İSKENDERUN TEKNİK ÜNİVERSİTESİ

2D PLATFORM OYUNU YAPIMI

FİNAL RAPORU

MUHAMMET HÜSEYİN ONGUN

ARALIK 2024

TEŞEKKÜRLER

Bu çalışmanın hazırlanmasında bana destek olan herkese teşekkürlerimi sunarım. Öncelikle, bu süreçte bilgi ve tecrübeleriyle bana yol gösteren danışman hocam Hasa Kuzu'ya teşekkür ederim. Ayrıca, beni her zaman destekleyen ve motive eden aileme, arkadaşlarıma ve bu projeye katkı sağlayan tüm sevdiklerime minnettarım. Proje boyunca öğrendiklerim ve edindiğim deneyimler, hem akademik hem de kişisel gelişimime büyük katkı sağladı. Bu süreçte bana ilham veren, desteklerini esirgemeyen tüm öğretmenlerime teşekkür ederim.

ÖZET

Bu rapor, Foxlandia adlı oyun projesinin geliştirilmesi sürecini ve kullanılan temel oyun mekanizmalarını incelemektedir. Proje, etkileşimli bir oyun deneyimi sunmayı amaçlamakta ve oyuncuya çeşitli oyun dinamikleriyle zorluklar sunmaktadır. Oyunda, oyuncunun çevreyle etkileşime girebileceği öğeler, parkurlar, düşman yapay zekası ve fiziksel mekanizmalar gibi unsurlar yer almaktadır. Bu unsurlar, oyuncunun oyun dünyasında daha aktif bir rol üstlenmesini ve stratejik kararlar almasını teşvik etmektedir. Oyun, oyuncuya sadece zorluklar sunmakla kalmayıp, aynı zamanda etkileşimli öğelerle oyun deneyimini derinleştirmeyi amaçlamaktadır. Proje, oyun tasarımı, programlama ve etkileşimli oyun öğeleri kullanarak, oyun dünyasında daha dinamik ve sürükleyici bir deneyim sunmayı hedeflemiştir.

İÇİNDEKİLER

Sayfa

GİRİŞ

Oyunlar, günümüzde hem eğlence hem de eğitim alanlarında önemli bir araç haline gelmiştir. Bu tez, etkileşimli ve dinamik bir oyun deneyimi yaratmayı amaçlayan bir proje üzerine odaklanmaktadır. Oyun geliştirme süreci, birçok farklı disiplinin bir araya geldiği karmaşık bir süreçtir. Grafik tasarımı, programlama, yapay zeka, oyun mekaniği ve kullanıcı deneyimi gibi unsurlar, başarılı bir oyun projesi için bir arada çalışmalıdır. Bu projede, oyuncuların etkileşimde bulunabileceği çeşitli mekanizmalar ve dinamik düşman davranışları gibi özellikler üzerine yoğunlaşılmıştır. Proje, oyuncunun oyun dünyasında aktif bir rol üstlenmesini sağlayacak, stratejik kararlar almasını teşvik edecek şekilde tasarlanmıştır. Bu raporda, kullanılan yöntemler ve araçlar açıklanarak, oyun geliştirme sürecinin her aşaması detaylı bir şekilde ele alınacaktır.

1.BÖLÜM

LİTERATÜR TARAMASI

2D PLATFORM OYUNU

1.1. NEDİR?

2D platform oyunları, oyuncunun karakterini ekranın yatay veya dikey yönde hareket ettirerek, platformlardan zıplarken ve zorlukları aşarken ilerlemesini sağlayan oyun türüdür. Genellikle macera, bulmaca ve aksiyon öğelerini bir araya getirir.

1.2. TARİHÇESİ

2D platform oyunlarının kökeni, 1980'lerin başlarında Atari'nın “Donkey Kong”, “Space Invaders” ve “Pac-Man” gibi oyunlarıyla başlar. Bu tür, 1985'te Nintendo'nun "Super Mario Bros" oyunu ile popülerlik kazanmış ve oyun dünyasında önemli bir yer edinmiştir. 2000’ler ve sonrası, bağımsız geliştiricilerin 2D platform oyunlarına yönelmesiyle birlikte, “Celeste” ve “Hollow Knight” gibi oyunlarla çeşitlenmiştir.

1.3. ÇEŞİTLERİ

2D platform oyunları çeşitli alt türler içerir:

1.3.1. Klasik Platform Oyunları

Temel zıplama ve atlayış mekanikleri ile öne çıkan oyunlardır.

1.3.2. Açık Dünya Platform Oyunları

Oyuncunun büyük bir dünyada serbestçe hareket etmesine olanak tanıyan oyunlardır.

1.3.3. Pikselli Platform Oyunları

Retro tarzda, düşük çözünürlüklü grafiklerle tasarlanmış oyunlar.

1.4. NASIL ÇALIŞIR?

2D platform oyunları, genellikle bir oyun motoru kullanılarak geliştirilir. Bu motorlar, fizik motorları, animasyon sistemleri ve oyun mekaniği için gerekli araçları sağlar. Tasarımcılar, seviyeleri, karakterleri ve düşmanları programlayarak oyuncunun etkileşimde bulunabileceği bir dünya yaratmayı hedefler. Oyun mekaniği, kullanıcı girişi (klavye, fare veya kontrolcü) ile tetiklenir.

1.5. PROJE YAPMAK İÇİN GEREKLİ YAZILIM VE DONANIMLAR

1.5.1. Donanım

İşlemimiz için gerekli donanımlara ihtiyaç duyarız:

1.5.1.1. Bilgisayar

Proje yapmak için bir bilgisayar gerekir, tabi buna dahil içerisinde orta seviye bir ekran kartı ve işlemcisi olması mecburidir.

1.5.1.2. Monitör

Bilgisayarımızdan görüntü almak için monitör gerekmektedir.

1.5.1.3. Ses Ekipmanı

Projemizde kullanacağımız sesleri bize iletmeye yarar. Örneğin; Kulaklık veya Hoparlör.

1.5.2. Yazılım

İşlemimiz için sadece donanım yetmez donanımla birlikte kullanacağımız, bize oyun yapmamızı sağlayacak bazı yazılım ve uygulamalara ihtiyaç duyarız.

1.5.2.1. Oyun Motorları

Oyunumuzu yapmak için bir oyun motoru gerekir. Oyun motorlarından birkaçı şunlardır; Unity, Unreal Engine, Godot ve Construct 3

1.5.2.2. Grafik Tasarım Yazılımları

Oyunumuzun yapımında eklememiz gereken görsel unsurlar vardır. Bunlar; karakterler, düşmanlar, çevre, arka plan, efekt ve animasyonlar şeklindedir. Grafik tasarım yazılımlarından birkaçı ise şunlardır; Aseprite, Photoshop ve GIMP.

1.5.2.3. Ses düzenleme yazılımları

Oyunumuza, animasyon sesi, tema müziği, efekt sesi ve yürüme sesi gibi sesler eklememiz gerektiğinde ses düzenleme yazılımlarına ihtiyaç duyarız. Bunlardan birkaçı ise şunlardır; FL Studio, Audacity ve Auditool.

1.5.2.4. Kodlama ve Scripting Yazılımları

Oyunun temel işleyişini oluşturan bu yazılımlar, oyunumuzdaki karakterler hareketlerini, düşman davranışlarını, oyunun fizik motorunun işleyişini, oyun içindeki veri yönetimini, ses efektleri ve kullanıcı arayüzü etkileşimlerini oluşturur ve denetler. Bu yazılımlardan, Unity tarafından ve Visual Studio aracılığıyla kullanılan bazı diller ise şunlardır; C#, C++, Phyton.

1.6. BU KONUDA YAPILMIŞ ÖRNEK PROJELER VE DETAYLARI

Tarihe ve 2D platform oyunu sektörüne damgasını vurmuş bazı projeler vardır;

1.6.1. Super Mario Bros

Nintendo tarafından geliştirilen, 2D platform oyunlarının klasik örneği olan Super Mario Bros, oyuncuları Mario karakteriyle prensesi kurtarmak için farklı dünyalarda zıplayarak, düşmanlarla savaşarak ve engelleri aşarak ilerler. Oyunun basit mekanikleri ve bağımlılık yapıcı oynanışı, onu dünya çapında popüler hale getirmiştir.

1.6.2. Sonic the Hedgehog

SEGA tarafından geliştirilen, hız odaklı platform oyunu olan Sonic the Hedgehog, oyunculara hızlı tempolu bir oynanış sunar. Sonic, düşman Dr. Robotnik'i yenmek için çeşitli seviyelerde koşar, zıplar ve spin dash yapar. Oyunun renkli grafikleri ve etkileyici müzikleri, onu unutulmaz kılar.

1.6.3. Rayman Legends

Ubisoft tarafından geliştirilen Rayman Legends, çarpıcı görselleri ve yaratıcı seviyeleri ile bilinen bir platform oyunudur. Oyuncular, Rayman ve arkadaşlarıyla farklı dünyalarda zıplayarak, düşmanlarla savaşarak ve bulmacaları çözerek ilerler. Oyunun kooperatif oynanışı ve çeşitli karakter seçenekleri, oyunu daha eğlenceli yapar.

1.6.4. Celeste

Matt Makes Games tarafından geliştirilen Celeste, zorlu harita dizaynı ve derin bir hikaye ile öne çıkan modern bir 2D platform oyunudur. Oyuncu, Madeline adlı karakteri kontrol ederek Celeste Dağı'na tırmanmaya çalışır. Oyunun mükemmel kontrolleri ve etkileyici hikayesi, çok sayıda ödül kazanmasını sağlamıştır.

1.7. PROJEMİN ÖZGÜN YANLARI

2D platform oyun türünün kökeni 1980'lerin başında dayandığı ve bu sektörün çok yönlü ilerleme kat ettiğinden dolayı ve özellikle bu işlere yeni başlayan bir öğrenci olduğumdan dolayı mevcut standartları yakalama şansım olmayacak. Kendi projemde temel karakter kabiliyetleri, ufak çaplı bulmaca ve düşman öldürme unsurları bulunacaktır. Grafiksel yönden zayıf olduğum için karakter modellemesi, çevre, arka plan, düşman modellemeleri ve animasyonlar için hazır assetler kullanacağım. Oyunumda ses efektleri de bulunacaktır. Bana özgün bir yanı olmayacaktır.

2.BÖLÜM

UNİTY OYUN MOTORU

2.1. UNİTY KURULUMU

2.1.2. Unity Hub'ı indirme

Unity projelerinizi ve sürümlerinizi yönetmek için Unity Hub'ı Unity'nin resmi web sitesi (https://unity.com/download) üzerinden indirin.

2.1.3. Kurulum

İndirme tamamlandığında UnityHubSetup.exe dosyasını çalıştırın ardından lisans sözleşmesini kabul edin, kurulum dizinini seçin ve kurulumu tamamlayın.

2.1.4. Unity Hub'a giriş

Unity Hub'ı açarak giriş ekranına ulaşın. Bir hesabınız varsa "Sign in" seçeneğiyle giriş yapın. Yoksa "Create account" ile yeni bir hesap oluşturun.

