🗺️ Modül 1: Dünyayı İnşa Etmek

Hedef: Boş bir projeden başlayıp, pikselleri bozulmamış net görüntüler elde etmek ve Tilemap sistemini kullanarak ilk seviyeyi çizmek.

---

1. Adım: Proje Kurulumu ve Asset İçe Aktarma

Öğrencilerinizle birlikte sıfırdan başlıyoruz.

1. **Unity Hub:** `New Project` > `2D Core` şablonunu seçin.
2. **İsim:** "TinyHero" (Türkçe karakter ve boşluk yok!).
3. **Klasör Düzeni:** `Assets` klasörü içine sağ tıklayıp şu klasörleri açtırın:
    - `_Sprites` (Başına alt çizgi koyuyoruz ki en üstte dursun).
    - `_Scripts`
    - `_Scenes`

📥 **Asset Paketini Yükleme:**

- İndirdiğiniz **Pixel Adventure 1** klasörünü sürükleyip Unity'deki `Assets > _Sprites` klasörünün içine bırakın.

---

2. Adım: "Bulanıklık" Sorunu ve Çözümü (Çok Kritik!) ⚠️

Unity varsayılan olarak görselleri yumuşatır (anti-aliasing). Ama biz keskin kareler (Pixel Art) istiyoruz. Bu ayarı tüm sınıfa aynı anda yaptırmalısınız.

1. Proje penceresinden `Pixel Adventure 1` klasörüne girin.
2. `Assets > Terrain > Terrain (16x16)` görselini seçin (Ve karakter görsellerini de).
3. **Inspector Ayarları:**
    - **Pixels Per Unit (PPU):** `100` yerine **`16`** yazın.
        - *Neden?* Çünkü bizim çizimlerimiz 16x16 piksellik karelerden oluşuyor. Bunu yapmazsak karakterler ekranda nokta kadar görünür.
    - **Filter Mode:** `Bilinear` yerine **`Point (no filter)`** seçin.
        - *Neden?* Bulanıklığı giderir, pikselleri cam gibi net yapar.
    - **Compression:** `Normal Quality` yerine **`None`** seçin.
        - *Neden?* Renklerin bozulmasını engeller.
4. Aşağıdaki **Apply** butonuna basın.

---

3. Adım: Görselleri Dilimlemek (Sprite Slicing)

Elimizdeki `Terrain (16x16).png` tek bir resim dosyası ama içinde yüzlerce zemin parçası var. Unity'e bunları nasıl ayıracağını öğretmeliyiz.

1. `Terrain (16x16)` görselini seçin.
2. Inspector'da **Sprite Mode:** `Single` yerine **`Multiple`** yapın ve **Apply** deyin.
3. **Sprite Editor** butonuna tıklayın.
4. Açılan pencerede:
    - Üst menüden **Slice**'a tıklayın.
    - Type: **Grid By Cell Size**.
    - Pixel Size: **X: 16, Y: 16**.
    - **Slice** butonuna basın (Resimlerin etrafında ince beyaz çizgiler belirmeli).
    - Sağ üstten **Apply** diyerek pencereyi kapatın.

Artık o tek resmin altında yüzlerce küçük parça oluştu!

---

4. Adım: Tilemap Sistemi (Unity'nin Boya Paleti) 🎨

Tek tek resimleri sahneye sürüklemek yerine, fırçayla boyama yapacağız.

1. **Grid Oluşturma:**
    - Hierarchy penceresinde boşluğa sağ tıkla > `2D Object` > `Tilemap` > `Rectangular`.
    - Bunun adını **"Ground"** (Zemin) olarak değiştirin.
2. **Tile Palette Açma:**
    - Üst menüden `Window` > `2D` > `Tile Palette`.
    - Açılan pencereyi editörün sağına veya altına sabitleyin.
3. **Palet Oluşturma:**
    - "Create New Palette" deyin. Adına `LevelPalette` diyebilirsiniz.
    - Dilimlediğimiz `Terrain` görselini (proje klasöründen) sürükleyip bu Palet pencresinin içine bırakın.
    - Unity, bu parçaları kaydetmek için bir klasör soracak. `_Sprites/Tiles` diye bir klasör açıp oraya kaydettirin.

---

5. Adım: Sahne Tasarımı (Level Design) 🖌️

Artık ellerindeki fırça (Brush) aracıyla sahneyi boyayabilirler.

- **Boyama:** Paletten bir zemin seç ve sahneye (Scene ekranına) tıkla.
- **Silme:** Shift tuşuna basılı tutarak tıkla (veya silgi ikonunu seç).
- **Toplu Boyama:** Kutu aracını (Box Fill) kullanarak büyük alanları doldur.
- Düz bir zemin.

- Atlamak için havada duran birkaç platform.
- Düşebilecekleri boşluklar.

---

6. Adım: Katmanlar (Sorting Layers) 🍰

Bu kısım karakteri eklediğimizde sorun yaşamamak için gerekli.

1. Inspector'da **Tilemap Renderer** bileşenini bulun.
2. **Sorting Layer** kısmına tıklayın > `Add Sorting Layer`.
3. İki yeni katman ekleyin:
    - `Background` (Arka plan - En üstte/ilk sırada olsun).
    - `Ground` (Zemin).
    - `Player` (Oyuncu).
    - `Foreground` (Ön plan - Süslemeler).
4. Hierarchy'deki **"Ground"** objesini seçip Sorting Layer'ını **`Ground`** yapın.

