

# Uçak Simülasyon Yazılımı

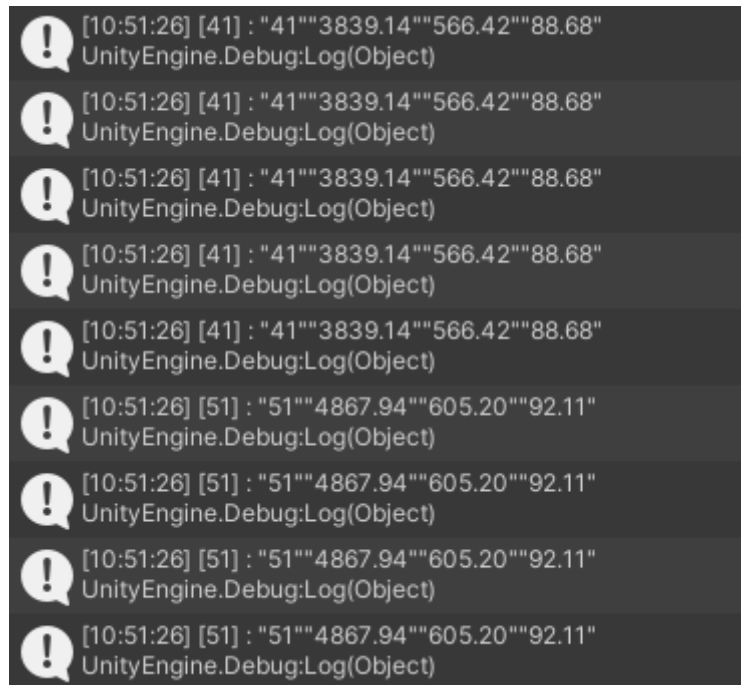
Hazırlayan: Kerim Görkem Çelebiler

Bu döküman Excel dosyasından alınan uçak motoru verilerini kullanarak bir simülasyon yapmayı amaçlamaktadır. Bu projede C# ve Unity3d kullanılmıştır.

Excel dosyamızda uçağın Throttle verilerine bağlı olarak değişen, RPM, FTIT ve Fuel değerleri mevcut. İlk önce yapmamız gereken şey bu değerleri programa aktarmak olacak.

```
private string GetData(string throttle)
{
    string csvFile = "Assets/throttle_valuecsv.csv";
    foreach (string line in File.ReadLines(csvFile))
    foreach (string value in line.Replace("\"", "").Split('\r', '\n', ','))
        if (value.Trim() == throttle.Trim())
            return "[" + value + "] : " + line;
    return "";
}
```

Bu fonksiyon ile verilen bir Throttle değerine göre ilgili yerdeki tüm verileri tek bir String’de toplamış oluyoruz. “csvFile” dosyamızın bulunduğu yer. “File.ReadLines(csvFile)” ile ilgili dosyadaki tüm değerleri okuyoruz. Replace fonksiyonu ile verilerin içinde çift tırnakları boşluklar ile değiştiriyoruz. Split ile bu boşlukları kullanarak her bir veriyi birbirinden ayırıp en sonda döndürüyoruz. Bu işlem bize aşağıdaki gibi bir sonuç döndürüyor.



The screenshot shows the Unity console with 10 log entries. The first 5 entries correspond to throttle value 41, and the next 5 correspond to throttle value 51. Each entry shows the throttle value, a list of values in brackets, and the corresponding line from the CSV file.

```
[10:51:26] [41] : "41""3839.14""566.42""88.68"
UnityEngine.Debug:Log(Object)

[10:51:26] [41] : "41""3839.14""566.42""88.68"
UnityEngine.Debug:Log(Object)

[10:51:26] [41] : "41""3839.14""566.42""88.68"
UnityEngine.Debug:Log(Object)

[10:51:26] [41] : "41""3839.14""566.42""88.68"
UnityEngine.Debug:Log(Object)

[10:51:26] [41] : "41""3839.14""566.42""88.68"
UnityEngine.Debug:Log(Object)

[10:51:26] [51] : "51""4867.94""605.20""92.11"
UnityEngine.Debug:Log(Object)

[10:51:26] [51] : "51""4867.94""605.20""92.11"
UnityEngine.Debug:Log(Object)

[10:51:26] [51] : "51""4867.94""605.20""92.11"
UnityEngine.Debug:Log(Object)

[10:51:26] [51] : "51""4867.94""605.20""92.11"
UnityEngine.Debug:Log(Object)

[10:51:26] [51] : "51""4867.94""605.20""92.11"
UnityEngine.Debug:Log(Object)
```

