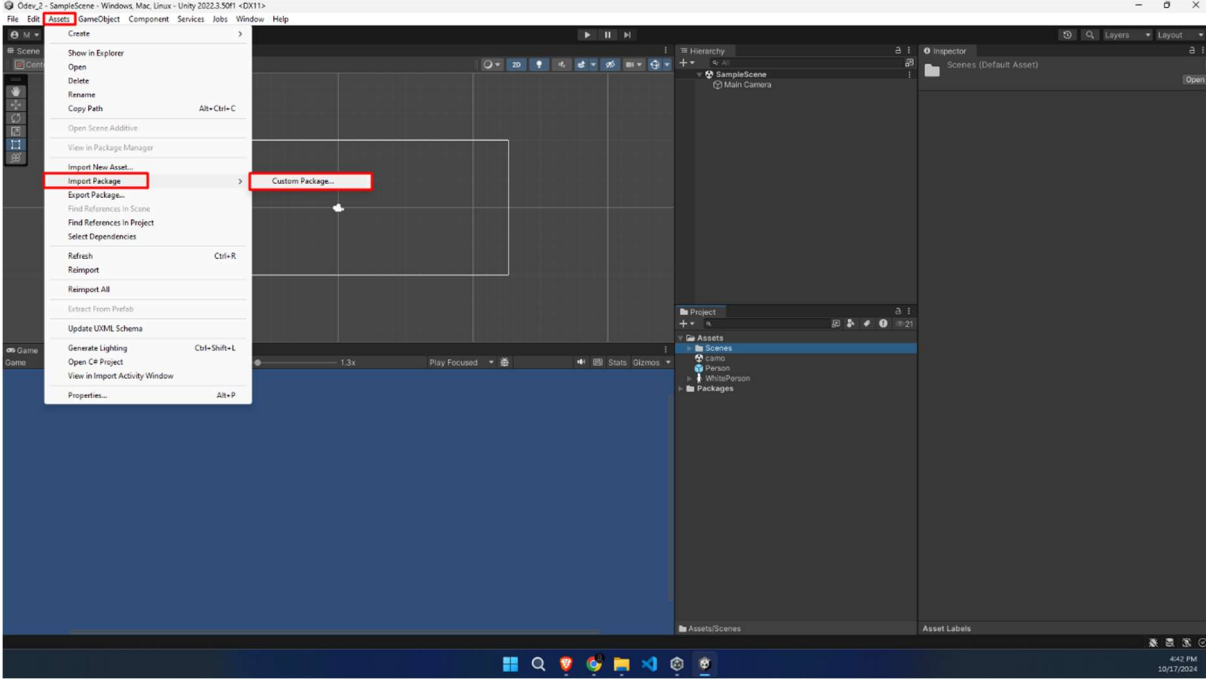


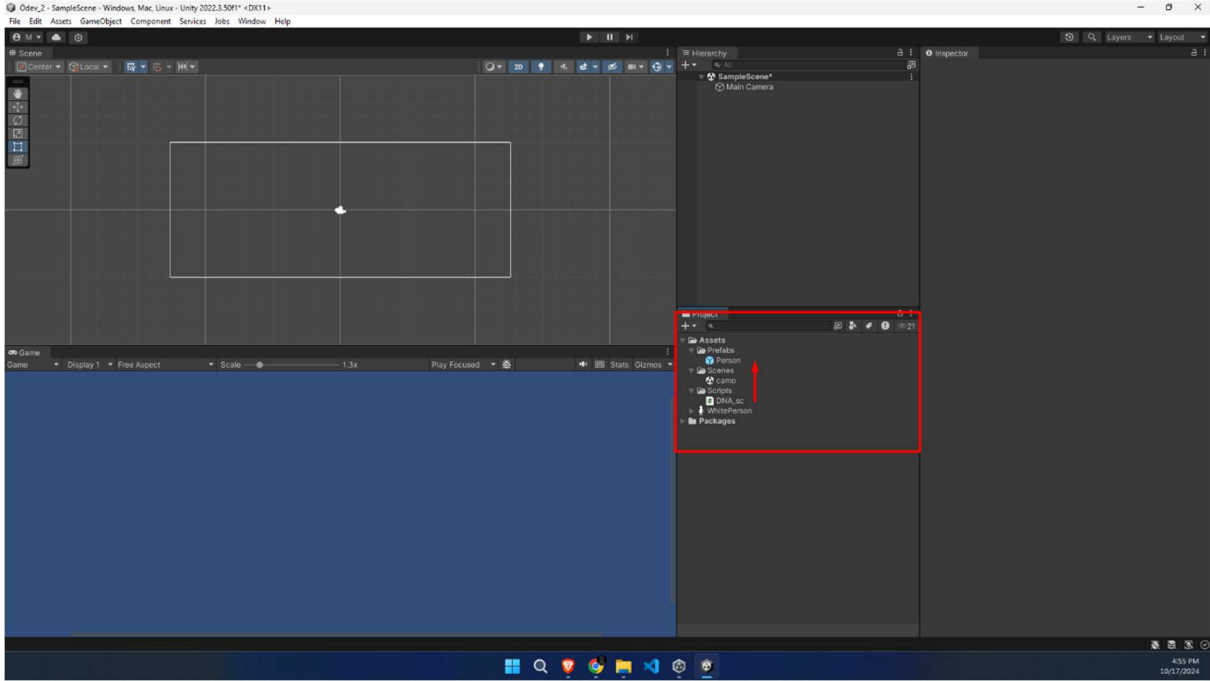
# Bilgisayar Oyunlarında Yapay Zekâ - Ödev 2

## 1 – Hazır paket yükleme:

1.1: Üst menüden Assets->Import Package->Custom Package adımlarını izleyerek “CamoGATraining.unitypackage” adlı dosyayı seçiyoruz.



1.2: Project sekmesinde bulunan Assets klasörü içine “Prefabs” ve “Scripts” adında iki klasör oluşturuyoruz. “Person” nesnesini Prefabs klasörüne ekliyoruz. Scenes klasörü altındaki SampleScene’yi sildikten sonra “camo” dosyasını Scenes klasörüne atıyoruz ve üzerine iki kez tıklayarak sahneye geçiş yapıyoruz. Scripts klasörü altında DNA\_sc adında bir dosya oluşturuyoruz.



## 2 – DNA\_sc scriptini düzenleme

### 2.1 – Renk kodları için:

// Renk kodları geni için değişkenler

```
public float r;
```

```
public float g;
```

```
public float b;
```

### Ölüm ve yaşam süresi takibi için:

// Ölüm ve yaşam süresi takibi için değişkenler

```
public bool isDead = false;
```

```
public float timeToDie = 0f;
```

### Nesne öldükten sonra görünürlüğünü ayarlamak için:

```
private SpriteRenderer sRenderer;
```

### Nesne öldükten sonra çarpışma durumunu ayarlamak için:

```
private Collider2D sCollider;
```

kodlarını ekliyoruz.