KONTROL LİSTESİ

1. [ ] Görüntüler bulanık değil, net pixel art şeklinde.
2. [ ] Sahneye 16x16 ızgaraya oturmuş bir zemin çizilmiş.
3. [ ] Game ekranında (Kamerada) harita görünüyor.

🏃 Modül 2: Karakter Canlanıyor (Fizik ve Animasyon)

Hedef: Karakteri sahneye eklemek, yerçekimine tabi tutmak ve koşu animasyonlarını hazırlamak.

⚠️ Adım 0: Zemin Kontrolü (Önceki Dersten Kritik Ekleme)

Karakteri sahneye atmadan önce, basacağı zeminin "Katı" olduğundan emin olmalıyız. Yoksa karakter sonsuzluğa düşer.

1. Hierarchy'den **"Ground"** (Tilemap) objesini seçin.
2. Inspector > `Add Component` > **Tilemap Collider 2D**.
3. Hemen yanına bir de **Composite Collider 2D** ekleyin.
4. Tilemap Collider 2D içindeki **"Used By Composite"** kutucuğunu işaretleyin.
5. Rigidbody 2D (Composite ile otomatik gelir) içindeki **Body Type** ayarını `Dynamic` yerine **`Static`** yapın.
    - *Neden?* Çünkü zemin hareket etmez ve düşmez.

---

1. Adım: Karakter Hazırlığı (Sprite Settings) 🥷

Tıpkı zemin gibi, karakterimiz de bulanık görünmemeli.

1. `Pixel Adventure 1 > Main Characters > Ninja Frog` klasörüne gidin.
2. Buradaki tüm görselleri seçin (Idle, Run, Jump vb.).
3. Inspector Ayarları:
    - **Pixels Per Unit:** `16` (Diğer her şey gibi).
    - **Filter Mode:** `Point (no filter)`.
    - **Compression:** `None`.
4. **Apply** deyin.

---

2. Adım: Sahneye Aktarım ve Katmanlar

1. `Idle (32x32)` resminin yanındaki oka tıklayın. İlk kareyi (Idle_0) sahneye sürükleyin.
2. Objenin adını **"Player"** yapın.
3. Inspector > **Sorting Layer** > `Player` seçin. (Karakter artık zeminin önünde görünür).

---

3. Adım: Fizik Bedeni (Rigidbody & Collider) 🍎

Karaktere fizik kurallarını öğretiyoruz.

1. Player seçiliyken `Add Component` > **Rigidbody 2D**.
    - **Gravity Scale:** Bunu `1` bırakırsanız ayda yürüyormuş gibi yavaş düşebilir. Platform oyunlarında genelde `3` veya `4` daha tok bir his verir.
    - ⚠️ **Kritik Ayar:** `Constraints` başlığını açın > **Freeze Rotation Z** işaretleyin. (Bunu yapmazsak karakter top gibi yuvarlanır).
2. `Add Component` > **Capsule Collider 2D**.
    - *Neden Kutu değil?* Kutunun köşeleri zemine takılabilir. Kapsülün altı yuvarlaktır, kayar gider.
    - **Edit Collider:** Butonuna basıp yeşil çerçeveyi karakterin bedenine tam oturacak şekilde daraltın. (Boşlukları alırsanız oyun daha adil hissettirir).

---

4. Adım: Animasyon Klipleri (Clips) 🎬

Unity'de animasyon yapmak aslında bir "Flipbook" (Defter kenarına çizilen hareketli resim) mantığıdır.

1. **Animation Penceresi:** `Window > Animation > Animation` yolunu izleyerek penceryi açın (Aşağıya sabitleyin).
2. Player objesi seçiliyken, Animation penceresinde **"Create"** butonuna basın.
3. `Assets` içine `_Animations` diye bir klasör açın ve dosyaya `Idle` (Durma) adını verip kaydedin.
4. **Kareleri Ekleme:**
    - Proje klasöründen `Ninja Frog > Idle` içindeki **bütün resimleri** seçin.
    - Animation penceresindeki zaman çizelgesine sürükleyip bırakın.
5. **Hız Ayarı:** Play'e bastığınızda karakter çok hızlı nefes alıyorsa, `Samples` değerini **20** yapın. (Samples görünmüyorsa pencerenin sağ üstündeki üç noktadan "Show Sample Rate" deyin).
6. **Koşu (Run) Animasyonu:**
    - Animation penceresinde sol üstteki `Idle` yazısına tıklayın > `Create New Clip`.
    - Adını `Run` koyun.
    - Bu sefer `Run (32x32)` içindeki resimleri sürükleyin.

---

5. Adım: Animasyon Beyni (Animator Controller) 🧠

Klipleri yaptık ama Unity ne zaman koşup ne zaman duracağını bilmiyor. Bunu bir şema ile anlatacağız.

1. Player seçiliyken **Animator** penceresini açın (Window > Animation > Animator).

2. Ekranda `Entry`, `Idle`, `Run` gibi kutular göreceksiniz.
    - (Turuncu kutu varsayılan animasyondur. `Idle` turuncu değilse, sağ tıkla > Set as Layer Default State de).
3. **Parametre Ekleme:**
    - Sol üstteki **Parameters** sekmesine geçin.
    - `+` butonuna basın > **Bool** seçin.
    - Adını **`isRunning`** koyun (Küçük/büyük harf uyumu çok önemli!).
4. **Geçişleri (Transitions) Kurma:**
    - `Idle` kutusuna sağ tıkla > **Make Transition** > Oku `Run` kutusuna bağla.
    - `Run` kutusuna sağ tıkla > **Make Transition** > Oku `Idle` kutusuna geri bağla.
5. **Kuralları Yazma:**
    - `Idle -> Run` okuna tıkla (Inspector açılır):
        - **Has Exit Time:** Tikini kaldır (Beklemeden hemen geçsin).
        - **Conditions:** `+`'ya bas. `isRunning` -> `true`.
    - `Run -> Idle` okuna tıkla:
        - **Has Exit Time:** Tikini kaldır.
        - **Conditions:** `+`'ya bas. `isRunning` -> `false`.

