BLM0364 Oyun Programlama

Component’ler

Transform (dönüşüm) işlemleri

- Pozisyon
- Yön
- Büyüklük

Arş. Gör. Muhammed Ömer Faruk Selvi

Bursa Teknik Üniversitesi

Komponentler

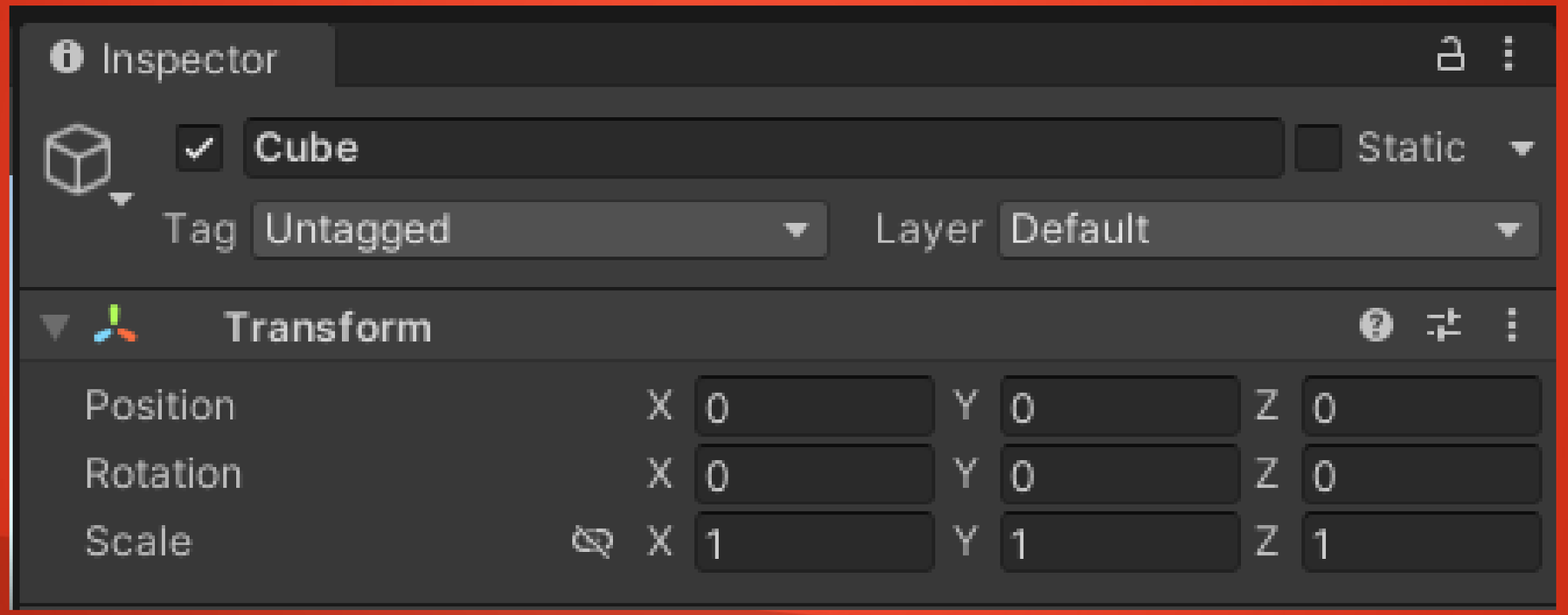
Komponentler (component) oyun sahnesinde yer alan nesnelere atanan modüllerdir. Nesne Tabanlı Programlama (OOP) açısından bakacak olursak; sahnedeki her oyun nesnesi aslında bir GameObject nesne örneğidir (* instance). GameObject nesne örneklerinde (*) bulunan her komponent de ayrı birer sınıfın nesnesidir (*).