2.1.5. Lisans ekleme

Giriş yaptıktan sonra karşılaşılan lisans uyarısında, sağ üstteki "Manage licenses" seçeneğine tıklayın. "Add" butonuna basarak "Get a free personal license" seçeneği ile ücretsiz bir lisans ekleyin.

2.1.6. Editör indirme

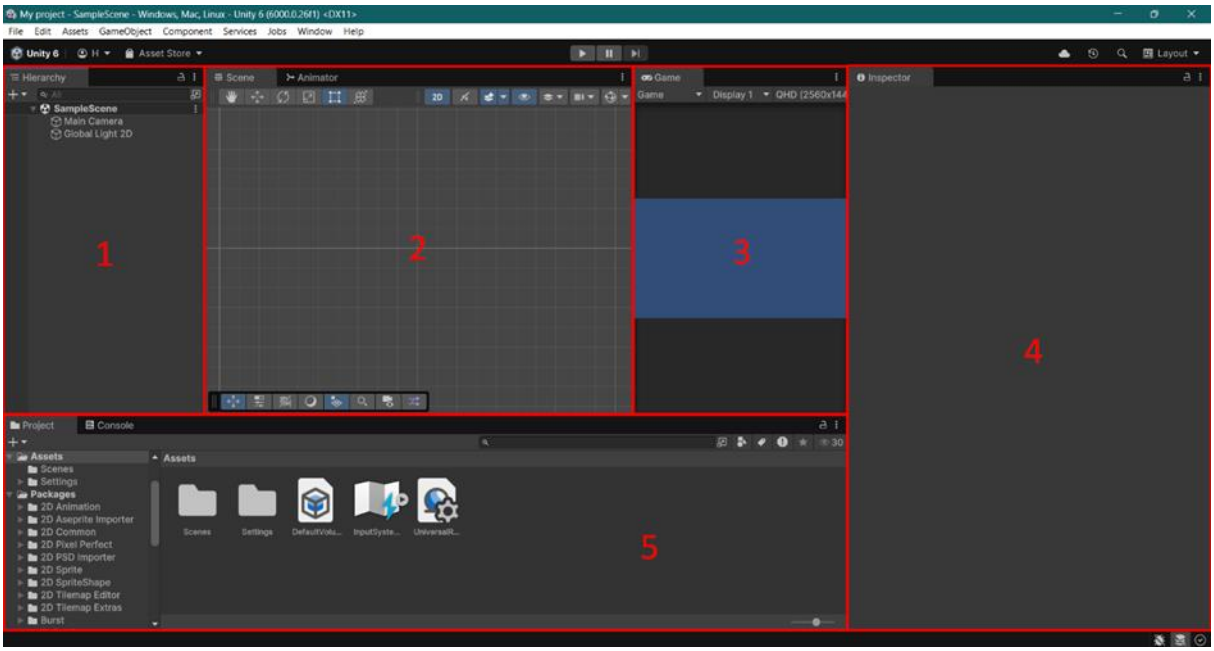
Lisans işlemi tamamlandıktan sonra "Installs" sekmesinden "Install" butonuna tıklayın. Unity'nin önerdiği sürümlerden birini seçerek editörü indirin.

2.1.7. Proje oluşturma

“Projects” sekmesine gidin ve sağ üstteki “New project” butonuna tıklayın. Proje türünü belirleyin (örneğin, 2D projeler için “Universal 2D”) ve “Create project” ile projeyi oluşturun.

2.2. UNİTY 2D TEMEL ARAYÜZ TANITIMI

Unity projemizde sıklıkla kullanacağımız temel sekmelerin (Resim2.1) bütün görünümü.



Resim2.1

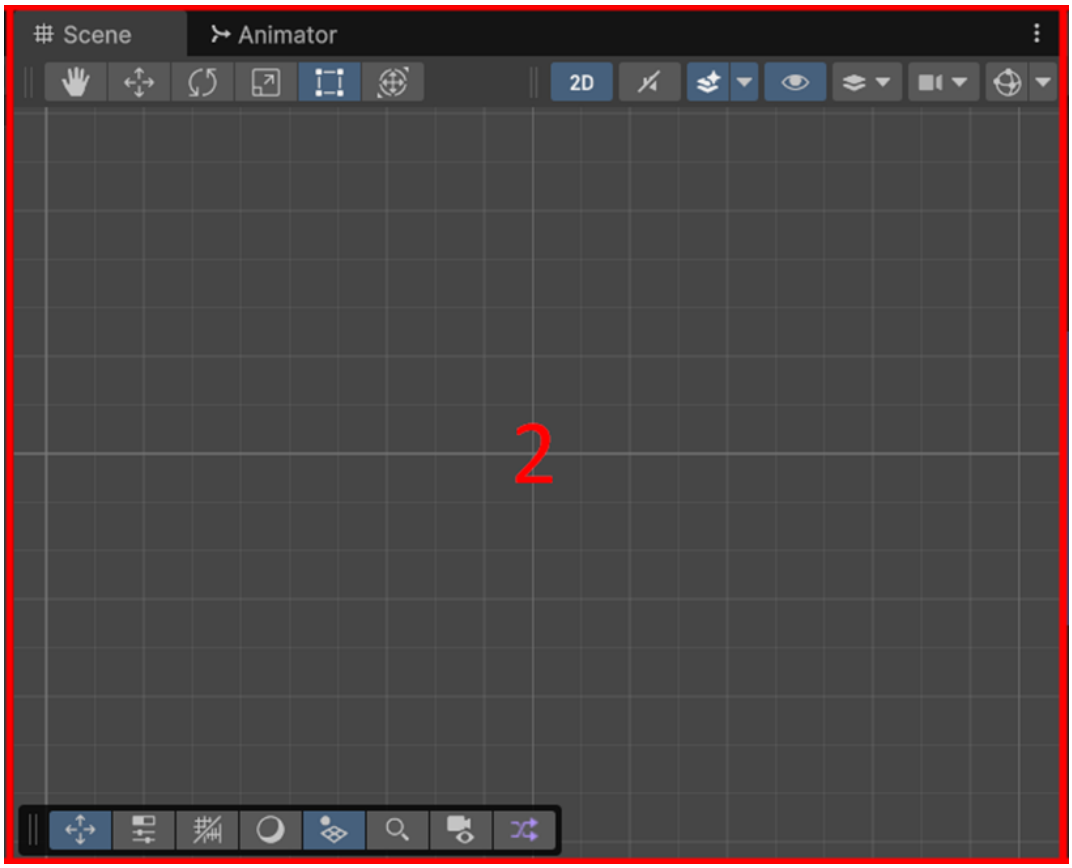
2.2.1. Hierarchy (Hiyerarşi) sekmesi

Sahnedeki tüm oyun nesnelerini (GameObject'leri) listeler (Resim2.2). Karakterler, kameralar ve ışıklar gibi nesneler burada organize edilir ve düzenlenir.

Resim2.2

2.2.2. Scene (Sahne) sekmesi

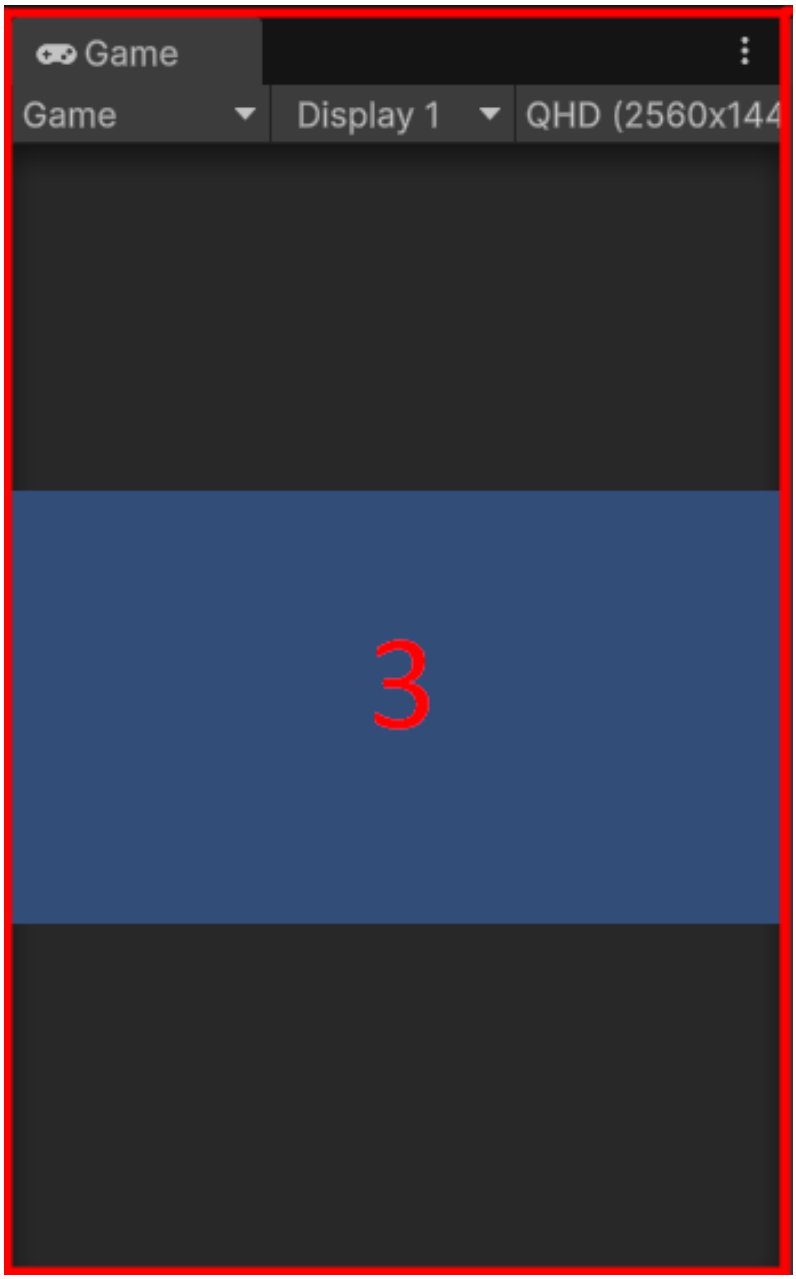
Oyun dünyanızı görsel olarak düzenleyebileceğiniz alandır (Resim2.3). Nesneleri eklemek, taşımak, ölçeklendirmek ve döndürmek için kullanılır. Sahneyi farklı açılardan görebilir, yakınlaştırıp uzaklaştırabilirsiniz.



Resim2.3

2.2.3. Game (Oyun) sekmesi

Oyunun gerçek zamanlı olarak nasıl göründüğünü gösterir (Resim2.4). Play butonuna basarak oyununuzun önizlemesini burada test edebilirsiniz. Play butonu ekranın üst orta kısmındadır.



