**KARADENİZ TEKNİK ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

****

**UNİTY3D ile MULTİPLAYER LABİRENT OYUNU**

**BİTİRME PROJESİ**

**Gökçenur ZENGİNAL**

**2017-2018 GÜZ DÖNEMİ**

**KARADENİZ TEKNİK ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**UNİTY3D ile MULTİPLAYER LABİRENT OYUNU**

**BİTİRME PROJESİ**

**Gökçenur ZENGİNAL**

|  |
| --- |
| **Bu projenin teslim edilmesi ve sunulması tarafımca uygundur.**  **Danışman : Öğr. Gör. Ömer ÇAKIR .........................** |

**2017-2018 GÜZ DÖNEMİ**

|  |  |  |
| --- | --- | --- |
| ieee.jpg | **IEEE Etik Kuralları**  **IEEE Code of Ethics** | **ieee.jpg** |

Mesleğime karşı şahsi sorumluluğumu kabul ederek, hizmet ettiğim toplumlara ve üyelerine en yüksek etik ve mesleki davranışta bulunmaya söz verdiğimi ve aşağıdaki etik kurallarını kabul ettiğimi ifade ederim:

1. Kamu güvenliği, sağlığı ve refahı ile uyumlu kararlar vermenin sorumluluğunu kabul etmek ve kamu veya çevreyi tehdit edebilecek faktörleri derhal açıklamak;
2. Mümkün olabilecek çıkar çatışması, ister gerçekten var olması isterse sadece algı olması, durumlarından kaçınmak. Çıkar çatışması olması durumunda, etkilenen taraflara durumu bildirmek;
3. Mevcut verilere dayalı tahminlerde ve fikir beyan etmelerde gerçekçi ve dürüst olmak;
4. Her türlü rüşveti reddetmek;
5. Mütenasip uygulamalarını ve muhtemel sonuçlarını gözeterek teknoloji anlayışını geliştirmek;
6. Teknik yeterliliklerimizi sürdürmek ve geliştirmek, yeterli eğitim veya tecrübe olması veya işin zorluk sınırları ifade edilmesi durumunda ancak başkaları için teknolojik sorumlulukları üstlenmek;
7. Teknik bir çalışma hakkında yansız bir eleştiri için uğraşmak, eleştiriyi kabul etmek ve eleştiriyi yapmak; hatları kabul etmek ve düzeltmek; diğer katkı sunanların emeklerini ifade etmek;
8. Bütün kişilere adilane davranmak; ırk, din, cinsiyet, yaş, milliyet, cinsi tercih, cinsiyet kimliği, veya cinsiyet ifadesi üzerinden ayırımcılık yapma durumuna girişmemek;
9. Yanlış veya kötü amaçlı eylemler sonucu kimsenin yaralanması, mülklerinin zarar görmesi, itibarlarının veya istihdamlarının zedelenmesi durumlarının oluşmasından kaçınmak;
10. Meslektaşlara ve yardımcı personele mesleki gelişimlerinde yardımcı olmak ve onları desteklemek.

IEEE Yönetim Kurulu tarafından Ağustos 1990’da onaylanmıştır.

**ÖNSÖZ**

Karadeniz Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü “Bitirme Projesi” olarak “UNİTY3D ile MULTİPLAYER LABİRENT OYUNU” konulu bir çalışma gerçekleştirilmiştir.

Öğrenim hayatım boyunca benden maddi manevi desteklerini esirgemeyen beni bugünlere getiren aileme, bu süreçte yanımda bulunan sevgili arkadaşlarıma ve proje boyunca yanımda olan danışmanım Öğr. Gör. Ömer ÇAKIR’a çok teşekkür ederim.

Adı SOYADI

Gökçenur ZENGİNAL

Trabzon 2017

**İÇİNDEKİLER**

|  |  |
| --- | --- |
|  | Sayfa No |
| IEEE ETİK KURALLARI | II |
| ÖNSÖZ | III |
| İÇİNDEKİLER | IV |
| ÖZET | VI |
| 1. GENEL BİLGİLER | 1 |
| 1.1. Oyun Motoru Nedir? | 1 |
| 1.2. Unity Nedir? | 1 |
| 1.2.1. Neden Tercih Edilmeli? | 2 |
| 1.2.2. Hangi Platformları Destekler? | 2 |
| 1.2.3. Hangi İşletim Sistemlerinde Çalışır? | 3 |
| 1.2.4. Hangi Programlama Dilleriyle Kullanılır? | 3 |
| 1.2.5. Unity ile Yapılan Bazı Oyunlar | 3 |
| 1.2.6. Teknik Tanımlamalar | 3 |
| 1.2.7. Nesneler | 5 |
| 1.2.8. Sınıflar | 5 |
| 1.3. 2D ve 3D Oyunlar Arasındaki Farklılıklar | 5 |
| 2. YAPILAN ÇALIŞMALAR | 6 |
| 2.1. Giriş | 6 |
| 2.2. 2D Platformer Oyunu | 6 |
| 2.2.1. Animasyon Eklenmesi | 8 |
| 2.2.2. Scriptlerin Eklenmesi | 9 |
| 2.2.2.1. Oyuncu Scripti | 9 |
| 2.2.2.2. havadaMi Scripti | 12 |
| 2.2.2.3. KameraTakip Scripti | 13 |
| 2.2.2.4. Hud Scripti | 13 |
| 2.2.2.5. Diken Scripti | 14 |
| 2.2.2.6. DusmanUzak Scripti | 14 |
| 2.2.2.7. AtesAraligi Scripti | 15 |
| 2.2.2.8. Mermi Scripti | 16 |
| 2.2.2.9. DusenPlatform Scripti | 16 |
| 2.2.3. Müzik Eklenmesi | 16 |
| 2.2.4. Oyunun Tamamlanması | 16 |
| 2.3. Multiplayer Labirent Oyunu | 17 |
| 2.3.1. Scriptlerin Eklenmesi | 18 |
| 2.3.1.1. OyuncuKontrolleri Scripti | 18 |
| 2.3.1.1. Mermi Scripti | 19 |
| 2.3.1.3. Saglik Scripti | 20 |
| 2.3.1.4. KamerayaBak Scripti | 21 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| 3. SONUÇLAR |  |
| 4. ÖNERİLER |  |
| 5. KAYNAKLAR |  |
| STANDARTLAR ve KISITLAR FORMU |  |

**ÖZET**

“Unity ile Multiplayer Labirent Oyunu” adlı bu çalışma Karadeniz Teknik Üniversitesi Bilgisayar Mühendisliği Anabilim Dalında Bitirme Projesi olarak hazırlanmıştır.

