GENETİK KODLAMA “PLATFORMDA KALMA”

HAFTA4 PROJESİ

23435004013-MELIKA JIBRIL SEID

3/11/2024 | **BURSA TEKNIK UNIVERSITESI**

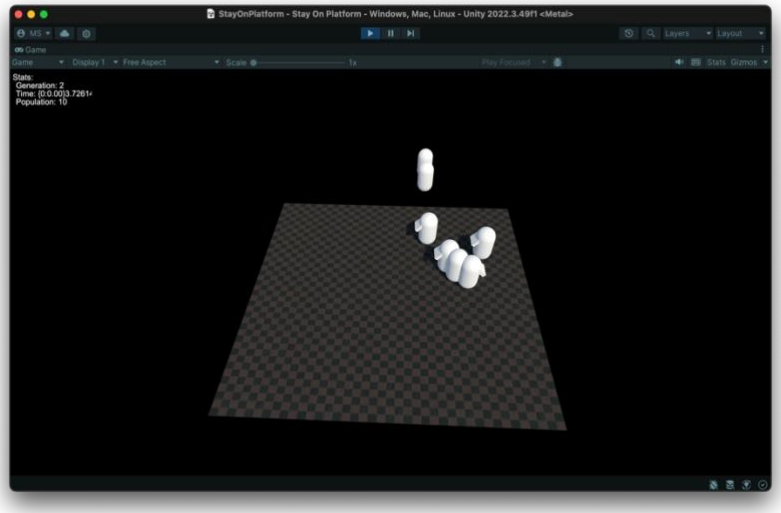
İÇİNDEKİLER

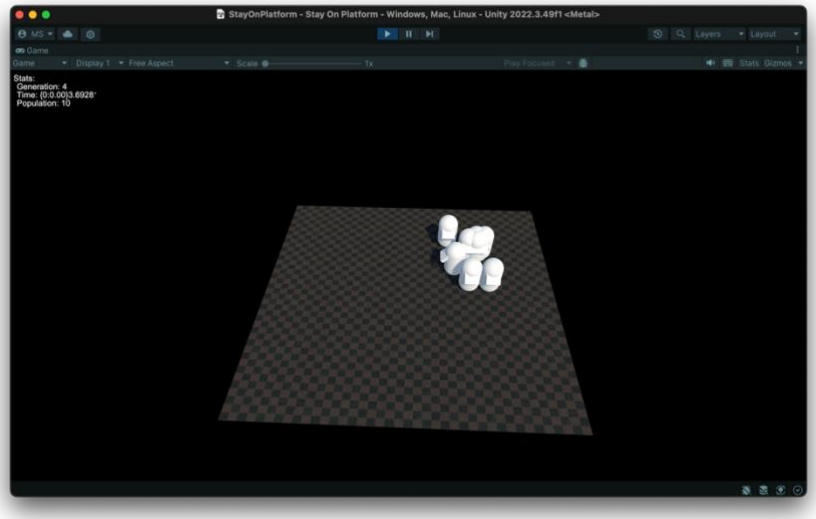
PROJENİN TANITIMI

Bu proje genetik algoritmayı uygulayan bir 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/Hafta4

OYUNUN MANTIĞI

Bu oyun “en güçlü olanın hayatta kalması” prensibini takip etmektedir. Oyun çalıştırıldığında ekranda botların 50 klonlarından oluşan bir nesil çıkmaktadır. Bir nesilin yaşam süresi 10 saniye olup bu süre içerisinde platformdan çıkanlar ölmektedirler. Yaşam süresi bitince ekran yenilenir ve yeni nesil botlar ekranda yerleşir. Ölmeden önce gittikleri mesafeye bağlı olarak en uzun yaşayabilen botlar genlerini bir sonraki nesle aktarabilir. DNA olarak hareket tipleri nesilden nesile aktarılır. Botlar önlerini görebilmektedir. Öne baktıklarında platformu görüp görmemeleri durumunda çalışacak 2 genetik kodları var. Hareket tipleri ilerlemek, sağa dönmek ve sola dönmektir.Bunlardan her çocuk iki hareket ebeveyninden edinir. Fakat bu süreçte mutasyon olabilmektedir ve birkaç tanesi sıra dışı bir genetik kodlamaya sahip olabilmektedir. Nesiller gittikçe tekdüze bir genetiğe sahip olmaktadır.

