Yapay Zeka Proje

15011020 Mustafa Kara

14011095 Kirubel Afrassa

1- Proje Tanımı:

1-a-- Oyun:

2048 16(4x4) kareden oluşan bir tabloda bulunan sayıları birleştirerek 2048 sayısına ulaşmaya çalışılan bir oyundur.

Oyunda 4 adet hareket mevcuttur. Bu hareketler: Sağ, Sol, Yukarı ve Aşağıdır. Hareketler ile tahtada bulunan bütün taşlar hareketin gösterdiği yöne doğru kayarlar. Oyuna her hamlenin sonucunda belirli bir olasılığa bağlı olarak "2" veya "4" numaralı yeni bir taş eklenir. Hareket sonucunda eğer aynı numaralı iki taş bir biri üzerine gelirse bu taşlar birleşir ve taşların değerinin 2 katı olan yeni bir taş, bu taşların yerini alır.

Oyun 2048 değerli taşa ulaşıldığında veya tanımlanan 4 hamleden hiç birini yapamayacak duruma gelene kadar devam eder.

1-b-- Algoritma:

Depth Limited Search aynı depth first search gibi bir stack mantığı kullanır.

Kökten başlanarak kökün bütün çocukları bir listeye eklenir. Daha sonra bu listeden, bu listeye son olarak eklenen çocuk çekilir ve bu çocuğun çocukları listeye eklenir. Böylece istenilen durum bulunana kadar yada listede bir eleman kalmayana kadar bu işlem devam eder.

Depth limited Search ise bu arama sırasında bir limit koyarak ağacın belirli bir derinliğinden aşağıya bakmadan aramasını yapar.

1-c—Proje:

Projemiz 2048 adlı oyunun Depth Limited Search algoritması kullanarak çözülmesidir.

Proje Python dilinde yazılmıştır. Oyunun gerçeklenmesi "TKinter" adlı kütüphane ile yapılmıştır.

Oyunda yeni taş %20 ihtimalle "4", %80 ihtimalle "2" olur. Oyun Hareket seçerken bizim tanımladığımız bir fonksiyon çağırır ve sıradaki adımı öğrenir. Oyun tahtası bir matris içerisinde tutulur ve bu matristeki değerlere göre grafik ara yüzde farklı değerlerde ve farklı renklerde taşlar gösterilir. Her hareketin sonunda oyun tamamlandı mı diye kontrol edilir ve sonuca göre kullanıcı uyarılır.

2-Projenin Uygulaması:

2-a—Aramanın Uygulanması:

DLS Algoritmasını recursive fonksiyonla gerçekledik. Bu fonksiyon kendi içerisinde her bir farklı yön için 4 kere çağrılır. Bu çağırma işleminde “Path” adlı bir parametre ile oyun tahtası gönderilir. “Path” adlı parametre tahtanın şu anki durumuna ulaşmak için yapılan hamleleri gösterir(“LURD”-> left,up,right,down).

Fonksiyon Derinliği “Path” adlı parametre sayesinde kontrol eder ve bu parametrenin boyutu Derinlik sınırına ulaştığında tahtanın şu anki durumu için bulduğu skoru geri döndürür.

Arama yaparken bazı hamleler yapılamaz. Bu durum eğer ağacın kök dışındaki bir yerinde meydana geldiyse yukarıdaki node’a bir işaret gönderilir ve yukarıdaki node’un skoru köke geri döndürülür. Eğer bu durum kökte meydana geldiyse bu durumun skoru negatif bir değer alır. Aynı şekilde oyunun kaybedildiği durumlarda böyle değerlendirilir.

2-b—Skor:

Skor hesaplanırken 3 farklı olaya bakılır:

1- Taşların yerlerine: oyun tahtasının her bir noktasının farklı bir katsayısı var. Bu katsayı değeri tahtanın bu noktasının ne kadar değerli olduğunu gösteriyor.

```
Mask=[[26,24,10,-1],[28,22,12,-1],[30,20,14,-1],[16000,18,16,-1]]
```

Biz oyunda en değerli nokta olarak sol en altı belirledik. Ve oyunun sol duvara yaslanmış bir şekilde ilerlemesini istedik. Bu yüzden bu noktaların katsayısı fazla. Tahtada bulunan bütün taşların değerleri bulundukları noktanın katsayısı ile çarpılır ve toplanır. Bu toplam skora gönderilir.

2-Tahtada bulunan değerler: eğer tahtada aynı değerden birden fazla varsa bu gereksiz yer kaybı olduğu anlamına gelir. Bu yüzden eğer aynı değerden birden fazla varsa bu skora bu değerin bulunduğu **katsayı x taşın değeri x 0.05** kadar etki eder. Böylece oyunda aynı taştan birden fazla bulunmasını azaltmaya çalışıyoruz.

3- Taşın yanında Bulunan taşlar: eğer bir taşın üstünde ve sağ yanında bu taşın yarısı kadar yada aynı değerde taş bulunuyorsa bunu artı değer olarak skora ekliyoruz. Böylece oyunda taşların yan yana konulmasına çalışıyoruz.

Bu üç farklı olayın toplamı skoru oluşturur.