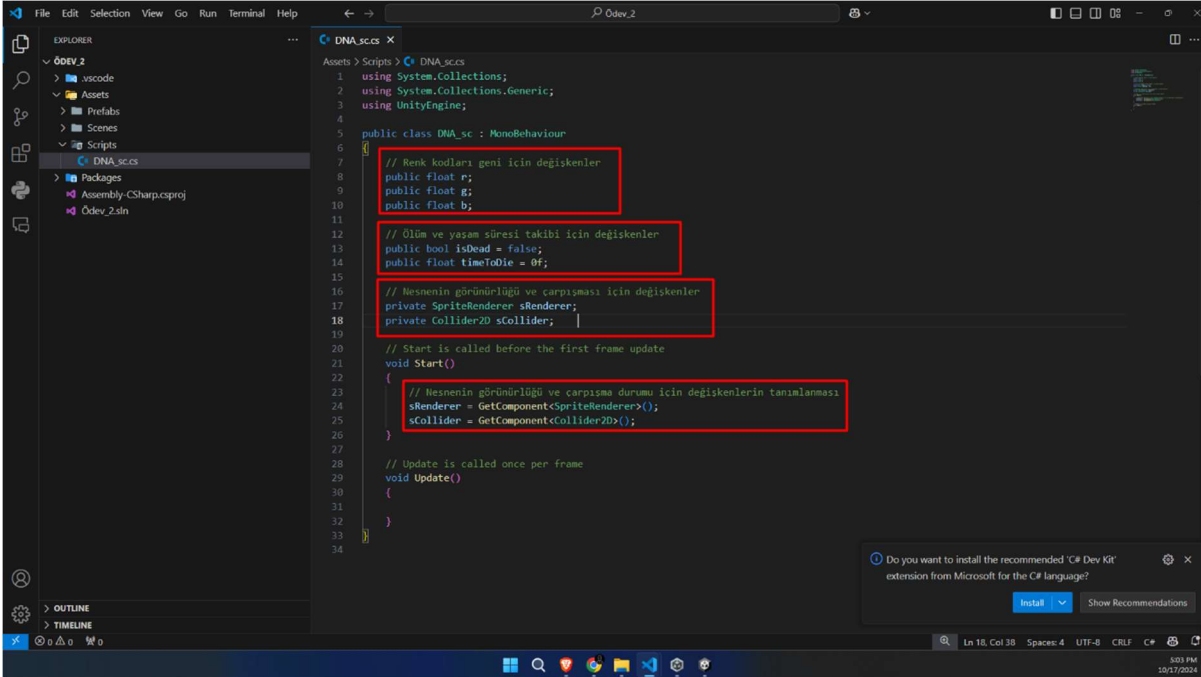
Nesne arpıřma durumu ve grnrlğn durumunu tanımlamak iin Start fonksiyonu iine:

// Nesnenin grnrlğ ve arpıřma durumu iin değışkenlerin tanımlanması

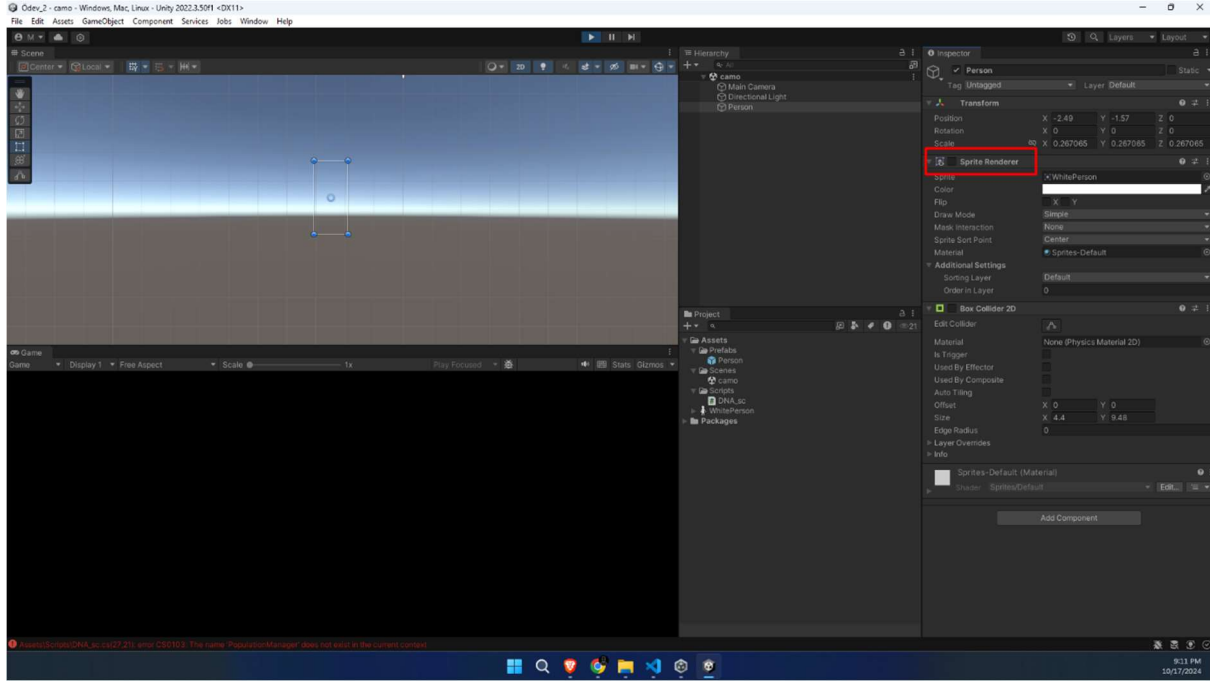
sRenderer = GetComponent<SpriteRenderer>();

sCollider = GetComponent<Collider2D>();

kodlarını ekliyoruz.



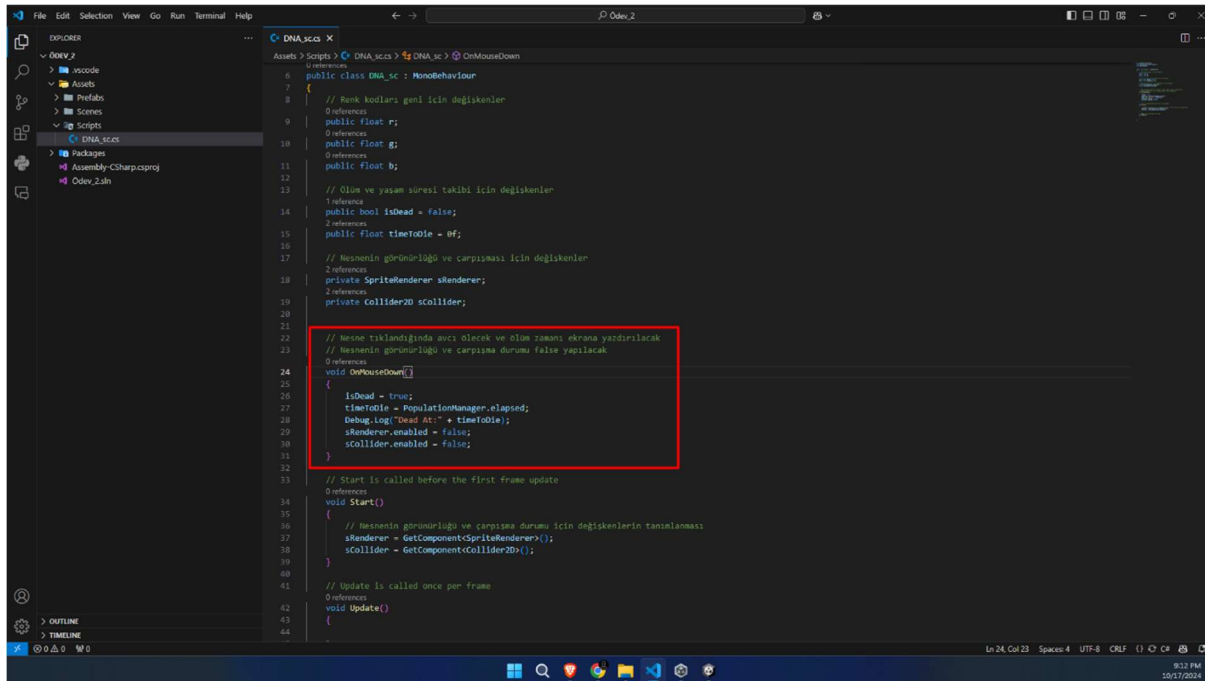
Yaptığımız deęiřikliklerin uygulanıp uygulanmadığını kontrol edebilmek için Person nesnesini Hierarchy sekmesinin içine sürükleyeruz. Person nesnesi üzerine tıklayınca sağda çıkan Inspector sekmesinde bulunan Sprite Renderer başlığındaki tiki kaldırırız. Person nesnesi görünmez duruma gelmiş oldu. Ardından tiki tekrar açırız.