Projede Unity oyun motoru kullanılarak üç boyutlu engelleri aşarak labirent sonuna erişilmesi gereken bir oyun yapılmıştır. Bu oyundaki amaç önündeki engelleri aşarak labirentten çıkmaktır. Oyun multiplayer bir şekilde de oynanabilmektedir. Bu durumda oyuncu rakibinden önce labirent sonuna ulaşmalıdır. Bu proje pc tabanlı oyunlar teması çerçevesinde yapılmıştır.

Projede dil olarak C# kullanılmış, Unity’nin yayınladığı kendi eğitim videolarından faydalanılmıştır.

1. **GENEL BİLGİLER**
   1. **Oyun Motoru Nedir?**

Oyun motorları önceden planlanmış ve yazılmış veriler içerir. Oyun tasarlayan kişi de bu verileri kullanarak oyununu geliştirir. Özellikle farklı oyun türleri için geliştirilmiş farklı oyun motorları da bulunur. Oyun motorlarının var olmasının asıl sebeplerinden biri, her yeni oyunda yazılması gereken kodların büyük bir kısmı için hazır paketler sunuyor olmaları. Böylece iş yükünü hafifletmeye de yarıyorlar.

Oyun motorlarına örnek verecek olursak;

* Unreal Engine
* Unity 3D
* 3D GameStudio
* CryEngine 3
* Fox Engine

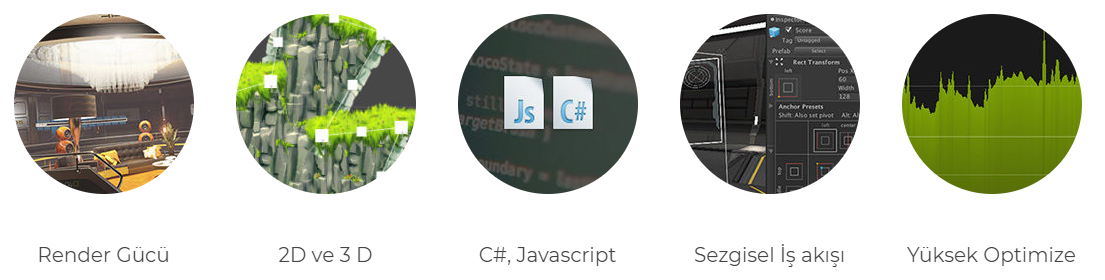
Bunların yanında yerli oyun motorları da mevcuttur;

* Actor
* CPAGE
* Yogurt3D
* Zinek Engine
* Piyodix
  1. **Unity Nedir?**

Unity3D, Unity Technologies şirketi tarafından geliştirilmiş ücretli bir oyun motorudur. İçerisinde programlama dilleri kullanılarak tanımlanan fonksiyonlar sayesinde az kod yazarak çok iş yapmak amaç edinilmektedir.

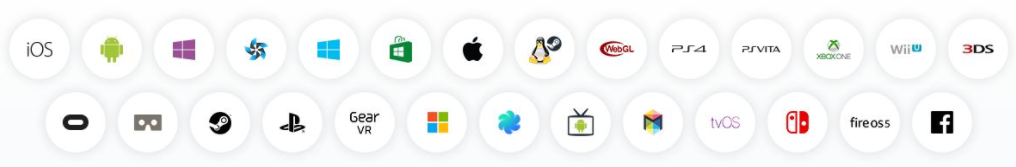
Unity ile oyun geliştirmeye başlamak için, Unity Technologies şirketinin resmi sayfasına gidip, Unity’i bilgisayarınıza indirip kurmanız gerekiyor.

* + 1. **Neden Tercih Edilmeli?**



* Kişisel kullanım olarak ücretsiz.
* Endüstri standardı konumunda, çıkan uygulamaların %70i Unity ile yapılıyor.
* Öğrenilmesi ve uzmanlaşması kolay
* Tek bir tıklama ile PC için yapılan oyun diğer platformlara çevrilebilir.
* Gelişmiş oyun motorlarının sunduğu fizik motoru, shader yazılımı, animation yapımını size sunar.
* Daha fazla dokümantasyona erişim şansı vardır.
* Oyununuza eklenmek üzere; ses, çevre elemanları (duvar, oda, ev vb.), ortam, obje gibi elemanlar seçip oyununuza ekleyebilirsiniz.
* Büyük bir topluluk
* Mobil cihaz performansı
* Hot Build süreleri çok kısa. Bu da geliştirme sürecini hızlandırır. Hot Build, play mod sırasında kodda yapılan değişikliklerin otomatik olarak yansımasıdır.
  + 1. **Hangi Platformları Destekler?**

Aslında çıkış amacı, oyunları bilgisayara yüklemenize gerek kalmadan Unity Web Player ile internet üzerinden oynatabilmekti. Böylece hem siz yeni bir bilgisayar almaya ihtiyaç duymayacaktınız hem de korsan oyunların önüne geçilmiş olunacaktı. Fakat Unity3D ekibi bununla yetinmedi ve her platforma destek verdi. Desteklenen platformlar;