*Instance: genellikle new anahtar kelimesi ile, bir **sınıftan** oluşturulmuş olan ve heap'te tutulan nesne.*

Temel Oyun Motoru İlkeleri

```
class Vector3
{
    public double x,y,z;
}
class Transform
{
    public Vector3 position, rotation, scale;
}

class Hareket { /*bir takım aksiyonlar*/  }

class OrtakDavranislar
{
    public Transform transform;

    void Start()
    {

    }
    void Update()
    {

    }
}

public class Karakter : OrtakDavranislar
{
    public Hareket bilinmeyenIsim;
}
```

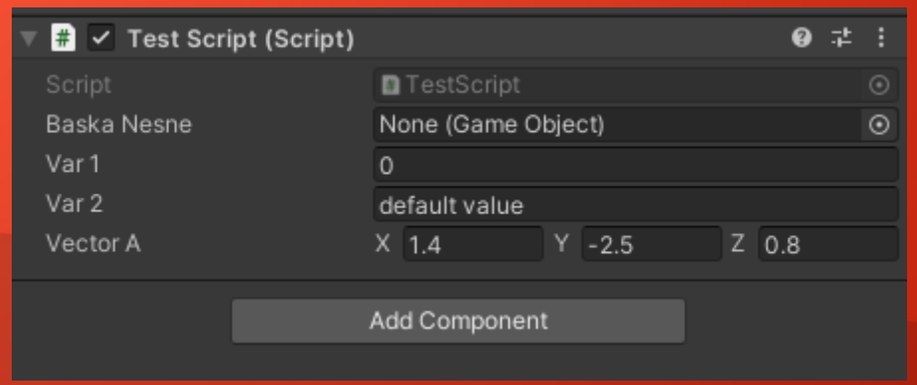
Component'ler



Komponent Parametreleri

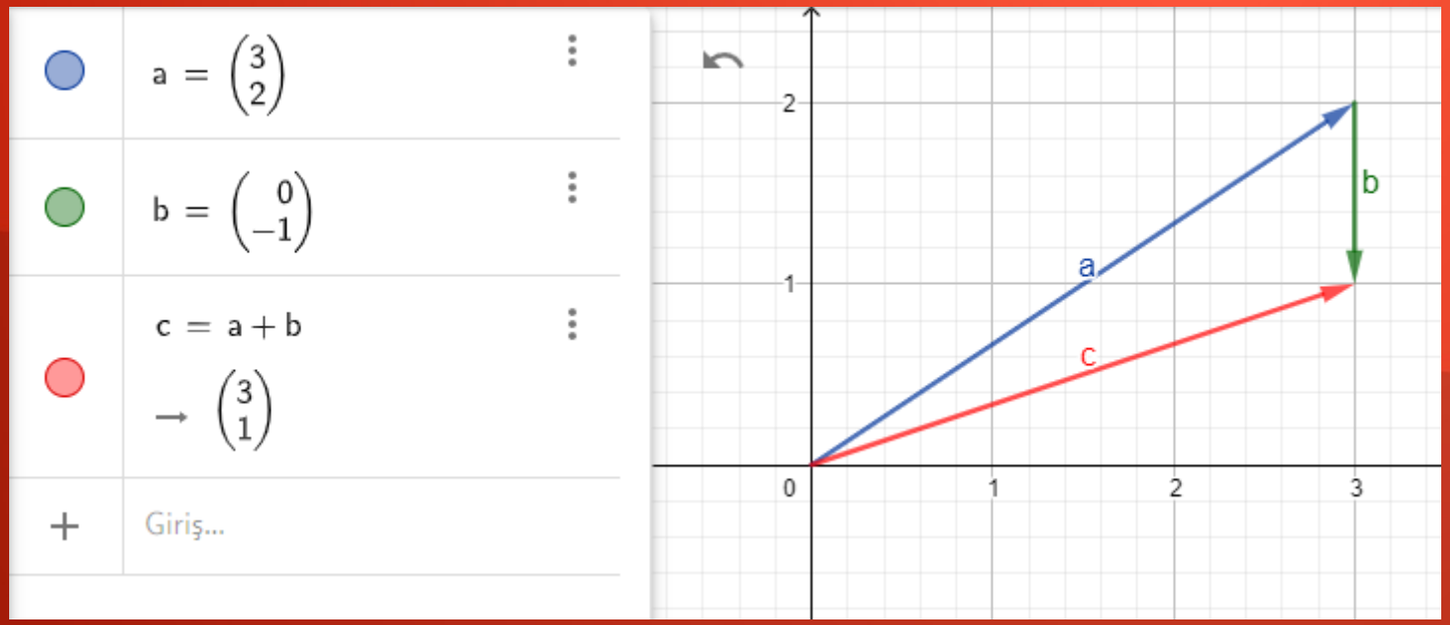
```
public GameObject baskaNesne;

public float var1;

public string var2 = "default value";

[SerializeField]

private Vector3 vectorA = new Vector3(1.4f, -2.5f, .8f);

private Vector3 vectorB;
```

