İÇİNDEKİLER

GİRİŞ

Bu proje, Unity oyun motoru ve C# programlama dili kullanılarak geliştirilmiş bir Düşük Poligon (Low-Poly) estetiğine sahip Üçüncü Şahıs Nişancı (TPS) oyunu prototipidir.

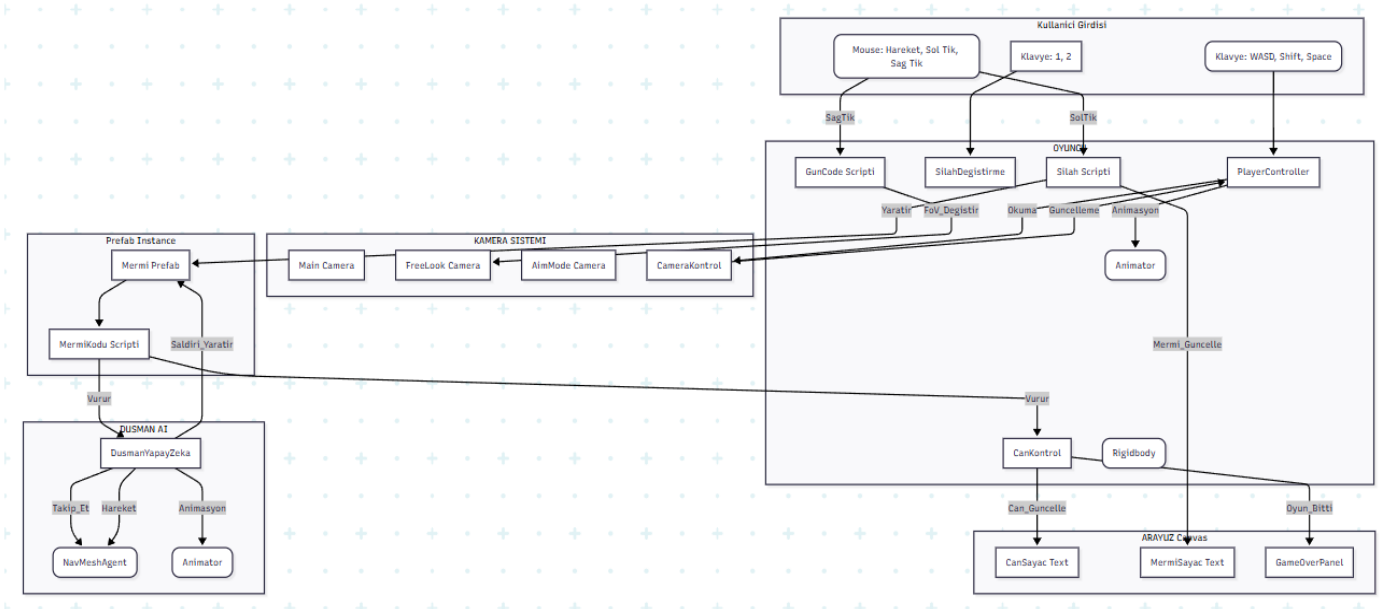
Projenin temel amacı, modern oyun geliştirme süreçlerini ve Nesne Yönelimli Programlama (OOP) prensiplerini uygulamalı olarak deneyimlemektir. Bu kapsamda, basit bir "sandbox" seviyesinde, baştan sona çalışır bir oyun döngüsü hedeflenmiştir.

Bu hedefe ulaşmak için, aşağıdakiler de dahil olmak üzere birçok temel TPS bileşeni sıfırdan geliştirilmiştir:

Fizik tabanlı (Rigidbody) bir karakter kontrolcüsü.
State Machine (Durum Makinesi) desenini kullanan bir düşman yapay zekası.
Soyutlama (Abstraction) ve "Kalıtım" (Inheritance) prensiplerine dayalı modüler bir silah sistemi.
Raycasting tabanlı dinamik 3D nişangah ve Cinemachine tabanlı modern bir kamera mekaniği.

Bu rapor, projenin tasarım kararlarını, teknik mimarisini (sistem şeması, OOP prensipleri), uygulanan mekaniklerin detaylı blok diyagramlarını, grafik tasarım sürecini ve geliştirme sürecinden elde edilen sonuçları detaylı bir şekilde sunacaktır.

1. Geliştirilen Sistemin Şeması ve Mimarisi



Oluşturulan sistem şeması, oyunun ana bileşenlerinin (Oyuncu, Düşman AI, UI, Kamera) birbirleriyle nasıl iletişim kurduğunu ve veri akışının nasıl sağlandığını göstermektedir. Proje, merkezi bir yönetici (GameManager) yerine, sorumlulukları doğrudan ilgili GameObject'lere dağıtan dağınık bir mimari kullanmaktadır.

Sistemler arasındaki iletişim, GetComponent veya GameObject.Find gibi Unity'nin temel bileşen referanslama yöntemleriyle sağlanmaktadır.

Aşağıda, şemadaki ana sistemlerin işleyişi detaylandırılmıştır:

1.1 Kullanıcı Girdisi (Input)

Sistemde merkezi bir InputManager script'i bulunmamaktadır. Girdiler, ilgili bileşenler tarafından doğrudan Update() metotları içinde okunur:

- **PlayerController:** Hareket (WASD), zıplama (Space) ve sprint (Shift) girdilerini yönetir.
- **GunCode:** Nişan alma (Mouse Sağ Tık) girdisini okur.
- **Silah Scripti:** Ateş etme (Mouse Sol Tık) girdisini okur.
- **SilahDegistirme:** Silah değiştirme (Klavye 1, 2) girdilerini okur.

1.2 Oyuncu Sistemi

Oyuncu GameObject'i, oyunun ana mantığının büyük bir kısmını barındırır ve sistemin merkezinde yer alır:

- **PlayerController:** Oyuncunun fizik tabanlı (Rigidbody) hareketinden, zıplamasından ve animasyonlarının tetiklenmesinden sorumludur.
- **CanKontrol:** Oyuncunun can değerini tutar. Hasar aldığında bu değeri düşürür ve arayüzdeki CanSayac metnini günceller. Can sıfıra düştüğünde Oyun_Bitti panelini aktif eder ve Time.timeScale = 0 yaparak oyunu durdurur.
- **Silah Scripti :** Ateş etme ve şarjör doldurma mantığının temelini oluşturur. arayüzdeki MermiSayac metnini doğrudan referans alarak günceller.
- **SilahDegistirme:** '1' ve '2' tuş girdilerine göre aktif silah GameObject'ini (Glock veya AK) değiştirir ve ilgili animasyon ile rig ayarlarını yapar.
- **GunCode:** Nişan alma girdisini aldığında, CinemachineFreeLook kamerasının Görüş Alanı değerini değiştirerek zoom efekti sağlar.