Resim2.4

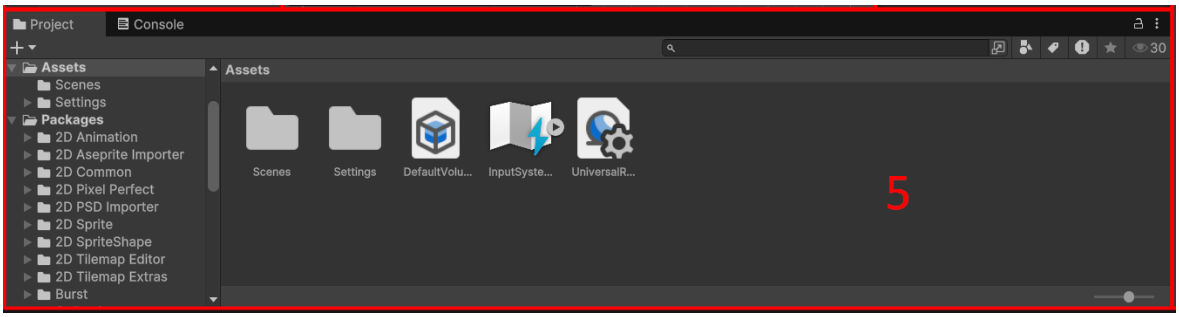
2.2.4. Inspector (İnceleme) sekmesi

Seçilen nesnenin tüm özelliklerini ve bileşenlerini gösterir (Resim2.5). Örneğin, bir karakter seçildiğinde transform, script ve diğer bileşenler buradan düzenlenir.

Resim2.5

2.2.5. Project (Proje) sekmesi

Projedeki tüm varlıkları (assets) organize eder ve listeler (Resim2.6). Sprite, ses, script gibi dosyalar bu sekmede saklanır ve sahneye buradan eklenir.



Resim2.6

2.3. SAHNE TASARLAMA, ANA KARAKTER EKLEME, ANİMASYON VE KOD OLUŞTURMA

2.3.1. Asset ekleme

Eğer tasarımcı değilseniz, görseller, karakter modelleri ve animasyonlar gibi assetleri Unity Asset Store'dan edinebilirsiniz. (https://assetstore.unity.com).

2.3.2. Asset'i projeye ekleme

Seçilen asset, Asset Store > My Assets yoluyla Paket Yöneticisi kullanılarak projeye eklenir. "Import to project" butonuna tıklayarak asset'i Project sekmesine dahil edebilirsiniz.

2.3.3. Sahne ve platform oluşturma

Asset'lerinizi kullanarak sahne tasarımı yapın. Platformları oluşturmak için sürükle-bırak yöntemiyle nesneleri sahneye yerleştirin.

2.3.4. Karakter ekleme

Karakterinizi tasarladığınız platformun üzerine yerleştirin.

2.3.5. Gerekli temel bileşenleri ekleme

Karakterin ve diğer nesnelerin fiziksel ve çarpışma özelliklerini tanımlamak için Inspector > Add Component yoluyla bileşenler ekleyin.

2.3.6. Karaktere Animasyon Ekleme

Karakterinizi seçip Inspector panelinden Animator bileşenini ekleyin. Sıralı resimlerden bir animasyon oluşturarak karakterin nefes alma (idle) animasyonunu yapın. Oluşturduğunuz animasyon kontrolcüsünü Animator bileşenine bağlayarak Animator Paneli üzerinden yönetebilirsiniz.

2.3.7. Kod ekleme

Projenizde nesnelerin nasıl çalışacağını tanımlamak için kod oluşturun. Project panelinde sağ tıklayıp Create > MonoBehaviour Script seçeneği ile yeni bir script oluşturun. Script'i karakter veya diğer nesnelerinize atayarak davranışlarını tanımlayın.

3.BÖLÜM

1. HAFTA

3.1. Karakter hareketleri ekleme

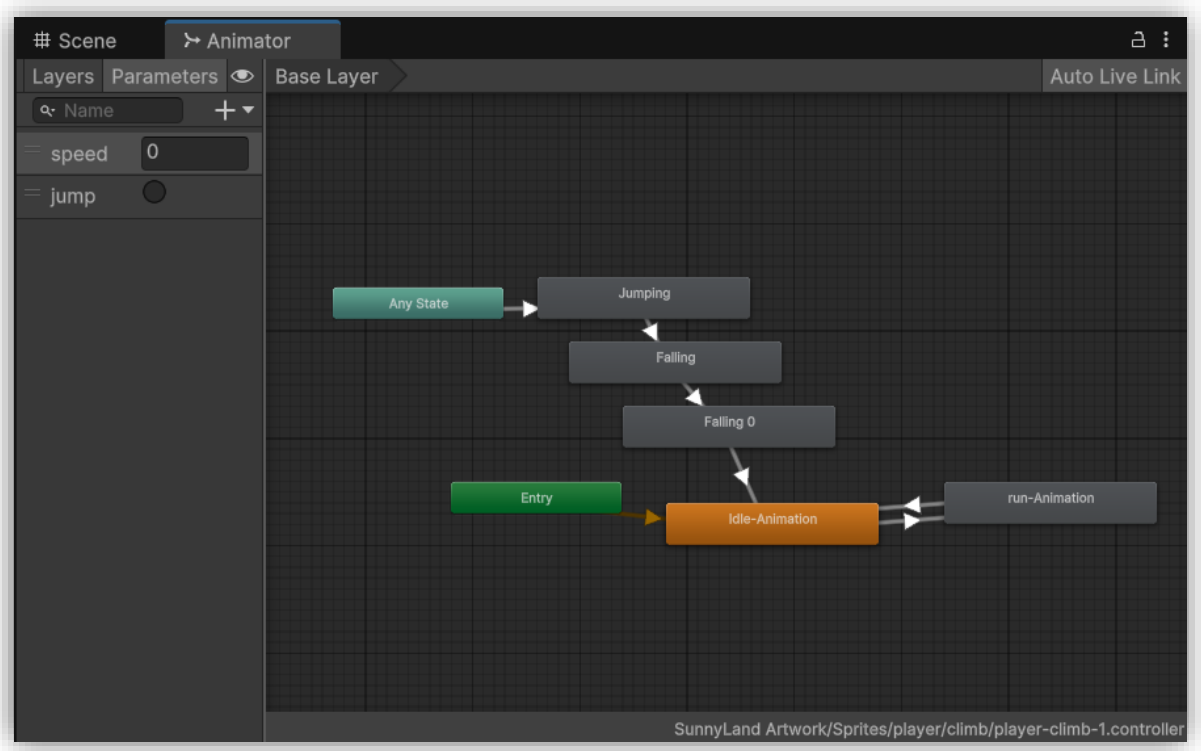
Karakterin hareket edebilmesi için bir kod yazıldı (Resim3.1). Artık “A” tuşu ile sola, “D” tuşu ile sağa hareket ediyor ve “Space” tuşu ile zıplıyor.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 0 başvuru
public class PlayerController : MonoBehaviour
{
    private Rigidbody2D rb;
    Vector3 velocity;
    public Animator animator;
    public float speedAmount = 5f;
    public float jumpAmount = 5f;
    private bool isGrounded = false;
    // Start is called before the first frame update
    Unity İletisi | 0 başvuru
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
    }

    // Update is called once per frame
    Unity İletisi | 0 başvuru
    void Update()
    {
        velocity = new Vector3(Input.GetAxis("Horizontal"), 0f);
        transform.position += velocity * speedAmount * Time.deltaTime;
        animator.SetFloat("speed", Mathf.Abs(Input.GetAxis("Horizontal")));

        if (Input.GetButtonDown("Jump") && Mathf.Approximately(rb.linearVelocity.y, 0f))
        {
            rb.AddForce(Vector3.up * jumpAmount, ForceMode2D.Impulse);
        }

        if (Input.GetAxisRaw("Horizontal") == -1)
        {
            transform.rotation = Quaternion.Euler(0f, 180f, 0f);
        }
        else if (Input.GetAxisRaw("Horizontal") == 1)
        {
            transform.rotation = Quaternion.Euler(0f, 0f, 0f);
        }
        if (Input.GetKeyDown(KeyCode.Space))
        {
            animator.SetTrigger("jump");
        }
```

Resim3.1

3.2. Karakter hareketlerine animasyon ekleme

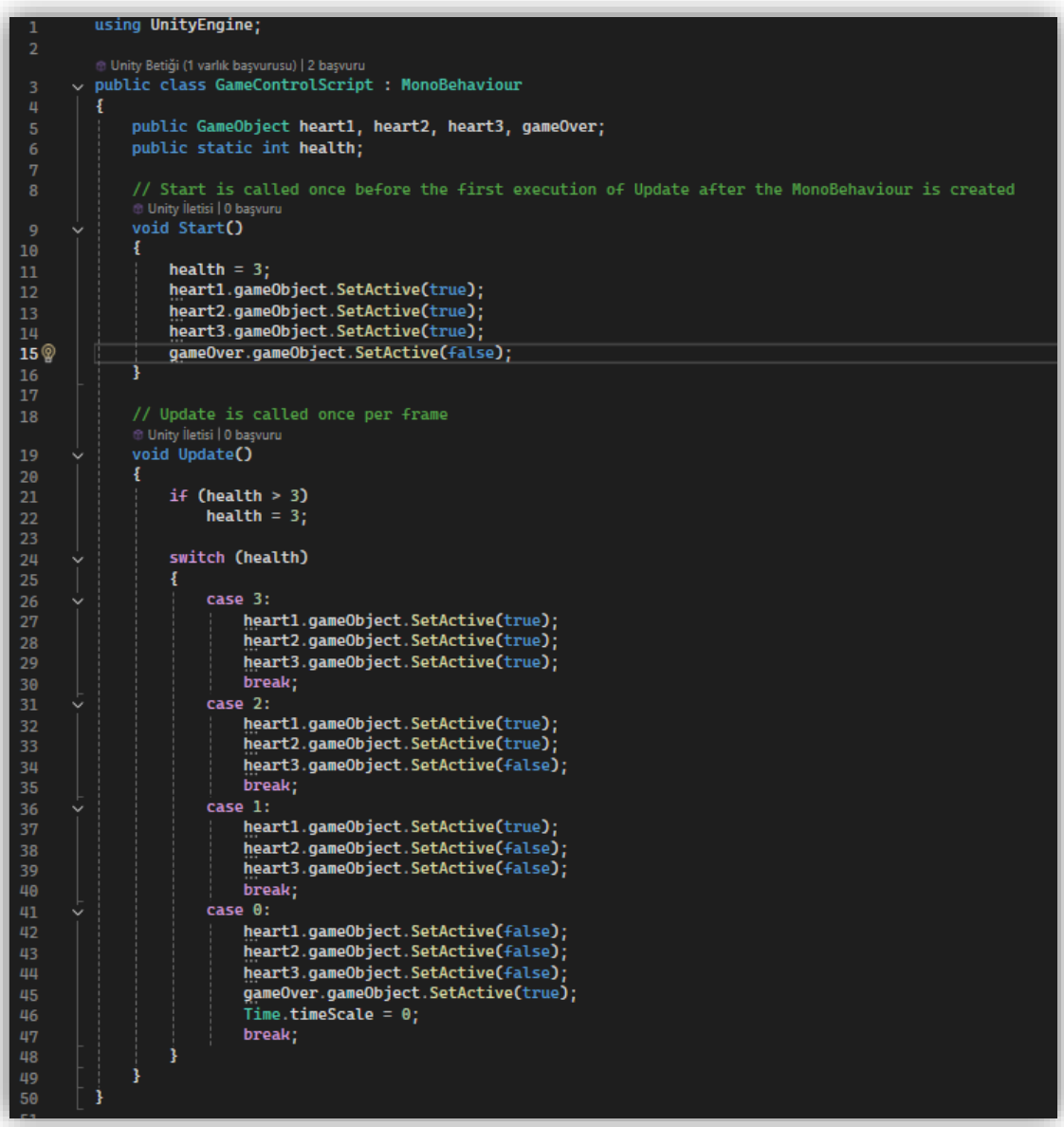
Karakterin hareketlerini görselleştirmek için animasyonlar eklendi. Hareket animasyonlarını çalıştırmak için gerekli kodlar ve parametreler kullanılarak Animator düzenlemeleri yapıldı (Resim3.2). Animasyonlar, karakterin hareketleriyle uyumlu bir şekilde çalışıyor.



Resim3.2

3.3. Sağlık sistemi ekleme

Düşman ve tuzaklardan alınan hasarları göstermek için bir sağlık sistemi oluşturuldu (Resim3.3). İlk olarak bir kalp görseli seçildi ve 3 adet kopyalanarak sırasıyla “heart1”, “heart2” ve “heart3” olarak adlandırıldı. Bu sistemi yönetmek için “GameControl” adlı bir boş GameObject oluşturuldu ve sağlık sistemi için gerekli kod bu objeye atandı.

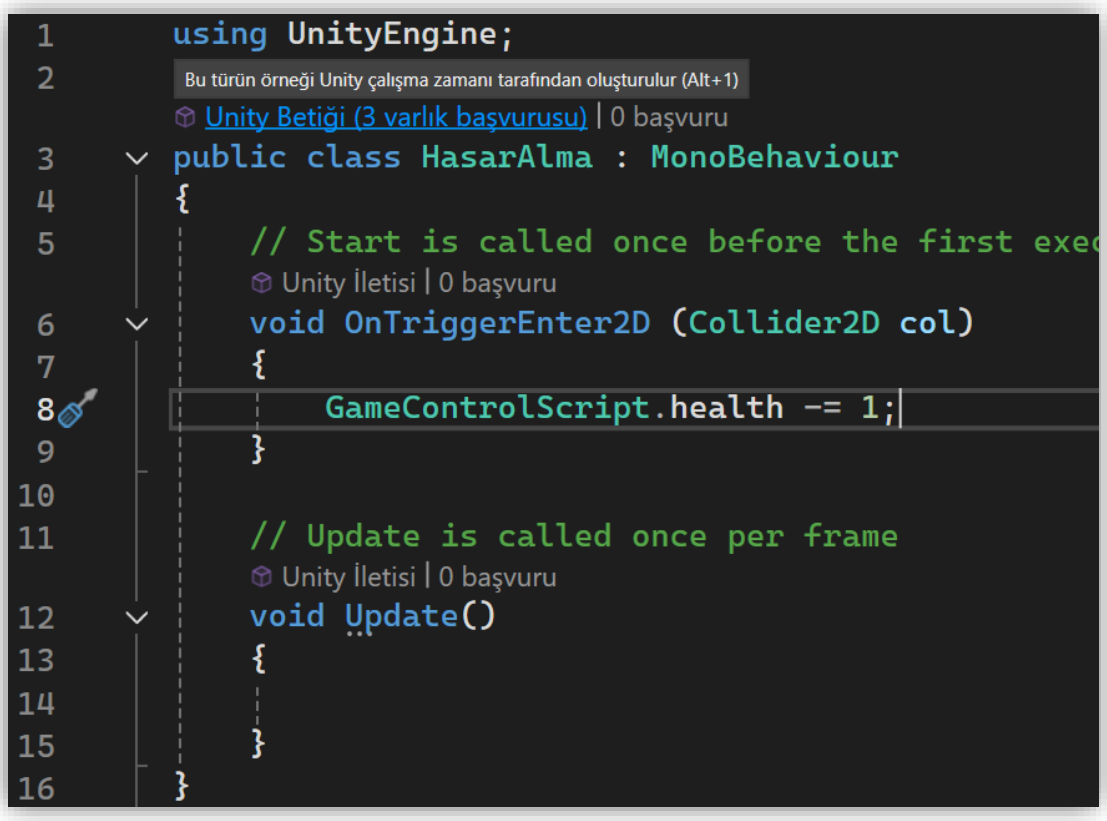


