimum değer değildir. Ama girilen değerler 'float' ise döndürülen değer maksimum değere de eşit olabilir.

`Random.value: "Random.Range( 0.0f, 1.0f );` " komutu ile tamamen aynıdır, 0.0 ile 1.0 arasında (Bu 2 değer de dahil olmak üzere) bir sayı döndürür.

`Vector3.Distance( Vector3 birinciKonum, Vector3 ikinciKonum )`: İki nokta arasındaki uzaklığın hesaplanmasına yarar.

`Vector3.Normalize()`: Girilen Vector3 türünden vektörün x,y ve z değerlerini, vektörün uzunluğu 1 olacak şekilde düzenler.

`Vector3.normalized`: Vector3.Normalize()'den tek farkı, bu fonksiyon girilen vektörün değerini değiştirmez, onun yerine yeni bir vektör döndürür

`Mathf.Clamp( float floatBirSayi, float minimumDeger, float maksimumDeger )`: Bir değişkenin değerinin, bir minimum ve bir maksimum olmak üzere 2 değerin arasında olmasını garantiler.

`Mathf.Clamp01( float floatBirSayi )`: Bu komut Mathf.Clamp(Float bir sayı, 0, 1) ile tamamen aynı şeyi ifade eder. İçine girilen sayının 0 ile 1'in dışına çıkmamasını sağlar.

`Mathf.Lerp( float birinciDeger, float ikinciDeger, float sayi )`: Öncelikle içine girilen "sayı" parametresinin değeri eğer 0'dan küçükse, Unity onu otomatik olarak 0 yaparken 1'den büyükse de otomatik olarak 1 yapar

Scriptin içine girilen "sayı"nın değeri ne kadar 0'a yaklaşırsa scriptin döndüreceği değer o kadar "Birinci Değer"e yaklaşır, "sayı" 0 olursa script "Birinci Değer"in kendisini döndürür. "sayı"nın değeri 1'e ne kadar yaklaşırsa scriptin döndüreceği değer de "İkinci Değer"e o kadar yaklaşır, "sayı" 1 olursa script "İkinci Değer"i döndürür. Eğer "sayı" 0.5 olursa, yaptığım açıklamalardan da çıkarılabileceği üzere "Birinci Değer" ile "İkinci Değer"in aritmetik ortalamasını döndürür.

`Mathf.InverseLerp( float birinciDeger, float ikinciDeger, float sayi ):`
Mathf.Lerp'in tam tersini yapar. Yani mesela biz Mathf.Lerp(10, 50, 0.7) deyince nasıl 38 bulmuşsak; Mathf.InverseLerp(10, 50, 38) yazarsak da bu sefer 0.7 buluruz. Yani sayı'nın Birinci Değer ile İkinci Değer arasındaki oransal olarak konumunu buluruz. Buradan da anlaşılacağı üzere döndürülen değer bir float'tır ve değeri 0 ile 1 arasındadır.

`Mathf.LerpAngle( float birinciDeger, float ikinciDeger, float sayi )`:
Mathf.Lerp ile aynı görevi yapan bir fonksiyondur. Ondan tek farkı; bu fonksiyon özellikle derece cinsinden açılarla işlem yapmak için olduğu için, eğer girilen değerlerden herhangi biri 360'dan büyükse, o değer 360'dan küçük olana kadar ondan otomatik olarak 360 çıkarılır. Ancak negatif bir

değere 360 eklenmez (aslında negatif değerlerle mümkünse pek uğraşmayın çünkü onlarla oluşan sonuçlar karmaşık olabiliyor)

`Mathf.PingPong( float birinciDeger, float ikinciDeger )`: İçine girilen birinci değerin 0 ile İkinci değer arasında kalmasını sağlar. Ancak Mathf.Clamp veya Mathf.Lerp fonksiyonlarından bariz farkı vardır. O da şudur ki; Birinci değer, İkinci değer'den büyük olmaya başladığında bu sefer döndürülen sonuç İkinci değer'den 0'a doğru azalmaya başlar ve değer 0'a ulaşınca tekrar İkinci değer'e doğru yükselmeye başlar. İşte bu yüzden adı PingPong'tur. Yine bu yüzden İkinci değer 0'dan küçük olamaz.

