BURSA TEKNİK ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ (T) BÖLÜMÜ

BİLGİSAYAR OYUNLARDA YAPAY ZEKA

Ödev-2 Raporu

Github: https://github.com/MoussaBane/BOYZ-Genetik-Programlama-I

MOUSSA BANE

24435004029

Proje Tanıtımı:

Bu projede, **Unity 2D** platformunda **Genetik Algoritmalar** kullanarak bir kamuflaj eğitimi simülasyonu gerçekleştirilmiştir. Nesnelerin genetik özelliklerine (renk, büyüklük) dayanarak yaşam süreleri hesaplanmış, bu özelliklere göre yeni nesiller oluşturulmuştur. Her jenerasyonda daha uzun süre hayatta kalan nesneler, genlerini bir sonraki nesle aktarmıştır. Bu süreç birkaç jenerasyon boyunca tekrar ederek popülasyonun genetik yapısının evrimi gözlemlenmiştir.

Proje Adımları

1. Yeni Unity Projesi Oluşturma

Unity'de yeni bir 2D projesi oluşturun ve projenizi bir klasöre kaydedin.

2. Proje Dosyalarını İçe Aktarma

Verilen **CamoGATraining** dosyasını Unity projenize import edin. **Assets** > **Import Package** menüsünü kullanarak dosyaları içe aktarın.

- **Camo** sahnesini **Scenes** klasörüne taşıyın.
- Varsayılan sahneyi silin.

3. Prefab ve Script Dosyalarının Oluşturulması

Person Prefab'ı

- **Assets** klasörünün altında bir **Prefabs** klasörü oluşturun.
- **Person** adlı bir nesneyi sahneye ekleyip, **Prefabs** klasörüne sürükleyerek **Person** prefab'ını oluşturun.

Scripts

- **Assets** altında bir **Scripts** klasörü oluşturun.
- İçinde DNA_sc.cs ve PopulationManager_sc.cs adlı iki adet **C# script** dosyası oluşturun.

4. DNA_sc Script'inin Yazılması

Bu script, nesnelerin genetik özelliklerini ve ölüm durumunu yönetir. **Person** prefab'ına bu script'i ekleyin.

```
using UnityEngine;

public class DNA : MonoBehaviour
{

    public float r; // Kırmızı renk genetik kodu
    public float g; // Yeşil renk genetik kodu
    public float b; // Mavi renk genetik kodu
    public bool isDead = false; // Nesnenin ölüp ölmediğini takip eder
    public float timeToDie = 0; // Nesnenin ölüm zamanı
    SpriteRenderer sRenderer; // Görüntü bileşeni
    Collider2D sCollider; // Çarpışma bileşeni

    // Başlangıçta bileşenleri ilklendirir
    void Start()
    {
        sRenderer = GetComponent<SpriteRenderer>();
        sCollider = GetComponent<Collider2D>();
        sRenderer.color = new Color(r, g, b); // Nesnenin rengini atar
    }

    // Nesneye tıklandığında ölür
    void OnMouseDown()
    {
        isDead = true;
        timeToDie = PopulationManager_sc.elapsed; // Geçen süreyi alır
        sRenderer.enabled = false; // Görüntüyü kapatır
        sCollider.enabled = false; // Çarpışmayı kapatır
        Debug.Log("Öldü: " + timeToDie);
    }

}
```

5. PopulationManager Script'inin Yazılması

Bu script, nesilleri oluşturur ve her neslin genetik yapısına göre evrim sürecini yönetir.