🚀 DERS SONU TESTİ

Play tuşuna basın.

1. **Fizik Testi:** Karakter yere düşüp "Ground" üzerinde duruyor mu? (Evet ise Rigidbody ve Collider çalışıyor).
2. **Animasyon Testi:** Karakter olduğu yerde nefes alıp veriyor mu? (`Idle` çalışıyor).
3. **Geçiş Testi:**
    - Oyun çalışırken **Animator** penceresini yan ekrana alın.
    - `isRunning` kutucuğuna elle tik atın.
    - Karakter koşmaya başlıyor mu? Tik'i kaldırınca duruyor mu?

🎮 Modül 3: Hareket Kodu (Player Controller Script)

Hedef: Karakteri klavye ile sağa/sola yürütmek, zıplatmak ve bu hareketleri animasyonlarla senkronize etmek.

---

1. Adım: Script Oluşturma ve Tanımlamalar

Önce beyni oluşturalım.

1. Project penceresinde `_Scripts` klasörüne girin.
2. Sağ tık > `Create` > `C# Script`. Adını **`PlayerMovement`** koyun. (Boşluk bırakmayın!).
3. Scripti sürükleyip sahnedeki **Player** objesinin üzerine bırakın.
4. Scripti çift tıklayıp açın.

Kodun içine girelim ve hazırlık kampındaki bilgilerimizi kullanalım. Karakteri kontrol etmek için 3 şeye ihtiyacımız var:

- Fizik bedeni (`Rigidbody2D`)
- Animasyon beyni (`Animator`)
- Sprite görseli (`SpriteRenderer` - Yönünü çevirmek için)

```
C#
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    // --- DEĞİŞKENLER (Kutularımız) ---
    private Rigidbody2D rb;
    private Animator anim;
    private SpriteRenderer sprite;

    private float dirX = 0f; // Yatay yön bilgisi (-1, 0, 1)

    // Editörden değiştirebileceğimiz ayarlar
    [SerializeField] private float moveSpeed = 7f;
    [SerializeField] private float jumpForce = 14f;

    // --- BAŞLANGIÇ (Tanışma) ---
    private void Start()
    {
        // Script başladığında karakterin üzerindeki bileşenleri bulup
```

```
kutulara koyuyoruz.
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        sprite = GetComponent<SpriteRenderer>();
    }
}
```

---

2. Adım: Yürüme Mantığı (Movement) 🏃

Fiziksel hareketi kodlayacağız. Burada `Input.GetAxisRaw` kullanacağız çünkü platform oyunlarında tuştan elini çekince karakterin *hemen* durmasını isteriz (Kaymasını değil).

Kodu `PlayerMovement` sınıfının içine, `Update` fonksiyonu olarak ekleyelim:

```
C#
   // --- OYUN DÖNGÜSÜ (Her Karede Çalışır) ---
    private void Update()
    {
        // 1. Klavyeden Yönü Oku (Sağ: 1, Sol: -1, Hiçbiri: 0)
        dirX = Input.GetAxisRaw("Horizontal");

        // 2. Fiziği Uygula (Hız Değiştirme)
        // Yeni Hız = (Yatay Yön * Hız, Mevcut Dikey Hız)
        rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);

        // 3. Animasyonu Güncelle
        UpdateAnimationState();
    }
```

Test Zamanı: Kodu kaydedip Unity'e dönün. Play'e basın. Karakter sağa sola gidiyor mu? (Evet, ama muhtemelen "Moonwalk" yapıyor, yani geri geri kayıyor ve animasyon yok).

---

3. Adım: Zıplama Mantığı (Jumping) 🦘

Zıplamak için dikey (Y) ekseninde ani bir hız vereceğiz.

Kodu `Update` fonksiyonunun içine, yürüme kodunun hemen altına ekleyin:

```
C#
        // Zıplama Tuşu (Space)
         if (Input.GetButtonDown("Jump")) // Space tuşu
         {
             rb.velocity = new Vector2(rb.velocity.x, jumpForce);
         }
```

---

4. Adım: Animasyon ve Yön Düzeltme 🎭

Karakter sağa giderken sağa, sola giderken sola bakmalı. Ve koşarken "Run" animasyonu girmeli. Bunun için ayrı bir metot yazmak kodu temiz tutar.

`Update` fonksiyonunun altına şu yeni metodu ekleyin:

```
C#
    private void UpdateAnimationState()
     {
         // --- KOŞU KONTROLÜ ---
         if (dirX > 0f) // Sağa gidiyor
         {
             anim.SetBool("isRunning", true); // Koşu animasyonunu aç
             sprite.flipX = false;            // Yüzünü sağa dön
         }
         else if (dirX < 0f) // Sola gidiyor
         {
             anim.SetBool("isRunning", true); // Koşu animasyonunu aç
             sprite.flipX = true;             // Yüzünü sola dön (Flip)
         }
         else // Duruyor (dirX == 0)
         {
             anim.SetBool("isRunning", false); // Idle animasyonuna dön
         }
     }
```

---

5. Adım: Sonsuz Zıplama Hatası (The Bug) 🐛

Şu an kodumuzda büyük bir "Hata" (aslında eksiklik) var. Play'e basıp Space tuşuna arka arkaya basarsanız karakterin uçarak ekrandan çıktığını göreceksiniz. Çünkü "Yerde miyim?" diye sormuyoruz.