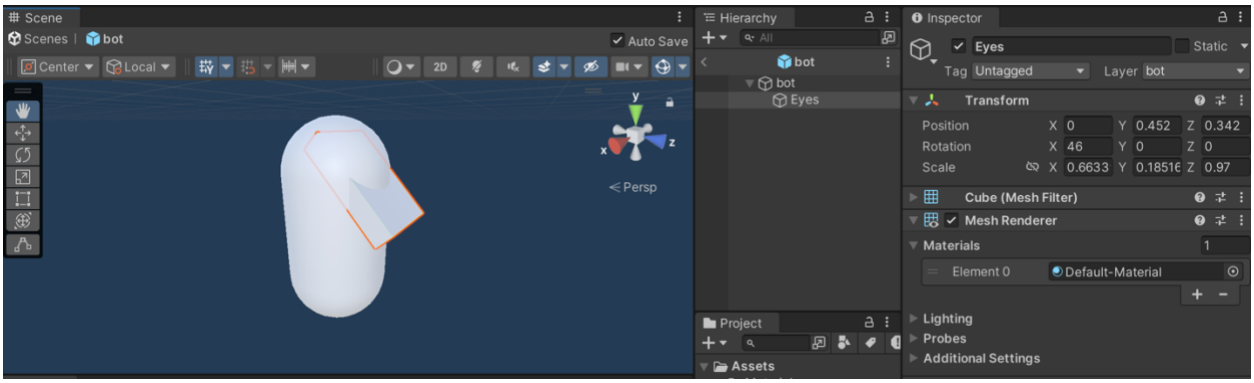




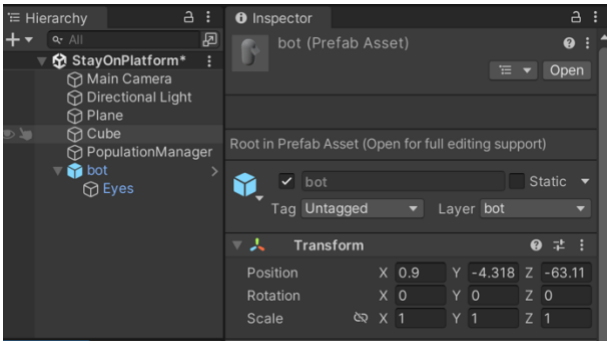
İLK HAZIRLIKLAR

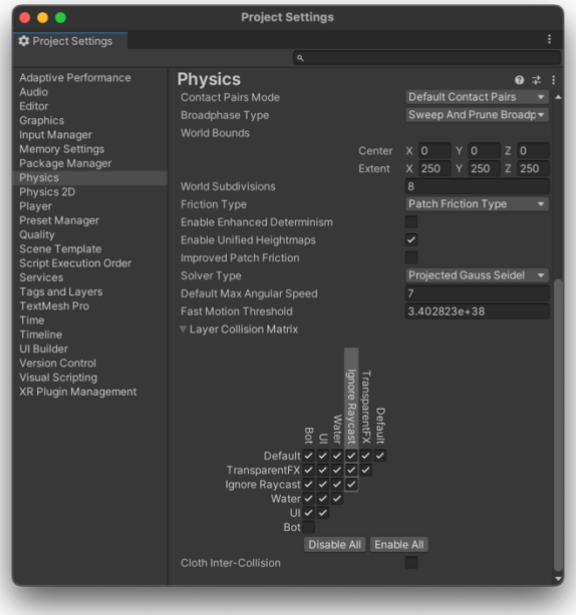
Projeyi başlamak için bu adımları takip etmek gerekmektedir.