Bir oyun nesnesi bir başka oyun nesnesinin pointer'ını tutabilir.



Translate (hareket)

```
transform.position += new Vector3(0.1f, 0.1f, 0.0f);

transform.position = transform.position + new Vector3(0.1f, 0.1f, 0.0f);

baskaNesne.transform.position += new Vector3(0.1f, 0.1f, 0.0f);
```



Nesneleri hareket ettirmek için konum vektörlerine hareket vektörü eklenir.

Unity'deki Vector3 nesnesi + operatörünü desteklemektedir.

Ek olarak transform.Translate() fonksiyonu da kullanılabilir fakat eksenleri ayarlamanın en iyi yöntemi vektörleri doğrudan kullanmaktır.

Rotation (döndürme)

$a = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$

$ang = 30$

0 — 360

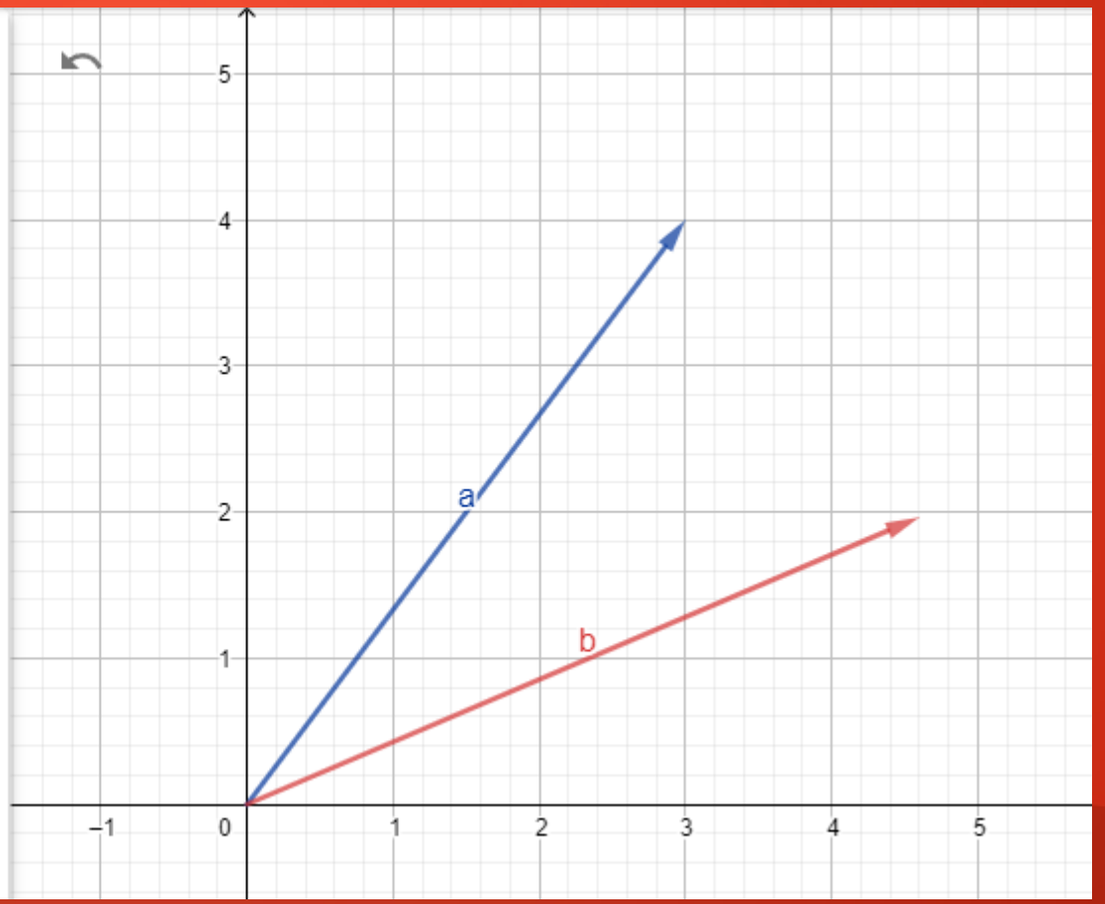
$rot = \begin{pmatrix} \cos(ang°) & \sin(ang°) \\ -\sin(ang°) & \cos(ang°) \end{pmatrix}$

