KOCAELİ ÜNİVERSİTESİ

TEKNOLOJİ FAKÜLTESİ

BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ BÖLÜMÜ

A logo with text and symbols

AI-generated content may be incorrect.

Bitirme-A Raporu

Eren ÖZCAN / 201307070

Öğr. Gör. Alper METİN

Bahar/2025

Cosmic Rumble: Unity 2021.3 LTS ile Çok Gezegenli, Özel Yerçekimi Alanlarına Sahip Sıra Tabanlı 2D Nişancı Oyunu Geliştirme

# İÇİNDEKİLER

[İÇİNDEKİLER 2](#_Toc200737985)

[1.TANIMLAR 4](#_Toc200737986)

[2. PROJENİN AMACI 5](#_Toc200737987)

[3. ÖNEM VE KATKI 6](#_Toc200737988)

[3.1 Çoklu Yerçekimi Mekaniklerinin Gösterimi 6](#_Toc200737989)

[3.2 Parçalanabilir Ortam 6](#_Toc200737990)

[3.3 Sıra Tabanlı Oyun Altyapısı 6](#_Toc200737991)

[3.4 Çok Oyunculu Genişleme Potansiyeli 6](#_Toc200737992)

[3.5 Eğitsel Değer 6](#_Toc200737993)

[4. SINIRLILIKLAR 7](#_Toc200737994)

[4.1 Kullanılmayan ve Eksik Scriptler 7](#_Toc200737995)

[4.2 Mermi Yönetimi ve Bellek Kullanımı 7](#_Toc200737996)

[4.3 Mobil Platform Desteği Eksikliği 7](#_Toc200737997)

[4.4 Kamera ve Tekil Nesne (Singleton) Güvenliği 7](#_Toc200737998)

[4.5 Lisans ve Dokümantasyon Eksikliği 7](#_Toc200737999)

[4.6 Parçalanabilir Gezegen Performansı 7](#_Toc200738000)

[5. VARSAYIMLAR 8](#_Toc200738001)

[5.1 Tur Başına Tek Yetenek 8](#_Toc200738002)

[5.2 2D Yerçekimi Hesaplamaları 8](#_Toc200738003)

[5.3 Kullanılabilir UIManager 8](#_Toc200738004)

[5.4 Masaüstü Giriş Sistemi 8](#_Toc200738005)

[5.5 Gezegen Dağılımı 8](#_Toc200738006)

[5.6 Sabit Kare Hızı 8](#_Toc200738007)

[5.7 Yerel Oynanış 8](#_Toc200738008)

[6. YÖNTEM 9](#_Toc200738009)

[6.1 Geliştirme Araçları ve Ortam 9](#_Toc200738010)

[6.2 Temel Sistemler 9](#_Toc200738011)

[6.2.1 Yerçekimi Sistemi 9](#_Toc200738012)

[6.2.2 Sıra Sistemi 10](#_Toc200738013)

[6.2.3 Yetenek Sistemi 11](#_Toc200738014)

[6.2.4 Kullanıcı Arayüzü Yönetimi 11](#_Toc200738015)

[6.2.5 Yörünge Tahmini Sistemi 12](#_Toc200738016)

[6.2.6 Parçalanabilir Gezegenler 13](#_Toc200738017)

[6.3 Performans Kriterleri 13](#_Toc200738018)

[6.3.1 Mermi Havuzu (Object Pooling) 13](#_Toc200738019)

[6.3.2 Yerçekimi Hesaplamaları 14](#_Toc200738020)

[6.3.3 UIManager Erişimi 14](#_Toc200738021)

[6.4 Kod Organizasyonu 14](#_Toc200738022)

[6.5 Ölçeklenebilirlik ve Gelecek Çalışmalar 15](#_Toc200738023)

[6.5.1 Ağ Tabanlı Çok Oyunculu 15](#_Toc200738024)

[6.5.2 Yeni Giriş Sistemine Geçişin Tamamlanması 15](#_Toc200738025)

[6.5.3 Mobil Uyarlama 15](#_Toc200738026)

[6.5.4 Yetenek Yelpazesinin Genişletilmesi 15](#_Toc200738027)

[6.5.5 Birim Testleri ve CI/CD Süreci 15](#_Toc200738028)

[7. KAYNAKÇA 17](#_Toc200738029)

[8. EKLER 18](#_Toc200738030)

[Ek A – Kod Örneği: Yerçekimi Uygulaması 18](#_Toc200738031)

[Ek B – Proje Klasör Yapısı (Basitleştirilmiş) 18](#_Toc200738032)

[Ek C – Sıra Zamanlayıcı UI Tasarımı Örneği 19](#_Toc200738033)

[Ek D – Ekran Görüntüsü Açıklamaları 19](#_Toc200738034)

[9. SONUÇ 21](#_Toc200738035)

# TANIMLAR

**Yerçekimi**

 Bu projede yerçekimi, karakterleri ve mermileri yakındaki gezegenlerin merkezine doğru çeken kuvveti ifade eder. Bu, etki yarıçapı dahilindeki nesnelere çekim kuvveti uygulayan GravitySource bileşeni aracılığıyla gerçekleştirilmiştir. Kuvvetin büyüklüğü Newton’un ters kare yasasına uyar; yani nesne gezegenden uzaklaştıkça kuvvet zayıflar.