1. Unity Hubı açıp 3D proje oluşturunuz
2. Yukarıda verilen githubdan “StayOnPlatformStarter2022.unitypackage” dosyasını indirin ve çift tıklayarak projenize ekleyiniz
3. Projede olan “Sample scene”i silip “StayOnPlatform” adlı sahneyi sürükle bırak ile hierarchy kısmına getiriniz..
4. Assetlerde bot kapsülü bulunuz ve üzerinde çift tıklayınız. Bir kapsül ekrana gelmesi gerekiyor ve heirarchyde sadece o kapsül görünmesi lazım.
5. Heirarchyde sağ tıklayıp “create empty”ı tıklayarak boş bir obje oluşturunuz ve ona “population manager” adını veriniz
6. Heirarchydeki bota sağ tıklayıp “3D”ı tıkladıktan sonra “cube” tıklayıp boş bir küp nesnesi oluşturunuz. Doğru oluşturulduysa botun altında girintili olacak ve ekranda kapsülün tam ortasında olacak. Resimdeki olacak şekilde küpün boyut, konum ve rotasyonunuu ayarlayınız. Bu küp botumuzun gözleri olacağı için inspector kısmından “Eyes” ismini veriniz.



7. Assets kısmında sağ tıklayıp “create”, sonra ise “folder”ı basarak script adlı bir dosya oluşturunuz.
8. Script dosyanın içinde sağ tıklayıp “create”, sonra ise “C# script”i basarak “PopulationManager_sc”, “Brain_sc” ve ”DNA_sc” adlı üç tane script oluşturunuz.
9. PopulationManager scriptini hierarchyde aynı isimli obje üzerinde sürükle bırak ile atayınız.
10. DNA isimli dosyayı da Assets içindeki “bot” isimli kapsül 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
11. Kendi katmanımızı oluşturmak için tag -> new tag tıkladıktan sonra 6. kutudan itibaren olan kutular içinde katmanımızın adını yazabiliriz. Bot objemize bot isimli layer verelim
12. Edit-> project settings tıklayıp aşağa sükükleyip physcis kısmından bot-bot layerlerin çarpılmasını kaldırınız





DNA.CS

Bu script adından anlaşılacağı üzere sineklerimizin DNAsını veya özelliklerin bulunduğu bir scripttir. İçinde farklı özellikleri tutan değişkenler, değişkenlerin ilk ataması yapıldığı constructor fonksiyonu, genleri anne ve babadan alıp birleştiren “combine” fonksiyonu, genleri erişim sağlayan setter ve getter fonksiyonlar ve mustasyon durumunu yöneten “mutate” fonksiyonları bulunmaktadır. Bu dosyayı normal bir C# class gibi yazılması gerektiğinden Monobehavior kısmın class tanımından silinmesi gerekmektedir.

DEĞİŞKENLER

Bu kısımda hareket genin uzunluğu ve alabileceği maximum değeri sayısını (hareket tiplerin sayısını) belirleyen bir değişken tanımlandı.

```
List<int> genes = new List<int>();
    int dnaLength =0;
    int maxValues=0;
```

CONSTRUCTOR FONKSİYONU

Bu fonksiyon bir nesne oluşturunca ilk çalışan fonksiyondur. İçinde ise az önce tanımladığımız değişkenlere atama yapılır.

```
public DNA_sc(int l, int v)
    {
        dnaLength = l;
        maxValues = v;
        SetRandom();
    }
```

COMBINE FONKSİYONU

Bu fonksiyon iki ebeveyenin genetik kodunu alıp %50 bir şans ile birini seçip çocuğa atar.

```
public void Combine(DNA_sc d1, DNA_sc d2){
    for (int i = 0; i < dnaLength; i++)
    {
        if(Random.Range(0,10)>5){
            genes[i]= d1.genes[i];
        }
        else
        {
            genes[i] = d2.genes[i];
        }
    }
}
```