`Mathf.Repeat( float birinciDeger, float ikinciDeger )`: Aslında Mathf.PingPong'un neredeyse aynısı, sadece bir fark var aralarında. O da şu: PingPong fonksiyonunda Birinci değer İkinci değeri aşınca İkinci değer'den 0'a geri yönelme varken Repeat fonksiyonunda geri yönelme diye bir şey yok. Birinci değer İkinci değer'den büyük olunca direkt 0'a geri döner ve yine İkinci değer'e doğru yükselmeye başlar. Yani mesela Mathf.PingPong(4,3)=2 iken Mathf.Repeat(4,3) ise 1'e eşittir. Aslında bu fonksiyon 'Birinci değer % İkinci Değer' ifadesiyle neredeyse tamamen aynı işi yapıyor, yani bu fonksiyonu kullanmana pek gerek de yok. Ama Unity kılavuzunda bir farktan bahsediliyor ki bu farkı ben nedense göremedim

`Mathf.Abs( float birSayi ):` İçine girilen sayının mutlak değerini (Absolute) döndürür. Yani negatif bir sayı girersen onun pozitifini verir. (Mesela -12 girersen 12, -3.5 girersen 3.5, 6 girersen 6 döndürür.)

`Mathf.Sign( float birSayi )`: İçine girilen sayının işaretini döndürür. Eğer sayı 0'a eşit ya da ondan büyükse 1 döndürürken sayı 0'dan küçükse de -1 döndürür.

Mathf.ClosestPowerOfTwo(int birSayi): İçine girilen sayıya en yakın olan 2'nin kuvvetini bulur. 2'nin kuvvetleri sırayla 1-2-4-8-16-32-64-128-256-512-1024-… diye devam eder ve bu komutun içindeki sayı bunların hangisine en yakınsa o sayı döndürülür.

Mathf.IsPowerOfTwo(int birSayi): İçine girilen sayının 2'nin kuvveti olup olmadığını true ya da false olarak döndürür.

Mathf.NextPowerOfTwo(int birSayi): İçine girilen sayıdan büyük olan ve 2'nin kuvveti olan ilk sayıyı verir. Sayı zaten 2'nin kuvveti ise sayının kendisini verir.

Mathf.Approximately(float birSayi, float baskaBirSayi): İçine girilen 2 float değerin aynı olup olmadığını, true ya da false olarak döndürür. Bunu özel kılan şey ise şu ki float'larda noktadan sonrası çok net olmadığı için, mesela 1.0 == 10.0/10.0 işlemi true döndürülmezken (ki aslında bu true bir ifadedir) Mathf.Approximately(1.0, 10.0/10.0) yazdığımızda bu sefer true döndürülür.

float Mathf.PI: Pi sayısını (□) ifade eder. Değeri değiştirilemez.

float Mathf.Infinity: +□ (Sonsuz) sayısını ifade eder. Değeri değDeğeri değiştirilemez.

float Mathf.NegativeInfinity: -□ (Eksi sonsuz) sayısını ifade eder. Değeri değiştirilemez.

float Mathf.Deg2Rad: Derece cinsinden bir açıyı radyan cinsine çevirir (Yani dereceyi 2□/360 ile çarpar.). Mesela 30 derece □/6 radyana (Unity buradaki □'yi ifade etmek yerine onu kendi PI değeri ile (3.1415…) çarpar. Yani 3.14/6 □ 0.5'e yakın bir değer döndürür.) eşittir. Bu bir fonksiyon değildir, bu yüzden bir fonksiyon gibi kullanılamaz. Bu sadece *2Mathf.PI/360 demenin bir başka yoludur. Ve mesela 60 dereceyi radyana çevireceksen 60*Mathf.Deg2Rad şeklinde kullanırız.

