GENETİK KODLAMA

HAFTA 2 PROJESİ

23435004013-MELIKA JIBRIL SEID

20/10/2024 | **BURSA TEKNIK UNIVERSITESI**

İÇİNDEKİLER

PROJENİN TANITIMI

Bu proje genetik algoritmayı uygulayan bir sinek öldürme oyunudur. Genetik algoritma, biyolojik evrimi yönlendiren süreç olan doğal seçilime dayalı hem kısıtlı hem de kısıtsız optimizasyon problemlerini çözmek için yarayan bir yöntemdir. Genetik algoritma, bireysel çözümlerden oluşan bir popülasyonu tekrar tekrar değiştirir. Genetik algoritma her adımda mevcut popülasyondan bireyleri ebeveyn olarak seçer ve bunları bir sonraki nesil için çocukları üretmek için kullanır. Ardışık nesiller boyunca, popülasyon optimum bir çözüme doğru "evrimleşir". Bu oyun bu mantığı kullanarak C# dilinde kodlanmıştır ve Unity platformunda geliştirilmiştir. Gerekli materyaller ve kodlar github üzerinden paylaşılmıştır

Github linki: https://github.com/m3likaj/GameProgramming/tree/main/Hafta2

OYUNUN MANTIĞI

Bu oyun “en güçlü olanın hayatta kalması” prensibini takip etmektedir. Oyun çalıştırıldığında ekranda farklı boyutları ve farklı renkleri olan “sinekler” ekrana çıkmaktadır. Sineklerin yaşam süresi 15 saniye olup bu süre içerisinde oyuncu sinekler üzerinde tıklayınca ölüp ekrandan kaybolurlar. Yaşam süresi bitince ekran yenilenir ve yeni nesil sinekler ekranda yerleşir. Kullanıcının öldürme sırasına bağlı olarak en uzun süre hayata kalabilen sinekler genlerini bir sonraki nesle aktarabilir. Fakat bu süreçte mutasyon olabilmektedir ve birkaç tanesi sıra dışı bir renk ve boyuta sahip olabilmektedir. Nesiller gittikçe tekdüze bir genetiğe sahip olmaktadır fakat bu da kullanıcının yapacağı seçimler sonucu tekrar değişebilmektedir. Bu değişimi resim 1 ve resim 2’de görebilirsiniz. İlk resimde oyunun başlangıç hali olan ilk nesil (gerenation) gösterilirken en büyük ve en parlak renkli olan sineklerin önce öldürülmesi sonunda sineklerin 15. nesilde neredeyse tüm sineklerin küçük ve soluk renkli oldukları gözlemlenebilir.

Resim 1: Oyunun başlangıcındaki durum

Resim 2: 15. nesilin görüntüsü

İLK HAZIRLIKLAR

Projeyi başlamak için bu adımları takip etmek gerekmektedir.

1. Unity Hubı açıp 2D proje oluşturunuz
2. Yukarıda verilen githubdan “CamoGATraining.unitypackage” dosyasını indirin ve çift tıklayarak projenize ekleyiniz
3. Projede olan “Sample scene”i silip “camo” adlı sahneyi sürükle bırak ile hierarchy kısmına getiriniz. Bu adımda “Game mode”a tıklayınca siyah bir arka plan olması gerekmektedir.
4. Heirarchyde sağ tıklayıp “create empty”I tıklayarak boş bir obje oluşturunuz ve ona “population manager” adını veriniz
5. Assets kısmında sağ tıklayıp “create”, sonra ise “folder”ı basarak script adlı bir dosya oluşturunuz.
6. Script dosyanın içinde sağ tıklayıp “create”, sonra ise “C# script”i basarak “PopulationManager” ve ”DNA” adlı iki tane script oluşturunuz.
7. PopulationManager scriptini hierarchyde aynı isimli obje üzerinde sürükle bırak ile atayınız.
8. DNA isimli dosyayı da Assets içindeki “person” isimli objeye sürükle bırak ile atınız. Her iki atamanın başarılı olduğunu kontrol etmek için ilgili nesnenin üstünde tıklayıp inspector kısmında gerekli scriptin olup olmasını bakmanız gerekir