Bunu çözmek için Unity'nin **Layer (Katman)** sistemini kullanacağız.

1. **Layer Ayarı:** Unity sağ üst köşede `Layers` > `Edit Layers`.
2. Boş bir yere (örn: Layer 6) **"Ground"** yazın.
3. Hierarchy'den **Grid > Ground** objesini seçin ve Inspector'dan Layer'ını **"Ground"** yapın. (Çıkan soruya "Yes, change children" deyin).
4. **Koda Dönüş:**

Kodu şu şekilde güncelleyeceğiz. Önce en üste yeni bir değişken ekleyin:

C#

[SerializeField] private LayerMask jumpableGround; // Hangi katman zıplanabilir?

private BoxCollider2D coll; // Kendi collider'ımız

`Start` fonksiyonuna collider'ı ekleyin:

```
C#
   coll = GetComponent<BoxCollider2D>(); // CapsuleCollider
kullanıyorsan CapsuleCollider2D yaz!
```

`Update` içindeki zıplama kodunu şununla değiştirin:

```
C#
   // Eğer zıplama tuşuna basıldıysa VE Yerdeysek (IsGrounded)
    if (Input.GetButtonDown("Jump") && IsGrounded())
    {
        rb.velocity = new Vector2(rb.velocity.x, jumpForce);
    }
```

Ve en alta yeni bir "Kontrolcü Metot" yazın:

```
C#
   // Yerde miyiz kontrolü yapan özel fonksiyon
    private bool IsGrounded()
    {
        // Collider'ımız "jumpableGround" katmanına değiyor mu?
        return Physics2D.BoxCast(coll.bounds.center, coll.bounds.size,
0f, Vector2.down, .1f, jumpableGround);
    }
```

---

6. Adım: Son Dokunuşlar (Editör Ayarı)

Unity'e geri dönün. Script'i kaydettiniz ama karakter hala sonsuz zıplayabilir veya hiç zıplamayabilir. Çünkü editörde ayar yapmadık.

1. **Player** objesini seçin.
2. **Player Movement (Script)** bileşenine bakın.
3. **Jumpable Ground** kısmında "Nothing" yazar. Onu tıklayıp **"Ground"** seçin.
4. **Move Speed** ve **Jump Force** değerleriyle oynayarak en keyifli hissi yakalayın.

🚀 KONTROL LİSTESİ

Artık öğrencileriniz şunları yapabiliyor olmalı:

1. [ ] Karakter klavye ile sağa/sola koşuyor.
2. [ ] Sola giderken yüzünü sola dönüyor.
3. [ ] Durunca nefes alıyor (Idle), yürüyünce koşuyor (Run).
4. [ ] Sadece yerdeyken zıplayabiliyor (Sonsuz uçuş yok).

🍒 Modül 4: Ödül Sistemi (Collectibles & UI)

Hedef: Sahneye toplanabilir meyveler eklemek, bunlara temas edince yok etmek ve ekrandaki skoru güncellemek.

---

1. Adım: Elmanın Yaratılışı (Animation) 🍎

Önce nesneyi görsel olarak hazırlayalım.

1. **Görseller:** `Pixel Adventure 1 > Items > Fruits > Apple` klasörüne gidin.
2. Klasör içindeki **bütün resimleri** seçip sahneye sürükleyin.
3. Unity otomatik olarak "Create New Animation" diyecek. `_Animations` klasörüne `Apple_Idle` adıyla kaydedin.
4. Objenin adını Hierarchy'de **"Apple"** olarak değiştirin.
5. **Sorting Layer:** Meyveler önde durmalı. Inspector'dan Sorting Layer'ı `Foreground` (veya Player ile aynı) yapın.

---

2. Adım: Hayalet Fizik (Is Trigger) 👻

Meyveye bir fizik ekleyeceğiz ama karakterin ona çarpıp kafasını vurmasını istemiyoruz. İçinden geçip gitmeli.

1. **Collider Ekleme:** Apple seçiliyken `Add Component` > **Circle Collider 2D**.
2. ⚠️ **Kritik Ayar:** Collider bileşenindeki **"Is Trigger"** kutucuğunu işaretleyin.
   - *Metafor:* Normal Collider "Duvar" gibidir, çarpınca durdurur. Trigger ise "Lazer Sensörü" gibidir, içinden geçersin ama alarm çalar.
3. **Etiketleme (Tag):**
   - Inspector'ın en üstünde `Tag: Untagged` yazar.
   - Açılır menüden `Add Tag` deyin.
   - `+` butonuna basıp **"Fruits"** (veya Apple) adında bir Tag oluşturun.
   - **Önemli:** Tekrar Apple objesini seçip Tag kısmından yeni oluşturduğunuz **"Fruits"** etiketini seçmeyi unutmayın!

---

3. Adım: Toplama Kodu (Scripting Interaction) 📜

Şimdi "Temas edince ne olsun?" sorusunu kodlayacağız. Bu kodu meyveye değil, **Player**'a yazacağız (çünkü puanı Player tutacak).