$\rightarrow \begin{pmatrix} 0.8660254037844 & 0.5 \\ -0.5 & 0.8660254037844 \end{pmatrix}$

$b = (rot\ a)$

$\rightarrow \begin{pmatrix} 4.5980762113533 \\ 1.9641016151378 \end{pmatrix}$

Giriş...



Rotation (döndürme)

```
transform.rotation *= Quaternion.Euler(new Vector3(30,60,90));

transform.rotation = transform.rotation * Quaternion.Euler(new Vector3(30,60,90));
```

Nesneleri döndürmek için Quaternion’lar kullanılır.

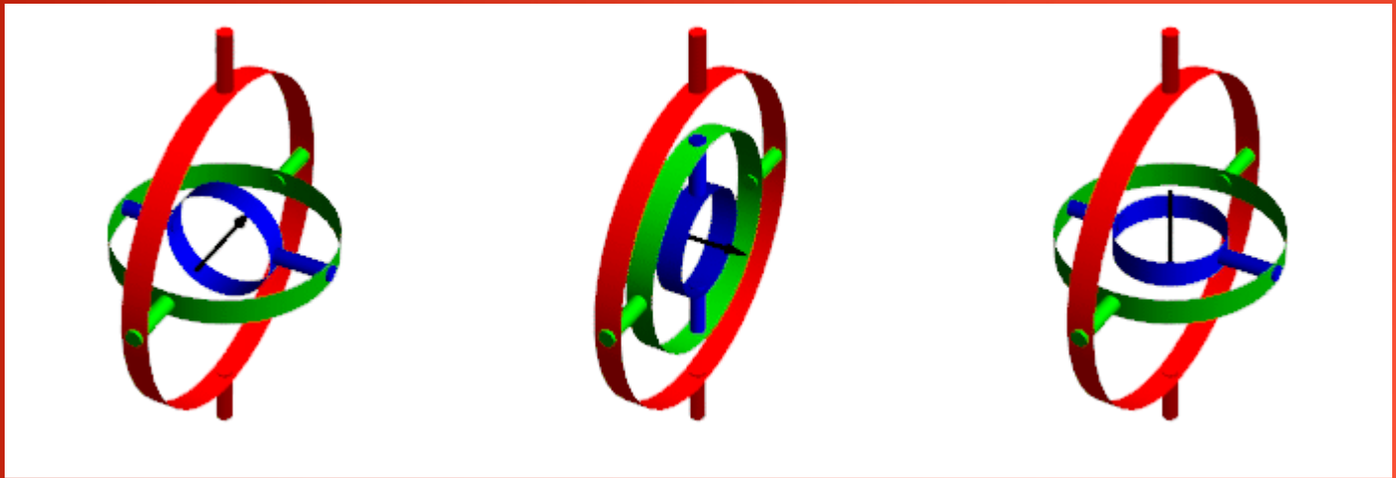
Unity’deki Quaternion nesnesi * operatörünü desteklemektedir. İfadenin sağındaki ve solundaki elemanların sırası önemlidir.

