UYGULAMALI BİLİMLER FAKÜLTESİ

BİLİŞİM SİSTEMLERİ VE TEKNOLOJİLERİ

PLATFORM MOBİL OYUN PROJESİ

LİSANS BİTİRME PROJESİ

MUSTAFA TAŞAN

tarafından

LİSANS

derecesi şartını sağlamak için hazırlanmıştır.

Mayıs, 2024

Bölüm: Bilişim Sistemleri ve Teknolojileri

RUHANİ KAÇIŞ İSİMLİ MOBİL OYUN PROJESİ

MUSTAFA TAŞAN

tarafından

OKAN ÜNİVERSİTESİ

Bilişim Sistemleri ve Teknolojileri Bölümü

LİSANS

derecesi şartını sağlamak için sunulmuştur.

Onaylayan:

Dr.Öğr.Üyesi Nurşen Topçubaşı

Danışman

Mayıs, 2024

Bölümü: Bilişim Sistemleri ve Teknolojileri

ÖZET

Bu tez, mobil oyun projesini detaylı bir şekilde ele almayı ve tasarım, geliştirme ve oynanabilirlik süreçlerini incelemeyi hedeflemektedir. Oyun, oyuncuların heyecan verici bir kaçış deneyimi yaşayabileceği, stratejik becerilerini kullanarak ilerleme kaydedebileceği bir platform sunmaktadır.

Oyunun temel amacı, oyuncuların elmaları toplayarak düşmanları yenerek en yüksek skoru elde etmelerini sağlamaktadır. Oyuncular, ana karakterin kontrollerini sağlayarak oyun dünyasında ilerlerken strateji ve reflekslerini test etmektedirler. Mobil platformlarda oynanabilen ve karakter odaklı bir oynanış sunan bir oyundur.

Tezin başlangıç bölümünde, oyunun genel tasarım süreci ele alınmaktadır. Tasarım aşamasında, oyun için gerekli olan grafik ve ses unsurları belirlenmiş, sahne tasarımı oluşturulmuş ve ana karakterin kontrolleri ve saldırı mekanikleri tasarlanmıştır. Ayrıca, ana karakterin sağlık durumunu gösteren kalp şeklindeki can barı da bu aşamada entegre edilmiştir. Daha sonra, oyunun geliştirme süreci incelenmektedir. Oyunun Unity oyun motoru üzerinde nasıl geliştirildiği, kullanılan programlama dili ve araçlar, karşılaşılan zorluklar ve çözümler detaylı bir şekilde ele alınmaktadır. Oyunun farklı seviyeleri ve mekanikleri nasıl entegre edildiği üzerinde durulmaktadır.

Bu tez, mobil oyun projesinin tasarım ve geliştirme süreçlerini detaylı olarak ele alarak oyun geliştirme alanında yeni bir katkı sağlamayı amaçlamaktadır. Oyunun teknik ve yaratıcı yönleri üzerinde yapılan analizler, mobil oyun geliştirme sürecini anlamak ve gelecekteki projeler için yol gösterici olmak amacıyla sunulmaktadır.

Anahtar Kelimeler: Mobil Oyun, Dijital Oyun, Geleceğin Mobil Oyunları.

İÇİNDEKİLER

TABLO LİSTESİ

ŞEKİL LİSTESİ

1.GİRİŞ

Oyun, doğal olarak ortaya çıkan ve amacı olmayan, insanları eğlendiren serbest bir aktivitedir. Duyusal, sinirsel ve zihinsel düzeyde gerçekleşir, tekrarlanabilir ve belirli zaman ve mekânlarda gerçekleşir. Oyunlar, başlangıçtan ustalığa kadar çeşitli gelişim aşamaları gösterir ve katılımcılarına keyifli bir deneyim sunar, bu da onları popüler hale getirir (Akın, 2008).

1947 yılında, 2. Dünya Savaşı'nın etkileriyle ilham alınarak tasarlanan Cathode Ray Tube Amusement Device (Katod Işın Tüplü Eğlence Cihazı), dijital bir oyun olmamakla birlikte, etkileşimli elektronik oyunların ilk örneğidir. Bilgisayar oyunlarının kökenleri 1950'lerde ortaya çıkmaya başlamıştır. 1958'de geliştirilen Tennis for Two (İki Kişilik Tenis), ilk bilgisayar oyunu olarak kabul edilir. Spacewar, ticari ve toplumsal etki yaratan ilk bilgisayar oyunudur. 1960'ların ikinci yarısından itibaren üniversitelerdeki çalışmalar, bilgisayar oyunlarının yaygınlaşmasının ilk adımlarını atmıştır. İlk oyun konsolu olan "Brown Box", 1968'de piyasaya sürüldü ve bilgisayar oyunlarının sokaktaki insanlar tarafından da ucuz bir şekilde oynanabilir hale gelmesini sağlamıştır (Kızılkaya, 2010).

Oyun programlama, dijital oyunlarının yazılım geliştirme sürecini ifade eden ve oyun geliştirmenin önemli bir parçası olan bir alandır. Bu alanda çalışmak için, yazılım mühendisliği ve bilgisayar programlama konularında derin bir uzmanlık gereklidir. Ayrıca, simülasyon, bilgisayar grafikleri, yapay zeka, fizik, ses programlama ve giriş gibi alanlarda da uzmanlaşma gerekebilir. Özellikle çok oyunculu oyunlar için ağ programlama bilgisine sahip olmak önemlidir; bu, oyunun performansını etkileyen ve oyuncular tarafından "netcode" olarak adlandırılan kod geliştirme süreçlerini içerir. Bazı oyun türlerinde, özellikle dövüş oyunları gibi, ileri düzey ağ programlama becerileri gerekebilir çünkü netcode ve özellikleri (örneğin, gecikme) oyun deneyiminin kalitesini belirleyen önemli faktörlerdir. Büyük çaplı

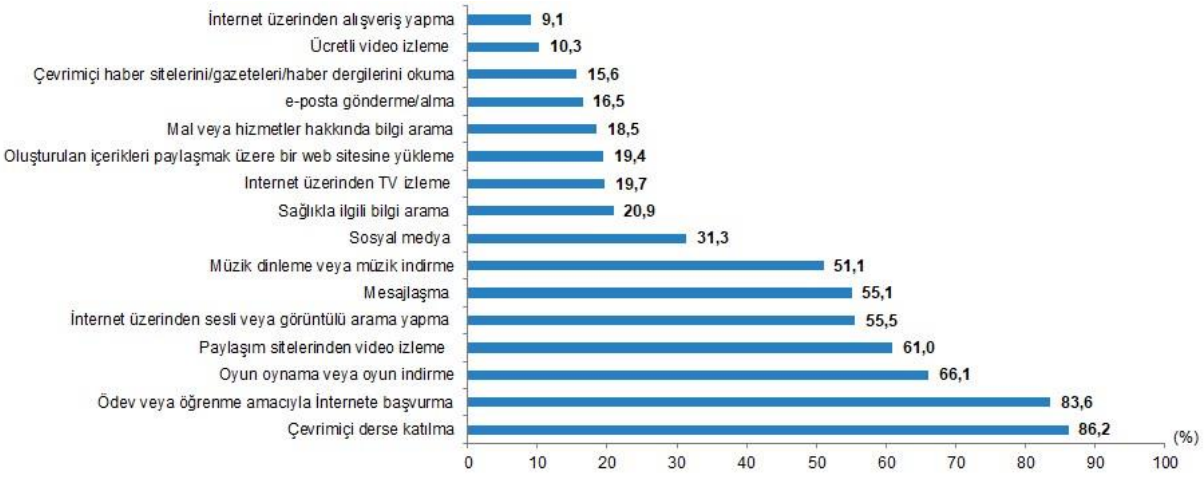
çok oyunculu online oyunlar için ise veri tabanı programlama ve ileri düzey ağ bilgisine daha fazla ihtiyaç duyulabilir.

2020 yılında COVID-19 salgınının etkisiyle dijital oyun sektörü %9,3'lük bir artışla 159 milyar dolarlık bir değere ulaşarak önemli bir büyüme kaydetti. Rapora göre, dijital oyun pazarının 2026 yılına kadar 295 milyar doları aşması bekleniyor. Salgın döneminde en büyük artış mobil oyunlarla gerçekleşti; mobil oyunlar 2020'de 77,2 milyar dolarlık gelir elde ederek dijital oyun pazarında önemli bir yer edindi. Bu büyüme, teknolojideki ilerlemeler ve kullanıcı dostu arayüzler gibi faktörlerle desteklenmekte olup, gelecekte mobil oyunların yanı sıra artan sanal gerçeklik ve artırılmış gerçeklik teknolojilerinin de büyümeye katkı sağlaması beklenmektedir (BTK, 2021).

Ülkemizde oyun sektörü, 2020 yılında pandeminin de etkisiyle mobil oyun alanında özellikle hızlı bir büyüme yaşadı. 2019 yılına göre %29'luk bir büyümeyle rekor seviyelere ulaşan mobil oyun sektörü, Türk oyunlarının dünya çapında ilgi gördüğüne işaret ediyor; Türk yapımı oyunlar dünya ortalamasının iki katı indirilme sayısına ulaştı. Türkiye'de oyun geliştirici şirketlerin sayısı da 239'a yükseldi. Rapora göre, Türkiye'deki yetişkinlerin %79'u mobil oyun oynuyor; bu oranın %81,7'si kadın ve %76,5'i erkeklerden oluşuyor. Evde geçirilen zamanın artmasıyla birlikte insanlar günlük ortalama 4 saatten fazla oyun oynamaya başladılar. Bu veriler, Türkiye'deki oyun sektörünün dinamik büyümesini ve mobil oyunların popülerliğini yansıtmaktadır (BTK, 2021).

