**T.C.**

**FIRAT ÜNİERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

# UNITY 3D OYUN MOTORU KULLANARAK TAKSİ SİMULATOR OYUNU

Mertcan Yaşar - 215260302

Tez Danışmanı:

Doç. Dr. Erkan Duman

BİTİRME TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

**ELAZIĞ**

**2025**

**T.C.**

**FIRAT ÜNİERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

# UNİTY 3D OYUN MOTORU KULLANARAK TAKSİ SİMULATOR OYUNU

Mertcan Yaşar - 215260302

BİTİRME TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Bu bitirme tezi …./…./2025 tarihinde, aşağıda belirtilen jüri tarafından oybirliği/oyçokluğu ile başarılı/başarısız olarak değerlendirilmiştir.

Doç. Dr. Erkan DUMAN Doç. Dr. Burhan ERGEN Doç. Dr. Erdal ÖZBAY

# ÖZGÜNLÜK BİLDİRİMİ

Bu çalışmada, başka kaynaklardan yapılan tüm alıntıların, ilgili kaynaklar referans gösterilerek açıkça belirtildiğini, alıntılar dışındaki bölümlerin, özellikle projenin ana konusunu oluşturan teorik çalışmaların ve yazılım/donanımın benim tarafımdan yapıldığını bildiririm.

Furat Üniversitesi 19/06/2025

Bilgisayar Mühendisliği Mertcan Yaşar

23119 Elazığ

# BENZERLİK BİLDİRİMİ

Mertcan Yaşar - 215260302 Bitirme Tezi 2025.docx

ORİJİNALLİK RAPORU

| %2 | %2 | %0 | %1 |
|---|---|---|---|
| BENZERLİK ENDEKSİ | İNTERNET KAYNAKLARI | YAYINLAR | ÖĞRENCİ ÖDEVLERİ |

BİRİNCİL KAYNAKLAR

| | | |
|---|---|---|
| 1 | Submitted to Fırat Üniversitesi<br>Öğrenci Ödevi | %1 |
| 2 | wiki2.org<br>İnternet Kaynağı | <%1 |
| 3 | bergsjostolen.reflection-network.eu<br>İnternet Kaynağı | <%1 |
| 4 | acikbilim.yok.gov.tr<br>İnternet Kaynağı | <%1 |
| 5 | tr.wikipedia.org<br>İnternet Kaynağı | <%1 |
| 6 | dspace.cvut.cz<br>İnternet Kaynağı | <%1 |
| 7 | www.lense.fr<br>İnternet Kaynağı | <%1 |
| 8 | docplayer.com.br<br>İnternet Kaynağı | <%1 |
| 9 | korkuoyunu.net<br>İnternet Kaynağı | <%1 |
| 10 | www.freeradicals2013.de<br>İnternet Kaynağı | <%1 |
| 11 | ir.lib.uth.gr<br>İnternet Kaynağı | <%1 |
| 12 | www.essays.se<br>İnternet Kaynağı | <%1 |
| 13 | www.giikorea.co.kr<br>İnternet Kaynağı | <%1 |
| 14 | "Advances on Mechanics, Design Engineering and Manufacturing V", Springer Science and Business Media LLC, 2025<br>Yayın | <%1 |

# TEŞEKKÜRLER

Bitirme projem, Fırat Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nde, Sayın Doç. Dr. Erkan Duman'ın yönlendirmesi ve gözetimi altında hazırlanmıştır. Projemin hazırlaması sürecinde bilgi, görüş ve eleştirilerinden yararlandığım hocam Sayın Erkan DUMAN'a en içten duygularımla teşekkür ederim.

# İÇİNDEKİLER

# ŞEKİLLER LİSTESİ

# KISALTMALAR LİSTESİ

**GPS** : Küresel Konumlandırma Sistemi
**C#** : C Keskin
**API** : Application Programming Interface
**PC** : Personal Computer
**PSN** : PlayStation Network
**iOS** : iPhone Operating System
**VR** : Virtual Reality
**AR** : Augmented Reality
**HDRP** : High Definition Render Pipeline
**URP** : Universal Render Pipeline
**WebGL** : Web Graphics Library
**IAP** : In App Purchasing
**ADS** : Unity Ads
**AI** : Artificial Intelligence
**HTC** : High Tech Computer Corporation
**UI** : User Interface
**CPU** : Central Processing Unit
**GPU** : Graphics Processing Unit
**HUD** : Heads-Up Display
**HTML** : HyperText Markup Language
**CSS** : Cascading Style Sheets
**SQL** : Structured Query Language

# ÖZET

Bu bitirme projesi kapsamında geliştirilen "Taksi Simülatör " adlı oyun, oyunculara gerçekçi ve etkileşimli bir taksi sürüş deneyimi sunmayı hedefleyen bir simülasyon oyunudur. Projenin amacı, oyuncunun bir şehir içinde yolcu alıp belirli rotalara zaman sınırlı bir şekilde ulaştırmasını sağlayan bir mekanik üzerine kurulu oyun sistemi geliştirmektir. Oyuncular yalnızca sürüş becerilerini değil; aynı zamanda stratejik düşünme, zaman ve kaynak yönetimi becerilerini de kullanmak zorundadır.

Geliştirme süreci, Unity oyun motoru ve C# programlama dili kullanılarak gerçekleştirilmiştir. Oyun içerisinde farklı kamera açıları, yakıt sistemi, dinamik trafik ve gece-gündüz döngüsü gibi detaylar entegre edilerek gerçekçi bir şehir simülasyonu oluşturulmuştur. Ayrıca oyuncuların araçlarını özelleştirmesi, yükseltmesi ve kazançlarını yeni araçlar almak için kullanabileceği bir ekonomik sistem geliştirilmiştir.

Projenin teknik alt yapısında 3D modelleme, ses tasarımı, performans optimizasyonları ve asset entegrasyonu gibi pek çok unsur kullanılmıştır. Tasarım sürecinde kullanıcı deneyimi ön planda tutulmuş, oyunculara sezgisel ve akıcı bir oynanış sunulması hedeflenmiştir.

Sonuç olarak, bu tez çalışması, sadece bir oyun üretme sürecini değil, aynı zamanda oyun tasarımının kuramsal temellerini, geliştirici araçları kullanmayı ve bir yazılım projesinin başından sonuna kadar nasıl yürütüleceğini kapsamlı biçimde ele almaktadır. Geliştirilen oyun, ilerleyen süreçte Steam ve Epic Games platformlarında yayınlanarak daha geniş bir kullanıcı kitlesine sunulması planlanan ticari bir ürün niteliğindedir.

# ABSTRACT

This thesis presents the development of a simulation game titled "Taxi Simulator: The Driver's Journey", aimed at providing players with an immersive and interactive taxi driving experience. The primary goal of the project is to create a game system where players pick up passengers and transport them to designated locations within a time limit, while managing fuel and navigating through a dynamic city. Players are required to use not only their driving skills but also strategic thinking, time, and resource management abilities.

The development process of the game utilized the Unity game engine and C# programming language. The game includes various features such as different camera angles, fuel management, dynamic traffic, and a day-night cycle, all integrated to create a realistic city simulation. Furthermore, players can customize and upgrade their vehicles, and an economic system was introduced that allows them to use earnings for purchasing new vehicles.

The technical framework of the project involved 3D modeling, sound design, performance optimizations, and asset integration. The design process prioritized user experience, ensuring that the gameplay is intuitive and smooth for the player.

As a result, this thesis not only covers the process of creating a game but also provides a comprehensive view of the theoretical foundations of game design, the use of development tools, and the management of a software project from start to finish. The developed game is intended to be released on platforms such as Steam and Epic Games, targeting a broader audience for commercial distribution.

Keywords: Taxi simulator, Unity, game development, C#, game design, simulation, game economy, monetization, user experience

# 1.GİRİŞ

Günümüz dijital dünyasında oyunlar, yalnızca eğlenceli vakit geçirme araçları olmaktan öteye geçerek, aynı zamanda eğitim, simülasyon ve psikolojik deneyim alanlarında da kullanılır hale gelmiştir. Özellikle simülasyon türündeki oyunlar, oyunculara farklı dünyaların kapılarını aralayarak, onlara yeni beceriler kazandırmayı ve farklı bakış açıları geliştirmelerini sağlamayı amaçlar. Taxi Simulator, oyunculara gerçekçi bir şehir simülasyonu deneyimi sunmayı hedefleyen bir taksi simülasyonu oyunudur. Bu oyun, taksi sürücüsünün perspektifinden şehri keşfetmek ve yolcu taşımak gibi sorumlulukları üstlenerek, oyunculara zorlu bir deneyim yaşatmaktadır.

Bu tez, Taxi Simulator adlı oyun projelerinin tasarım, geliştirme ve optimizasyon süreçlerine dair ayrıntılı bir inceleme sunmaktadır. Oyun, oyunculara yalnızca bir taksi sürücüsünün görevlerini yerine getirme imkânı tanımakla kalmaz, aynı zamanda trafik akışını, yolcu taşıma süreçlerini ve şehirdeki diğer dinamikleri yönetme konusunda da stratejik düşünme gerektiren bir deneyim sunmaktadır. Oyun, aynı zamanda Unity oyun motoru ve C# programlama dili kullanılarak geliştirilmiştir ve bu teknolojilerin oyun tasarımındaki rolü detaylı bir şekilde ele alınacaktır.

Taxi Simulator oyununda oyuncular, harita üzerinde rastgele belirlenen yolcuları alarak, hedef noktalara en kısa sürede ulaşmaya çalışırlar. Ancak bu süreç, sadece zamanın değil, aynı zamanda aracın yakıt seviyesinin de yönetilmesini gerektirir. Oyun, oyuncuları hem fiziksel hem de stratejik açıdan zorlayarak, zaman yönetimi, karar verme becerileri ve kaynakları verimli kullanma gibi becerilerin gelişmesine katkı sağlar.

Tez, Taxi Simulator oyununun tasarım aşamalarını, kullanılan teknolojileri, oyuncu etkileşimlerini, oyun içi ekonomik sistemleri ve görsel-işitsel tasarımları kapsamlı bir şekilde inceleyecektir. Oyun tasarımında dikkat edilen en önemli faktörlerden biri, oyuncu deneyiminin her aşamasında etkileşimi artırmaktır. Bu bağlamda, oyunun görsel unsurları, ses tasarımı, animasyonlar ve şehir içi dinamiklerin etkili bir şekilde entegre edilmesi sağlanmıştır. Ayrıca, oyuncunun oyun içindeki ilerlemesi ve başarıya ulaşması için belirli bir ekonomik model de geliştirilmiştir.

Projenin hedeflerinden biri, Taxi Simulator oyununu sadece bir eğlence aracı olarak değil, aynı zamanda ticari olarak da başarılı bir ürün haline getirmektir. Bu bağlamda, oyun

monetizasyonu, Steam ve Epic Games gibi dijital platformlarda nasıl yer alacağı ve bu platformlar üzerinden nasıl gelir elde edilebileceği konusu da derinlemesine incelenecektir.

Bu tez, oyun tasarım sürecini baştan sona ele alarak, oyun dünyasına dair kapsamlı bir bakış açısı sunmakta ve oyun sektöründeki önemli dinamikleri oyuncular, geliştiriciler ve oyun meraklıları için analiz etmektedir. Aynı zamanda, oyunun gelecekteki geliştirilme süreci ve ek özellikleri üzerine de öngörülerde bulunulacaktır.

***Şekil 1.1. Taxi Simulator Kapak Fotoğrafı***

# 2. ANA MEKANİKLER

## 2.1. Yolcu Alma Ve Taşıma

Taxi Simulator oyununda yolcu alma ve taşıma mekaniği, oyuncunun temel görevlerini oluşturur ve oyunun ilerleyişinde merkezi bir rol oynar. Bu mekanizma, oyuncunun oyun dünyasında ne kadar başarılı olacağını belirleyen başlıca faktörlerden biridir. Oyuncular, şehri keşfederken farklı noktalarda yolcuları alacak ve onları belirlenen hedeflere taşımak zorunda kalacaklardır. Bu süreç, oyun içindeki zaman yönetimi ve stratejik kararlar almak için oyuncuyu sürekli olarak zorlayacaktır.

Yolcu alma süreci, oyuncunun şehrin çeşitli bölgelerinde rastgele belirlenen yolcuları alarak, onları hedef noktalara taşımaktan sorumlu olduğu bir aşamadır. Oyuncular, harita üzerinde belirli noktalarda bulunan yolcuları almak için bu alanlara yönlendirilir. Her yolcu, hedefe gidiş için belirli bir rota ve zaman sınırına sahiptir. Bu unsurlar, oyunculara her yolculukta farklı stratejiler geliştirme imkanı sunar. Şehirdeki yolcular genellikle farklı alanlarda beklemektedir ve bu bekleme noktaları, şehirdeki önemli bölgelerde yer alır.

Oyuncular yolcuları almak için harita üzerindeki simgeleri takip ederler. Bu simgeler, yolcunun alınacağı noktayı gösteren işaretlerdir ve oyuncu, bu işaretleri takip ederek yolcuyu alacağı yere ulaşır. Her yolcunun taşıma süreci farklıdır ve bazı yolcular, daha hızlı ve kısa mesafeler talep ederken, bazıları uzun mesafeler isteyebilir. Bu çeşitlilik, oyunun stratejik dinamiklerini arttırır ve her yolculuk farklı bir deneyim sunar.

Yolcu alma işlemi sırasında oyuncuların dikkat etmeleri gereken birkaç önemli faktör vardır. Şehirdeki trafik yoğunluğu, yolcunun varış süresi üzerinde doğrudan etkili olabilir. Eğer trafik yoğun ise, oyuncuların hedefe zamanında ulaşması zorlaşabilir. Aynı zamanda, oyuncuların seçim yaparken kısa mesafeli ve uzun mesafeli yolcular arasındaki farkları göz önünde bulundurması gerekir. Uzun mesafeli yolcular genellikle daha fazla kazanç sağlasa da, bu tür yolculuklar, oyunculara daha fazla risk ve stratejik düşünme fırsatı sunar.

Yolcu taşıma süreci, yolcuyu alıp hedefe ulaşmaya kadar devam eder. Bu aşama, zaman sınırlı teslimatları ve hızlı kararlar gerektiren bir mekaniği içerir. Yolcuyu taşıyan oyuncular, hedefe en hızlı ve en verimli şekilde ulaşmaya çalışırken, şehri ve çevresel faktörleri göz önünde bulundurmalıdırlar. Oyuncular, şehirdeki trafik akışına, yol çalışmaları ve trafik ışıkları gibi

etkenlere dikkat etmek zorundadır. Bu faktörler, oyuncunun seçtiği rotayı ve taşıma sürecindeki hızını etkiler.

Trafik durumu, oyuncunun yolculuk süresi üzerinde doğrudan bir etkiye sahiptir. Eğer trafik sıkışmışsa, oyuncu zamanında varış noktasına ulaşamayabilir. Bu yüzden, oyuncuların en verimli rotayı seçmeleri ve mümkünse trafik yoğunluğundan kaçınmaları gerekir. Bu, zaman yönetimi becerilerini geliştirmeyi ve stratejik kararlar almayı zorunlu kılar.

Yolcu taşıma sürecinde oyuncular aynı zamanda aracın yakıt seviyesini de kontrol etmek zorundadırlar. Eğer araçta yeterli yakıt yoksa, yolculuk yarıda kalabilir ve oyuncu cezalarla karşılaşabilir. Bu nedenle, oyuncuların yakıt seviyesini yönetmek ve gerektiğinde yakıt almak için uygun noktalara gitmek, oyun içindeki başarıyı belirleyen kritik bir faktördür.

Yolcu taşıma süreci aynı zamanda ekonomiyi de etkiler. Oyuncular, taşıdıkları yolculardan kazandıkları parayı yakıt alımları, araç yükseltmeleri veya yeni araçlar satın almak için harcayabilirler. Ekonomik yönetim, oyuncunun oyun boyunca ilerlemesi ve başarı elde etmesi için büyük önem taşır. Uzun mesafeli ve riskli yolculuklar daha fazla kazanç sağlasa da, doğru kararlar almak, oyuncunun başarısını belirleyen unsurlardır.

*Şekil 2.1. Yolcu Alma Ve Gps*

## 2.2. Zamana Dayalı Teslimat

Taxi Simulator oyununda zaman, oyuncular için kritik bir faktördür. Oyuncular her yolcuyu aldıklarında, belirli bir süre içinde hedefe ulaşmak zorundadırlar. Zamanla yarışmak,

*Şekil 2.2. Oyun Ekran Görüntüsü*

oyunculara sürekli olarak baskı uygular ve onları daha hızlı düşünmeye zorlar. Bu bağlamda zamana dayalı teslimatlar, oyuncunun en hızlı ve en verimli şekilde hedefe ulaşmasını gerektiren bir mekanizmadır. Oyuncular yalnızca trafikteki engelleri aşmakla kalmaz, aynı zamanda verilen süreyi en iyi şekilde yönetmek zorundadırlar.

Her yolcunun varış süresi belirli bir zaman dilimine tabidir ve bu süre, oyuncunun yolculuk süresi boyunca dikkat etmesi gereken önemli bir faktördür. Yolcular, şehirdeki farklı noktalardan alınarak belirli bir hedefe yönlendirilir. Yolcu taşıma sürecinde, oyuncular bu süreyi aşmadan hedefe ulaşmayı başarmalıdır. Eğer oyuncu belirlenen süre içinde varış noktasına ulaşamazsa, yolcudan alınan ücret azalır ve oyuncu ceza alır. Bu, oyun içinde oyuncunun ekonomik başarısını doğrudan etkiler.