**Yerçekimi Alanı**

 Her gezegen, kendine özel bir GravitySource bileşeniyle tanımlanmış bir yerçekimi alanına sahiptir. Bu alan tipik olarak dairesel olup, menziline giren nesneleri gezegen merkezine çeker. Alan; mermileri, karakterleri ve diğer fiziksel cisimleri etkiler.

**Yetenek (Skill)**

 Oyuncuların kendi sıralarında etkinleştirebildiği özel güçlerdir. Bu projedeki örnekler arasında Tabanca, RPG, El Bombası, Süper Zıplama gibi deneysel yetenekler bulunur. Her yetenek, belirli bir bekleme süresi ile sınırlanmıştır ve bazıları sınırlı cephaneye sahiptir.

**Yetenek Kullanımının Tur Tabanlı Sınırlandırılması**

Bu projede, her oyuncu kendi turunda yalnızca bir adet yetenek kullanabilir.  
Bu kural, oyuncunun her turu daha dikkatli ve stratejik şekilde değerlendirmesini sağlar. Aynı tur içinde birden fazla yetenek kullanımına izin verilmediğinden, oyuncular her hamlesini öncelik sırasına göre planlamak zorundadır.  
Bu sistem, hem oyunun taktiksel derinliğini artırmakta hem de oyun dengesini korumaktadır.

**Mermi (Projectile)**

 Bir yetenekten fırlatılan, sahnede hareket eden ve patlama veya hasara neden olabilen objelerdir. Bunlar mermi, roket, el bombası gibi nesneleri kapsar. Yerçekiminden etkilenir ve yok edilebilir gezegenlerle ya da karakterlerle çarpışabilir.

**Yörünge (Trajectory)**

 Bir merminin fırlatıldıktan sonra izleyeceği tahmini yoldur. Projede LineRenderer bileşeni ile eğrisel bir çizgi çizilerek bu yol gösterilir; bu, oyuncuların daha isabetli nişan almasına yardımcı olur.

**Sıra Tabanlı (Turn-Based)**

 Oyun, oyuncuların aynı anda değil sırayla hareket ettiği sıra tabanlı bir sistemle oynanır. Bu sistemde TurnManager adlı bir script, aktif oyuncuyu kontrol eder ve yalnızca bir karakterin hareket etmesini sağlar.

**Kullanıcı Arayüzü Öğeleri (UI Elements)**

 Kullanıcı arayüzü öğeleri; sağlık çubuğu, yetenek ikonları, cephane sayaçları, bekleme süresi göstergeleri ve sıra zamanlayıcısı gibi bileşenleri içerir. UIManager bu bileşenleri düzenler, oyuncuya anlık geri bildirimler sağlar ve yetenek durumlarını takip etmelerini kolaylaştırır.

# 2. PROJENİN AMACI

Bu projenin temel amacı, oyun mekaniği ile fiziksel simülasyonu birleştiren yenilikçi bir oyun deneyimi sunmaktır. Proje kapsamında, yerçekiminin oynanışta merkezi rol oynadığı sıra tabanlı bir 2D nişancı oyunu geliştirilmiştir. Geleneksel nişancı oyunları genellikle düz çizgisel nişan alma sistemiyle sınırlıdır; bu projede ise gezegenlerden yayılan yerçekimi alanları, mermi yörüngelerini karmaşık hale getirerek eğrilik, sapma ve düşüş gibi dinamikleri devreye sokar.

Bu durum, her turun stratejik kararlar içermesini sağlar. Oyuncular, merminin hedefe ulaşmasını etkileyen yerçekimi etkilerini önceden tahmin ederek hareket etmeli ya da saldırı planlamalıdır.

Teknik açıdan proje; çoklu yerçekimi kaynaklarının, Parçalanabilir arazilerin ve sıra tabanlı hareket sisteminin Unity oyun motoru üzerinde nasıl uygulanacağını göstermektedir. Newton fiziğine dayanan yerçekimi hesaplamaları, projeye fiziksel otantiklik kazandırır. Parçalanabilir gezegenler ise oyun sırasında çevresel değişikliklere olanak tanıyarak dinamik bir alan yaratır.

Bu prototip, gelecekte çok oyunculu ağ altyapısı, gelişmiş yetenek sistemleri ve çoklu platform desteği gibi özelliklerle genişletilebilecek bir temel sağlar. Sonuçta amaçlanan, teknik olarak sağlam ve oynanabilir bir oyun yapısının oluşturulmasıdır.

# 3. ÖNEM VE KATKI

## 3.1 Çoklu Yerçekimi Mekaniklerinin Gösterimi

Çok az sayıda 2D oyun, sıra tabanlı savaş sistemini gerçekçi yerçekimi simülasyonlarıyla birleştirir. Bu projede geliştirilen özel yerçekimi sistemi ve yörünge tahmin algoritması, fizik temelli oynanışı oyun tasarımına entegre etmenin güçlü bir örneğidir.

## 3.2 Parçalanabilir Ortam

Projede kullanılan **Destructible2D** paketi sayesinde gezegenler hasar aldıkça şekil değiştirebilmektedir. Unity oyun motoru varsayılan olarak bu tür bir gerçek zamanlı sprite yıkımı sunmadığı için bu çözüm, benzer projelerde kullanılabilecek değerli bir uygulama örneğidir.

