YAPAY ZEKA DESTEKLİ TPS OYUNU GELİŞTİRMEDE HİBRİT MİMARİ VE MODERN GELİŞTİRME ARAÇLARI

Grup Üyelerinin İsimleri Mustafa Mehmet Aslandağ, Mustafa Kerem Çekici,Oğuzhan Erbil
Bilişim Sistemleri Mühendisliği, Teknoloji Fakültesi, Kocaeli Üniversitesi {231307067, 231307121, 231307021}@kocaeli.edu.tr

ÖZET

Bu çalışma, Kocaeli Üniversitesi Yazılım Geliştirme Laboratorı - I dersi kapsamında geliştirilen "Yapay Zeka Destekli TPS Oyunu" projesinin teknik altyapısını ve geliştirme süreçlerini sunmaktadır. Unity oyun motoru kullanılarak C# dili ile geliştirilen projede, geleneksel Nesne Yönelimli Programlama (OOP) prensipleri olan Abstract Class ve Interface kullanımının, Unity'ye özgü ScriptableObject tabanlı veri odaklı (data-driven) mimari ile hibrit bir yapıda nasıl birleştirildiği detaylandırılmıştır. Seviye tasarımı için ProBuilder ve Polybrush eklentilerinden faydalanılmış; NPC yapay zekası için ise Sonlu Durum Makinesi (FSM) ile birlikte, Unity'nin NavMesh sistemine alternatif olarak geliştirilen özel script tabanlı bir yol bulma çözümü uygulanmıştır. Son olarak, ekip çalışmasını yönetmek için kullanılan Git iş akışları tartışılmıştır.

Anahtar Kelimeler: TPS, Unity, OOP, Abstract Class, Interface, ScriptableObject, ProBuilder, NavMesh, Git, Versiyon Kontrolü

I. GİRİŞ

Bu proje, Kocaeli Üniversitesi Teknoloji Fakültesi Bilişim Sistemleri Mühendisliği 2025-2026 Güz Dönemi Yazılım Geliştirme Laboratorı - I dersi kapsamında gerçekleştirilmiştir. Projenin temel amacı, temel TPS (Third Person Shooter) mekaniklerini ve yapay zeka destekli NPC davranışlarını içeren bir oyun prototipi geliştirmektir.

Bu proje bir "öğrenme projesi" olarak ele alınmış; bu doğrultuda, bir problemi çözmek için birden fazla teknik ve mimari (hem geleneksel OOP hem de veri odaklı tasarım) araştırılmış ve uygulanmıştır. Bu rapor, bu farklı yaklaşımların projeye nasıl entegre edildiğini, karşılaşılan zorlukları ve geliştirilen özel çözümleri sunmaktadır.

II. METODOLOJİ VE KULLANILAN TEKNOLOJİLER

Proje, Unity oyun motoru üzerinde C# programlama dili kullanılarak geliştirilmiştir. Bu bölüm, projenin temelini oluşturan hibrit mimariyi, araçları ve algoritmaları kapsamaktadır.

A. OOP Mimarisi ve Veri Odaklı Tasarım (Data-Driven Design)

Projede, esnek ve genişletilebilir bir sistem kurmak için Nesne Yönelimli Programlama'nın (OOP) hem geleneksel hem de modern, veri odaklı yaklaşımları bir arada kullanılmıştır.

Temel OOP ve Bileşen Tabanlı Mimari:

Kalıtım (Inheritance) ve Kapsülleme (Encapsulation): Geleneksel OOP ilkeleri, temel MonoBehaviour sınıfından kalıtım alınarak ve kritik veriler (örn: health) private tutularak (Kapsülleme) uygulanmıştır.

Prefab Sistemi: Unity'nin Prefab sistemini, OOP'deki "sınıf" (class) ve "nesne" (instance) mantığının bir uzantısı olarak kullandık. Bir düşman karakterini (Enemy_NPC) tüm script'leri, materyalleri ve FSM mantığıyla bir kez tasarlayıp Prefab'a dönüştürdük.