### 3 – Fare tıklama kontrolü

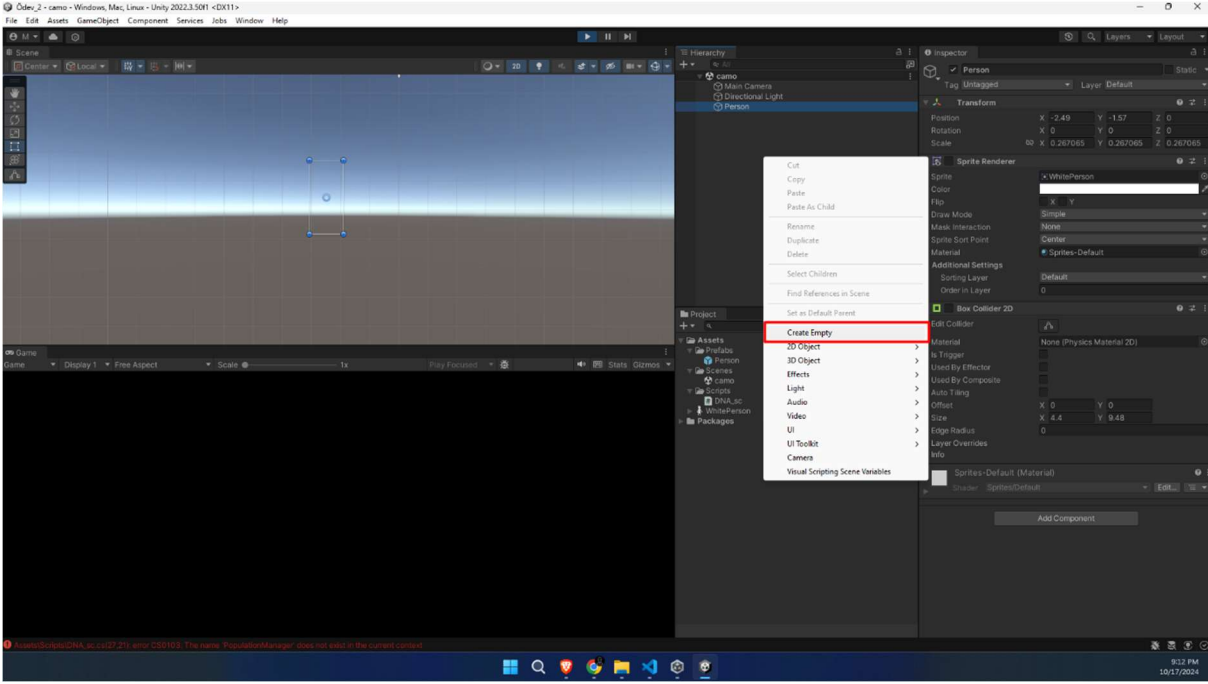
3.1 – Nesnemizin üzerine tıklandığında nesnemizin ölmesini istiyoruz. Bu yüzden Fare sol tıklamasını kontrol eden OnMouseDown fonksiyonunu ekleyerek içerisinde karakterimizin ölme durumunu kontrol ediyoruz.

```
void OnMouseDown()  
{  
    isDead = true;  
    timeToDie = PopulationManager.elapsed;  
    Debug.Log("Dead At: " + timeToDie);  
    sRenderer.enabled = false;  
    sCollider.enabled = false;  
}
```