1.3 Düşman AI Sistemi

Dusman AI prefabı, DusmanYapayZeka script'i aracılığıyla otonom bir davranış sergiler:

- **Davranış:** Physics.CheckSphere ile oyuncuyu algılar ve State Machine (Durum Makinesi) deseni kullanarak Patrolling (Devriye), Chasing (Takip) veya Attacking (Saldırı) durumları arasında geçiş yapar.
- **Hareket:** NavMeshAgent bileşeni sayesinde NavMeshMap üzerinde akıllı hareket sağlar.
- **Saldırı:** Oyuncu menzile girdiğinde, Mermi Prefab'ını Instantiate ederek (yaratarak) saldırı yapar.
- **Hasar Alma:** Oyuncunun mermisi isabet ettiğinde TakeDamage fonksiyonu tetiklenir, canı azalır ve canı sıfıra düştüğünde "vefat” durumuna geçer.

1.4 Prefab (Mermi) Sistemi

Mermi, hem oyuncu hem de düşman tarafından kullanılan, dinamik olarak yaratılan bir bileşendir:

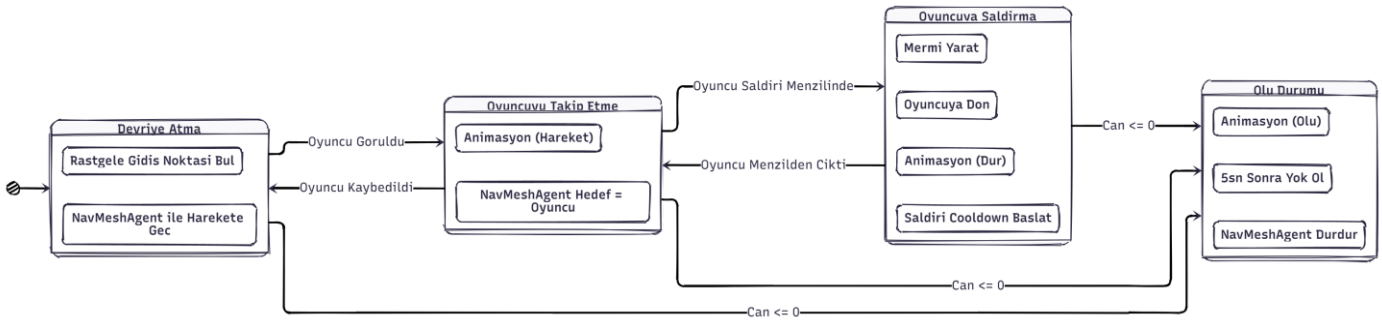
- **Yaratılma:** Silah Scripti veya DusmanYapayZeka tarafından Instantiate edilir.
- **MermiKoduScripti:** Merminin tüm mantığını içerir.
- **Çarpışma (Hit Detection):** Physics.OverlapSphere kullanarak hedefe veya bir engele çarpıp çarpmadığını kontrol eder.
- **Etkileşim:** Çarptığı nesnenin katmanına göre işlem yapar. Eğer çarptığı nesne Düşman ise, o düşmanın DusmanYapayZeka script'indeki TakeDamage fonksiyonunu çağırır. Eğer çarptığı nesne Oyuncu ise, oyuncunun CanKontrol script'indeki HasarAlma fonksiyonunu çağırır.

1.5 Arayüz (UI) ve Kamera Sistemleri

- **ARAYÜZ (Canvas):** CanKontrol ve Silah Scripti tarafından doğrudan güncellenen metin (TextMeshProUGUI) bileşenlerini (CanSayac, MermiSayac) barındırır.
- **KAMERA SİSTEMİ:** Cinemachine kullanılarak yönetilir. FreeLook Camera ve AimMode Camera (nişan alma modu) arasında geçiş yapar. CameraKontrol script'i, kameranın oyuncuyu takip etmesini ve oyuncunun yönünün kameraya göre ayarlanmasını sağlar.

2.Oyun Mekanikleri

2.1 Düşman Yapay Zekası



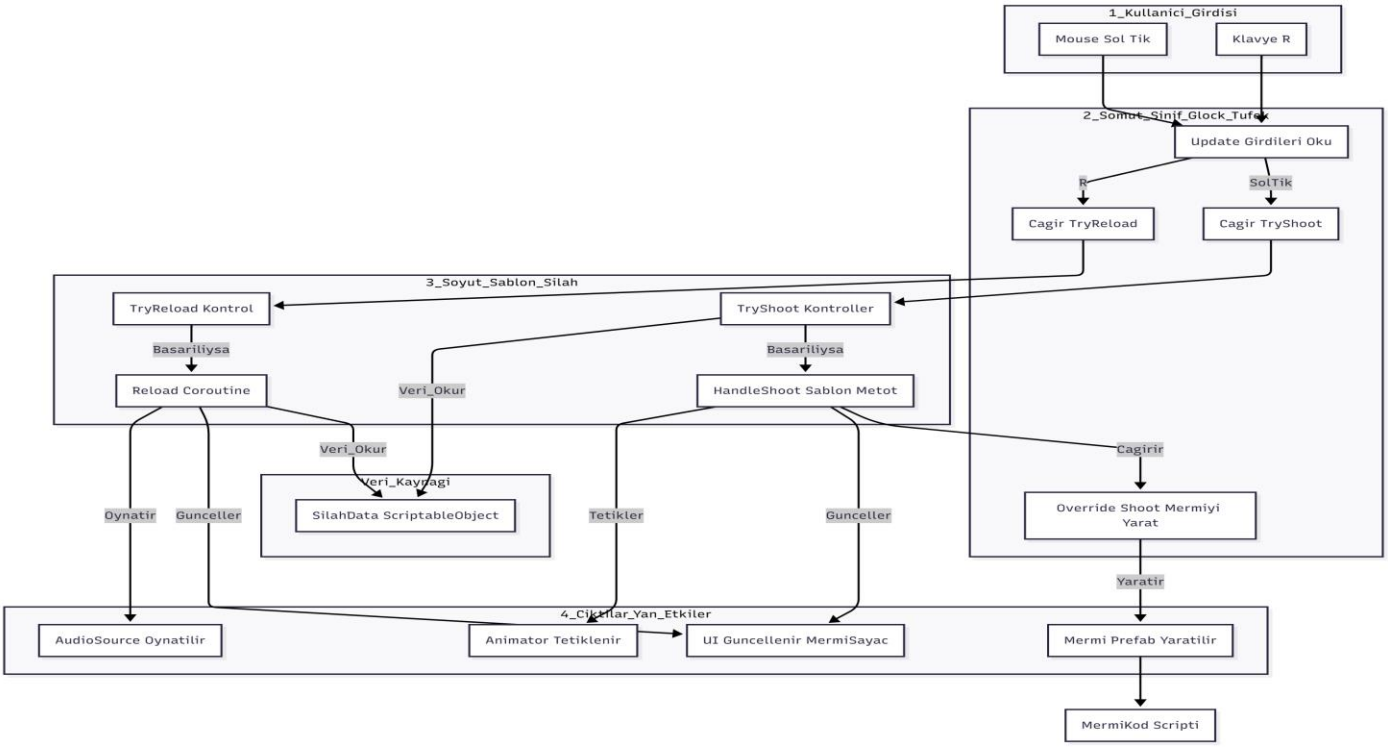
Projemizdeki DusmanYapayZeka.cs script'i, bu Durum Makinesi desenini Update() metodu içerisinde bir dizi if koşulu ile basit ve etkili bir şekilde uygular:

1. **Update() Metodu (Ana Kontrolcü):**
    a. İlk olarak if return; kontrolü yapılır. Eğer düşman "Vefat" durumundaysa, başka hiçbir mantık çalıştırılmaz. Bu, "Vefat" durumunu terminal bir durum yapar.
    b. Her karede playerInSightRange ve playerInAttackRange boolean'ları Physics.CheckSphere kullanılarak güncellenir.
    c. Bu boolean'ların durumuna göre Patrolling(), Chasing() veya Attacking() fonksiyonlarından sadece biri çağrılır.
2. **Durumlar (State'ler):**
    a. **Patrolling() (Devriye):** Düşmanın varsayılan durumudur. SearchWalkPoint() ile rastgele bir nokta belirler ve agent.SetDestination() ile o noktaya gider.
    b. **Chasing() (Takip):** Oyuncu görüş alanına girdiğinde tetiklenir. agent.SetDestination(player.position) komutu ile NavMeshAgent'ın hedefi doğrudan oyuncu olarak ayarlanır.

c. **Attacking() (Saldırı):** Oyuncu saldırı menziline de girdiğinde tetiklenir. Düşman agent.SetDestination(transform.position) ile durdurulur, transform.LookAt(player) ile oyuncuya döner ve MermiPrefab'ı yaratarak saldırır. alreadyAttacked boolean'ı ve Invoke("ResetAttack", ...) komutu, saldırılar arasında bir bekleme süresi olmasını sağlar.

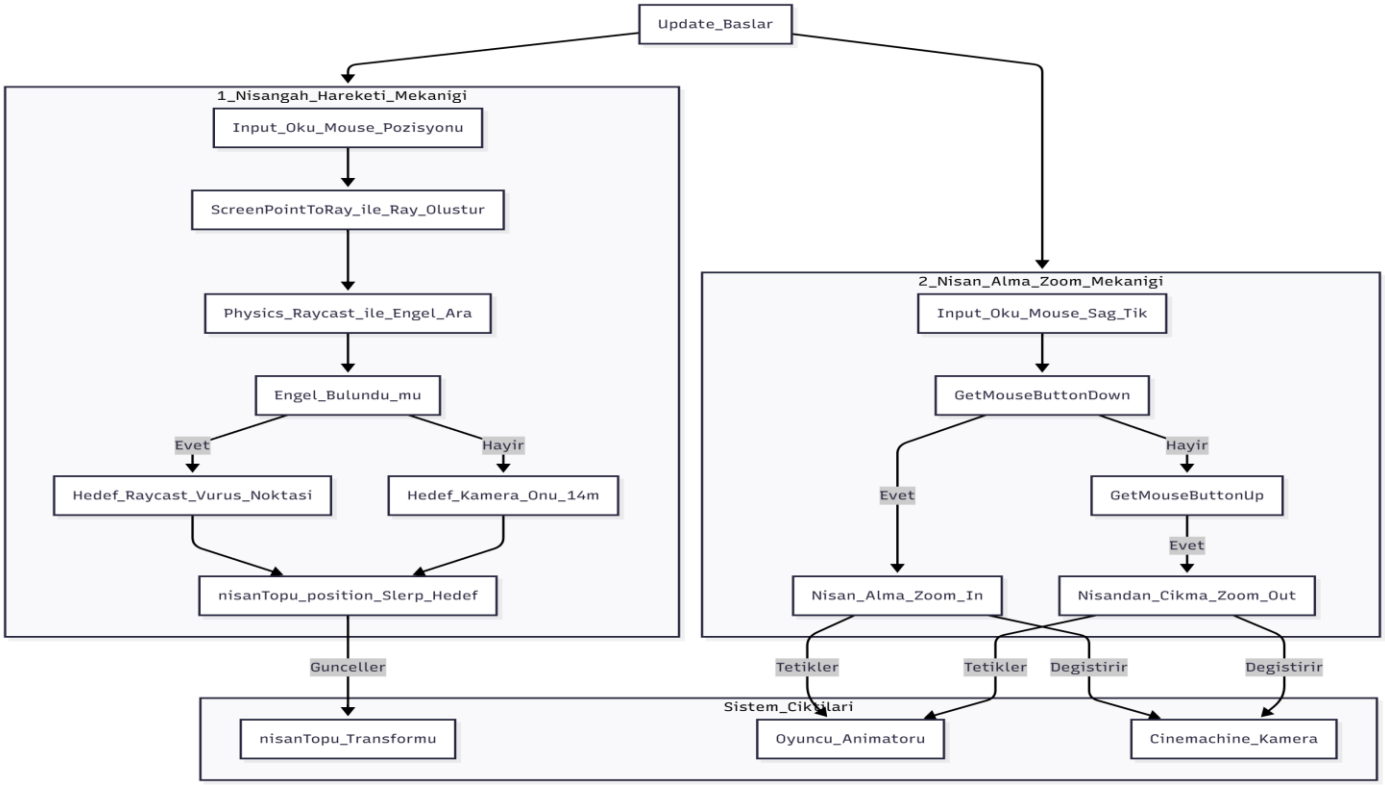
3. **Durum Geçişi (Transition):**
   a. Düşman hasar aldığında TakeDamage(float Damage) fonksiyonu tetiklenir.
   b. Eğer health <= 0 olursa, durum "Vefat" olarak değiştirilir (vefatmi = true). Bu, Update() metodunun başında kontrol edilerek yapay zeka mantığını kalıcı olarak durduran geçiş koşuludur.

2.2 Silah Sistemi



Silah soyut sınıfı, SilahData'dan bir veri referansı alır.
Hem Glock hem de Tufek, Silah sınıfından kalıtım alır. Bu, onların birer "Silah" olduğunu gösterir.
Silah sınıfındaki Shoot()* metodunun soyut (abstract) olduğunu, Glock ve Tufek içindeki Shoot() metotlarının ise bu soyut metodu ezdiğini (override) gösterir.
Hem Glock hem de Tufek sınıfları, kendi Shoot() metotları içinde MermiKod script'ine sahip bir prefab'ı yaratir.