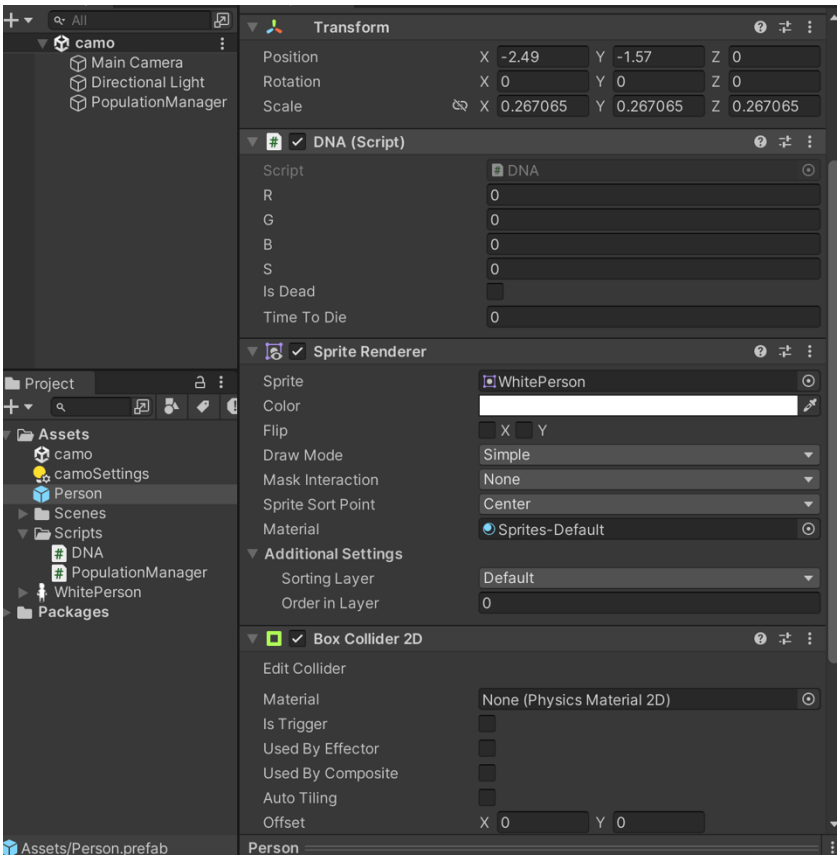
DNA.CS

Bu script adından anlaşılacağı üzere sineklerimizin DNAsını veya özelliklerin bulunduğu bir scripttir. İçinde farklı özellikleri tutan değişkenler, değişkenlerin ilk ataması yapıldığı start fonksiyonu ve üzerinde tıklanınca ekrandan kaybolmasını sağlayan OnMouseDown fonksiyonu bulunmaktadır.

DEĞİŞKENLER

Bu kısımda sineğin renklerini temsil eden ve 0 ile 1 arası değer alabilen kırmızı, yeşil ve mavi değerleri için r,g,b değişkenleri, sineğin boyutunu tutan s değişkeni, sineğin ekranda görünmesini sağlayan rengini ayarlayan SpriteRenderer öge tipinden sRenderer ve sineğin başka sinek ile çarpmasını engelleyen Collider2D tipinden sCollider değiştenleri bulunmaktadır. Bu öğeler unityde person üzerinde tıklanınca inspector kısmında görünmektedir (resim 3). Onun dışında ise sineğin hayatta olup olmamasını gösteren isDead ve ölüm zamanını tutan timeToDie değişkenleri tanımlanır.

```
public float r;
public float g;
public float b;
public float s;
public bool isDead = false;
public float timeToDie = 0;
SpriteRenderer sRenderer;
Collider2D sCollider;
```



