

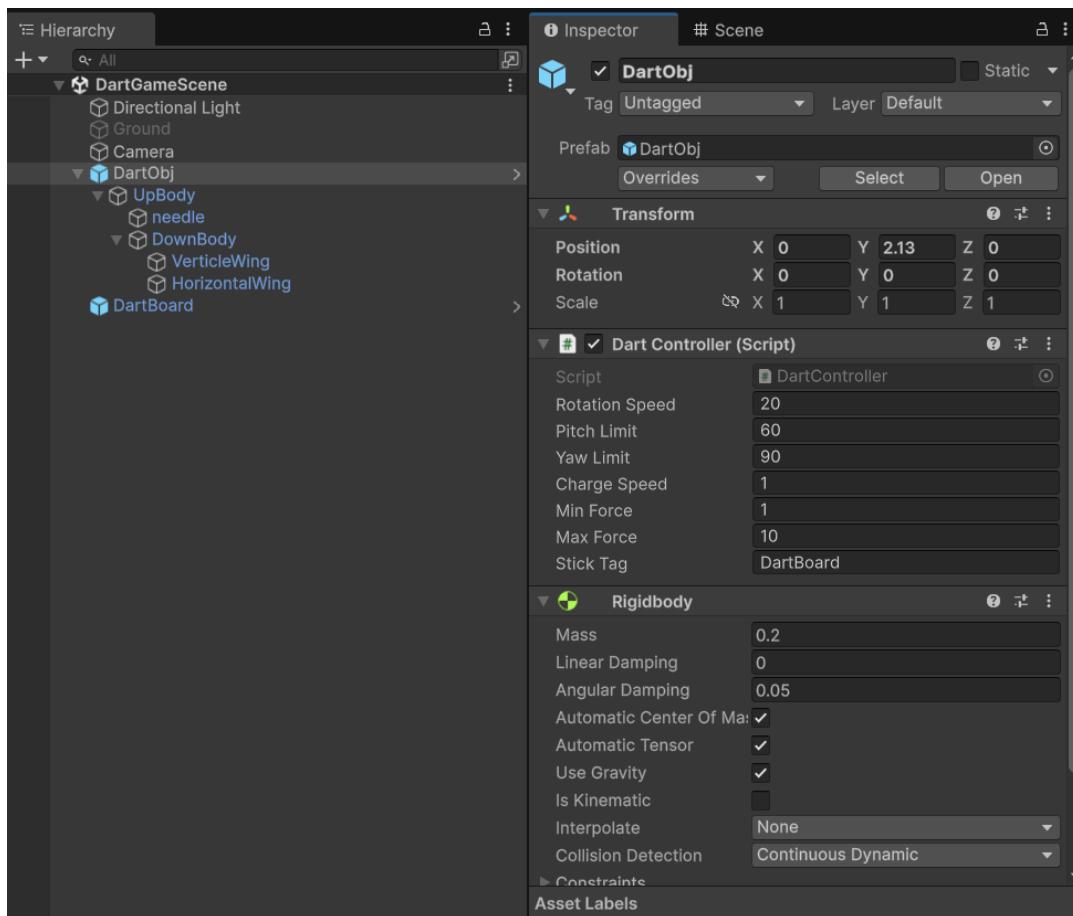
Yapılacaklar :

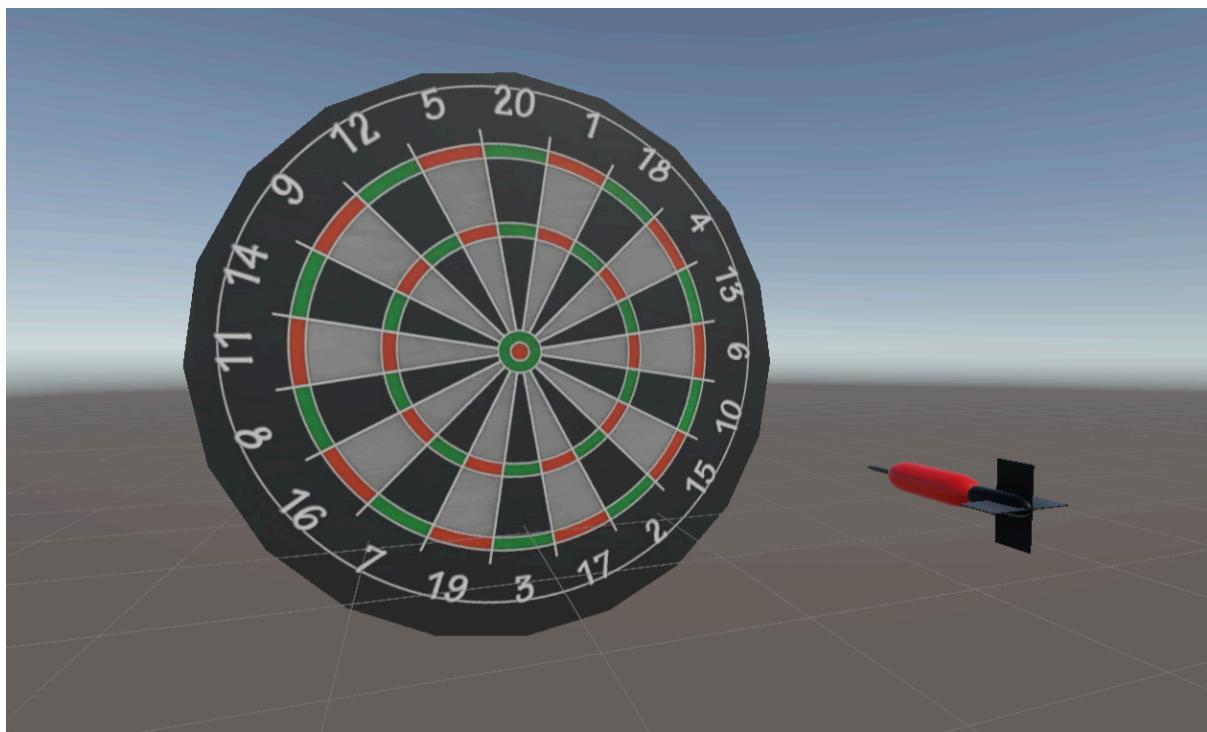
Gerçekçi bir dart fırlatma hissi veren, kameraları değiştirilebilir ve farklı dart tiplerini destekleyebilecek bir mini oyun hazırlıyoruz. Proje boyunca üç temel hedefimiz var:

1. Fizik tabanlı bir dart kontrol sistemi (yönlendirme, güç toplama, fırlatma, hedefe yapışma).
2. Dartı takip eden, gerekli olduğunda kamera açıları arasında geçiş yapılabilen bir izleme sistemi.
3. İlleride farklı dart özellikleri tanımlayabilmek için ScriptableObject tabanlı veri tutma.

Adım 1 – Dart ve Sahne Kurulumu

- Dart hiyerarşisi: Dart adında kök objeyi oluşturuyoruz. Kökte Rigidbody ve DartController yer alacak. Altında:
 - Body: Mesh, materyal, ışıklandırma vb. görsel öğeler burada.
 - RigidBody ayarları:
 - Mass 0.2–0.3 civarı tutulabilir.
 - Collision Detection = Continuous Dynamic yapıyoruz bu sayede çarpışmalar olması gerektiği noktada çok daha iyi sonuç verecek.
 - Hedef (Dart Tahtası): Mesh Collider ekle, stickTag değeri ile aynı tagi ver (örn. "DartBoard").





Adım 2 – Girişler ve Fırlatma Mekanığı

DartController scripti şu davranışlarını sağlayacak:

- Yönlendirme: W/S pitch (Yukarı aşağı eğim), A/D yaw (Sağa Sola Eğim) ayarlayacak. rotationSpeed (W,A,S,D tuşlarının dartı ne kadar hızlı çevireceği), pitchLimit, yawLimit (En fazla kaç dereceye kadar döndürülebileceği) inspector'dan değiştirilebilir olacak.
- Güç toplama Space tutulduğça charge 0–1 arasında gidip gelir. Bu sayede kısa süreli güç ayarı yapılabilir (Space'i uzun tutarsan güç maksimumdan düşmeye başlar).
- Fırlatma: Space bırakıldığında dart impulse alır, gravity açılır. Fırlatma dartın baktığı yöne doğru hızlanır.
- Saplanma: OnCollisionEnter içinde sadece stickTag eşleşmesi kontrol edilir. Eşleşirse Rigidbody kinematik+gravity false olur; dart olduğu yerde kalır.
- Reset (R tuşu): Başlangıç pozisyonu ve rotasyonu kayıtlıdır. R ile dart eski haline döner, tekrar hedeflenebilir hale gelir.

Collider notu: Rigidbody kökte olduğu sürece child collider'lar sorunsuz çalışır.

Adım 3 – Kamera ve Takip Sistemi

- DartFollower script, kamerayı dartın arkasında belirli offset'te tutar ve dartın hız vektörüne göre bakış açısını günceller. Inspector alanları:
 - `target` (dart transform'u), `targetBody` (dart Rigidbody'si)
 - `offset`, `moveSpeed`, `turnSpeed`, `lookAhead`, `useVelocityLook`

Kodların Yazılması :

1. DartController.cs

- RequireComponent(typeof(Rigidbody)) ile Rigidbody zorunlu.
- Inspector'dan ayarlanan alanlar: `rotationSpeed`, `pitchLimit`, `yawLimit`, `chargeSpeed`, `minForce`, `maxForce`, `stickTag`.
- Temel fonksiyonlar:
 - ReadRotation() – pitch/yaw değerlerini klavye girişlerine göre günceller.
 - ReadCharge() – Space basılıyken güç osilasyonunu yönetir; bırakıldığında Launch() çağırır.
 - Launch() – Gravity'yi açar, AddForce ile kuvvet uygular.
 - AlignWithVelocity() – Uçuşta modeli hız vektörüyle hizalar.
 - OnCollisionEnter – stickTag eşleşince body.isKinematic = true, body.useGravity = false yapar bu sayede dart çarptığı noktada kalır.
 - ResetDart() – R tuşunda çağrılır; tüm state değerlerini sıfırlar.

2. DartFollower.cs

- Kameraya atanır; dartı offset ile takip eder.
- LateUpdate() içinde pozisyon `Vector3.Lerp`, rotasyon `Quaternion.Slerp` ile yumuşatılır.
- useVelocityLook açıkken dartın hız yönünü baz alınarak hizalanır; aksi halde target.forward ile dartın baktığı yön ile hizalanır.