6-15 yaş arası düzenli internet kullanan çocukların internet üzerinden yaptıkları faaliyetler incelendiğinde, çevrimiçi derslere katılma %86,2 ile en yaygın faaliyet olarak belirlendi. İkinci sırada %83,6 ile ödev veya öğrenme amacıyla internet kullanımı yer aldı. Bu yaş grubundaki çocukların %66,1'i oyun oynamak veya oyun indirmek için interneti tercih ederken, %61,0'u paylaşım sitelerinde video izlemeyi tercih etti. İnternet üzerinden sesli veya görüntülü arama yapma oranı %55,5 olarak kaydedildi. Çocukların en az yaptığı internet

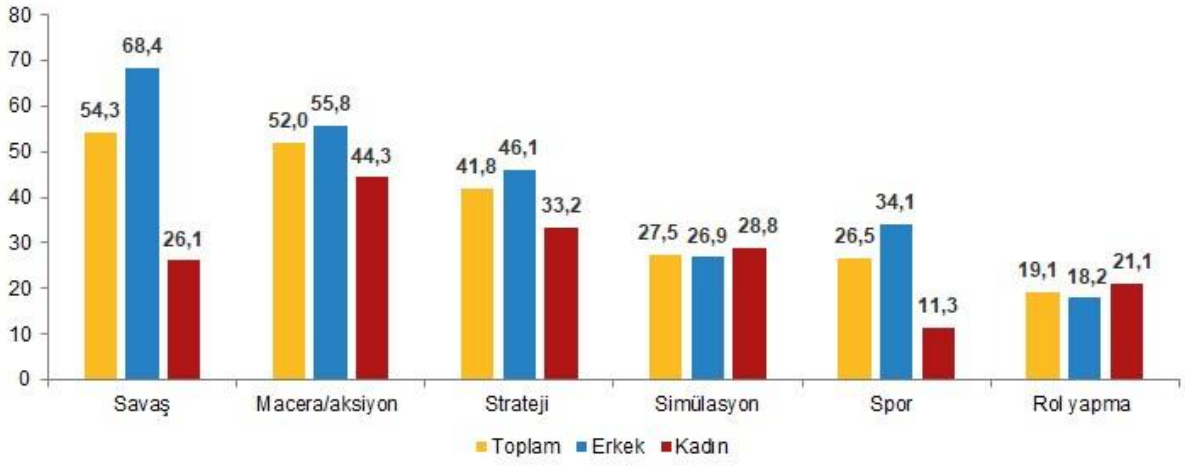
faaliyeti ise %9,1 ile internet üzerinden alışveriş yapmak oldu. Bu veriler, çocukların interneti öğrenme ve eğitim amacıyla yaygın bir şekilde kullandığını, oyun ve video izleme gibi eğlence amaçlı faaliyetlerin de önemli olduğunu göstermektedir (TÜİK, 2021).



Şekil 1 Düzenli İnternet kullanan çocukların İnternet kullanım amaçları (TÜİK, 2021).

Çocukların dijital oyun oynama alışkanlıkları yaş ve cinsiyet bazında incelendiğinde ilginç bulgular ortaya çıkmaktadır. Toplamda %36,0'lık bir oranla dijital oyun oynayan çocukların içinde, 6-10 yaş grubundaki çocukların %32,7'si ve 11-15 yaş grubundaki çocukların %39,4'ü dijital oyunlarla vakit geçirmektedir.. Cinsiyet bazında ise, 6-15 yaş arası erkek çocukların %46,1'i dijital oyun oynarken, kız çocuklarının bu oranı %25,4 olarak kaydedildi. Yaş gruplarına göre incelendiğinde, 6-10 yaş arası erkek çocukların %38,7'si oyun oynarken, kız çocuklarının bu oranı %26,4 oldu. 11-15 yaş arası erkek çocuklarda ise dijital oyun oynama oranı %53,7 iken kız çocuklarının oranı %24,4 olarak belirlendi. Bu veriler, genel olarak erkek çocukların dijital oyunlara daha fazla ilgi gösterdiğini ve yaşla birlikte bu ilginin arttığını göstermektedir (TÜİK, 2021).

Düzenli dijital oyun oynayan 6-15 yaş grubundaki çocukların oyun tercihleri incelendiğinde, %54,3'ü savaş oyunlarını tercih ettiğini belirtmiştir. Bu oranı %52,0 ile macera/aksiyon oyunları, %41,8 ile strateji oyunları, %27,5 ile simülasyon oyunları ve %26,5 ile spor oyunları takip etmektedir. Rol yapma oyunları ise %19,1 ile en az tercih edilen kategori olmuştur. Cinsiyet bazında bakıldığında, 6-15 yaş arası erkek çocukların en çok tercih ettiği oyun türü %68,4 ile savaş oyunları iken kız çocuklarında bu oran %44,3 ile macera/aksiyon oyunlarıdır. Yaş gruplarına göre incelendiğinde ise, 6-10 yaş grubunda en çok oynanan dijital oyun türü %49,6 ile macera/aksiyon iken 11-15 yaş grubunda bu oran %66,3 ile savaş oyunları olarak belirlenmiştir. Bu veriler, çocukların oyun tercihlerinin cinsiyet ve yaşa bağlı olarak farklılık gösterdiğini ve özellikle savaş ve macera/aksiyon türündeki oyunların popüler olduğunu göstermektedir (TÜİK, 2021).



Şekil 2 Düzenli dijital oyun oynayan çocukların cinsiyetine göre oynadıkları oyun türleri (%) (TÜİK, 2021).

Tablo 1'de belirtilen mobil oyun tercihini etkileyen bağımsız değişkenler ve bu değişkenlere bağlı alt düzeyler, araştırmanın bağımlı değişkeni olan bireylerin mobil oyun tercihlerini anlamak için kritik öneme sahiptir. Bu bağımsız değişkenler arasında oyun tasarımı, kullanıcı arayüzü, karakterler, oyun oynama şekli, internet bağlantısı, oyuncu sayısı, şarj tüketimi, oyun için verilen değerlendirme ve yorumlar, oyuncu adına paylaşımda bulunma, kurulum dosyası

büyüklüğü, işgal ettiği hafıza, gereksinim duyduğu sistem, oyun içerisinde reklam gösterimi, yüklenme süresi ve diğer kişilerce indirilme gibi faktörler yer almaktadır.

Diğer bağımsız değişkenler de benzer şekilde literatürden ve önceki çalışmalardan elde edilmiştir. Bu değişkenler, mobil oyun tercihlerini analiz etmek için kullanılan temel faktörlerdir ve alt düzeyleri detaylı bir şekilde incelenmiştir. Araştırmanın bağımlı değişkeni olan mobil oyun tercihlerinin belirlenmesinde, bu bağımsız değişkenlerin etkileri ve ilişkileri üzerinde ayrıntılı bir analiz yapılmıştır. Bu bilgiler, mobil oyun geliştirme ve pazarlama stratejileri için önemli bir yol haritası sağlamaktadır.

Bağımsız Değişkenler		DÜZEYLER	
	1.Düzey	2.Düzey	3.Düzey
Oyun Tasarımı	Çekici	İtici	
Kullanıcı Arayüzü	Basit	Karmaşık	
Karakterler	Çok Çeşitli	Sade	
Kurulum Dosyası Büyüklüğü	Düşük	Yüksek	
İşgal Ettiği Hafıza	Düşük	Yüksek	
Gereksinim Duyduğu Sistem	Yüksek Donanımlı	Standart Donanımlı	
Oyun Oynama Şekli	Tuşlu	Dokunmatik	
İnternet Bağlantısı	Gerekli	Gerekli Değil	
Oyun İçerisinde Reklam Gösterimi	Var	Yok	
Oyuncu Sayısı	Tek Oyunculu	Çok Oyunculu	
Şarj Tüketimi	Düşük	Yüksek	
Diğer Kişilerce İndirilme	Az	Fazla	
Yüklenme Sayısı	Düşük	Yüksek	
Oyun İçin Verilen Yıldız	3 Yıldız	4 Yıldız	5 Yıldız
Oyuna Yönelik Yorumlar	Olumlu	Olumsuz	
Oyuncu Adına Paylaşımda Bulunması	Evet	Hayır	

Tablo 1: Mobil Oyun Tercihini Etkileyen Değişkenler ve Düzeyleri

Asya-Pasifik pazarı, oyun sektörünün gelişiminde büyük bir etkiye sahiptir. Hem video oyunları hem de mobil oyunlarda, bu bölge yüksek rekabet ve geniş kullanıcı kitlesi ile öne çıkar. 'Asya Kaplanları' olarak bilinen Tayvan, Singapur, Hong Kong ve Güney Kore, 2000'li

yılların başındaki teknolojik atılımlarıyla mobil oyun sektöründe öncü oldular. Dünya genelinde en büyük ve en değerli oyun şirketlerinin çoğu Asya kökenlidir. En değerli 10 oyun şirketinden ikisi Çin, ikisi Japonya ve altısı ABD merkezlidir. Asya-Pasifik bölgesi, küresel oyun pazarında kritik bir rol oynamaktadır (Tablo 2).

Türkiye'deki oyun sektörünün temelleri, 2002 yılında Harbiye'deki Askeri Müzede düzenlenen kapsamlı bir oyun etkinliğiyle atılmıştır. Bu etkinlik, yurt dışında oyun geliştiren yazılımcılar, oyun sektörünü bir vizyon olarak gören akademisyenler ve oyun oynamayı ciddi bir iş olarak gören meraklılar tarafından büyük ilgi görmüştür. Daha sonra, oyun fikri olanlar için ODTÜ Teknokent'te ATOM adında bir merkez kurulmuştur. Ardından Türkiye Oyun Geliştiricileri Derneği (TOGED) kurularak sektörün profesyonel anlamda gelişimine önemli katkılar sağlamıştır. Türkiye'nin oyun sektörü, özellikle e-gameshow ve GİST gibi sektörel fuarlarla yeni pazarlara açılarak büyümüştür. Ayrıca "Game Jam" adı verilen oyun geliştirme yarışmalarıyla da sektör güçlenmiş ve yenilikçi projelere olan ilgi artmıştır. Türkiye'deki oyun sektörü, bu girişimler ve organizasyonlar sayesinde hızla büyüyerek uluslararası alanda tanınan bir sektör haline gelmiştir (Öztürk & Bülbül, 2022).

Markalar	Genel Merkez	Büyüme Oranı (2020)	Yıllık Gelir (2020)
Tencent	Çin	%34	27,441 Milyar $
Sony	Japonya	%27	17,498 Milyar $
Apple	ABD	%19	13,020 Milyar $
Microsoft	ABD	%34	11,695 Milyar $
Google	ABD	%23	9,142 Milyar $
NetEase	Çin	%16	7,839 Milyar $
Nintendo	Japonya	%49	7,449 Milyar $
Activision Blizzard	ABD	%27	7,399 Milyar $
Electronic Arts	ABD	%5	5,670 Milyar $
Take-Two Interactive	ABD	%15	3,294 Milyar $

Tablo 2. Dünya'nın En Değerli 10 Oyun Şirketi

2020 yılında Türk oyun sektörünün dünya sıralamasında 18. olduğu belirtilmiştir. Sektör, 36 milyon oyuncu sayısına ulaşarak toplamda 880 milyon dolar gelir elde etmiştir. Türkiye'deki yetişkinlerin %79'u mobil oyunlarla ilgilenmektedir. 2020 yılında mobil oyun sektörünün hacmi 450 milyon dolara ulaşırken, bilgisayar oyunları 230 milyon dolar ve konsol oyunları ise 200 milyon dolarlık hacimle mobil oyunların ardından (Öztürk & Bülbül, 2022).