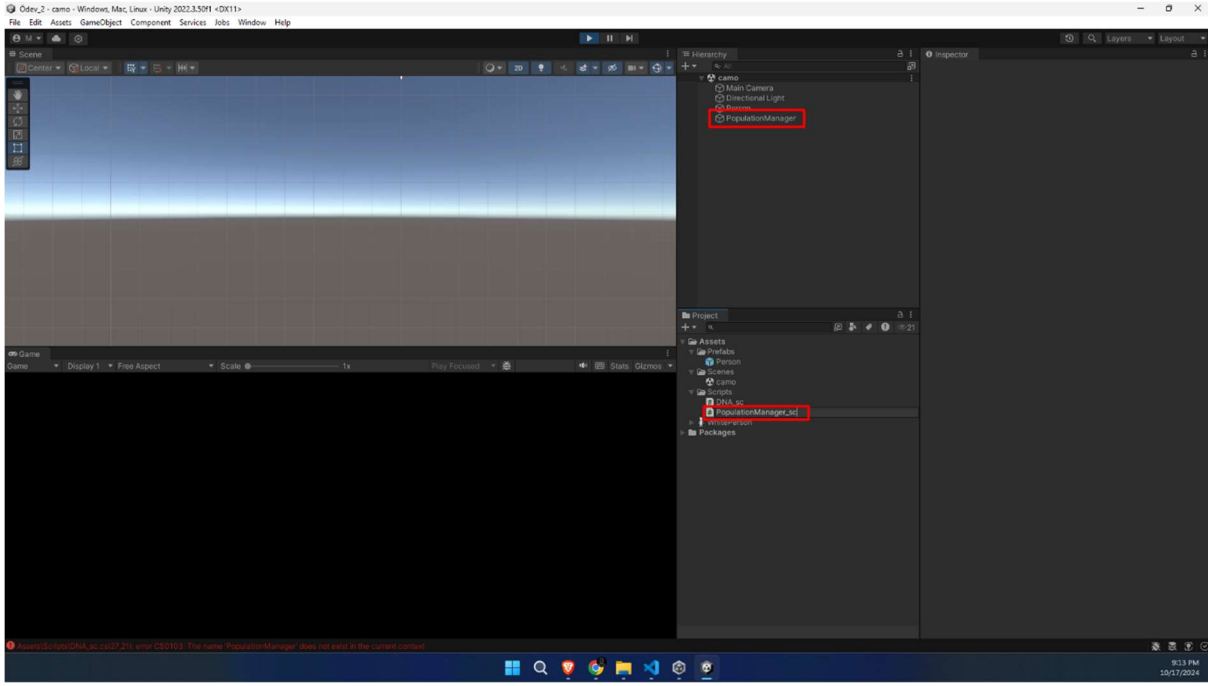


## 4 – ProductManager\_sc düzenlemeleri

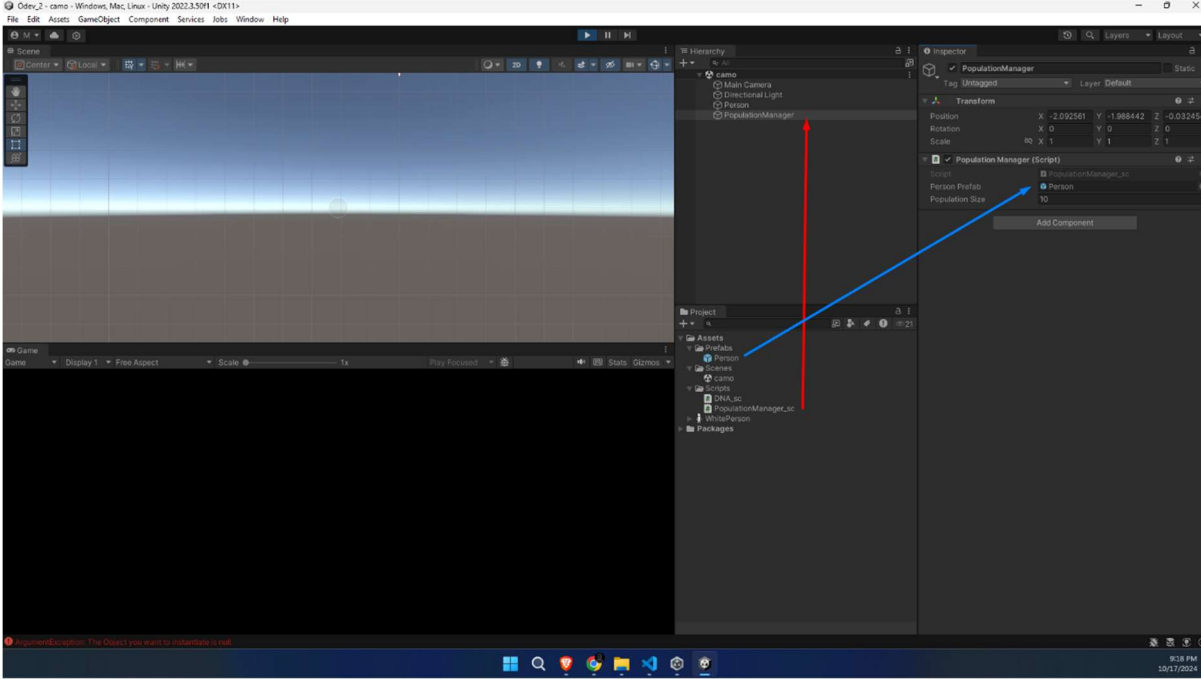
4.1 Popölasyon kontrolü için Hierarchy altında boş bir yerde sağ tıklayıp Creaty Empty sekmesine tıklıyoruz. Oluşan nesnenin adını PopulationManager yapıyoruz.



#### 4.2 – Project sekmesinde Scripts dosyasında boş bir script dosyası oluşturarak adını PopulationManager\_sc yapıyoruz.



4.3 – PopulationManager\_sc dosyasını PopulationManager nesnesi üzerine sürüklüyoruz. Daha sonra PopulationManager nesnesine tıklayarak açılan Inspector menüsünde Person Prefab kısmına Person nesnesini sürüklüyoruz.



4.4- PopulationManager\_sc dosyasını açıyoruz. Farklı nesilleri temsil etmek için döngüler (cycle) tanımlayacağız.

Person prefab'ine referans için bir değişken tanımlayalım. Unity ekranından değer ataması yapabilmek için public tanımlıyoruz:

- public GameObject personPrefab;

Populasyon büyüklüğü için:

- public int populationSize = 10;

Oluşturulan tüm popülasyon için array tanımlıyoruz:

- List<GameObject> population = new List<GameObject>();

Her yeni döngü içinde zamanın ne kadar ilerlediğini saklamak için:

- public static float elapsed = 0;

Döngü oluşturmak için Start() içerisine döngümüzü yazıyoruz.

```
{  
    for (int i = 0; i < populationSize; i++)  
    {  
        Vector3 pos = new Vector3(Random.Range(-9.5f, 9.5f), Random.Range(-3.4f, 5.4f), 0);  
        GameObject o = Instantiate(personPrefab, pos, Quaternion.identity);  
        o.GetComponent<DNA_sc>().r = Random.Range(0.0f, 1.0f);  
        o.GetComponent<DNA_sc>().g = Random.Range(0.0f, 1.0f);  
        o.GetComponent<DNA_sc>().b = Random.Range(0.0f, 1.0f);  
    }  
}
```

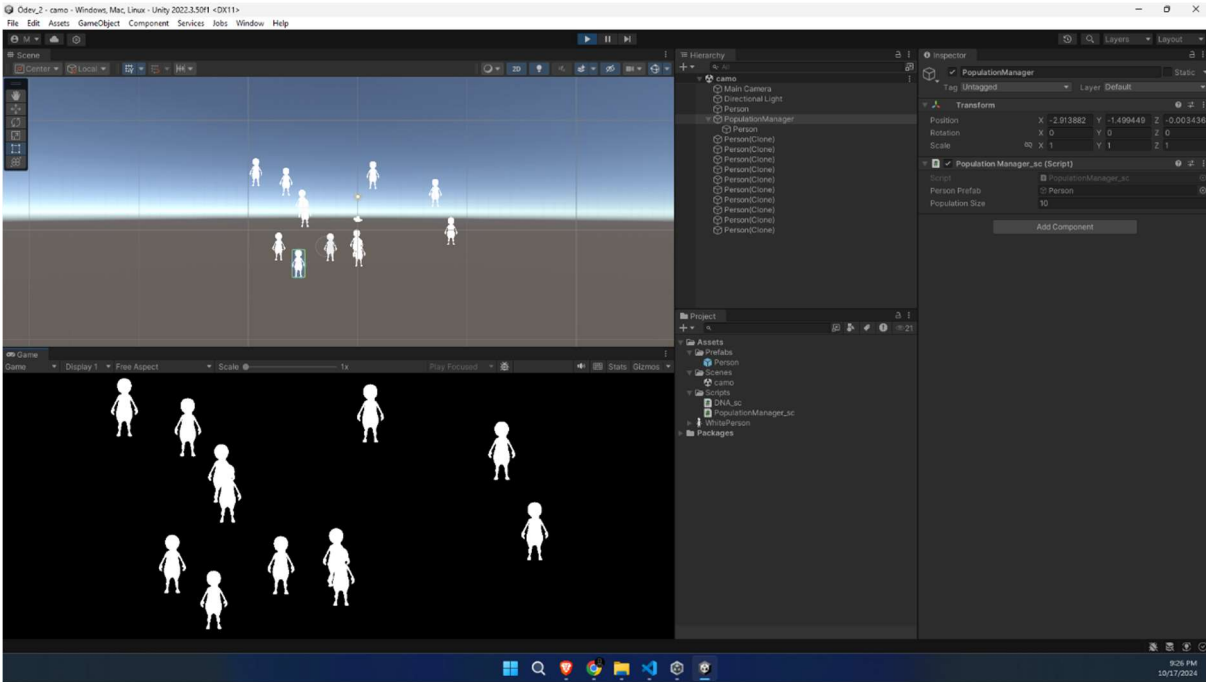


```
population.Add(o);
```

```
}
```

```
}
```

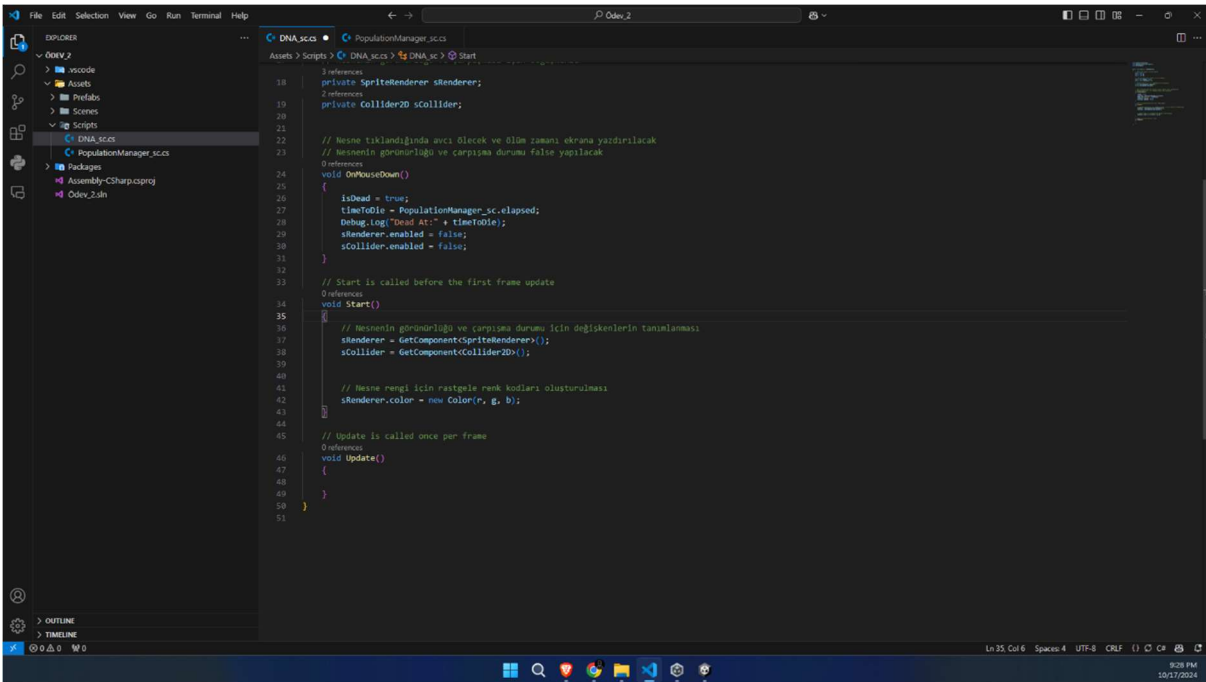
Unity ekranından oynat tuşuna basarak rastgele 10 nesne oluşturduğunu gözlemliyoruz.