**

## 3.3 Sıra Tabanlı Oyun Altyapısı

TurnManager ve ilgili scriptler, oyuncu sıralarının yönetildiği tekrar kullanılabilir bir yapı sunar. Bu yapı, oyun geliştiriciliğine yeni başlayanlar için durum geçişlerini ve sıra yönetimini anlamada öğretici bir model sağlar.

## 3.4 Çok Oyunculu Genişleme Potansiyeli

Mevcut proje çevrimdışı çalışmakla birlikte, kod mimarisi Photon Fusion gibi Unity tabanlı çok oyunculu sistemlerle entegre edilmeye uygundur. Ağ destekli oyunlar günümüz oyunlarının çoğunda bulunduğundan, bu yönde planlanan genişleme önemlidir.

## 3.5 Eğitsel Değer

Kod tabanı, bazı eksiklerine rağmen öğretici bir yapıdadır. İçerisinde nesne havuzu kullanımı (planlanmış fakat eksik), bekleme süresi (cooldown) yönetimi ve olay tabanlı kullanıcı arayüzü güncellemeleri gibi önemli yazılım geliştirme konuları yer almaktadır. Öğrenciler için gerçek dünya projelerinde karşılaşılabilecek sorunların analizi ve çözüm yollarını içeren bir örnek teşkil eder.

# 4. SINIRLILIKLAR

Projenin statik kod analizi, aşağıda ayrıntılı olarak açıklanan çeşitli sınırlılıkları ortaya koymuştur:

## 4.1 Kullanılmayan ve Eksik Scriptler

TeleportSkill.cs, ObjectSpawnSkill.cs ve TrajectoryPredictor.cs gibi bazı scriptler eksik veya kullanılmaz durumdadır. Bu kısımlar ileriki zamanda tamamlanacak kısımlar arasındadır.

## 4.2 Mermi Yönetimi ve Bellek Kullanımı

Mermiler, her atışta Instantiate() komutu ile oluşturulmakta ve çarpışmanın ardından yok edilmektedir. Bu yaklaşım, uzun oyun seanslarında bellekte yetersiz kaynak yönetimi nedeniyle performans kayıplarına neden olabilir. Nesne havuzu (object pooling) kullanılmaması bir eksikliktir.

## 4.3 Mobil Platform Desteği Eksikliği

Oyunun mobil versiyonu hedeflenmiş olmasına rağmen, dokunmatik kontroller ve mobil arayüz tasarımları henüz uygulanmamıştır.

## 4.4 Kamera ve Tekil Nesne (Singleton) Güvenliği

Bazı scriptler, sahnede her zaman Camera.main ve UIManager.Instance gibi singleton nesnelerin mevcut olduğunu varsayar. Ancak bu nesnelerin sahnede bulunmadığı özel test senaryolarında NullReferenceException hataları oluşabilir.

## 4.5 Lisans ve Dokümantasyon Eksikliği

Projede açık bir lisans dosyası (LICENSE) bulunmamaktadır. Bu durum, katkı sağlayacak kişilere kullanım hakları konusunda belirsizlik yaratır.

## 4.6 Parçalanabilir Gezegen Performansı

Gezegenlerin her patlamada collider'larının (çarpışma alanı) tamamen yok edilip yeniden oluşturulması, bellek yönetimi açısından verimsizdir ve özellikle zayıf cihazlarda performans sorunlarına neden olabilir.

# 5. VARSAYIMLAR

Bu çalışmanın temelini oluşturan bazı teknik ve yapısal varsayımlar şunlardır:

## 5.1 Tur Başına Tek Yetenek

Oyuncuların her turda yalnızca bir yetenek kullanmasına izin verilmektedir. Bu varsayım, sıra yönetimi yapısını basitleştirmekte ancak hızlı tempolu oyun deneyimini sınırlamaktadır.

## 5.2 2D Yerçekimi Hesaplamaları

Tüm yerçekimi hesaplamaları iki boyutlu düzlemde yapılır. Z (derinlik) ekseni tamamen göz ardı edilmiştir. Bu yaklaşım, sistemin yönetimini kolaylaştırmaktadır.

## 5.3 Kullanılabilir UIManager

Kullanıcı arayüz sistemi, UIManager.Instance isimli singleton kalıbını kullanır. Bu nedenle, oyun içindeki tüm scriptler her sahnede mevcuttur.

## 5.4 Masaüstü Giriş Sistemi

Kodlar, klavye girdisi (Input.GetKeyDown) gibi masaüstü tabanlı kontrolleri temel alır.

## 5.5 Gezegen Dağılımı

Sahnede bulunan tüm gezegenlerin GravitySource ve çarpışma bileşenlerine (collider) sahiptir. Mermi yörüngeleri, bu varsayıma dayanılarak hesaplanır.

## 5.6 Sabit Kare Hızı

Fiziksel hesaplamalar ve yörünge simülasyonları, düzenli bir güncelleme hızı (frame rate) üzerine inşa edilmiştir.

## 5.7 Yerel Oynanış

Projede herhangi bir ağ kodlaması (multiplayer) bulunmadığından tüm işlemler yerel olarak gerçekleştirilir. Gelecekteki çok oyunculu genişlemelerde bu varsayımların yeniden ele alınması gerekecektir.