GETTER VE SETTER

Bu fonksiyonlar gen değişkeninin değerini almak ve değiştirmek için kullanılır. Set random genlere rastgele bir hareket atar

```
public void SetRandom(){
    genes.Clear();
    for (int i = 0; i < dnaLength; i++)
    {
        genes.Add(Random.Range(0,maxValues));
    }
}

public void SetInt(int pos, int value){
    genes[pos] = value;
}
public int GetGene(int pos){
    return genes[pos];
}
```

MUTATE FONKSİYONU

Bu fonksiyon ise constructordaki ile aynı işlemi yapıyor ve rastgele bir gen değişkenine rastgele bir değer atıyor. Burada 2 gen kodlardan sadece birinde mutasyon olur.

```
public void Mutate(){
    genes[Random.Range(0,dnaLength)] = Random.Range(0, maxValues);
  }
```

BRAIN.CS

Bu kısımda botların hareketleri belirlenmektedir. Burada hareketi belirlemek için bazı fonksiyonlar tanımlanmıştır.

DEĞİŞKENLER

Burada genlere ulaşmak için DNA tipinde bir DNA değişkeni, botun yaşam süresini ve yaşama durumunu tutan timeAlive ve isAlive değişkenleri, botun ileri baktığında platformu görüp görmemesini kontrol eden canSeeGround ve bot nesnemizin gözleri olan ve hazırlık yaparken eklediğimiz eyes obejisini tutan değişkenleri bulunur. Botumuzun öne bakınca platformu görme durumuna bağlı olarak hareket edeceği için DNAnın uzunluğu 2 dir. Bot üzerinde tıklayarak brain_sc öğesindeki eyes değişkenine botun içindeki eyes nesnesini sürükleyip bırakılmalıdır.

```
  int DNALength =2;
  public float timeAlive;
  public DNA_sc dna_sc;
  public GameObject eyes;
  bool isAlive = true;
  bool canSeeGround = true;
```

ONCOLLISIONENTER FONKSİYONU

Bu unity'e özel fonksiyonlardan biridir ve iki nesne birbirine temas ettiği anda yapılması gerekenler içinde yazılır. Bu fonksiyonu kullanarak platformun altındaki siyah alan (Cube) nesnesinin temas etme durumunda botların ölmesini sağlayacağız. Nesnelere erişmek için bazen tagleri kullanırız. Bunu inspector kısımdan tag üzerinde tıklayarak yapabiliriz. Kendi taglerimizi oluşturmak için tag -> new tag tıkladıktan sonra 6. kutudan itibaren olan kutular içinde taglerimizi yazabiliriz. Daha sonra ise istediğimiz nesneye atayabiliriz. Burada tüm çarpışmaları değil de sadece tagı "dead" olan objeler ile çarpışma olması durumluda botu öldürüyoruz.

```
void OnCollisionEnter(Collision other){
    if(other.gameObject.tag == "dead"){
        isAlive = false;
    }
}
```

INIT FONKSİYONU

Bu fonksiyon az önce tanımladığımız değişkenlere ilk atama yapan fonksiyondur. Hareket tipleri ilerleme, sağa dönme ve sola dönmek olmak üzere 3 çeşit olduğu için DNA nesnemizi oluştururken parametre olarak 3 veririz.

```
public void Init(){
    // 0->forward, 1-> turn left, 2-> turn right
    dna_sc = new DNA_sc(DNALength, 3);
    timeAlive = 0;
    isAlive = true;

}
```