Davranışsal Soyutlama (Abstract Class & Interface): "Soyutlama" (Abstraction) ve "Çok

Biçimlilik" (Polymorphism) ilkelerini, farklı nesnelerin sahip olabileceği ortak davranışları tanımlamak için kullandık. Örneğin, oyunda hasar alabilen her şeyin (oyuncu, düşman, kırılabilir sandık vb.) IDamageable (Hasar Alabilir) adında bir Interface (Arayüz) uygulamasını sağladık. Bu arayüz, TakeDamage(float amount) adında bir metodu zorunlu kıldı. Bu sayede, bir merminin script'i, çarptığı nesnenin ne olduğunu bilmek zorunda kalmadan sadece "Bu nesne IDamageable mı?" diye kontrol edip TakeDamage metodunu çağırabilmiştir.

Veri Soyutlama (ScriptableObject): Projemize esneklik katan ikinci bir soyutlama katmanı da ScriptableObject mimarisi olmuştur. Bu yaklaşımı, davranışları değil, verileri soyutlamak için kullandık. Bir "Karakter" veya "Düşman" için gereken tüm özellikleri (Sağlık puanı, hareket hızı, kullanılacak 3D model, saldırı gücü vb.) içeren bir CharacterData adında bir ScriptableObject şablonu oluşturduk. Bu sayede:

Hiç kod yazmadan, "Hızlı Düşman" (ScriptableObject asset'i, Hız: 10) ve "Yavaş Tank Düşman" (ScriptableObject asset'i, Hız: 2, Sağlık: 500) gibi onlarca farklı düşman varyantı oluşturabildik.

IDamageable arayüzü ile davranışı, ScriptableObject ile veriyi birbirinden ayırarak (Decoupling) son derece esnek ve yönetilebilir bir sistem kurduk.

B. Seviye Tasarımı Araçları: ProBuilder ve Polybrush

Proje dokümanı, "Temel bir harita ya da sahne tasarlanabilir" ifadesiyle seviye tasarımını serbest bırakmıştır. Verimli ve hızlı bir iş akışı için harici 3D modelleme yazılımları yerine Unity ekosistemi içinde kalarak şu eklentileri kullandık:

ProBuilder: Hızlı prototipleme (grey-boxing) ve temel seviye geometrisini (duvarlar, rampalar, siper alanları) oluşturmak için kullanılmıştır.

Polybrush: Oluşturulan temel geometriyi detaylandırmak, "yontmak" (sculpting) ve doku boyaması (texture painting) yaparak seviyenin görsel kalitesini ve doğallığını artırmak amacıyla öğrenildi ve uygulandı.

C. Yapay Zeka ve Özel Yol Bulma (Pathfinding) Çözümü

Projenin en kritik gereksinimlerinden biri yapay zeka destekli NPC'lerdir. Bu gereksinimi proje dokümanında belirtildiği gibi iki aşamada çözdük:

Davranış Modeli (FSM): NPC'lerin davranışlarını yönetmek için bir Sonlu Durum Makinesi (Finite State Machine - FSM) tasarladık. Bu FSM, bir düşmanın Idle (Boşta), Patrol (Devriye), Chase (Kovalama) ve Attack (Saldırı) durumları arasında geçiş yapmasını sağlayan bir script olarak kodlanmıştır.

Yol Bulma (Pathfinding) Araştırması ve Özel Çözüm:

Proje dokümanı, yol bulma için Unity'nin AI Navigator ve NavMesh sistemini önermiştir . Bu teknolojiyi öğrendik ve ilk denemelerimizde uyguladık.

Karşılaşılan Zorluk: Ancak, oyunumuzun ihtiyaç duyduğu dinamik NPC doğurma (spawning) mekanikleri sırasında, NavMesh "Bake" (pişirme) işleminin statik yapısının çeşitli sorunlara yol açtığını tespit ettik.

Getirilen Çözüm: Bu zorluğu aşmak için NavMesh sistemini kullanmaktan vazgeçtik. Bunun yerine, FSM'in Chase ve Patrol durumları için hedefe yönelmeyi ve temel engellerden kaçınmayı sağlayan özel, script tabanlı bir hareket ve yönelim mantığı geliştirdik. Bu, proje gereksinimlerini karşılarken karşılaştığımız teknik bir soruna kendi çözümümüzü üretmemizi sağlamıştır.