Bu çıkan verileri “words” isimli tek bir değişkene aktarıyoruz. “GetData” fonksiyonumuzun parametresi olarak Throttle değerini gönderiyoruz. Şimdi bunun içinden RPM, FTIT ve Fuel’i ayırtmamız lazım. Excel verilerine göre Fuel 1, FTIT 2 ve RPM 3. indexte duruyorlar. “Trim” fonksiyonu ile sayıların başında ve sonundaki tırnak işaretlerini kesip Float’a çeviriyoruz. 100’e bölmemizin sebebi ise Excelden çekilen verilerde noktalama işaretlerinin kaybolması sebebiyle görünenden çok büyük sayılar çıkması.

```
words = GetData(Math.Floor(throttle).ToString()).Split(',');
rpm = float.Parse(words[3].ToString().Trim('')) / 100;
fuel = float.Parse(words[1].ToString().Trim('')) / 100;
ftit = float.Parse(words[2].ToString().Trim('')) / 100;
```

Şimdi sıra geldi bu gelen veriyi göstergelerde göstermeye. Bunun için göstergenin iğnesini ilk önce en düşük noktaya sonra da en yüksek noktaya getirip sırayla o konumdaki açılarını bir değişkene atıyoruz. Ama en düşük noktadan en yüksek noktaya giderken kısa yoldan değil tüm sayıların üzerinden geçerek gitmemiz gerekiyor.

En Düşük Nokta



En Yüksek Nokta



Şimdi bu değişkenleri kullanarak bir açı bulmamız gerekiyor. En düşük noktadan en yüksek noktayı çıkardığımızda iğnenin dönebileceği toplam alanı elde etmiş oluruz. Daha sonra gelen veriyi normalleştirmemiz gerekiyor. Aşağıdaki örnekte olduğu gibi “rpm / rpmMax” yaptığımız zaman RPM değerimiz 0 ile 1 arasında bir değer almış olacak. 0 olduğunda iğnenin 0’ı, 1 olduğunda ise iğnenin maksimum değeri göstermesini istiyoruz. En son olarak bu yaptığımız hesabı en düşük noktanın üzerine ekleyerek geri döndürüyoruz. Çıkartma işlemi yapmamızın sebebi Unity’de açılar saat yönünün tersine döner. Ama biz saat yönünde dönmesini istediğimiz için tam tersini alıyoruz.

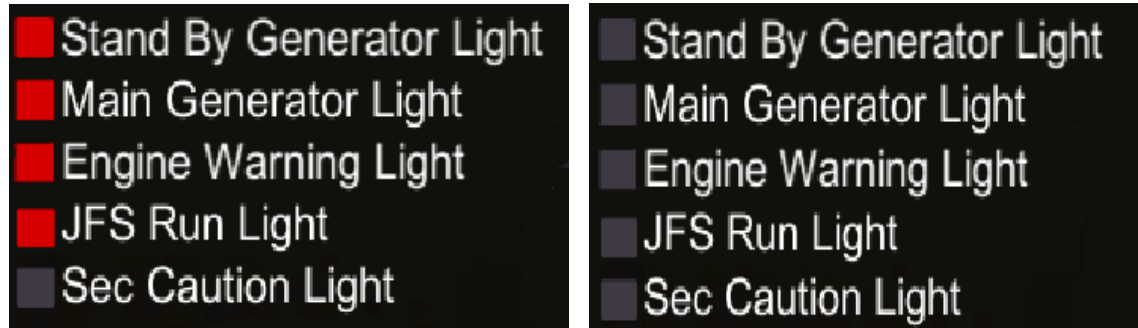
```
private float GetRPMRotation()//returns specific angle for the RPM
{
    float totalAngleSize = zero_angle_rpm - max_angle_rpm;
    float rpmNormalized = rpm/rpmMax;
    return zero_angle_rpm - rpmNormalized * totalAngleSize;
}
```