2-c—Oyunun oynanışı:

Oyun her bir adımda uygulamak için algoritmadan bir adım istiyor. Adım istediği bu fonksiyon içerisinde gidilebilecek 4 yön için arama algoritmasını çağırıyor. Bu dört yön için skorları aldıktan sonra bu skorlar arasından büyük olan yönü Oyuna gönderiyor. Oyun bitene kadar bu işlemler tekrarlanıyor.

2-d—Oyun içerisindeki olasılık durumu hakkında:

Oyun her bir harekette rastgele olarak bir taşı boş olan karelerden birisine koyuyor. Bu durum oyuna bir olasılık durumu katmakta. Arama yapılırken ağaçta derine indikçe oyun gerçekten uzaklaşıyor.

Bu durum eğer skora hiç katılmazsa skor gerçekten en uzakta olacağı biçimi alıyor. Bu durumda oyun sonuca ulaşamıyor. Bu olasılık durumunun skora katkısını min max algoritmasındaki utility katmanı gibi uygulamaya karar verdik.

DLS Algoritması arama yaparken ağaçtan her yeni hareket uygulandığında oyun tahtasında bulunan her bir boş karede sanki bir taş varmış gibi yeni bir tahta üretiliyor(diyelim ki şu anki hareket sonucunda tahtada 5 boş nokta var ise o zaman 5 farklı durum oluşmakta). Bu durumların her biri üzerinde 4 farklı hareket uygulanacak biçimde ağaca ekleniyor.(yine yukarıda verilen örnek gibi 5 tane boş kare varsa, normalde olacağı gibi bu durumun 4 çocuğu değil de 20 çocuğu oluşmakta). DLS bu çocuklardan her birine bakar ve skor hesaplanırken bütün durumların aynı komutları toplanır ve maksimum skor döndürülür(yine aynı örnekte “L” komutu için toplamda 5 farklı skor oluşur. Bu skorların toplamı “L” hareketinin skorunu belirtir). Böylece olasılıklar eşit olarak skora etki eder.

Burada ayrıca 2 veya 4 olma olasılığı da var. Ancak bizim için skora etki eden en büyük etken taşın nerede olduğu. Çünkü 5 adımda 2 veya 4 olması pek bir şey değiştirmezken en büyük değerli taşın sol attaki köşeden çıkması skoru baya değiştirmektedir.

3-Performans:

3-a—Oyun Sayısal Başarısı:

Sayısal başarıyı belirli bir sayıda oyunun kaç tanesinde 2048'e ulaşabiliyor olarak belirledik. Yaptığımız testlerde 30 oyunun 23'ünü başarıyla tamamladığını gördük. Buda **%76,67** başarı demektir.

3-b—Başarıya Etki Eden Etmenler:

Oyunun içerisinde ki olasılık durumu başarıya etki eden en önemli etken.

Olasılık ilk olarak yaptığımız hesaplamaların bir tahmin durumuna düşürüyor. Bu da belirlediğimiz en iyi hareket içerisinde oyunu çok kötü duruma sokabilecek ihtimallerde bulunmakta. Bu durum ağaçta ne kadar çok aşağıya inersek(Depth ne kadar fazla ise) gerçekten aslında o kadar uzaklaştığımızı ve hesapladığımız skorun o kadar anlamsızlaştığı anlamına gelmekte.

Bu durum ne kadar ileri bakabileceğimizi limitlemekte. Ancak eğer çok yüzeysel bakarsak algoritmanın yapacağı stratejiler o kadar basit olacak. Buda bir iki hamle ilerisinde ulaşılabilecek en iyi durumların kaçırılması anlamına geliyor.

Aynı zamanda eğer ileriye bakıldığında yukarıda ki durum gerçekleşmese bile ağaçta yarattığı artış nedeniyle hesaplama uzuyor ve bir noktada artık imkansız hale geliyor. Yani DLS de bulunan Dept bir trade off içerisinde. Buda algoritma için en optimal bir düzey olduğu anlamına geliyor.1 ve iki beklendiği gibi çok basit gelmekte ve 4 için yeterli donanımımız olmadığı için bütün denemeler Depth=3 için gerçekleştirilmiştir. Teorik olarak 3'ten büyük bir depth için daha iyi bir başarı şansı yakalayabilir.

Oyunda bulunan bu olasılık yaptığımız denemeleri tekrarlamamız halinde başka bir sonuçta elde edebileceğimiz anlamına geliyor. Bu sıkıntıyı tekrar sayısını arttırarak gidere bilsekte zaman ve donanım sıkıntılarından dolayı tekrar sayısı az tutulmuştur.

Yine bu olasılıktan dolayı gerçek bir insan bu oyunu oynadığı zaman da kazanma garantisi yoktur. Ancak haliyle bir insanın donanım sıkıntısı olamadığı için yaptığımız algoritmadan daha karmaşık stratejiler oluşturabileceği için insanın bu oyundaki başarası algoritmamızdan fazladır. Teorik olarak daha yüksek bir depth sayısının insan başarısını geçme ihtimali vardır.

4-Referanslar:

1- https://github.com/yangshun/2048-python: 2048 Oyunu Orijinal Kodu.