UPDATE FONKSİYONU

Bu fonksiyonun içindeki kodlar hareket biçimleri belirler. İlk önce botumuzun hayatta olup olmasıdığını kontrol ederiz. Eğer öldüyse return ile fonksiyondan çıkarız. Eğer hayatta ise göz nesnemizden ışıltı çıkmasını sağlayan bir sonraki satırlar çalışır. Eğer gözden çıkan ışıltı “platform” taglı nesne ile çarpışışa botumuzun önüne platform var demektir. Bu durumda canSeeGround değişkenine “true” ataması yaparız. Karakterimizin x ve z ekseninde hangi yönde hareket edeceğin DNAsındaki gen değerine göre belirlenir. Rotasyon öğesi Turn değişkendeki değere dayanarak sağa ve sola dönmeyi sağlar. Translate öğesi ise Move değişkendeki değeri hız olarak kullanarak ileriye doğru gitmesini sağlar. Hangi hareket edileceğini botumuzun gen değerleri belirler. Daha önce bahsedildiği üzere platformu görmesi veya görmemesi halinde edeceği hareketi tutan 2 uzunlukta bir genetik yapısına sahiptir. Yine aynı durum kontrol edilerek genetik kodta ne yazıyorsa o hareket gerçekleşir.

```
void Update()
{
    if (!isAlive) return;
    Debug.DrawRay(eyes.transform.position, eyes.transform.forward * 10, Color.red, 5);
    canSeeGround =false;
    RaycastHit hit;
```

```
    if (Physics.Raycast(eyes.transform.position, eyes.transform.forward * 10, out hit))
    {
        if (hit.collider.gameObject.tag == "platform")
        {
            canSeeGround =true;
        }
    }
    //timeAlive = PopulationManager_sc.elapsed;
    float turn = 0;
    float move = 0;

    if (canSeeGround)
    {
        if (dna_sc.GetGene(0)== 0) move =1;
        else  if (dna_sc.GetGene(0)== 1) turn = -90;
        else  if (dna_sc.GetGene(0)== 2) turn = 90;
    }
    else
    {
        if (dna_sc.GetGene(1)== 0) move =1;
        else  if (dna_sc.GetGene(1)== 1) turn = -90;
        else  if (dna_sc.GetGene(1)== 2) turn = 90;
    }
    this.transform.Translate(0,0, move*0.1f);
    this.transform.Rotate(0,turn,0);
}
```

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 50, generasyonun ilk değeri olarak da 1 verilmiştir. “trialTıme” neslin maksimum yaşayabileceği zamanı belirler. Burada 5 verilmiş ancak değiştirilebilir. Her 5 saniyede nesil değişiyor. Ondan sonra nesildeki botları tutan population adlı liste ve tek botı temsil eden botPrefab nesneleri vardır. botPrefab nesnesine botu atmak için unityde

populationManager nesnesine tıklayarak orada skriptin içinde prefab değişkenin yanındaki kutuya Assets dosyasında bulunan bot nesnesinin sürükle bırak ile atılması gerekmektedir

```
public GameObject botPrefab; // bot means an NPC (non playing character)
    public int populationSize = 50;
    List<GameObject> population = new List<GameObject>();
    public static float elapsed = 0;
    public float trialTime = 5;
    int generation = 1;
    GUIStyle guiStyle = new GUIStyle();
```

START FONKSİYONU

Burada botlarımız init fonksiyonu ile oluşturulup population listeye ekleniyor. Instantiate bir objeyi klonlamak veya kopyasını çıkarmak için kullanılan bir fonksiyondur. Burada önce tanımladığımız prefab nesnesinin klonu oluşturuluyor. StartPos değişkeni ise botın ekrandaki başlangıç konumunu beliriliyor. Bu konum da her bot için farklı olacağından rastgele atanıyor. Bu döngü ilk olarak belirlediğimiz nüfüs sayısına kadar (50 kez) tekrarlanıyor. Oluşturulan her nesne ise en sonda bulunan population.Add(bot) fonksiyon ile population listemize ekleniyor.

```
void Start()
  {
     for (int i = 0; i < populationSize; i++)
     {
        Vector3 startPos = new Vector3(this.transform.position.x + Random.Range(-2,2),
                           this.transform.position.y,
                           this.transform.position.z + Random.Range(-2,2));
       GameObject bot =  Instantiate(botPrefab, startPos, this.transform.rotation);
        bot.GetComponent<Brain_sc>().Init();
        population.Add(bot);
     }
  }
 }
```