2.3 Nişan Alma

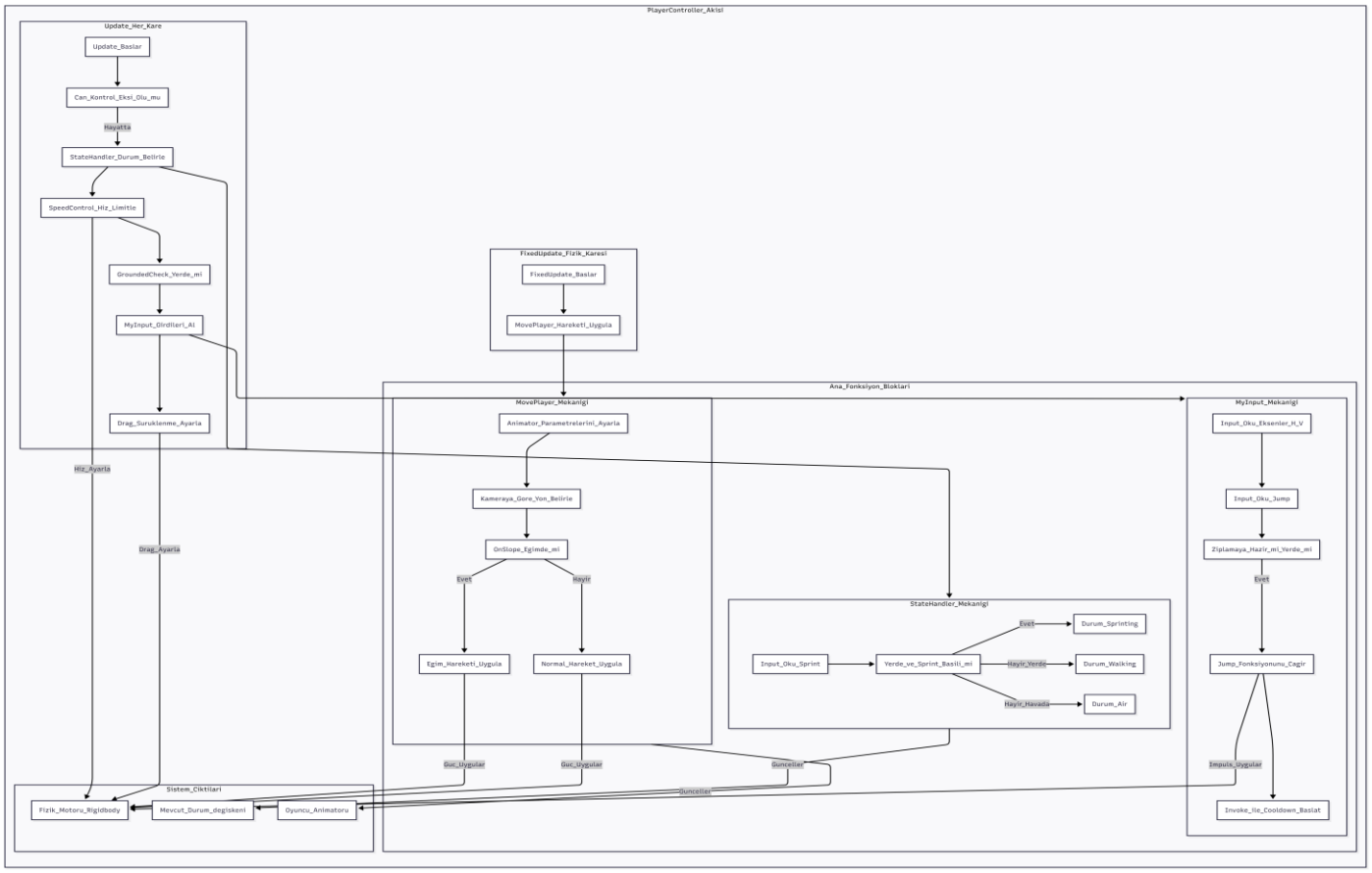


Diyagram, GunCode.cs script'inin Update() metodunda çalışan iki paralel süreci göstermektedir:

1. **Nişangah Hareketi Mekaniği :**
    a. Sürekli olarak mouse'un pozisyonunu okur.
    b. Bu pozisyondan 3D dünyaya bir Raycast fırlatır.
    c. Bir engele çarpıp çarpmamasına bağlı olarak bir Hedef noktası belirler.
    d. nisanTopu objesinin pozisyonunu bu hedefe doğru yumuşatarak günceller. Bu, Ciktilar'daki nisanTopu transformunu etkiler.
2. **Nişan Alma (Zoom) Mekaniği :**
    a. Mouse Sağ Tık girdisini dinler.
    b. Eğer tuşa basıldıysa (Karar_Basildi), Nisan_Al bloğunu tetikler. Bu, Ciktilar'daki animatörü ve Cinemachine kameranın FoV'unu 30'a ayarlayarak etkiler.
    c. Eğer tuş bırakıldıysa (Karar_Birakildi), Nisan_Birak bloğunu tetikler. Bu da animatörü ve Cinemachine FoV'unu 45'e geri döndürerek etkiler.

2.4 Oyuncu Kontrolleri

Oyuncu Hareket Akışı



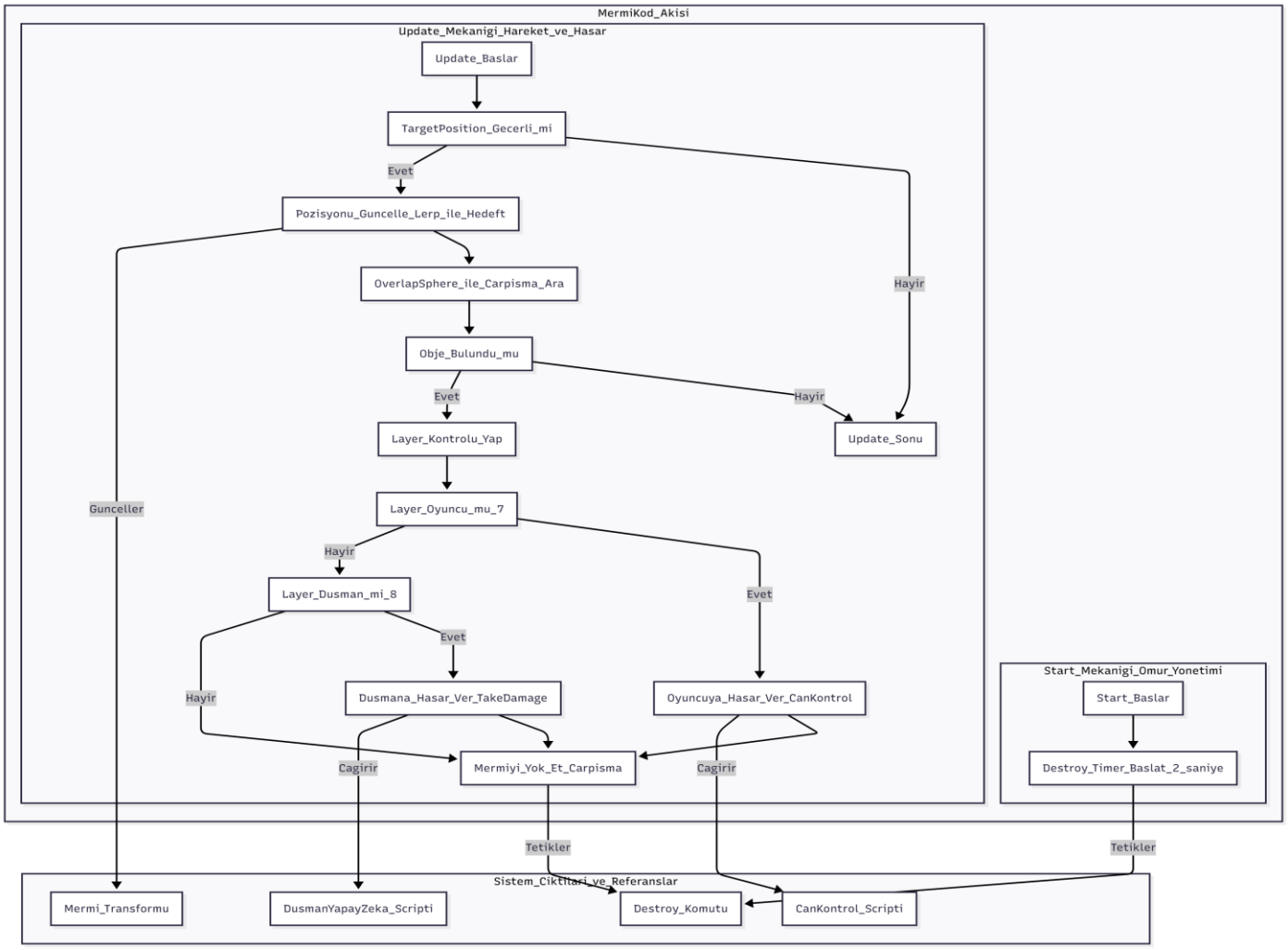
Bu diyagram, PlayerController script'inin Update ve FixedUpdate olmak üzere iki ana döngüde çalıştığını göstermektedir:

1. **Update_Her_Kare (Grafik Döngüsü):**
    a. Her karede çalışır.
    b. Can_Kontrol ile başlar.
    c. StateHandler mekaniğini çağırarak oyuncunun o anki MovementState'ini (sprinting, walking, air) belirler.
    d. SpeedControl ile maksimum hızı sınırlar.
    e. GroundedCheck (Raycast) ile oyuncunun yerde olup olmadığını belirler.
    f. **MyInput_Mekanigi**'ni çağırarak hareket ve zıplama girdilerini alır. Zıplama yapılırsa Invoke ile bekleme süresi başlatılır.
    g. Son olarak Drag ayarını yapar.

2. **FixedUpdate_Fizik_Karesi (Fizik Döngüsü):**
    a. Sabit bir zaman aralığında (genellikle 0.02 saniye) çalışır.
    b. Sadece fizik işlemlerinden sorumlu olan MovePlayer mekaniğini çağırır.

3. **MovePlayer_Mekanigi (Ana İş Yükü):**
    a. Animator'ü günceller.
    b. Hareket yönünü kameranın baktığı yöne göre belirler.
    c. OnSlope (eğim) kontrolü yapar.
    d. Karakterin durumuna göre Rigidbody'ye AddForce ile fiziksel bir kuvvet uygular (Sistem_Ciktilari'na giden oklar).

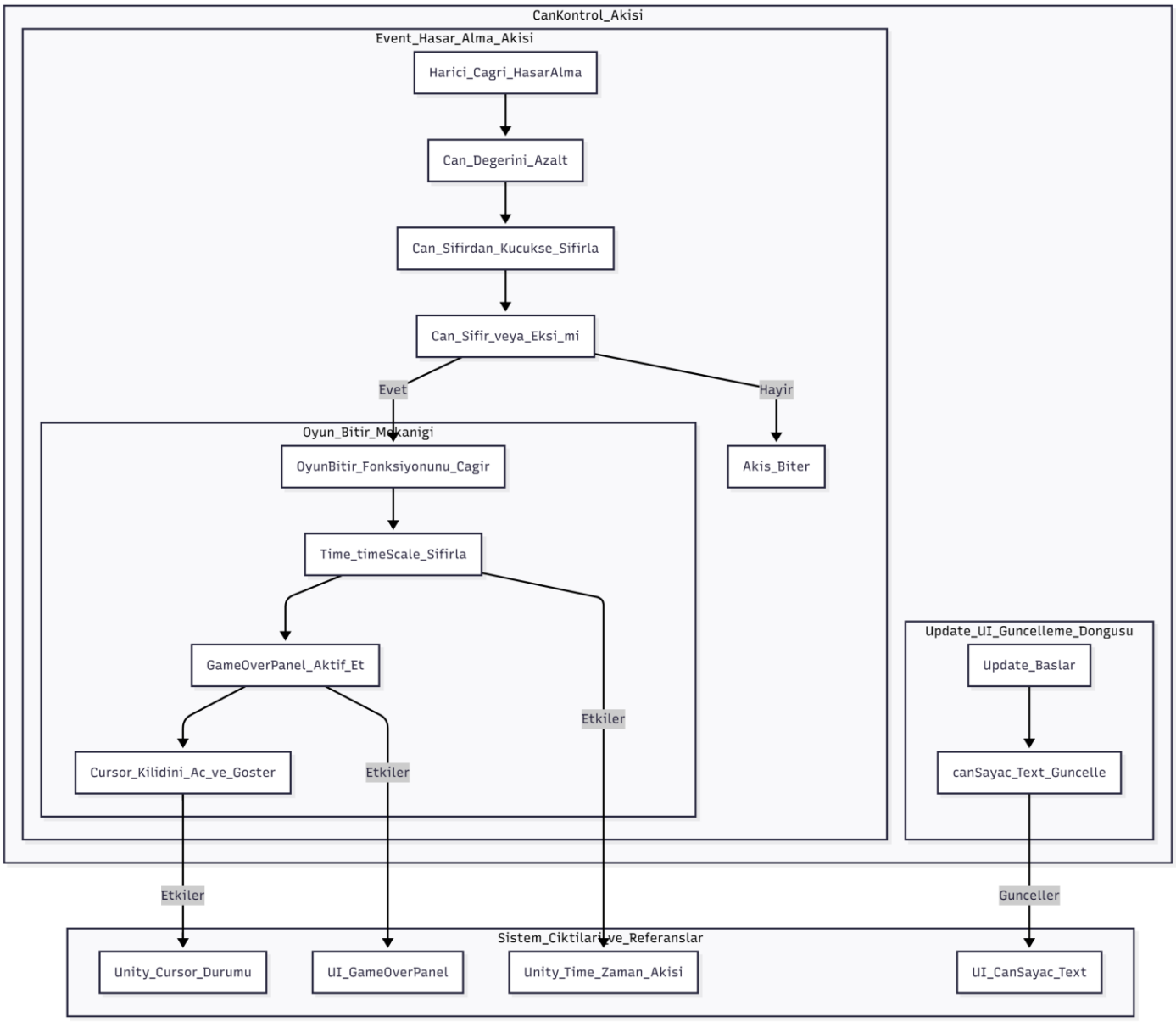
2.5 Mermi Sistemi



Diyagram, merminin iki ana sürecini gösterir:

1. **Start Akışı:** Mermi yaratıldığı anda 2 saniyelik bir yok olma zamanlayıcısı başlatır.
2. **Update Akışı:** Her karede mermi, hedefe doğru Lerp ile hareket eder. OverlapSphere ile bir çarpışma arar. Eğer bulursa, çarptığı nesnenin Layer'ına (Oyuncu veya Düşman) göre ilgili script'e (CanKontrol veya DusmanYapayZeka) hasar komutu gönderir ve ardından kendini yok eder.