# 6. YÖNTEM

## 6.1 Geliştirme Araçları ve Ortam

**Unity Sürümü:**

 Bu proje, Unity 2021.3 LTS sürümü kullanılarak geliştirilmiştir. LTS (Long-Term Support) sürümleri, uzun vadeli destek ve kararlılık sunduğu için özellikle profesyonel ve sürdürülebilir projelerde tercih edilmektedir.

**Programlama Dili:**

 Kodlama dili olarak C# tercih edilmiştir. Unity ekosistemi için standart olan bu dil sayesinde MonoBehaviour, olay yönetimi ve coroutine yapıları gibi bileşenler etkin bir şekilde kullanılmıştır.

**Kullanılan Paketler:**

 Projede çeşitli Unity paketleri manifest.json üzerinden tanımlanmıştır. Dikkate değer bazıları:

* **Universal Render Pipeline (URP)**
* **Visual Scripting**
* **Timeline**
* **Destructible2D** (Gezegenlerin Parçalanabilirliğini sağlamak amacıyla)

## 6.2 Temel Sistemler

### 6.2.1 Yerçekimi Sistemi

**GravitySource.cs**

 Her gezegen, yerçekimi kuvveti ve etki yarıçapı gibi parametreleri içeren bir GravitySource bileşeni taşır. Bu bileşen, sahnedeki tüm yerçekimi kaynaklarını AllSources listesinde toplar ve böylece diğer scriptler tarafından erişilebilir hale getirir. Kuvvet hesaplaması Newton’un yasalarına dayanmaktadır (2D sadeleştirmesiyle):

Vector2direction=(source.position-body.position);   
float forceMagnitude = gravityConstant \* source.mass / direction.sqrMagnitude;   
Vector2gravity=direction.normalized\*forceMagnitude;

**GravityBody.cs**

 Bu script, bir objeye uygulanan toplam yerçekimi kuvvetlerini hesaplar.

FixedUpdate() içerisinde her kaynakla mesafe ölçülür, kuvvet toplanır ve Rigidbody2D bileşenine aktarılır. Aynı zamanda objenin yönelimi, yerçekimine zıt olacak şekilde ayarlanır.

**GravityManager.cs**

GravityManager.cs, oyundaki tüm yerçekimi kaynaklarının (GravitySource) merkezi bir şekilde yönetilmesi amacıyla oluşturulmuş olan bir singleton sınıftır.Bu sınıf, sahnede bulunan tüm GravitySource bileşenlerini bir koleksiyon içerisinde tutarak, performans ve kod bütünlüğü açısından önemli avantajlar sağlamayı hedefler.

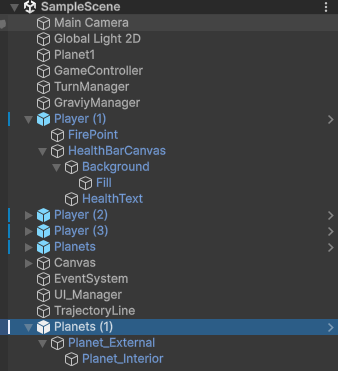
Normalde her yetenek veya mermi script’i sahnedeki tüm gezegenleri FindObjectsOfType<GravitySource>() ile bulmakta ve bu işlem her çalıştığında performans maliyeti oluşturmaktadır. GravityManager, bu işlemi tek bir noktada yaparak tekrarlı taramaları engellemeyi amaçlar.

**Gezegenin Katmanlı (İç-Dış) Yapısı**

Her gezegen, iki ayrı GameObject bileşeninden oluşan bir **katmanlı mimariye** sahiptir:

* **Planet\_External** objesi, gezegenin Parçalanabilir dış yüzeyini temsil eder. Bu objede DestructiblePlanet scripti ve Polygon Collider 2D bileşeni bulunur. Mermiler ve bombalar bu dış katmana çarparak deformasyon yaratır.
* **Planet\_Interior** objesi ise gezegenin sabit çekirdek kısmıdır. İçerisinde GravitySource scripti barındırır ve çekim alanının merkezi olarak görev yapar. Dış yüzey yıkılsa bile iç kısım yerçekimi uygulamaya devam eder.

Bu ayrım sayesinde, görsel hasar ve fiziksel çekim birbirinden bağımsız olarak yönetilebilir. Yüzey tahribatı, gezegenin çekim kuvvetini doğrudan etkilemez, bu da hem tasarımsal esneklik hem de oyuncu deneyiminde süreklilik sağlar.



A screenshot of a video game

AI-generated content may be incorrect.

### 6.2.2 Sıra Sistemi

**TurnManager.cs**

 Bu script, oyuncu sıralarını yönetir. Karakterler bir dizi içerisinde tutulur ve aktif oyuncu izlenir. Her turun başında karakterin yetenekleri yenilenir, süre sıfırlanır ve oyuncunun hamlesi beklenir. Örneğin: bir yetenek kullanımı sonrası Tab tuşuna basılarak sıra sonlandırılır.

**TurnTimerUI.cs**

 Her tur için bir sayaç görseli güncellenir. Geri sayım sırasında doluluk oranı ve renk değişimi animasyonlarla yansıtılır. Süre bitince otomatik olarak bir sonraki oyuncuya geçiş sağlanır.

** **

### 6.2.3 Yetenek Sistemi

Yetenekler, MonoBehaviour’dan türeyen ayrı scriptlerdir. Örnekler:

* Pistol.cs
* Rpg.cs
* HandGrenade.cs
* SuperJump.cs
* ShieldSkill.cs

 vb.