Şimdi bu aldığımız açıyı gösterge iğnelerine uygulamalıyız. Gösterge iğnesinin “eulerAngles” değişkenine yeni Vector3 atayarak bunu sağlayabiliriz.

```
rpm_needle.eulerAngles = new Vector3(0, 0, GetRPMRotation());
```

Simülasyonda belli ışıklar var. Bu ışıklar Throttle’ın sürdüğü RPM göstergesinin değerine bağlı olarak yanıyorlar (Işığın 1 olması yanıyor olması demek). StandBy Gen Lt, Main Gen Lt, Engine Warning Lt, JFS Run Lt ve Sec Caution Lt olmak üzere 5 adet ışık var. StandBy Gen Lt ve Engine Warning Lt eğer RPM 60’dan küçükse yanıyorlar. Main Gen Lt, RPM 60.225’ten küçükse yanıyor. JFS Run Lt, RPM 55’ten küçükse yanıyor. Son olarak Sec Caution Lt ise RPM 20’den küçükse yanıyor. Unity’de Canvas’a bunları ekliyoruz. Işık olarak ben basit bir button kullandım. Eğer ışık yanıyorsa buttonun rengini kırmızı yapıyorum. Bunu basit bir if-else bloğuyla yapabildiğim için sadece tek bir örnek koyacağım. StandBy Gen ve Engine Warning için kullandığım kod aşağıdaki gibi:

```
if (rpm < 60){
    standbygen.color = Color.red;
    enginewarning.color = Color.red;
}
if (rpm >= 60){
    standbygen.color = defaultColor;
    enginewarning.color = defaultColor;
}
```



Gelelim arızalar kısmına. Uçağı düzgün kullanmazsak oluşacak belli başlı arıza tipleri var. Bunlardan birincisi Throttle’ı 100’ün üstüne getirirsek uçak titremeye başlar. Eğer Throttle’ı düşürmezsek belli bir süre sonra motor yanabilir. Bunun için projeye kamera sallanma efekti ve ateş efekti eklememiz lazım.

Kamera sallantı efekti için Unity’de bulunan Main Camera’nın pozisyonunu uçağın konumuna bağlı olarak rastgele değiştirebiliriz. Bunun için kameranın pozisyonuna ihtiyacımız var. “camTransform” değişkeni kameramızın asıl pozisyonunu tutuyor. “originalCamPos” diye bir değişken oluşturup program başlar başlamaz “camTransform”u “originalCamPos”a eşitliyoruz. Artık “originalCamPos” kameramızın başlangıç pozisyonunu tutuyor. “shakeAmount” ise kamerayı ne kadar sallayacağımızı belirleyen değer. UnityEngine kütüphanesindeki Random metodunu kullanarak rastgele bir sayı döndürüyor. Bunu

“shakeAmount” ile çarparak kameramızın asıl pozisyonu olan “originalCamPos”a ekleyip “camTransform”a atıyoruz. Bununla birlikte artık Throttle ne zaman 100’den fazla olursa kameramız rastgele olarak yer değiştiriyor. Yani sallanıyor. “engineStatus\_Text”e ise uyarı mesajı yazdırıyoruz.

```
if (throttle > 100)//camera shake when plane is too fast
{
    camTransform.localPosition = originalCamPos + UnityEngine.Random.insideUnitSphere * shakeAmount;
    planeTransform.localPosition = new Vector3(0, camTransform.localPosition.y - 0.6f, camTransform.localPosition.z - 0.786f);
    engineStatus_Text.text = "Reduce Throttle";
    engineStatus_Text_2.text = "";
    shakeTime += Time.deltaTime;
}
```