Oyun içindeki GPS yönlendirmeleri, oyunculara en hızlı rotayı seçme konusunda yardımcı olan bir araçtır. Oyuncu, GPS rehberliği ile belirlenen rotayı takip ederek yolculuğunu en hızlı şekilde gerçekleştirebilir. Ancak, GPS sadece bir yol gösterici işlevi görür; oyuncular rotayı değiştirebilir, alternatif yollar seçebilirler. Bu, oyuncuya hem esneklik hem de stratejik kararlar alma imkânı tanır. GPS yönlendirmeleri, oyuncunun harita üzerinde doğru bir şekilde ilerlemesine yardımcı olurken, oyuncular alternatif yolları keşfetmeye teşvik eder.

Farklı yoldan gitme seçeneği, oyunculara özgürlük tanır. Her yolculuk, oyuncuya sadece GPS tarafından önerilen yolu değil, aynı zamanda alternatif yolları da keşfetme fırsatı sunar. Bu alternatif yollar, bazen daha kısa mesafeler, daha az trafik veya daha az engel içerebilir. Ancak, oyuncular bu alternatif yolları seçerken, yeni yolların güvenliği ve ne kadar hızlı bir şekilde varış noktasına ulaşacaklarını da göz önünde bulundurmalıdır. Farklı yoldan gitmek, oyunculara zaman kazancı sağlayabilir ancak bu, her zaman garantili bir sonuç vermez. Bazı alternatif yollar daha

fazla risk taşıyabilir ya da daha uzun süre alabilir. Bu nedenle, oyuncular yolculuk sırasında sürekli olarak kararlar almalı ve stratejik düşünmelidirler.

Normal yolcu teslimatı, her oyuncunun karşılaştığı en temel görevlerden biridir. Bu tür teslimatlarda, oyuncular yolcuları belirli bir noktadan alır ve onları şehirdeki farklı bir hedefe taşır. Bu teslimatlar genellikle zaman baskısı olmadan yapılır, ancak bazı teslimatlar yine de belirli bir süre içinde tamamlanmalıdır. Yolcu teslimatları, oyuncuların oyun içindeki kazançlarını artırırken, aynı zamanda oyunculara yeni araçlar alabilme ya da araçlarını yükseltebilme fırsatı sunar.

Yolcu taşıma süreci, her yolcunun taşıma süresi boyunca verilen süreyi aşmamak için dikkatli bir strateji gerektirir. Bu, hem oyuncunun ekonomik yönetimi hem de zaman yönetimi becerilerini geliştirmesini sağlar. Oyuncular, yolda karşılaştıkları her türlü engelle mücadele ederken, aynı zamanda araçlarının yakıt seviyesini de yönetmelidirler. Yakıt, oyuncunun oyun sürecinde karşılaştığı diğer bir önemli faktördür. Eğer oyuncu yeterince yakıt almadan yolculuğa çıkarsa, hedefe ulaşamadan yolculukları kesilebilir ve bu da oyunun ilerlemesinin engellenmesine yol açar.

Her yolculuk, farklı stratejik seçimler ve kararlar gerektirir. Oyuncular, belirli hedeflere en kısa sürede ulaşmak için doğru rotayı seçmelidirler. Aynı zamanda, karşılaştıkları trafik durumlarını, yol çalışmalarını ve diğer engelleri göz önünde bulundurarak, yolculuk sürecini planlamalıdırlar. Zaman, trafik ve rota seçenekleri arasındaki dengeyi kurmak, Taxi Simulator oyunundaki başarının anahtarlarından biridir.

### 2.3. Yakıt Mekaniği

Taxi Simulator oyununda, oyuncular yalnızca zamanla değil, aynı zamanda araçlarının yakıt durumu ile de mücadele ederler. Bu mekanizma, oyunculara sadece sürüş becerilerini değil, aynı zamanda kaynaklarını nasıl yönetmeleri gerektiğini de öğretir. Oyun içindeki yakıt durumu, oyuncunun ilerlemesini etkileyen önemli bir faktördür ve zamanında yakıt alımı, oyunun başarılı bir şekilde tamamlanabilmesi için kritik öneme sahiptir.

Yakıt durumu, oyuncunun araçlarının yakıt seviyesini ifade eder. Oyuncular, yolculuk boyunca araçlarının yakıt seviyesini kontrol etmek zorundadırlar. Eğer yakıt seviyesi tükenirse, yolculuk kesilir ve oyuncu hedefe ulaşamadan görevini tamamlayamaz. Bu, oyuncunun oyun içindeki başarı oranını doğrudan etkileyen önemli bir unsurdur. Yakıt seviyesi, ekranın köşesinde

sürekli olarak gösterilir, bu da oyuncunun ne kadar yakıtı kaldığını her an takip edebilmesini sağlar.

Yakıt seviyesi, her yolculukta değişir ve yolculuğun uzunluğuna göre azalır. Uzun mesafeli yolculuklar, daha fazla yakıt tüketir, kısa mesafeli yolculuklar ise daha az yakıt gerektirir. Bu durum, oyuncuları hem stratejik düşünmeye hem de kaynaklarını verimli kullanmaya zorlar. Yolculuk sırasında yakıt seviyesinin sürekli olarak azalması, oyunculara oyun dünyasında nasıl kaynak yönetimi yapmaları gerektiğini öğretir.

Oyuncular, yakıt seviyelerini sürekli olarak takip etmek zorundadırlar. Bu takip, sadece zamanla değil, aynı zamanda aracın performansını da etkileyen bir unsurdur. Yakıt seviyesi azaldıkça, oyuncular aracın hızını ve sürüşünü etkileyen zorluklarla karşılaşabilirler. Eğer oyuncu, yeterli yakıtı yoksa, yolculuk esnasında hız kaybı yaşanabilir veya araç yolda kalabilir. Bu durumda, oyuncuların en yakın benzin istasyonuna ulaşarak yakıt almaları gerekir.

Benzin istasyonları, harita üzerinde belirli noktalarda yer alır ve oyuncular bu istasyonları kullanarak araçlarının yakıt seviyesini yenileyebilirler. Ancak, yakıt alımının da bazı stratejik yönleri vardır. Yakıt almak, zaman kaybına neden olabilir ve bu da oyuncunun teslimat süresini etkiler. Bu nedenle, oyuncuların zaman yönetimini de göz önünde bulundurarak yakıt almaları gerekir. Yakıt almak, yalnızca gerektiğinde yapılmalı ve oyuncunun ilerlemesi için zaruri hale gelmelidir.

Yakıt alımı, aynı zamanda ekonomik bir karar sürecidir. Oyuncular, aldıkları yakıtı, oyun içindeki kazançlarıyla dengeleme yoluna giderler. Eğer oyuncu, yolda gereksiz yere çok fazla yakıt alırsa, bu, ekonomik olarak zarar etmelerine neden olabilir. Bu nedenle, oyuncuların yakıt alımını dengeli bir şekilde yapmaları gerekir. Ayrıca, bazı durumlarda, yolculuk sırasında yakıt ikmali yapmak zorunda kalabilirler ve bu durumda benzin istasyonlarına ulaşmak, zaman açısından önemli bir risk faktörüdür.

Oyun içindeki yakıt yönetimi, oyuncuların başarısını etkileyen önemli bir stratejik unsurdur. Oyuncular, yakıtı verimli bir şekilde kullanarak, hem zaman kazanabilirler hem de daha fazla kar elde edebilirler. Uzun mesafeli yolculuklar sırasında, yakıt tasarrufu yapmak, oyuncuların daha uzun süre boyunca oyun içinde kalmasını sağlar. Bu nedenle, oyuncuların yakıt yönetimi konusunda dikkatli olmaları gerekmektedir.

Bazı oyuncular, düşük yakıt seviyeleriyle yola çıkmayı tercih edebilirler, ancak bu strateji risklidir. Diğer oyuncular ise yolculuğa çıkmadan önce her zaman yeterli yakıt almayı tercih edebilirler. Bu stratejik kararlar, her oyuncunun oyun tarzına ve oyun içindeki hedeflerine bağlı olarak değişir. Bu çeşitlilik, oyunun dinamik yapısını güçlendirir ve her oyuncunun oyun içinde farklı bir deneyim yaşamasına olanak tanır.


*Şekil 2.3. Taxi Simulatorde Yakıt Alınan Benzin İstasyonu*

**2.4. Kamera Açıları**

Taxi Simulator oyununda, kamera açıları oyuncunun oyun içindeki deneyimini doğrudan etkileyen önemli bir unsurdur. Oyuncular, şehri keşfederken farklı kamera açılarıyla etkileşimde bulunarak, hem sürüş deneyimini hem de oyunun görsel sunumunu yönetirler. Kamera açıları, oyuncuların çevreyi daha iyi gözlemlemelerine, sürüş sırasında kararlarını daha hızlı ve doğru bir şekilde almalarına yardımcı olur. Aynı zamanda, oyun içindeki atmosferin oluşturulmasında da önemli bir rol oynar.

Oyun, oyunculara birden fazla kamera açısı seçeneği sunarak farklı deneyimler yaşamalarına olanak tanır. Başlangıçta, oyun genellikle dış kamera açısından başlar, bu açı oyuncunun aracı ve çevresindeki şehri geniş bir perspektiften görmesini sağlar. Dış kamerada oyuncu, aracı ve etrafındaki tüm öğeleri rahatça görebilir. Bu açı, oyuncunun çevreyi daha iyi gözlemlemesine ve özellikle şehirdeki trafik akışını takip etmesine olanak tanır. Ayrıca, dış

kamera açısı oyuncuya geniş bir alanın görünmesini sağlar, bu da özellikle şehirdeki engelleri ve diğer araçları fark etmelerini kolaylaştırır.

Bir diğer yaygın kullanılan kamera açısı, iç kamera açısıdır. Bu açı, oyuncunun aracın içinde birinci tekil bakış açısıyla hareket etmesini sağlar. İç kamera, oyuncuya aracın kontrol panelini ve direksiyon simidini görme imkânı sunar. Bu açı, oyuncuya daha derin bir immersiyon deneyimi yaşatır ve gerçek bir taksi sürücüsü gibi hissetmelerini sağlar. İç kamera açısı, özellikle sürüş simülasyonları için önemli bir özelliktir çünkü oyuncuya gerçekçi bir sürüş deneyimi sunar. Bu kamera açısından, oyuncu yalnızca dış dünyayı görmekle kalmaz, aynı zamanda aracın içindeki tüm detayları da gözlemleyebilir.

Kamera açıları, aynı zamanda oyunculara oyun sırasında çeşitli stratejik avantajlar sağlar. Serbest kamera açısı, oyuncuların istedikleri zaman araçlarının etrafını dönebilmesine olanak tanır. Bu açı, oyunculara daha fazla özgürlük ve esneklik sunar. Araçlarındaki hareketleri izlerken, çevreyi farklı açılardan görmek, oyunculara daha fazla kontrol sağlar ve oyun dünyasıyla etkileşimlerini güçlendirir. Ayrıca, yakın kamera açısı da bazı durumlar için kullanılır, bu açı daha kısa mesafedeki yolculuklarda, özellikle dar yollarda ve yoğun trafiğin olduğu bölgelerde oyuncuların daha fazla detay görmesine yardımcı olur.

Kamera açıları, aynı zamanda oyunun atmosferini etkileyen bir başka unsurdur. Örneğin, gece sürüşü sırasında dış kamera açısı daha dramatik bir hava yaratmak için kullanılabilir. Şehir ışıkları ve gece atmosferi, oyuncuya gerçek bir taksi sürücüsünün hissettiği ortamı yansıtmak için özel olarak tasarlanmıştır. Aynı şekilde, gündüz sürüşü için farklı kamera açıları kullanılarak oyuncuların görsel deneyimi zenginleştirilebilir.

Farklı kamera açıları, oyunculara oyun dünyasında nasıl hareket etmeleri gerektiğini de gösterir. Dinamik kamera açısı, özellikle trafik yoğunluğunun arttığı durumlarda devreye girer. Trafikte sıkışan oyuncular, dinamik kamera açısı sayesinde çevrelerindeki araçları daha iyi görebilir ve yönlendirme yaparken daha az hata yapabilirler. Bu açı, oyuncuların çevrelerine daha dikkatli bakmalarını sağlar ve onları doğru rotalar üzerinde yönlendirir.

Sonuç olarak, kamera açıları, oyunculara farklı deneyimler sunan ve onları sürekli olarak yönlendiren bir oyun tasarımı aracıdır. Her kamera açısı, oyuncunun oyun içindeki etkileşimini değiştiren ve oyunun görsel bütünlüğünü sağlayan önemli bir faktördür. Oyuncular, farklı kamera açıları sayesinde oyun dünyasında daha derinlemesine bir deneyim yaşar, oyun içindeki dinamiklere uyum sağlamak için bu açıları kullanarak stratejilerini oluştururlar.

*Şekil 2.4. Birinci Şahıs Bakış Açısı*

# 3. OYNANIŞ ÖZELLİKLERİ

## 3.1. Araç Yükseltmeleri

Taxi Simulator oyununda araç yükseltmeleri, oyuncuların oyun sürecindeki başarısını artırmak ve deneyimlerini daha eğlenceli hale getirmek için önemli bir özelliktir. Oyuncular, elde ettikleri kazançlarla araçlarını çeşitli açılardan geliştirebilirler. Bu yükseltmeler, araçların performansını iyileştirir ve oyunculara daha zorlu yolculukları başarıyla tamamlama imkanı sunar. Araç yükseltmeleri, oyunun ilerleyen seviyelerinde kritik bir rol oynar, çünkü daha güçlü araçlarla daha uzun ve daha karmaşık yolculuklar yapılabilir.

Araç yükseltmeleri, oyunculara farklı stratejik kararlar alma fırsatı tanır. Oyuncular, hangi yükseltmenin kendileri için daha uygun olduğuna karar verirken, hem ekonomik durumlarını hem de oyun içindeki hedeflerini göz önünde bulundurmalıdırlar. Yükseltmeler, oyunculara yalnızca daha hızlı araçlar sunmakla kalmaz, aynı zamanda daha fazla güvenlik, yakıt verimliliği, daha hızlı hızlanma gibi çeşitli avantajlar da sağlar. Bu, oyunun zorluk seviyesinin arttığı ilerleyen bölümlerde oyunculara büyük bir avantaj sağlar.

İlk aşamalarda, araç yükseltmeleri genellikle hız, yakıt kapasitesi ve direksiyon hassasiyeti gibi temel unsurları iyileştirir. Oyuncular, bu temel yükseltmelerle araçlarını daha verimli hale getirirken, daha uzun yolculuklar yapabilecek ve daha fazla yolcu taşıyabilecek duruma gelirler. Ancak, oyun ilerledikçe araçlar daha fazla gelişim gerektirir. Bu noktada, oyuncular yeni araçlar satın alabilir veya mevcut araçlarını daha güçlü motorlar, yüksek performanslı fren sistemleri, daha geniş yakıt tankları gibi özel parçalarla donatabilirler.

Araç yükseltmeleri, aynı zamanda oyunun ekonomik sistemini de etkiler. Yükseltme almak için oyuncular, yolculuklardan kazandıkları parayı kullanır. Bu, oyuncuların oyunun ekonomik dengesini düşünerek hangi yükseltmeyi yapacaklarına karar vermelerini gerektirir. Yükseltmelerin maliyetleri, araçların türüne ve seviyesine göre değişir. Bu da oyuncuya, hangi araç yükseltmesini alacağı konusunda stratejik bir karar verme imkanı sunar.

Bunun dışında, yolculukları daha hızlı tamamlama için motor gücü, trafikte daha rahat manevra yapabilme için direksiyon hassasiyetini artırma veya yakıt tasarrufu sağlamak için yakıt verimliliğini yükseltme gibi farklı özellikler oyuncuların tercihine göre geliştirilebilir. Bu

yükseltmelerin her biri, oyuncuların oyun içindeki hedeflerine ulaşmalarına ve zorlukları aşmalarına yardımcı olacak şekilde tasarlanmıştır.

Araç yükseltmeleri aynı zamanda, oyuncuların oyun tarzlarını ve tercihlerini yansıtır. Örneğin, bazı oyuncular hız odaklı yükseltmeler tercih ederken, bazı oyuncular güvenliği ve yolculuk sırasında karşılaştıkları engelleri aşabilmeyi ön planda tutabilirler. Bu çeşitlilik, her oyuncunun oyunu farklı bir şekilde deneyimlemesine olanak tanır ve oyunun tekrar oynanabilirliğini artırır.

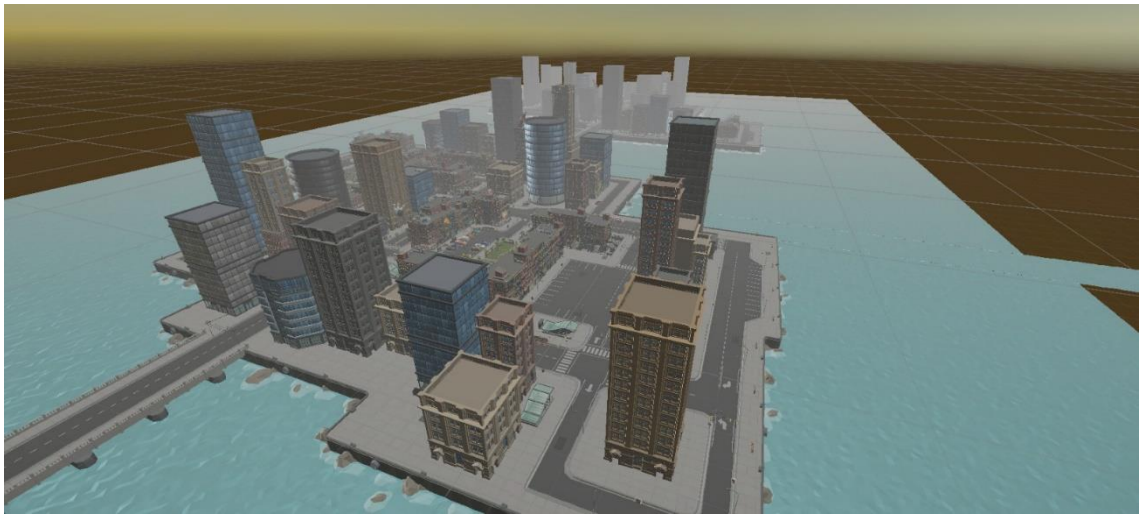
Araç yükseltmeleri, yalnızca oyunun görsel yönlerini de etkiler. Yükseltilmiş araçlar daha estetik görünüme sahip olabilir, yeni renkler, grafikler ve özel tasarımlar eklenmiş araçlar, oyunculara daha özelleştirilmiş bir deneyim sunar. Bu görsel yükseltmeler, oyunun daha fazla kişiselleştirilmesini sağlar ve oyuncuların oyun içindeki araçları daha çok sahiplenmelerini sağlar.