Neden Quaternion’lar?

Vector3 verileri kullanılarak da döndürme işlemi yapılabilir fakat x, y ve z eksenleri için ayrı ayrı dönüş miktarları belirlemek Gimbal Lock adı verilen eksen kilitlenmesi problemine sebep olur.

Quaternion’lar dönüş işlemini matematik işlemleriyle tüm eksenler için aynı anda yaptığı için bu problemin ortaya çıkmasını engeller.

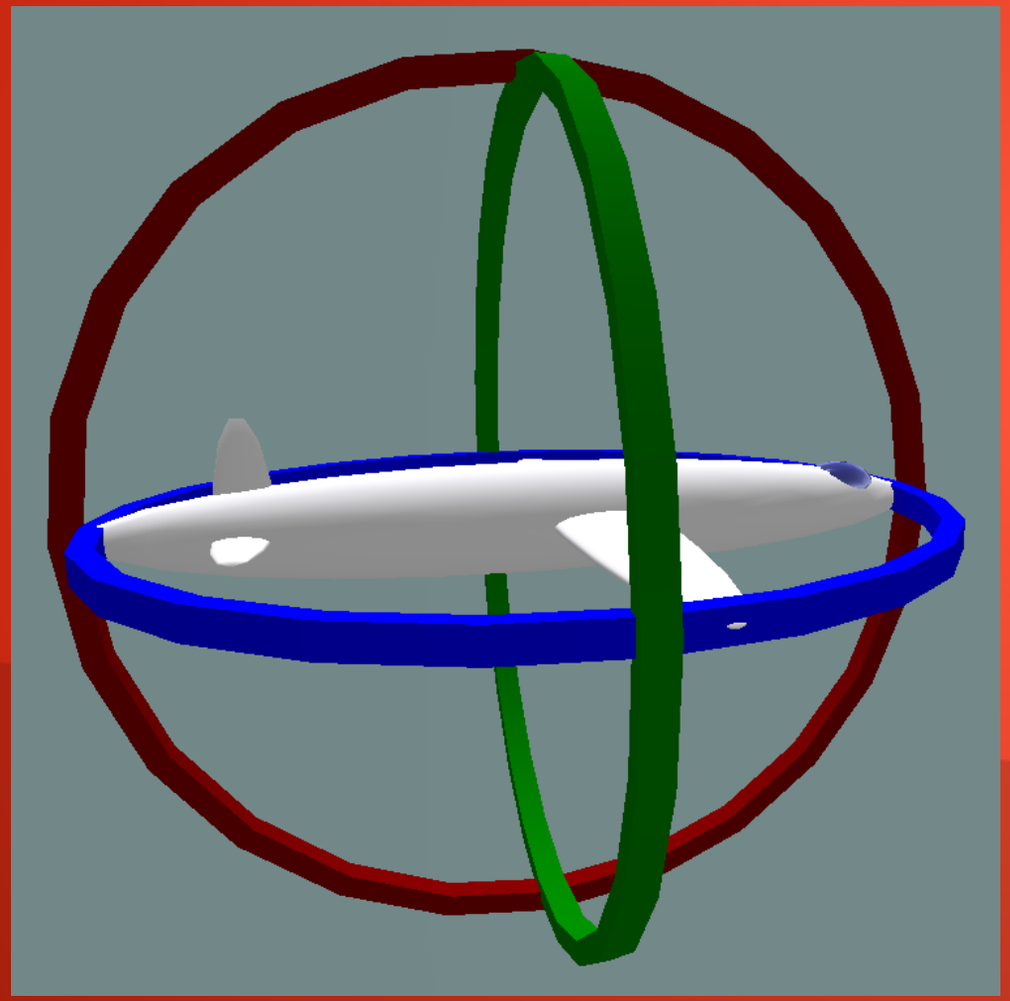
Gimbal Lock

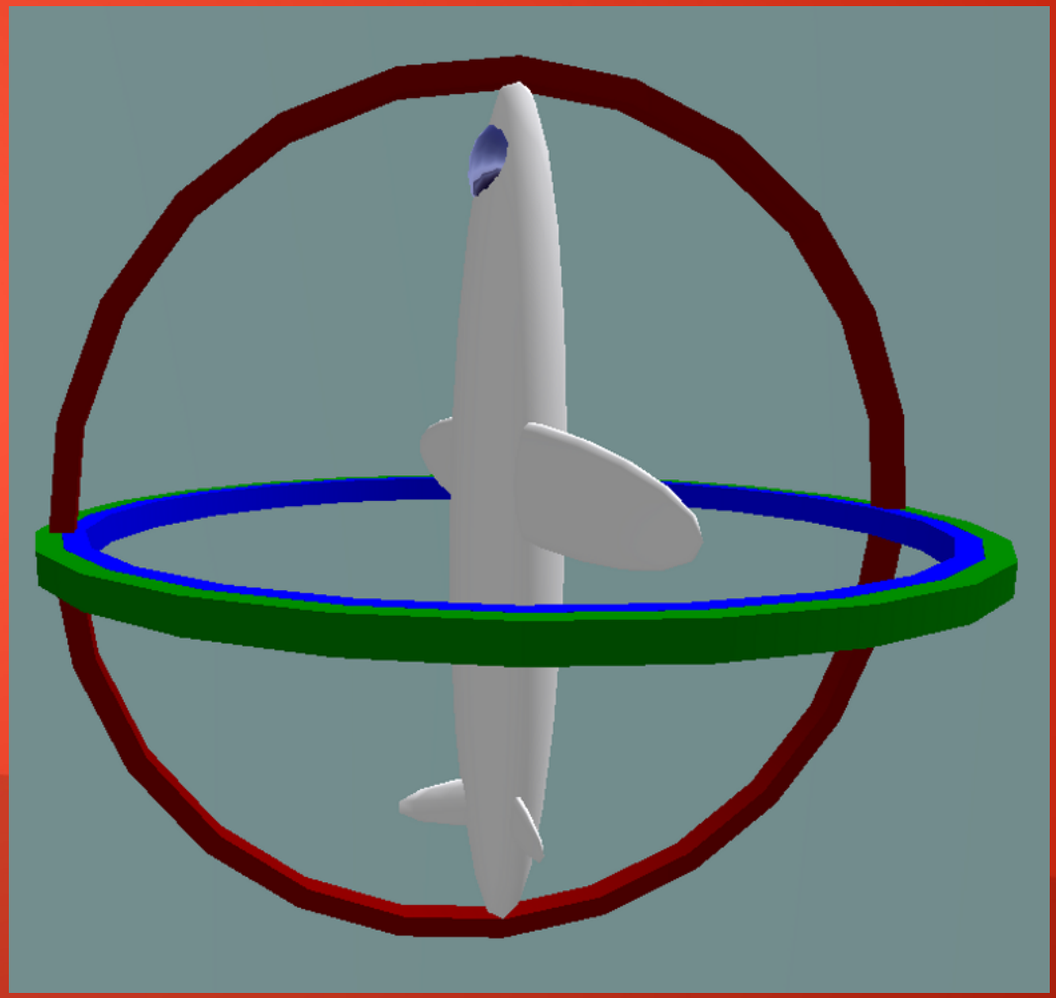


Src:
https://www.allaboutcircuits.com/technical-articles/dont-get-lost-in-deep-space-understanding-quaternions/

Src:
https://upload.wikimedia.org/wikipedia/commons/5/5e/KUKA_Industrial_Robots_IR.jpg