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

public class PopulationManager_sc : MonoBehaviour
{
    public GameObject personPrefab; // Person nesnesi referansı
     public int populationSize = 10; // Popülasyon büyüklüğü
     public static float elapsed = 0; // Zaman takibi
     private List<GameObject> population = new List<GameObject>(); // Popülasyon listesi
     private int trialTime = 10; // Her döngünün süresi
     private int generation = 1; // Jenerasyon sayısı
     GUIStyle guiStyle = new GUIStyle(); // GUI yazı stili

     // Başlangıçta popülasyonu oluşturur
     void Start()
     {
         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>().r = Random.Range(0.0f, 1.0f);
             o.GetComponent<DNA>().g = Random.Range(0.0f, 1.0f);
             o.GetComponent<DNA>().b = Random.Range(0.0f, 1.0f);
             population.Add(o);
         }
     }
```

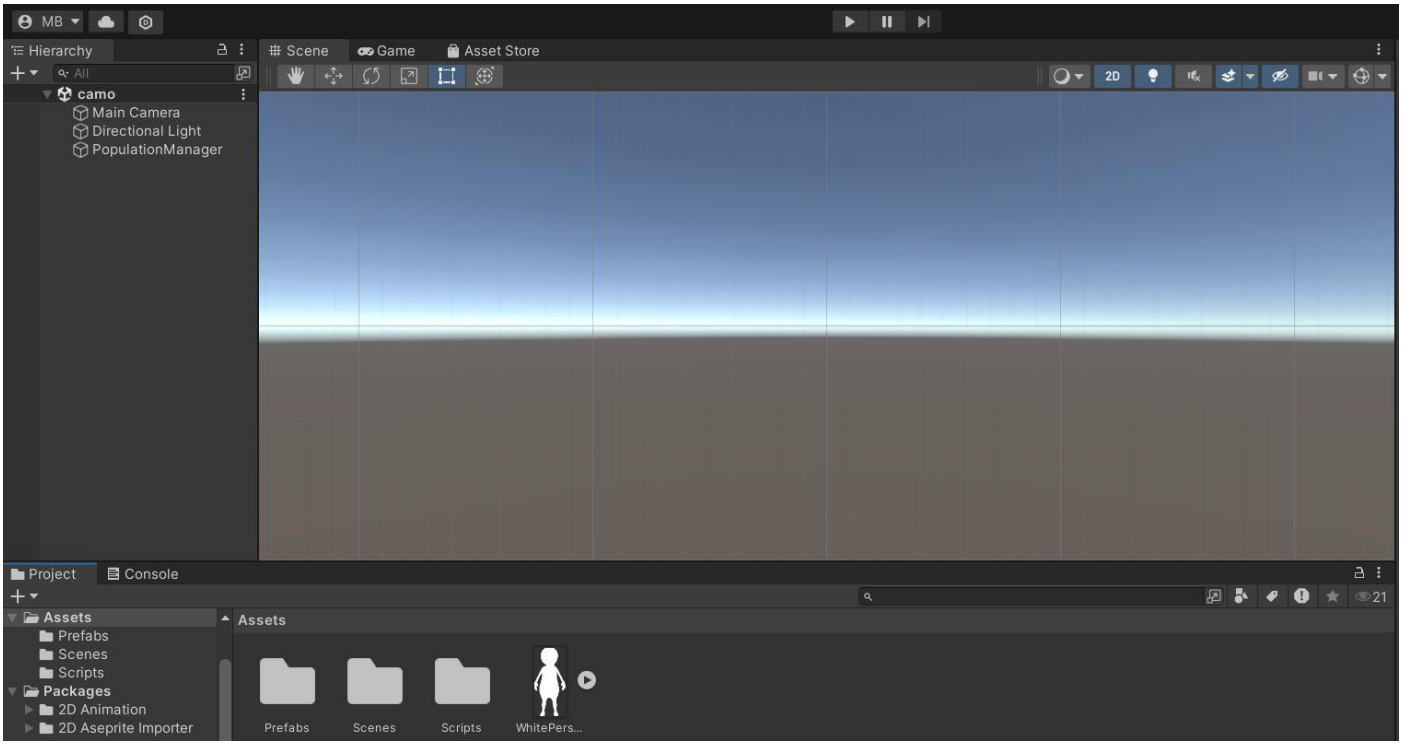
```

     // Her jenerasyonun zamanını günceller ve jenerasyonu değiştirir
     void Update()
     {
         elapsed += Time.deltaTime;
         if (elapsed > trialTime)
         {
             BreedNewPopulation(); // Yeni jenerasyon oluştur
             elapsed = 0;
         }
     }