2. LİTERATÜR

2.1 BENZER OYUNLAR

2.1.1 Brawlhalla

Brawlhalla, Blue Mammoth Games tarafından geliştirilen ücretsiz bir dövüş oyunudur. Oyunun yapımında ise oyun motoru olarak Adobe AIR kullanıOyun ilk olarak 2017 yılında macOS, PlayStation 4 ve Windows platformları için piyasaya sürüldü; daha sonra Nintendo Switch, Xbox One, Android ve iOS platformlarına da taşındı. Oyunun tam çapraz oynanabilirlik desteği tüm platformlarda mevcuttur. Brawlhalla'nın geliştirme süreci, Nisan 2014'te PAX East fuarında tanıtıldı ve aynı ayın sonunda oyuncuların erişimine sunulan alfa sürümüyle başladı. Oyunun açık beta süreci ise Kasım 2015'te başladı ve ardından Ekim 2017'de tam sürümü yayınlandı.

Brawlhalla, dünya çapında 100 milyondan fazla oyuncuya sahip geniş bir oyuncu kitlesine hitap etmektedir. Tek bir maçta 8'e kadar çevrimiçi oyuncu ile oynanabilme özelliği sunar. Oyunda iki oyuncunun karşılıklı yarışması ve oyuncuların genellikle bir veya daha fazla oyuncu olmayan karakter rakibe karşı takım arkadaşları olarak birlikte çalışmasına olanak sağlayan modlarına ek olarak 20'den fazla farklı oyun modu bulunmaktadır. Bu modlar, oyunculara çeşitli deneyimler sunmaktadır.

Şekil 3 Brawlhalla Oyunun Resmi

İlk olarak değinmek gerekirse bu oyun gerçekten güzel bir oyun. Ama bu oyunun mobile versionuna karşı yapılan haksızlık sebebiyle ben bu oyuna 1 puan vereceğim. Eğer mümkünse bilgisayardan oynayın. Mobile versionunda, bilgisayar da olan çoğu özellik bulunmamakta. Bu yetmezmiş gibi, karşıya bilgisayar oyuncularıda geliyor. Tam anlamıyla bir rezalet.

Şekil 4 Brawlhalla Oyununa Yazılmış Bir Yorum

2.1.2 BLACKMOOR 2

BLACKMOOR 2, Four Fats Limited tarafından geliştirilen ve 6 Mayıs 2019'da piyasaya sürülen bir arcade platform oyunudur. Bu oyun, dövüş ve retro klasik ile modern oyun öğelerinin benzersiz bir karışımını sunar ve türünün tek örneği olarak öne çıkmaktadır. BLACKMOOR 2, kıvrımlı ve dönüşlü bir hikaye moduyla gelmektedir. Oyuncular, on üç farklı kahramanın yer aldığı zorlu bölümleri keşfederken, bölüm sonu canavarlarıyla epik savaşlara girişmektedir. Hikaye modu, karakterler ve düşmanlarla dolu zengin bir oyun sunmaktadır.

Oyunun minimum sistem gereksinimleri 1,5 GB RAM ve Android 8.0+ işletim sistemini gerektirir. Oyunun indirilmesi için 400 MB depolama alanı gereklidir. Çok oyunculu ve zindan modları için internet bağlantısı gereklidir.

Şekil 5 Blackmoor 2 Oyunun Resmi

berbat bir oyun adam oyunda kendi kendine ölüyo diğer büyük caravarlarda bazen can gitmiyo canavarlar arka arkaya aynı hareketi yaparken biz niye yapamıyoruz oyunu tavsiye etmem sonra oyuna sinir olup benim gibi olmayın bence

Şekil 6 Blackmoor 2 Oyununa Yazılmış Bir Yorum

2.1.3 Apple Knight Action Platformer

Apple Knight, Limitless, LLC tarafından geliştirilen ve 12 Eylül 2019'da piyasaya sürülen modern bir aksiyon platform oyunudur. Oyun, hassas dokunmatik kontroller, akıcı hareketler ve akıcı animasyonlarla dikkat çekmektedir. Apple Knight, aksiyon dolu bir platform oyunu olup oyunculara heyecan verici bir macera sunar. Oyuncular, çeşitli seviyelerde ilerlerken engelleri aşmalı, düşmanları yenmeli ve sırları keşfetmelidir. Oyunun temposu hızlı olup akıcı kontrolleri sayesinde oyunculara rahat bir deneyim sunmaktadır. Apple Knight'ı oynamak için aşağıdaki minimum sistem gereksinimlerine ihtiyaç vardır:

İşletim Sistemi: Windows 10 (v2004)

Depolama: 10 GB kullanılabilir depolama alanına sahip Katı Hal Sürücüsü (SSD)

Grafik: Intel® UHD Graphics 630 GPU veya eşdeğeri

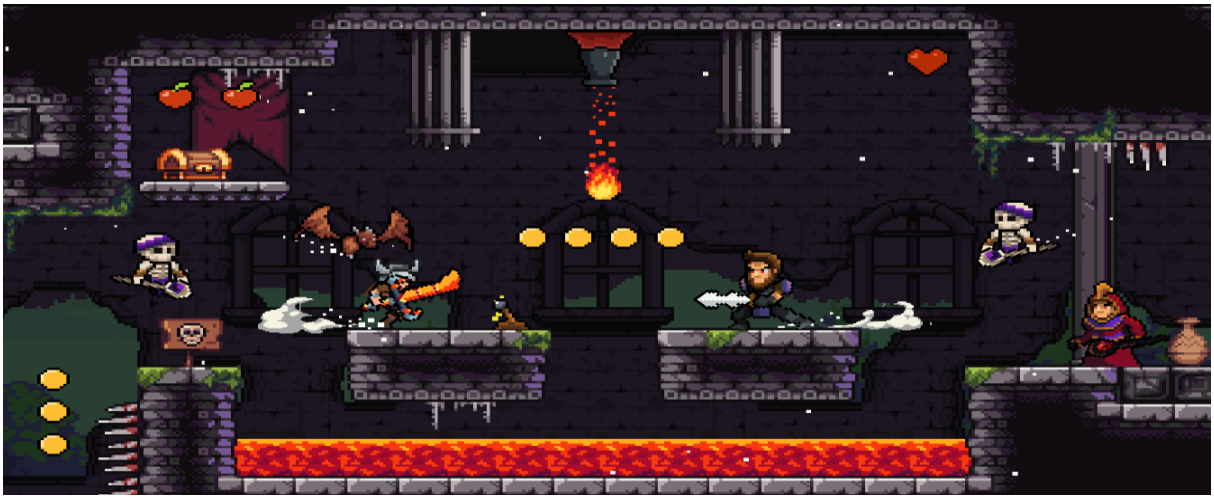
İşlemci: 4 CPU fiziksel çekirdek (Bazı oyunlar için Intel CPU gereklidir.)

Bellek: 8 GB RAM

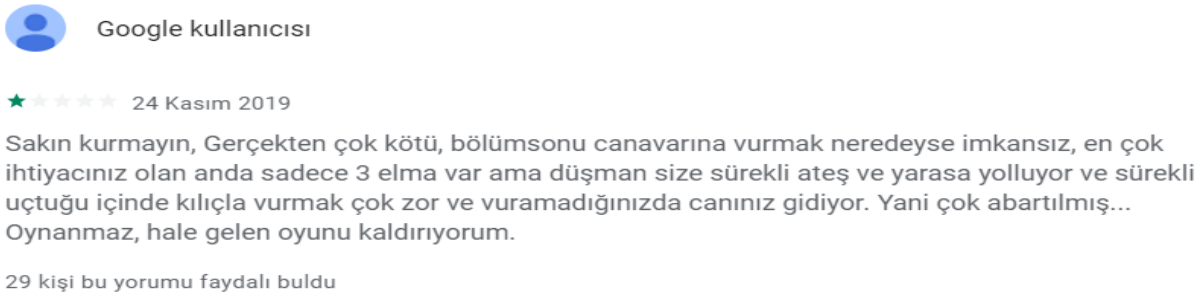
Windows yönetici hesabı

Donanım sanallaştırmanın açık olması gerekir

Bu gereksinimler, oyunun en iyi performansı için önerilen donanım özellikleridir.



Şekil 7 Apple Knight Limitless Oyununun Resmi



Şekil 8 Apple Knight Limitless Oyununa Yazılmış Bir Yorum

2.1.4 Draconian:Action Platformer 2D

Draconian, Winterdreams tarafından geliştirilen ve 3 Mayıs 2021'de piyasaya sürülen retro piksel sanat grafiklerine sahip bir aksiyon platform oyunudur. Bu oyun, Android 5.1 ve üzeri işletim sistemlerinde çalışmaktadır ve 100.000'den fazla indirilmiştir. Oyunun özellikleri arasında, el yapımı animasyonlarla desteklenen retro piksel sanat grafikleri bulunmaktadır.

Draconian'da oyuncular, farklı düşmanlarla dolu 4 farklı bölgeyi keşfederler ve 5 epik patronla savaşırlar. Oyun, hikaye odaklı bir deneyim sunarak oyuncuları derin bir maceranın içine çekmektedir. Draconian, çevrimdışı veya çevrimiçi olarak istenilen zaman oynanabilir. Bu özellik, oyunculara oyunu kendi tercihlerine göre deneyimleme özgürlüğü sağlamaktadır.

Şekil 9 Draconian:Action Platformer 2D Oyununa Ait Resim

2.1.5 Gladihoppers - Gladiator Fight

Dreamon Studios'un geliştirdiği bir mobil oyun olan bu oyun, 19 Aralık 2018'de piyasaya sürüldü ve Android 5.1 ve üzeri işletim sistemlerinde çalışmaktadır. Oyun, 5 milyondan fazla indirilmiştir. Oyunun özellikleri arasında çevrimiçi çok oyunculu modlar bulunmaktadır. Oyuncular, rastgele rakiplere veya arkadaşlarına karşı çevrimiçi savaşabilirler. Oyun, her

vücut parçası için isabet kutularıyla hızlı tempolu yakın dövüş ve menzilli savaşlar sunmaktadır.