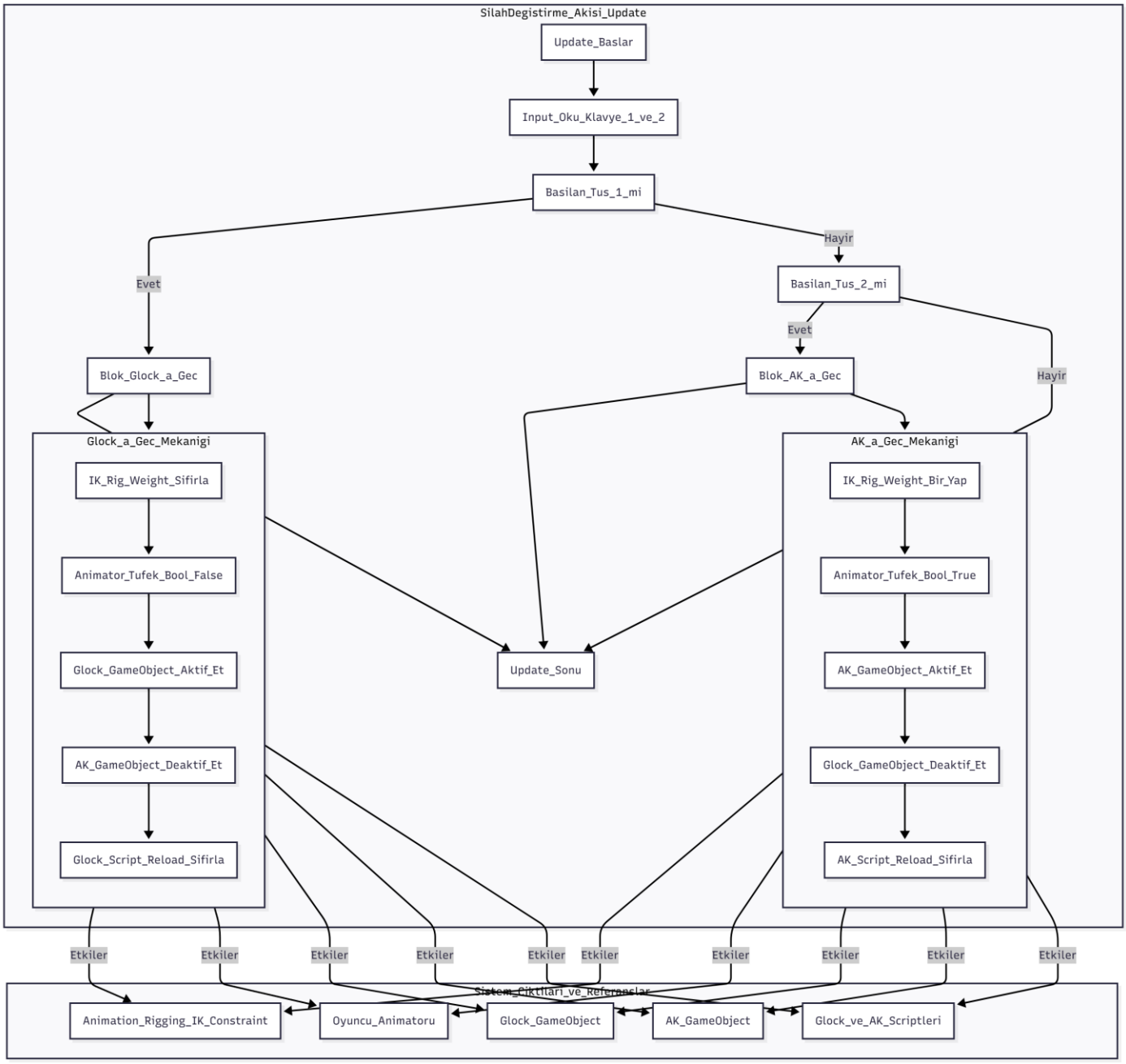
2.6 Can Kontrol Sistemi



Bu diyagram CanKontrol script'inin iki sürecini gösterir:

1. **Update Akışı (Sol Taraf):** Her karede çalışır ve sadece canSayac text'ini günceller.
2. **HasarAlma Akışı (Sağ Taraf):** Bu bir döngü değildir, MermiKod gibi harici bir script tarafından tetiklenir.
    a. Canı azaltır ve sıfırın altına düşmesini engeller.
    b. Can_Sifir_veya_Eksi_mi kontrolü yapar.
    c. Eğer can bitmişse, Oyun_Bitir_Mekanigi'ni tetikler. Bu blok, Time.timeScale'i 0 yaparak oyunu durdurur, GameOverPanel'i gösterir ve mouse'u serbest bırakır.

2.7 Silah Değiştirme



Script, '1' veya '2' tuşuna basılıp basılmadığını kontrol eder.

'1' Tuşuna Basılırsa (Glock_a_Gec):

IK_Rig'in ağırlığını 0 yapar (sol el serbest kalır).

Animator'e "Tufek" durumunun False olduğunu bildirir.

Glock modelini Aktif hale getirir, AK modelini Deaktif yapar.

Glock script'inin isReloading bayrağını sıfırlar.

'2' Tuşuna Basılırsa (AK_a_Gec):

IK_Rig'in ağırlığını 1 yapar (sol el tüfeği kavrar).

Animator'e "Tufek" durumunun True olduğunu bildirir.

AK modelini Aktif hale getirir, Glock modelini Deaktif yapar.

AK script'inin isReloading bayrağını sıfırlar.

Kullanılan Ana Mimariler, Yöntemler ve Teknikler

Soyutlama (Abstraction) ve Kalıtım (Inheritance): Silah.cs soyut sınıfı aracılığıyla tüm silahların ortak kodlarını (şarjör doldurma, ateş etme kontrolü) tek bir yerde toplamak ve Glock.cs/Tufek.cs gibi alt sınıfların bu kodları miras alarak kod tekrarını önlemesi için kullanılmıştır.

Çok Biçimlilik (Polymorphism) ve Template Method Deseni: ana HandleShoot metodunun ateş etme algoritmasının şablonunu (mermi azalt, animasyon oynat) çalıştırması, ancak asıl merminin yaratılma (Shoot) işleminin alt sınıflar (Glock, Tufek) tarafından ezilerek (override) belirlenmesi amacıyla uygulanmıştır.

ScriptableObject Mimarisi: Silahların hasar, atış hızı ve şarjör boyutu gibi verilerini SilahData.cs içinde koddan ayırarak, bu istatistiklerin Unity editöründen yönetilebilmesi için kullanılmıştır.

if/else Tabanlı Durum Makinesi (State Machine): DusmanYapayZeka.cs script'inin, oyuncunun menzilde olup olmadığına (playerInSightRange) bağlı olarak Patrolling, Chasing ve Attacking durumları arasında geçiş yapması için uygulanmıştır.

Enum Tabanlı Durum Makinesi: PlayerController.cs script'inin, MovementState enum'unu (walking, sprinting, air) kullanarak oyuncunun mevcut durumuna göre hareket hızını (moveSpeed) dinamik olarak ayarlaması için kullanılmıştır.