“shakeTime” değişkeni uçak sallanmaya başladığı zaman artmaya başlıyor. Eğer 5 saniyeden uzun süre bu durumdan kurtulmazsak motor yanıyor. Bu durum yaşandığında ilk önce kamerayı “originalCamPos” değişkenini kullanarak eski haline geri getiriyoruz. Unity’nin parçacık sistemi(particle system)’ni kullanarak patlama ve yanma efekti ekliyoruz. Bu efektlerin çalışması Play() fonksiyonunu kullanıyoruz. “engineStatus\_Text”e ise uyarı mesajı yazdırıyoruz.

```
if (rpmtime > 5)
{
    camTransform.localPosition = originalCamPos;
    planeTransform.localPosition = originalPlanePos;
    engineStatus_Text.text = "Burned";
    transform.position += transform.forward * Time.deltaTime * throttle;
    engineStatus_Text_3.text = "";
    isCrashed = true;
    explosion.Play();
    fire.Play();
}
```



İkinci arıza tipimiz “No Start”. Yani uçak motorunun hiç çalışmaması. Bunun için motoru tekrar açıp kapatmak gerekiyor. Motoru açma kapama tuşunu istediğiniz tuş yapabilirsiniz. Ben “F” yaptım. Motoru açtığımız zaman “random” değişkenine 1 ile 10 arasında rastgele sayı dönüyor. “start” adlı değişken oluşturup 0’a eşitliyoruz. Motoru açtığımız zaman “start”ı arttırıyoruz.

```
if (Input.GetKeyDown(KeyCode.F) && !isCrashed)
{
    random = UnityEngine.Random.Range(1, 10);
    start++;
}
```

Uçağın sağa sola öne arkaya hareketini “position” ve “Rotate” bileşenlerini değiştirerek yapıyoruz. Uçağın baktığı yön “transform.forward” ile alınıyor. Bu yönde “position” değişkenini arttırsak uçak ileri gitmeye başlayacaktır. Unity’nin kendi ayarlarında “Vertical” olarak belirtilen tuşlar arasında “W ve S”, “Horizontal”de ise “A ve D” var. “Input” sınıfından bunları çağırıp “Rotate” fonksiyonuna verdiğimizde uçağın sağ sol yukarı aşağı hareketini sağlıyoruz.

```
transform.position += transform.forward * Time.deltaTime * throttle;
transform.Rotate(Input.GetAxis("Vertical"), 0.0f, -Input.GetAxis("Horizontal"));
```

“start” değişkeninin 2 ile bölümünden kalanı 1 ise motor çalışacak, 0 ise motor kapanacak.

Eğer “start” 1, “random” 6’dan büyük ve “isCrashed” yere çakılma operatörleri doğruysa, uçak normal bir şekilde çalışacak ve Throttle 10’ar 10’ar yani normal hızda hızlanacaktır.

```
if (start % 2 == 1 && random > 6 && !isCrashed)
{
    enginestatus_text.text = "Engine On";
    Fly(10);
}
```

Eğer “start” 0 ise motor kapanacak ve uçağa uygulanan fizik kuvvetleri devreye sokulacaktır. Bunu uçağın rigidBody bileşeninin “detectCollisions”ını doğru, “isKinematic”ini yanlış yaparak sağlıyoruz. Bu sayede eğer motor havada kapanırsa uçak yere düşmeye başlayacaktır.

```
if (start % 2 == 0 && !isCrashed)
{
    rb.detectCollisions = true;
    rb.isKinematic = false;
    enginestatus_text.text = "Engine Off";
}
```

“random” değişkeni eğer 3’ten küçükse “No Start” hatası verdiriyoruz. Bu olasılıkları istediğiniz gibi belirleyebilirsiniz. “engineStatus\_Text”e motoru yeniden açması için bir uyarı mesajı yazdırıyoruz.

```
if(start%2==1 && random <= 3 && !isCrashed)
{
    enginestatus_text.text = "No Start";
    enginestatus_text_2.text = "Restart the Engine";
    transform.position += transform.forward * Time.deltaTime * throttle;
    rpmtime = 0;
}
```



Üçüncü arıza tipimiz Slow Start. Yani uçağın normalden daha yavaş hızlanması. Eğer “random” değişkeni 4 ila 6 arasındaysa “Slow Start” olarak ayarladım. Bu olasılığı daha önce belirttiğim üzere değiştirebilirsiniz. Fly fonksiyonuna normal hızlanma olan 10 yerine 2 gönderip “Slow Start” olmasını sağlayabilirsiniz.

```
if(start%2==1 && random>=4 &&random<=6 && !isCrashed)
{
    enginestatus_text.text = "Slow Start";
    enginestatus_text_2.text = "Increase Speed";
    if (throttle >= 30) random = 7;
    Fly(2);
}
```