Dreamon Studios'un oyununda gladyatörünüzü zafere taşımanızı sağlayacak çeşitli taktikler ve stratejiler içeren kariyer modu da bulunmaktadır. Ayrıca, oyuncuların grupları, orduları ve kasabaları fethedeceği 'Spartacus Savaşı' adlı strateji modu ve neredeyse sonsuz dövüş deneyimi sunan 'İmparatoru Kurtar' adlı arcade modu da mevcuttur.

Dreamon Studios'un oyunu, çok yönlü oyun modları ve zengin içeriğiyle dikkat çeken bir mobil platform oyunudur. Çevrimiçi çok oyunculu savaşlar, stratejik elementler ve heyecan verici dövüş mekanikleriyle oyun, oyunculara eğlenceli ve rekabetçi bir deneyim sunmayı hedeflemiştir.

Şekil 10 Gladihoppers-Gladiator Fight Oyununa Ait Resim

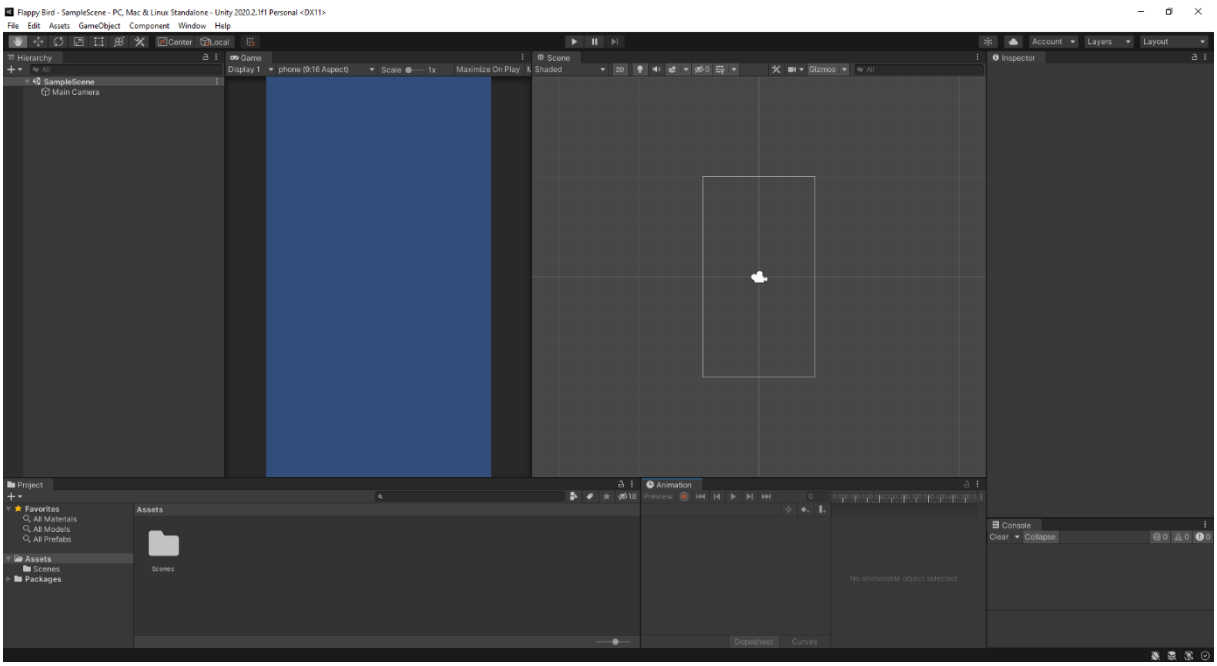
Şekil 11 Gladihoppers-Gladiator Fight Oyununa Yazılmış Bir Yorum

2.2 KULLANILAN UYGULAMALAR

2.2.1 Unity Oyun Motoru

Unity, video oyunları ve simülasyonlar için geliştirilen ve Unity Technologies tarafından üretilen bir çapraz platform oyun motorudur. İlk olarak 2005 yılında yalnızca OS X için duyurulan Unity, sonrasında 27 farklı platformu hedeflemek üzere genişletilmiştir. Unity oyun motoru, sadece video oyunları değil aynı zamanda film endüstrisi, otomotiv sektörü, mimari, mühendislik ve inşaat gibi farklı sektörler tarafından da benimsenmiş ve kullanılmıştır.

Unity'nin en belirgin avantajlarından biri, geliştirilen bir oyunun farklı platformlara (PC, Mac, Web, iOS, Android, Windows Phone, Playstation, Xbox vb.) uyumlu olarak derlenebilmesidir. Bu özellik sayesinde, PC için geliştirilen bir oyun tek tıklamayla Mac'e uyumlu hale getirilebilmektedir. Ayrıca, Unity son derece pahalı diğer gelişmiş oyun motorlarının sunduğu özellikleri (gelişmiş shader yazılımı, fizik motoru, animasyon editörü, occlusion culling vb.) ücretsiz olarak sunmaktadır, bu da uygulama ve oyun geliştiricilerine önemli bir maliyet avantajı sağlamaktadır. Unity'nin bir diğer üstünlüğü ise geliştirme sürecinde grafik ve kodun birlikte çalışmasıdır. Bu yaklaşım, geliştiricilere esneklik sağlamakta ve geliştirme sürelerini kısaltmaktadır. Ayrıca Unity ile hazırlanan oyunlar düşük ve orta seviyedeki bilgisayarlarda sorunsuz bir şekilde oynanabilmektedir. Unity ayrıca eğitim amaçlı simülasyonlar oluşturmak için de kullanılabilmektedir. Son versiyonu kişisel kullanımlar için ücretsiz olarak sunulmuş ve daha geniş bir kitleye hitap etmektedir.



Şekil 12 Unity Örnek Arayüz

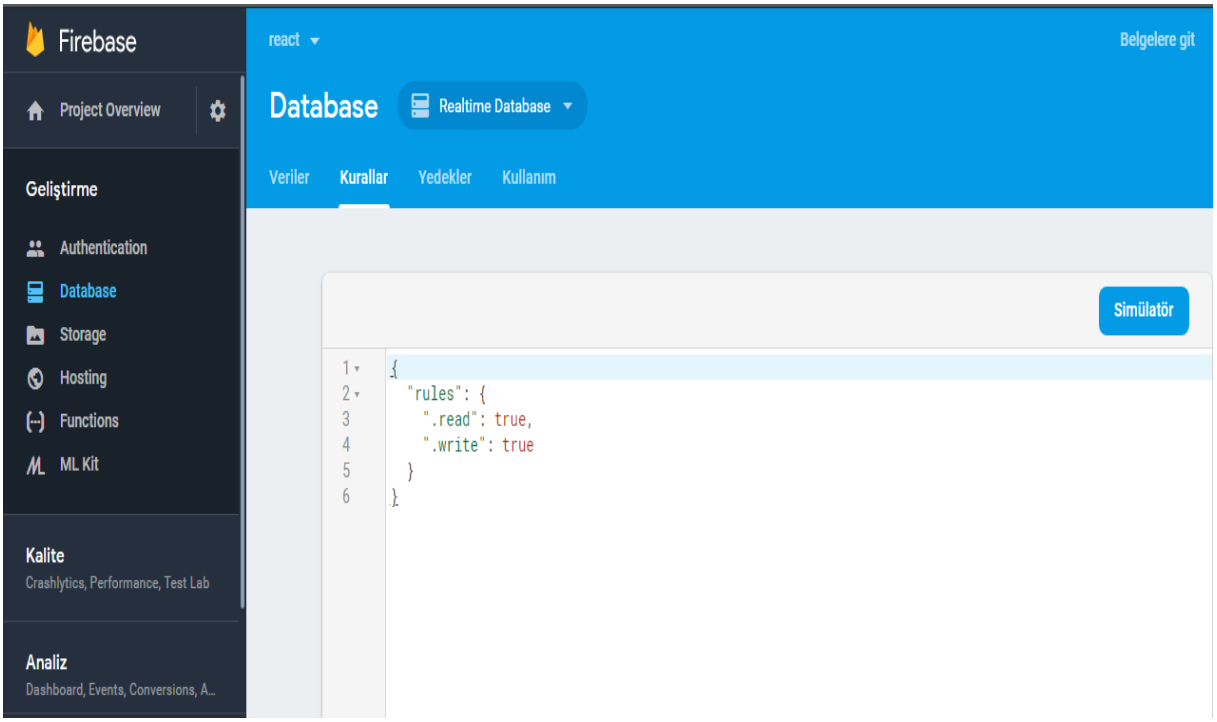
2.2.2 Firebase Anlık Veri Tabanı

Firebase, mobil ve web uygulamaları geliştirmek için özel olarak tasarlanmış bir platform olarak dikkat çekmektedir. 2011 yılında kurulan ve 2014'te Google tarafından satın alınan Firebase, uygulama geliştirme alanında Google'ın öncü hizmetlerinden biri haline gelmiştir.

Firebase, geliştiricilere bulut tabanlı bir dizi hizmet sunmaktadır. Bu hizmetler arasında veritabanı, kullanıcı kimlik doğrulama, dosya depolama, analiz, mesajlaşma ve push bildirimleri gibi özellikler bulunmaktadır. Platform, gerçek zamanlı veritabanı özelliği ve hızlı prototip oluşturma imkanıyla öne çıkmaktadır.

Geliştirme sürecini hızlandıran Firebase, Google'ın altyapısıyla entegre olarak çalışır ve uygulamaların Google Cloud hizmetlerinden tam olarak yararlanmasını sağlar. Firebase'in sunduğu kolaylık ve gelişmiş özellikler, mobil ve web uygulamaları geliştirme süreçlerini daha verimli hale getirmektedir.

Google'ın sürekli olarak Firebase'e yeni özellikler eklemesi ve geliştirmeler yapması, platformun güncel ve rekabetçi kalmasını sağlamaktadır. Firebase, geliştiricilere kullanıcı deneyimini iyileştirmek için gerekli araçları sunarak modern uygulama geliştirme süreçlerine katkı sağlamaktadır.