* Windows
* OSX
* Linux
* Unity
* Webplayer bilşenleri
* Android
* iOS
* Windows Phone
* Blackberry
* Xbox One
* XBox 360
* PS4
* PS3
* PSP Vita
* Playstation Mobile
* WiiU
* Wi
  + 1. **Hangi İşletim Sistemlerinde Çalışır?**

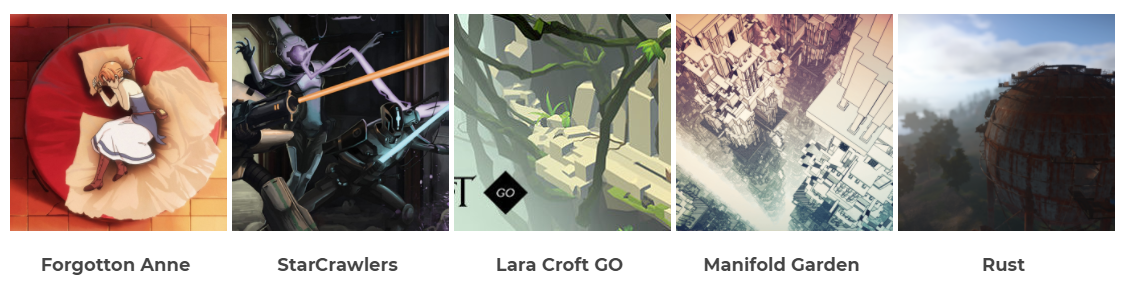
Unity sadece Windows ve OSX kurulu bilgisayarlarda çalışabiliyor.

* + 1. **Hangi Programlama Dilleriyle Kullanılır?**

Unity3D oyun motoru Unity Engines tarafindan C/C++ ile gelistirilmistir. Unity 4.0 sürümü ile C#, JavaScript, Boo ve DirectX dilleri ile oyun gelistirmenize destek vermektedir.

* + 1. **Unity ile Yapılan Bazı Oyunlar**

Battlestar Galactica Online, Temple Run, Temple Run 2, Dead Trigger 2, Bad Piggies, Bladeslinger, Legends of Aethereus



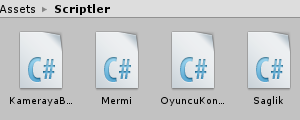
* + 1. **Teknik Tanımlamalar**

Build: Yazılımı çalıştırılmak istenen platforma göre derlenme işlemidir.

Prefab: Aynı nitelikte objeleri tekrar tekrar oluşturmak yerine bir defa oluşturup sahnede bu objeyi kullanma işlemidir.



Scripts: Sistem yönetimi için tipik yönetim işlemlerini yapmak için yazılan komut ya da komut dosyalarıdır. Unity, C# ve Javascript yazılım dillerini destekler.



Material: Objeleri kaplamak için kullanılır.



Asset Kits: Uygulama yapımını kolaylaştıracak ücretli ve ücretsiz varlık paketlerini ifade eder.

Sprites: Uygulamada nesnelere giydirilen resimlerdir. PNG formatındadır. Unity’nin sağladığı Asset Store’dan ücretli ve ücretsiz olarak indirilebilir.



Animations: Spriteların birleşimi ile oluşturulan animasyonlardır. Bir objeye atanır ve obje render edildikçe animasyonu da render edilir.



Animator: Animasyonların başlayabilmesi için kullanılan bir araçtır.

Scenes: Oyunda pek çok sahne olabilir: Başlangıç sahnesi, oyun sahnesi, oyun bitiş sahnesi gibi. Scenes bunları temsil eder.



* + 1. **Nesneler**

Particle System: Sis, ateş, patlama gibi tanecikli yapıların oluşturulmasında kullanılır.

Kamera: Oyunun istenilen açılardan görüntülenmesi için kullanılır. Normalde yalnızca sahneye bakarken bazı scriptler eklenerek oyuncuyu takip etmesi sağlanabilir.

Işık Nesneleri: Oyundaki aydınlatmayı sağlar. Böylece üç boyutlu olunlarda aydınlanma ve gölge oluşur. Üç çeşit ışık nesnesi vardır: Directional Light, Point Light, ve Spot Light.

Küp, Küre, Silindir

* + 1. **Sınıflar**

Collision2D Sınıfı: iki boyutlu fizik fonksiyonlarınca çarpışma detaylarını döndürür. Bunu sağlayan fonksiyonlar şunlardır: OnCollisionEnter2D, OnCollisionStay2D, OnCollisionExit2D

Collider2D Sınıfı: iki boyutlu oyunda kullanılan BoxCollider2D, CircleCollider2D gibi collider tiplerinin parent classıdır.

GameObject Sınıfı: Unity sahnesindeki tüm özelliklerle ilgili ana sınıftır.

Transform Sınıfı: sahneye eklenen nesnelerin hareketi, boyutu ve rotasyonu ile ilgilenen sınıftır. GameObject’in alt sınıfıdır.

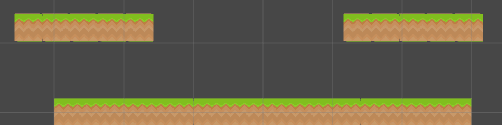
* 1. **2D ve 3D Oyunlar Arasındaki Farklılıklar**

2D , 2 boyut anlamına gelip eni ve boyu olan çizimlerdir. 3D ise 3 boyut anlamına gelip eni, boyu ve derinliği olan çizimlerdir.

1. **YAPILAN ÇALIŞMALAR**
   1. **Giriş**

Çalışmalara ilk olarak Mario benzeri bir *2D Platformer Oyun* yapılarak başlanıldı. Oyunda multiplayer özelliği eklenilmesiyle oyun Ateş ve Su benzeri *3D Multiplayer Labirent Oyunu*na dönüştürüldü.