1. `_Scripts` klasörüne yeni bir C# Script oluşturun: **`ItemCollector`**.
2. Scripti **Player** objesinin üzerine sürükleyin.
3. Scripti açın ve şu kodları yazın:

```
C#
using UnityEngine;
using UnityEngine.UI; // UI işlemleri için gerekli kütüphane

public class ItemCollector : MonoBehaviour
{
    private int fruits = 0; // Toplanan meyve sayısı

    // --- TEMAS ANI (Sadece Trigger olanlar için çalışır) ---
    private void OnTriggerEnter2D(Collider2D collision)
    {
        // Çarptığımız şeyin etiketi "Cherry" mi?
        if (collision.gameObject.CompareTag("Fruits"))
        {
            Destroy(collision.gameObject); // Meyveyi sahneden yok et
            fruits++; // Puanı 1 artır
            Debug.Log("Meyve Sayısı: " + fruits); // Konsola yaz
(Şimdilik)
        }
    }
}
```

Test: Oyunu başlatın. Karakter meyvenin içinden geçince meyve kayboluyor mu? Konsolda sayı artıyor mu?

---

4. Adım: Arayüz (UI) ve Skor Tablosu 💯

Konsoldaki yazıyı oyuncu göremez. Ekrana bir yazı yazmamız lazım.

1. **Canvas Oluşturma:** Hierarchy'de sağ tık > `UI` > **Text - TextMeshPro**.
    - *Uyarı:* Unity "Import TMP Essentials" diye bir pencere açarsa **"Import TMP Essentials"** butonuna basın.
2. **Canvas Ayarı:** Oluşan `Canvas` objesini seçin. Inspector'da `UI Scale Mode` ayarını `Scale With Screen Size` yapın. (Böylece oyun penceresi büyüse de yazılar bozulmaz).
3. **Text Ayarı:**
    - Oluşan `Text (TMP)` objesini seçin.
    - Metne "Meyve: 0" yazın.
    - Fontu büyütün, rengini sarı yapın ve ekranın sol üst köşesine yerleştirin.

- Objenin adını **"ScoreText"** yapın.

---

5. Adım: Kodu UI ile Bağlama 🔗

Yazdığımız `ItemCollector` kodunu güncelleyerek bu yazıyı değiştirmesini sağlayacağız.

Kodu şu şekilde güncelleyin:

```
C#
using UnityEngine;
using TMPro; // TextMeshPro için gerekli kütüphane! (Bunu unutma)

public class ItemCollector : MonoBehaviour
{
    private int fruits = 0;

    // Editörden sürükleyip bırakacağımız yazı kutusu
    [SerializeField] private TextMeshProUGUI fruitsText;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Fruits"))
        {
            Destroy(collision.gameObject);
            cherries++;
            // Yazıyı güncelle: "Meyve: 7"
            fruitsText.text = "Meyve: " + fruits;
        }
    }
}
```

Son Bağlantı:

1. Unity'e dönün. **Player** objesini seçin.
2. **Item Collector (Script)** bileşeninde `Fruits Text` kutusunun boş olduğunu göreceksiniz.
3. Hierarchy'deki **"ScoreText"** objesini sürükleyip bu boş kutuya bırakın.

---

6. Adım: Seri Üretim (Prefabs) 🏭

Tek bir elmamız var ve çalışıyor. Ama sahneye 50 tane elma koymak için hepsini baştan mı yapacağız? Hayır! **Prefab** kullanacağız.

1. Sahnedeki **Apple** objesini sürükleyip `Assets` penceresindeki bir klasöre (örn: `_Prefabs`) bırakın.
2. Objenin ikonu maviye dönüşecektir.
3. Artık bu mavi **Apple** dosyasını sahnenin istediğiniz yerine sürükleyip bırakabilirsiniz. Hepsi aynı özelliklere (Animasyon, Trigger, Tag) sahip olacaktır.
4. Eğer bir özelliği değiştirmek isterseniz, klasördeki Prefab'a çift tıklayıp değiştirirseniz sahnedeki hepsi güncellenir!

🚀 KONTROL LİSTESİ

Öğrencileriniz şunları başarmış olmalı:

1. [ ] Meyveler havada dönüyor (Animasyon).
2. [ ] İçinden geçince yok oluyorlar (Trigger & Destroy).
3. [ ] Sol üstteki yazı her meyve toplandığında artıyor (UI).
4. [ ] Sahneye Prefab sistemiyle kolayca çok sayıda meyve ekleyebiliyorlar.

💀 Modül 5: Tuzaklar ve Ölüm (Traps & Game Over)

Hedef: Sahneye ölümcül tuzaklar eklemek, oyuncu bunlara değdiğinde ölmesini sağlamak ve bölümü baştan başlatmak.

---

1. Adım: Tehlikeyi Yaratmak (Spikes & Saws) ⚙️

Önce sahnemize can yakacak objeler ekleyelim.

Seçenek A: Dikenler (Spikes - Basit)

1. `Pixel Adventure 1 > Traps > Spikes` klasörüne gidin.
2. `Idle (16x16)` görselini sahneye sürükleyin.
3. Objenin adını **"Spike"** yapın.
4. `Add Component` > **Box Collider 2D**.
5. **Tag Oluşturma:**
    - Tag menüsü > Add Tag > **"Trap"** (Tuzak) isminde yeni bir etiket oluşturun.
    - Spike objesini seçip Tag'ini **"Trap"** yapın.
6. **Layer:** Sorting Layer'ı `Foreground` yapın ki karakterin arkasında kalmasın.

Seçenek B: Testere (Saw - Hareketli ve Havalı)

1. `Pixel Adventure 1 > Traps > Saw` klasörüne gidin.
2. Bütün resimleri seçip sahneye atın (Animasyon oluşsun: `Saw_Spin`).
3. `Add Component` > **Circle Collider 2D**. (Testere yuvarlaktır!).
4. Tag: **"Trap"**.

