**GAZI ÜNIVERSITESI BIGISAYAR MÜHENDISLIĞI**

**YAPAY ZEKA DERSI UYGULAMALI PROJE**


**Proje:** Pacman oyunu tasarlanacak. Bir kaynağa ulaşmak için DFS, BFS, DLS (Sınırlı Derinlikte Arama), A* Manhattan, A* Eucledian algoritmaları kullanılacak ve karsilastirilaccaktir. Ayrıca 4 farklı hedefi Pacman Haritasının köşelerine konumlandırarak A* algoritması ile hepsine erişen optimum yol çizilecektir.

Öğretmen: Yılmaz ATAY

Öğrenci: Bahattin AKSOY

Son Teslim Tarihi: 16.12.2020

Numara: 171180004

## Bölüm-1: PLATFORM

Pacman Projesini Windows işletim sistemli bir bilgisayar ile yaptım.

Kullandığım platform Visual Studio 2019 dur. Tercih etmemin sebebi daha önce proje gerçekleştirdiğim bir platform olduğundandır.

Programlama dili olarak C# tercih ettim.

## Bölüm-2: PROBLEM

Problem Pacman'e uygun sekilde yön vererek hedefe ulaşmasını sağlamak.

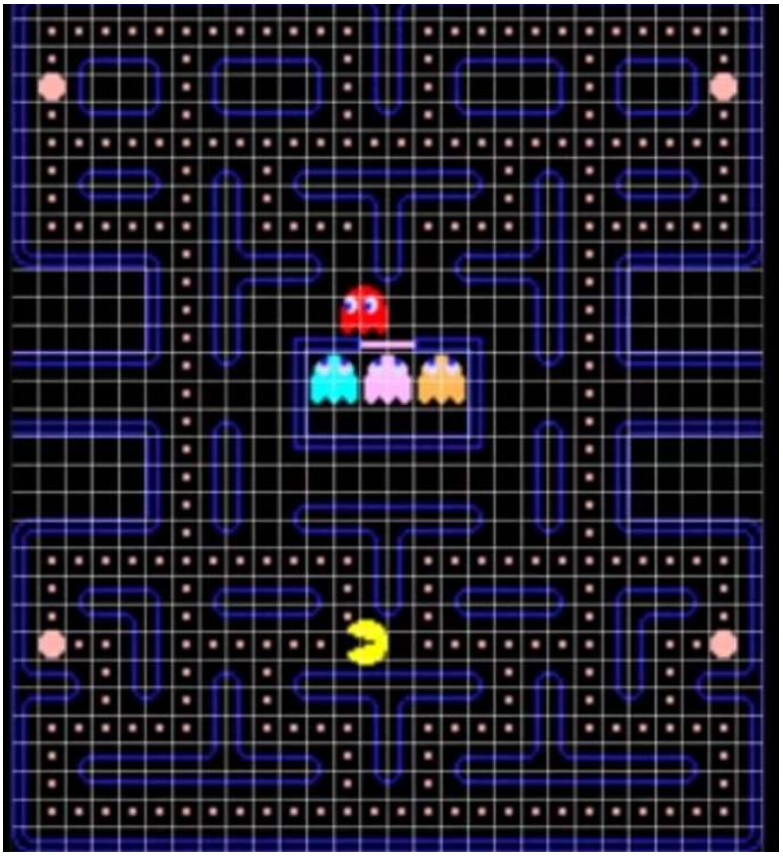
## Bölüm-3: KOD BİLGİLERİ

1.Adım Panel

Içerisindeki nesleri sıralayabilen bir dizi eleamnlarına erişilebilen boş bir panel oluşturdum. Arka plan resmini Pacman Map yaptim.

2.Adım Map Matris

Arayüzü  kendim oluşturduğum için resim üzerindeki yolları bilgisayarın keşfetmesi olanaksız. İki boyutlu matris oluşturdum. Haritayı uygun parçalara ayırdıktan sonra duvar için 0, yol için 1 değerlerini oluşturduğum diziye ekledim. Bu sayede yollar bilgisayar ortamında tanımlamış oldu.



3.Adım Kutular

Matrisdeki deger kadar kutu ürettim ve bu kutuları panele ekledim. Kutuların resimlerini değiştirerek pacman geçişlerini sağladım.

4.Adım Hareket ve genel fonksiyonlar

Kodları proje dosyalarından erişilebilir. Genel olarak Fonksiyonlar (Algoritamaya göre farklılık gösteriyor):

Matris_doldur: Haritayı bilgisayarın algılayabileceği diziye dönüstürü.