**Aktivasyon:**

 1–6 numaralı tuşlarla seçilir. Hedef belirlemek için mermi yörüngesi bir çizgi ile gösterilir. Onaylama Enter veya fare tuşu ile yapılır.

**Bekleme Süresi:**

 Her yetenek bir turda sadece bir kere kullanılabilir. Kullanıldıktan sonra yeniden kullanılabilir hâle gelmesi için bir sonraki turn’e kadar beklenmelidir. UIManager bu süreyi grafiksel olarak günceller. Cephane sayısı da takip edilir; cephane sıfırsa ikon siyah renge bürünür.

**Mermi Fırlatma:**

 Yetenek kullanımı sonrası mermi prefab'ı oluşturulur, hedefe yönlendirilir ve başlangıç hızı atanır. Ardından yerçekimi etkisiyle doğal bir hareket izler.

**Farklı Sistemi Olan Yetenekler:**

1. ***El Bombası (Hand Grenade)***

El bombası diğer mermilerin aksine, çarpma anında patlamaz, bunun yerine zaman gecikmeli bir patlama mekanizması ile çalışır.

El bombası nesnesi, HandGrenadeProjectile sınıfı aracılığıyla sahneye yerleştirilir. Bu sınıf içinde Invoke(nameof(Explode), delay) komutu kullanılarak belirli bir süre (örneğin 7 saniye) sonra Explode() fonksiyonu tetiklenir. Bu gecikme sayesinde bomba çarptıktan sonra bir süre sahnede kalır, bu da oyuncuya daha geniş bir stratejik kullanım alanı tanır.

Patlama gerçekleştiğinde:

* Patlama merkezi etrafında bir çarpışma kontrolü yapılır.
* Yakın çevredeki gezegen dokusu hasar alır ve Destructible2D sistemiyle deformasyon uygulanır.
* Eğer patlama menzili içerisinde bir karakter varsa, bu karaktere hasar verilir.

Ek olarak, bombanın erken yok edilmesi durumunda (örneğin bir gezegene saplanmadan yok edilirse), CancelInvoke() gibi mekanizmaların çağrılarak patlama fonksiyonunun gereksiz yere çalıştırılmaması sağlanması gerekir. Bu optimizasyon, özellikle nesne havuzu (object pool) sistemine geçiş yapılması durumunda önem kazanacaktır.

Bu yapı sayesinde el bombası, ani değil zaman ayarlı ve stratejik bir tehdit unsuru olarak kullanılabilir. Patlama süresini tahmin ederek hareket etmek, oyuncuların konum alma ve saldırı kararlarını daha dikkatli planlamasını gerektirir.

1. ***Süper Zıplama (Super Jump):***

Süper Zıplama, oyuncu karakterinin standart zıplama hareketine kıyasla çok daha güçlü bir şekilde yerçekimi yönünün tersine doğru fırlamasını sağlayan özel bir yetenektir. Bu yetenek, oyuncunun bulunduğu gezegenin yüzeyine bağlı olarak farklı yönlere doğru gerçekleşebilir. Örneğin; karakter üstte, altta veya yan yüzeyde yer alıyorsa, zıplama daima yüzeye dik doğrultuda uygulanır.

**Uygulama Detayları:**

* SuperJump.cs script’i, Rigidbody2D.AddForce() fonksiyonu kullanarak yerçekimi yönünün tersine, belirli bir çarpan (superMultiplier) ile kuvvet uygular.
* Yetenek, sadece oyuncunun kendi sırası içinde kullanılabilir ve kullanıldıktan sonra bir sonraki tura kadar tekrar kullanılamaz.
* Zıplama sonrası isGrounded durumu yeniden güncellenene kadar oyuncu havada kalır.
* Süper Zıplama yeteneği, UIManager üzerinden görsel olarak takip edilebilir. Cephane sayısı ve yetenek kullanımı arayüzde gerçek zamanlı yansıtılır.

Bu sistem, oyuncuya dikey hareket kabiliyeti sunarak hem kaçış hem de stratejik yeniden konumlanma imkânı sağlar. Özellikle farklı gezegen yüzeyleri arasında geçiş yapmak için etkili bir manevra sunar.

1. ***Kalkan (Shield):***

Kalkan yeteneği, oyuncu karakterinin bir tam tur boyunca gelen hasarları tamamen engelleyen defansif bir beceridir (sıra tekrar karaktere gelene kadar). Kullanıldığında, karakter üzerine görsel bir “enerji bariyeri” uygulanır ve bu bariyer aktif olduğu sürece karakter düşman saldırılarından etkilenmez.

***Teknik İşleyiş:***

* ShieldSkill.cs script’i, karakter üzerinde isShielded isminde bir bool değerini true olarak işaretler.
* Bu durum, CharacterHealth.cs veya mermi çarpışması kontrolü yapan scriptler tarafından kontrol edilir. Eğer isShielded == true ise hasarın %50’sinden korur.
* Görsel olarak aktiflik durumu, karakterin Sprite Renderer rengi üzerinden bir filtre efektiyle gösterilir (örneğin mavi renk tonlaması).

Bu yetenek, oyuncunun hamlesini güvenli şekilde gerçekleştirmesini sağlar ve özellikle düşman saldırılarına açık konumdayken kullanılabilecek güçlü bir savunma mekanizması sunar.