Gimbal Lock





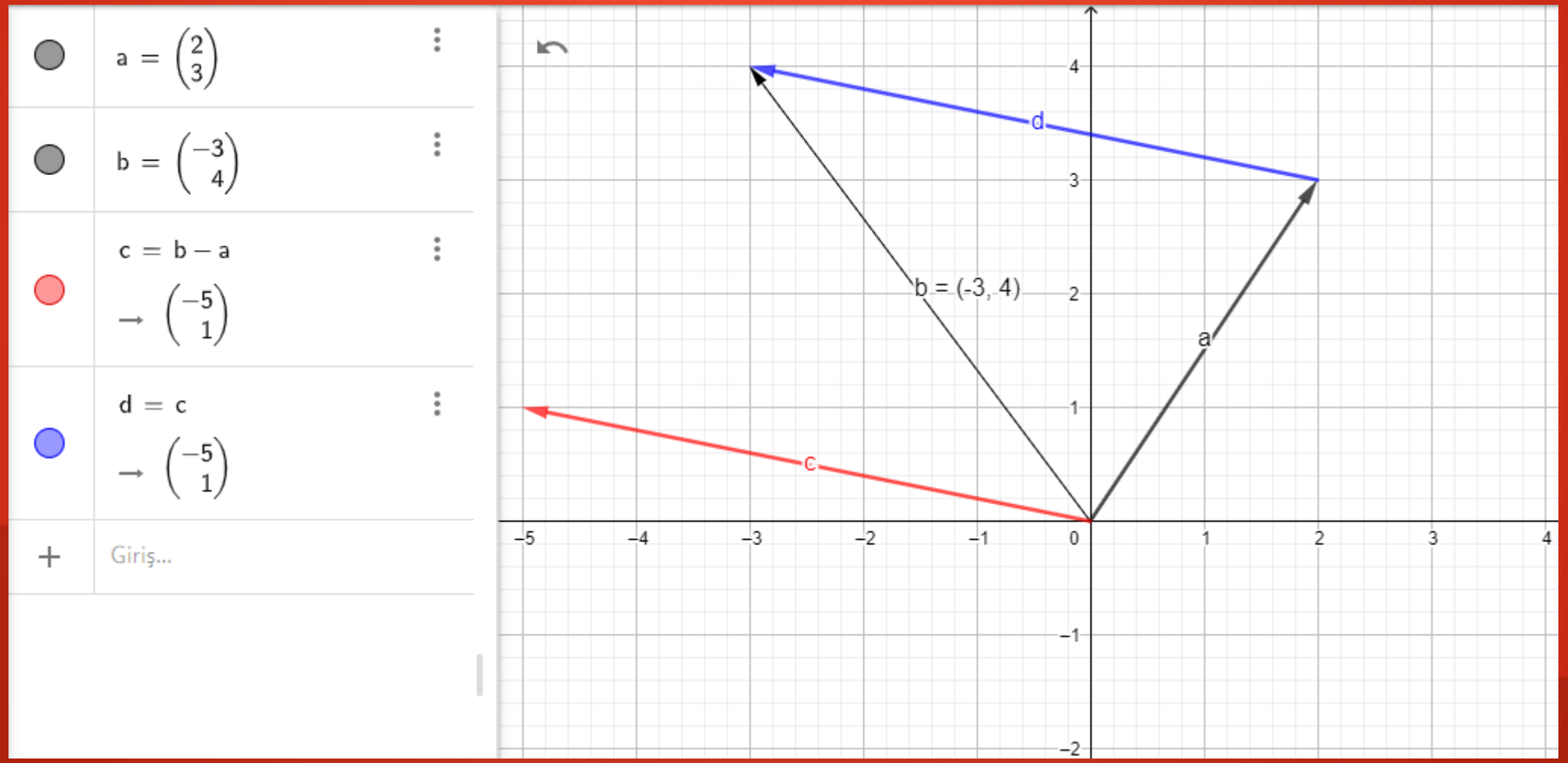
Scale (ölçeklendirme)

```
transform.localScale = new Vector3(1,2,1);
```

Bir vektörün bir skaler değer ile veya elemanlarının başka bir vektörün elemanlarıyla karşılıklı çarpılmasıyla hesaplanır.