* 1. **2D Platformer Oyunu**
* Oyun motorunun daha kullanışlı olması için gerekli klasörler oluşturuldu: Scriptler, Sahneler, Spritelar, Müzikler, Animasyonlar, Prefabler.
* Oyunda kullanılacak spritelar import edildi.
* Platform tasarımı yapıldı. Yapılan platformlara BoxCollider2D eklendi. Platform tekrar tekrar kullanılacağı için prefabler klasörüne eklendi.



Şekil 2.2.a. Platform görünümü

* Oyuncu tasarımı yapıldı, BoxCollider2D, Rigidbody2D (Fiziki etki olan tüm objelerde bulunmak zorundadır.) eklendi.



Şekil 2.2.b. Oyuncu görünümü

* Hud tasarımı yapıldı, Image şeklinde oyuna eklendi ve Text objesi eklenerek skor değeri gösterildi.

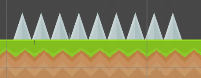


Şekil 2.2.c Hud görünümü



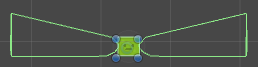
Şekil 2.2.d Skor görünümü

* Diken objelerinin tasarımı yapıldı. Yapılan dikenlere BoxCollider2D eklendi. Dikenler tekrar tekrar kullanılacağı için prefabler klasörüne eklendi.



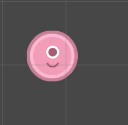
Şekil 2.2.e. Diken görünümü

* Düşman tasarımı yapıldı. Yapılan düşmana BoxCollider2D’li boş bir nesne eklendi, böylece düşmanın içinden geçilmesi engellenmiş oldu. Ateş edeceği mesafeyi ayarlamak için sağ ve sol olmak üzere iki tane trigger’lı ve PolygonCollider2D’li boş nesneler eklendi. Namlusunu ayarlamak için sağ ve sol olmak üzere iki tane boş nesne eklendi.



Şekil 2.2.f. Düşman görünümü

* Mermi objesinin tasarımı yapıldı. BoxCollider2D, Rigidbody2D eklendi. Mermiler tekrar tekrar kullanılacağı için prefabler klasörüne eklendi.



Şekil 2.2.g. Oyuncu mermi görünümü



Şekil 2.2.h. Düşman mermi görünümü

* Altın objesinin tasarımı yapıldı. Trigger’lı BoxCollider2D eklendi.



Şekil 2.2.i Altın görünümü

* Düşen platform tasarımı yapıldı. BoxCollider2D ve Rigidbody2D eklendi. Düşen platformlar tekrar tekrar kullanılacağı için prefabler klasörüne eklendi.



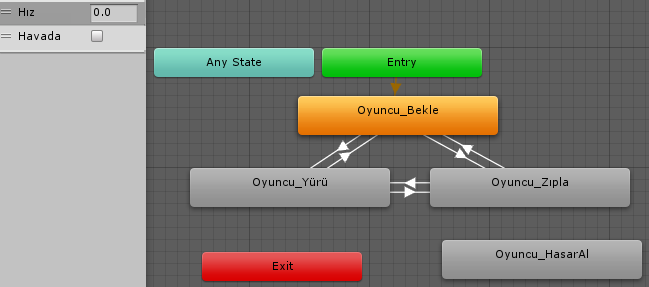
Şekil 2.2.j. Düşen platform görünümü

* Bitiş noktasının tasarımı yapıldı. BoxCollider2D eklendi.



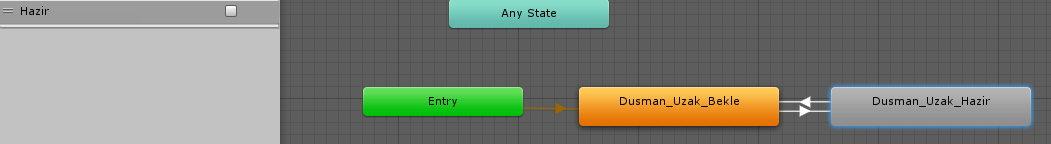
Şekil 2.2.k. Bitiş noktası görünümü

* + 1. **Animasyon Eklenmesi**
* Oyuncuya animasyonlar eklendi: Oyuncu\_Bekle, Oyuncu\_Yürü, Oyuncu\_Zıpla, Oyuncu\_HasarAl.
* Animasyonların başlayabilmeleri için oyuncuya animator’lar eklendi.



Şekil 2.2.1.a. Oyuncu animator akış diyagramı

* Düşmana animasyonlar eklendi: Dusman\_Uzak\_Bekle, Dusman\_Uzak\_Hazir.
* Animasyonların başlayabilmeleri için düşmana animator’lar eklendi.



Şekil 2.2.1.b. Düşman animator akış diyagramı

* Altın animasyonu eklendi: Altin.



Şekil 2.2.1.c. Altın animator akış diyagramı

* + 1. **Scriptlerin Eklenmesi**
       1. **Oyuncu Scripti**
    - Değişkenlerin atamaları gerçekleştirildi.
    - Oyuncuya hareket etme kabiliyeti verildi.
      * Oyuncuya hareket etme kabiliyeti verildi.

|  |
| --- |
| void FixedUpdate(){  float h = Input.GetAxis("Horizontal"); rb2d.AddForce(Vector2.right \* hiz \* h);  if (rb2d.velocity.x > maxHiz){  rb2d.velocity = new Vector2(maxHiz, rb2d.velocity.y);  }  if (rb2d.velocity.x < -maxHiz){  rb2d.velocity = new Vector2(-maxHiz, rb2d.velocity.y);  }  } |