### 6.2.4 Kullanıcı Arayüzü Yönetimi

**UIManager.cs**

 Projede kullanıcı arayüzü işlemlerini yöneten UIManager scripti, singleton (tekil) yapıdadır. Yetenek ikonları, cephane sayaçları, bekleme süreleri ve sıra zamanlayıcısı gibi tüm bileşenler bu script üzerinden güncellenir. Kullanıcı arayüzü (UI), CharacterAbilities tarafından gönderilen olaylara tepki vererek arayüzdeki bilgileri anlık olarak günceller.

**TurnTimerUI.cs**

 Bu bileşen, her tur için çalışan dairesel zamanlayıcıyı kontrol eder. UpdateTimerDisplay() fonksiyonu, zaman azaldıkça dairesel doluluk oranını ve renkleri değiştirir. 5 saniyenin altına düştüğünde zamanlayıcı kırmızıya döner, bu da oyuncuya uyarı işlevi görür.

**HealthBarUI.cs**

 Her karakterin üzerinde bir sağlık barı bulunur. CharacterHealth bileşeni ile bağlantılıdır. Hasar alındığında barın doluluk oranı azalır. İsteğe bağlı olarak sayısal sağlık değeri de gösterilebilir.

** **

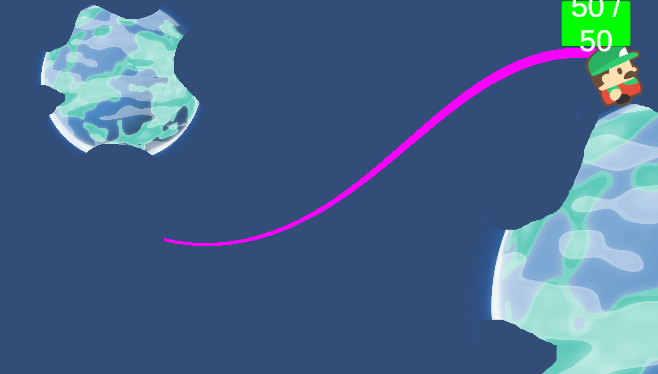
### 6.2.5 Yörünge Tahmini Sistemi

Oyuncu bir yetenek seçtiğinde, merminin izleyeceği yörünge LineRenderer ile görselleştirilir. Bu sistem sayesinde oyuncu, yerçekiminin mermi üzerindeki etkisini önceden görebilir ve daha doğru hedefleme yapabilir.

**Yöntem:**

1. Silah pozisyonundan, oyuncunun fare yönüne doğru bir ışın (ray) çizilir.
2. Merminin izleyeceği yol, adım adım simüle edilir.
3. Her adımda, tüm yerçekimi kaynaklarının etkisi hesaplanır.
4. LineRenderer bileşenine bu noktalar aktarılır.

Bu algoritma, birkaç düzine adımda hesaplama yaparak eğri bir çizgi oluşturur. Böylece yerçekimi altında merminin izleyeceği gerçekçi yol tahmin edilir.

**

### 6.2.6 Parçalanabilir Gezegenler

Projede **Destructible2D** eklentisi kullanılarak gezegenlerin gerçek zamanlı olarak hasar alması sağlanmaktadır. DestructiblePlanet.cs scriptinde yer alan ExplodeAtPoint() gibi fonksiyonlar, sprite dokusunda kraterler oluşturarak görsel ve fiziksel değişim yaratır.

Bu işlemin ardından PolygonCollider2D bileşeni yeniden oluşturulur. Böylece karakterler ve mermiler, yeni şekillenmiş arazi ile etkileşime girmeye devam edebilir.

**Performans Geliştirme Notu:**

 Collider her seferinde silinip yeniden oluşturulmaktadır. Bu işlem bellekte çöp üretimine yol açar. Mevcut yapının yerine collider yollarının doğrudan güncellenmesi, daha verimli bir yöntem olacaktır.

### 6.2.7 Yüzey Hareketleri

Oyuncu karakterinin, gezegen yüzeylerinde doğal ve fiziksel olarak tutarlı bir biçimde yürüyebilmesi için, yerçekimi yönüne göre dinamik hizalama işlemleri yapılmaktadır. Her fizik güncellemesinde karakterin transform.up vektörü, en yakın GravitySource bileşeninden hesaplanan yerçekimi yönünün tersi olacak şekilde ayarlanır. Böylece karakter, gezegenin yüzeyine her zaman "ayaklarıyla" hizalanır ve 360° yüzey hareketi mümkün hale gelir.

Zıplama işlemleri de bu hizalamaya göre gerçekleşir. Zıplama kuvveti, o anki yerçekimi yönüne dik olarak uygulanır. Bu sayede karakter üstte, altta ya da yanda olsa da zıplama her zaman yüzeyden dik şekilde ayrılacak biçimde gerçekleşir. Ayrıca, karakterin yere temas durumu (isGrounded) veya yüzeye olan mesafesi sürekli kontrol edilerek, zıplamanın veya sürtünmenin uygulanıp uygulanmayacağına karar verilir. Böylece karakter, yüzeyde sabitlenebilir, kontrollü bir şekilde yürüyebilir ya da havada süzülebilir.