Şekil 13 Firebase Örnek Arayüz

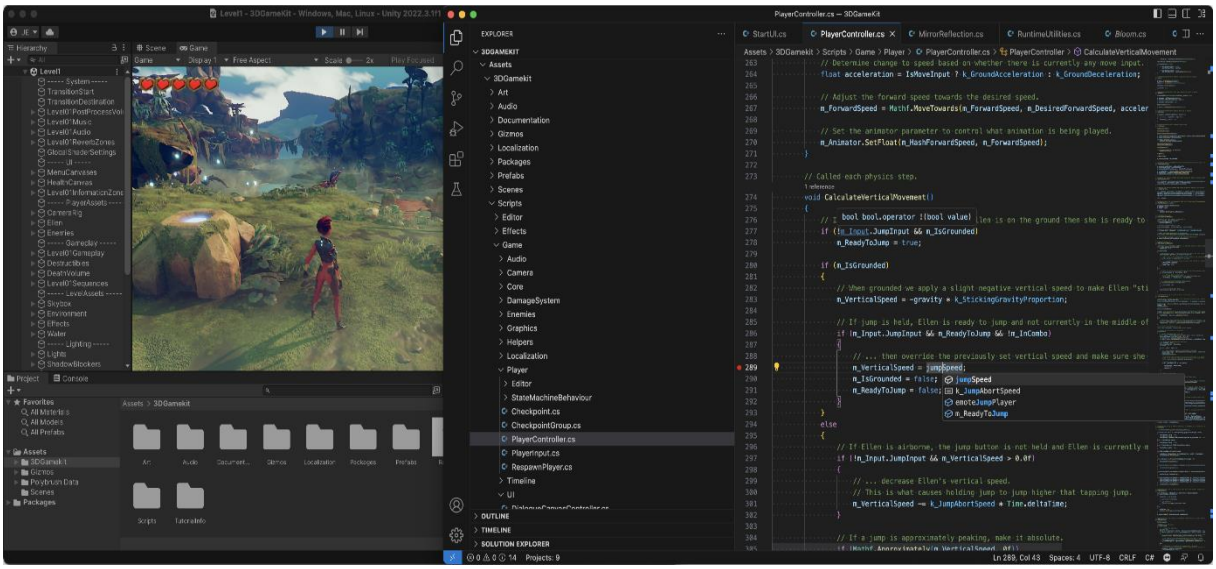
2.2.3 C# Oyun Programlama Yazılım Dili

C# (C sharp), Microsoft tarafından geliştirilen ve oyun programlama alanında yaygın olarak tercih edilen bir programlama dilidir. Nesne yönelimli programlama (OOP) prensiplerine dayanan C#, varlık yönetimi, etkileşimler, olaylar ve işlevsellikler gibi oyun geliştirme kavramlarını kolayca uygulamaktadır. Bu dil, verimli ve hızlı oyun geliştirme süreçleri için

geniş bir standart kütüphane ve zengin bir ekosistem sunmaktadır. Ayrıca, C# ile geliştirilen oyunlar popüler grafik ve ses kütüphaneleriyle uyumlu çalışabilmektedir.

Oyun geliştirme sürecinde C# programlama dili, proje planlama, tasarım, kodlama, grafik ve ses entegrasyonu, test ve hata ayıklama, dağıtım ve yayın gibi aşamalarda kullanılmaktadır. Proje planlama ve tasarım aşamasında oyun fikri belirlenir ve oyun mekaniği detaylı olarak tasarlanır. Kodlama sürecinde, C# ile oyunun temel bileşenleri oluşturulur ve oyun motorlarıyla entegrasyon sağlanır. Grafik tasarımları ve ses efektleri C# kodlarıyla oyun motoruna entegre edilir. Son olarak, oyun test edilir, hatalar düzeltilir ve farklı platformlara derlenerek yayınlanır.

C# programlama dili, oyun geliştirme süreçlerinde güçlü bir araç olarak kullanılmaktadır ve oyun endüstrisinin teknolojik gelişimine katkı sağlamaktadır. Geliştiriciler, C# dilinin sağladığı kolaylık ve verimlilik sayesinde yaratıcı ve etkileyici oyunlar oluşturabilmektedirler.



Şekil 14 C# ile Unity Kullanımı Hakkında Örnek Arayüz

3.YÖNTEM

3.1 Projenin Tanımı

Bu tez, çocukların ve gençlerin eğlenceli ve öğretici bir deneyim yaşayabileceği "Platform" adlı mobil oyun projesini detaylı bir şekilde tanıtmaktadır. Oyun, Unity 2D oyun motoru ve C# programlama dili kullanılarak geliştirilmiştir. Ayrıca Firebase platformu, özellikle Firebase Realtime Database'i, anlık veri kaydı ve oyun verilerinin depolanması için aktif olarak kullanılmıştır.

3.2 Projenin Modeli

Bu mobil oyun projesi için Unity 2D oyun motoru, C# programlama dili ve Firebase platformunun tercih edilmesi belirli avantajlar sağlamıştır. Unity 2D, mobil oyun geliştirme için popüler bir platformdur ve 2D grafiklerle zengin, interaktif oyunlar oluşturmayı kolaylaştırır. "Platform" projesi için Unity 2D, hızlı prototipleme, kolay kullanım ve geniş topluluk desteği gibi özellikleriyle tercih edilmiştir.

C# programlama dili, Unity'nin resmi dilidir ve Unity'nin sunduğu zengin API'ler ve yeteneklerle entegrasyonu kolaylaştırır. Bu proje için C# dilinin tercih edilmesi, oyun mekaniğinin ve arayüzünün etkin bir şekilde kodlanmasını sağlamış ve geliştirme sürecini optimize etmiştir.

3.3 Veritabanı Sunucusu

Firebase ise bu projede veri yönetimi ve kullanıcı etkileşimini kolaylaştırmak amacıyla kullanılmıştır. Firebase Realtime Database, kullanıcı ilerlemesi, skorlar ve diğer oyun

verilerinin anlık olarak kaydedilmesini ve senkronize edilmesini sağlayarak kullanıcı deneyimini geliştirmiştir. Ayrıca Firebase Authentication, kullanıcı kimlik doğrulamasını sağlamış ve oyun içi etkileşimi güvenli bir şekilde yönetmemizi sağlamıştır.

3.1 Mobil Oyun Projesi

Oyunun amacı, kullanıcıların farklı bölümlerde karşılaşacakları zorlukları aşarak ruhani bir serüvene atılmalarını sağlamaktır. Platform, mobil platformlarda oynanabilen bir oyun olup, kayıtlı olmayan oyuncuların önce kayıt olmaları ve ardından giriş yapmaları gerekmektedir. Oyuncular, oyunun başlangıç ekranından kayıt işlemlerini tamamladıktan sonra bilgilerini girerek oyuna erişebilirler. Oyun, üç farklı bölümden oluşmaktadır ve her bir bölümde oyuncular, elmaları toplayarak düşmanları yenerek en yüksek skoru elde etmeye yönelik tasarlanmıştır.

3.2 Projenin Tasarımı

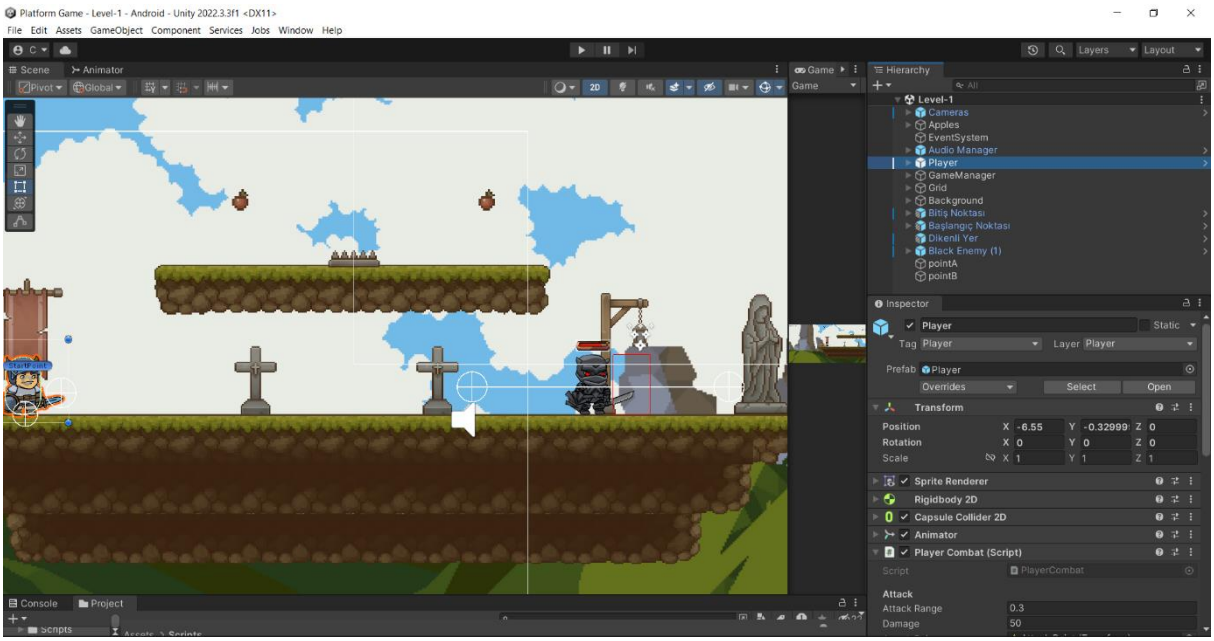
Oyunun ilk aşaması, geliştirme sürecindeki en önemli adımlardan biri olan tasarım aşamasıdır. Bu aşamada, oyunun görünümü, mekanikleri ve oynanabilirliği belirlenmekte ve hazırlık süreci tamamlanmaktadır.

Tasarım sürecine başlamadan önce, uygun grafik ve ses assetleri bulunarak oyunun temel bileşenleri indirilmiş ve projeye entegre edilmiştir. Oyunun konseptine uygun grafikler seçilmiş ve kullanılacak olan karakterler, arka planlar, objeler ve efektler belirlenmiştir.

Daha sonra, sahne tasarımı için bir tilemap paleti kullanılarak oyun dünyası oluşturulmuştur. Tilemap paleti, farklı parçalardan oluşan ve kolayca birleştirilebilen önceden tasarlanmış grafiklerin bulunduğu bir araçtır. Bu palet kullanılarak, oyunun farklı seviyelerinde geçecek sahneler oluşturulmuş ve düzenlenmiştir. Elma ağaçları, yollar, engeller ve düşmanların yerleşeceği alanlar tilemap üzerine yerleştirilmiştir.