* + - * + Kullanıcının yatay hareketi için *h* değişkeni tanımlandı.
        + Oyuncuya bir kuvvet etki ettiriliyor ve bu kuvvetle de oyuncu sağa ve sola hareket edebiliyor. Vector2.right, kullanıcıya sadece sağı gösteren bir vektör veriyor.
        + Oyuncunun sonsuz hızla hareket etmesini engellemek için if bloklarıyla kontrol edildi.
    - Animasyon scriptlerine geçiş yapıldı.

|  |
| --- |
| void Update(){  anim.SetFloat("Hız", Mathf.Abs(rb2d.velocity.x));  if (Input.GetAxis("Horizontal") < 0f){  transform.localScale = new Vector3(-1, 1, 1);  }  if (Input.GetAxis("Horizontal") > 0f){  transform.localScale = new Vector3(1, 1, 1);  }  } |

* + - * Oyuncunun her zaman sağa doğru yürür gibi görünmesini engellemek için if blokları yazıldı. Sprite’ın yönü değiştirilerek bu sorun ortadan kaldırılmış oldu.
      * Zıplama animasyonuna geçişi yapmak için, oyuncunun havada olup olmadığı kontrol edildi. Bunun için oyuncuya IsTrigger özellikli BoxCollider2D’li boş bir nesne eklendi. Bu sayede kutunun içine herhangi bir obje gelirse oyuncu yerde, gelmezse havada olduğu anlaşılıyor.
      * Ayrıca havada olup olmadığını kontrol etmek için yeni bir script (havadaMi) yazıldı.
      * Zıplama animasyonuna geçişi yapmak için, oyuncunun havada olup olmadığı kontrol edildi. Bunun için oyuncuya IsTrigger özellikli BoxCollider2D’li boş bir nesne eklendi. Bu sayede kutunun içine herhangi bir obje gelirse oyuncu yerde, gelmezse havada olduğu anlaşılıyor.
      * Ayrıca havada olup olmadığını kontrol etmek için yeni bir script (havadaMi) yazıldı.

|  |
| --- |
| anim.SetBool("Havada", havada); |

* + - * Oyuncunun zıplamasını kontrol etmek için Update() methodu içine;

|  |
| --- |
| if (Input.GetButtonDown("Jump")){  if (!havada){  rb2d.AddForce(Vector2.up \* ziplamaGucu);  ikiliZiplamaYapabilir = true;  }  else  {  if(ikiliZiplamaYapabilir){  rb2d.velocity = new Vector2(rb2d.velocity.x, 0);  rb2d.AddForce(Vector2.up \* ziplamaGucu / 1.5f);  ikiliZiplamaYapabilir = false;  }  }  } |

* + - * + Yerdeyken zıplamaya basıldığında oyuncuya ikili zıplama şansı veriliyor. Havadayken zıplamaya çalışılırsa, ilk zıplayışına oranla daha düşük kuvvetle zıplama şansı veriliyor ve bir daha zıplama hakkı verilmiyor.
    - Oyuncuya başlangıçta belirli bir can veriliyor ve oyuncu zarar gördükçe bu can değerinde azalma gerçekleştiriliyor, oyuncu topladığı nesnelerle canını arttırabilir bu can değerinin maksimum canı geçmemesi için if bloğuyla kontrol gerçekleştiriliyor. Bu işlemler Update() metodunda çağrılıyor.

|  |
| --- |
| if (saglik <= 0){  Oldur();  }  if (saglik >= maxSaglik){  saglik = maxSaglik;  } |

* Oyuncu öldüğünde Oldur() metodu çağrılıyor ve oyun yeniden başlamış oluyor.
* Oyuncuyu kalan canı hakkında bilgilendirmek için yeni bir script (Hud) yazıldı.
  + - Oyuncuya zarar vermek için HasarAl() adında bir metot tanımlaması yapıldı. Diğer scriptlerden metoda ulaşabilmek için public olarak oluşturuldu.

|  |
| --- |
| public void HasarAl(int hasar){  saglik -= hasar;  gameObject.GetComponent<Animation>().Play("Oyuncu\_HasarAl");  } |

* Bu metot her çağırıldığında verilen hasar değeri kadar sağlıktan düşüş yapılıyor.
* Dikenlerin oyuncuya hasar verilmesi için yeni bir script (Diken) yazıldı.
* Aynı zamanda oyuncunun hasar aldığının belli olması için *Animator* eklendi. Oluşturulan animator bileşenine ulaşıldı.
  + - Oyuncu hasar aldığı zaman pozisyonunun değişmesi için bir metot tanımlandı.

|  |
| --- |
| public IEnumerator GeriItme(float itmeSuresi, float itmeGucu, Vector3 itmeYonu){  float zamanlayici = 0;  rb2d.velocity = new Vector2(rb2d.velocity.x, 0);  while (itmeSuresi > zamanlayici){  zamanlayici += Time.deltaTime;  rb2d.AddForce(new Vector3(itmeYonu.x \* -200, itmeYonu.y \* itmeGucu, transform.position.z));  }  yield return 0;  } |

* + - * Metot zamanlayıcı içerdiği için IEnumerator ile çağırıldı.
      * IEnumerator fonksiyonlarının sonuçlanması için yield return 0 kullanılır.
    - Oyuncu altın topladıkça skorunun artması ve bölüm sonuna geldiği zaman oyunun tamamlanması için;

|  |
| --- |
| void OnTriggerEnter2D(Collider2D diger){  if (diger.CompareTag("Altin")){  Destroy(diger.gameObject);  skor++;  }  if (diger.CompareTag("BolumSonu")){  SceneManager.LoadScene(0);  }  } |

* + - 1. **havadaMi Scripti**
* Değişkenlerin atamaları gerçekleştirildi.
* Oyuncuya eklenen boş nesneye değen herhangi bir şey varsa;

|  |
| --- |
| void OnTriggerEnter2D(Collider2D col){  oyuncu.havada = false;  } |