float Mathf.Rad2Deg: Radyan cinsinden bir açıyı derece cinsine çevirir. Bu sadece 360/2*Mathf.PI demenin bir başka yoludur. Kullanımı tıpkı Deg2Rad'daki gibidir. (Mesela 0.5*Mathf.Rad2Deg gibi..)

float Mathf.Epsilon: 0'a olabilecek en yakın float tarzındaki sayıdır (Tabi bu da 'sonsuz' sayı gibi; işlemlerde kolaylık sağlaması için kullanılan, gerçekte var olmayan bir sayıdır. Lise 4 matematiğinde görülen lim x->0+ (Sıfıra sağdan gelirken limit) gibi bir şeydir.). Bunu Unity kendisi Mathf.Approximately fonksiyonunda kullanır.

Mathf.DeltaAngle(float birSayi, float baskaBirSayi): Derece cinsinden 2 açının arasındaki en küçük açıyı bulur. Mesela açılardan biri 360'tan büyükse de, 360'tan küçük olana kadar ondan otomatik olarak 360 çıkarır (Açıyı bir değişkende saklıyorsan değişkenin değerini değiştirmez, merak etme.). Burada açıların konumu önemlidir. Lisede trigonometride görüldüğü gibi burada da negatif sonuçlar döndürülebilir. Mesela Mathf.DeltaAngle(10,350) yazarsan -20 döndürür. Çünkü birim çemberde açılar saat yönünde negatif değer alırken saat yönünün tersinde pozitif değerler alıyordu. Aynen öyle de, 10 dereceyi ve 350 dereceyi birim çemberde çizersen aralarındaki en küçük açının 10'dan 350'ye saat yönünde

-20 derece olduğunu rahatça görebilirsin. Yok ama eğer Mathf.DeltaAngle(350,10) deseydik bu sefer +20 döndürürdü. Eğer ki sonucun sürekli pozitif çıkmasını istiyorsan Mathf.Abs(Mathf.DeltaAngle(float, float)) şeklinde yazabilirsin.

Mathf.Cos(float birSayi): İçine float olarak girilen radyan cinsinden açının kosinüsünü verir.

Mathf.Sin(float birSayi): İçine float olarak girilen radyan cinsinden açının sinüsünü verir.

Mathf.Tan(float birSayi): İçine float olarak girilen radyan cinsinden açının tanjant'ını verir.

Mathf.Acos(float birSayi): İçine girilen değerin radyan cinsinden arccosinüs'ünü verir.

Mathf.Asin(float birSayi): İçine girilen değerin radyan cinsinden arcsinüs'ünü verir.

Mathf.Atan(float birSayi): İçine girilen değerin radyan cinsinden arctanjant'ını verir.

Mathf.Exp(float birSayi): Doğal logaritmada da kullanılan '**e**' sayısının 'float'ıncı kuvvetini, yani **efloat değer** sonucunu verir. **e**'nin normal değeri ise yaklaşık 2.718282…'dir.

Mathf.Pow(float birinciDeger, float ikinciDeger): birinciDeger'in ikinciDeger'inci kuvvetini döndürür. Yani **birinciDegerikinciDeger** işlemini yapar.

Mathf.Sqrt(float birSayi): İçine girilen sayının karekökünü bulur.

Mathf.Log(): İki farklı kullanımı vardır: *Mathf.Log(float a, float b)* ile **loga(b)**'yi bulursun. İkinci kullanımı ise *Mathf.Log(float a)* şeklindedir. Bunda ise doğal logaritma kullanılır, yani **loge(a)** ya da bir başka deyişle **ln(a)** bulunur.

Mathf.Log10(float birSayi): *Mathf.Log(10, float* birSayi *)* ile tamamen aynı şeydir, onu kısayoludur. 10 tabanında logaritma almaya yarar (En meşhur logaritma).

Mathf.Max(En az 2 float sayı (Aralarında virgül konulmalı)): Girilen sayılar arasından en büyük olan sayıyı döndürür. Burada dikkat

edilmesi gereken husus; sayılardan biri *int* ise diğerleri de *int*, biri *float* ise diğerleri de *float* olmak zorundadır.