```
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 2 başvuru
public class GameControlScript : MonoBehaviour
{
    public GameObject heart1, heart2, heart3, gameOver;
    public static int health;

    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity İletisi | 0 başvuru
    void Start()
    {
        health = 3;
        heart1.gameObject.SetActive(true);
        heart2.gameObject.SetActive(true);
        heart3.gameObject.SetActive(true);
        gameOver.gameObject.SetActive(false);
    }

    // Update is called once per frame
    Unity İletisi | 0 başvuru
    void Update()
    {
        if (health > 3)
            health = 3;

        switch (health)
        {
            case 3:
                heart1.gameObject.SetActive(true);
                heart2.gameObject.SetActive(true);
                heart3.gameObject.SetActive(true);
                break;
            case 2:
                heart1.gameObject.SetActive(true);
                heart2.gameObject.SetActive(true);
                heart3.gameObject.SetActive(false);
                break;
            case 1:
                heart1.gameObject.SetActive(true);
                heart2.gameObject.SetActive(false);
                heart3.gameObject.SetActive(false);
                break;
            case 0:
                heart1.gameObject.SetActive(false);
                heart2.gameObject.SetActive(false);
                heart3.gameObject.SetActive(false);
                gameOver.gameObject.SetActive(true);
                Time.timeScale = 0;
                break;
        }
    }
}
```

Resim3.3

3.4. Hasar alma sistemi

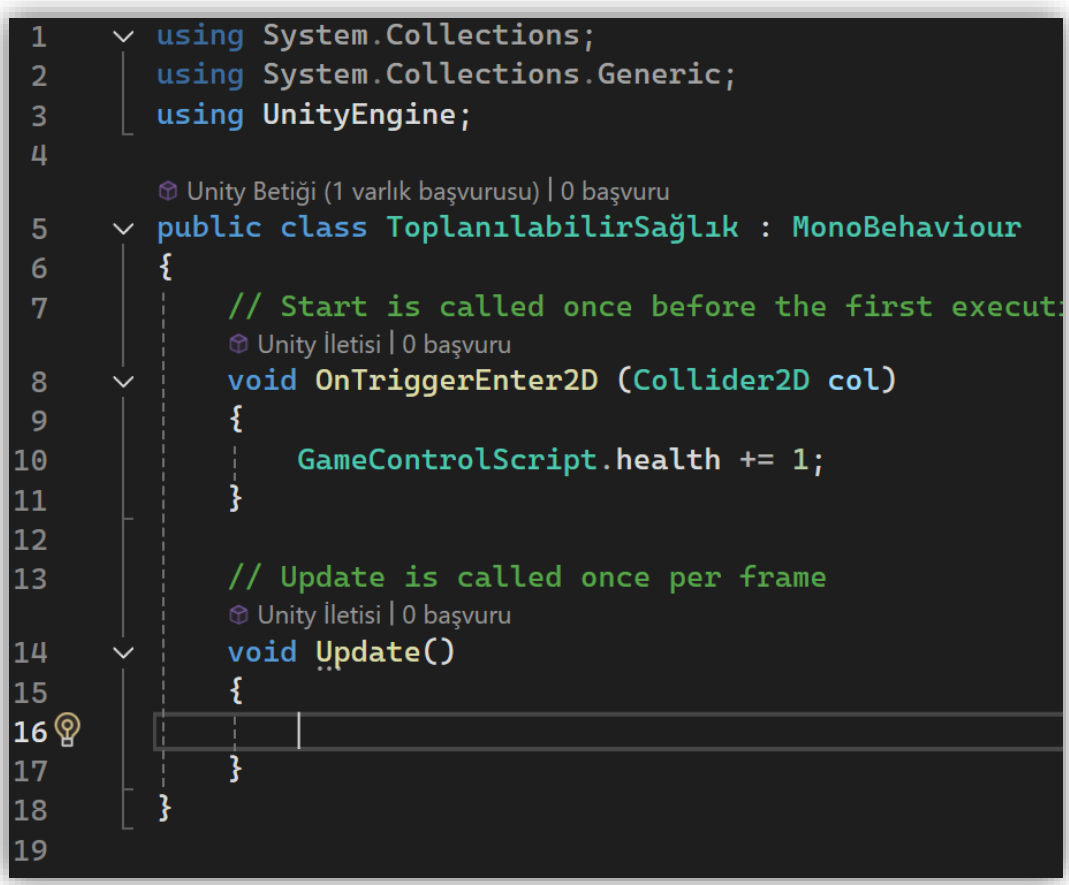
Karakter bir objeye çarptığında, sağlık göstergesinden bir kalp eksilmesi sağlandı. Bu kod (Resim3.4) sayesinde karakterin sağlık sistemi, düşmanla çarpışma durumlarını doğru şekilde yönetiyor.



Resim3.4

3.5. Sağlık kazanma mekaniği

Oyun içinde toplanabilir kalplerle karakterin sağlığı artırılabilecek bir sistem yazıldı (Resim3.5). Ancak sağlık barı maksimum 3 olduğu için, 3'ten fazla alınan kalpler eklenmiyor.

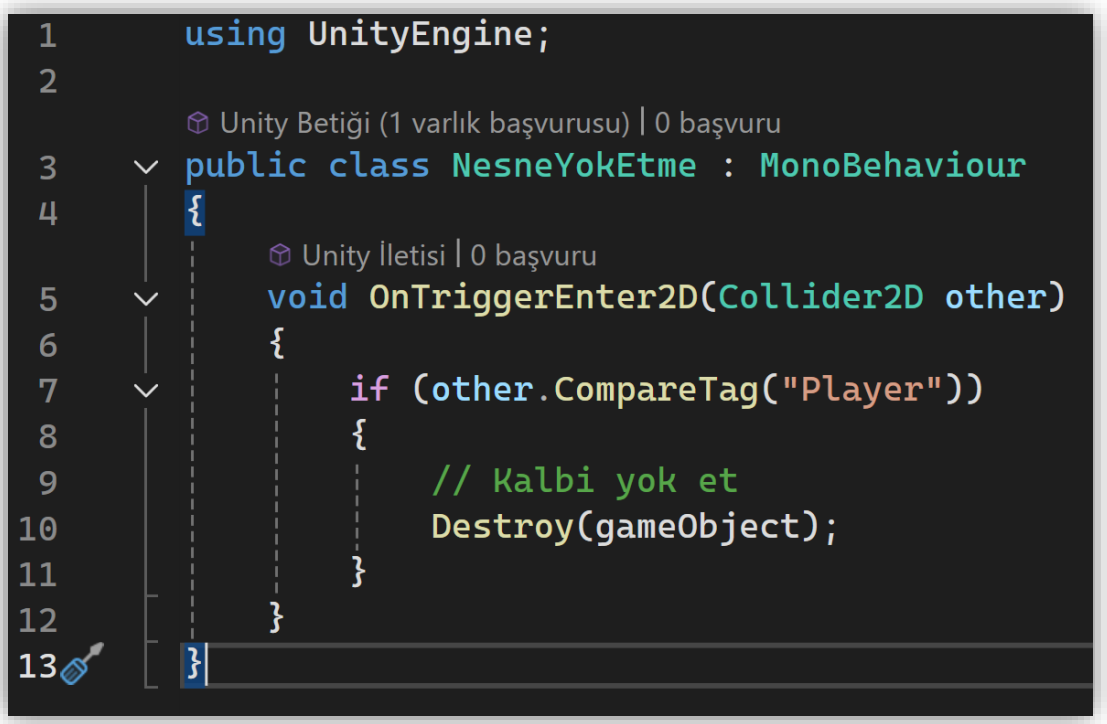


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 0 başvuru
public class ToplanılabilirSağlık : MonoBehaviour
{
    // Start is called once before the first execut
    Unity İletisi | 0 başvuru
    void OnTriggerEnter2D (Collider2D col)
    {
        GameControlScript.health += 1;
    }

    // Update is called once per frame
    Unity İletisi | 0 başvuru
    void Update()
    {

    }
}
```

Resim3.5

3.6. Toplanılabilir kalbi yok etme

Karakter bir sağlık kalbine temas ettiğinde, bu kalp sahneden silinecek şekilde ayarlandı (Resim3.6). Böylece aynı kalp tekrar alınamıyor. Bunun için kalplere “Player” tag’ı kontrolü eklendi.