* Oyuncuya eklenen boş nesneye değen herhangi bir şey yoksa;

|  |
| --- |
| void OnTriggerExit2D(Collider2D col) {  oyuncu.havada = true;  } |

* Oyuncunun yerde olduğunu belli etmek için;
  + - 1. **KameraTakip Scripti**
* Bölüm boyutu büyüdükçe oyuncu kamera açısından çıkıyordu. Bunu engellemek için yeni bir script yazıldı.
* Değişkenlerin atamaları gerçekleştirildi.
* Hareketle alakalı olduğu için kodlar FixedUpdate() methodunda çağrıldı.

|  |
| --- |
| void FixedUpdate(){  float xCoord = Mathf.SmoothDamp(transform.position.x,oyuncu.transform.position.x,ref kameraHizi.x,yumusatX);  float yCoord = Mathf.SmoothDamp(transform.position.y,oyuncu.transform.position.y,ref kameraHizi.y,yumusatY);  transform.position = new Vector3(xCoord,yCoord,transform.position.z);  transform.position = new Vector3(transform.position.x, Mathf.Clamp(transform.position.y, minKameraPozisyonu.y, maxKameraPozisyonu.y), transform.position.z);  } |

* + Kameranın nerede olduğunu tutmak için değişkenler tanımlandı ve bu değer oyuncu ile kamera arasında bir yumuşatma yaparak kameranın yeni konumunu tutmuş oluyor.
  + Kameranın yeni pozisyonu oluşturulan vektöre eşitlendi.
  + Kamera konumunun belli değerler dışına çıkması engellenmiş oldu.
    - 1. **Hud Scripti**
* Hud ile ilgili olaylar Main Camera içinde ayarlandı.
* Değişkenlerin atamaları gerçekleştirildi.
* Oyuncunun sağlık değeri kontrol edildi.

|  |
| --- |
| void Update (){  SaglikHUD.sprite = SaglikGorselleri[oyuncu.saglik];  skorText.text = oyuncu.skor+""; //int stringe dönüyor +""  } |

* + Hud sprite’ında sağlık değerine göre değişiklikler yapıldı.
    - 1. **Diken Scripti**
* Değişkenlerin atamaları gerçekleştirildi.

|  |
| --- |
| void OnTriggerEnter2D(Collider2D diger){  if (diger.CompareTag("Player")){  oyuncu.HasarAl(2);  StartCoroutine(oyuncu.GeriItme(0.01f,200,oyuncu.transform.p osition));  }  } |

* + Scriptin eklendiği objenin trigger’ına oyuncunun giriş yaptığı her zaman bu metot otomatik olarak çağırılıyor.
  + Oyuncu hasar aldığı zaman pozisyonunun değişmesi için gerekli olan kod parçası eklendi.
    - 1. **DusmanUzak Scripti**
* Değişkenlerin atamaları gerçekleştirildi.
* Animator ile ilgili kodlar *Awake()* metodunda çağırılır.

|  |
| --- |
| void Awake(){  anim = gameObject.GetComponent<Animator>();  } |

* Animator’daki değişken ayarlandı.

|  |
| --- |
| void Update (){  anim.SetBool("Hazir", hazir);  MesafeHesapla();  } |

* + Oyuncu ve düşman arasındaki mesafeyi ayarlamak için MesafeHesapla() metodu çağırıldı.

|  |
| --- |
| void MesafeHesapla(){  mesafe = Vector3.Distance(transform.position, hedef.transform.position);  if (mesafe < hazirlanmaMesafesi){  hazir = true;  }  else{  hazir = false;  }  } |

* + - * + Oyuncu ve düşman arasındaki mesafe belirlenen mesafeden küçükse düşman ateş etmeye hazırlanıyor.
* Düşmanın ateş etmesi için AtesEt() metodu tanımlandı.

|  |
| --- |
| public void AtesEt(bool sagaAtesEt){  mermiZamanlayicisi += Time.deltaTime;  if (mermiZamanlayicisi >= mermiSikligi){  Vector2 mermiYonu = hedef.transform.position - transform.position;  mermiYonu.Normalize();  if (sagaAtesEt == true){  GameObject mermiKopyasi;  mermiKopyasi = Instantiate(mermi,namluSag.transform.position,namluSag.transform.rotation);  mermiKopyasi.GetComponent<Rigidbody2D>().velocity = mermiYonu \* mermiHizi;  mermiZamanlayicisi = 0;  }  if (sagaAtesEt == false){  GameObject mermiKopyasi;  mermiKopyasi = Instantiate(mermi, namluSol.transform.position, namluSol.transform.rotation);  mermiKopyasi.GetComponent<Rigidbody2D>().velocity = mermiYonu \* mermiHizi;  mermiZamanlayicisi = 0;  }  }  } |

* + Prefab kullanarak sıfırdan bir obje oluşturmak istendiği zaman Instantiate() metodu kullanılır.
    - 1. **AtesAraligi Sciripti**
* Değişkenlerin atamaları gerçekleştirildi.
* Düşman alanına oyuncunun girdiği süre boyunca, düşmanın ateş etmeye devam etmesi için bir metot tanımlandı.

|  |
| --- |
| void OnTriggerStay2D(Collider2D diger){  if (diger.CompareTag("Player")){  if(sagAtesAraligi){  dusmanUzak.AtesEt(true);  }  else{  dusmanUzak.AtesEt(false);  }  }  } |

* + - 1. **Mermi Scripti**

Mermi oyuncuyla temas ettiği zaman oyuncuya zarar vermesi için bir metot yazıldı.

|  |
| --- |
| void OnTriggerEnter2D(Collider2D diger){  if (diger.isTrigger != true){  if (diger.CompareTag("Player")){  diger.GetComponent<Oyuncu>().HasarAl(1);  }  Destroy(gameObject); // oyuncu kayboluyor  }  } |

* + - 1. **DusenPlatform Scripti**
* Değişkenlerin atamaları gerçekleştirildi.
* Platforma temas eden obje oyuncu ise platformun düşmesi sağlandı.

|  |
| --- |
| void OnCollisionEnter2D(Collision2D diger){  if (diger.collider.CompareTag("Player")){  StartCoroutine(Dus());  }  } |

* + Platformun düşmesi için *Dus()* metodu tanımlandı.