Mathf.Min(En az 2 float sayı (Aralarında virgül konulmalı)): Girilen sayılar arasından en küçük olan sayıyı döndürür. Burada dikkat edilmesi gereken husus; sayılardan biri *int* ise diğerleri de *int*, biri *float* ise diğerleri de *float* olmak zorundadır.

Mathf.Round(float birSayi): İçine girilen sayıyı en yakın tamsayıya yuvarlar. Özel durum olarak; eğer girilen sayının sonu "*.5*" ile biterse, o sayıya en yakın çift tamsayı döndürülür.

Mathf.RoundToInt(float birSayi): *Mathf.Round*'dan tek farkı, bu fonksiyonda döndürülen değerin *float* tipinde değil de *int* tipinde olmasıdır. Bu farkın bize etkisi ise; bir değişkene bu fonksiyonun döndürdüğü değeri atayacaksak, hata almamak için değişkenin türüne göre bu fonksiyonu ya da *Mathf.Round* fonksiyonunu kullanırız.

Mathf.Ceil(float birSayi): İçine girilen sayıyı –eğer zaten bir tamsayıya eşit değilse– yukarı yuvarlamaya yarar. Mesela içine 5.3 girersen 6, 9.0 girersen 9, 2.04 girersen 3, -4.6 girersen -4 döndürür.

Mathf.CeilToInt(float birSayi): Bunun *Mathf.Ceil*'dan tek farkı; bu komut *int* türünden sonuç döndürürken *Mathf.Ceil* ise *float* cinsinden sonuç döndürür.

Mathf.Floor(float birSayi): İçine girilen sayıyı –eğer zaten bir tamsayıya eşit değilse– aşağı yuvarlamaya yarar. Mesela 5.3 girersen 5, 9.0 girersen 9, 2.78 girersen 2, -4.6 girersen -5 döndürür.

Mathf.FloorToInt(float birSayi): Bunun *Mathf.Floor*'dan tek farkı; bu komut *int* türünden sonuç döndürürken *Mathf.Floor* ise *float* cinsinden sonuç döndürür.

▼ $Uygulamayla İlgili Komular$

UnityEngine.SceneManagement.SceneManager.LoadScene("Geçilecek Bölümün (Scene) Adı"): Başka bir scene'e (bölüm) geçmeye yarar. Ancak bu kodun işe yaraması için, geçiş yapılacak bölümün "*File – Build Settings*"deki "*Scenes In Build*" kısmına eklenmesi gereklidir.

Application.Quit(): Oyunu sonlandırmaya yarar.

Application.OpenURL("http://…./"): İlgili web sayfasını tarayıcıda açmaya yarar.

string UnityEngine.SceneManagement.SceneManager.GetActiveScene().name: En son yüklenen scene'in adını depolar ki normal şartlarda bu oyuncunun o anda bulunduğu bölümdür. Mevcut bölümün hangisi olduğunu test edip ona göre işlem yapmak için ideal bir scripttir. Değeri sadece okunabilir, değiştirilemez.

RuntimePlatform Application.platform: Oyunun oynandığı cihazın – platformun adını depolar. Değeri, oynanılan platforma göre genelde "*RuntimePlatform.WindowsPlayer*" "*RuntimePlatform.WindowsWebPlayer*" "*RuntimePlatform.Android*" "*RuntimePlatform.IPhonePlayer*" "*RuntimePlatform.FlashPlayer*" "*RuntimePlatform.OSXPlayer*" "*RuntimePlatform.OSXWebPlayer*" değerlerinden birini alır. (Bunlardan başka birkaç değer daha var.)