Sonuç olarak, araç yükseltmeleri, Taxi Simulator oyununda oyuncuların stratejik düşünme, zaman yönetimi ve ekonomik kararlar alma becerilerini geliştirmelerine yardımcı olur. Her oyuncunun tercihlerine ve stratejilerine göre farklı yükseltmeler yapabilmesi, oyunun dinamik yapısını güçlendirir ve her oyuncuya özel bir deneyim sunar. Yükseltme mekanizması, oyuncuların oyun dünyasında daha fazla ilerlemelerini sağlarken, zorluk seviyelerinin de artmasına neden olur, bu da oyun deneyiminin daha heyecan verici ve zorlu hale gelmesini sağlar.

### 3.2. Dinamik Şehir Ortamı

Taxi Simulator oyununda şehir, yalnızca statik bir arka plan değil, oyuncunun eylemlerine ve çevresel faktörlere tepki veren dinamik bir yapıdır. Şehirdeki trafik yoğunluğu, günün saati, hava durumu ve diğer etkenlere göre değişir. Bu, oyuncuların her seferinde farklı bir deneyim yaşamasını sağlar. Dinamik şehir ortamı, oyunculara sürekli olarak yeni stratejik kararlar almayı gerektirir ve bu da oyun deneyimini daha heyecanlı ve sürükleyici hale getirir.

Gün boyunca şehre giren araç sayısı, sabah, öğle ve akşam saatlerine göre değişiklik gösterir. Sabah saatlerinde, özellikle işe gitmek için yola çıkan araçlar, şehirde yoğunluğun artmasına neden olur. Öğle saatlerine doğru trafik, iş yerlerinden uzaklaşan araçlarla biraz daha hafifler, ancak akşam saatlerinde, iş çıkışının etkisiyle tekrar yoğunlaşır. Bu yoğunluk, oyuncuların yolculuklarını yaparken karşılaştıkları zorlukları arttırır.



*Şekil 3.1. Dinamik Şehir Oyun İçi Ekran Görüntüsü*

Bu trafik değişimleri, oyuncuların kararlarını doğrudan etkiler. Oyuncular, sabah saatlerinde iş çıkışı yoğunluğundan kaçınmak için yollarını değiştirebilir veya daha hızlı bir rota seçebilirler. Aynı şekilde, akşam saatlerinde de benzer şekilde, trafik tıkanıklığını önlemek için alternatif yollar tercih edilebilir. Trafik yoğunluğu, şehirdeki farklı bölgelerde farklı yoğunluklar gösterir. Örneğin, şehir merkezinde sabah saatlerinde trafik çok daha yoğunken, banliyölerdeki yollar daha boş olabilir. Oyuncular bu yoğunluğu göz önünde bulundurarak yolculuklarını planlamak zorundadırlar.

Oyunda gece-gündüz döngüsü, oyunculara farklı bir oyun deneyimi sunar. Günün her saatinde şehirdeki atmosfer değişir. Unity oyun motorunun sunduğu dinamik ışıklandırma özellikleri kullanılarak, güneş gerçekçi bir şekilde doğar ve batar. Güneşin doğmasıyla birlikte şehir, aydınlanmaya başlar; bu, oyunculara gündüz sürüşü için daha net bir görüş sunar. Gündüz saatlerinde, şehir daha canlıdır; insanlar yürür, araçlar hareket eder ve etraf daha hareketlidir. Bu, oyuncuların şehri keşfetme ve yolculuklarını yapma sürecini etkiler.

Gece olduğunda ise ışıklar açılır, yollar daha karanlık hale gelir ve oyuncuların görüş mesafesi kısıtlanır. Bu, gece sürüşünü daha zorlu hale getirir. Trafik yoğunluğu gece saatlerinde daha düşük olabilir, ancak oyuncuların daha dikkatli olmaları gerekir çünkü gece görüşü zayıflar. Ayrıca, gece sürüşlerinde daha az ışık kaynağı olduğundan, oyuncular farları kullanarak yollarını görmelidirler. Gece-gündüz döngüsü, oyuncuların oyun içindeki çevresel etmenlere uyum sağlamalarını gerektirir. Bu döngü, hem görsel atmosferi zenginleştirir hem de oyuncuların şehirdeki yönlendirmelerini ve stratejilerini değiştirir.

Unity oyun motoru, gerçekçi ışıklandırma ve gölgeleme özellikleri sunar. Bu özellikler, Taxi Simulator oyununda şehirdeki gece-gündüz döngüsünün doğru bir şekilde yansıtılmasını

sağlar. Güneşin doğmasıyla birlikte şehirdeki tüm yüzeyler aydınlanır ve yolculuk yapmak daha kolay hale gelir. Aynı şekilde, güneş batarken ışık azalır ve uzun gölgeler oluşur. Unity'nin ışık kaynakları ve Directional Light (yönlendirilmiş ışık) gibi özellikleri, güneşin hareketini simüle etmek için kullanılır. Bu ışık kaynağı, güneşin gün içindeki hareketini takip eder ve oyunculara her an farklı bir atmosfer sunar. Gece olduğunda ise, şehirdeki ışıklar devreye girer. Şehirdeki sokak lambaları, araç farları ve bina ışıkları gibi unsurlar, gece sürüşünü aydınlatır ve oyunculara yönlerini kaybetmemeleri için yardımcı olur.

Gece-gündüz döngüsü, ayrıca hava koşullarıyla da etkileşime girer. Örneğin, gece yağmur yağarken, şehirdeki ışıklar su yüzeyinde yansır ve bu da görsel açıdan etkileyici bir atmosfer yaratır. Aynı şekilde, gündüz saatlerinde güneşin sıcak ışığı, şehirdeki görsel tonları değiştirir ve oyunculara daha canlı bir şehir manzarası sunar. Bu etkileşimler, oyun dünyasında her zaman taze bir deneyim yaşatır.

Şehirdeki dinamik yapılar oyuncuların şehirdeki yolculuklarını etkileyen diğer önemli unsurlardır. Şehirdeki farklı bölgeler, yol koşulları ve engeller günün saatine ve çevresel faktörlere göre değişebilir. Trafik yoğunluğu, yolların durumu, hava koşulları ve şehirdeki diğer etmenler, oyuncunun stratejilerini şekillendirir. Trafik ışıkları, yol çalışmalarına rastlanması ve diğer araçlarla etkileşim gibi faktörler de dinamik şehri etkileyen unsurlar arasında yer alır.

Ayrıca, gece saatlerinde şehrin sessizliği ve daha az araç yoğunluğu, oyunculara sakin bir sürüş deneyimi sunar. Ancak, gece sürüşlerinde düşük görüş mesafesi ve karanlık yollar, oyuncuyu daha dikkatli ve temkinli olmaya zorlar. Gece sürüşünde, araçların farları önemli bir rol oynar ve oyuncuların yollarını güvenli bir şekilde bulmalarını sağlar.

### 3.3. Ekonomi Sistemi

Taxi Simulator oyununda ekonomi sistemi, oyuncuların oyun içindeki ilerlemelerini ve araçlarını geliştirmelerini sağlayan temel unsurlardan biridir. Oyuncular, yolculuklar sırasında elde ettikleri kazançlarla araçlarını yükseltir, yakıt alır ve şehri keşfederken karşılaştıkları çeşitli zorluklarla başa çıkmak için stratejik kararlar alırlar. Ekonomi sistemi, oyunun zorluk seviyesini ve oyun içindeki etkileşimleri doğrudan etkiler, bu da oyunculara sürekli olarak ekonomik düşünmeyi ve karar verme yeteneklerini geliştirmeyi teşvik eder.

Oyuncular, her yolculukta taşıdıkları yolculardan kazanç elde ederler. Yolculukların uzunluğu, zorluğu ve taşıma süreleri, kazançların miktarını belirler. Kısa mesafeli yolculuklar, genellikle daha az kazanç sağlar, ancak hızlı tamamlandıkları için daha fazla yolcu taşınabilir.

Uzun mesafeli yolculuklar ise daha yüksek kazançlar sunar, ancak bu yolculuklar daha fazla risk içerir, çünkü yolculuk süresi uzar ve trafik gibi engellerle karşılaşma ihtimali artar. Kazançlar, oyuncuların şehri keşfederken elde ettiği para ile sınırlıdır. Yolculuk sırasında dikkatli olunmazsa, oyuncular yakıtlarını ve diğer kaynaklarını verimli kullanamayabilir, bu da gelir kaybına yol açabilir. Ekonomik sistemde, yakıt alımı, araç bakımı ve yükseltmeler gibi giderler de önemli rol oynar. Oyuncular, kazandıkları parayı bu giderler için harcayabilirler.

Her yolcu, oyuncuya belirli bir ücret kazandırır. Ancak, kazancın miktarı, yolculuğun uzunluğuna ve zorluk seviyesine bağlıdır. Oyuncular, hızlı bir şekilde ulaşacakları kısa mesafeli yolcuları taşıyarak küçük kazançlar elde edebilirken, uzun mesafeli yolcuları taşıyarak daha büyük kazançlar sağlayabilirler. Ancak uzun yolculuklar, zaman ve trafik gibi faktörler nedeniyle daha fazla risk içerir. Bu nedenle, oyuncuların her yolculukta ne kadar kazanacaklarına karar verirken, yolculuğun ne kadar süre alacağı ve yolculuğun sonunda elde edecekleri kazanç hakkında düşünmeleri gerekmektedir.

Oyun içindeki ekonomik sistemin bir diğer önemli yönü, araç yükseltmeleridir. Oyuncular, kazandıkları parayı, araçlarını iyileştirmek ve yeni araçlar satın almak için kullanabilirler. Araç yükseltmeleri, oyunculara daha güçlü araçlar, daha fazla hız, daha fazla yakıt kapasitesi ve diğer avantajlar sağlar. Ancak araç yükseltmeleri, oyuncunun ekonomik kaynaklarını tüketen maliyetli işlemlerdir. Bu nedenle, oyuncular hangi yükseltmeleri yapacaklarına karar verirken dikkatli olmalıdırlar. Bazı oyuncular, hızlanmayı ön planda tutarak motor güçlerini yükseltirken, diğerleri güvenliği arttırmak için fren sistemini iyileştirebilirler. Bu seçimler, oyunun ilerleyişinde oyuncuya farklı stratejik avantajlar sunar.

Oyun ekonomisinin önemli bir parçası da yakıt alımıdır. Oyuncular, yolculukları tamamlayabilmek için araçlarına yakıt almak zorundadırlar. Ancak, yakıt almak da bir maliyet içerir ve bu, oyuncunun kazancını etkiler. Oyuncular, yalnızca gerektiğinde yakıt alarak kazançlarını verimli bir şekilde kullanmak zorundadırlar. Eğer oyuncular, her yolculuk öncesinde fazla yakıt alırlarsa, bu durum oyun ekonomisi üzerinde olumsuz etkiler yaratabilir. Bu nedenle, oyuncuların yakıt alımlarını dengeli bir şekilde yapmaları gerekmektedir. Ayrıca, araç bakım ve onarımları da oyuncuların harcamalarına dahil olan diğer önemli giderlerdir. Araçlar, zamanla aşınabilir ve çeşitli arızalar meydana gelebilir. Bu durumda, oyuncular aracını tamir ettirmek zorunda kalacaklardır. Tamir ve bakım masrafları, oyuncuların kazançlarıyla doğru orantılı olmalıdır. Oyuncular, kazandıkları parayı doğru bir şekilde yöneterek hem aracın bakımını yapabilir hem de yeni araç yükseltmeleri için bütçe oluşturabilirler.

Taxi Simulator'daki ekonomik sistem, oyuncuların sadece araçlarını yükseltmelerini değil, aynı zamanda paralarını nasıl yöneteceklerini de düşünmelerini gerektirir. Oyuncular, her yolculuktan elde ettikleri gelirle bir sonraki adımlarını belirlerler. Hangi araçları alacaklarına

*Şekil 3.2. Yaya Oyun İçi Ekran Görüntüsü*

karar verirken, hangi yükseltmeleri yapacaklarına karar verirken ve yakıt alırken her zaman ekonomik dengeyi gözetmeleri gerekmektedir. Oyuncular, düşük bütçeyle başlarken düşük performanslı araçlarla yolculuk yaparak kazanacakları parayı arttırarak daha yüksek seviyelerdeki araçları satın alabilirler. Ancak, araç satın almak ya da yükseltme yapmak, oyunculara daha fazla güç ve esneklik sağlasa da, buna bağlı giderler de artacaktır. Bu nedenle, her karar, oyuncuların uzun vadeli ekonomik stratejilerine göre şekillendirilmelidir.

Oyun tasarımında, oyuncuların ekonomiyi yönetebilmeleri için çeşitli araçlar sağlanır. Oyuncular, her yolculuk sonrasında kazandıkları parayı araçlarına harcayarak, daha iyi performans alacak araçlar kullanabilirler. Bu, oyunun ekonomik dinamiklerini sürekli olarak etkileyen ve oyuncuyu her aşamada düşünmeye zorlayan bir özelliktir. Ayrıca, daha fazla kazanmak isteyen oyuncular, daha zorlu yolculukları tercih ederek yüksek kazançlar elde edebilirler. Ancak, her yolculukta ekonomik kararlar almak, zamanla oyuncuları daha dikkatli ve stratejik düşünmeye yönlendirir.

Oyun içindeki ekonomi sistemi, Taxi Simulator'ın temel yapısını oluşturan ve oyuncuların stratejik kararlar almasını zorunlu kılan önemli bir özelliktir. Oyuncular, sadece hız ve yolculuk değil, aynı zamanda finansal kararlar alarak oyun içindeki başarılarını sürdürmelidirler. Ekonomik sistem, oyunun hem zorluk seviyesini artıran hem de oyuncuya daha fazla özgürlük tanıyan dinamiklerinden biridir. Bu sistem, oyunculara sadece kazanç elde etmeyi

değil, aynı zamanda bu kazancı verimli bir şekilde nasıl kullanacaklarını öğretir. Oyuncular, ekonomik dengesini doğru kurarak oyunu en verimli şekilde oynayabilirler.

### 3.4. Liderlik Tablosu

Taxi Simulator oyununda, çevrim içi liderlik tablosu, oyuncuların dünya çapında diğer oyuncularla rekabet etmelerini sağlayan önemli bir özellik sunar. Bu sistem, oyun deneyimini daha sosyal ve rekabetçi hale getirir, oyuncuların kendi başarılarını ve kazançlarını başkalarıyla kıyaslamalarına olanak tanır. Unity Gaming Service üzerinden entegre edilen Leaderboard (Liderlik Tablosu) bu rekabeti yönlendiren ve oyun içi başarıları takip eden önemli bir bileşendir. Oyuncular, her yolculuktan, tamamladıkları görevlerden ve elde ettikleri kazançlardan elde ettikleri puanlarla sıralamalara katılırlar.

Unity Gaming Services, oyun içindeki verilerin yönetilmesini sağlayan bir araçtır ve Leaderboard entegrasyonu ile oyuncuların sıralamalarını dinamik bir şekilde düzenlemeyi mümkün kılar. Bu özellik, oyuncuların sadece kendi kazançlarını değil, dünya çapındaki en yüksek başarıları da gözlemlemelerine olanak tanır. Leaderboard API'si sayesinde sıralamalar, oyuncuların puanlarına, başarılarına ve performanslarına göre anlık olarak güncellenir. Oyun içindeki her yolculuk, görev tamamlama süresi veya araç yükseltmeleri gibi başarılar, oyuncuların sıralamalarda nasıl bir yerde olduklarını belirler. Bu özellik, her oyuncuya bir hedef sunar ve onları daha yüksek sıralarda yer alabilmek için motive eder.

Oyuncular, oyun içinde kazandıkları puanları ve paraları kullanarak araçlarını yükseltir, yeni araçlar satın alır ve çeşitli oyun içi başarılar elde ederler. Liderlik tablosundaki sıralama, oyuncuların kazançları ve başarıları ile doğrudan bağlantılıdır. Yüksek sıralarda yer almak, oyuncuların sadece rekabetçi bir hedefi değil, aynı zamanda daha fazla ödül ve avantaj elde etmelerini de sağlar. Leaderboard sayesinde oyuncular, hangi başarıların daha değerli olduğunu ve nasıl daha yüksek puanlar kazanabileceklerini gözlemleyebilirler.

Liderlik tablosu, sadece oyuncuların sıralamalarda ne kadar yüksek olduklarını görmekle kalmaz, aynı zamanda onları daha fazla kazanmak, zorlu görevleri tamamlamak ve diğer oyunculardan daha fazla kazanç elde etmek için teşvik eder. Bu ekonomik model, oyun içindeki başarının sadece beceriye değil, aynı zamanda stratejik kararlar almaya da bağlı olduğunu gösterir. Oyuncular, kazandıkları puanları nasıl harcayacaklarına ve hangi yükseltmeleri alacaklarına karar verirken ekonomik stratejilerini iyi bir şekilde yönetmelidirler. Bu da oyuncuyu sadece yarışta ilerlemek değil, aynı zamanda oyun içindeki kaynaklarını da verimli bir şekilde kullanmaya zorlar.