Ardından, oyun karakterlerine fizik özellikleri eklenerek sahne tasarımı tamamlanmıştır. Karakterler için Unity'nin fizik motoru olan Rigidbody2D ve çarpışma algılaması için BoxCollider2D bileşenleri eklenmiştir. Bu sayede, karakterlerin oyun dünyasıyla etkileşimi sağlanmış ve oyuncunun karakteri kontrol edebilmesi mümkün hale gelmiştir.

Tasarım aşaması boyunca, oyunun görsel ve işlevsel olarak nasıl olacağı belirlenmiş ve önemli bileşenler bir araya getirilmiştir. Grafik tasarımı ve sahne düzenlemesiyle birlikte, oyunun temel mekanikleri de bu aşamada belirlenmiş ve uygulanmıştır. Bu süreç, oyunun ilerleyen aşamalarında geliştirme ve optimizasyon için sağlam bir temel oluşturmayı amaçlamıştır.



Şekil 15 Oyun Sahnesinin Birinci Bölüm Tasarımı

3.3 Ana Karakter Kontrolleri

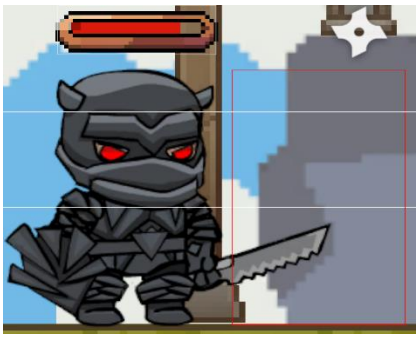
Oyunun bu aşamasında, ana karakterin kontrolleri için sağa ve sola hareket yetenekleri eklenmiştir. Karakterin yatay yönde hareket edebilmesi, oyuncunun oyun dünyasında

serbestçe dolaşabilmesini sağlamaktadır. Ayrıca, çift zıplama özelliği de karakter kontrollerine eklenmiştir. Bu özellik sayesinde oyuncular, iki kez zıplayarak daha yüksek yerlere ulaşabilir veya engelleri aşabilmeleri hedeflenmiştir. Bunun yanı sıra, karakterin saldırı mekanikleri de oyunun bu aşamasında geliştirilmiştir. Oyuncular, düşmanlarla savaşmak için saldırı komutunu kullanabilirler. Saldırı mekanikleri, düşmanlara zarar vermek için kullanılan bir yetenektir ve oyuncuların stratejik becerilerini test etmelerini sağlamaktadır. Saldırı mekanikleri için ise karakterin önünde bir saldırı alanı oluşturulmuş ve bu alana temas eden düşmanlar zarar görmüştür. Saldırı komutu, oyuncunun istediği zaman kullanabileceği bir yetenek olarak düzenlenmiştir. Karakterin sağlık durumunu göstermek için kalp şeklinde bir can barı kullanılmaktadır. Can barı, oyuncunun karakterin ne kadar sağlığa sahip olduğunu görsel olarak temsil eder ve oyuncunun oyun dünyasındaki hayatta kalma yeteneğini yansıtmaktadır.

Şekil 16 Oyunun Ana Karakterinin Resmi

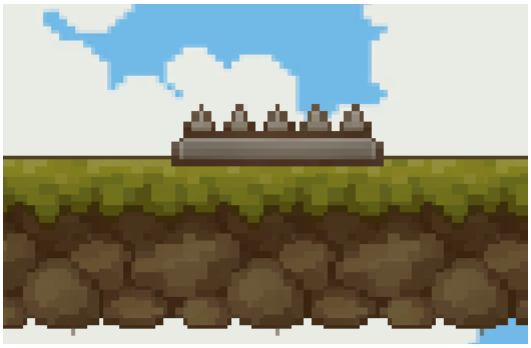
3.4 Düşman Karakter ve Tuzaklar

Oyunda, düşman karakter belirli bir alanda gidip gelerek hareket eder. Ana karakter o bu alana girince düşman, ana karaktere doğru ilerleyerek saldırır. Bu mekanik, oyuncuların düşmanlarla etkileşimde bulunurken stratejik kararlar almasını sağlamak amacıyla tasarlanmıştır.



Şekil 17 Oyundaki Düşman Karakterin Resmi

Oyun içerisinde çeşitli yerlerde tuzaklar bulunmaktadır. Tuzaklar, ana karakter engellerden atlarken değmesi halinde birer birer canı gitmektedir. Bu durum oyuncuların engelleri aşarken daha dikkatli olması için konulmuştur.



Şekil 18 Oyundaki Tuzak Nesnesinin Resmi

3.5 Skor Toplama

Elmaları toplama işlemi, oyuncuların oyun dünyasında dağılmış olan elmaları toplayarak skor kazanmalarını sağlar. Her bir elma, belirli bir skor değeri taşır ve oyuncu elmayı topladığında skorları artar. Oyuncular, en yüksek skoru elde etmek için mümkün olduğunca çok elma toplamaya çalışmaktadırlar.

Düşmanları yenme mekanikleri ise oyuncuların düşman karakterlerle etkileşimde bulunarak skor kazanmalarını sağlar. Düşmanı yenmek, oyunculara ek skor sağlar ve ilerleyen seviyelere

geçmelerine yardımcı olmaktadır. Düşmanları yenme süreci genellikle strateji ve reflekslerin test edildiği bir deneyim sunmaktadır.

Oyuncular, elmaları toplama ve düşmanları yenme işlemleriyle skorlarını artırarak liderlik tablosunda yüksek sıralara yükselmeyi hedeflemesini sağlamaktadır. Liderlik tablosu, oyuncuların diğer oyuncularla rekabet etmelerini ve en yüksek skoru elde etmek için çaba göstermelerini teşvik etmektedir.

Şekil 19 Oyun İçerisindeki Elma Nesnesinin Resmi

3.6 Can Barı

Ana karakterin can barı, ekranın üst kısmında kalp sembolleri olarak gösterilmektedir. Her kalp sembolü, karakterin belirli bir sağlık miktarını temsil etmektedir. Üç kalp sembolü olan bir can barı, karakterin toplam üç canına sahip olduğunu gösterir. Karakter, düşman saldırılarına maruz kaldıkça veya zararlı unsurlarla etkileşimde bulundukça canı azalmaktadır.

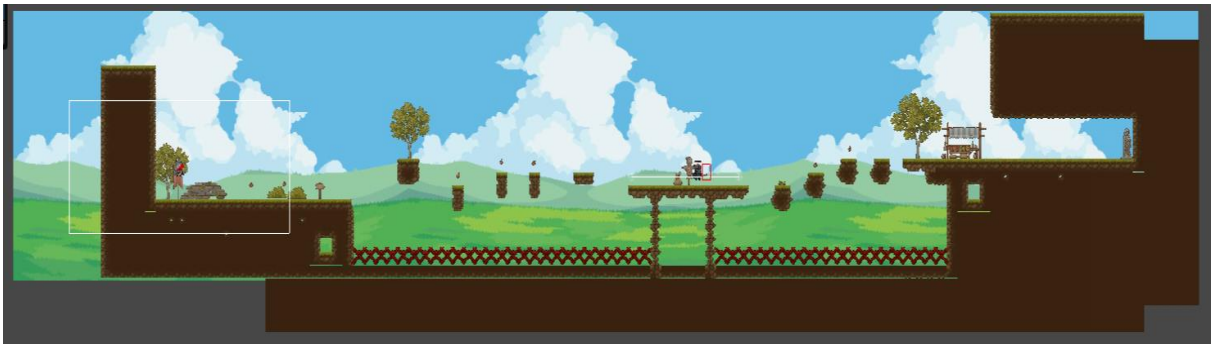
Oyuncu, karakterin can durumunu sürekli olarak gözlemleyerek ne kadar sağlığa sahip olduğunu anlayabilir. Eğer karakterin canı tamamen tükenirse, oyuncu oyunu kaybedebilir veya belirli bir noktadan yeniden başlamak zorunda kalabilir.

Can barı sistemi, oyunun zorluk seviyesini belirleyen ve oyuncunun stratejik kararlar almasını sağlayan önemli bir özelliktir. Oyuncular, düşman saldırılarından kaçınarak veya düşmanları etkili bir şekilde yenerek canlarını korumaya çalışması sağlanmıştır.

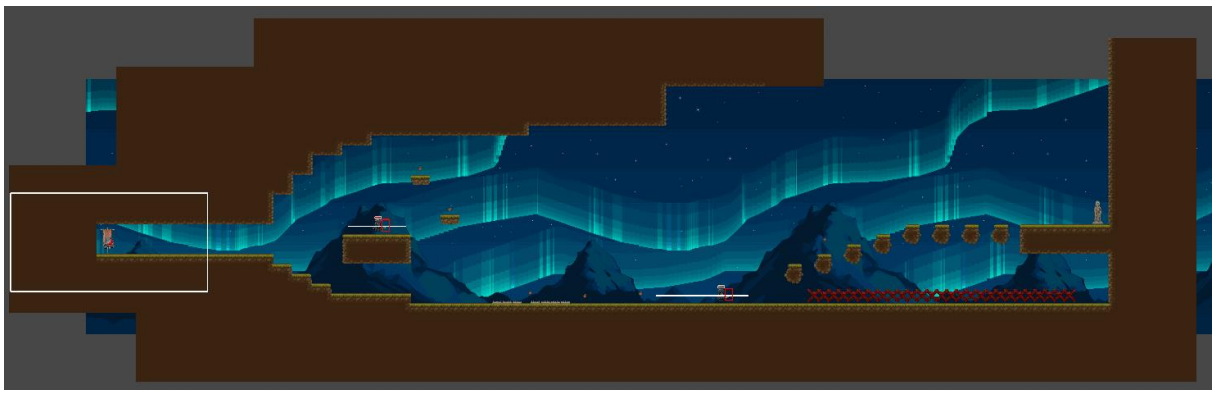
Şekil 20 Ana Karakterin Can Barı

3.7 Oyun Bölümleri

Oyun 3 bölümden oluşmaktadır. Oyundaki anıt nesnesi ile etkileşime geçildiği zaman bir sonraki bölüme ışınlamaktadır.