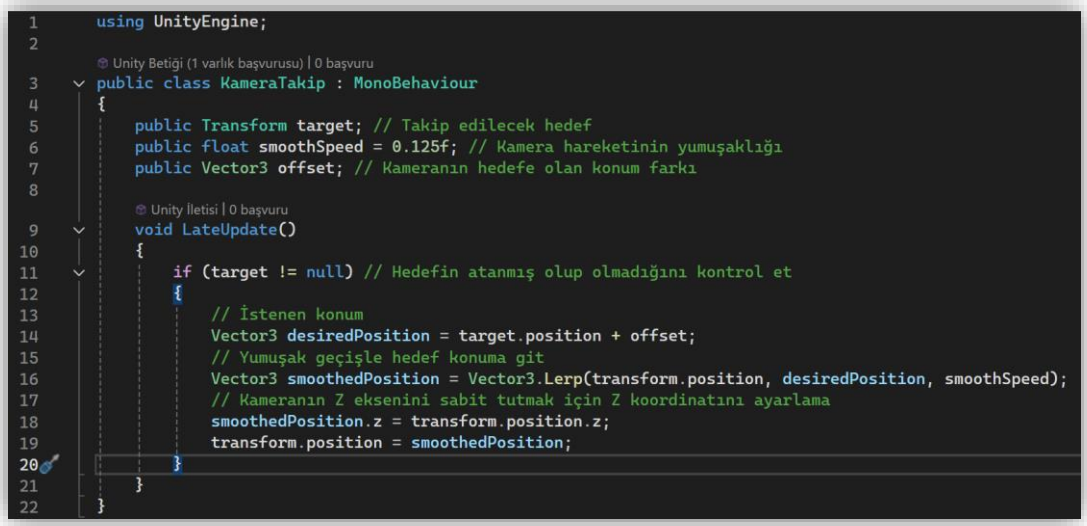


```
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 0 başvuru
public class NesneYokEtme : MonoBehaviour
{
    Unity İletisi | 0 başvuru
    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            // Kalbi yok et
            Destroy(gameObject);
        }
    }
}
```

Resim3.6

3.7. Kamera takip sistemi

Kameranın karakteri takip etmesi için bir kod yazıldı (Resim3.7) ve bu kod “Main Camera”ya atandı. Böylece kamera, karakter merkezli hareket ediyor.



```
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 0 başvuru
public class KameraTakip : MonoBehaviour
{
    public Transform target; // Takip edilecek hedef
    public float smoothSpeed = 0.125f; // Kamera hareketinin yumuşaklığı
    public Vector3 offset; // Kameranın hedefe olan konum farkı

    Unity İletisi | 0 başvuru
    void LateUpdate()
    {
        if (target != null) // Hedefin atanmış olup olmadığını kontrol et
        {
            // İstenen konum
            Vector3 desiredPosition = target.position + offset;
            // Yumuşak geçişle hedef konuma git
            Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition, smoothSpeed);
            // Kameranın Z eksenini sabit tutmak için Z koordinatını ayarlama
            smoothedPosition.z = transform.position.z;
            transform.position = smoothedPosition;
        }
    }
}
```

Resim3.7

3.8. Sağlık barı sabitleme

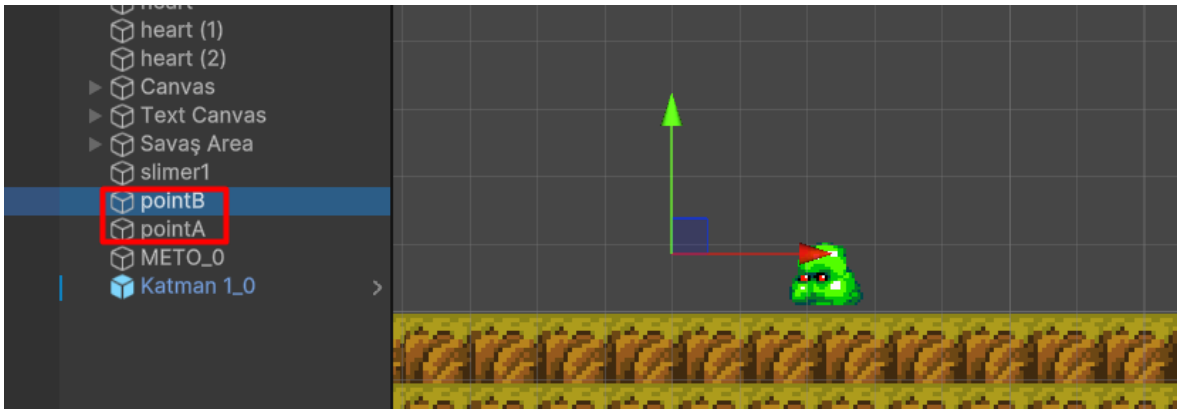
Kamera karakteri takip ederken sağlık barının sabit bir konumda kalması için bir Canvas eklendi. Canvas içine kalp objeleri yerleştirildi ve pozisyonları Rect Transform bileşeninde ayarlandı. Böylece sağlık barı ekranın sol üst köşesinde sabit bir şekilde görünüyor.

4.BÖLÜM

2.HAFTA

4.1. Düşman Davranışları

Oyuna düşman karakterler eklendi ve düşmanların belirli iki nokta arasında devriye atması sağlandı. Bu işlem için sahneye “pointA” ve “pointB” adında iki GameObject oluşturuldu ve düşman bu noktalar arasında gidip geliyor (Resim4.1). Ayrıca düşman, bu hareketi sırasında yürüme animasyonu oynatıyor ve noktadan noktaya ulaştığında yön değiştiriyor. Kullanılan kodlar, düşmanın devriye hareketini ve animasyonlarını kontrol ederken, karakterle temas ettiğinde yön değiştirme özelliğini de ekledi.



Resim4.1

4.2. Düşman hasar verme mekaniği

Düşmanla çarpışma durumunda karakterin sağlık barından bir kalp eksilmesi için bir sistem oluşturuldu. Başlangıçta tuzaklar için kullanılan “Hasar Alma” kodu denendi ancak bozulmalar nedeniyle düşman için ayrı bir kod yazıldı (Resim4.2). Bu kod, çarpışmayı algılıyor ve sağlık sistemine bir kayıp ekliyor.

```
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 0 başvuru
public class DüşmanHasarAlma : MonoBehaviour
{
    Unity İletisi | 0 başvuru
    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            GameControlScript.health -= 1;
        }
    }
}
```

Resim4.2

4.3. Karaktere fırlatılabilir alev topu ekleme

Düşmanları etkisiz hale getirebilmek için karaktere bir alev topu fırlatma mekanizması eklendi. İnternetten bir alev topu görseli bulunarak düzenlendi ve arka planı temizlendi. Görsel sahneye eklenip Prefab olarak tanımlandı. Alev topu objesine Box Collider 2D ve Rigidbody 2D bileşenleri eklendi. Fare sol tıklandığında, karakterin bir alev topu fırlatmasını sağlayan bir kod yazıldı (Resim4.3).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 0 başvuru
public class PlayerThrow : MonoBehaviour
{
    public GameObject throwablePrefab;
    public Transform throwPoint;
    public float throwForce = 10f;

    Unity İletisi | 0 başvuru
    void Update()
    {
        if (Input.GetButtonDown("Fire1")) // "Fire1" tuşuna basıldığında (genellikle sol fare tuşu)
        {
            ThrowObject();
        }
    }

    1 başvuru
    void ThrowObject()
    {
        GameObject throwable = Instantiate(throwablePrefab, throwPoint.position, throwPoint.rotation);
        Rigidbody2D rb = throwable.GetComponent<Rigidbody2D>();
        rb.AddForce(throwPoint.right * throwForce, ForceMode2D.Impulse);
        rb.gravityScale = 0;
    }
```

Resim4.3

4.4. Alev topunun düşman ile olan etkileşimi

Alev topu ile düşman arasında bir çarpışma sistemi oluşturuldu (Resim4.4). Collision2D bileşeni kullanılarak, alev topunun düşmana çarpmasını algılıyor ve çarpışma durumunda düşmanı yok ediyor. Ayrıca, düşmana çarpmayan alev toplarının sahnede birikmesini önlemek için “Destroy(gameObject)” kodu eklendi. Bu sayede alev topu herhangi bir objeye çarptığında yok oluyor. Ayrıca Kodların çalışması için düşmana “Enemy” tagı verilmesi gerekti.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 0 başvuru
public class Throwable : MonoBehaviour
{
    Unity İletisi | 0 başvuru
    void OnCollisionEnter2D(Collision2D collision)
    {
        // Çarpışılan nesne düşman mı kontrol et
        if (collision.gameObject.CompareTag("Enemy"))
        {
            Destroy(collision.gameObject); // Düşmanı yok et
        }

        // Alev topunu her durumda yok et
        Destroy(gameObject);
    }
}
```