Resim 3 değişkenlerde çağırılan özellikler

START FONKSİYONU

Burada sRenderer ve sCollider öğesi oluşturulduktan sonra sRendererin renk (color) özelliğine r,g,b değişkenleri atanır. Ayrıca bu nesnenin transform componentinin localScale özelliğini kullanarak nesnenin boyutunu belirleyebilirsiz. Bunun için yeni bir vector oluşturup içinde s değerini vermeliyiz. Bu çağırılan özellikleri unityde resim 3'te görebilirsiniz

```
void Start()
    {
         sRenderer = GetComponent<SpriteRenderer>();
        sCollider = GetComponent<Collider2D>();

        sRenderer.color = new Color(r,g,b);
        this.transform.localScale = new Vector3(s,s,s);
    }
```

ON MOUSE DOWN FONKSİYONU

Bu fonksiyon sineğin üzerine tıklanınca olanları belirler. İlk olarak sineğin öldüğünü gösteren isDead değişkenini true yapar, sonra ölüm zamanını kayıt altına alır ve unitynin log kısmında gösterir. Son olarak Sprite renderer öğesini kapatır. Son satırı unityde person oğesini heirarchye getirdiktan sonra resim 3te gösterilen sprite renderer öğesinin yanındaki kutuyu tıklayıp tiki kaldırarak manual olarak nasıl yapıldığını görebilirsiniz.

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

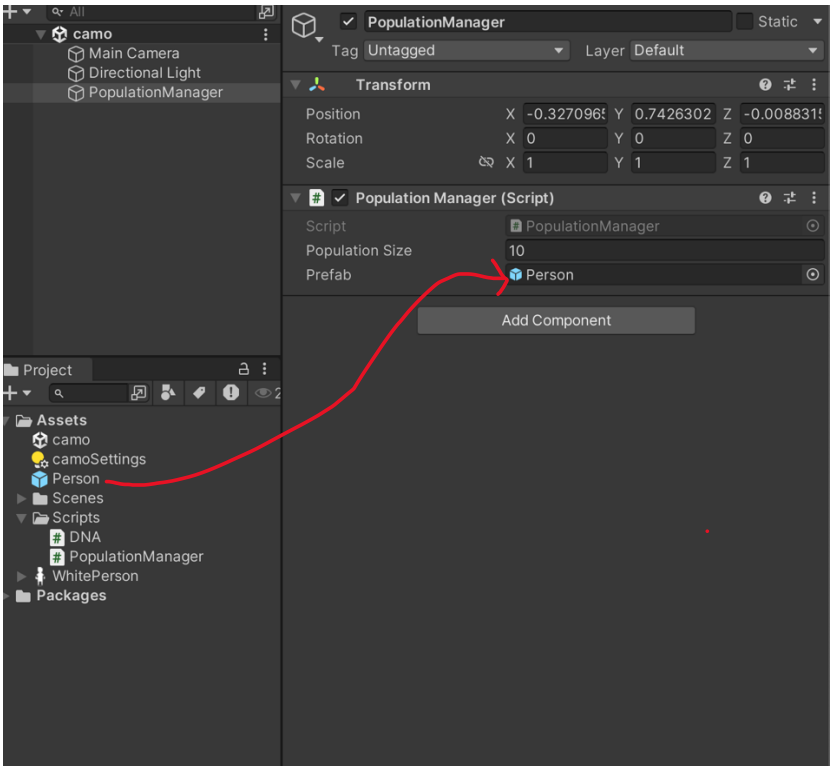
POPULATIONMANAGER.CS

Bu script adından anlaşılaağı üzere sinek nüfüsünü yönetmektedir. İçinde değişken tanımlamalar, start ve update fonksiyonları, ekrana yazı eklemek için kullanılan onGUI fonksiyonu ve nesilin oluşturulmasını sağlayan breed ve breedNewPopulation fonksiyoları bulunur.