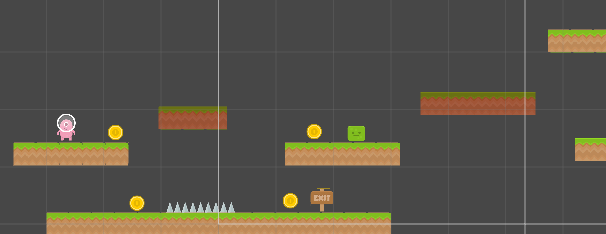
|  |
| --- |
| IEnumerator Dus(){  yield return new WaitForSeconds(gecikme);  rb2d.isKinematic = false;  GetComponent<Collider2D>().isTrigger = true;  yield return 0;  } |

* + 1. **Müzik Eklenmesi**

Müziğin her zaman çalması istendiği için Main Camera’ya eklenmesi tercih edildi. Audio Source bileşeni ile seçilen müzik oyuna eklenmiş oldu.

* + 1. **Oyunun Tamamlanması**

Son olarak bitiş noktası da eklendi ve oyun tamamlanmış oldu.



Şekil 2.2.4. Oyun görüntüsü

* 1. **Multiplayer Labirent Oyunu**
* Oyun motorunun daha kullanışlı olması için gerekli klasörler oluşturuldu: Scriptler, Sahneler, Prefabler, Materyeller, Builder, Spritelar.
* Oyunda kullanılacak spritelar import edildi.
* Oyuncu tasarımı için boş bir nesne oluşturuldu ve CapsuleCollider eklendi. Oyuncunun vücudunun tasarımı yapıldı ve BoxCollider eklendi. Oyuncunun baktığı yönün anlaşılması için vücuduna boş bir nesne eklendi. Daha sonra oyuncu prefabine ihtiyaç duyulacağı için, prefabler klasörüne eklendi.



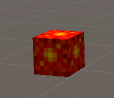
Şekil 2.3.a. Oyuncu görünümü

* Multiplayer oyununu yönetebilmek için *Ağ Yöneticisi* adında boş bir nesne oluşturuldu. Bu nesneye Network Manager, Network Manager Hud bileşenleri eklendi.
* Oyuncuya *Network Identity* bileşeni eklendi. Bu sayede multiplayer oyunda oyuncunun kontrol etmesi gereken bir obje olduğu belirtilmiş oldu. Burada *Local Player Authority* seçildiği zaman, oyuncu objesinin oyunculardan biri tarafından kontrol edildiği belirtildi. Yani kontrol eden bilgisayar değil, server değil.
* Oyuncular server tarafından spawn edileceği için, Ağ Yöneticisine oyuncunun prefabi eklendi.
* Oyuncu için bir silah tasarımı yapıldı. Silah oyuncuya ait gözlük olacak şekilde tasarlandı. Silahın ateş edeceği noktayı belirlemek için boş bir nesne eklendi.



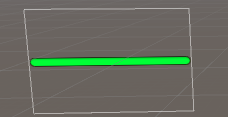
Şekil 2.3.b. Silah görüntüsü

* Mermi tasarımı yapıldı ve RigidBody eklendi. Prefablere kaydedildi. Network Identity, Network Transform bileşenleri eklendi.



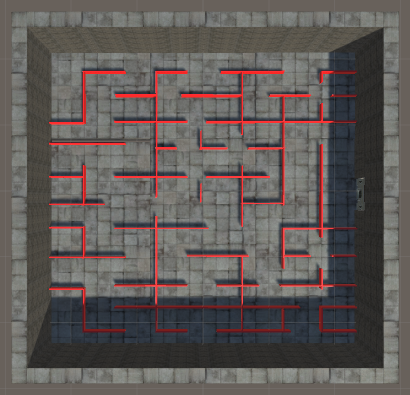
Şekil 2.3.c. Mermi görüntüsü

* Oyuncunun sağlık seviyesinin görülebileceği bir nesne yapıldı.



Şekil 2.3.d. Sağlık çubuğu görünümü

* Oyun alanının tasarımı yapıldı. Yapılan alana BoxCollider eklendi.



Şekil 2.3.e. Oyun alanı görünümü

* + 1. **Scriptlerin Eklenmesi**
       1. **OyuncuKontrolleri Scripti**
* Değişkenlerin atamaları gerçekleştirildi.
* Klavyeden gelen girdileri alarak oyuncuya hareket etme özelliği eklendi.

|  |
| --- |
| void Update(){  float x = Input.GetAxis("Horizontal") \* Time.deltaTime \* 150.0f;  float z = Input.GetAxis("Vertical") \* Time.deltaTime \* 3.0f;  transform.Rotate(0, x, 0);  transform.Translate(0, 0, z);  } |

* + Tuşa basılı olduğu süre boyunca geçerli olması için *Time.deltaTime* kullanıldı.
* Tüm oyuncularda aynı script olduğu için, biri hareket ederken diğer oyuncular da aynı harekette bulunacaklardır. Bunu önlemek için Update() metoduna aşağıdaki kod eklendi. Aynı zamanda oyuncuya *Network Transform* bileşeni eklendi.

|  |
| --- |
| if (!isLocalPlayer){  return;  } |

* Update() metodu güncellenerek oyuncuya ateş etme özelliği verildi.