Application.CaptureScreenshot("Oluşturulacak resim dosyasının adı.png", int boyut): Ekranın resmini çekmeye ve onu oyunun olduğu klasöre kaydetmeye yarar. Eğer "*boyut*"un değeri 1 yapılırsa çekilen resmin boyutu oyun ekranının boyutuyla aynı olur. Resmin boyutu oyunun oynandığı ekranın boyutuyla doğru orantılıdır. Eğer "*boyut*"u yükseltirsen resmin çözünürlüğü de artar ve hatta bu esnada grafik kalitesinden hiçbir şey kaybolmaz. Mesela bu değeri 4 yaparsan ekranın çözünürlüğünün 4 katı boyutlarında bir resim kaydedilir ve normal oyuna göre çok daha kaliteli bir görsele sahip olur. Yani resim çekerken oyundaki görsellerden daha kaliteli görseller elde edebilirsin bu sayede. Ancak değer ne kadar büyürse resmin çekilme süresi o kadar uzar ve boyutu da bir o kadar artar.

string Application.dataPath: Oyunun yüklü olduğu dizini döndürür.

string Application.persistentDataPath: Oyunun save dosyalarını kaydetmek için ideal bir konum döndürür.

▼ *AnimasyonKomutları*

GetComponent<Animation>().Play("Animasyon Adı"): Objede ismi girilen animasyonun oynatılmasını sağlar. Bu animasyonlar önceden tanımlı olmalı. Bir objedeki varolan animasyonları onu seçince Inspector'daki *Animation* componentinin altındaki *Animations* sekmesinden görebilirsin.

GetComponent<Animation>().Stop(): Objede çalışmakta olan tüm animasyonları durdurmaya yarar. Veyahutta " GetComponent<Animation>

().*Stop("Animasyon Adı");*" komutu kullanılarak sadece belli bir animasyonun durdurulması da sağlanabilir.

GetComponent<Animation>().CrossFade("Animasyon Adı"): İsmi girilen animasyonu oynatmaya yarar. "*GetComponent<Animation>().Play()*"den farklı olarak, ismi girilen animasyonu direkt geçiş yapılmaz, mevcut animasyondan o animasyona yumuşak bir geçiş yapılır. Böylece animasyonlar arası keskin hareket değişikliklerinin önüne geçilmiş olur. Çok kullanışlı bir özellik yani.

GetComponent<Animation>().Blend("Animasyon Adı", int hiz): İsmi girilen animasyonla mevcut oynanmakta olan animasyonu birleştirir. Örneğin karakterin ileri ve sağa gitmek için 2 ayrı animasyonu olsun. Karakter sağ ileri giderken bu 2 animasyonu kombine etmek için bu komut kullanılabilir. Ancak öncesinde " *GetComponent<Animation>().SyncLayer()*" kullanımı önemlidir. Eğer "*hiz*" değeri 1 yapılırsa yeni animasyon gerçek hızıyla mevcut animasyona dahil olur. Ama eğer "*hiz*" değeri 0 yapılırsa ismi girilen animasyonun bu kombine animasyonla beraber oynaması durdurulur.

GetComponent<Animation>().SyncLayer(int layerNumarasi): Birden çok animasyon "*GetComponent<Animation>().Blend()*" ile kombine edilecekse bunların eş zamanlı, düzgün bir şekilde beraber oynamatılmasını sağlar. Bunun nasıl başarıldığı pek önemli değil, önemli olan işe yaraması. *Start()* fonksiyonunda kullanımı önerilir. Eğer bir animasyonun layer'ı "*GetComponent<Animation>()["Animasyonun Adı"].layer = layer;*" komutu ile değiştirilmezse, her animasyonun varsayılan başlangıç layer'ı 0'dır.

float animasyon.normalizedTime: Bir animasyonun en baştan değil de ortadan ya da başka bir konumdan başlamasını sağlar. Değeri 0 yapılırsa animasyon en baştan, 1 yapılırsa en sondan (Aynı şey), 0.5 yapılırsa ortadan başlar.

▼ $DosyaYönetimiKomutları$

Directory.Exists(string konum): İçine bir klasöre giden yol (path) girilir ve metod bu klasörün var olup olmadığını döndürür. Eğer belirtilen yolda gerçekten ilgili klasör varsa *true*, yoksa *false* döndürür.