“Fly” fonksiyonunun içinde gönderdiğimiz sayıyı, “throttle” değerini arttırmak için kullanıyoruz. Eğer “Mouse0” yani farenin sol tuşuna basarsak artıyor, “Mouse1” yani farenin sağ tuşuna basarsak azalıyor.

```
void Fly(int speed)
{
    if (Input.GetKeyDown(KeyCode.Mouse0)) throttle += speed;
    if (Input.GetKeyDown(KeyCode.Mouse1)) throttle -= speed;
    .
    .
    .
}
```

Dördüncü arıza tipimiz Hot Start. Eğer RPM değeri çok düşük seyrederse belirli bir süre sonra Hot Start olur. Bunun sonucunda “FTIT” değeri artar. Eğer RPM yükseltirirse motor normale döner. Bu süreç boyunca “engineStatus\_Text”e uyarı mesajı yazdırırız.

```
if (rpm < 80){
    rpmtime += Time.deltaTime;
    enginestatus_text_2.text = "Increase RPM";
}
if (rpm > 80 && throttle<=100)
{
    rpmtime = 0;
    ftit = float.Parse(words[2].ToString().Trim('')) / 100;
    engineStatus_Text.text = "Fine";
}
if (rpmtime > 5)
{
    explosionTime += Time.deltaTime;
    engineStatus_Text.text = "Hot Start";
    engineStatus_Text_2.text = "Increase RPM";
    ftit += 200;
}
```





Beşinci ve son arıza tipimiz Hung Start. Hot Start durumunda bir şey yapmazsak belli bir süre sonra Hung Start olur. Motor patlar. Hot Start sırasında “explosiontime” değişkenini arttırmaya başlamıştık. Eğer bu değişken 5’ten fazla olursa Hung Start’ı başlatıyoruz.

```
if (explosionTime > 5)
{
    enginestatus_text.text = "Hung Start";
    explosion.Play();
    fire.Play();
    isCrashed = true;
}
```



Uçağın yüksekliği için uçağımızın “Y” konumundaki yerini yazdırmamız yeterli. Çünkü Unity’nin kullandığı birim metredir.

```
altitude_text.text = Math.Floor(transform.position.y).ToString() + " meters";
```

Durum göstergesi için uçağın “Z” rotasyonunu almalıyız. Bu rotasyonu “rot\_turn” adlı bir değişkene attım. Daha sonrasında bunu direk göstergeye uygulayabiliriz. Göstergenin sadece “Z” rotasyonunu değiştiriyoruz. Ancak eksi ile çarpılmış halini gönderiyoruz. Çünkü uçak ters döndüğünde de yukarıyı göstermesi gerekiyor.

```
rot_turn = transform.eulerAngles.z;  
adi_transform.localRotation = Quaternion.Euler(0, 0, -rot_turn);
```



Eğer motor kapalıysa uçak yere düşmeye başlar. Bunu Unity’nin kendi fizik motoru ile yapıyoruz. Unity, eğer bir obje başka bir objeyle ile çarpışırsa otomatik olarak “OnCollisionEnter” fonksiyonu çağırılır. Bu fonksiyonun içinde eğer çarpıştığımız objenin etiketi “Ground” ise yere düşmüştür diyeceğiz. Bunun sonucunda patlama ve ateş efektlerini oynatırız. “start” değişkenini ondalıklı hale getirerek motorun yeniden çalışmasına engel oluruz. Çünkü “start” değişkeninin ikiye bölümünden kalan ile motorun açma kapamasını kontrol ettiğimiz için ondalıklı sayı yaptığımızda bu olay olmayacaktır.

```
private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag=="Ground")
    {
        start = 0.1f;
        isCrashed = true;
        engineStatus_Text.text = "Crashed";
        explosion.Play();
        isExpPlayed = true;
    }
}
```