Move: pacmanin komsularına bakan ve maliyet hesabı yaparak rekürsif bir şeklide kendisini çağırır. Içerisinde ek olarak (ayrı ayrı fonksiyon yerine) geri adım atma fonksiyonunu içerir. Paneldeki kutuların resimlerini değiştirerek Pacaman adımlarını ve Pacamni görsel oalrak takip etmeyi sağlar (Bu amaca yönelik fonksiyonun içerisinde 80ms bekleme komutu vardır).

List_olustur: Matrisdeki bilgi ile gerekli algoritamaya göre verileri sınıf şeklinde oluşturur. Komşularını ekler ve liste haline getirir.

Panel_doldur: Paneli matristeki veriler kadar kutu ile doldurur.

Eucledian: eucledian mesafesini hesaplar

Maliyet: Baslangic noktasına olan adım mesafesini hesaplar.

5.Adım Veri

Matris üzerindeki her 1 için yani yol için Verileri class şeklinde oluşturdum. Ve bu tipteki klasları tutan bir liste elemanı oluşturdum, komsularınıda içerisindeki listede tuttum. Bu sekilde Pacman in hareket edebileceği tüm dallanmayı elde ettim.

```
List<node> list=new List<node>();
15 references
class node
{
    public int idx=0;
    public int x = 0;
    public int y = 0;
    public double eucledian = 0.0;
    public int maliyet = 0;
    public bool ziyaret_edildimi = false;
    public List<node> komsular = new List<node>();
}
```

6.Adım Diger Yığınlar

Her algoritmada gerekli yığını oluşturdum.

1: Pacman'in geri adım atabilmesi için sondan eklediğim ve çıkardığım yığın

```
List<int> stack;
```

2: Tüm yolların paneldeki pozisyonunu tutan yığın

```
List<int> position_1 = new List<int>();
```

3: Bfs için gittiği adımlarda enlemesine aramayı sağlamak için baştan veri ekleyip,silen yığın.

```
List<node_dfs> bfs_list;
```

```
bfs_move(bfs_list[0]);
```

## Bölüm-4: SONUÇLAR

Bilgisiz arama yöntemleri verinin nerede olduğu ile bilgisi olmadığından programın çalışması daha hızlı oluyor fakat hedefein yanından geçip gidebiliyor.

DFS: Pacaman için ideal bir arama yöntemi değil çünkü gidebildiği en uzak noktaya gidiyor ve orada arama yapıyor. Hedefin yanından geçtiği için geri dönmesi extra maliyet yaratıyor. Süresinin uzun olmasının sebebi ise adım sayısının fazla olması.

BFS:Pacman için uygun olmayan algoritma. Sürekli enlemesine arama yapmak pacman için hareket atlamasına karşılık geliyor. Atlamak istenilen yerler bulunan pozisyona komsu olmuyor. DFS'ye oranla biraz daha yavas ve parent node tutmak için extra hafızaya ihtiyaç duyuyor. Süre olark ise ortalama DFS ye oranla daha iyi.

Sınırlı Derinlikte Arama: DFS  ye alterneatif olarak uzak noktalara gitmesi engelleniyor. Doğru limit ile DFS'ye oranla çok daha iyi sonuç veriyor. Aynı hızda olduklarından daha az adım ile daha az sürede hedefe ulaşmış oluyor.

A*: Extra veri tutulması gerekiyor. Ve bu verilerin hesaplanması için bilgisayar ayrı bir çaba sarfediyor.

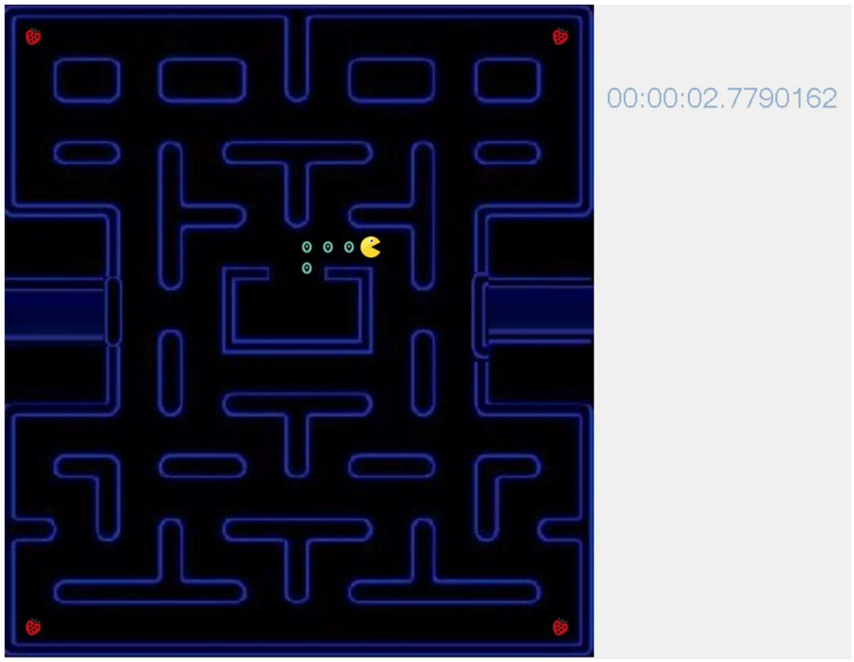
Fakat optimum yolu daha az adımda bulabiliyor. Veriler oluştuktan sonra ise daha hızlı çalışıyor. Sadece extra veri tutulmuş oluyor.