---

2. Adım: Çarpışma Mantığı (Trigger vs Collision) 💥

Elmalarda `Is Trigger` açmıştık çünkü içinden geçmek istiyorduk. Tuzaklarda ise genelde `Is Trigger` **KAPALI** tutulur (yani kutucuk işaretsiz). Böylece oyuncu tuzağa fiziksel olarak çarpar.

Bu farkı kodda da göreceğiz:

- Elma (Trigger) -> `OnTriggerEnter2D`
- Tuzak (Solid) -> `OnCollisionEnter2D`

---

3. Adım: Yaşam Döngüsü Scripti (PlayerLife) 🧬

Karakterin hareket kodlarını `PlayerMovement` içine yazmıştık. Ölüm kodlarını oraya yazıp kodu çorbaya çevirmeyelim. Yeni bir script açalım.

1. `_Scripts` klasörüne **`PlayerLife`** adında bir script oluşturun.
2. Scripti **Player** objesine atın.
3. İçini şöyle dolduralım:

```
C#
using UnityEngine;
using UnityEngine.SceneManagement; // Sahne işlemleri için MUTLAKA
gerekli!

public class PlayerLife : MonoBehaviour
{
    private Animator anim;
    private Rigidbody2D rb;

    private void Start()
    {
        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
    }

    // --- ÇARPIŞMA ANI (Katı objeler için) ---
    private void OnCollisionEnter2D(Collision2D collision)
    {
        // Çarptığımız şey "Trap" etiketine sahip mi?
        if (collision.gameObject.CompareTag("Trap"))
        {
            Die(); // Ölüm fonksiyonunu çağır
        }
    }

    private void Die()
    {
        // 1. Fiziksel hareketi durdur (Static yaparsak olduğu yerde
donar)
        rb.bodyType = RigidbodyType2D.Static;

        // 2. Ölüm animasyonunu oynat
        anim.SetTrigger("death");

        // 3. Bölümü yeniden başlat (Ama hemen değil, animasyon bitince)
    }
```

```
    // Bu fonksiyonu birazdan animasyonun sonuna ekleyeceğiz
    private void RestartLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

---

4. Adım: Ölüm Animasyonu Kurulumu ☠️

Kodda `anim.SetTrigger("death")` dedik ama Unity bunu henüz bilmiyor.

1. **Animasyon Oluşturma:**
    - Player seçiliyken Animation penceresi > `Create New Clip` > **`Death`**.
    - `Main Characters > Ninja Frog > Desappearing (96x96)` veya `Hit` görsellerini kullanabilirsiniz. (Genelde patlama efekti kullanılır).
2. **Animator Ayarı:**
    - Animator penceresini açın.
    - `Any State` kutusunu bulun (Mavi kutu).
    - `Any State`'ten `Death` kutusuna ok çekin (Transition).
        - *Anlamı:* Koşarken de, dururken de, zıplarken de ölebilirim.
3. **Parametre (Trigger):**
    - Parameters sekmesi > `+` > **Trigger** > Adını **"death"** koyun.
4. **Ok Ayarları:**
    - Oka tıklayın.
    - **Conditions:** `death` ekleyin.
    - **Has Exit Time:** Kapalı.
    - *İpucu:* Can Settings (Settings okuna tıkla) > **Transition Duration**'ı `0` yapın ki anında ölsün, gecikme olmasın.

---

5. Adım: Animation Events (Büyülü Dokunuş) ✨

Öldükten hemen sonra oyun başa sararsa, oyuncu ölüm animasyonunu izleyemez. "RestartLevel" fonksiyonunu animasyonun **tam bittiği anda** çağırmalıyız.

Bunun için harika bir yöntem var: **Animation Events**.

1. **Player** objesini seçin.
2. **Animation** penceresini açın ve `Death` klibini seçin.
3. Zaman çubuğunu (beyaz çizgi) animasyonun en son karesine getirin.

4. Soldaki küçük **"Add Event"** butonuna (Dikdörtgen içinde artı işareti gibi görünür) tıklayın.
5. Timeline üzerinde küçük beyaz bir çentik oluşacak. Ona tıklayın.
6. Inspector'da **Function** kısmında scriptimizdeki `RestartLevel` fonksiyonunu seçin.

Eğer listede `RestartLevel` görünmüyorsa scripti kaydettiğinizden ve fonksiyonun `private` olmadığından emin olun (Public yapmanız gerekebilir, ama Animation Event genelde private'ı da görür).

🚀 KONTROL LİSTESİ

Artık oyunun bir döngüsü var!

1. [ ] Dikenlere veya testereye dokununca karakter olduğu yerde donuyor (Rigidbody Static).
2. [ ] Patlama/Ölüm animasyonu oynuyor.
3. [ ] Animasyon bitince sahne en baştan başlıyor.

🤖 Modül 6: Hareketli Platformlar ve Düşmanlar

Hedef: Nesnelerin iki nokta (A ve B) arasında gidip gelmesini sağlamak ve karakterin hareketli zemin üzerinde kaymadan durmasını çözmek.

---

1. Adım: Hareketli Platform Görseli 🚋

Önce üzerinde durabileceğimiz hareketli bir zemin hazırlayalım.

1. `Pixel Adventure 1 > Traps > Platforms` klasörüne gidin.
2. `Brown Off` (veya On) görselini sahneye atın.
3. Adını **"MovingPlatform"** yapın.
4. **Componentler:**
    - `Box Collider 2D`.
    - **Layer:** `Ground` yapın (Böylece üzerine zıplayabiliriz).
    - **Tag:** `Untagged` kalsın (Tuzak değil sonuçta).

---

2. Adım: Yol Noktaları (Waypoints) Oluşturma 📍

Unity'de bir objeyi hareket ettirmek için ona "Nereye gideceğini" söylememiz lazım.

1. Hierarchy'de boş bir obje oluşturun (`Create Empty`). Adını **"PlatformSystem"** koyun.
2. Az önce oluşturduğunuz **"MovingPlatform"** objesini sürükleyip bu **"PlatformSystem"**in içine atın (Child yapın).
3. **"PlatformSystem"**in içine iki tane daha boş obje oluşturun:
    - Adları **"Point A"** ve **"Point B"** olsun.
    - Bu noktaları sahnede platformun gidip gelmesini istediğiniz yerlere taşıyın. (Sarı veya mavi ikon vererek görünür yapabilirsiniz: Inspector sol üstündeki küp ikonundan).

Hiyerarşi şöyle görünmeli:

- 🔻 PlatformSystem
    - MovingPlatform (Görselimiz)
    - Point A (Hedef 1)
    - Point B (Hedef 2)

---

3. Adım: Hareket Kodu (WaypointFollower) 📜

Şimdi platforma "Sırayla bu noktalara git" diyen kodu yazacağız.

1. `_Scripts` klasörüne **`WaypointFollower`** adında bir script açın.
2. Bu scripti **MovingPlatform** objesine (System'e değil, görsel olana!) atın.