Unity'de 'scale' değeri Vector3 türündedir ve bir nesnenin Transform komponentinin localScale elemanına erişilerek değiştirilebilir.

Konum vektöründen yön vektörüne



Konum vektöründen yön vektörüne

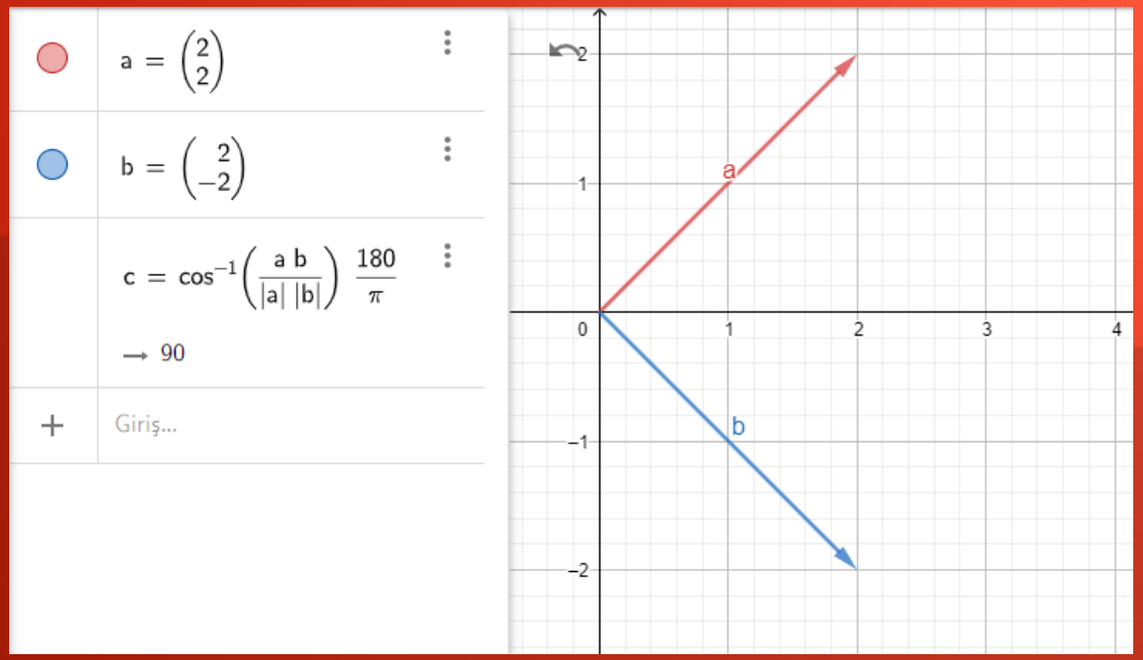
```
Vector3 direction_from_me_to_other = baskaNesne.transform.position - this.transform.position;
Vector3 direction_from_me_to_other = baskaNesne.transform.position - transform.position;
```

B pozisyonundan A pozisyonu çıkarılırsa, A’dan B’ye yön vektörü hesaplanmış olur.

Bu yön vektörü A’nın pozisyonuna eklenirse A, B’ye yaklaşır; B’nin pozisyonuna eklenirse B, A’dan aynı doğrultuda uzaklaşır.

Vektörler arası açı

```
Vector3 targetDir = baskaNesne.transform.position - transform.position;

float angle = Vector3.Angle(targetDir, transform.forward);

if (angle < 5.0f)

{

    print("Nesne kameranın görüş alanında");

}
```




```
Vector3 targetDir2 = transform.position - baskaNesne.transform.position;

float angle2 = Vector3.Angle(targetDir2, baskaNesne.transform.forward);

if (angle2 < 5.0f)

{

    print("Oyuncu, nesnenin görüş alanında");

}
```

Nesnelerin birbirlerinin görüş alanında olup olmadığını tespit etmek gibi çeşitli işlere yarayabilir.

Neyse ki, Unity bu vb. fonksiyonları zaten implement etmiş. Trigonometrik fonksiyonlarla sürekli uğraşmamız gerekmiyor.