|  |
| --- |
| if (Input.GetKeyDown(KeyCode.Space)){  CmdAtesEt();  } |

* Oyuncunun ateş edebilmesi için yeni bir metot tanımlaması yapıldı.

|  |
| --- |
| [Command]  void CmdAtesEt(){  GameObject mermi = Instantiate(mermiPrefab, namlu.position, namlu.rotation);  mermi.GetComponent<Rigidbody>().velocity = mermi.transform.forward \* 6.0f;  NetworkServer.Spawn(mermi);  Destroy(mermi, 2);  } |

* + - * + Mermi objesi oluşturuldu. Oluşturulan objeye belirli bir sürat verildi ve belirli bir süre sonra yok olması sağlandı.
        + AteşEt() metodunun clientta değil de serverda çağırılıp diğer clientlara dağıtılması istendiği için metodun başına *[Command]* komutu yazıldı.
* Mermilerin herhangi bir nesneye çarptığı zaman kaybolması için yeni bir script (Mermi) yazıldı.
* Tüm oyuncuların bilgisayarlarında farklı renkte görünmeleri için bir metot yazıldı.

|  |
| --- |
| public override void OnStartLocalPlayer(){  GetComponent<MeshRenderer>().material.color = Color.blue;  } |

* + - 1. **Mermi Scripti**
* Mermi herhangi bir objeyle temas ettiği zaman mermi yok ediliyor.

|  |
| --- |
| void OnCollisionEnter(Collision col){  Destroy(gameObject);  } |

* Yazılan metoda gerekli kodlar eklenerek vurulan nesnenin zarar görmesi sağlandı.

|  |
| --- |
| GameObject vurulanObje = col.gameObject;  Saglik saglik = vurulanObje.GetComponent<Saglik>();  if (saglik != null){  saglik.HasarAl(5);  } |

* + - 1. **Saglik Scripti**
* Değişkenlerin atamaları gerçekleştirildi.
* Oyuncu zarar gördüğü zaman sağlık değerinin azalması için yeni bir metot yazıldı.

|  |
| --- |
| public void HasarAl(int hasar){  saglik -= hasar;  if (saglik <= 0){  saglik = maxSaglik;  RpcRespawn();  }  } |

* Sağlık değerindeki değişikliği sağlık çubuğunda göstermek için;

|  |
| --- |
| void SaglikCubugunuYonet(int saglik){  saglikCubugu.sizeDelta = new Vector2(saglik \* 2, saglikCubugu.sizeDelta.y);  } |

* Değişikliklerin bütün clientlara senkron bir şekilde iletilmesi istendiği için sağlık değişkeni tanımlaması düzenlendi.

|  |
| --- |
| [SyncVar(hook = "SaglikCubugunuYonet")] public int saglik = maxSaglik; |

* HasarAl() metodunun sadece serverda çalışması istendiği için metoda aşağıdaki kod parçası eklendi.

|  |
| --- |
| if (!isServer){  return;  } |

* Ölen oyuncuların başlangıç noktalarında yeniden doğmaları sağlandı.

|  |
| --- |
| [ClientRpc]  void RpcRespawn(){  if (isLocalPlayer){  Vector3 spawnNoktasi = Vector3.zero;  if (spawnNoktalari != null && spawnNoktalari.Length > 0){  spawnNoktasi = spawnNoktalari[Random.Range(0, spawnNoktalari.Length)].transform.position;  }  }  } |

* + Metot serverda dönüyor ancak bütün clientlarda da çağrılmış oluyor. Bütün oyuncuların aynı anda respawn olmasını engellemek için if bloğu içinde kodlar yazıldı.
  + Oyuncunun sürekli aynı noktada doğmasını engellemek için belirli noktalar belirlendi. Oyuncu rastgele bunların birinde yeniden doğuyor.
    - 1. **KamerayaBak Scripti**

Oyuncular sürekli hareket edecekleri için sağlık çubuğu da 3Boyutlu bir ortam olduğu için görünürlüğünü kaybedebilir. Bunu önlemek için bir script yazıldı.

Update() metodu içinde tanımlanan kod sayesinde bu sorun ortadan kaldırılmış oldu.

|  |
| --- |
| transform.LookAt(Camera.main.transform); |

**STANDARTLAR ve KISITLAR FORMU**

Projenin hazırlanmasında uyulan standart ve kısıtlarla ilgili olarak, aşağıdaki soruları cevaplayınız.

1. Projenizin tasarım boyutu nedir? (Yeni bir proje midir? Var olan bir projenin tekrarı mıdır? Bir projenin parçası mıdır? Sizin tasarımınız proje toplamının yüzde olarak ne kadarını oluşturmaktadır?)

|  |
| --- |
|  |

1. Projenizde bir mühendislik problemini kendiniz formüle edip, çözdünüz mü? Açıklayınız.

|  |
| --- |
|  |

1. Önceki derslerde edindiğiniz hangi bilgi ve becerileri kullandınız?

|  |
| --- |
|  |

1. Kullandığınız veya dikkate aldığınız mühendislik standartları nelerdir? (Proje konunuzla ilgili olarak kullandığınız ve kullanılması gereken standartları burada kod ve isimleri ile sıralayınız).

|  |
| --- |
|  |

1. Kullandığınız veya dikkate aldığınız gerçekçi kısıtlar nelerdir? Lütfen boşlukları uygun yanıtlarla doldurunuz.

a) Ekonomi

|  |
| --- |
|  |

b) Çevre sorunları:

|  |
| --- |
|  |

c) Sürdürülebilirlik:

|  |
| --- |
|  |

d) Üretilebilirlik:

|  |
| --- |
|  |

e) Etik:

|  |
| --- |
|  |

f) Sağlık:

|  |
| --- |
|  |

g) Güvenlik:

|  |
| --- |
|  |

h) Sosyal ve politik sorunlar:

|  |
| --- |
|  |