     // Yeni jenerasyonu oluşturur
     void BreedNewPopulation()
     {
         List<GameObject> sortedList = population.OrderBy(o => o.GetComponent<DNA>().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]); // Eski nesneleri yok et
         }

         generation++;
     }
```

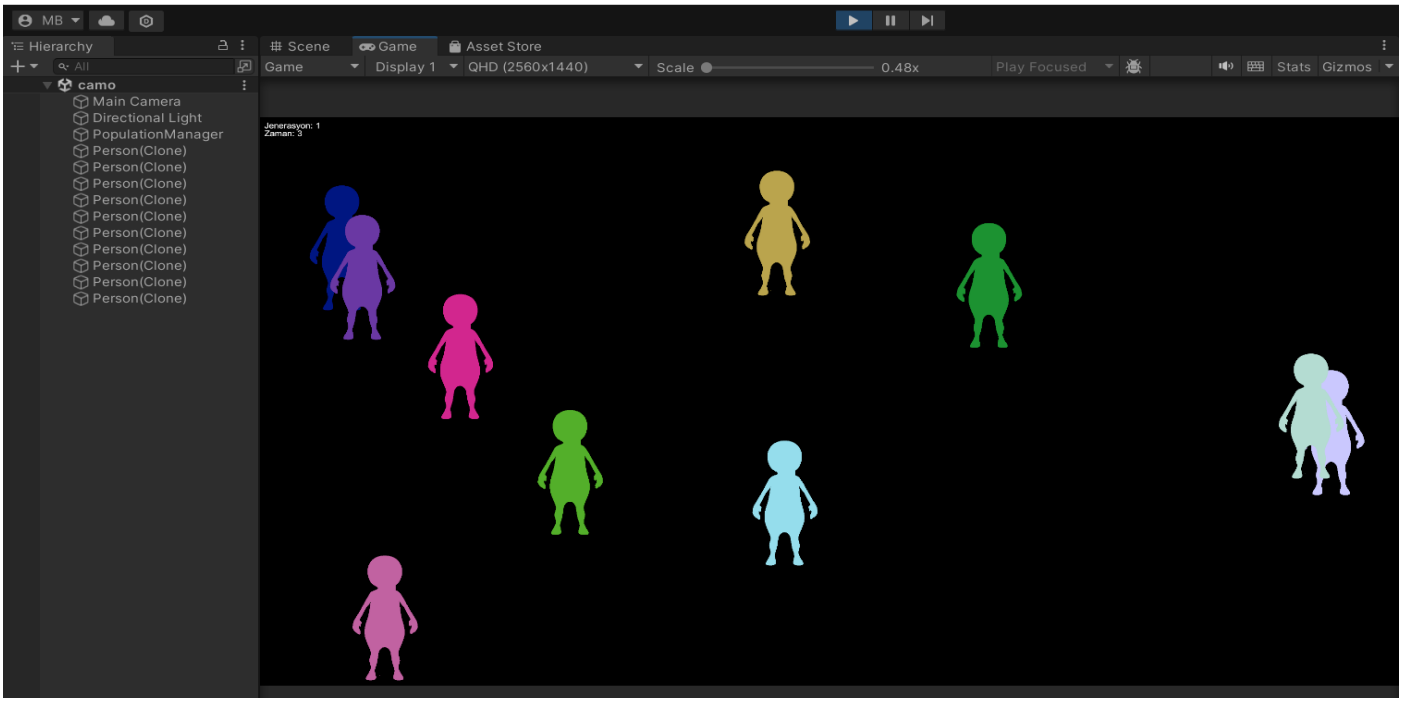
```
    // İki ebeveynden yeni bir nesil oluşturur
    GameObject Breed(GameObject parent1, GameObject parent2)
    {
        Vector3 pos = new Vector3(Random.Range(-9.5f, 9.5f), Random.Range(-3.4f, 5.4f), 0);
        GameObject offspring = Instantiate(personPrefab, pos, Quaternion.identity);
        DNA dna1 = parent1.GetComponent<DNA>();
        DNA dna2 = parent2.GetComponent<DNA>();