Directory.CreateDirectory(string konum): Konumu girilen klasörü oluşturmaya yarar.

Directory.Delete(string konum , bool herSeySilinsin): Konumu girilen klasörü silmeye yarar. Eğer *herSeySilinsin true* ise klasörün içeriği de

silinirken *false* ise klasörün içeriği silinmez.

Directory.Move(string eskiKonum , string yeniKonum): Bir klasörü *eskiKonum*'dan *yeniKonum*'a taşır.

Directory.GetDirectories(string konum): Konumu girilen klasörün içerisindeki tüm alt klasörlere giden yolları bir String array'inde depolar ve döndürür.

Directory.GetFiles(string konum): Konumu girilen klasörün içindeki tüm dosyalara giden yolları bir String array'inde depolar ve döndürür.

File.Exists(string konum): İçine bir dosyaya giden yol (path) girilir ve metod dosyanın var olup olmadığını bool cinsinden döndürür.

File.WriteAllText(string konum, string icerik): Girilen konumda girilen *icerik*'e sahip bir dosya oluşturur.

File.AppendAllText(string konum, string icerik): Girilen konumdaki dosyanın mevcut içeriğinin en sonuna *icerik*'i ekler.

File.ReadAllText(string konum): Konumu girilen dosyayı açar ve içeriğini *String* cinsinden döndürür.

File.Delete(string konum): Konumu girilen dosyayı siler.

▼ $KalanKomutlar$

float color.a: Bir renk değişkeninin '*Alpha*'sını, yani saydamlığını değiştirir. Değeri 0-1 arasında olmalıdır.

float color.r – color.g – color.b: Bir renk değişkeninin RGB (Kırmızı-Yeşil-Mavi) değerlerini ayrı ayrı değiştirmeye yarar. Değerleri 0-1 arasında olmalıdır.

gameObject.SetActivey(true): Scriptin yazılı olduğu GameObject'i aktif eder.

listDegisken.RemoveAt(1): Bir List'in 2. elemanını silmeye yarar (çünkü dizilerde numaralandırma 0'dan başlar ve bu da dizinin 1. elemanına denk gelir. Yani numarası 1 olan elemanı da dizinin 2. elemanına denk gelmektedir). Bu elemandan sonra başka elemanlar varsa her birinin numarası, aradaki bu boşluğu doldurmak için, 1 düşer

listDegisken.Add(birSey): List'in en son elemanından sonra yeni bir eleman oluşturur ve ona birSey'in değerini verir (birSey'in türü, List'in türü ile aynı olmalıdır).

int listDegisken.Count: List'in kaç elemandan oluştuğunu söyler.

int listDegisken.IndexOf(birSey): List'in içerisindeki '*birSey*' elemanının, List'in kaçıncı elemanı olduğunu bulur ve *int* olarak geri döndürür. Eğer List'te '*birSey*'in değeri yoksa -1 döndürür.

GetComponent<Collider>().ClosestPointOnBounds(Vector3 birVektor): Bir objenin, girilen *Vector3* şeklindeki birVektor noktasına olan en yakın noktasını *Vector3* şeklinde döndürmeye yarar.

int Screen.width: Oyun ekranının pixel cinsinden genişliğini verir.

int Screen.height: Oyun ekranının pixel cinsinden yüksekliğini verir.

float kamera.pixelWidth: Girilen kameranın pixel cinsinden genişliğini verir. Tek kameralı oyunlarda değeri genelde Screen.width ile aynıdır. Ancak bir kamera örneğin minimap görevi görüyorsa; bir başka deyişle oyun ekranının sadece bir kısmını kapsıyorsa o zaman bu değişkenin değeri Screen.width'in değerinden farklı olur ve minimapın pixel cinsinden genişliğini döndürür.