DEĞİŞKENLER

Burada geçen zamanı tutacak olan elapsed değişkeni her nesne için aynı olması gerektiğinden static olarak tanımlanmıştır, GUI arayüzü temsil etmekte olup GUIStyle de arayüzün stilini temsil eder. Population size veya nesilde olan nüfüs sayışına 10, generasyonun ilk değeri olarak da 1 verilmiştir. Ondan sonraki değişken nesilin maximum yaşayabileceği zamanı belirler.Burada 15 verilmiş ancak değiştirilebilir. Her 15 saniyede nesil değişiyor. Ondan sonra nesildeki sinekleri tutan population adlı liste ve tek sineği temsil eden prefab nesneleri vardır. Prefab nesnesine sineği atmak için unityde populationManager nesnesine tıklayarak orada sriptin içinde prefab değişkenin yanındaki kutuya Assets dosyasında bulunan person nesnesinin sürükle bırak ile atılması gerekmektedir (resim 4)

```
    Public static float elapsed;
    GUIStyle guiStyle =  new GUIStyle();
    public int populationSize = 10;
    int generation = 1;
    int trialTime = 15;
    List<GameObject> population;
    public GameObject prefab;
```



Resim 4: prefab nesneyi tanımlamak

START FONKSİYONU

Burada ise ilk olarak yeni bir population listesi oluşturuluyor ve sonraki satırlarda bu listeye eklenmek üzere yeni sinek nesnesi oluşturup rastgele DNA değerleri atanıyor. Instantiate bir objeyi klonlamak veya kopyasını çıkarmak için kullanılan bir fonksiyondur. Burada önce tanımladığımız prefab nesnesinin klonu oluşturuluyor. Pos değişkeni ise sineği ekrandaki konumunu beliriliyor. Bu konum da her sinek için farklı olacağından rastgele atanıyor. Bu döngü ilk olarak belirlediğimiz nüfüs sayısına kadar (10 kez) tekrarlanıyor. Oluşturulan her nesne ise en sonda bulunan population.Add(o) fonksiyon ile population listemize ekleniyor.

```
void Start()
    {
       population = new List<GameObject>();
       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(prefab, 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);
        o.GetComponent<DNA>().s = Random.Range(0.1f,0.3f);
        population.Add(o);
       }
    }
```

UPDATE FONKSİYONU

Burada sürekli olarak geçilen süre elapsed değişkeninde tutuluyor ve neslin ölüm zamanı (trial time)a ulaşıldığında o süre sıfırlanıyor.

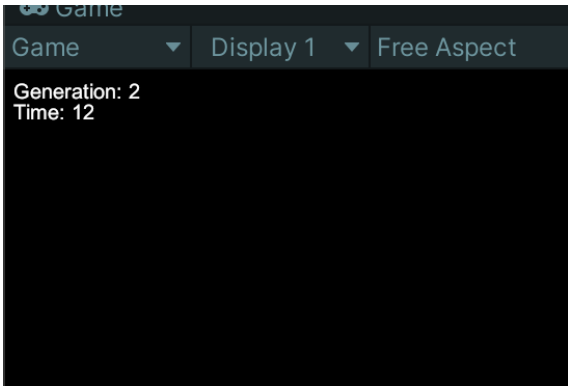
```
void Update()
    {
        elapsed += Time.deltaTime;
        if(elapsed> trialTime){
            BreedNewPopulation();
            elapsed=0;
        }
    }
```

ONGUI FONKSİYONU

Bu oyun başlatıldığında sol üst kenarda generasyon numarasını ve geçen süreyi göstermemize sağlayan kod parçasıdır (resim 4). Değişkenler kısmında tanımladığımız yeni arayüz stilini burada özelleştiriyoruz. Font size ile yazı büyüklüğü, text color ile yazı rengini ayarlayabiliriz. Label ile yazıları arayüzüne ekleyebiliriz. Label fonksiyonunda ilk olarak yazılacağı yazının konumunu dikdörtgen şeklinde belirleriz, sonrasında yazıları ekleriz, en sonda ise hangi arayüz stiline ekleyeceğimizi yazarız.

```
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);

    }
```