Rekabetçi dinamikler, oyuncuların sadece diğer oyuncularla sıralamada yarışmalarını sağlamaz, aynı zamanda oyun içindeki etkileşimleri de artırır. Oyuncular, yüksek sıralarda yer almak için sadece kendi yeteneklerini değil, aynı zamanda diğer oyuncuların stratejilerini ve oyun içi başarılarını göz önünde bulundurarak daha verimli kararlar almalıdırlar. Unity Gaming Services Leaderboard, aynı zamanda oyunculara etkinlik bazlı sıralamalar sunma olanağı da sağlar. Örneğin, belirli bir süre zarfında (örneğin bir hafta boyunca) en fazla kazancı elde eden oyunculara özel ödüller verilebilir. Bu tür etkinlikler, oyuncular arasında daha fazla rekabet ve sosyal etkileşim yaratır.

Oyuncuların kazançları, Leaderboard'daki sıralamalarla doğrudan bağlantılıdır. Kazandıkları paralar, araç yükseltmeleri ve yeni araçlar satın almak için harcanabilir. Bu da oyunculara, sadece daha yüksek sıralara ulaşmak için değil, aynı zamanda oyunun sunduğu çeşitli avantajları elde etmek için de motive eder. Oyuncular, her yolculuk sonrasında kazandıkları parayı sadece araçlarına harcamakla kalmaz, aynı zamanda liderlik tablosunda daha üst sıralara çıkabilmek için de bu parayı stratejik olarak kullanmalıdırlar. Bu, oyun ekonomisini daha derinlemesine anlamalarını ve daha etkili bir şekilde oyun içindeki başarılarını yönetmelerini sağlar.

Sonuç olarak, Unity Gaming Services Leaderboard entegrasyonu, oyunculara küresel bir başarı düzeyi sunarak, Taxi Simulator'ı daha sosyal ve rekabetçi bir oyun haline getirir. Oyuncular, kazandıkları puanlar ve paralar ile sadece oyun içindeki gelişimlerini izlemekle kalmaz, aynı zamanda diğer oyuncularla rekabet ederken ödüller kazanırlar. Bu sistem, oyuncuları daha stratejik düşünmeye zorlar, çünkü sadece hız ve yetenek değil, aynı zamanda ekonomik kararlar da başarıyı etkileyen faktörlerdir. Liderlik tablosu, oyunculara gerçek zamanlı olarak küresel bir başarı düzeyi sunarak, onların daha fazla etkileşimde bulunmalarını sağlar ve oyunun dinamik yapısını zenginleştirir.

# 4. TEKNİK DETAYLAR

## 4.1. Hedef Platformlar

Taxi Simulator oyununun hedef platformları, oyunun geniş bir oyuncu kitlesine ulaşabilmesi için stratejik olarak seçilmiştir. Oyun, her oyuncunun erişebileceği ve rahatça oynayabileceği bir deneyim sunmayı hedefler. Bu nedenle, oyunun platformları, hem kullanıcı tabanını genişletmek hem de her platformun sunduğu avantajlardan faydalanarak en iyi oyun deneyimini sağlamak amacıyla dikkatle belirlenmiştir.

PC, Taxi Simulator için ana hedef platformlardan biridir. PC oyuncuları, oyunları genellikle daha yüksek performansla oynama şansına sahip olup, geniş bir oyun kütüphanesine ve modifikasyon imkanlarına erişim sağlarlar. Oyun, Windows ve Mac işletim sistemlerinde sorunsuz çalışacak şekilde optimize edilmiştir. PC oyuncuları genellikle daha güçlü sistemlere sahip oldukları için oyun, yüksek çözünürlükte grafikler ve daha fazla görsel detaya sahip olabilir. Bu, özellikle görsel tasarımlarına ve şehirlerin dinamik yapısına önem veren bir oyun için büyük bir avantaj sağlar. Ayrıca, PC oyuncuları genellikle mouse ve klavye kombinasyonu ile oynayacakları için oyun içindeki detaylı kontrol ve hassasiyet de oldukça önemlidir.

Windows kullanıcıları için, oyun DirectX ve Vulkan API gibi platform bağımsız teknolojilerle uyumlu hale getirilmiştir, bu da oyun içindeki grafiklerin yüksek kalitede ve optimize şekilde sunulmasını sağlar. Mac kullanıcıları için ise, oyun Metal API kullanılarak optimize edilmiştir. Bu platformda, özellikle görsel öğeler ve grafiksel efektler özenle tasarlanarak daha az donanım gücüyle bile yüksek performans sağlanması hedeflenmiştir. PC platformu, oyunculara modlar ve eklentiler gibi kişiselleştirme seçenekleri sunarak oyunun ömrünü uzatır. Bu platformda, kullanıcılar topluluk tarafından üretilen içerikleri kolayca entegre edebilir ve oyun deneyimlerini değiştirebilirler. Ayrıca, PC platformu, çevrimdışı ve çevrimiçi çok oyunculu özelliklerle oyunculara farklı oyun modları sunma imkanı tanır.

Oyunun bir diğer hedef platformu da konsollardır. PlayStation ve Xbox platformları, daha geniş bir oyuncu kitlesine ulaşmayı sağlayan popüler oyun platformlarıdır. Taxi Simulator'ın PlayStation 4, PlayStation 5, Xbox One ve Xbox Series X/S gibi konsollarda sorunsuz bir şekilde çalışacak şekilde optimize edilmesi hedeflenmiştir. Konsol versiyonları, kullanıcıların oyunları

büyük ekranlarda rahatça oynamalarına ve daha dinamik bir deneyim elde etmelerine olanak tanır. Konsollarda oynama deneyimi genellikle gamepad kullanımı ile sınırlıdır. Bu durum, oyun içindeki kontrollerin oldukça akıcı ve kolay erişilebilir olmasını gerektirir. Taxi Simulator'da, özellikle araç kullanımı, yolculuk yapma ve şehirde navigasyon gibi işlemler için gamepad desteği güçlendirilmiş ve en rahat şekilde yapılabilecek bir kontrol mekanizması geliştirilmiştir. Ayrıca, konsol versiyonları için daha hızlı menü geçişleri, basitleştirilmiş arayüzler ve oyun içinde daha erişilebilir navigasyon sistemleri de tasarlanmıştır.

Konsol oyuncuları için de, PlayStation Network (PSN) ve Xbox Live gibi çevrimiçi sistemlere entegrasyon sağlanarak, oyuncular arasında liderlik tabloları ve çok oyunculu etkinlikler üzerinden etkileşim sağlanacaktır. Konsol platformları, aynı zamanda ödüller ve başarımlar gibi entegre sistemlerle oyunculara ekstra motivasyon sağlayarak, oyun içi deneyimi daha çekici hale getirir. Konsol sürümleri, daha büyük ekranlarda daha görsel bir deneyim sunarak, Taxi Simulator'ı daha etkileyici bir oyun haline getirir.

Mobil cihazlar, Taxi Simulator için de önemli bir hedef platformdur. iOS ve Android işletim sistemlerine sahip telefon ve tabletlerde de bu oyunun oynanabilmesi sağlanacaktır. Mobil versiyon, daha geniş bir oyuncu kitlesine ulaşmak için önemli bir fırsat yaratır. Mobil platformlarda, oyun kontrolleri, özellikle dokunmatik ekranlar için özelleştirilmiş olacak şekilde tasarlanmıştır. Bunun yanı sıra, mobil cihazlarda daha kompakt bir ekran deneyimi sunduğu için arayüzde de basitlik ve kullanıcı dostu özellikler ön planda tutulacaktır. Mobil versiyon, daha küçük ekranlar ve sınırlı işlem gücü göz önünde bulundurularak optimize edilecektir. Bu platformda grafikler, yüksek çözünürlükte olmasa da, performansı etkileyebilecek gereksiz detaylardan kaçınılarak, mobil cihazlarda daha hızlı ve verimli bir deneyim sağlanacaktır. Ayrıca, mobil cihazlar için yenilikçi kontrol mekanizmaları (örneğin, eğme sensörleri, ekran dokunuşlarıyla hızlanma ve frenleme) gibi özellikler sunulacak, böylece oyuncular mobil cihazlarında da rahatça araç kullanabileceklerdir.

Mobil oyunlar, oyunculara yolda giderken, kısa süreli oyun seansları oynama imkanı tanır. Bu, özellikle zaman kısıtlaması olan oyuncular için büyük bir avantajdır. Mobil cihazlarda oyun deneyimi genellikle daha kısa süreli, ancak daha erişilebilir olur. Bu nedenle, oyun tasarımında, oyuncuların kısa süreli oyun seanslarında eğlenebileceği ve sürekli ilerleyebileceği bir sistem geliştirilmiştir.

Bir diğer önemli özellik, cross-play yani çapraz platform oyun desteği olacaktır. Bu, oyuncuların farklı platformlardan (PC, PlayStation, Xbox ve mobil cihazlar) birbirleriyle etkileşime girebileceği anlamına gelir. Taxi Simulator, oyunculara sadece kendi platformlarındaki oyuncularla değil, tüm platformlardaki oyuncularla da oyun oynama fırsatı sunar. Bu, oyun

topluluğunun birleşmesini ve genişlemesini sağlar. Çapraz platform desteği, özellikle çok oyunculu oyunlar için önemlidir ve oyuncuların farklı cihazlarda da birlikte oyun oynayabilmelerine olanak tanır.

Taxi Simulator'ın hedef platformları, her türden oyuncuya ulaşmak amacıyla çeşitlendirilmiştir. PC, konsollar ve mobil cihazlar olmak üzere üç ana platformda oyun sunulacak, her platformun kendine özgü avantajları kullanılarak en iyi deneyim sağlanacaktır. Oyunun bu kadar geniş bir platform yelpazesinde yer alması, oyunculara farklı cihazlarda ve farklı zamanlarda rahatça oynama imkanı tanır. Ayrıca, Unity Gaming Service sayesinde sağlanan özellikler ve platformlar arası entegrasyon, oyunculara sosyal bir deneyim sunarak oyun içindeki rekabeti artırır ve daha dinamik bir topluluk oluşturur.

### 4.2. Kullanılan Oyun Motoru Unity

Unity, oyun geliştirme dünyasında en popüler ve güçlü araçlardan biridir. Hem bağımsız oyun geliştiricilerinin hem de büyük stüdyoların tercihi olan bu motor, çok platformlu oyun geliştirme ve gerçek zamanlı 3D ve 2D grafikler sunma konusunda benzersiz özelliklere sahiptir. Taxi Simulator gibi oyunların temelinde yer alan Unity, oyun dünyasında devrim yaratmış bir oyun motorudur.

Unity'nin tarihçesi, 2000'li yılların başlarına dayanmaktadır. Unity Technologies, 2004 yılında Danmark, Kopenhag'da David Helgason, Joachim Ante ve Nicholas Francis tarafından kurulmuştur. İlk başlarda, yalnızca Mac platformu için geliştirilmiş bir araç olarak başlayan Unity, zamanla platform bağımsız bir oyun motoru haline gelmiştir. Oyun motorunun hızla popülerleşmesi, özellikle bağımsız oyun geliştiricilerine sunduğu kolaylıklar ve uygun fiyatlandırma sayesinde olmuştur. Unity'nin hızla yaygınlaşmasında önemli bir rol oynayan diğer bir etken, motorun sunduğu gelişmiş araçlar ve özellikler ile oyun geliştiricilerinin, daha kısa sürelerde daha kaliteli oyunlar yapabilmesini sağlamış olmasıdır.

Unity'nin büyümesi, 2005'te Windows platformu desteği eklenmesiyle ivme kazandı. Ardından, PlayStation, Xbox, Nintendo Switch gibi konsollarla da uyumlu hale gelerek, farklı platformlarda oyun geliştirmenin önünü açtı. Bugün, Unity, PC, konsol, mobil, VR, AR ve web platformlarında oyun geliştirmeyi mümkün kılar ve her platformda yüksek performans sunar. Bu çoklu platform desteği, oyun geliştiricilerinin farklı cihazlarda aynı oyunu çalıştırabilmesine olanak tanır ve oyunların daha geniş bir oyuncu kitlesine ulaşmasını sağlar.

Unity, dünya çapında birçok başarılı oyunun geliştirilmesinde kullanılmıştır. Bu oyunlar, Unity'nin ne kadar güçlü ve esnek bir motor olduğunu gösterir. Öne çıkan bazı Unity ile yapılmış oyunlar arasında, Pokémon GO, Monument Valley 2, Cuphead, Hollow Knight, Among Us, Fall Guys, Untitled Goose Game ve Subnautica yer alır. Bu oyunlar, Unity'nin farklı türlerde ve tarzlarda oyunlar geliştirmek için ne kadar esnek bir motor olduğunu göstermektedir. Unity, her türden oyunun geliştirilmesine olanak tanır ve geliştiricilere geniş bir araç seti sunar.

Unity'nin sunduğu teknolojiler ve servisler, oyun geliştirmeyi daha verimli ve etkileşimli hale getiren unsurlar arasında yer alır. Unity, oyunların grafiklerini, fizik hesaplamalarını, animasyonlarını ve yapay zekalarını yönetmek için geliştirilmiş birçok yerleşik araca sahiptir. Bunun dışında, Unity Asset Store üzerinden sağlanan varlıklar (3D modeller, ses dosyaları, animasyonlar ve diğer içerikler) sayesinde, oyun geliştiriciler hızla projelerine yeni içerikler ekleyebilir ve zaman kazandırabilirler.

Unity'nin grafik teknolojisi açısından oldukça güçlü olduğu söylenebilir. High Definition Render Pipeline (HDRP) ve Universal Render Pipeline (URP) gibi gelişmiş render sistemleri, oyunların görsel kalitesini artırırken, farklı platformlarda yüksek performans sağlar. Ayrıca, PhysX fizik motoru, oyunlarda gerçekçi fizik hesaplamaları yapmayı mümkün kılar ve oyunculara gerçekçi araç kullanım deneyimi sunar. Unity, aynı zamanda real-time lighting ve shadow mapping gibi özelliklerle oyun dünyasında gerçekçilik ve görsellik sağlar.

Unity'nin multiplatform desteği, oyun geliştiricilerinin tek bir oyun kodu ile birçok farklı platformda oyun geliştirmelerini mümkün kılar. Bir kez yazdığınız oyun kodu, iOS, Android, Windows, PlayStation, Xbox, Nintendo Switch, WebGL ve hatta VR (Sanal Gerçeklik) ve AR (Artırılmış Gerçeklik) gibi cihazlarda çalışabilir. Bu, geliştiricilerin oyunlarını çeşitli platformlarda yayınlama imkânı tanır ve daha geniş bir oyuncu kitlesine ulaşmalarını sağlar. Unity'nin bu özelliği, özellikle oyun dünyasında daha fazla etkileşim ve oyuncu deneyimi yaratmak adına önemli bir avantaj sunar.

Unity'nin sunduğu Unity Services, oyun geliştirmenin her aşamasında yardımcı olan çeşitli servisleri içerir. Unity Analytics, geliştiricilerin oyun içi oyuncu davranışlarını analiz etmelerini sağlar ve oyunlarını bu verilere göre optimize etmelerine yardımcı olur. Unity Ads, geliştiricilere reklam ekleyerek oyunlardan gelir elde etme imkânı sunar. Bu, özellikle ücretsiz oyunlar için önemli bir gelir kaynağı olabilir. Unity IAP (In-App Purchasing), oyun içindeki mikro işlemleri yönetir ve oyuncuların oyun içi satın alımlar yapmasını sağlar. Ayrıca, Unity Multiplayer, çok oyunculu oyun deneyimleri için altyapı sağlar ve oyuncuların internet üzerinden birbirleriyle etkileşimde bulunmalarını mümkün kılar.

Unity'nin Asset Store'u, oyun geliştiricilerinin yalnızca oyun içi varlıkları (modeller, animasyonlar, sesler, vb.) almasına değil, aynı zamanda gameplay sistemleri, araçlar ve plug-in'ler gibi geliştirme araçlarını da satın almasına olanak tanır. Bu özellik, geliştiricilerin kendi projelerinde yer alacak yeni özellikleri hızla entegre etmelerini sağlar. Oyun geliştiricilerinin projelerini daha hızlı bir şekilde tamamlamalarına yardımcı olan Asset Store, Taxi Simulator gibi oyunların hızlı bir şekilde gelişmesini sağlar.

Unity'nin sunduğu Yapay Zeka (AI) ve Pathfinding teknolojileri, oyunlardaki karakterlerin, araçların ve diğer öğelerin dünyada akıllıca hareket etmelerini sağlar. Unity'nin NavMesh teknolojisi, oyun içindeki karakterlerin, araçların ve diğer dinamik öğelerin engellerden kaçınarak doğru yolları takip etmelerini sağlar. Bu teknoloji, özellikle Taxi Simulator gibi açık dünya oyunlarında, yolcuların ve diğer araçların gerçekçi bir şekilde hareket etmelerini sağlamak için kullanılır.

VR ve AR teknolojilerinin gelişmesiyle, Unity, oyun içi deneyimleri daha etkileyici hale getirecek birçok özellik sunar. Taxi Simulator gibi oyunlar, sanal gerçeklik (VR) ile oynanabilir hale getirildiğinde, oyuncuların gerçekçi bir şekilde sürüş yapmalarını sağlayan eşsiz bir deneyim ortaya çıkar. Unity, HTC Vive, Oculus Rift ve PlayStation VR gibi cihazlarla uyumlu oyunlar geliştirebilmek için tüm altyapıyı sağlar. Benzer şekilde, artırılmış gerçeklik (AR) oyunları da Unity motoruyla yapılabilir ve oyunculara farklı cihazlar üzerinden yeni ve ilginç oyun deneyimleri sunulabilir.

Unity'nin modüler yapısı, geliştiricilerin ihtiyaç duydukları özellikleri seçerek oyunlarına entegre etmelerini sağlar. Taxi Simulator'da, örneğin, araçların gerçekçi bir şekilde hareket etmesi için Rigidbody bileşeni kullanılabilirken, UI (Kullanıcı Arayüzü) için ise Canvas bileşeni kullanılabilir. Bu tür modüller sayesinde, oyun geliştiricileri, sadece oyunlarının ihtiyaç duyduğu fonksiyonları kullanarak, daha verimli ve hafif bir oyun yaratabilirler.