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;
```

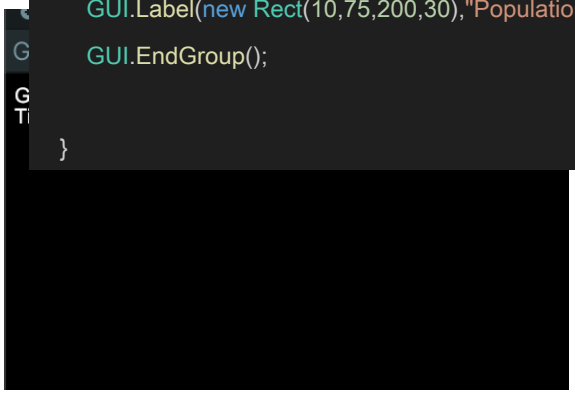
```
    if (elapsed>=trialTime)
    {
        BreedNewPopulation();
        elapsed = 0;
    }
  }
```

ONGUI FONKSİYONU

Bu oyun başlatıldığında sol üst kenarda generasyon numarasını ve geçen süreyi göstermemize sağlayan kod parçasıdır . 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 = 25;
    guiStyle.normal.textColor = Color.white;
    GUI.BeginGroup(new Rect(10,10,250,150));
    GUI.Box(new Rect(0,0,140,140), "Stats: ",guiStyle );
    GUI.Label(new Rect(10,25,200,30),"Generation: " + generation, guiStyle);
    GUI.Label(new Rect(10,50,200,30),"Time: {0:0.00}" + elapsed, guiStyle);
    GUI.Label(new Rect(10,75,200,30),"Population: " + population.Count, guiStyle);
    GUI.EndGroup();

  }
```



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 startPos ile çocuğun konumunu rastgele olarak atar, daha sonra instantiate fonksiyonu ile offspringi oluştutur. Bizim offspring de bir bot olacağından botPrefab nesnesini kopyalıyor. Bir sonraki satırlarda ise offspringin hareket biçimini tanımlarız. İlk önce brain tipinden bir değişken tanımlayıp nesnemizin brain componente erişim sağlarız.

Daha sonra o script içinde bulunan Init fonksiyonu ile botumuzu oluştururuz. Eğer 0-100 arasında oluşturduğumuz rastgele sayı (0 dahil, 100 hariç olmak üzere) 1 ise offspringte bir mutasyon gerçekleşir. Kalan durumlarda ise combine fonksiyonu kullanarak hareket özelliklerini anne veya babasından alır. Burada daha önce anlatıldığı gibi anne veya babadan alma şansı %50dir ancak oranlar isteğe bağlı olarak değiştirilebilir.

```
GameObject Breed(GameObject parent1, GameObject parent2){
      Vector3 startPos = new Vector3(this.transform.position.x + Random.Range(-2,2),
                              this.transform.position.y,
                              this.transform.position.z + Random.Range(-2,2));
        GameObject offspring =  Instantiate(botPrefab, startPos, this.transform.rotation);
        Brain_sc brain = offspring.GetComponent<Brain_sc>();
         brain.Init();
         if (Random.Range(0,100)==1)
         {
            brain.dna_sc.Mutate();
         }
         else{
            brain.dna_sc.Combine(parent1.GetComponent<Brain_sc>().dna_sc,
parent2.GetComponent<Brain_sc>().dna_sc);
         }
         return offspring;
   }
```

BREED NEW POPULATION FONSKSİYONU

Bu fonksiyon hangi botların 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 sıralıyor. Sonra var olan population listeyi boşaltıyor. Yeni nesli oluşturmak için nesilin daha uzun yaşayanlarından (sıralı listenin ikinci yarısından) ardışık iki bot seçerek Breed fonksiyonu ile yeni bir bot üretir. Daha sonra üretilen botlar az önce boşaltılan population listesine tek tek eklenir. Ancak ikinci yarıyı kullanmak sadece 25 bot 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<Brain_sc>().timeAlive).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++;
   }

    generation++;
```