Resim 4: GUI Arayüzü

BREED FONKSİYONU

Bu fonksiyon nesiller arasındaki DNA aktarımını sağlayan fonksiyondur. Parametre olarak 2 ebeveyn rolünü alacak nesne almaktadır ve offspringi veya çocuğu geri döndürmektedir. İlk önce pos ile çocuğun konumunu rastgele olarak atar, daha sonra instantiate fonksiyonu ile offspringi oluştutur. Bizim offspring de bir sinek olacağından prefab nesnesini kopyalıyor. Daha sonra ebeveyenlerinin DNAsını tutmak uzere dna1 ve dna2 adlı DNA tipinde değişkenler tanımlayıp onlara ilgili ebeveynin DNA öğesini atanır. Bir sonraki satırlarda ise offspringin renk ve konum özelliklerini tanımlarız. Eğer 0-10 arasında oluşturduğumuz rastgele sayı (0 dahil, 10 hariç olmak üzere) 8den küçük ise (%80 ihtimal) offspring özelliklerini anne veya babasından alır. Onu da hangi ebeveynden alacağını belirlemek için rastgele sayıları kullanırız. Burada anne veya babadan alma şansı %50dir ancak oranlar isteğe bağlı olarak değiştirilebilir. Anne veya babadan almaması durumunda ise (%20 ihtimal) start fonksiyonunda yapıldığı gibi rastgele değer atanır. Böylece 2 ebeveynin bilgilerinden bir çocuk üretilir.

```
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(prefab, pos, Quaternion.identity);
        DNA dna1 = parent1.GetComponent<DNA>();
        DNA dna2 = parent2.GetComponent<DNA>();
        if(Random.Range(0,10)< 8)
        {
            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;
            offspring.GetComponent<DNA>().s= Random.Range(0,10)<5 ? dna1.s : dna2.s;
        }
        else
        {
            offspring.GetComponent<DNA>().r= Random.Range(0.0f,1.0f);
            offspring.GetComponent<DNA>().g= Random.Range(0.0f,1.0f);
            offspring.GetComponent<DNA>().b= Random.Range(0.0f,1.0f);
            offspring.GetComponent<DNA>().s = Random.Range(0.1f,0.3f);
         }
        return offspring;
    }
}
```

BREED NEW POPULATION FONSKSİYONU

Bu fonksiyon hangi sineklerin genleri bir sonraki nesle aktarılacağını belirleyen ve sonraki nesli oluşturulan nesli yöneten bir fonksiyondur. İlk önce population değişkeninde bulunan nesli sortedList adlı listeye kopyalar. Ardından ölüm zaman sırasına göre en kısa yaşayandan en üzün süre yaşayana doğru sıralıyor. Sonra var olan population listeyi boşaltıyor. Yeni nesli oluşturmak için nesilin daha uzun yaşayanlarından (sıralı listenin ikinci yarısından) ardışık iki sinek seçerek Breed fonksiyonu ile yeni bir sinek üretir. Daha sonra üretilen sinekler az önce boşaltılan population listesine tek tek eklenir. Ancak ikinci yarıyı kullanmak sadece 5 sinek üreteceğinden nesil sayısını muhafaza etmek için ebeveynlerin yerini değiştirerek aynı işlemi iki kez

yapar. Bunu yaptıktan sonra sıralı listeye artık ihtiyaç kalmadığı için yok edilir ve bir sonraki nesle geçildiğini göstermek için generation sayısı bir artırılır.

```
   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++){ //start
from the middle because the first half have short lives
            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++;
    }
```