```
C#
using UnityEngine;

public class WaypointFollower : MonoBehaviour
{
    // Gidilecek noktaların listesi
    [SerializeField] private GameObject[] waypoints;
    private int currentWaypointIndex = 0; // Şu an hangi noktaya
gidiyoruz?

    [SerializeField] private float speed = 2f;

    private void Update()
    {
        // 1. Hedef ile aramızdaki mesafeyi ölç
        // Vector2.Distance(BenimKonumum, HedefKonum)
        if
(Vector2.Distance(waypoints[currentWaypointIndex].transform.position,
transform.position) < .1f)
        {
            // Hedefe çok yaklaştık, sonraki noktaya geç
            currentWaypointIndex++;

            // Eğer son noktaya geldiysek başa dön (Mod alma işlemi)
            if (currentWaypointIndex >= waypoints.Length)
            {
                currentWaypointIndex = 0;
            }
        }

        // 2. Hedefe doğru hareket et
        // Vector2.MoveTowards(Neredeyim, Nereye, Hızım)
        transform.position = Vector2.MoveTowards(transform.position,
waypoints[currentWaypointIndex].transform.position, Time.deltaTime *
speed);
    }
}
```

Bağlantıyı Kurma:

1. Unity'e dönün. **MovingPlatform** objesini seçin.
2. Scriptteki `Waypoints` listesini açın. Size: 2 yapın.
3. Element 0'a **Point A**'yı, Element 1'e **Point B**'yi sürükleyin.

Play'e basın. Platform gidip geliyor mu? Evet! 🎉 Ama üzerine atlayınca büyük bir sorun göreceksiniz: **Karakter kayıp düşüyor!**

---

4. Adım: Yapışkan Platform Sorunu (Sticky Platform) 🍯

Gerçek hayatta bir otobüse bindiğinizde, otobüsle birlikte hareket edersiniz. Ama Unity'de karakter platformun üstündedir ama onun "parçası" değildir. Platform altından kayar gider.

Bunu çözmek için karakteri platformun **"Çocuğu (Child)"** yapacağız.

1. `_Scripts` klasörüne **`StickyPlatform`** adında yeni bir script açın.
2. Bunu **MovingPlatform** objesine atın.

```
C#
using UnityEngine;

public class StickyPlatform : MonoBehaviour
{
    // Üstüme biri bastığında
    private void OnCollisionEnter2D(Collision2D collision)
    {
        // Çarpan şeyin adı "Player" ise
        if (collision.gameObject.name == "Player")
        {
            // Onu benim çocuğum yap (Böylece ben nereye, o oraya)
            collision.gameObject.transform.SetParent(transform);
        }
    }