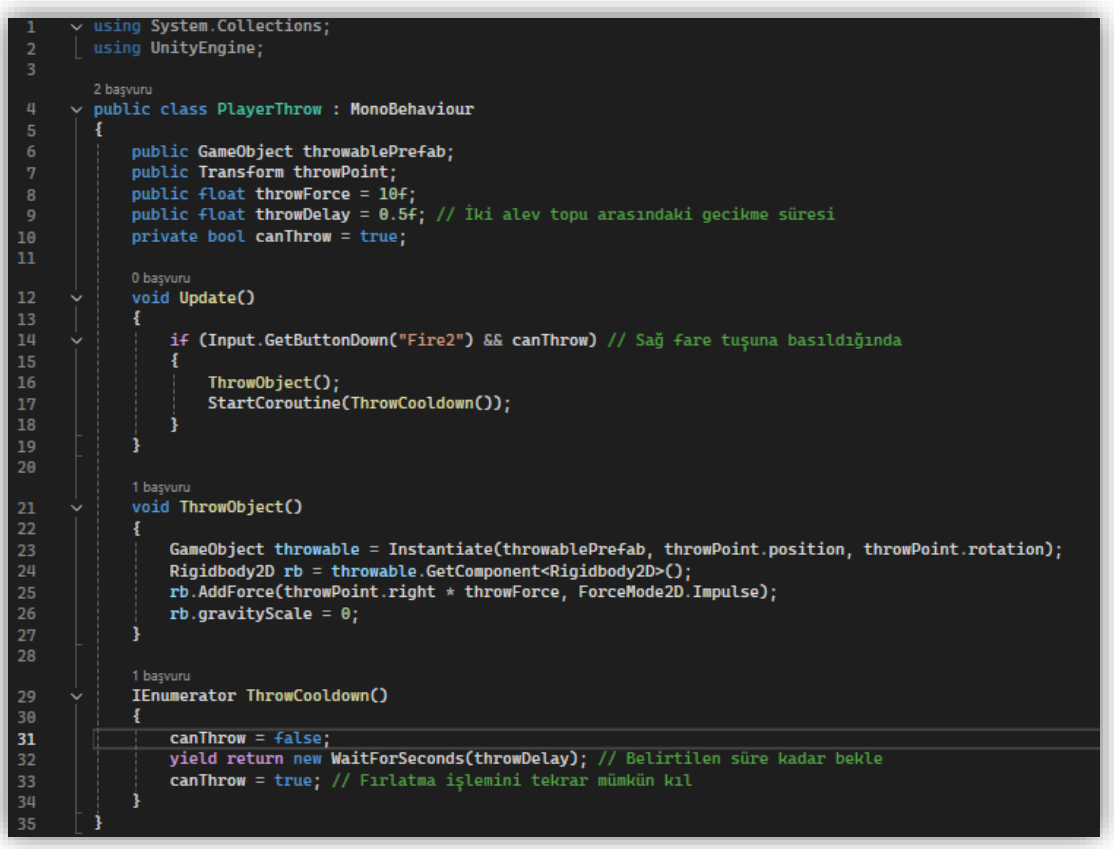
Resim4.4

5.BÖLÜM

3. HAFTA

5.1. Fırlatılabilir objeye gecikme ekleme

Karakterin alev topu fırlatma mekanizmasında, fırlatma sıklığını azaltmak için gecikme süresi eklendi. Bu, aynı anda birden fazla alev topu fırlatma hatasını önledi. Ayrıca kontrol tuşu, sol fare tuşundan (“Fire1”) sağ fare tuşuna (“Fire2”) taşındı. Kodlar güncellendi (Resim5.1)ve sistem artık daha kararlı çalışıyor.

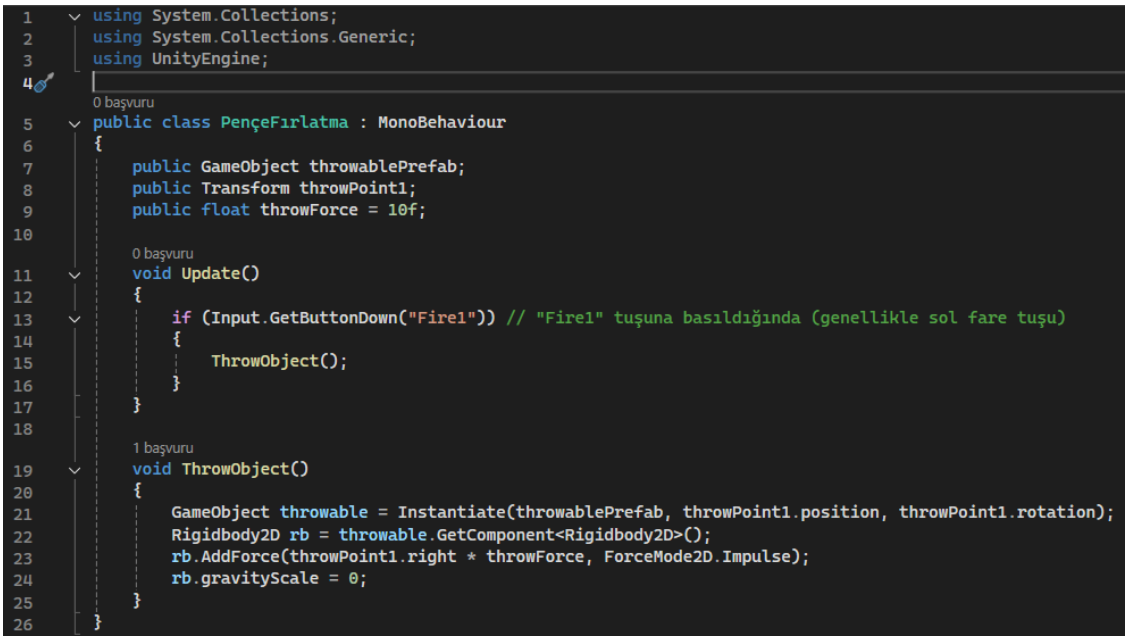


```
using System.Collections;
using UnityEngine;

2 başvuru
public class PlayerThrow : MonoBehaviour
{
    public GameObject throwablePrefab;
    public Transform throwPoint;
    public float throwForce = 10f;
    public float throwDelay = 0.5f; // İki alev topu arasındaki gecikme süresi
    private bool canThrow = true;

    0 başvuru
    void Update()
    {
        if (Input.GetButtonDown("Fire2") && canThrow) // Sağ fare tuşuna basıldığında
        {
            ThrowObject();
            StartCoroutine(ThrowCooldown());
        }
    }

    1 başvuru
    void ThrowObject()
    {
        GameObject throwable = Instantiate(throwablePrefab, throwPoint.position, throwPoint.rotation);
        Rigidbody2D rb = throwable.GetComponent<Rigidbody2D>();
        rb.AddForce(throwPoint.right * throwForce, ForceMode2D.Impulse);
        rb.gravityScale = 0;
    }

    1 başvuru
    IEnumerator ThrowCooldown()
    {
        canThrow = false;
        yield return new WaitForSeconds(throwDelay); // Belirtilen süre kadar bekle
        canThrow = true; // Fırlatma işlemini tekrar mümkün kıl
    }
}
```

Resim5.1

5.2. Yakın vuruş sistemi ekleme

Alev topuna ek olarak, düşmanlarla yakın temasta etkili olacak bir pençe saldırı sistemi oluşturuldu. İnternetten bir pençe görseli bulundu ve düzenlendi. Sahneye “throwPoint1” adında yeni bir GameObject eklendi ve karakterin objesine Child olarak atandı. Pençe objesinin, fırlatma noktasından çıkıp hızlıca yok olmasını sağlayarak gerçekçi bir yakın saldırı hissiyatı oluşturuldu. Kodlar (Resim5.2), alev topu sistemine benzer şekilde yeniden yapılandırılarak yakın vuruş için optimize edildi.

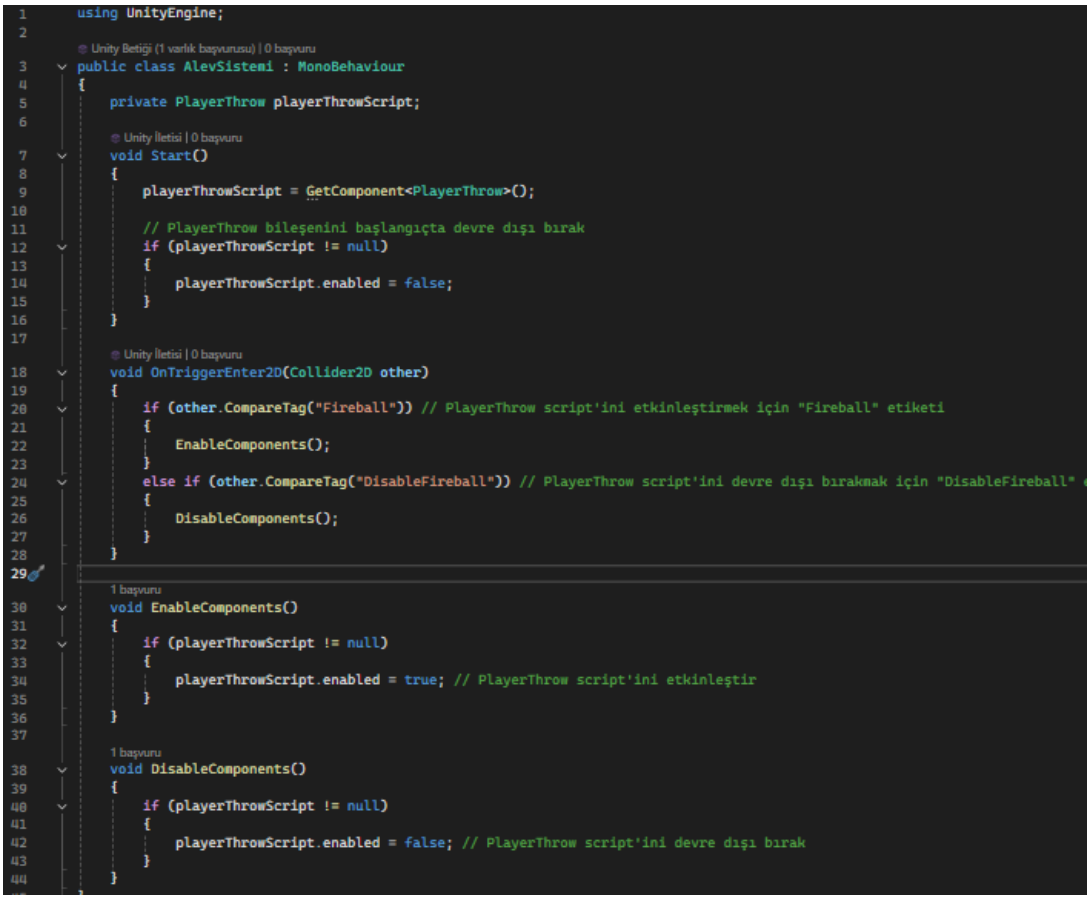


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

0 başvuru
public class PençeFırlatma : MonoBehaviour
{
    public GameObject throwablePrefab;
    public Transform throwPoint1;
    public float throwForce = 10f;

    0 başvuru
    void Update()
    {
        if (Input.GetButtonDown("Fire1")) // "Fire1" tuşuna basıldığında (genellikle sol fare tuşu)
        {
            ThrowObject();
        }
    }

    1 başvuru
    void ThrowObject()
    {
        GameObject throwable = Instantiate(throwablePrefab, throwPoint1.position, throwPoint1.rotation);
        Rigidbody2D rb = throwable.GetComponent<Rigidbody2D>();
        rb.AddForce(throwPoint1.right * throwForce, ForceMode2D.Impulse);
        rb.gravityScale = 0;
    }
}
```

Resim5.2

5.3. İkincil saldırı için Aç-Kapa sistemi

Alev topu sistemi, oyun ilerleyişine bağlı olarak açılıp kapatılabilir hale getirildi (Resim5.3). Bölümde belirli bir objeyle temas edildiğinde alev topu saldırısı aktif hale geliyor. Başka bir objeyle temas edildiğinde ise bu saldırı deaktif ediliyor. Bu işlem için ilgili objelere sırasıyla “Fireball” ve “DisableFireball” tag'leri atandı.