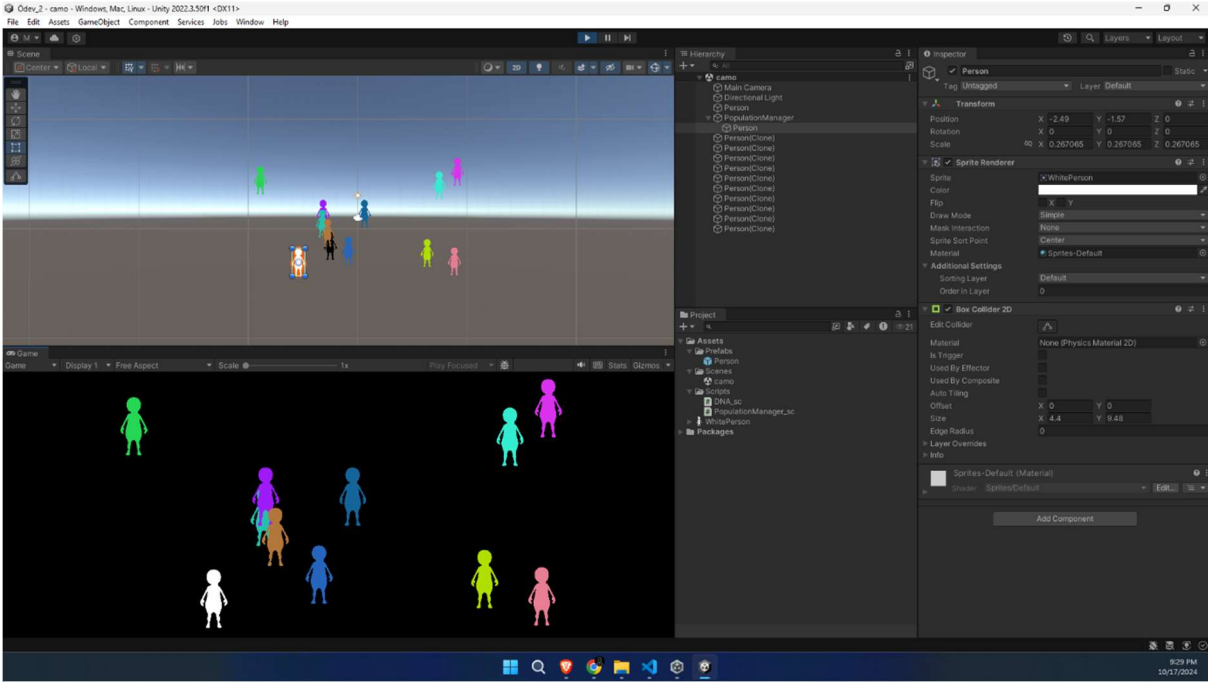
4.5 – Renklendirme için DNA\_sc dosyasını açarak Start() içerisine

```
sRenderer.color = new Color(r, g, b);
```

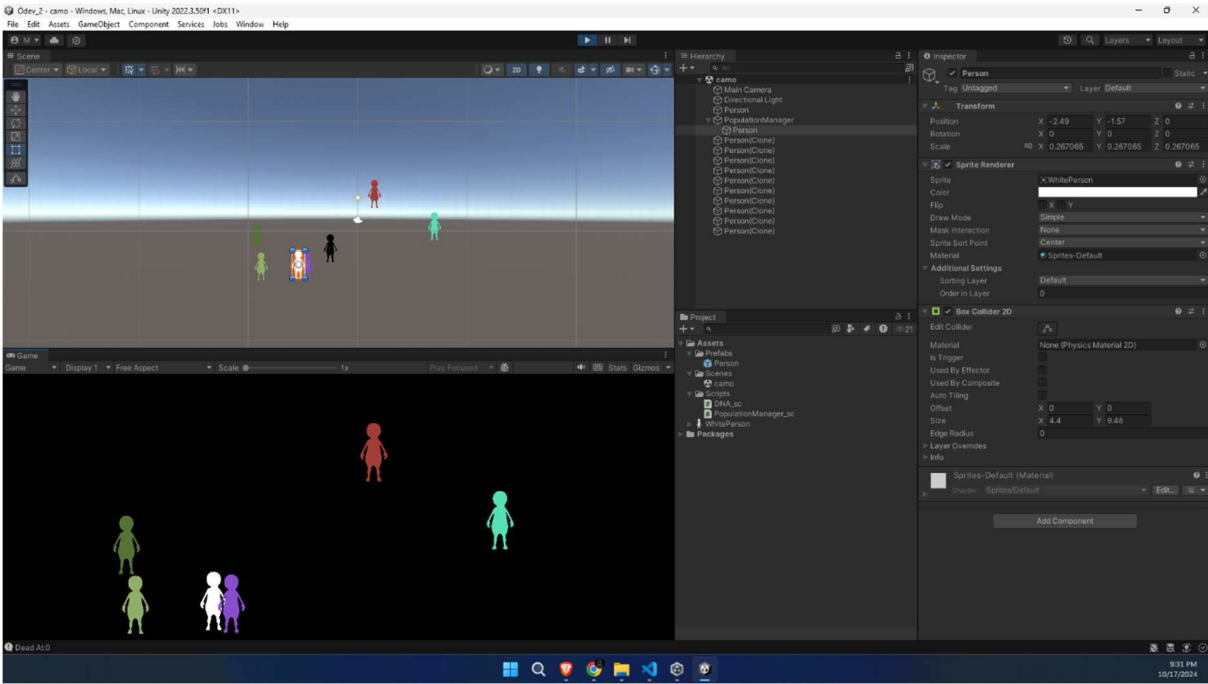
kodunu tanımlıyoruz.



4.5 – Renk kontrolü için Unity ekranında play tuşuna basarak karakterlerin renklendiğini gözlemliyoruz.



4.6 – Karakter üzerine tıklayarak karakterin yok olma durumunu test ediyoruz.