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

        return offspring;
    }

    // Ekranda jenerasyon ve süre bilgisi gösterir
    void OnGUI()
    {
        guiStyle.fontSize = 20;
        guiStyle.normal.textColor = Color.white;
        GUI.Label(new Rect(10, 10, 100, 20), "Jenerasyon: " + generation, guiStyle);
        GUI.Label(new Rect(10, 30, 100, 20), "Zaman: " + (int)elapsed, guiStyle);
    }

}

```

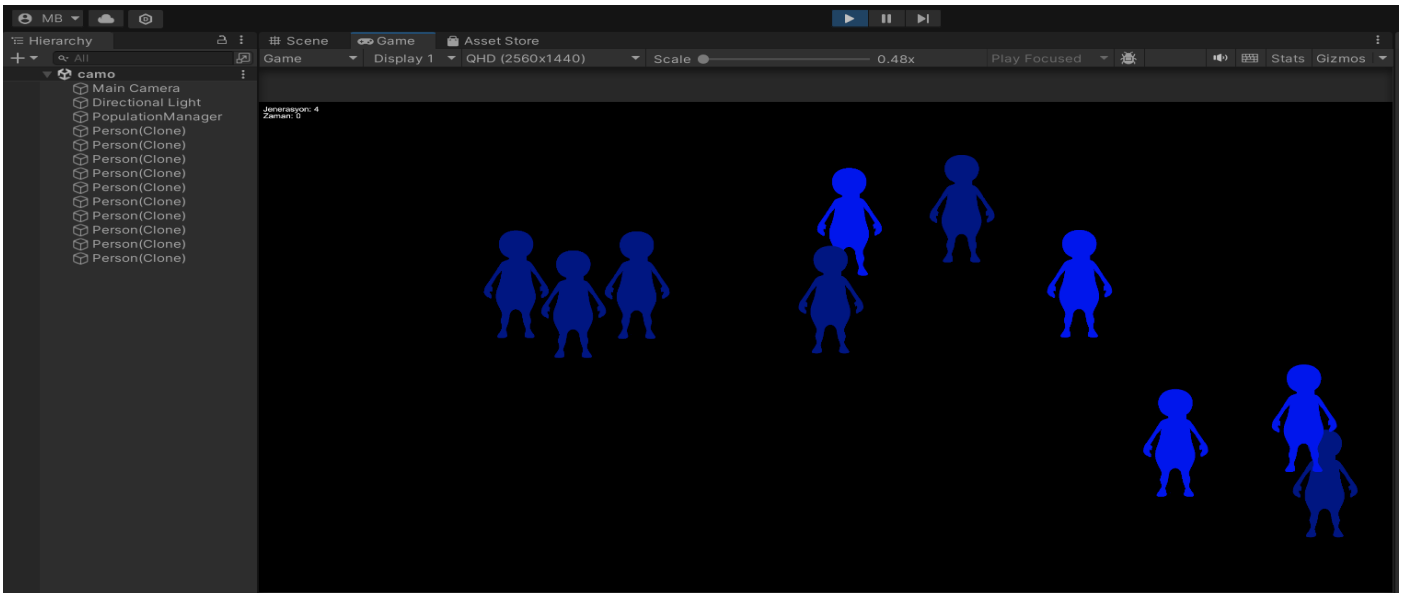
6. Sonuçların Gözlemlenmesi

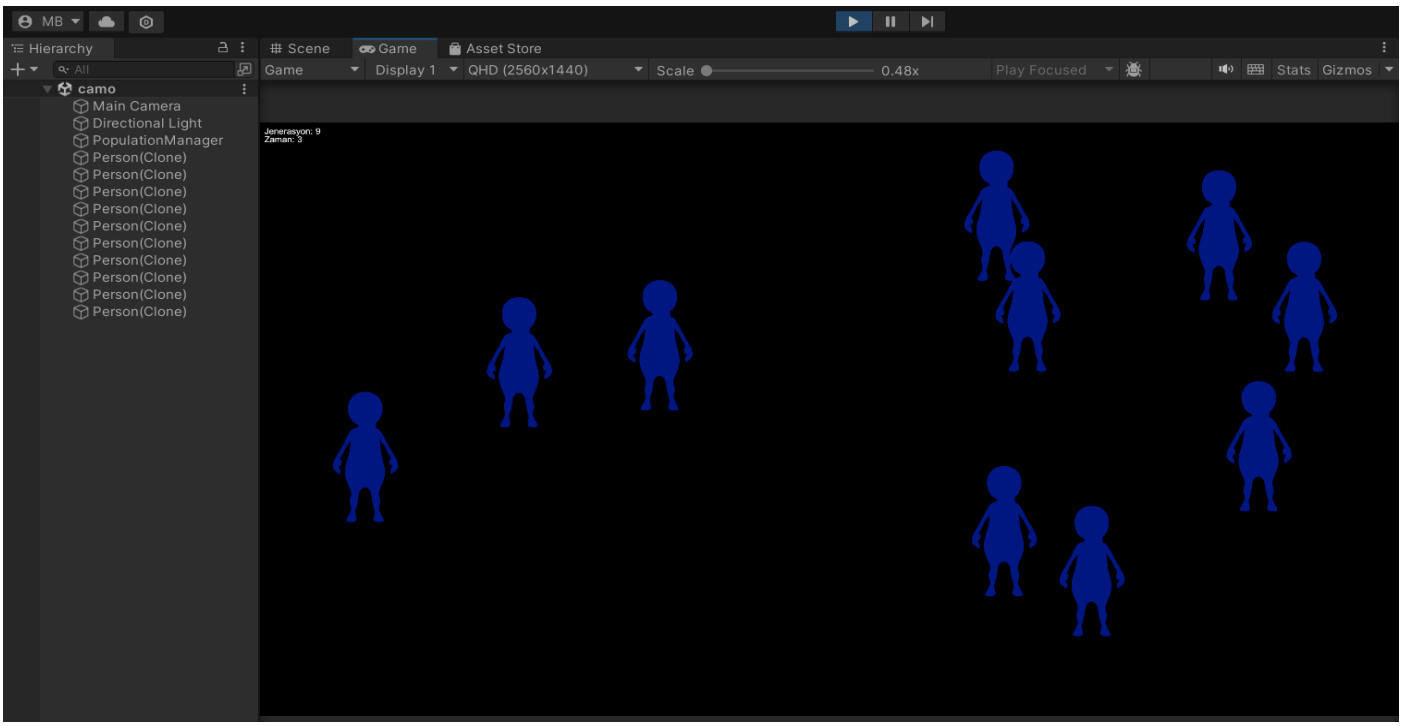
Oyun çalıştırıldığında ekrandaki nesnelerin farklı renklerde rastgele oluştuğunu ve fare ile tıkladığınızda kaybolduklarını gözlemleyin. Zamanla, hayatta kalan nesneler daha mat renkli hale gelir ve bu süreç jenerasyonlar boyunca devam eder.

Bu projede genetik algoritmaların nasıl çalıştığını ve popülasyonun zaman içinde nasıl evrimleştiğini gözlemledik. Unity kullanarak bu sürecin simülasyonunu gerçekleştirdik. Renk ve yaşam süresi gibi genetik özelliklere dayanan bir evrim mekanizması ile kamuflaj becerisi olan nesneler elde edilmiştir.




Hierarchy
Scene
Game
Asset Store
Game
Display 1
QHD (2560x1440)
Scale
0.48x
Play Focused
Stats
Gizmos
camo
Main Camera
Directional Light
PopulationManager
Person(Clone)
Jenerasyon: 1
Zaman: 3


Hierarchy
Scene
Game
Asset Store
Game
Display 1
QHD (2560x1440)
Scale
0.48x
Play Focused
Stats
Gizmos
camo
Main Camera
Directional Light
PopulationManager
Person(Clone)
Jenerasyon: 4
Zaman: 0


Hierarchy
Scene
Game
Asset Store
Game
Display 1
QHD (2560x1440)
Scale
0.48x
Play Focused
Stats
Gizmos
camo
Main Camera
Directional Light
PopulationManager
Person(Clone)
Jenerasyon: 9
Zaman: 3