```
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 0 başvuru
public class AlevSistemi : MonoBehaviour
{
    private PlayerThrow playerThrowScript;

    Unity İletisi | 0 başvuru
    void Start()
    {
        playerThrowScript = GetComponent<PlayerThrow>();

        // PlayerThrow bileşenini başlangıçta devre dışı bırak
        if (playerThrowScript != null)
        {
            playerThrowScript.enabled = false;
        }
    }

    Unity İletisi | 0 başvuru
    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Fireball")) // PlayerThrow script'ini etkinleştirmek için "Fireball" etiketi
        {
            EnableComponents();
        }
        else if (other.CompareTag("DisableFireball")) // PlayerThrow script'ini devre dışı bırakmak için "DisableFireball"
        {
            DisableComponents();
        }
    }

    1 başvuru
    void EnableComponents()
    {
        if (playerThrowScript != null)
        {
            playerThrowScript.enabled = true; // PlayerThrow script'ini etkinleştir
        }
    }

    1 başvuru
    void DisableComponents()
    {
        if (playerThrowScript != null)
        {
            playerThrowScript.enabled = false; // PlayerThrow script'ini devre dışı bırak
        }
    }
```

Resim5.3

5.4. Düşman çeşitliliği

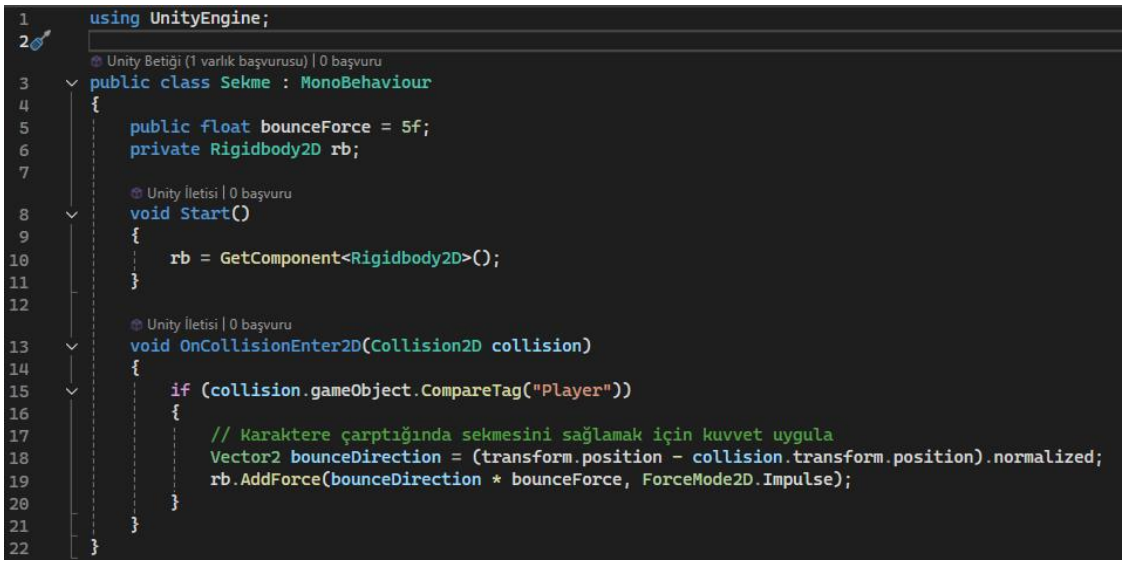
Oyun dünyasına, 2. haftada eklenen balçık düşmanına ek olarak, kartal düşmanı da eklendi (Resim5.4). Bu düşman, yukarı ve aşağı doğru devriye hareketleri yaparken bir yandan da bir trigger alanı ile donatıldı. Kartalın menziline girildiğinde, karakterin üzerine doğru saldırıya geçmesi sağlandı. Hasar verme mekanizması için balçık düşmanında kullanılan kodlar baz alınarak gerekli düzenlemeler yapıldı. Böylece düşman çeşitliliği artırılarak oyuna daha dinamik ve heyecanlı bir yapı kazandırıldı.

```
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 0 başvuru
public class EagleEnemy : MonoBehaviour
{
    public float patrolSpeed = 2f;
    public float chaseSpeed = 4f;
    public float patrolHeight = 2f;
    public float detectionRange = 5f;

    private Transform player;
    private Vector3 startPosition;
    private bool isChasing = false;

    Unity İletisi | 0 başvuru
    void Start()
    {
        startPosition = transform.position;
        player = GameObject.FindGameObjectWithTag("Player").transform;
    }

    Unity İletisi | 0 başvuru
    void Update()
    {
        float distanceToPlayer = Vector3.Distance(transform.position, player.position);

        if (distanceToPlayer < detectionRange)
        {
            isChasing = true;
        }
        else
        {
            isChasing = false;
        }

        if (isChasing)
        {
            ChasePlayer();
        }
        else
        {
            Patrol();
        }
    }

    1 başvuru
    void Patrol()
    {
        float newY = Mathf.PingPong(Time.time * patrolSpeed, patrolHeight) + startPosition.y;
        transform.position = new Vector3(startPosition.x, newY, startPosition.z);
    }

    1 başvuru
    void ChasePlayer()
    {
        transform.position = Vector3.MoveTowards(transform.position, player.position, chaseSpeed * Time.deltaTime);
    }
}
```

Resim5.4

5.5. Düşman ile karakterin iç içe geçme sorunu

Kartal, karaktere saldırdığında iç içe geçip karakteri sıkıştırma problemi yaşandı. Bu sorun, çarpışma algılama ve pozisyon düzenleme kodlarıyla çözüldü (Resim5.5). Artık düşman ve karakter arasında fiziksel bir sınır korunuyor.



```
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 0 başvuru
public class Sekme : MonoBehaviour
{
    public float bounceForce = 5f;
    private Rigidbody2D rb;

    Unity İletisi | 0 başvuru
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    Unity İletisi | 0 başvuru
    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            // Karaktere çarptığında sekmesini sağlamak için kuvvet uygula
            Vector2 bounceDirection = (transform.position - collision.transform.position).normalized;
            rb.AddForce(bounceDirection * bounceForce, ForceMode2D.Impulse);
        }
    }
}
```

Resim5.5

6.BÖLÜM

4.HAFTA

6.1. Ana menü

Oyuna kendi hazırladığım menü görseli (Resim6.1) ile birlikte ayrı bir sahneye “UI, Canvas ve Button” öğelerini kullanarak, içinde “Play” ve “Quit” butonları olan bir ana menü ekledim. Bu butonlar play tuşu ile oyunu başlatıyor quit tuşu ile oyunda çıkmamızı sağlıyor.

Resim6.1

```
using UnityEngine;
using UnityEngine.SceneManagement;

0 başvuru
public class MainMenu : MonoBehaviour
{
    0 başvuru
    public void StartGame()
    {
        SceneManager.LoadScene("GameScene"); // Oyunun sahnesini yükler
    }

    0 başvuru
    public void QuitGame()
    {
        Application.Quit(); // Oyunu kapatır
        Debug.Log("Game Quit!"); // Editörde test etmek için
    }
}
```

Resim6.2: Kullanılan kodlar dizini

6.2. Durdurma menüsü

Oyuna ana menü yöntemiyle aynı şekilde yaptığım, içinde “Resume” ve “Main Menu” butonları olan bir durdurma menüsü ekledim. Klavyeden “Escape” tuşuna bastığımızda durdurma menüsü açılacak, açılan menüdeki resume tuşu oyunu devam ettirecek main menü tuşu ise bizi ana menüye yönledirecek.

```
using UnityEngine;
using UnityEngine.SceneManagement;

0 başvuru
public class PauseMenu : MonoBehaviour
{
    public GameObject pauseMenu;
    public bool isPaused;

    0 başvuru
    void Start()
    {
        pauseMenu.SetActive(false);
    }

    0 başvuru
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (isPaused)
            {
                ResumeGame();
            }
            else
            {
                PauseGame();
            }
        }
    }

    1 başvuru
    public void PauseGame()
    {
        pauseMenu.SetActive(true);
        Time.timeScale = 0f;
        isPaused = true;
    }

    1 başvuru
    public void ResumeGame()
    {
        pauseMenu.SetActive(false);
        Time.timeScale = 1f;
        isPaused = false;
    }

    0 başvuru
    public void LoadMainMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu");
    }
}
```

Resim6.3:Kullanılan kodlar dizini

6.3. Checkpoint

Bölümler arası ilerleme kat etmekte kullanacağımız bir checkpoint sistemi ekledim, böylece karakterimiz bölüm atladığındaki olası bir ölme durumunda en baştan başlamayıp bu checkpointlerden doğacak.

```
using UnityEngine;

Unity Betiği (1 varlık başvurusu) | 0 başvuru
public class Checkpoint : MonoBehaviour
{
    public Transform player; // Oyuncunun Transform'u
    private static Vector3 checkpointPosition; // Tüm checkpoint'ler aras

    Unity İletisi | 0 başvuru
    void Start()
    {
        // İlk checkpoint yüklendiğinde oyuncunun pozisyonunu kaydet
        if (checkpointPosition == Vector3.zero)
        {
            checkpointPosition = player.position;
        }
    }

    Unity İletisi | 0 başvuru
    void Update()
    {
        // Sağlık sıfır olduğunda oyuncuyu checkpoint'e taşı
        if (GameControlScript.health <= 0)
        {
            RespawnAtCheckpoint();
        }
    }

    1 başvuru
    private void RespawnAtCheckpoint()
    {
        player.position = checkpointPosition; // Oyuncuyu son kaydedilen
        GameControlScript.health = 3; // Sağlığı sıfırla
        Time.timeScale = 1; // Oyunu devam ettir
    }

    Unity İletisi | 0 başvuru
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            checkpointPosition = transform.position; // Bu checkpoint'i k
            Debug.Log($"Checkpoint güncellendi: {checkpointPosition}");
        }
    }
}
```

Resim6.4: Kullanılan kodlar dizini

6.4. Zıplatan yay

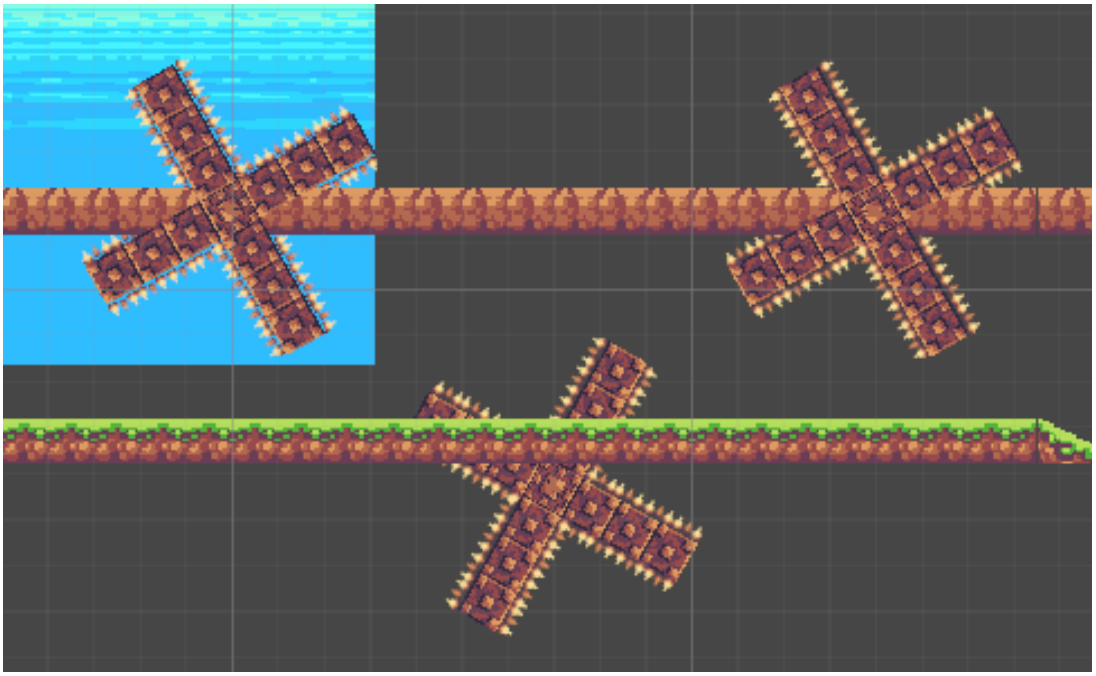
Karakterimizin, teması halinde yukarıya doğru bir kuvvet uygulayacak bir yay platformu ekledim. Bu, bölümler içerisinde parkurlara çeşitlilik katacak. Ayrıca bu yayın zıplatma kuvveti ve zıplatma yönünü değiştirebiliyoruz.