4.7 – Döngü süresini tanımlamak için:

- int trialTime = 10;

Hangi jenerasyonda olduğumuzun bilgisini tutmak için:

- int generation = 1;

Ekranda jenerasyon no ve geçen süreyi göstermek için:

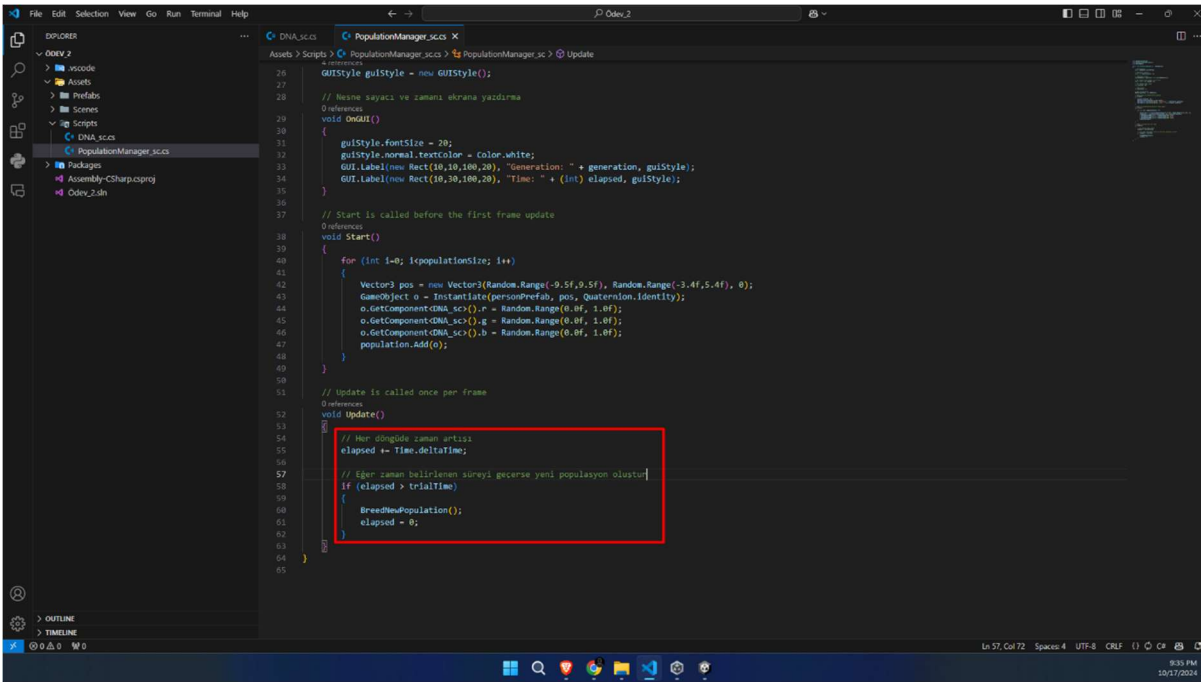
```
void OnGUI()
{
    guiStyle.fontSize = 20;
    guiStyle.normal.textColor = Color.white;
    GUI.Label(new Rect(10, 10, 100, 20), "Generation: " + generation, guiStyle);
    GUI.Label(new Rect(10, 30, 100, 20), "Time: " + (int)elapsed, guiStyle);
}
```

Zamanı ilerleterek jenerasyon kontrolünü sağlamak için Update() içine:

```
// Her döngüde zaman artışı
elapsed += Time.deltaTime;

// Eğer zaman belirlenen süreyi geçerse yeni populasyon oluştur
if (elapsed > trialTime)
{
    Debug.Log("New Population");
    BreedNewPopulation();
    elapsed = 0;
}
```

Kodlarını ekliyoruz. BreedNewPopulation fonksiyonunu birazdan tanımlayacağız.



#### 4.7 – Sıralama işlemleri için kullanacağım kütüphaneyi ekliyoruz:

- using System.Linq;

Yeni popülasyon eklemesi için:

```

void BreedNewPopulation()
{
    List<GameObject> newPopulation = new List<GameObject>();

    List<GameObject> sortedList = population.OrderBy(o =>
o.GetComponent<DNA_sc>().timeToDie).ToList();

    population.Clear();

    for (int i = (int)(sortedList.Count / 2.0f) - 1; i < sortedList.Count - 1; i++)
    {
        population.Add(Breed(sortedList[i], sortedList[i + 1]));
        population.Add(Breed(sortedList[i + 1], sortedList[i]));
    }

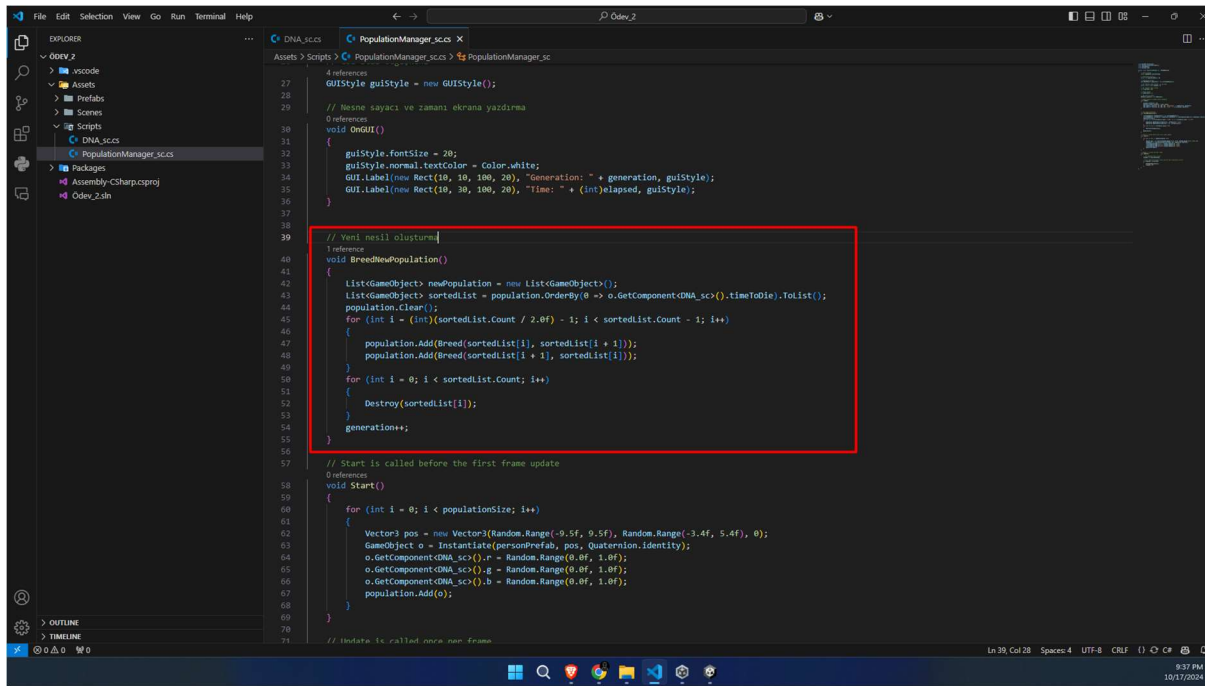
    for (int i = 0; i < sortedList.Count; i++)
    {
        Destroy(sortedList[i]);
    }

    generation++;
}

```

Kodlarını ekliyoruz. Bu kod:

- Popülasyondakileri (ilk ölen ilk sırada olacak şekilde) yaşam sürelerine göre sıralar
  - Popülasyonun daha uzun yaşayanlarını kapsayacak şekilde yarısını dikkate alarak (%75, vb. de olabilirdi) çiftleştirme yapar
  - Bu sayede daha uzun süre hayatta kalanlar kendi genlerini sonraki nesle aktarır.
  - Her yeni nesilde genlerini aktaranların özellikleri daha baskın hale gelir
- Breed fonksiyonunu birazdan tanımlayacağız.