Sonuç olarak, Unity oyun motoru, Taxi Simulator gibi oyunların geliştirilmesi için güçlü bir platform sağlar. Unity'nin sunduğu yüksek kaliteli grafikler, çoklu platform desteği, kullanıcı dostu araçlar ve oyun geliştirmeyi kolaylaştıran hizmetler, bu motoru oyun dünyasında lider bir araç haline getirmiştir. Hem küçük bağımsız oyun geliştiricileri hem de büyük stüdyolar, Unity ile projelerini hayata geçirebilir ve daha geniş bir oyuncu kitlesine ulaşabilirler. Oyun dünyasında yer edinmek isteyen her geliştirici, Unity'nin sunduğu olanaklardan yararlanarak daha etkileyici ve verimli oyunlar yaratabilir.

### 4.3. Kontroller

Taxi Simulator tamamlayarak, yolcuları taşıyarak ve trafik engellerinden kaçınarak, şehirdeki zorluklarla başa çıkmak zorundadırlar. Bu nedenle, kontrollerin doğruluğu ve akıcılığı son derece önemlidir.

PC platformu için kontroller, klavye ve fare kombinasyonu ile yapılandırılmıştır. Oyuncular, araçları yönlendirmek için W, A, S, D tuşlarını kullanarak hareket ederler. W tuşu hızlanmayı, S tuşu ise frenlemeyi sağlar. A ve D tuşları ise aracı sola ve sağa yönlendirmek için kullanılır. Fare ise oyuncuya kamerayı yönlendirme imkanı tanır, böylece oyuncu etrafındaki çevreyi kolayca inceleyebilir ve araçla ilgili hassas manevraları yapabilir. Mouse kullanılarak, oyuncular aracın bakış açısını değiştirebilir, bu da özellikle dönüşlerde ve trafikteki dar alanlarda oyuncunun doğru hamle yapmasını sağlar.

Konsol platformlarında, oyuncuların gamepad kullanması beklenmektedir. Konsol versiyonlarında, araç kontrolü için joystick kullanımı daha etkili bir seçenek olarak tasarlanmıştır. Sağ analog joystick, oyuncuya aracı çevirebilme ve yönlendirme imkanı tanırken, sol analog joystick ise aracın hızını kontrol etmelerini sağlar. X ve B tuşları, aracın fren ve hızlanma işlemleri için atanırken, A tuşu, aracın dönüş sistemine odaklanarak, oyuncunun manevra yapabilmesi için kullanılır. Ayrıca, L1 ve R1 tuşları, oyunculara hızlanma ve frenleme işlemlerinde ek hassasiyet sunar. Bu, konsol oyuncularına daha hızlı ve rahat bir oyun deneyimi sağlamaktadır.

Mobil cihazlar için kontroller, dokunmatik ekran tabanlı bir tasarım kullanılarak özelleştirilmiştir. Ekranın sağ kısmı, oyuncunun araç kontrolünü sağlar. Burada, dokunmatik pedal simgeleri yer alır, bu simgeler oyuncunun aracı hızlandırıp yavaşlatmasına imkan verir. Ekranın sol kısmı, oyuncunun aracı yönlendirmesini sağlayan sanal bir joystick içerir. Bu, oyunculara daha hassas ve çevik bir sürüş deneyimi sunar, ancak ekranın küçüklüğü nedeniyle, kontrollerin basit ve hızlı bir şekilde erişilebilir olması önemlidir. Bu nedenle, mobil cihazlarda kullanılan kontroller, kullanıcı dostu olacak şekilde tasarlanmış, karmaşık komutlardan kaçınılmıştır.

Kontrol şeması ise oyun içinde, her cihaz için farklı bir yapı sunmaktadır. PC oyuncuları için, W, A, S, D tuşları ile araç yönlendirmeleri, farenin sağ tuşu ile de araç kameralarını ayarlama işlemi yapılabilir. Konsol oyuncuları, joystick'leri ve yön tuşlarıyla aracı kontrol ederken, mobil oyuncular sadece ekrana dokunarak aracın yönünü ve hızını ayarlayabilirler. Bu kontrol şeması, her platformda oyuncunun rahatça erişebileceği, sezgisel ve hızlı bir kontrol mekanizması

oluşturur. Ayrıca, her platformun kendine özgü gereksinimlerine ve kullanıcı alışkanlıklarına göre optimize edilmiş bir deneyim sağlar.

Kontrol şeması, özellikle araç kullanımıyla ilgili karmaşık görevlerde oyuncuya rehberlik edecek şekilde tasarlanmıştır. Oyuncu, araçları farklı hızlarda kullanarak, şehirdeki trafiği aşmak ve yolcularını zamanında teslim etmek zorundadır. Bu nedenle, kontrollerin hassas ve hızlı olması gerekir. Ayrıca, her platform için ayrı bir kontrol şeması geliştirilmesi, oyuncuların her cihazda benzer bir deneyimi yaşayabilmesini sağlar. Her bir kontrol şeması, oyuncuların oyun içindeki etkileşimlerini daha sezgisel hale getirir, böylece oyun deneyimi olabildiğince akıcı olur.

Taxi Simulator oyununda, kontrollerin optimizasyonu, oyunculara gerçekçi bir sürüş deneyimi sunma amacını taşır. Bu, oyun dünyasında araç kullanımı sırasında karşılaşılan zorlukları aşarken oyuncuları teşvik eder. Kontrollerin doğru, hassas ve kolay erişilebilir olması, oyuncunun dikkatini sadece oyunun amacına yönlendirmesini sağlar. Oyun içindeki her hareket, oyuncunun oyun deneyimiyle olan etkileşimini güçlendirir.

oyununda kontroller, oyuncuların oyun içindeki etkileşimlerini kolay ve verimli bir şekilde yönetebilmeleri için büyük bir öneme sahiptir. Oyuncunun oyunla etkileşim kurarken rahat bir deneyim yaşaması, oyunun başarısının temel unsurlarından biridir. Bu nedenle, kontroller hem basit hem de oyun içindeki görevlerin gerektirdiği doğruluğa ve hassasiyete uyumlu olacak şekilde tasarlanmıştır.

Kontroller, özellikle oyuncunun oyun dünyasında araçları yönetmesini sağlamak için optimize edilmiştir. Taxi Simulator'da araç kullanımı, oyuncuya gerçekçi bir sürüş deneyimi sunacak şekilde düzenlenmiştir. Bu kontroller, aracın hızını, yönünü, fren sistemini ve diğer önemli fonksiyonları yönetme imkanı verir. Oyuncular, oyun dünyasında görevleri

Sonuç olarak, Taxi Simulator'ın kontrol şeması, oyuncuların her platformda oyunla rahatça etkileşime girmelerini sağlamak amacıyla dikkatlice tasarlanmıştır. PC, konsol ve mobil cihazlar için özelleştirilmiş kontroller, her oyuncunun deneyimini optimize etmek için farklı özellikler sunar. Kontroller, sadece oyunu yönetmek için değil, aynı zamanda oyuncunun stratejik kararlar almasına olanak tanır. Bu denge, oyuncuya bir yandan oyun içinde yön verme özgürlüğü tanırken, bir yandan da oyun içindeki gerçekçilik ve zorluklarla başa çıkabilme imkanı sağlar. Bu nedenle, kontrollerin tasarımı, Taxi Simulator'ın başarısındaki önemli bir faktördür.

### 4.4. Performans Ve Optimizasyon

Taxi Simulator oyununda, yüksek performans ve akıcı bir oyun deneyimi sunmak amacıyla çeşitli optimizasyon teknikleri uygulanmıştır. Bu teknikler, hem görsel kaliteyi korumayı hem de donanım kaynaklarını verimli bir şekilde kullanmayı hedefler. Oyun, oyunculara daha iyi bir deneyim sunarken, donanım gereksinimlerini mümkün olduğunca düşük tutmayı amaçlar. Bu nedenle, oyun geliştirilirken her ayrıntı üzerinde özenle çalışılmıştır. Her platform için özel optimizasyon ayarları yapılmış ve her bir platformda oyun performansını en üst düzeye çıkarmak için gerekli değişiklikler yapılmıştır.

Çoğu bilgisayar oyununda, mini harita ve GPS sistemi, yeni bir kamera renderlanarak oluşturulur. Ancak Taxi Simulator'da, bu sistemin tasarımı farklıdır. Oyunun mini harita ve GPS sistemi, sahnenin kuş bakışı fotoğrafı üzerine, sahnedeki objelerin X ve Y koordinatlarının yansıtılmasıyla oluşturulmuştur. Bu yaklaşım, yeni bir kamera renderlamaya gerek kalmadan, mevcut sahne verileriyle mini harita ve GPS bilgisini sunar. Sonuç olarak, ek bir kamera renderlama işlemi yapılmadığı için, gereksiz hesaplamalardan kaçınılmış ve performans kazancı sağlanmıştır. Bu sayede, oyun performansını etkileyebilecek ekstra renderlama işlemleri ortadan kaldırılmıştır ve aynı zamanda oyun içi görseller daha hızlı bir şekilde yansıtılabilmektedir.

Oyuncular, taksiye binen yolcuları alıp, belirlenen varış noktalarına bırakırlar. Yolcular, taksiden indikten ve görünür alandan kaybolduktan sonra, oyuncuya kazançları tamamlanır. Bu noktada, yolcuların oyun dünyasında varlıkları devam etmez. Bu sayede, bellekten temizlenerek, gereksiz bellek tüketimi önlenmiş ve performans artırılmıştır. Ayrıca, bu yöntem, oyun dünyasının daha dinamik ve akıcı olmasına katkı sağlar. Bellek yönetimi, özellikle büyük açık dünya oyunlarında kritik öneme sahiptir; çünkü gereksiz nesnelerin bellekte tutulması oyun performansını doğrudan etkileyebilir. Bu optimizasyon yöntemi sayesinde, oyun belleği verimli bir şekilde yönetilmiş ve gereksiz veri yükü azaltılmıştır.

Oyun bu veriyi oyun sırasında kullanarak, hangi objelerin renderlanması gerektiğini belirler. Bu sayede, yalnızca görünür öğeler işleme alınır ve renderlama süresi kısalır. Ayrıca, Unity'nin occlusion culling sistemi, sahne objelerini hücrelere ayırarak, her hücre için görünürlük verisi oluşturur. Bu yapı, sahnedeki büyük objelerin küçük hücrelere bölünerek daha verimli bir şekilde yönetilmesini sağlar. Bu optimizasyon sayesinde, oyun dünyası genişlemesine rağmen, performans kaybı yaşanmaz. Yüksek kaliteli görseller sunulurken, sistem kaynakları verimli bir şekilde kullanılmaktadır.

Occlusion culling'in etkin bir şekilde çalışabilmesi için, sahne objelerinin doğru bir şekilde etiketlenmesi gerekir. Unity'de, objeler "Static" olarak işaretlenerek, bu objelerin hareket etmeyen ve sabit olduğunu belirtiriz. Ayrıca, "Occluder Static" ve "Occludee Static" etiketleriyle, objelerin diğer objeleri engelleyip engellemediği belirtilir. Bu etiketlemeler, occlusion culling işleminin doğruluğunu artırır ve gereksiz render işlemlerinin önüne geçer. Occlusion culling verilerinin oluşturulması (bake işlemi), sahnenin karmaşıklığına bağlı olarak zaman alabilir. Ancak, bu veriler doğru bir şekilde oluşturulduğunda, oyun performansında belirgin bir iyileşme sağlanır. Sahne içinde gereksiz objelerin bulunmaması, objelerin doğru etiketlenmesi ve occlusion culling verilerinin optimize edilmesi, performansın artırılmasında önemli rol oynar.

Occlusion culling, genellikle statik objeler için kullanılır. Ancak, dinamik objeler için de occlusion culling uygulanabilir. Unity, dinamik objeler için Occlusion Area bileşenini kullanarak, bu objelerin görünürlük bilgisini hesaplar. Occlusion Area, dinamik objelerin bulunduğu alanı tanımlar ve bu alanda hangi objelerin görünür olduğunu belirler. Bu sayede, dinamik objelerin renderlanması daha verimli hale gelir ve performans artışı sağlanır. Dinamik objeler için occlusion culling, sahnedeki gerçek zamanlı değişikliklere uyum sağlamak için oldukça önemlidir. Bu sayede, hareketli ve dinamik objeler de oyun dünyasında performans kaybı yaşamadan doğru bir şekilde işlenebilir.

Yapılan testler, occlusion culling'in oyun performansına olan etkisini göstermektedir. Occlusion culling etkinleştirildiğinde, sahnedeki görünmeyen objelerin renderlanması engellenir ve bu da CPU ve GPU üzerindeki yükü azaltır. Sonuç olarak, oyun daha akıcı çalışır ve daha düşük donanım gereksinimleriyle daha iyi performans elde edilir. Testler, sahnedeki karmaşıklığın arttığı durumlarda bile, occlusion culling sayesinde performans kaybı yaşanmadığını ve oyunun hızının arttığını göstermektedir. Bu testler, oyun dünyasında etkili bir şekilde occlusion culling uygulandığında, oyunculara yüksek kaliteli bir deneyim sunulabileceğini ortaya koymuştur.

Sonuç olarak, Taxi Simulator oyununda, performans ve optimizasyon, oyunculara kesintisiz bir deneyim sunmak için kritik öneme sahiptir. Mini harita ve GPS sisteminin geleneksel yöntemlerden farklı bir yaklaşımla tasarlanması, yolcu yönetimi ve bellek optimizasyonu, arka plan objelerinin occlusion culling yöntemiyle yönetilmesi ve Unity'nin occlusion culling teknolojisinin etkin kullanımı, bu hedeflere ulaşılmasında önemli adımlardır. Bu tekniklerin birleşimi, oyunun genel performansını artırarak, oyunculara daha keyifli bir deneyim sunar. Oyun dünyasında, her ayrıntı ve her teknik optimizasyon, performansın iyileştirilmesinde büyük bir rol oynar ve Taxi Simulator'da bu optimizasyonlar başarılı bir şekilde entegre edilmiştir.

dünyasında, arka planda görünen binalar, gökyüzü, ağaçlar, animasyonlar ve yollar gibi öğeler, sürekli olarak renderlanmak yerine, occlusion culling (görünürlük engelleme) yöntemiyle yönetilir. Bu yöntem, kameranın görüş alanında olmayan objelerin renderlanmasını engeller. Böylece, sadece oyuncunun görebileceği öğeler işleme alınır ve gereksiz render işlemlerinden kaçınılır. Occlusion culling, CPU ve GPU üzerindeki yükü azaltarak, oyun performansını artırır. Bu teknik, özellikle geniş ve açık dünya oyunlarında performansı ciddi anlamda iyileştirir çünkü tüm dünyayı aynı anda renderlamak yerine, sadece oyuncunun etrafında aktif olan öğeler işleme alınır. Bu sayede, sahnedeki gereksiz objeler için işlem yapılmaz ve yalnızca oyuncunun görebileceği unsurlar işleme alınarak verimlilik sağlanır.

Unity, occlusion culling işlemini daha verimli hale getirmek için Umbra teknolojisini kullanır. Umbra, sahnedeki potansiyel olarak görünür öğelerin hiyerarşik bir yapısını oluşturur ve

# 5. KAYNAK KODLAR VE TASARIM

## 5.1. C# Kodları Ve Açıklamaları

Bu bölümde, Taxi Simulator oyununun temel yapı taşlarını oluşturan kodlar, animasyonlar ve tasarım unsurları üzerinde duracağız. Oyun dünyasının işleyişini yöneten kodlar, oyuncu etkileşimlerinden araç kontrolüne kadar pek çok fonksiyonu gerçekleştirir. Bu bölümde, kullanılan C# kodlarının nasıl yazıldığına, animasyonların nasıl tetiklendiğine ve oyun tasarımına nasıl katkı sağladığına dair detaylı bir inceleme yapılacaktır.

İlk olarak, kodlar ve oyun mantığı üzerinde duracağız. Oyun içindeki temel işlevlerin, özellikle araç kullanımı, yolcu alma ve bırakma gibi mekaniklerin nasıl yazıldığına ve işlediğine odaklanacağız. Burada kullanılan C# kodları, oyun mantığının her aşamasında oyuncuya akıcı bir deneyim sunmak için kritik bir rol oynamaktadır. Ayrıca, animasyonların nasıl çalıştığını ve hangi tetikleyicilerle aktive olduğunu da inceleyeceğiz.

Animasyonlar, oyuncunun oyundaki etkileşimlerini daha dinamik ve gerçekçi hale getirir. Bu bölümde, animasyon sistemlerinin nasıl oluşturulduğu ve hangi animasyon geçişlerinin kullanıldığına bakacağız. Unity'nin Animator Controller ile yapılan animasyon düzenlemeleri ve bu animasyonların oyun içindeki obje veya karakterlere nasıl entegre edildiğini detaylı bir şekilde ele alacağız.

Son olarak, görsel tasarım unsurları ve UI (Kullanıcı Arayüzü), oyunun görünümünü ve kullanıcı etkileşimini nasıl yönettiği üzerine konuşacağız. Oyun içindeki menüler, butonlar ve HUD elemanları ile nasıl bir deneyim sunulduğu ve kullanıcı verilerinin UI üzerinden nasıl görüntülendiği gibi konuları detaylandıracağız. Bu bölümde ayrıca ses tasarımı ve animasyonlarla olan entegrasyonu inceleyerek oyun içindeki görsel ve işitsel etkileşimin nasıl bir bütün oluşturduğuna bakacağız.