```
using UnityEngine;

Unity Betiği (2 varlık başvurusu) | 0 başvuru
public class YayZıplatma : MonoBehaviour
{
    public float launchForce = 10f; // Yaydan fırlatılacak kuvvet
    public Vector2 launchDirection = Vector2.up; // Yön (yukarı)

    Unity İletisi | 0 başvuru
    private void OnCollisionEnter2D(Collision2D collision)
    {
        // Eğer çarpan nesne oyuncuysa
        if (collision.gameObject.CompareTag("Player"))
        {
            Rigidbody2D playerRb = collision.gameObject.GetComponent<Rigidbody2D>();

            // Eğer oyuncu Rigidbody2D'ye sahipse
            if (playerRb != null)
            {
                // Oyuncuyu yukarı doğru fırlat
                playerRb.linearVelocity = Vector2.zero; // Mevcut hızını sıfırlıyoruz
                playerRb.AddForce(launchDirection * launchForce, ForceMode2D.Impulse);
            }
        }
    }
}
```

Resim6.5: Kullanıla kodlar dizini

6.5. Dönen tuzak

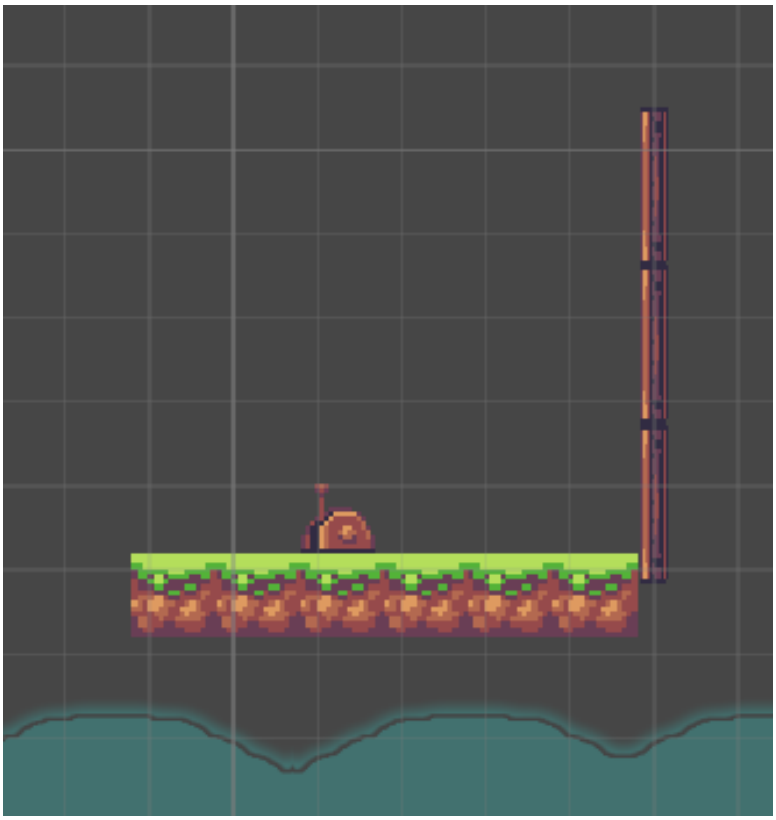
Projeye dönen bir tuzak mekanizması (Resim6.6) ekledim. Bu tuzaklar varsayılan olarak sola doğru dönmektedir. Ancak, ters yönde çalışmasını istediğimiz durumlarda, Inspector panelindeki Rotation Direction parametresini -1 olarak ayarlayarak dönüş yönünü değiştirebiliriz.



Resim6.6

6.6. Hendek kapısı mekanizması

Oyunuma, hendek kapısı sistemine sahip bir kapı mekanizması (Resim6.7) ekledim. Bu mekanizmada, kapının nereden açılacağını belirlemek için bir GameObject oluşturarak bunu menteşe noktası olarak ayarladım ve kapıyı bu noktaya bağladım. Ayrıca, kapının açılabilmesi için bir trigger alanı tasarlayıp, bu alanı bir kol görselinin üzerine yerleştirdim. Yalnızca bu kolun yanında E tuşuna basıldığında kapı açılacak şekilde sistemi düzenledim. Ek olarak, kola bir kod ekleyerek, E tuşuna basıldığında kolun yukarı ve aşağı hareket etmesini ve görselinin değişmesini sağladım. Bu detaylar, kapının kolla birlikte açıldığı hissini güçlendirirken mekanizmanın gerçekçiliğini de önemli ölçüde artırıyor.



Resim6.7

6.7. Kutu itme ve yerleştirme mekaniği

Oyunuma, karakterin ilerleyebilmesi için bir kutu itme ve belirli bir noktaya yerleştirme mekanizması ekledim. Bu mekanizmayı oluşturmak için oyuna bir kutu objesi ekledim ve kutuya Box Collider 2D ile Rigidbody 2D bileşenlerini ekleyerek fiziksel etkileşimlerini sağladım. Bu sistem, oyuncunun kutuyu doğru noktaya iterek ilerlemesini gerektiriyor ve oyunun etkileşimli yapısını güçlendiriyor.

6.8. Kırılabilir obje mekaniği

Oyunuma, kırılabilir objeler ekledim ve bu sistemi oldukça basit bir şekilde tasarladım. Kırılabilir objelere Enemy tagı ekledim ve karakterin mevcut pençe mekaniği, bu taglı objeleri yok edebildiği için ek bir işlem yapmama gerek kalmadı. Bu sayede, karakterin pençe mekaniğiyle kırılabilir objeler kolayca yok edilebiliyor.

6.9. Düşen obje tuzağı

Oyunuma, karakterin yaklaştığında belirli bir gecikmeyle düşen objeler ekledim. Bu mekanizma, objelerin karakterin yaklaşmasıyla tetiklenerek düşmesini sağlıyor. Yaklaşma mesafesi ve düşme gecikmesi gibi değerler Inspector panelinden kolayca ayarlanabiliyor. Bu özellik, oyuna daha dinamik bir etkileşim ekleyerek, oyuncunun hareketlerine tepki veren objeler oluşturuyor.

```
using UnityEngine;

0 başvuru
public class FallingTrap : MonoBehaviour
{
    [Header("Tuzak Ayarları")]
    public Transform player; // Oyuncu nesnesi
    public float triggerDistance = 5f; // Tuzak ile oyuncu arasındaki mesafe
    public float fallDelay = 0.5f; // Düşmeden önceki gecikme süresi

    private Rigidbody2D rb; // Tuzak için Rigidbody2D bileşeni
    private bool hasFallen = false; // Tuzak bir kez düştükten sonra tekrar düşmesi

    0 başvuru
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        rb.bodyType = RigidbodyType2D.Kinematic; // Başlangıçta sabit kalsın
    }

    0 başvuru
    void Update()
    {
        // Oyuncu ile tuzak arasındaki mesafeyi hesapla
        float distance = Vector2.Distance(transform.position, player.position);

        // Mesafe tetikleme mesafesinden küçükse ve tuzak daha önce düşmediyse
        if (distance <= triggerDistance && !hasFallen)
        {
            hasFallen = true; // Tuzak bir kez düşmeye başlasın
            Invoke("ActivateFall", fallDelay); // Belirtilen gecikme ile düşmeyi ba
        }
    }

    0 başvuru
    void ActivateFall()
    {
        rb.bodyType = RigidbodyType2D.Dynamic; // Tuzak düşmeye başlasın
    }
}
```

Resim6.8: Kullanılan kodlar dizini

6.10. Final düşmanı

Oyunuma, final düşmanı olarak bir kurbağa ekledim. Bu kurbağa, belirli aralıklarla karakterin üstüne doğru zıplıyor ve davranışları Inspector panelinden kolayca düzenlenebiliyor. Zıplama aralığı, zıplama yüksekliği ve zıplama hızı gibi parametreler oyuncunun oyun deneyimini çeşitlendirmek için ayarlanabilir şekilde tasarlandı. Kurbağanın canı belirli bir seviyenin altına düştüğünde, bir kereliğine arenanın ortasına giderek canını yeniliyor ve ardından saldırılarına kaldığı yerden devam ediyor. Maksimum can değeri ve can yenileme miktarı da yine Inspector panelinden ayarlanabiliyor. Bu özellikler, kurbağayı hem zorlu hem de dinamik bir final düşmanı haline getiriyor.

KAYNAKÇA

Yapay zeka araçları: Chatgpt (https://chatgpt.com), Copilot (https://copilot.microsoft.com), Gemini (https://gemini.google.com/app?hl=tr)

2D PATHFINDING - Enemy AI in Unity
(https://www.youtube.com/watch?v=jvtFUfJ6CP8&t=929s)

Creating SMART enemies from scratch! | Devlog
(https://www.youtube.com/watch?v=wC9iu7cuQjI&t=37s)

[Unity3D] 2D Mobil Platformer Yapıyoruz - #7 - Basit Düşman Sistemi
(https://www.youtube.com/watch?v=HuSCOh-aUnk)

Simple 2D Enemy Patrolling Unity tutorial
(https://www.youtube.com/watch?v=RuvfOl8HhhM)

Unity 2D Platformer for Complete Beginners - #4 SHOOTING
(https://www.youtube.com/watch?v=PUpC44Q64zY&t=908s)

Ayni anda birden fazla animasyon oynatma - Avatar Mask – Unity
(https://www.youtube.com/watch?v=8C4ytycbpyw)

Unity 2D CHECKPOINTS Tutorial (Simple and Easy!) | Unity 2D Platformer Tutorial #9
(https://www.youtube.com/watch?v=VE_bkPrrZdE&t=104s)

How to Create a PAUSE MENU in Unity ! | UI Design Tutorial
(https://www.youtube.com/watch?v=MNUYe0PWNNs&t=317s)

Pull and Push Block - Unity Tutorial
(https://www.youtube.com/watch?v=Qz2qMxmtxpQ&t=377s)

Unity3D 5.5 - Bölüm geçme - Level atlama (Load Scene)
(https://www.youtube.com/watch?v=MLkeexsmFcA&t=452s)