#### 4.8 – Breed fonksiyonu:

GameObject Breed(GameObject parent1, GameObject parent2)

```

{
    Vector3 pos = new Vector3(Random.Range(-9.5f, 9.5f), Random.Range(-3.4f, 5.4f), 0);
    GameObject offspring = Instantiate(personPrefab, pos, Quaternion.identity);
    DNA_sc dna1 = parent1.GetComponent<DNA_sc>();
    DNA_sc dna2 = parent2.GetComponent<DNA_sc>();

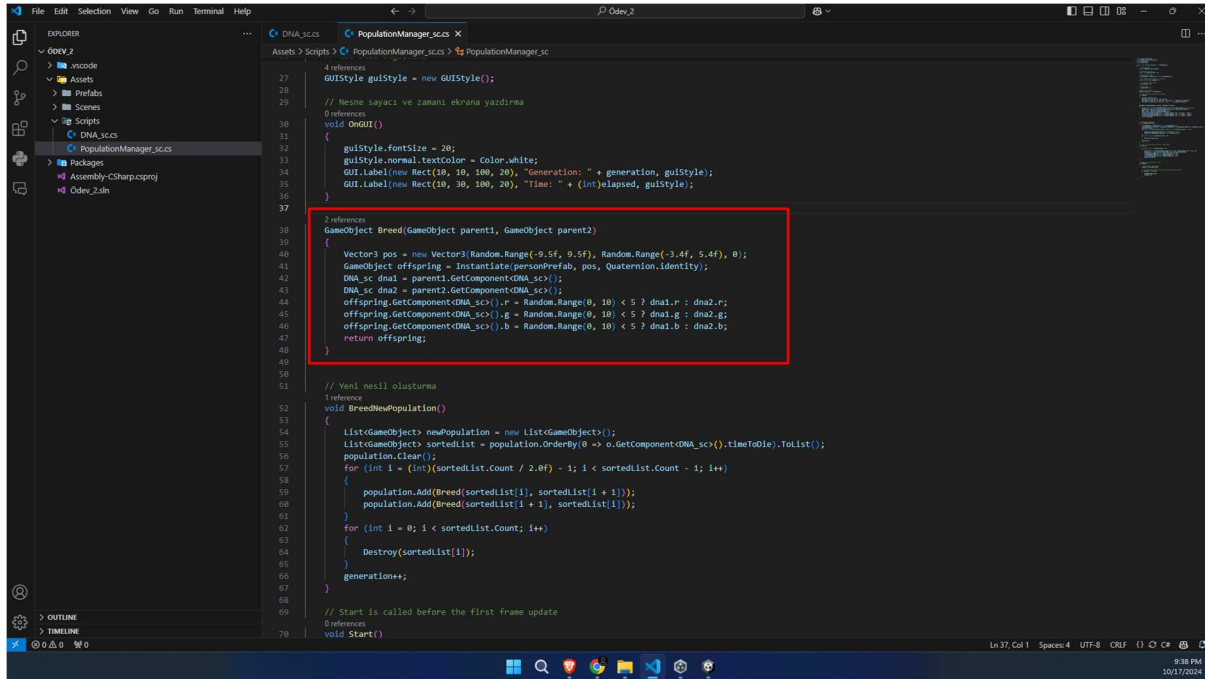
    offspring.GetComponent<DNA_sc>().r = Random.Range(0, 10) < 5 ? dna1.r : dna2.r;
    offspring.GetComponent<DNA_sc>().g = Random.Range(0, 10) < 5 ? dna1.g : dna2.g;
    offspring.GetComponent<DNA_sc>().b = Random.Range(0, 10) < 5 ? dna1.b : dna2.b;

    return offspring;
}

```

Bu kod sayesinde:

- Eğer parlak renkli olanları önce seçerseniz mat renkli olanlar daha uzun yaşar ve sonraki nesle kendi genlerini aktarırlar
- Bu şekilde tekrar eden birkaç nesil sonra popülasyonun rengi gittikçe matlaşır

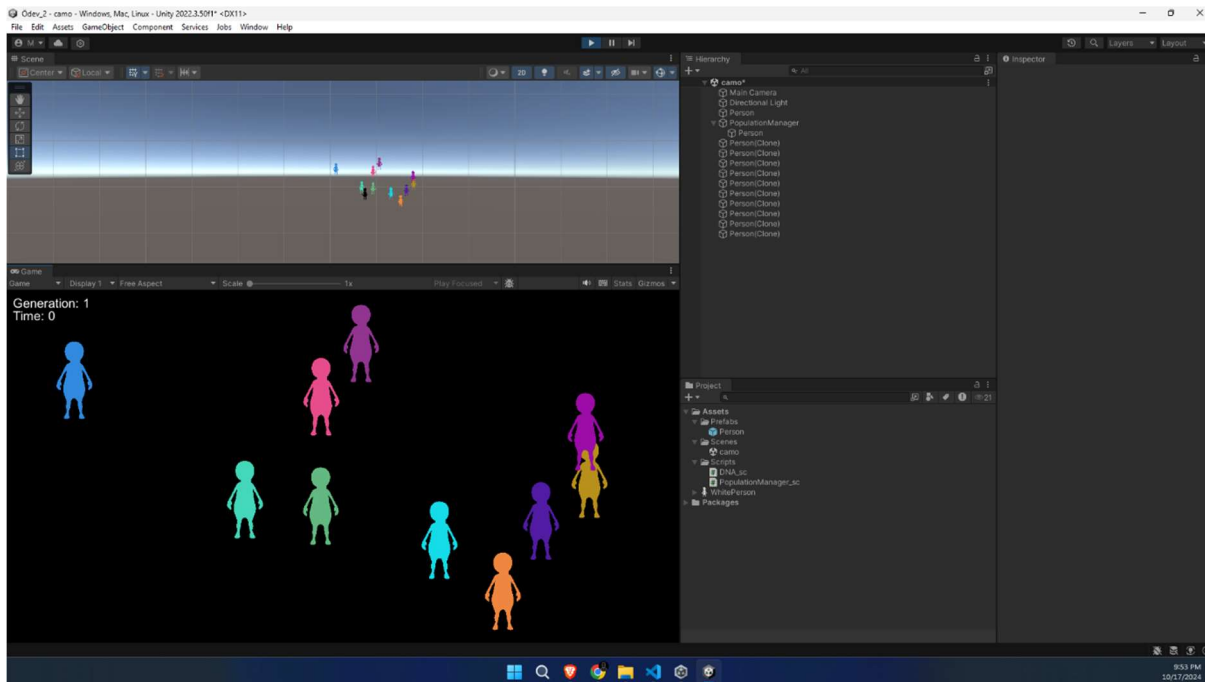


```
using UnityEngine;

public class PopulationManager : MonoBehaviour
{
    // DNA_scs
    // PopulationManager_scs
    // PopulationManager_sc

    27 GUIStyle guiStyle = new GUIStyle();
    28
    29 // Nesne sayacı ve zamanı ekrana yazdırma
    30 void OnGUI()
    31 {
    32     guiStyle.fontSize = 20;
    33     guiStyle.normal.textColor = Color.white;
    34     GUI.Label(new Rect(10, 10, 100, 20), "Generation: " + generation, guiStyle);
    35     GUI.Label(new Rect(10, 30, 100, 20), "Time: " + (int)elapsed, guiStyle);
    36 }
    37
    38 // 2 references
    39 GameObject Breed(GameObject parent1, GameObject parent2)
    40 {
    41     Vector3 pos = new Vector3(Random.Range(-9.5f, 9.5f), Random.Range(-3.4f, 5.4f), 0);
    42     GameObject offspring = Instantiate(personPrefab, pos, Quaternion.identity);
    43     DNA_sc dna1 = parent1.GetComponent<DNA_sc>();
    44     DNA_sc dna2 = parent2.GetComponent<DNA_sc>();
    45     offspring.GetComponent<DNA_sc>().r = Random.Range(0, 10) < 5 ? dna1.r : dna2.r;
    46     offspring.GetComponent<DNA_sc>().g = Random.Range(0, 10) < 5 ? dna1.g : dna2.g;
    47     offspring.GetComponent<DNA_sc>().b = Random.Range(0, 10) < 5 ? dna1.b : dna2.b;
    48     return offspring;
    49 }
    50
    51 // Yeni nesil oluşturma
    52 void BreedNewPopulation()
    53 {
    54     List<GameObject> newPopulation = new List<GameObject>();
    55     List<GameObject> sortedList = population.OrderBy(o => o.GetComponent<DNA_sc>().timeToDie).ToList();
    56     population.Clear();
    57     for (int i = (int)(sortedList.Count / 2.0f) - 1; i < sortedList.Count - 1; i++)
    58     {
    59         population.Add(Breed(sortedList[i], sortedList[i + 1]));
    60         population.Add(Breed(sortedList[i + 1], sortedList[i]));
    61     }
    62     for (int i = 0; i < sortedList.Count; i++)
    63     {
    64         Destroy(sortedList[i]);
    65     }
    66     generation++;
    67 }
    68
    69 // Start is called before the first frame update
    70 void Start()
    71 {
    72 }
```