III. EKİP ÇALIŞMASI VE VERSİYON KONTROLÜ

Üç kişilik bir ekip olarak çalışmak, proje yönetimini ve kod entegrasyonunu kritik bir hale getirmiştir. Proje dokümanı, ekip çalışmasının GitHub üzerinden takibini ve ekip üyelerinin orantılı katkısını (commit) şart koşmuştur. Bu gereksinimi karşılamak için Git versiyon kontrol sistemini öğrendik ve uyguladık.

1. **Kullanılan Araçlar:** Hem komut satırı deneyimi kazanmak için Git Bash hem de görsel arayüz kolaylığı için GitHub Desktop araçlarını aktif olarak kullandık.
2. **Unity için Git Yapılandırması:** Unity projelerinin Git ile düzgün çalışması için standart dışı adımlar gereklidir. Öğrendiğimiz en kritik iki nokta:
    - **.gitignore:** Unity'nin otomatik oluşturduğu Library, Temp, obj gibi devasa ve kişisel bilgisayara özgü klasörleri repoya göndermemek için özel bir Unity.gitignore dosyası yapılandırdık. Bu, "merge conflict" (birleştirme çakışması) sorunlarını büyük ölçüde azaltmıştır.
    - **Git LFS (Large File Storage):** Modeller (.fbx), dokular (.png, .jpg) ve ses dosyaları (.wav) gibi büyük ikili (binary) dosyalar, Git tarafından verimli bir şekilde yönetilemez. Bu dosyaları takip etmek için Git LFS'i etkinleştirdik.
3. **Çalışma Akışı (Workflow):** Ana main branch'i (dalı) her zaman kararlı ve çalışır durumda tuttuk. Her ekip üyesi, geliştireceği yeni bir özellik (örn: "Düşman Yapay Zekası" veya "Ana Menü Tasarımı") için kendi feature-branch (özellik dalı) üzerinde çalıştı. Bu özellik tamamlandığında, main branch'e bir "Pull Request" (birleştirme isteği) açarak kodun incelenmesini ve test edilmesini sağladık.

IV. SONUÇ

Bu proje, yapay zeka destekli bir TPS oyununun teknik gereksinimlerini karşılamanın yanı sıra, modern yazılım geliştirme pratiğinin temel taşlarını öğrenmemizi sağlamıştır. Nesne Yönelimli Programlama prensiplerini Unity'nin bileşen tabanlı yapısına uyarlayarak esnek ve genişletilebilir bir kod mimarisi kurduk. Prosedürel üretim ve Perlin noise üzerine yaptığımız araştırmalar, oyunumuzu gelecekte nasıl daha dinamik hale getirebileceğimize dair vizyonumuzu genişletti. En önemlisi, Git ve GitHub'ı profesyonel bir iş akışıyla kullanarak bir ekip olarak büyük bir Unity projesini nasıl yöneteceğimizi deneyimledik.

V. KAYNAKÇA (ÖRNEKTİR)

[1] Unity. (2024). *Separate Game Data and Logic with ScriptableObjects*. [Online]. Available: https://unity.com/how-to/separate-game-data-logic-scriptable-objects.

[2] Unity Learn. (2024). *Finite State Machines*. [Online]. Available: https://learn.unity.com/project/finite-state-machines-1.

[3] "ProBuilder: Advanced Level Design in Unity," *80.lv*, Feb. 15, 2018. [Online]. Available: https://80.lv/articles/probuilder-advanced-level-design-in-unity.

[4] M. Afsar. (2019). "How to set up Git and Git-LFS in Unity," *Medium*, Jan. 14, 2019. [Online]. Available: https://masudafsar.medium.com/how-to-in-unity-setting-up-git-and-git-lfs-d15109d9f11e.

[5] B. Trotter. (2023). "Interfaces vs Scriptable Objects," *GameDev.tv Community*, May 10, 2023. [Online]. Available: https://community.gamedev.tv/t/interfaces-vs-scriptable-objects/225909.

[6] S. S. L. (2015). "Unity 3D Simple Enemy Follow AI Script Without NavMeshAgent," *YouTube*, Feb. 23, 2015. [Online]. Available: https://www.youtube.com/watch?v=drTcfhULpLA.