float kamera.pixelHeight: Girilen kameranın pixel cinsinden yüksekliğini verir. (Örnek kullanımı: *Camera.main.pixelHeight*)

kamera.ViewportPointToRay(Vector3 pozisyon): Kameradan başlayan ve ekranın (pozisyon.x, pozisyon.y) koordinatlarından geçen bir ray döndürür. Bu yüzden pozisyon vektörünün z argümanı önemsizdir, ihmal edilir. Daha sonra bu ray ile Raycast yapılabilir. Ekranın en sağında pozisyon.x 1, en üstünde pozisyon.y değişkenleri 1 değerini alır. Bu döndürülen ray'i kullanarak oluşturulan Raycast'ı oluşturmanın uzun yolu ise "Physics.Raycast(Camera.main.transform.position, (Camera.main.ViewportToWorldPoint(pozisyon) - Camera.main.transform.position));" yapmaktır. Kısa yol ise "Physics.Raycast(kamera.ViewportPointToRay(pozisyon));"dur.

kamera.ScreenPointToRay(pozisyon : Vector3): ViewportPointToRay(pozisyon) komutundan sadece bir farkı var: Viewport komutunda ekranın en sağ üstü (1,1) iken bu komutta ekranın en sağ üstü (kamera.pixelWidth, kamera.pixelHeight). Yani kameranın verdiği görüntünün pixel cinsinden eni ve boyu.

bool Screen.lockCursor: Mouse'nin ekranda gözükmemesi ve olduğu yerden kımıldamaması, böylece oyunu oynarken yanlışlıkla başka yerlere tıklayarak oyun ekranının aktifliğini kaybetmemesi için güzel bir komut. Eğer "*Screen.lockCursor = true;*" yapılırsa mouse kilitlenir, "*Screen.lockCursor = false;*" yapılırsa kilit açılır.

transform.DetachChildren(): Objenin eğer varsa tüm child objelerini child'lık durumundan çıkarır, yani hiyerarşiyi kırar. Örneğin normalde

*Destroy(gameObject) :*komutu bir objeyi child objeleriyle beraber yok etmeye yarar ancak eğer child objelerin yok edilmesini istemiyorsak şöyle bir çözüm yolu bulunmaktadır

transform.IsChildOf(Transform birBaskaTransform): Bir objenin başka bir objenin child objesi olup olmadığını döndürür (Eğer child ise true, değilse false döndürür.).

Invoke("Metod İsmi", float saniyeMiktari): Script'teki ismi girilen metodu, girilen saniyeMiktari kadar saniye geçtikten sonra çalıştırmaya yarar

InvokeRepeating("Metod İsmi", float saniyeMiktari, float tekrarlamaSuresi): Script'teki ismi girilen metodu, girilen *saniyeMiktari* kadar saniye geçtikten sonra çalıştırmaya yarar. Ardından her *tekrarlanmaSuresi* kadar saniye geçtikçe, fonksiyonu tekrar çağırır.

CancelInvoke("Metod İsmi"): Kodun yazıldığı scriptte "*Metod İsmi*" isimli metod *Invoke* edilmişse bu *Invoke* işlemini iptal eder. Eğer "*Metod İsmi*" boş bırakılırsa (*CancelInvoke()* şeklinde) mevcut scriptte çalıştırılmış olan tüm *Invoke* işlemlerini iptal eder (*Invoke* edilmiş metodların isimlerine bakmaksızın).

Resources.Load<AssetTürü>("Assetin adı"): Project panelinde eğer herhangi bir yerde bir *Resources* klasörü varsa, onun içindeki bir asset ile işlem yapmaya yarar

[RequireComponent(typeof(AudioSource))]: Scriptin uygulandığı GameObject'te, scriptin çalışması için "Audio Source" component'inin

bulunmasını zorunlu kılar. Eğer yoksa kendi otomatik olarak ekler. Mesela scriptin yazıldığı objede bir ses dosyası çalınacaksa ‘Audio Source’ componenti gerekli olduğundan, bu componentin varlığından emin olmaya yarar. Bu kodda dikkat etmeniz gereken nokta, bu kodun sonuna ";" (Noktalı virgül) işareti konulmaz. Bu kod class isminin bir üst satırına yazılır.