#### 4.9 – Play tuşuna basarak test ediyoruz.





#### 4.10 – Mutasyon eklemek için kodu:

//%50 ihtimalle mutasyon

if (Random.Range(0, 10) < 5)

{

offspring.GetComponent<DNA\_sc>().r = Random.Range(0, 10) < 5 ? dna1.r : dna2.r;

offspring.GetComponent<DNA\_sc>().g = Random.Range(0, 10) < 5 ? dna1.g : dna2.g;

offspring.GetComponent<DNA\_sc>().b = Random.Range(0, 10) < 5 ? dna1.b : dna2.b;

offspring.GetComponent<DNA\_sc>().s = Random.Range(0, 10) < 5 ? dna1.s : dna2.s;

}

else

{

//%50 mutasyon ihtimali. Genelde mutasyon ihtimali çok daha düşüktür

offspring.GetComponent<DNA\_sc>().r = Random.Range(0.0f, 1.0f);

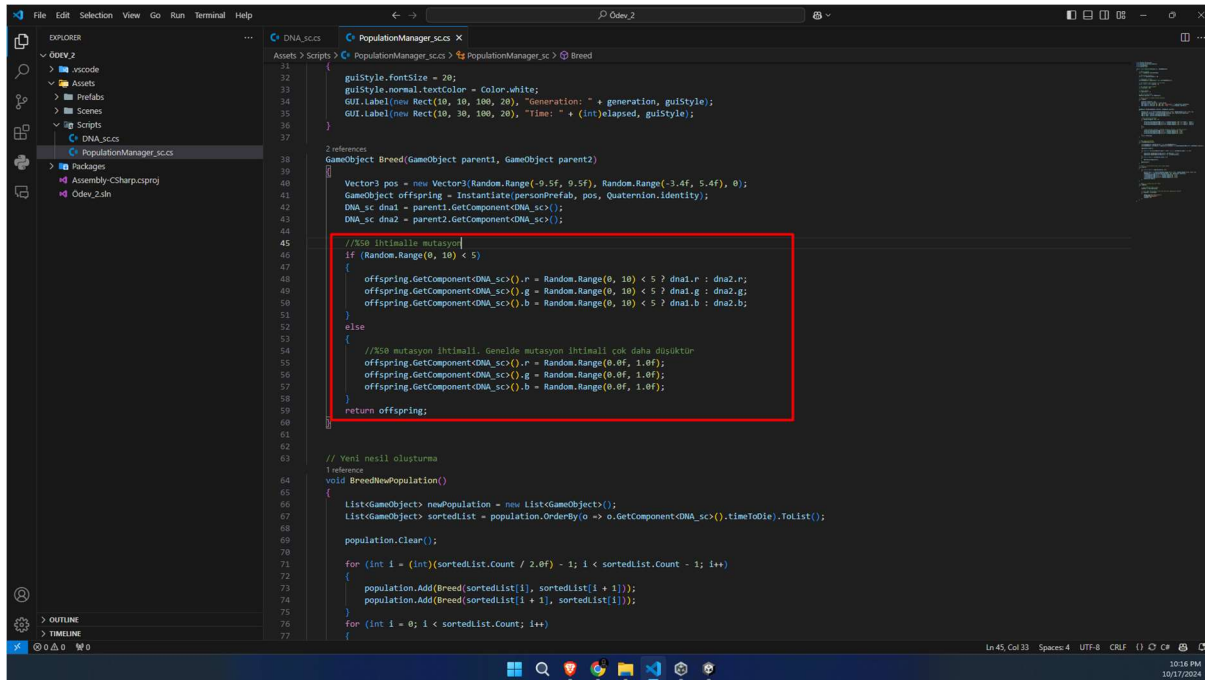
offspring.GetComponent<DNA\_sc>().g = Random.Range(0.0f, 1.0f);

offspring.GetComponent<DNA\_sc>().b = Random.Range(0.0f, 1.0f);

offspring.GetComponent<DNA\_sc>().s = Random.Range(0.1f, 0.3f);

}

Şeklinde değiştiriyoruz.





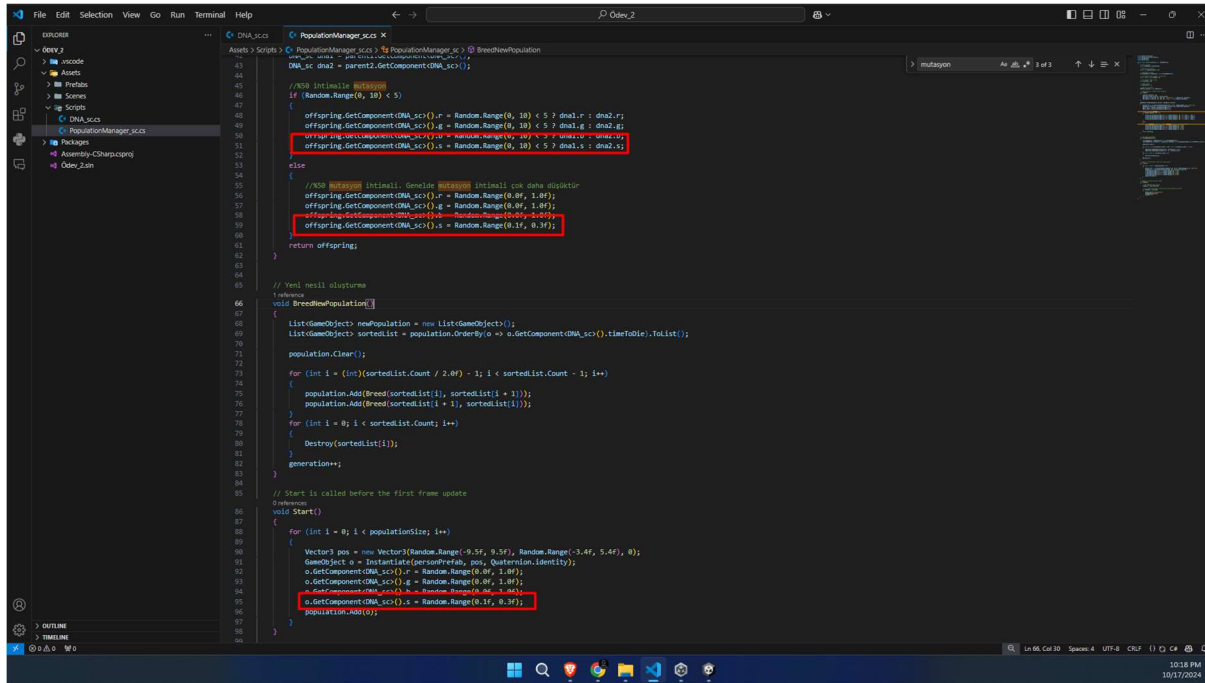
4.11 – Karaktere boyut eklemek için DNA\_sc dosyasına s adlı değişkeni ekliyoruz:

- public float s;

DNA\_sc dosyasında Start() içine aşağıdaki kodu ekleyerek initilaze ediyoruz:

- `this.transform.localScale = new Vector3(s, s, s);`

ProductManager\_sc dosyasının renk kodları altına s ile ilgili atamaları yapıyoruz:



#### 4.12 – Play tuşuna basarak değişiklikleri test ediyoruz.