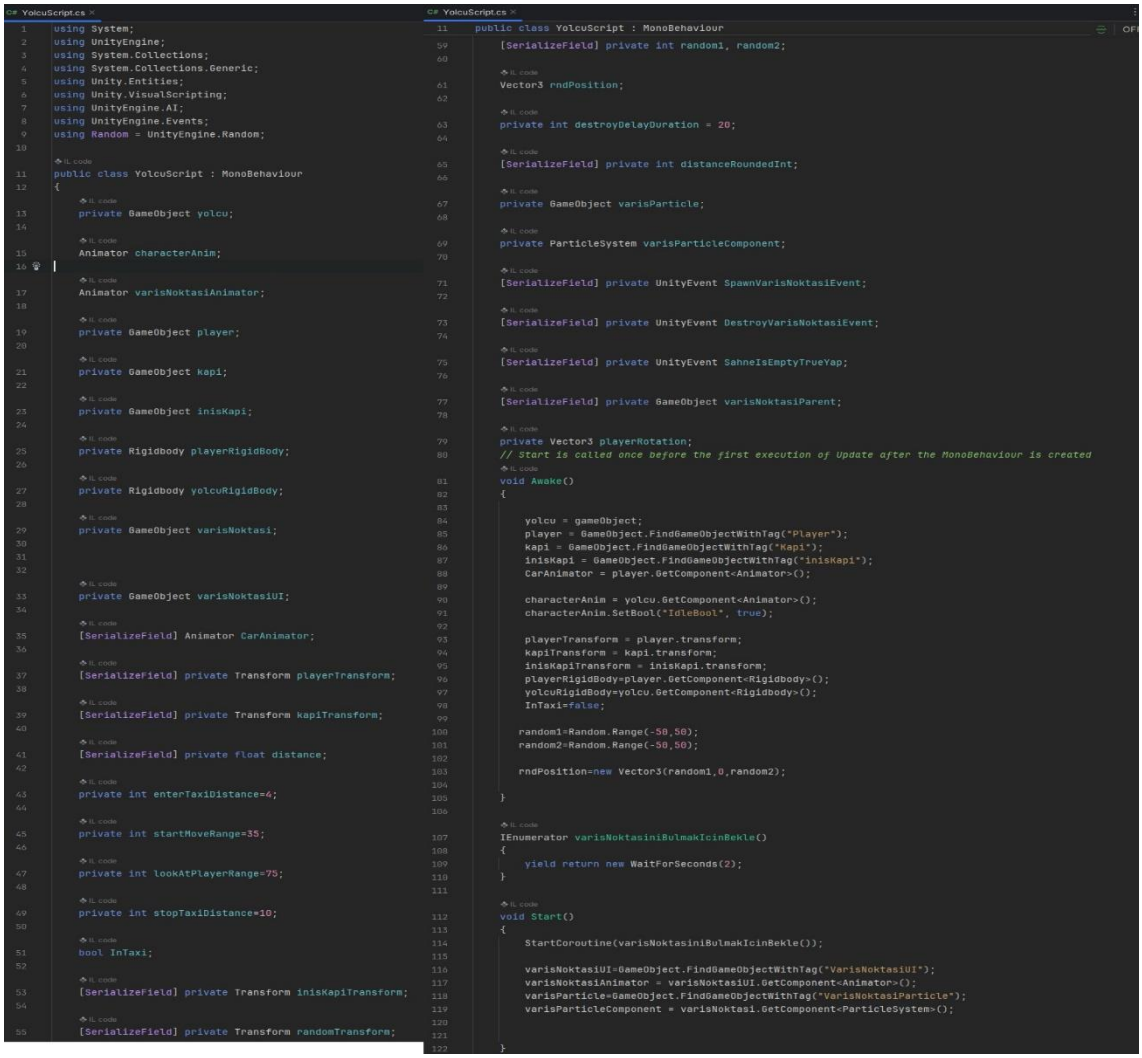
Bu başlıklar altında, sadece kullanılan teknolojilere değil, aynı zamanda tasarımın ve yazılımın oyun deneyimine nasıl katkı sağladığını da değerlendireceğiz.

### 5.1.1. Yolcu Kodu

*Şekil 5.1. Yolcunun Taksi Çağırması*

YolcuKod, taksideki yolcunun davranışlarını ve etkileşimlerini yöneten önemli bir bileşendir. Bu Kod, yolcunun animasyonları, taksinin kontrolü ve yolcunun varış noktalarına ulaşmasını sağlamak için kullanılan çeşitli fonksiyonları içerir. Oyuncunun oyunu oynarken taksiye binmesini, varış noktasına ulaşmasını ve taksiden inmesini simüle eden temel oyun mekaniği bu Kodle gerçekleştirilir.

Kod'teki yolcu değişkeni, yolcu GameObject'ini temsil eder. Bu objeye tüm işlemler ve animasyonlar uygulanacaktır. characterAnim ise yolcunun animasyonlarını kontrol etmek için kullanılan Animator bileşenidir. varisNoktasiAnimator varış noktası ile olan etkileşimi yöneten animatördür. player değişkeni, oyuncu karakterini temsil eden GameObject olup, yolcunun oyun içindeki hareketlerini etkileyecek olan temel objedir. kapi ve inisKapi, taksiye biniş ve iniş kapılarının GameObject'leridir. playerRigidBody ve yolcuRigidBody, oyuncu ve yolcu objelerinin fiziksel etkileşimlerini yöneten Rigidbody bileşenleridir. varisNoktasi, yolcunun gitmesi gereken varış noktasını temsil ederken, varisNoktasiUI varış noktası ile ilgili UI öğelerini temsil eder.



```
using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using Unity.Entities;
using Unity.VisualScripting;
using UnityEngine.AI;
using UnityEngine.Events;
using Random = UnityEngine.Random;

public class YolcuScript : MonoBehaviour
{
    private GameObject yolcu;

    Animator characterAnim;

    Animator varisNoktasiAnimator;

    private GameObject player;

    private GameObject kapi;

    private GameObject inisKapi;

    private Rigidbody playerRigidBody;

    private Rigidbody yolcuRigidBody;

    private GameObject varisNoktasi;



    private GameObject varisNoktasiUI;

    [SerializeField] Animator CarAnimator;

    [SerializeField] private Transform playerTransform;

    [SerializeField] private Transform kapiTransform;

    [SerializeField] private float distance;

    private int enterTaxiDistance=4;

    private int startMoveRange=35;

    private int lookAtPlayerRange=75;

    private int stopTaxiDistance=10;

    bool InTaxi;

    [SerializeField] private Transform inisKapiTransform;

    [SerializeField] private Transform randomTransform;
```

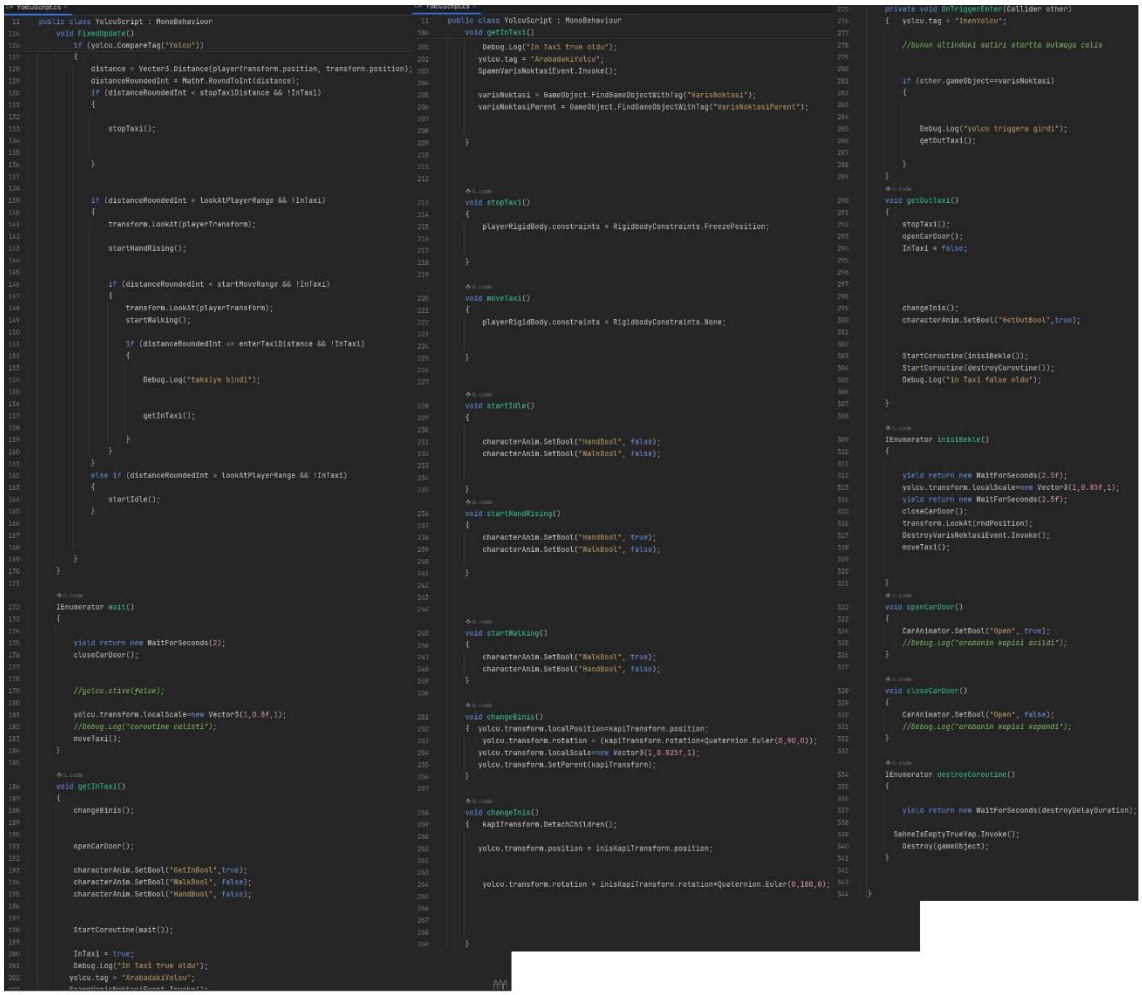
```
public class YolcuScript : MonoBehaviour
    [SerializeField] private int random1, random2;

    Vector3 rndPosition;

    private int destroyDelayDuration = 20;

    [SerializeField] private int distanceRoundedInt;

    private GameObject varisParticle;

    private ParticleSystem varisParticleComponent;

    [SerializeField] private UnityEvent SpawnVarisNoktasiEvent;

    [SerializeField] private UnityEvent DestroyVarisNoktasiEvent;

    [SerializeField] private UnityEvent SahneIsEmptyTrueYap;

    [SerializeField] private GameObject varisNoktasiParent;

    private Vector3 playerRotation;
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Awake()
    {

        yolcu = gameObject;
        player = GameObject.FindGameObjectWithTag("Player");
        kapi = GameObject.FindGameObjectWithTag("Kapi");
        inisKapi = GameObject.FindGameObjectWithTag("inisKapi");
        CarAnimator = player.GetComponent<Animator>();

        characterAnim = yolcu.GetComponent<Animator>();
        characterAnim.SetBool("IdleBool", true);

        playerTransform = player.transform;
        kapiTransform = kapi.transform;
        inisKapiTransform = inisKapi.transform;
        playerRigidBody=player.GetComponent<Rigidbody>();
        yolcuRigidBody=yolcu.GetComponent<Rigidbody>();
        InTaxi=false;

       random1=Random.Range(-50,50);
       random2=Random.Range(-50,50);

       rndPosition=new Vector3(random1,0,random2);

    }

    IEnumerator varisNoktasiniBulmakIcinBekle()
    {
        yield return new WaitForSeconds(2);
    }

    void Start()
    {
        StartCoroutine(varisNoktasiniBulmakIcinBekle());

        varisNoktasiUI=GameObject.FindGameObjectWithTag("VarisNoktasiUI");
        varisNoktasiAnimator = varisNoktasiUI.GetComponent<Animator>();
        varisParticle=GameObject.FindGameObjectWithTag("VarisNoktasiParticle");
        varisParticleComponent = varisParticle.GetComponent<ParticleSystem>();


    }
```

***Şekil 5.1.1. Yolcu Sınıfı Kodu***

CarAnimator, oyuncunun kullandığı taksinin animasyonlarını kontrol eder. random1 ve random2, yolcunun rastgele bir pozisyon oluşturmasını sağlayan değerlerdir. Bu değerler yolcunun rastgele bir başlangıç noktası seçmesine olanak tanır. destroyDelayDuration, yolcu objesinin yok edilmeden önce beklemesi gereken süreyi belirler. OluşturmaVarisNoktasiEvent ve DestroyVarisNoktasiEvent, varış noktası oluşturulması ve yok edilmesi için kullanılan Unity Event'leridir. SahneIsEmptyTrueYap, sahnenin boş olup olmadığını kontrol etmek için kullanılan bir event'tir. varisParticle ise varış noktasıyla ilişkili partikül sistemini temsil eder.

Awake() metodunda, Kod'in başlatılmadan önceki ilk aşaması olan bu metod ile oyun objelerinin referansları alınır. Yolcu objesi, oyuncu, kapılar ve animasyon bileşenleri bu metod içinde bulunur ve yolcu objesinin animasyonlarının başlatılması sağlanır. Ayrıca, yolcunun rastgele bir pozisyon seçmesi için random1 ve random2 değişkenleri oluşturulur. Start()

metodunda, varış noktası ve UI öğeleri ile ilgili animatörler ve partikül sistemleri ayarlanır. Ayrıca, varış noktası ile ilgili olaylar tetiklenmeden önce 2 saniyelik bir bekleme süresi başlatılır. Bu, yolcu ve taksi etkileşimlerinin düzgün bir şekilde başlatılabilmesi için gereklidir.



*Şekil 5.1.2. Yolcu Sınıfı Kodu*

FixedUpdate() metodunda, her bir oyun çerçevesinde düzenli olarak çağrılır ve burada yolcunun player (oyuncu) ile olan mesafesi ölçülür. Yolcu, oyuncuya belirli bir mesafeye yaklaştıkça farklı animasyonlar tetiklenir ve farklı olaylar gerçekleşir. Eğer yolcu, taksiye 10 birim mesafeye gelirse, stopTaxi() metodu çağrılır. Bu, oyuncunun aracının durmasını sağlar. Yolcu oyuncuya yaklaşırsa, startHandRising() metoduyla, yolcu elini kaldırır (bu animasyon, yolcunun taksi çağırmaya başladığı anlamına gelir). Yolcu oyuncuya 35 birim mesafeye yaklaşırsa, startWalking() metodu ile yürümeye başlar. Eğer yolcu, oyuncuya çok yaklaşırsa (4 birim mesafeye gelir), getInTaxi() metodu taksiye binmesini sağlar.

getInTaxi() metodu, yolcunun taksiye binme animasyonunu başlatır. changeBinis() metodu yolcunun pozisyonunu taksiye yerleştirir ve CarAnimator ile aracın kapısını açar. Yolcu

taksiye binmeden önce, animasyonları ve yürüyüş hareketleri yönetilir. InTaxi değişkeni true olarak ayarlanır ve yolcunun artık takside olduğu bilgisi sistemde tutulur. Varış noktası oluşturulmadan önce belirli bir işlem yapılır ve yolcu, "ArabadakiYolcu" etiketi ile işaretlenir.

stopTaxi() metodu, oyuncunun fiziksel hareketini durdurur ve taksinin durmasını sağlar. moveTaxi() metodu ise, oyuncunun hareketine tekrar izin verir ve taksinin hareket etmeye devam etmesini sağlar. startIdle(), yolcunun bekleme durumuna geçmesini sağlar. startHandRising() elini kaldırır ve startWalking() ise yolcunun yürümeye başlamasını sağlar.

changeBinis() metodunda, yolcunun taksiye binerken pozisyonu ve rotası ayarlanır. Yolcu, kapı ile aynı hizaya getirilir ve taksiye doğru yerleştirilir. changeInis(), yolcunun taksiden indiği zaman, yolcuyu kapıdan çıkarır ve yeni bir pozisyon ayarlar.

getOutTaxi() metodu yolcunun taksiden inme işlemidir. Yolcu, varış noktasına ulaştığında, taksiden inmek için gerekli animasyonlar başlatılır ve yolcu belirli bir süre sonra yok edilir. getOutTaxi() metodu, yolcunun taksiden inmesini sağlar ve taksinin kapısı kapanır. OnTriggerEnter() metodu, yolcunun varış noktasına ulaştığında tetiklenir. Yolcu varış noktasına geldiğinde, getOutTaxi() metodu çağrılır ve yolcu taksiden iner. Ayrıca, yolcunun yeni etiketi "InenYolcu" olarak değiştirilir.

destroyCoroutine() metodu, yolcunun taksiden indikten sonra belirli bir süre beklemesini ve ardından yok edilmesini sağlar. destroyDelayDuration süresi boyunca yolcu bekler, ardından yok edilir ve sahne boşalır. openCarDoor() ve closeCarDoor() metotları, aracın kapısının açılmasını ve kapanmasını kontrol eder. CarAnimator ile aracın kapı animasyonları tetiklenir. wait(), yolcunun taksiye binerken ve taksiden inerken geçireceği bekleme sürelerini kontrol eder. inisiBekle(), yolcu taksiden indikten sonra geçireceği bekleme süresi boyunca animasyonları ve sahne düzenlemelerini yönetir.

### 5.1.2. Yolcu Oluşturucu Kodu

YolcuOluşturucu Kod'i, oyunda yolcuların rastgele bir şekilde oluşturulmasini yöneten bir bileşendir. Bu Kod, belirli bir alanda yolcuların ortaya çıkmasını sağlar ve sahneye yeni yolcu eklemeye devam etmek için gereken ayarları yönetir. Ayrıca, yolcuların NavMesh üzerinde düzgün bir şekilde oluşturulması sağlanır. YolcuOluşturucu, yolcuların oyunda rastgele bir pozisyonda yer almasını ve taksiye binmeleri için oyun dünyasında aktif hale gelmelerini sağlar.