## 6.3 Performans Kriterleri

### 6.3.1 Mermi Havuzu (Object Pooling)

Mevcut sistemde, her mermi atıldığında sahneye Instantiate() ile yeni bir nesne eklenmekte ve çarpışmadan sonra Destroy() komutu ile yok edilmektedir. Bu yaklaşım, uzun vadede çöp üretimi (GC spikes) ve performans kayıplarına yol açabilir. Alternatif olarak nesne havuzu (object pooling) sistemiyle mermiler yeniden kullanılabilir hale getirilmelidir.

### 6.3.2 Yerçekimi Hesaplamaları

Her silah ve mermi scripti, sahnedeki yerçekimi kaynaklarını FindObjectsOfType<GravitySource>() ile taramaktadır.

### 6.3.3 UIManager Erişimi

Projede birçok yerde UIManager.Instance'a doğrudan erişim yapılmakta, ancak null kontrolü yapılmamaktadır. Eğer UI sistemi sahnede yüklenmemişse NullReferenceException hataları alınabilir. Basit bir if (UIManager.Instance != null) kontrolü, bu hataların önüne geçecektir.

## 6.4 Kod Organizasyonu

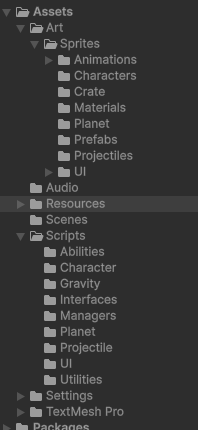
Projenin kaynak kodları işlevsel olarak mantıklı klasör yapıları içinde gruplanmıştır. Örneğin:

* **Abilities**: Pistol.cs, Rpg.cs, HandGrenade.cs gibi yetenek scriptleri
* **CharacterControllers**: Karakter hareketleri ve girdileri
* **Gravity**: Yerçekimi ile ilgili scriptler (GravityBody.cs, GravitySource.cs, GravityManager.cs)
* **Projectile**: Mermi ve roket gibi sahneye fırlatılan objeler
* **UI**: UIManager.cs, TurnTimerUI.cs, HealthBarUI.cs gibi kullanıcı arayüzü kontrol scriptleri

Bu yapı sayesinde her bileşen kendi sorumluluğuna odaklanmış ve bağımsız çalışabilir hale gelmiştir.

Ancak şu eksiklikler tespit edilmiştir:

* .asmdef (Assembly Definition) dosyaları eksiktir. Bu dosyalar olmadan tüm scriptler tek bir derleme birimi içinde yer almakta ve derleme süresi uzamaktadır.
* Kullanılmayan varlıklar ve scriptler klasörleri gereksiz şekilde kalabalıklaştırmaktadır (örneğin boş Resources klasörü, kullanılmayan ses dosyaları vb.).
* Düzenli bir bakım yapılmaması, projeyi büyüttükçe yönetimi zorlaştıracaktır.

**

## 6.5 Ölçeklenebilirlik ve Gelecek Çalışmalar

Projede kullanılan modüler yapı, aşağıdaki gelecek hedeflerine uygun zemin hazırlamaktadır:

### 6.5.1 Ağ Tabanlı Çok Oyunculu

Photon Fusion gibi çözümler entegre edilerek ağ tabanlı sıra yönetimi, yetenek senkronizasyonu, mermi hareketleri ve gezegen tahribatı gibi özellikler çok oyunculu ortama aktarılabilir.

### 6.5.2 Yeni Giriş Sistemine Geçişin Tamamlanması

Yeni giriş sistemiyle birlikte çoklu cihaz ve platform desteği daha esnek şekilde sağlanabilecektir. InputActions scriptleri zaten proje içinde bulunduğundan, mevcut sistemle bütünleştirmek kolaylaştırılmıştır.

### 6.5.3 Mobil Uyarlama

Mobil platformlar için ekran içi kontroller (joystick, butonlar) ve optimize edilmiş UI gereklidir. Mobil cihazlarda bellek daha sınırlı olduğundan nesne havuzu (object pooling) vazgeçilmez hale gelir. Ayrıca klavye yerine dokunmatik hareketler desteklenmelidir.

### 6.5.4 Yetenek Yelpazesinin Genişletilmesi

TeleportSkill ve ObjectSpawnSkill gibi şimdilik boş olan scriptler doldurulabilir. Ayrıca TrajectoryPredictor sisteminin bu yeni yeteneklerle entegre çalışması sağlanarak stratejik çeşitlilik artırılabilir.

### 6.5.5 Birim Testleri ve CI/CD Süreci

Projeye otomatik testler eklenerek yeni değişikliklerin hata üretip üretmediği kontrol edilebilir. GitHub Actions gibi araçlarla sürekli entegrasyon sistemi kurulabilir. Her commit sonrasında testlerin otomatik çalıştırılması ve build alınması sağlanabilir.

# 7. KAYNAKÇA

* Unity Technologies. *Unity Kullanıcı Kılavuzu 2021.3 LTS*.
* [https://docs.unity3d.com/2021.3/Documentation/Manual/](https://docs.unity3d.com/2021.3/Documentation/Manual/" \t "_blank)
* Unity Technologies. *Script Reference – Rigidbody2D*.

[https://docs.unity3d.com/ScriptReference/Rigidbody2D.html](https://docs.unity3d.com/ScriptReference/Rigidbody2D.html" \t "_blank)

* Unity Technologies. *Unity Input System*.