Şekil 21 Oyunun İkinci Bölüm Tasarımı



Şekil 22 Oyunun Üçüncü Bölüm Tasarımı

4.KODLAR

```
public class PlayerCombat : MonoBehaviour
{
    private Animator _animator;
    [Header("Attack")]
    [SerializeField] private float _attackRange = 0.5f;
    [SerializeField] private int damage = 20;
    [SerializeField] private Transform _attackPoint;
    [SerializeField] private LayerMask _enemyLayers;
    AudioManager audioManager;
    private bool _isAttacking;
    Unity İletisi | 0 başvuru
    private void Start()
    {
        audioManager = GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();
        _animator = GetComponent<Animator>();
    }
    Unity İletisi | 0 başvuru
    private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            Attack();
            UpdateAnimation();
        }
    }
    1 başvuru
    private void Attack()
    {
        Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(_attackPoint.position, _attackRange, _enemyLayers);
        foreach (Collider2D enemy in hitEnemies)
        {
            audioManager.PlaySFX(audioManager.playerAttack);
            enemy.GetComponent<Enemy>().TakeDamage(damage);
        }
    }
    1 başvuru
    private void UpdateAnimation()
    {
        _animator.SetTrigger("Attack");
    }
    Unity İletisi | 0 başvuru
    private void OnDrawGizmosSelected()
    {
        if (_attackPoint == null)
        {
            Debug.Log("PlayerCombat : Attack point değeri verilmedi");
            return;
        }
        Gizmos.DrawWireSphere(_attackPoint.position, _attackRange);
    }
}
```

```
public class HealthDisplay : MonoBehaviour
{
    [Header("Player Health")]
    public int _health;
    public int _maxHealth;
    [Header("Sprites Hearth")]
    [SerializeField] private Sprite _emptyHearth;
    [SerializeField] private Sprite _fullHearth;
    [SerializeField] private Image[] _hearts;
    public PlayerController _playerController;
    Unity İletisi | 0 başvuru
    private void Awake()
    {
        _health = _maxHealth;
        _playerController = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
    }
    Unity İletisi | 0 başvuru
    private void Update()
    {
        for (int i = 0; i < _hearts.Length; i++)
        {
            if (_health == 0)
            {
                /*
                _playerController.Die();
                _health = _maxHealth;
                */
            }
            else
            {
                if (i < _health)
                    _hearts[i].sprite = _fullHearth;
                else
                    _hearts[i].sprite = _emptyHearth;

                if (i < _maxHealth)
                    _hearts[i].enabled = true;
                else
                    _hearts[i].enabled = false;
            }
        }
    }

    2 başvuru
    public void TakeDamage(int damage)
    {
        _health -= damage;
        if (_health < 0)
        {
            _health = 0;
        }
    }

}
```

```
public class PlayerController : MonoBehaviour
{
    [Header("Movement")]
    public float movementSpeed = 10.0f;
    private float movementInputDirection;
    private bool isFacingRight = true;
    private bool isWalking;
    private bool isGrounded;
    public float groundCheckRadius;
    [Header("Jump")]
    public float jumpForce = 16.0f;
    public int amountOfJumps = 2;
    private int amountOfJumpsLeft;
    private bool canJump;
    private Rigidbody2D rb;
    private Animator anim;
    [Header("Level Start")]
    [SerializeField] private Transform _startPosition;
    [Header("Referances")]
    public Transform groundCheck;
    public LayerMask whatIsGround;
    public AudioManager audioManager;
    public GameManager gameManager;
    public HealthDisplay healthDisplay;
    Unity İletisi | 0 başvuru
    private void Awake()
    {
        gameManager = GameObject.FindGameObjectWithTag("GameManager").GetComponent<GameManager>();
        healthDisplay = GameObject.FindGameObjectWithTag("GameManager").GetComponent<HealthDisplay>();
        audioManager = GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();
        if(_startPosition == null)
        {
            _startPosition.position = gameObject.transform.position;
        }
    }
    Unity İletisi | 0 başvuru
    private void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();

        amountOfJumpsLeft = amountOfJumps;
    }
    Unity İletisi | 0 başvuru
    private void Update()
    {
        CheckInput();
        CheckMovementDirection();
        CheckIfCanJump();
        UpdateAnimations();
    }
```

```
private void FixedUpdate()
{
    ApplyMovement();
    CheckSurroundings();
}
1 başvuru
private void CheckInput()
{
    movementInputDirection = Input.GetAxisRaw("Horizontal");
    if (Input.GetButtonDown("Jump"))
    {
        Jump();
    }
}
1 başvuru
private void CheckMovementDirection()
{
    if (isFacingRight && movementInputDirection < 0)
    {
        Flip();
    }
    else if (!isFacingRight && movementInputDirection > 0)
    {
        Flip();
    }
    if (rb.velocity.x != 0)
    {
        isWalking = true;
    }
    else
    {
        isWalking = false;
    }
}
1 başvuru
private void CheckSurroundings()
{
    isGrounded = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, whatIsGround);
}
2 başvuru
private void Flip()
{
    isFacingRight = !isFacingRight;
    transform.Rotate(0.0f, 180.0f, 0.0f);
}
```

```
private void Jump()
{
    if (canJump)
    {
        rb.velocity = new Vector2(rb.velocity.x, jumpForce);
        amountOfJumpsLeft--;
    }
}
1 başvuru
private void CheckIfCanJump()
{
    if (isGrounded && rb.velocity.y <= 0)
    {
        amountOfJumpsLeft = amountOfJumps;
    }
    else if (amountOfJumpsLeft <= 0)
    {
        canJump = false;
    }
    else
    {
        canJump = true;
    }
}
1 başvuru
private void UpdateAnimations()
{
    anim.SetBool("isWalking", isWalking);
}
1 başvuru
private void ApplyMovement()
{
    rb.velocity = new Vector2(movementSpeed * movementInputDirection, rb.velocity.y);
}
Unity İletisi | 0 başvuru
private void OnDrawGizmos()
{
    Gizmos.DrawWireSphere(groundCheck.position, groundCheckRadius);
}
```

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Apple"))
    {
        audioManager.PlaySFX(audioManager.apple);
        gameManager.ScoreUpdate();
        Destroy(collision.gameObject);
    }
}
1 başvuru
public void TakeDamage(int damage)
{
    healthDisplay.TakeDamage(damage);
}
2 başvuru
public void Die()
{
    transform.position = _startPosition.position;
    audioManager.PlaySFX(audioManager.playerRespawn);
}
0 başvuru
public void WalkAudio()
{
    audioManager.PlaySFX(audioManager.playerWalk);
}
```

```
public class Enemy : MonoBehaviour
{
    [Header("Enemy")]
    [SerializeField] private float _deathTime = 1f;
    [Header("Health")]
    [SerializeField] private int _maxHealth = 100;
    private int _currentHealth;
    public HealthBar healthBar;
    private Animator _anim;
    AudioManager audioManager;
    Unity İletisi | 0 başvuru
    private void Awake()
    {
        audioManager = GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();
    }
    Unity İletisi | 0 başvuru
    private void Start()
    {
        _anim = GetComponent<Animator>();
        _currentHealth = _maxHealth;
        healthBar.SetMaxHealth(_maxHealth);
    }
    1 başvuru
    public void TakeDamage(int damage)
    {
        _currentHealth -= damage;
        _anim.SetTrigger("hurt");
        healthBar.SetHealth(_currentHealth);
        if (_currentHealth <= 0)
        {
            StartCoroutine(Die());
        }
    }
    1 başvuru
    private IEnumerator Die()
    {
        _anim.SetTrigger("die");
        audioManager.PlaySFX(audioManager.enemyDie);
        yield return new WaitForSeconds(_deathTime);
        GameObject.Destroy(gameObject);
    }
}
```

```
public class EnemyPatrol : MonoBehaviour
{
    [Header("Patrol Points")]
    [SerializeField] private Transform _leftEdge;
    [SerializeField] private Transform _rightEdge;
    [Header("Enemy")]
    [SerializeField] private Transform _enemy;
    [Header("Movement Parameters")]
    [SerializeField] private float _speed;
    private Vector3 _initScale;
    private bool _movingLeft;
    [Header("Idle Behaviour")]
    [SerializeField] private float _idleDuretion;
    private float _idleTimer;
    [Header("Enemy Animator")]
    [SerializeField] private Animator _anim;
    Unity İletisi | 0 başvuru
    private void Awake()
    {
        _initScale = _enemy.localScale;
    }
    Unity İletisi | 0 başvuru
    private void Update()
    {
        if (_movingLeft)
        {
            if (_enemy.position.x >= _leftEdge.position.x)
                MoveInDirection(-1);
            else
                DirectionChange();
        }
        else
        {
            if (_enemy.position.x <= _rightEdge.position.x)
                MoveInDirection(1);
            else
                DirectionChange();
        }
    }
```

```
    private void OnDisable()
    {
        _anim.SetBool("move", false);
    }
    2 başvuru
    private void DirectionChange()
    {
        _anim.SetBool("move", false);
        _idleTimer += Time.deltaTime;
        if (_idleTimer > _idleDuretion)
            _movingLeft = !_movingLeft;
    }
    2 başvuru
    private void MoveInDirection(int _direction)
    {
        _idleTimer = 0;
        _anim.SetBool("move", true);
        _enemy.localScale = new Vector3(Mathf.Abs(_initScale.x) * _direction, _initScale.y, _initScale.z);
        _enemy.position = new Vector3(_enemy.position.x + Time.deltaTime * _direction * _speed,
            _enemy.position.y, _enemy.position.z);
    }
}
```

```
public class HealthBar : MonoBehaviour
{
    public Slider _slider;
    [SerializeField] private Gradient _gradient;
    public Image _fill;
    1 başvuru
    public void SetMaxHealth(int health)
    {
        _slider.maxValue = health;
        _slider.value = health;
        _fill.color = _gradient.Evaluate(1f);
    }
    1 başvuru
    public void SetHealth(int health)
    {
        _slider.value = health;
        _fill.color = _gradient.Evaluate(_slider.normalizedValue);
    }
}
```

```
public class MeleeEnemy : MonoBehaviour
{
    [Header("Attack Parameters")]
    public float _attackCooldown;
    [SerializeField] private int _damage;
    [SerializeField] private float _range;
    [Header("Collider Parameters")]
    [SerializeField] private float _colliderDistance;
    [SerializeField] private BoxCollider2D _boxCollider;
    [Header("Player Layer")]
    [SerializeField] private LayerMask _playerLayer;
    [Header("Audio Manager")]
    public AudioManager audioManager;
    private Animator anim;
    public PlayerController playerController;
    public float _cooldownTimer = Mathf.Infinity;
    Unity İletisi | 0 başvuru
    private void Awake()
    {
        playerController = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
        audioManager = GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();
        anim = GetComponent<Animator>();
    }
    Unity İletisi | 0 başvuru
    private void Update()
    {
        _cooldownTimer += Time.deltaTime;
        if (PlayerInSight())
        {
            if (_cooldownTimer >= _attackCooldown)
            {
                _cooldownTimer = 0;
                anim.SetTrigger("meleeAttack");
            }
        }
    }
```