    // Üstümden ayrıldığında
    private void OnCollisionExit2D(Collision2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            // Ebeveynliğini iptal et (Özgür bırak)
            collision.gameObject.transform.SetParent(null);
        }
    }
}
```

Test: Tekrar deneyin. Artık karakter platformun üzerindeyken onunla birlikte seyahat etmeli.

---

5. Adım: Düşmanları Hareket Ettirmek (Testere) ⚙️

Yazdığımız `WaypointFollower` scripti o kadar genel bir kod ki, bunu düşmanlar için de kullanabiliriz!

1. Sahnede daha önce oluşturduğumuz veya yeni bir **"Saw"** (Testere) bulun. (Tag'i "Trap" olsun).
2. Tıpkı platformda yaptığımız gibi bir Hiyerarşi kurun:
    - 🔻 SawSystem
        - Saw (Script burada olacak)
        - Point 1
        - Point 2
        - Point 3 (İsterseniz üçgen çizdirin!)
3. **Saw** objesine `WaypointFollower` scriptini atın.
4. Noktaları sürükleyip bırakın.
5. Hızını (`Speed`) biraz artırın (Örn: 5).

Sonuç: Artık sadece duran dikenler değil, devriye gezen ölümcül testereleriniz var. Oyuncu bunlardan kaçmak için zamanlamayı tutturmak zorunda!

🚀 DERS SONU KONTROL LİSTESİ

Öğrencileriniz şunları başarmış olmalı:

1. [ ] Platform iki nokta arasında otomatik gidip geliyor.
2. [ ] Karakter platformun üzerine binince kaymıyor (Parenting mantığı).
3. [ ] Aynı mantıkla hareket eden bir Testere (Saw) veya Düşman var.

🏆 Modül 7: Bitiş Çizgisi ve Build Alma

Hedef: Bölüm sonuna bir kupa koymak, alınca sonraki bölüme geçmek ve oyunu bilgisayarda çalışacak bir dosya (.exe) olarak dışarı aktarmak.

---

1. Adım: Kupa Görseli (The Trophy) 🏺

Önce hedefini belirleyelim.

1. `Pixel Adventure 1 > Items > Checkpoints > End (Pressed) (64x64)` klasörüne gidin.
2. Bütün resimleri seçip sahneye atın.
3. Animasyon adı: **`Finish_Idle`**.
4. Objenin adı: **"Finish"**.
5. **Bileşenler:**
    - `Box Collider 2D`.
    - **Is Trigger:** ✅ (İşaretli). Oyuncu kupaya çarpmaz, içine girer.
6. **Tag:** Yeni bir tag oluşturmaya gerek yok, çünkü çarpışma kodunu kupanın kendisine yazacağız.

---

2. Adım: İkinci Bölümü Yaratmak (Scene Duplication) 🎬

Sonraki bölüme geçmek için... bir sonraki bölümün olması lazım! Sıfırdan yapmaya gerek yok, Level 1'i kopyalayacağız.

1. Project penceresinde `_Scenes` klasörüne gidin.
2. Mevcut sahneniz (adı `SampleScene` olabilir) seçiliyken **Ctrl+D** (Duplicate) yapın.
3. Kopyanın adını **"Level 2"** yapın. İlk sahnenin adını da **"Level 1"** olarak düzeltin.
4. **Level 2**'yi çift tıklayıp açın.
    - *Fark olsun diye:* Arka plan rengini değiştirin veya platformların yerini biraz kaydırın. Böylece geçiş yaptığımızı anlayalım.

---

3. Adım: Build Settings (Sahne Listesi) 📋

Unity'ye "Benim oyunumda şu sahneler var" dememiz lazım. Yoksa kodla sahne çağıramayız.

1. `File` > `Build Settings` menüsünü açın.
2. `_Scenes` klasöründeki **Level 1** ve **Level 2** dosyalarını sürükleyip bu penceredeki **"Scenes In Build"** kutusuna bırakın.
3. **Sıralama Önemli:**

- Level 1 (Sağında `0` yazar).
- Level 2 (Sağında `1` yazar).

---

4. Adım: Bitiş Kodu (FinishPoint Script) 📜

Şimdi kupaya şu zekayı vereceğiz: "Biri bana değerse, sıradaki sahneyi yükle."

1. `_Scripts` klasörüne **`Finish`** adında bir script açın.
2. Scripti **Finish** (Kupa) objesine atın.

```
C#
using UnityEngine;
using UnityEngine.SceneManagement; // Sahne kütüphanesi

public class Finish : MonoBehaviour
{
    private bool levelCompleted = false; // Aynı anda 2 kere çalışmasın
diye kilit

    private void OnTriggerEnter2D(Collider2D collision)
    {
        // Çarpan kişi Player mı VE Bölüm zaten bitmediyse
        if (collision.gameObject.name == "Player" && !levelCompleted)
        {
            levelCompleted = true;
            // Hemen geçmeyelim, zafer hissi için biraz bekleyelim
(opsiyonel ses eklenebilir)
            Invoke("CompleteLevel", 2f); // 2 saniye sonra
CompleteLevel'ı çalıştır
        }
    }

    private void CompleteLevel()
    {
        // Aktif sahnenin numarasını (Index) al ve 1 ekle
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex
+ 1);
    }
}
```

⚠️ **Önemli Not:** Eğer Level 2'den sonra Level 3 yoksa oyun hata verir veya durur. Basit çözüm için kodun sonunu şöyle güncelleyebilirsiniz (Döngüye sokmak):

```
C#
   // (Alternatif - Döngüsel Mantık)
    int nextSceneIndex = SceneManager.GetActiveScene().buildIndex + 1;

    // Eğer sıradaki numara toplam sahne sayısına eşitse, başa (0) dön
    if (nextSceneIndex == SceneManager.sceneCountInBuildSettings)
    {
        nextSceneIndex = 0;
    }

    SceneManager.LoadScene(nextSceneIndex);
```

---

5. Adım: Oyunu Paketlemek (Build & Run) 📦

İşte o an! Öğrencilerin emeklerini somutlaştırdığı an.

1. `File` > `Build Settings`.
2. **Platform:** `Windows, Mac, Linux` seçili olsun.
3. **Player Settings (Sol alt köşe):**
    - **Company Name:** Kendi adlarını yazsınlar.
    - **Product Name:** "Tiny Hero".
    - **Resolution:** `Fullscreen Mode` > `Windowed` (Pencere modu) yaparlarsa hataları daha rahat görürler, ama Fullscreen daha havalıdır.
4. Pencereyi kapatıp **Build And Run** butonuna basın.
5. Masaüstünde yeni bir klasör açın (Adı "Oyunum" olsun) ve onu seçin.

Unity derlemeyi bitirecek ve oyununuz tam ekran olarak açılacak! 🎮

3D Projeye geçerken VSCode'a kurulacak packler;
-Unity
-Unity Tools
-Unity Code Snippets
-C# Dev Kit
-Debugger for unity 301
-Prettier Code Formatter
-Rainbow brackets