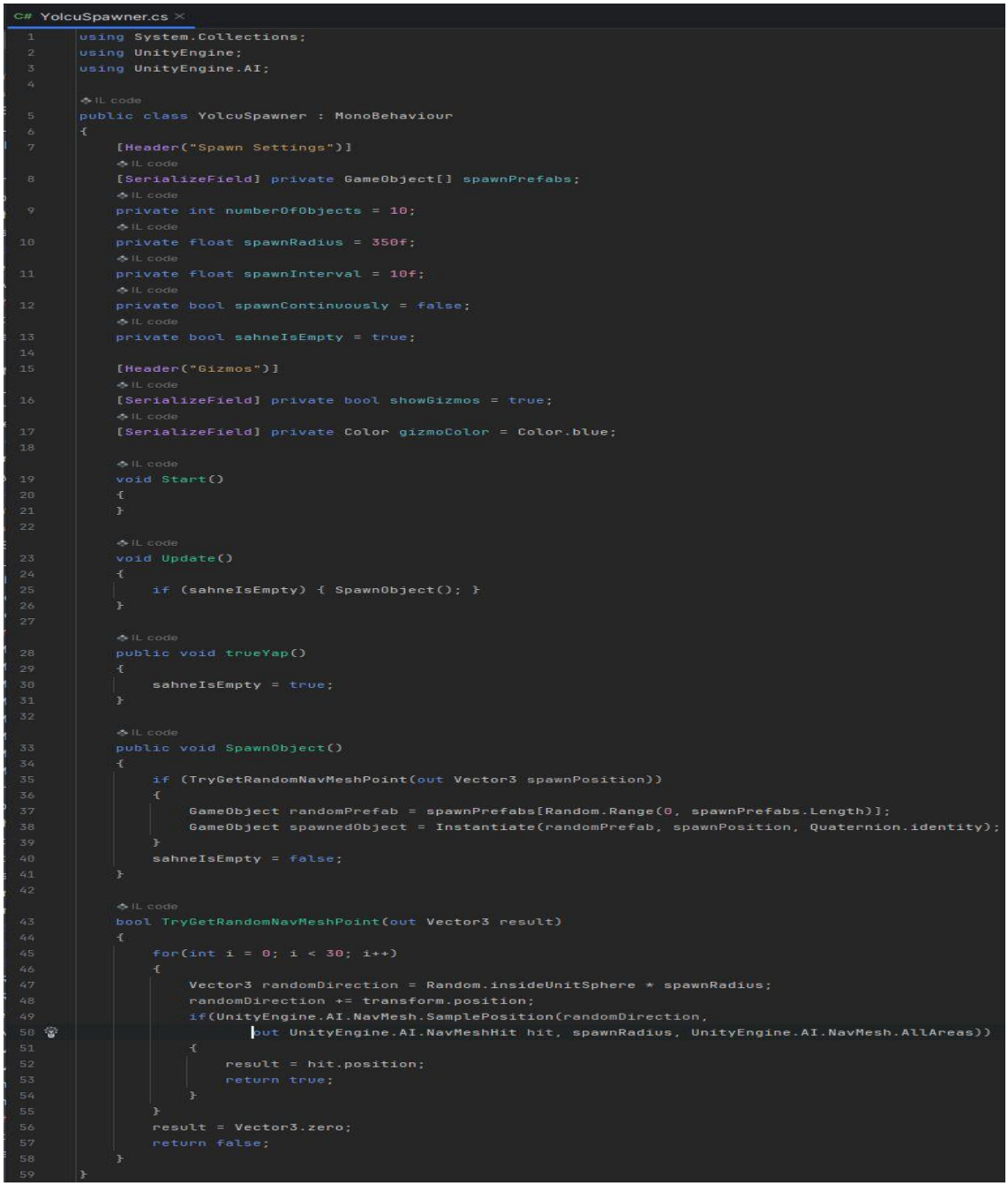
oluşturmaPrefabs dizisi, oluşturulacak yolcu prefab'larını içerir. GameObject türündeki bu array, oyunda oluşturulan tüm yolcu objelerinin türlerini belirler. numberOfObjects değişkeni,

sahneye kaç tane yolcu oluşturulacağını belirler. Bu, başlangıçta belirli bir sayıda yolcu oluşturulmasına olanak tanır. oluşturmaRadius ise oluşturma alanının yarıçapını belirler. Bu, yolcuların rastgele bir alanda oluşturulmasını sağlar. oluşturmaInterval yolcuların ne sıklıkla oluşturulacağını belirleyen aralıktır. Bu değeri artırarak ya da azaltarak, yolcuların daha hızlı ya da daha yavaş oluşturulmasi sağlanabilir. oluşturmaContinuously değişkeni, yolcuların sürekli olarak oluşturulup edilmeyeceğini belirler. Eğer bu değer true ise, oyun sürekli olarak yeni yolcular yaratacaktır. sahneIsEmpty değişkeni, sahnenin boş olup olmadığını kontrol eder. Yolcu oluşturma işlemi, sahne boşken yapılır. showGizmos ve gizmoColor Unity editor'ünde sahnede oluşturma alanının görselleştirilmesini sağlayan gizmos ile ilgili ayarlardır.

Start() metodunda, oluşturma işlemi için gerekli olan temel yapılandırmalar ve başlatmalar yapılmış olmalıdır. Ancak, kodun şu anki durumunda Start() metodu içerisine oluşturma işlemleri dahil edilmemiştir ve burada sadece temel ayarlamalar ve sahne elemanlarının başlatılması yapılır. Update() metodunda, eğer sahne boşsa (yani sahneIsEmpty değişkeni true ise), OluşturmaObject() metodunun çağrılması sağlanır. Bu, her güncellemeyle yeni yolcu oluşturulmasini sağlar. Ancak, oluşturma işleminin kontrolü sahneIsEmpty değişkeni üzerinden yapılır. Eğer sahnede halen bir yolcu var ise, yeni yolcu oluşturulmaz.

trueYap() metodu, sahneIsEmpty değişkenini true yaparak sahnenin boş olduğunu belirtir. Bu, yeni bir yolcu oluşturabilmek için gerekli olan şarttır. Sahne, yolcunun doğru bir şekilde yerleştirilip yok edilmesinden sonra yeniden boş olduğunda, bu metod çağrılarak yeni bir yolcu oluşturulur. OluşturmaObject() metodu, yolcunun oluşturulmasini sağlayan temel metodur. İlk olarak, TryGetRandomNavMeshPoint() metodu ile rastgele bir pozisyon belirlenir. Eğer geçerli bir pozisyon bulunursa, oluşturmaPrefabs dizisinden rastgele bir yolcu prefab'ı seçilir ve bu prefab seçilen pozisyonda oluşturularak oyunda aktif hale getirilir. Ardından, sahneIsEmpty değişkeni false yapılır, yani sahnede artık bir yolcu olduğu belirtilir. Bu işlem, yolcunun oluşturulmasının ardından yapılır.

TryGetRandomNavMeshPoint() metodu, yolcu için geçerli bir NavMesh pozisyonu bulmaya çalışır. Random.insideUnitSphere kullanılarak rastgele bir yön ve mesafe belirlenir. Bu rastgele yön, oluşturma alanının içinde olup olmadığı kontrol edilir. Eğer uygun bir pozisyon bulunursa, result olarak bu pozisyon döndürülür. Eğer geçerli bir pozisyon bulunmazsa, metod false döndürerek oluşturma işlemi gerçekleşmez. Bu, yolcunun NavMesh üzerinde uygun bir alanda oluşturulmasını garanti eder.



```
C# YolcuSpawner.cs
using System.Collections;
using UnityEngine;
using UnityEngine.AI;

IL code
public class YolcuSpawner : MonoBehaviour
{
    [Header("Spawn Settings")]
    IL code
    [SerializeField] private GameObject[] spawnPrefabs;
    IL code
    private int numberOfObjects = 10;
    IL code
    private float spawnRadius = 350f;
    IL code
    private float spawnInterval = 10f;
    IL code
    private bool spawnContinuously = false;
    IL code
    private bool sahneIsEmpty = true;

    [Header("Gizmos")]
    IL code
    [SerializeField] private bool showGizmos = true;
    IL code
    [SerializeField] private Color gizmoColor = Color.blue;

    IL code
    void Start()
    {
    }

    IL code
    void Update()
    {
        if (sahneIsEmpty) { SpawnObject(); }
    }

    IL code
    public void trueYap()
    {
        sahneIsEmpty = true;
    }

    IL code
    public void SpawnObject()
    {
        if (TryGetRandomNavMeshPoint(out Vector3 spawnPosition))
        {
            GameObject randomPrefab = spawnPrefabs[Random.Range(0, spawnPrefabs.Length)];
            GameObject spawnedObject = Instantiate(randomPrefab, spawnPosition, Quaternion.identity);
        }
        sahneIsEmpty = false;
    }

    IL code
    bool TryGetRandomNavMeshPoint(out Vector3 result)
    {
        for(int i = 0; i < 30; i++)
        {
            Vector3 randomDirection = Random.insideUnitSphere * spawnRadius;
            randomDirection += transform.position;
            if(UnityEngine.AI.NavMesh.SamplePosition(randomDirection,
                    out UnityEngine.AI.NavMeshHit hit, spawnRadius, UnityEngine.AI.NavMesh.AllAreas))
            {
                result = hit.position;
                return true;
            }
        }
        result = Vector3.zero;
        return false;
    }
}
```

***Şekil 5.2. Yolcu Oluşturucu Sınıfı Kodu***

Gizmos özellikleri, Unity'nin editöründe sahnede bir görselleştirme yapabilmek için kullanılır. showGizmos değişkeni true olduğunda, gizmos olarak bir oluşturma alanı görselleştirilir ve bu alanın sınırları gizmoColor ile belirlenir. Bu, geliştiricilerin oluşturma alanlarını daha iyi görselleştirerek düzeltmeler ve testler yapmalarını sağlar. NavMesh kullanılarak yolcuların doğru bir şekilde dünyada yerleşmesi ve taksiye binmeleri için yolcuların

oluşturulacağı alan düzgün bir şekilde hesaplanır. TryGetRandomNavMeshPoint() metodunda NavMesh.SamplePosition() ile uygun bir pozisyon belirlenir. Bu, yolcuların NavMesh üzerinde doğru bir şekilde hareket etmelerini sağlar.

Yolcu oluşturma işlemi, sahne boş olduğunda yapılır. Bu, bir yolcunun varış noktasına ulaşıp taksiden indikten sonra sahnenin yeniden boşalmasını ve yeni yolcuların oluşturulmasini sağlar. Bu işlem, performansı artırmak için önemlidir çünkü sahnede yalnızca aktif olan öğeler yer alır ve gereksiz objeler renderlanmaz. Sahne, yolcunun takside bindiği ve varış noktasına ulaştığı gibi belirli koşullar altında boşalır, böylece yeni yolcular oluşturulabilir.

### 5.1.3. Varış Noktası Oluşturucu Kodu

VarisNoktasiOluşturucuKod, oyundaki varış noktalarını oluşturmak için kullanılan bir bileşendir. Bu Kod, belirli bir alan içerisinde rastgele varış noktalarının yerleşmesini sağlar. Yolcu varış noktasına ulaştığında, bu varış noktası yok edilir ve yeni bir tane oluşturulması için sahne boş olduğu belirlenir. Bu işlem, yolcu ve varış noktası etkileşimlerinin düzgün çalışması için gereklidir.

oluşturmaPrefab değişkeni, oluşturulacak olan varış noktası prefab'ını temsil eder. Bu prefab, sahnede görünecek olan varış noktasının fiziksel temsilidir. numberOfObjects, sahneye oluşturulacak varış noktası sayısını belirler. Ancak, bu Kod'te şu anda sadece bir varış noktası oluşturulacağı için değeri 1 olarak ayarlanmıştır. oluşturmaRadius ise varış noktalarının oluşturulacağı alanın yarıçapını belirler. Bu değer, varış noktalarının rastgele yerleşebileceği bölgeyi kontrol eder.

instatiatedObject, oluşturulan varış noktası objesini temsil eder. Bu değişken, oluşturulan varış noktası objesini kontrol etmek için kullanılır. sahneIsEmpty değişkeni, sahnenin boş olup olmadığını kontrol eder. Bu, yeni bir varış noktası oluşturmak için gereklidir. Sahne boşsa, yeni bir varış noktası oluşturulur. Aksi takdirde, bir sonraki oluşturma işlemi yapılmaz.

showGizmos ve gizmoColor değişkenleri, Unity editöründe oluşturma alanının görselleştirilmesini sağlar. showGizmos true olduğunda, gizmos olarak oluşturma alanı gösterilir ve bu alanın sınırları gizmoColor ile belirlenir. Bu, geliştiricilerin oluşturma alanlarını daha iyi görselleştirerek düzeltmeler ve testler yapmalarını sağlar.

Start() metodunda, Kod'in çalışmaya başlamadan önce herhangi bir işlem yapılmaz. Kodda yorum satırlarına alınmış olan bölümler, başlangıçta varış noktalarının oluşturulmasini sağlayan kodları içermektedir. Ancak, şu anki Kod'te varış noktası başlangıçta oluşturulmaz.

Update() metodunda, herhangi bir işlem yapılmaz. Ancak, buraya daha sonra varış noktası oluşturulmaden önce belirli bir kontrol eklenebilir. Şu an için Update() metodu boş bırakılmıştır.

OluşturmaObject() metodu, yeni bir varış noktası oluşturmak için kullanılır. İlk olarak, TryGetRandomNavMeshPoint() metodunda, rastgele bir pozisyon bulunur. Eğer geçerli bir pozisyon bulunursa, oluşturmaPrefab dizisinden bir varış noktası prefab'ı seçilir ve bu prefab, rastgele bulunan pozisyonda oluşturulur. Burada, yeni varış noktası instantiation işlemi yapılır ve instatiatedObject değişkeni ile takip edilir. Bu metod, sahneye bir varış noktası yerleştirir ve bu işlem sahne boşken yapılır.

DestroyObject() metodu, daha önce oluşturulan varış noktasını yok etmek için kullanılır. Bu metod, varış noktası artık gereksiz olduğunda veya yolcu varış noktasına ulaştığında çağrılır. instatiatedObject değişkeni aracılığıyla varış noktası yok edilir ve sahnede artık varış noktası bulunmaz. Bu işlem, sahnenin temizlenmesini sağlar.

TryGetRandomNavMeshPoint() metodu, yolcu için geçerli bir NavMesh pozisyonu bulmaya çalışır. Random.insideUnitSphere kullanılarak rastgele bir yön ve mesafe belirlenir. Bu rastgele yön, oluşturma alanının içinde olup olmadığı kontrol edilir. Eğer uygun bir pozisyon bulunursa, result olarak bu pozisyon döndürülür. Eğer geçerli bir pozisyon bulunmazsa, metod false döndürerek oluşturma işlemi gerçekleşmez. Bu, varış noktasının NavMesh üzerinde uygun bir alanda oluşturulmasını garanti eder.

Gizmos özellikleri, Unity'nin editöründe sahnede bir görselleştirme yapabilmek için kullanılır. showGizmos değişkeni true olduğunda, gizmos olarak bir oluşturma alanı görselleştirilir ve bu alanın sınırları gizmoColor ile belirlenir. Bu, geliştiricilerin oluşturma alanlarını daha iyi görselleştirerek düzeltmeler ve testler yapmalarını sağlar. NavMesh kullanılarak varış noktalarının doğru bir şekilde yerleşmesi sağlanır. Bu, oluşturulan varış noktasının taksiye binmek için doğru bir pozisyonda yer almasını ve yolcunun doğru yere yönlendirilmesini sağlar.

C# VarisNoktasiSpawnerScript.cs

```
using System.Collections;
using UnityEngine;

IL code
public class VarisNoktasiSpawnerScript : MonoBehaviour
{
    [Header("Spawn Settings")]
    IL code
    [SerializeField] private GameObject spawnPrefab;
    IL code
    [SerializeField] private int numberOfObjects = 1;
    IL code
    [SerializeField] private float spawnRadius = 350f;

    IL code
    private GameObject instatiatedObject;
    IL code
    private bool sahneIsEmpty = true;
    [Header("Gizmos")]
    IL code
    [SerializeField] private bool showGizmos = true;
    {
        if(TryGetRandomNavMeshPoint(out Vector3 spawnPosition))
        {
            instatiatedObject = Instantiate(spawnPrefab, spawnPosition, Quaternion.identity);
        }
    }

    IL code
    public void DestroyObject()
    {
        Destroy(instatiatedObject);
    }

    IL code
    bool TryGetRandomNavMeshPoint(out Vector3 result)
    {
        for(int i = 0; i < 30; i++)
        {
            Vector3 randomDirection = Random.insideUnitSphere * spawnRadius;
            randomDirection += transform.position;
            if(UnityEngine.AI.NavMesh.SamplePosition(randomDirection, out UnityEngine.AI.NavMeshHit hit,
                   spawnRadius, UnityEngine.AI.NavMesh.AllAreas))
            {
                result = hit.position;
                return true;
            }
        }
        result = Vector3.zero;
        return false;
    }
}
```

*Şekil 5.3. Varış Noktası Oluşturucu Sınıfı Kodu*

Sahne boşken oluşturulan varış noktaları, yolcu varış noktasına ulaştığında yok edilir. Bu işlem, her bir varış noktasının sadece gerektiği zaman var olmasını sağlar. Yani, sahnede gereksiz objeler bulunmaz ve performans kaybı yaşanmaz. Yeni bir varış noktası oluşturulduğunda, sahne temizlenmeden önce eski varış noktası yok edilir.

### 5.1.4. Direksiyon Dönüş Animasyon Kodu

SteeringTry Kod'u, Unity oyun motorunda bir karakterin veya aracın yönlendirilmesi için kullanılan bir sistemdir. Bu Kod, belirli tuşlara basıldığında animasyonları kontrol eder ve yön değiştiren bir obje (örneğin, bir araç veya karakter) için derece bilgisi ile yön hareketlerini düzenler. Bu Kod, oyuncunun A ve D tuşlarına basarak objenin yönünü değiştirmesine olanak tanır ve animasyonları kontrol eder.