```
    public bool PlayerInSight()
    {
        RaycastHit2D hit = Physics2D.BoxCast(_boxCollider.bounds.center + transform.right * _range * transform.localScale.x * _colliderDistance,
            new Vector3(_boxCollider.bounds.size.x * _range, _boxCollider.bounds.size.y, _boxCollider.bounds.size.z),
            0, Vector2.left, 0, _playerLayer);
        if (hit.collider != null)
        {
            playerController = hit.transform.GetComponent<PlayerController>();
        }
        return hit.collider != null;
    }
    Unity İletisi | 0 başvuru
    private void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireCube(_boxCollider.bounds.center + transform.right * _range * transform.localScale.x * _colliderDistance,
        new Vector3(_boxCollider.bounds.size.x * _range, _boxCollider.bounds.size.y, _boxCollider.bounds.size.z));
    }
    0 başvuru
    private void DamagePlayer()
    {
        if (PlayerInSight())
        {
            playerController.TakeDamage(_damage);
            audioManager.PlaySFX(audioManager.enemyAttack);
        }
    }
}
```

```
public class Patrol : MonoBehaviour
{
    public GameObject pointA;
    public GameObject pointB;
    private Rigidbody2D rigidbody2;
    private Animator animator;
    private Transform currentPoint;
    public float speed;
    private MeleeEnemy meleeEnemy;
    Unity İletisi | 0 başvuru
    private void Start()
    {
        meleeEnemy = GetComponent<MeleeEnemy>();
        rigidbody2 = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
        currentPoint = pointB.transform;
        animator.SetBool("move", true);
    }
    Unity İletisi | 0 başvuru
    private void Update()
    {
        if(meleeEnemy.PlayerInSight() == false)
        {
            animator.SetBool("move", true);
            Vector2 point = currentPoint.position - transform.position;
            if (currentPoint == pointB.transform)
            {
                rigidbody2.velocity = new Vector2(speed, 0);
            }
            else
            {
                rigidbody2.velocity = new Vector2(-speed, 0);
            }

            if (Vector2.Distance(transform.position, currentPoint.position) < 0.5f && currentPoint == pointB.transform)
            {
                Flip();
                currentPoint = pointA.transform;
            }
            if (Vector2.Distance(transform.position, currentPoint.position) < 0.5f && currentPoint == pointA.transform)
            {
                Flip();
                currentPoint = pointB.transform;
            }
        }
```

```
        else
        {
            animator.SetBool("move", false);

            if (meleeEnemy._cooldownTimer >= meleeEnemy._attackCooldown)
            {
                // Attack
                meleeEnemy._cooldownTimer = 0;
                animator.SetTrigger("meleeAttack");
            }
        }
    }
    2 başvuru
    private void Flip()
    {
        Vector3 localScale = transform.localScale;
        localScale.x *= -1;
        transform.localScale = localScale;
    }
    Unity İletisi | 0 başvuru
    private void OnDrawGizmos()
    {
        Gizmos.DrawWireSphere(pointA.transform.position, 0.3f);
        Gizmos.DrawWireSphere(pointB.transform.position, 0.3f);
        Gizmos.DrawLine(pointA.transform.position, pointB.transform.position);
    }
}
```

```
public class AppleManager : MonoBehaviour
{
    [Header("Score Counter")]
    public int score;
    [Header("Score Text")]
    public TextMeshProUGUI textMeshPro;
    Unity İletisi | 0 başvuru
    private void Start()
    {
        score = 0;
    }
    Unity İletisi | 0 başvuru
    private void Update()
    {
        textMeshPro.text = "Score: " + score;
    }
}
```

```
public class AudioManager : MonoBehaviour
{
    [SerializeField] private AudioSource _musicSource;
    [SerializeField] private AudioSource _SFXSource;
    [Header("Sound Effects")]
    public AudioClip backgroundSound;
    public AudioClip playerRespawn;
    public AudioClip playerAttack;
    public AudioClip playerWalk;
    public AudioClip enemyAttack;
    public AudioClip enemyDie;
    public AudioClip apple;
    public static AudioManager instance;
    Unity İletisi | 0 başvuru
    private void Start()
    {
        DontDestroyOnLoad(gameObject);
        _musicSource.clip = backgroundSound;
        _musicSource.Play();
    }
    6 başvuru
    public void PlaySFX(AudioClip clip)
    {
        if (clip != null)
        {
            _SFXSource.PlayOneShot(clip);
        }
        else
        {
            Debug.LogWarning("Ses dosyası null olduğu için oynatılamadı.");
        }
    }
}
```

```
public class FinishPoint : MonoBehaviour
{
    public AudioManager audioManager;
    public GameManager gameManager;
    Unity İletisi | 0 başvuru
    private void Awake()
    {
        audioManager = GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();
        gameManager = GameObject.FindGameObjectWithTag("GameManager").GetComponent<GameManager>();
    }
    Unity İletisi | 0 başvuru
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            gameManager.NextLevel();
        }
    }

}
```

```
public class SceneController : MonoBehaviour
{
    public static SceneController instance;
    Unity İletisi | 0 başvuru
    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }
    }
    1 başvuru
    public void NextLevel()
    {
        SceneManager.LoadSceneAsync(SceneManager.GetActiveScene().buildIndex + 1);
    }
    0 başvuru
    public void LoadScene(string sceneName)
    {
        SceneManager.LoadSceneAsync(sceneName);
    }
}
```

```
public class ThornPlayece : MonoBehaviour
{
    [Header("Player Damage")]
    [SerializeField] private int damage;
    public HealthDisplay healthDisplay;
    Unity İletisi | 0 başvuru
    private void Awake()
    {
        healthDisplay = GameObject.FindGameObjectWithTag("GameManager").GetComponent<HealthDisplay>();
    }
    Unity İletisi | 0 başvuru
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            healthDisplay.TakeDamage(damage);
        }
    }
}
```

```
public class Thorns : MonoBehaviour
{
    [Header("Player Controller")]
    [SerializeField] private PlayerController playerController;
    public HealthDisplay healthDisplay;
    Unity İletisi | 0 başvuru
    private void Awake()
    {
        healthDisplay = GameObject.FindGameObjectWithTag("GameManager").GetComponent<HealthDisplay>();
        playerController = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
    }
    Unity İletisi | 0 başvuru
    private void Update()
    {
        if (healthDisplay._health <= 0)
        {
            PlayerDie();
        }
    }
    1 başvuru
    public void PlayerDie()
    {
        playerController.Die();
        healthDisplay._health = healthDisplay._maxHealth;
    }
    Unity İletisi | 0 başvuru
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.CompareTag("Player"))
        {
            healthDisplay._health = healthDisplay._maxHealth;
            playerController.Die();
        }
    }
}
```

DEĞERLENDİRME

Bu tez çalışması kapsamında geliştirilen mobil oyun projesi, oyun geliştirme sürecinde edinilen deneyimler ve elde edilen sonuçlar ile birlikte değerlendirilmiştir. Projenin geliştirme aşamaları incelenmiş ve yapılan çalışmalar literatürdeki benzer araştırmalarla karşılaştırılarak değerlendirilmiştir.

Oyunun tasarım sürecinde, karakter kontrolleri, düşman mekanikleri ve can barı gibi temel unsurların entegrasyonu önemli bir özenle gerçekleştirilmiştir. Oyunun oynanabilirliği ve kullanıcı deneyimi üzerinde yapılan testler ve alınan geri bildirimler dikkate alınarak iterasyonlar yapılmış ve oyunun geliştirilmesi sağlanmıştır.

Geliştirme sürecinde karşılaşılan zorluklar, özellikle teknik sorunlar ve dengeleme gereksinimleri, çeşitli araştırma ve çözüm yöntemleri kullanılarak başarıyla aşılmıştır. Bu süreçte edinilen deneyimler, mobil oyun geliştirme alanındaki bilgi ve yeteneklerin artmasına katkı sağlamıştır.

Oyunun kullanıcı testleri ve geri bildirimleri, projenin önemli bir aşamasını oluşturmuştur. Kullanıcıların oyun deneyimleri değerlendirilerek, oyunun güçlü yönleri ve iyileştirme alanları belirlenmiştir. Bu geri bildirimler, oyunun son halinin şekillenmesinde önemli rol oynamıştır.

Sonuç olarak, mobil oyun projesi, oyun geliştirme sürecinde edinilen deneyimlerin ve elde edilen sonuçların kapsamlı bir değerlendirmesini sunmaktadır. Proje sürecinde yaşanan zorluklar ve başarılar, mobil oyun geliştirme alanında çalışan öğrenci geliştiriciler için değerli bir öğrenme deneyimi olmuştur.

Kaynakça

Akın, E. (2008, Şubat). Elektronik Spor: Türkiye'deki Elektronik Sporcular Üzerine Bir Araştırma . Eskişehir. https://earsiv.anadolu.edu.tr/xmlui/bitstream/handle/11421/7016/462242.pdf?sequence=1

BTK. (2021, Haziran). BTK 2O2O Dijital Oyunlar Raporunu Yayınladı. https://www.btk.gov.tr/haberler/btk-2o2o-dijital-oyunlar-raporunu-yayinladi

Kızılkaya, E. (2010). Bilgisayar Oyunlarında İdeolojik Söylem ve Anlatı. İstanbul. https://www.proquest.com/openview/f7bf89a22ae29f05a6702cd089025177/1?pq-origsite=gscholar&cbl=2026366&diss=y

Öztürk, B. N., & Bülbül, S. (2022). Mobil Oyun Pazarında Marka Kimliği İncelemesi. https://dergipark.org.tr/en/download/article-file/2448153

TÜİK. (2021, Aralık). Çocuklarda Bilişim Teknolojileri Kullanım Araştırması. https://data.tuik.gov.tr/Bulten/Index?p=Cocuklarda-Bilisim-Teknolojileri-Kullanim-Arastirmasi-2021-41132

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı: Mustafa Taşan

Uyruğu: TC

Doğum Tarihi ve Yeri: 08/09/1999 Bakırköy/İSTANBUL

Tel: 0553 965 5530

Mail: mustafa.tsn@hotmail.com

EĞİTİM

Derece	Eğitim Birimi	Eğitim Tarihi
Lise	İOSB Meslek Lisesi	2013-2017
Ön Lisans	Beykent Üniversitesi	2019-2021
Lisans	Okan Üniversitesi	2022-2024