Fizik Tabanlı Kontrolcü (Rigidbody): Karakterin transform.position yerine Rigidbody.AddForce ile hareket ettirilerek fiziksel etkileşimlere (eğim, zıplama) duyarlı, ivmeli bir hareket sağlaması amacıyla tercih edilmiştir.

Raycasting (Işın Fırlatma): Hem GunCode.cs'da ekrandaki mouse pozisyonundan 3D dünyaya ışın atarak dinamik nişangahın yerini belirlemek hem de PlayerController.cs'da karakterin altındaki zeminin eğimini algılamak için kullanılmıştır.

Anlık Çarpışma Tespiti (OverlapSphere): MermiKod.cs script'inin, hızlı mermilerin hedefin içinden geçmesini (tunnelling) engellemek amacıyla OnCollisionEnter yerine Physics.OverlapSphere kullanarak her karede anlık çarpışma araması için kullanılmıştır.

Katman (Layer) Bazlı Filtreleme: MermiKod.cs script'inin, hit.gameObject.layer kontrolü yaparak merminin oyuncuya mı yoksa düşmana mı çarptığını ayırt etmesini ve hasarı doğru hedefe yönlendirmesini sağlamıştır.

NavMeshAgent (Yapay Zeka Navigasyonu): DusmanYapayZeka.cs script'inin, NavMeshMap üzerinde oyuncuyu akıllıca takip etmesi (Chasing) ve devriye atması (Patrolling) için kullanılmıştır.

Cinemachine (Modern Kamera): GunCode.cs script'inin, nişan alma (zoom) efektini CinemachineFreeLook kamerasının FieldOfView (Görüş Alanı) değerini değiştirerek kolayca uygulayabilmesi için entegre edilmiştir.

Animation Rigging (IK - Inverse Kinematics): SilahDegistirme.cs script'inin, silah değişimine göre (Glock vs AK) karakterin sol elinin TwoBoneIKConstraint ağırlığını (weight) değiştirerek silahı doğru kavramasını sağlaması için kullanılmıştır.

Oyun Durumu Yönetimi (Time.timeScale): CanKontrol.cs script'inin, oyuncu öldüğünde Time.timeScale = 0f komutuyla oyunun zaman akışını durdurması, "Oyun Bitti" panelini göstermesi ve mouse imlecini serbest bırakması amacıyla kullanılmıştır.

Tasarlanan Sayfalar

Oyun İçi Arayüzü

Sol Alt Köşe (Can Göstergesi): CanSayac olarak adlandırılan bu TextMeshPro alanı, CanKontrol.cs script'i tarafından yönetilir. Oyuncunun mevcut canını yeşil renkte gösterir (+100).
Sağ Alt Köşe (Mermi Göstergesi): MermiSayac olarak adlandırılan bu TextMeshPro alanı, aktif olan silahın (örn: Glock.cs veya Tufek.cs) Silah.cs script'inden miras aldığı mantıkla yönetilir. Mevcut mermi sayısını ve toplam şarjör kapasitesini gösterir (30/30)

Bu arayüz, oyuncunun canı sıfıra ulaştığında tetiklenir ve oyunun başarısızlık durumunu bildirir.

Ekran Ortası (Durum Metni): "OYUN BİTTİ" yazısı, GameOverPanel adlı bir GameObject içinde yer alır.
Tetiklenme: CanKontrol.cs script'i, oyuncunun canı sıfırlandığında OyunBitir() metodunu çağırır.
Eylemler: Bu metot, Time.timeScale'i durdurarak oyunu dondurur, bu GameOverPanel'i SetActive(true) yaparak görünür hale getirir ve oyuncunun menüde işlem yapabilmesi için mouse imlecini serbest bırakır.
Durum Yansıması: Görüntüde, oyunun "Oyun Bitti" durumuna geçtiği anda can göstergesinin +0 olduğunu ve mermi sayacının o anki durumu (-/30 şarjör değiştirirken) yansıttığını görebiliriz.

Grafikler ve Tasarım

Projenin görsel kimliği, bilinçli bir estetik tercih ve teknik bir gereklilik üzerine kurulmuştur. Gerçekçi (realistic) bir görünüm yerine, stilize ve "Low Poly" bir sanat tarzı benimsenmiştir.

Sanat Tarzı ve Gerekçesi

Oyunun "animasyon tarzı" görünümü, "Low Poly" estetiği ile sağlanmıştır. Bu tarzın seçilmesinin iki ana nedeni vardır:

1. **Estetik Uyum:** Düşük poligonlu modeller, sade ve temiz bir görünüm sunarak oyunun prototip aşamasında karmaşık ve dağınık görünmesini engeller.
2. **Optimizasyon:** Bu projedeki en önemli tasarım kararıdır. Düşük poligon sayılı (low poly) modeller kullanmak, üçgen ve köşe sayısını minimumda tutar. Bu, Unity motorunun her karede işlemesi (render) gereken geometrik yükü ciddi ölçüde azaltır. Bu sayede, daha düşük donanımlı sistemlerde bile akıcı bir oyun deneyimi sunmak hedeflenmiştir.

Varlık Üretim Süreci (Asset Pipeline)

Oyunun görsel varlıkları, Özgün Üretim ve Hazır Varlık olmak üzere iki farklı kaynaktan temin edilmiştir:

Özgün Tasarımlar (Blender): Projenin özgünlüğünü yansıtan ana unsurlar oyuncu karakter modeli ve oyuncunun kullandığı silah modelleri (Glock, AK) bizim tarafımızdan Blender yazılımı kullanılarak sıfırdan tasarlanmış ve modellenmiştir. Tasarımlar, projenin genel Low Poly sanat tarzına sadık kalarak oluşturulmuştur.
Harita Tasarımı (Unity Asset Store): Geliştirme sürecini hızlandırmak ve ana odağı kodlama, mekanik geliştirme ve özgün modellemeye ayırmak amacıyla, oyunun geçtiği tek seviye (level) ve çevresel objeler (binalar, konteynerler, duvarlar vb.) Unity Asset Store'dan temin edilen hazır bir Low Poly harita paketi kullanılarak tasarlanmıştır.

Proje Sürecinde Karşılaşılan Zorluklar ve Çözümler

Unity kullanımında deneyimsiz olunması proje başlangıcında birçok soruna sebep olmuştur. Öncelikle arayüz kontrolünü öğrenmede çok zaman harcanmıştır. Proje devamında karşılaştığımız bazı sorunlar ve çözümleri şunlardır:

Yapay zeka Hareketleri ve NavMesh Kullanımı: Asset olarak kullanılan map, öncesinde hazırlanan düşman yapay zekasının hareket kodlarını uygulamakta sorun yaratmıştır. Map öncesi hazırlanan hareket komutları yeni engellere ve zeminlere ayak uyduramamıştır. Bunu düzeltmek için NavMesh sistemini

defalarca otomatik tanıtma denenmiştir fakat başarılı olunamamıştır. En son çözümü yürünebilecek tüm zeminleri manuel olarak işaretleyerek düşman yapay zekasının hareket etmeme sorunu düzeltilmiştir.