[https://docs.unity3d.com/Packages/com.unity.inputsystem@1.5/manual/](https://docs.unity3d.com/Packages/com.unity.inputsystem@1.5/manual/" \t "_blank)

* Unity Asset Store. *Destructible2D (Gerçek Zamanlı Sprite Tahribatı)*.

[https://assetstore.unity.com/packages/tools/sprite-management/destructible-2d-12282](https://assetstore.unity.com/packages/tools/sprite-management/destructible-2d-12282" \t "_blank)

* Newton, I. *Philosophiae Naturalis Principia Mathematica*.

 (Yerçekimi fiziğinin tarihsel temeli olarak)

* Game Development Patterns. *Unity’de Singleton Tasarımı ve Riskleri*.

 (Singleton yapılarının avantajları ve dezavantajları üzerine referans)

# 8. EKLER

## Ek A – Kod Örneği: Yerçekimi Uygulaması

// GravityBody.cs'ten alıntı   
private void FixedUpdate()   
{   
    Vector2 totalForce = Vector2.zero;   
    foreach (var source in GravitySource.AllSources)   
    {   
        Vector2 dir = (source.transform.position - transform.position);   
        float sqrDist = dir.sqrMagnitude;   
        if (sqrDist > 0.01f)   
        {   
            totalForce += dir.normalized \* source.Force / sqrDist;   
        }   
    }   
    rb.AddForce(totalForce);   
    transform.up = -totalForce.normalized;   
} 

Bu kod parçası, bir objenin tüm yerçekimi kaynaklarından gelen kuvvetleri toplayarak net ivmeyi oluşturmasını gösterir.

## Ek B – Proje Klasör Yapısı (Basitleştirilmiş)

/Assets   
    /Scripts   
        /Abilities   
            Pistol.cs   
            Rpg.cs   
            HandGrenade.cs   
            Shotgun.cs   
        /CharacterControllers   
            CharacterMovement.cs   
        /Gravity   
            GravityBody.cs   
            GravitySource.cs   
            GravityManager.cs   
        /Projectile   
            Projectile.cs   
            HandGrenadeProjectile.cs   
        /UI   
            UIManager.cs   
            TurnTimerUI.cs   
            HealthBarUI.cs   
    /Resources   
        (boş, sadece .meta dosyaları var)   
    /Scenes   
        SampleScene.unity   
        MenuScene.unity   
/Packages   
    manifest.json   
/ProjectSettings   
    (Unity proje ayar dosyaları)   
README.md 

Bu yapı, işlevsel alanlara göre düzenlenmiş dosya yapısını gösterir. Ancak bazı kullanılmayan içerikler hâlâ klasörlerde bulunmaktadır.

## Ek C – Sıra Zamanlayıcı UI Tasarımı Örneği

* Dairesel bir sayaç ekranın köşesinde yer alır.
* Sayı ile gösterilen geri sayım değeri, ortada yer alır.
* Sayaç, 5 saniyeden az kalınca yeşilden kırmızıya döner.
* Oyuncuya hamlesini tamamlaması gerektiğini görsel olarak bildirir.

**

## Ek D – Ekran Görüntüsü Açıklamaları

1. **Oynanış Sahnesi:**

 İki karakter, farklı gezegenlerin üzerinde durur. Biri diğerine doğru eğimli bir yörünge çizer. Gezegenin birinde mermi isabeti sonucu oluşmuş bir krater yer alır.

*A screenshot of a video game

AI-generated content may be incorrect.*

1. **Yetenek Arayüzü:**

 Ekranın alt kısmında, tabanca, RPG, el bombası gibi yetenek ikonları bulunur. Her biri mevcut cephane sayısını gösterir. Seçilen yetenek vurgulanır.

*A blue shield with a person running

AI-generated content may be incorrect.*

1. **Yörünge Tahmini:**

 RPG hedef alındığında, ekranda eğimli ve kesikli bir çizgi hedefe doğru uzanır. Yerçekimi nedeniyle eğilen bu çizgi, oyuncunun atışı doğru yapmasına yardımcı olur.

# 9. SONUÇ

**Cosmic Rumble**, karmaşık yerçekimi etkilerini sıra tabanlı bir nişancı oyunu mekanizması ile başarıyla birleştiren yenilikçi bir prototiptir. Parçalanabilir gezegenler, eğimli yörüngeler ve stratejik yetenek kullanımı sayesinde oyunculara taktiksel bir deneyim sunmaktadır.

Proje; özel fizik kuvvetleri, olay tabanlı kullanıcı arayüzü ve sıra yönetimi gibi Unity’nin temel yeteneklerini etkili biçimde kullanmaktadır. Ancak şu eksikliklerin giderilmesi, projenin daha sağlam ve genişletilebilir hâle gelmesini sağlayacaktır:

* **Object Pooling (Nesne Havuzu) uygulanmalı**
* **Giriş sistemi tamamen yeni Input System’a geçirilmelidir**
* **Singleton erişimleri güvenli hâle getirilmelidir**

Bu eksikliklerin çözülmesi, çok oyunculu destek ve mobil uyumluluk gibi ileri seviye özelliklerin eklenmesini kolaylaştıracaktır. Akademik ve pratik olarak değerlendirildiğinde, bu proje hem oyun geliştiriciler hem de bilgisayar mühendisliği öğrencileri için öğretici bir örnek teşkil etmektedir.