C# SteeringTry.cs

```
using System.Collections;
using UnityEngine;
IL code
public class SteeringTry : MonoBehaviour
{
    IL code
    public Animator animator;
    IL code
    public int degree;
    IL code
    void Start()
    {
        animator = GetComponent<Animator>();
        degree = animator.GetInteger("Degree");
        StartCoroutine(steering());
    }
    IL code
    IEnumerator steering()
    {
        while (true)
        {
            if (Input.GetKey(KeyCode.D)||Input.GetKeyDown(KeyCode.D))
            {
                animator.SetBool("Dpush", true);

                degree += 10;
            }
            if (Input.GetKey(KeyCode.A)||Input.GetKeyDown(KeyCode.A))
            {
                animator.SetBool("Apush", true);
                degree -= 10;
            }
            if (degree > 0)
            {
                degree -= 20;
            }
            else
            {
                degree += 20;
            }
            animator.SetInteger("Degree", degree);
            yield return new WaitForSeconds(0.1f);
            animator.SetBool("Dpush", false);
            animator.SetBool("Apush", false);
        }
```

*Şekil 5.4. Direksiyon Dönüş Animasyon Kodu*

animator değişkeni, objenin animasyonunu kontrol etmek için kullanılan Animator bileşenini temsil eder. Animator bileşeni, animasyon geçişlerini yönetir ve animasyonları başlatmak için çeşitli parametreler alır. Bu Kod'in içinde, animasyonları tetiklemek için kullanılan Dpush ve Apush bool parametreleri vardır. degree değişkeni ise objenin yönünü belirtir ve bu değer, yön değişiklikleriyle güncellenir.

Start() metodu, Kod'in başlatıldığı anda ilk çalıştırılan metoddur. Burada, animator bileşeni, objenin animator'ını almak için kullanılır. Ayrıca, animatörden Degree parametresi alınarak degree değişkenine atanır. Bu, animasyonun çalışması için gerekli olan başlangıç derecesini sağlar. Ardından, steering() isimli coroutine başlatılır. Bu coroutine, sürekli olarak yön değiştirme işlemlerini kontrol eder ve animasyonları tetikler.

steering() metodu, bir coroutine (arka planda çalışan bir işlev) olarak tanımlanmıştır. Bu metot, sürekli olarak çalışarak oyuncunun tuşlara basıp basmadığını kontrol eder ve buna göre animasyonları ve objenin yönünü değiştirir. Coroutine, while (true) döngüsü içinde sürekli çalışır. Bu döngüde, her tuş basımı kontrol edilir ve tuşa basıldığında animasyonlar tetiklenir.

if (Input.GetKey(KeyCode.D) || Input.GetKeyDown(KeyCode.D)) satırı, oyuncunun D tuşuna basıp basmadığını kontrol eder. Eğer D tuşuna basılırsa, animator. SetBool("Dpush", true); kodu ile animasyonda Dpush bool parametresi true olarak ayarlanır ve degree değeri 10 artırılır. Bu, oyuncunun sağa yönelmesini simüle eder.

if (Input.GetKey(KeyCode.A) || Input.GetKeyDown(KeyCode.A)) satırı, oyuncunun A tuşuna basıp basmadığını kontrol eder. Eğer A tuşuna basılırsa, animator. SetBool("Apush", true); kodu ile animasyonda Apush bool parametresi true olarak ayarlanır ve degree değeri 10 azaltılır. Bu, oyuncunun sola yönelmesini simüle eder.

Sonrasında, degree değişkeni üzerinde bir kontrol yapılır. Eğer degree pozitifse, yani sağa doğru bir hareket varsa, degree değeri 20 azaltılır. Eğer degree negatifse, yani sola doğru bir hareket varsa, degree değeri 20 artırılır. Bu, objenin yönünün zamanla yavaşça düzeltmesini sağlar. Bu işlem, objenin belirli bir açıya gelmesini sağlar ve animasyonların akıcı bir şekilde geçiş yapmasını sağlar.

En son olarak, animator. SetInteger("Degree", degree); kodu ile degree değeri animatöre gönderilir. Bu, animatörün objenin yönünü doğru şekilde animasyonla göstermesini sağlar. Ardından, yield return new WaitForSeconds(0.1f); kodu ile coroutine 0.1 saniye bekler. Bu süre, animasyonun düzgün bir şekilde çalışması için gerekli bir bekleme süresidir. Son olarak, animator. SetBool("Dpush", false); ve animator. SetBool("Apush", false); kodları ile tuş bırakıldığında animasyonlar sıfırlanır ve objenin animasyon durumu eski haline döner.

Bu Kod, A ve D tuşları ile karakter veya aracın yönünü değiştiren ve animasyonlarını kontrol eden bir sistemdir. Animator bileşeni ile animasyonları yönetir ve degree parametresi ile objenin yönünü ayarlar. Kod, sürekli olarak çalışarak tuşlara basılması ile objenin yönünü günceller ve animasyonların tetiklenmesini sağlar.

### 5.1.5. Mini Haritada En Kısa Yoldan Varış Noktasına Çizgi Çizme Kodu

LineRendererKod, bir oyuncu ile varış noktası arasındaki yolu görsel olarak çizmek için kullanılan bir Unity Kod'idir. Bu Kod, LineRenderer bileşenini kullanarak, oyuncunun başlangıç noktasından varış noktasına kadar bir yol çizer. Bu yol, NavMesh kullanılarak hesaplanır ve çizilir. Kod, her güncellemeyle yolun konumlarını günceller ve çizgiyi çizer.

player değişkeni, oyuncu karakterini temsil eden GameObject'i tutar. Bu, yolun başlangıç noktasını belirlemek için kullanılır. path değişkeni ise LineRenderer bileşenini temsil eder. Bu bileşen, iki nokta arasındaki yolu çizmek için kullanılır. startingPoint ve endingPoint değişkenleri sırasıyla yolun başlangıç ve bitiş noktalarını temsil eder. Başlangıç noktası, oyuncunun pozisyonunu; bitiş noktası ise varış noktasını belirtir. varisNoktasi ise varış noktasını temsil eden GameObject'i tutar. Bu, yolun hedef noktasını gösterir. Path değişkeni, NavMeshPath türünde olup, oyuncudan varış noktasına giden yolu temsil eder. Bu yol, NavMesh sistemi ile hesaplanır.

PathUpdateSpeed değişkeni, yolun güncellenme hızını kontrol eder. Yani, çizginin her ne kadar sıklıkla yeniden çizileceğini belirler. PathHeightOffset ise çizilen yolun yükseklik ayarını yapar. Bu, yol çizgilerinin yerden ne kadar yukarıda olacağını belirler. mask değişkeni, belirli bir NavMesh alanında yol hesaplamak için kullanılan bir maske değerini tutar.

Start() metodu, Kod'in başlatıldığı andaki ilk aşamadır. Burada, mask değeri, "LineRenderer" adıyla NavMesh alanı tanımlanarak ayarlanır. player değişkeni, sahnede "Player" tag'ine sahip objeyi bulur. varisNoktasi değişkeni, bu Kod'in bağlı olduğu objeyi alır ve varış noktası olarak kabul edilir. path değişkeni, LineRenderer bileşenini alır ve çizgi oluşturulmak üzere yapılandırılır. positionCount değeri 2 olarak ayarlanır, yani başlangıç ve bitiş noktasını tutacak kadar pozisyon belirlenir. startingPoint ve endingPoint, sırasıyla oyuncunun ve varış noktasının pozisyonları olarak ayarlanır. Son olarak, Path bir NavMeshPath nesnesi olarak başlatılır ve DrawPath() coroutine'i başlatılır.

C# LineRendererScript.cs

```
using System;
using System.Collections;
using System.IO;
using System.Security.Cryptography.X509Certificates;
using UnityEngine;
using UnityEngine.AI;

IL code
public class LineRendererScript : MonoBehaviour
{
    IL code
    private GameObject player;

    IL code
    private LineRenderer path;

    IL code
    private Vector3 startingPoint;
    IL code
    private Vector3 endingPoint;

    IL code
    private GameObject varisNoktasi;

    IL code
    private NavMeshPath Path;

    IL code
    private float PathUpdateSpeed = 0.3f;

    IL code
    private float PathHeightOffset = 5f;

    IL code
    private int mask;



    // Start is called once before the first execution of Update after the MonoBehaviour is created
    IL code
    void Start()
    {
        mask= 1 << NavMesh.GetAreaFromName("LineRenderer");
        player = GameObject.FindGameObjectWithTag("Player");

        varisNoktasi = gameObject;


        path = varisNoktasi.GetComponent<LineRenderer>();
        path.positionCount = 2;
        //ine.enabled = false;

        startingPoint = player.transform.position;
        endingPoint=varisNoktasi.transform.position;

        Path = new NavMeshPath();

        StartCoroutine(DrawPath());
    }

    // Update is called once per frame



    IL code
    IEnumerator DrawPath()
    {
        Debug.Log("Draw Path Coroutinei Calisti");
        while (true)
        {
            Debug.Log("while dondu");

            startingPoint = player.transform.localPosition;

            NavMesh.CalculatePath(startingPoint, endingPoint, mask, Path);
            path.positionCount = Path.corners.Length;
            for (int i = 0; i < Path.corners.Length; i++)
            {
                path.SetPosition(i, Path.corners[i] + Vector3.up * PathHeightOffset);
            }


            yield return new WaitForSeconds(PathUpdateSpeed);
```

***Şekil 5.5. Mini Harita En Kısa Yol Kodu***

DrawPath() metodu, bir coroutine (arka planda çalışan bir işlev) olarak tanımlanmış ve sürekli çalışacak şekilde programlanmıştır. Bu metod, sürekli olarak yolun çizilmesini sağlar. while (true) döngüsü, sürekli olarak yolun çizilmesini sağlar. İlk olarak, oyuncunun mevcut pozisyonu startingPoint olarak alınır. Daha sonra, NavMesh.CalculatePath() fonksiyonu, oyuncunun mevcut pozisyonu ile varış noktasının arasındaki yolu hesaplar. mask kullanılarak bu yol, belirli bir alan içinde hesaplanır. path.positionCount değeri, yolun köşe noktalarının sayısını belirtir ve Path.corners.Length ile ayarlanır. Bu köşe noktaları, çizgi üzerinde gösterilen yolun her bir adımını temsil eder. path. SetPosition(i, Path.corners[i] + Vector3.up * PathHeightOffset) kodu ile, her bir köşe noktasına çizgi çizilir ve bu noktalar, yukarıya doğru PathHeightOffset kadar bir ofset ile ayarlanır. Böylece çizilen yolun yüksekliği düzenlenmiş olur. yield return new WaitForSeconds(PathUpdateSpeed) ile, belirli bir süre beklenir (yolun güncellenme hızını belirleyen değişken), ardından animasyonlar sıfırlanır ve yol yeniden çizilmeye başlar.

TryGetRandomNavMeshPoint() metodunda ise, belirli bir rastgele yön ve mesafe belirlenir. Bu, yolun çizileceği alanın içinde, geçerli bir NavMesh pozisyonu olup olmadığını kontrol eder. Eğer geçerli bir pozisyon bulunursa, result değişkeni bu pozisyonu alır ve yolun oluşturulabilmesi için true döndürülür. Eğer geçerli bir pozisyon bulunmazsa, result sıfırlanır ve false döndürülür.

LineRendererKod ile oyuncu ile varış noktası arasındaki yolu görsel olarak çizebilirsiniz. Bu Kod, oyuncunun hareketine göre dinamik olarak yol oluşturur ve her adımda çizilen yolu günceller. NavMesh ile hesaplanan yol, LineRenderer kullanılarak sahnede görünür hale getirilir. Bu, oyuncuya varış noktasına nasıl gideceği hakkında görsel bir rehber sağlar. Ayrıca, PathUpdateSpeed ile yolun güncellenme hızını kontrol edebilirsiniz, böylece yol çizgisi daha yavaş ya da hızlı bir şekilde yeniden çizilebilir. Bu Kod, özellikle yol gösterme, rotalama ve görselleştirme için kullanışlıdır.

# 6. SONUÇ

Bu tezde, Taxi Simulator adlı oyun projesinin geliştirilmesinde kullanılan ana mekanikler, oynanış özellikleri, teknik detaylar ve kullanılan oyun motorunun özellikleri detaylı bir şekilde incelenmiştir. Oyunun tasarımı ve geliştirilmesinde kullanılan araçlar ve yöntemler, oyunculara keyifli ve etkileşimli bir deneyim sunmayı hedeflemektedir. Bu süreçte, Unity oyun motoru ve C# programlama dili kullanılarak oyun dinamikleri oluşturulmuş ve çeşitli teknikler ile oyun performansı optimize edilmiştir.

Oyunun ana mekaniklerinden biri olan yolcu alma ve taşıma mekanizması, oyuncuya etkileşimli bir görev sunarken, zamana dayalı teslimat, GPS kullanımı ve yolcu rotası gibi özellikler, oyuncuya yolculuk yaparken gerçekçi ve dinamik bir deneyim sağlamaktadır. Bu mekanikler, kamera açıları ve yakıt durumu gibi unsurlarla desteklenerek, oyuncunun oyun içindeki etkileşimini daha derin hale getirmektedir.

Oynanış özellikleri kısmında, oyuncuya sürekli değişen bir şehir ortamı, dinamik trafik yoğunluğu, gece-gündüz döngüsü ve oyuncunun yaptığı her hareketin şehre olan etkisi sunulmuştur. Ayrıca, ekonomi sistemi ile oyunculara taksi işletmelerinde kazanma ve harcama yönetimini de deneyimleme imkanı tanınmıştır. Bu özellikler, oyunculara sadece bir oyun deneyimi değil, aynı zamanda stratejik düşünmeyi gerektiren bir ortam sunar.

Teknik detaylar bölümünde, kullanılan Unity oyun motoru ve NavMesh teknolojisi, yol hesaplamaları ve oyuncu etkileşimleri için önemli bir rol oynamıştır. LineRendererKod ve YolcuOluşturucu gibi Kod'ler, oyun dünyasındaki etkileşimleri yönlendirerek oyunun dinamik yapısını oluşturmuştur. Bu teknolojiler ve Kod'ler, özellikle yolculuk süreçlerinde oyuncunun yönlendirilmesini ve yolculukların izlenmesini sağlamak için kullanılmıştır.

Performans ve optimizasyon açısından, occlusion culling, NavMesh ve diğer optimizasyon teknikleri kullanılarak oyun dünyasında gereksiz kaynak tüketiminin önüne geçilmiştir. Bu teknikler, özellikle büyük ve karmaşık şehir haritası içinde daha verimli bir performans sağlamıştır. Ayrıca, Unity'nin optimizasyon özellikleri ve NavMesh pathfinding kullanılarak, yolculuklar sırasında herhangi bir takılma veya performans düşüşü yaşanmamıştır.

Sonuç olarak, bu oyun geliştirme süreci, oyuncuya sürükleyici, etkileşimli ve sürekli değişen bir deneyim sunmayı başarmıştır. Geliştirilen mekanikler ve teknik özellikler, oyunun amacına ulaşmasını sağlamış ve oyunculara gerçekçi bir taksi sürüş simülasyonu deneyimi sunmuştur. Taxi Simulator, oyun tasarımı ve teknik açıdan başarılı bir proje olarak, gelecekteki

benzer projeler için de bir referans noktası oluşturabilir. Bu oyun, sadece bir eğlence aracı değil, aynı zamanda strateji, planlama ve zaman yönetimi gibi becerileri geliştiren bir deneyimdir.

# 7. KAYNAKLAR


1. https://docs.unity.com/
2. https://docs.unity.com/scriptReference/
3. https://learn.unity.com/
4. https://assetstore.unity.com/
5. https://stackoverflow.com/questions/tagged/unity3d
6. http://www.gamasutra.com/
7. https://en.wikipedia.org/wiki/Unity_%28game_engine%29
8. https://en.wikipedia.org/wiki/Unity_Technologies
9. https://medium.com/@wota_mmorpg/unity-development-history-and-the-influence-of-this-game-engine-on-the-game-development-36dc7a7a3b9d
10. https://unity.com/products/unity-analytics
11. https://unity.com/products/unity-ads
12. https://unity.com/solutions/multiplayer
13. https://unity.com/solutions/gaming-services
14. https://en.wikipedia.org/wiki/Unity_Version_Control
15. https://en.wikipedia.org/wiki/Unity_%28user_interface%29
16. https://en.wikipedia.org/wiki/Game_engine

# 8. ÖZGEÇMİŞ

**Mertcan YAŞAR**

**Kişisel Bilgiler**

**Adı Soyadı:** Mertcan Yaşar

**Doğum Yeri:** Magosa/K.K.T.C.

**Doğum Tarihi:** 04/09/2000

**Askerlik Durumu:** Tecilli (31.12.2027)

**İletişim Bilgileri**

**Adres:** Çağdaşkent Mh. 93095 Sk. No:2/1 TOROSLAR/MERSİN

**Telefon:** +90 (531) 211 02 67

**E-Posta:** canyasar924@gmail.com

**Eğitim Bilgileri**

**Lise:** Mahmut Arslan Anadolu Lisesi (2014-2017) Altın Kupa Anadolu Lisesi (2017-2018)

**Ön Lisans:** ANADOLU ÜNİVERSİTESİ Web Tasarımı Ve Kodlama (2019-2021)

**Lisans:** FIRAT ÜNİVERSİTESİ Mühendislik Fakültesi Bilgisayar Mühendsiliği (2021-)

**Yabancı Dil**

**İngilizce:** Çok İyi

**Yetkinlikler**

**Bilgisayar:** HTML, CSS, JavaKod, C++, C#, Java, SQL, Unreal Engine, Unity, Oyun Geliştime, Oyun Tasarımı, Blender

**Ek Bilgiler**

**Sürücü Belgesi:** B Sınıfı

## TECRÜBE

2023: Kardelen Yazılım (Staj – 20 İş Günü)