Tasarımlar : Karakter ve silah tasarımlarını asset olarak kullanmak yerine, manuel tasarlanması tercih edilmiştir. Tasarımların hangi uygulamayla yapılacağı ve kullanılan uygulama hakkında bir bilgi sahibi olunmaması, tasarım sürecinde büyük zorluklara sebep olmuştur. Blender üzerinden tasarım yapılması karar verilmiş, ardından uygulama hakkında bilgi edinme sürecine girilmiştir. Tasarım sürecinin karmaşıklığı defalarca pes etme noktasına gelinmesine neden olmuştur. Silah tasarımları ve basit karakter tasarımı tamamlanmıştır. Oyun geliştirme süreci basit ana karakterle tamamlanmıştır. En son karaktere kıyafet ekleme aşamasında asıl sorun ortaya çıkmıştır. Blender üzerinden karaktere uygun kıyafet tasarımı bulunanamıştır, bulunanlar karaktere uymamıştır. Çözüm kıyafet giydirmek yerine karakter tasarımının parçalarının şekli ve rengi değiştirilerek bulunmuştur.

Tasarlanan Son Karakterin Oyuna Entegre Edilişi: Blender üzerinden tasarlanan son karakterin biten oyundaki basit karakter yerine oyuna ekleniş aşamasında birçok zorlukla karşılaşılmıştır. Bütün mekanikler basit karakterde olduğundan yeni karakterin baştan yapılışı çoğu mekanik bağlantısının tekrar yapılmasına yol açacaktır. Öncelikle Armature (iskelet) ve Cube (Materyal) kısımlarının basit karakterinkilerle değiştirilmesi denenmiştir fakat dosyaları export ve tekrardan import ederken değişen kod parçaları bir sürü hataya sebep olmuştur. Çözüm, atılan dosyalardaki kodların basit karakterdeki dosya isimleriyle manuel olarak teker teker değiştirilmesi yoluyla bulunmuştur.

Literatür Taraması ve Proje ile İlişkileri

Bu proje, Low-Poly sanat tarzını benimseyen bir Third-Person Shooter prototipidir. Literatür taraması, projemizin ilham aldığı veya benzer mekanikleri paylaştığı mevcut pazar örneklerini ve teknik yaklaşımları incelemeyi amaçlar.

Estetik Tarz:

Literatür : ***Unturned*** (2014)
Analiz: Unturned, Unity motoru kullanılarak tek bir bağımsız geliştirici tarafından yaratılan ve low-poly estetiğini bir hayatta kalma (survival) türüyle birleştiren ticari bir başarı örneğidir. Oyunun grafik tarzı projemizde de olduğu gibi iki ana amaca hizmet eder:

- **Stilistik:** Gerçekçilikten uzak, bloklu ve stilize bir dünya sunarak kendi özgün görsel kimliğini yaratır.
- **Optimizasyon:** Geometrik yükü minimumda tutarak (düşük poligon sayısı), çok geniş haritalarda ve düşük donanımlı sistemlerde bile akıcı bir performans (yüksek FPS) sağlar.

Proje Karşılaştırması: Projemiz, Unturned tarafından popülerleştirilen bu indie low-poly geliştirme felsefesini takip etmektedir. Karakter ve silah tasarımlarının projeye özel olarak Blender'da üretilmesi, bu literatürdeki en iyi uygulamaları takip ettiğimizi göstermektedir. Projemiz bu estetiği survival yerine saf bir TPS aksiyon prototipine uygulamaktadır.

Mekanik:

Literatür : Ravenfield (2017) (Not: FPS olmasına rağmen indie AI ve estetik açıdan benzer) veya benzeri bağımsız nişancı oyunları.

Analiz: Ravenfield gibi başarılı bağımsız nişancı oyunları, genellikle basit ama etkili çekirdek mekanikler üzerine kuruludur. Bu oyunlardaki yapay zeka genellikle DusmanYapayZeka.cs script'imizde uyguladığımız gibi basit Durum Makineleri kullanır (örn: Devriye At, Takip Et, Saldır). Oyuncu kontrolcüleri ise, CharacterController yerine Rigidbody (fizik tabanlı) kullanarak daha dinamik ve öngörülemez bir oynanış sunmayı hedefler.

Proje Karşılaştırması: Projemiz, bu bağımsız geliştirme ruhunu yansıtmaktadır.

Yapay Zeka: DusmanYapayZeka.cs script'imiz, literatürdeki bu indie oyunların kullandığı Patrolling, Chasing, Attacking durumlarına dayalı standart State Machine desenini uygulamaktadır.

Oyuncu Hareketi: PlayerController.cs script'imiz, Rigidbody kullanarak fizik tabanlı bir hareket sistemi sunar. OnSlope (eğim algılama) gibi teknikler, bu fizik tabanlı yaklaşımın zorluklarını aşmak için literatürde kabul görmüş çözümlerdir.

SONUÇ

Bu projenin tamamlanmasıyla birlikte, low-poly estetiğe sahip, oynanabilir ve teknik açıdan tutarlı bir Üçüncü Şahıs Nişancı (TPS) prototipi başarıyla hayata geçirilmiştir. Giriş bölümünde belirlenen hedeflere ulaşılmış; temel oynanış mekanikleri, yapay zeka, arayüzler ve seviye tasarımı, tek bir seviye içerisinde entegre bir şekilde çalışarak "oyun bitti" döngüsünü tamamlayan bir deneyim sunmuştur.

Projenin asıl katkısı, yalnızca nihai bir ürün ortaya koymak değil, aynı zamanda bu ürünün altında yatan yazılım mimarisinin kalitesidir. Özellikle Nesne Yönelimli Programlama (OOP) prensiplerinin oyun mekaniklerine doğrudan uygulanması, projenin en güçlü yönünü oluşturmaktadır. Silah sistemi özelinde uygulanan Soyutlama (Abstraction) ve Kalıtım (Inheritance) desenleri, kod tekrarını engelleyen, esnek ve kolayca genişletilebilir (yeni silahların eklenmesine olanak tanıyan) bir yapı kurmanın değerini kanıtlamıştır.

Bu mimari kararlar, geliştirme sürecine de doğrudan yansımıştır. Geliştirici, modern oyun geliştirme teknikleri konusunda kapsamlı bir pratik deneyim kazanmıştır. Unity'nin gelişmiş sistemlerini (fizik, animasyon, IK) birbiriyle senkronize etme, Blender gibi harici bir yazılımdan özgün 3D varlıkları bir oyun motoruna aktarma (asset pipeline) ve temel bir tekniği birden fazla (nişan alma ve mermi tespiti gibi) yaratıcı problem için uygulama becerisi edinilmiştir.

Sonuç olarak, bu proje sadece belirlenen hedefleri karşılamakla kalmamış, aynı zamanda hem yazılım mimarisi hem de edinilen pratik tecrübe açısından, gelecekte daha karmaşık ve kapsamlı projeler geliştirmek için sağlam bir temel oluşturmuştur.