A* Manhattan ve  A* Eucledian karsilatirması asğıdadır

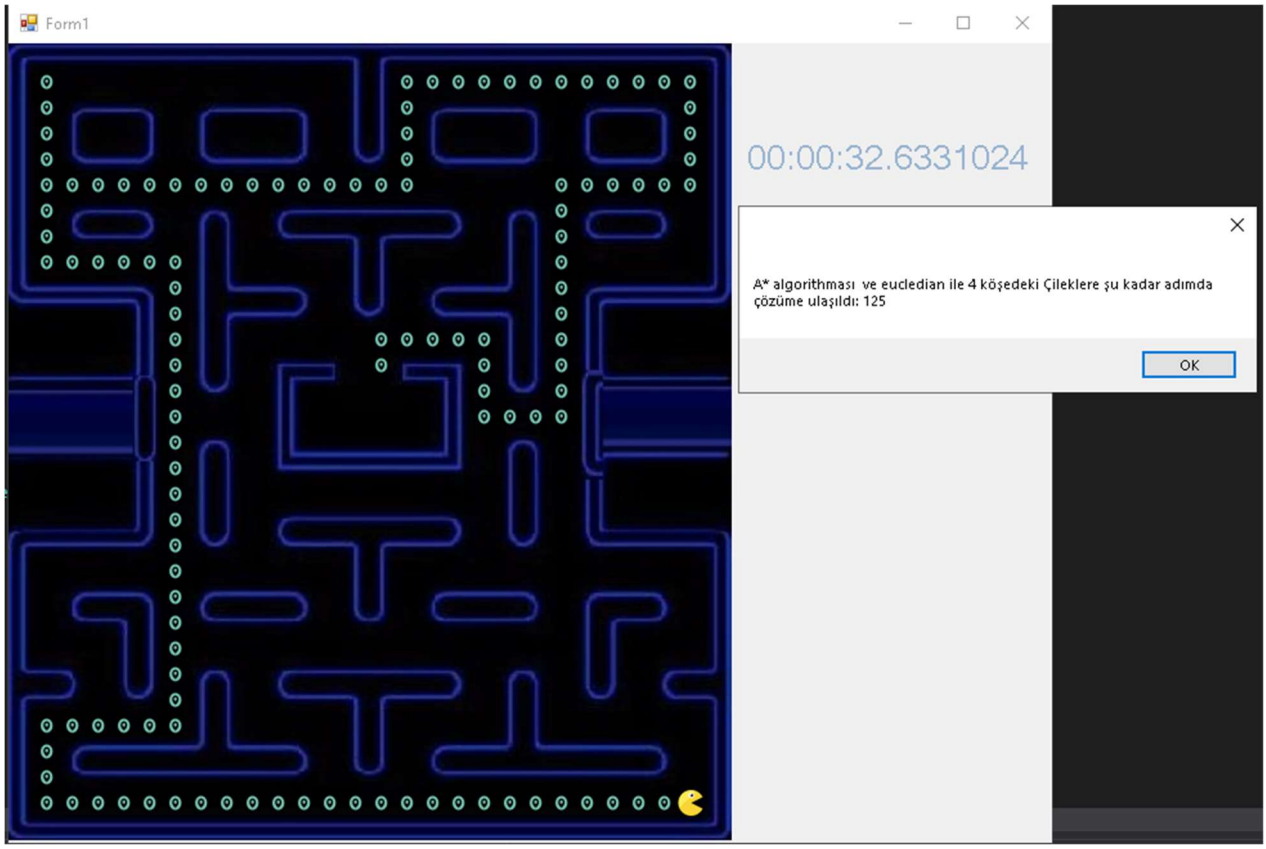
A* Eucledian 4 Hedef: Her hedefe gittiğimizde diğer hedefe kalan sezgisel maliyetin hesaplanması, baslangıç pozisyonu degistiği için her yolun roota göre maliyetinin hesaplanması çok fazla işlem gücü gerektiriyor ama A* ile optimum yol bulunuyor.

- Hedefe ulaştığında pacmanin yeni pozisyonuna göre en yakın hedef seçiliyor.
- Pacmane en yakın hedef bulunuyor. Veri yığınından bu çilek çıkartılıyor.
- Eucledian ve Maliyet tekrar hesaplanıyor.

Baslangıç:




Sonuç:

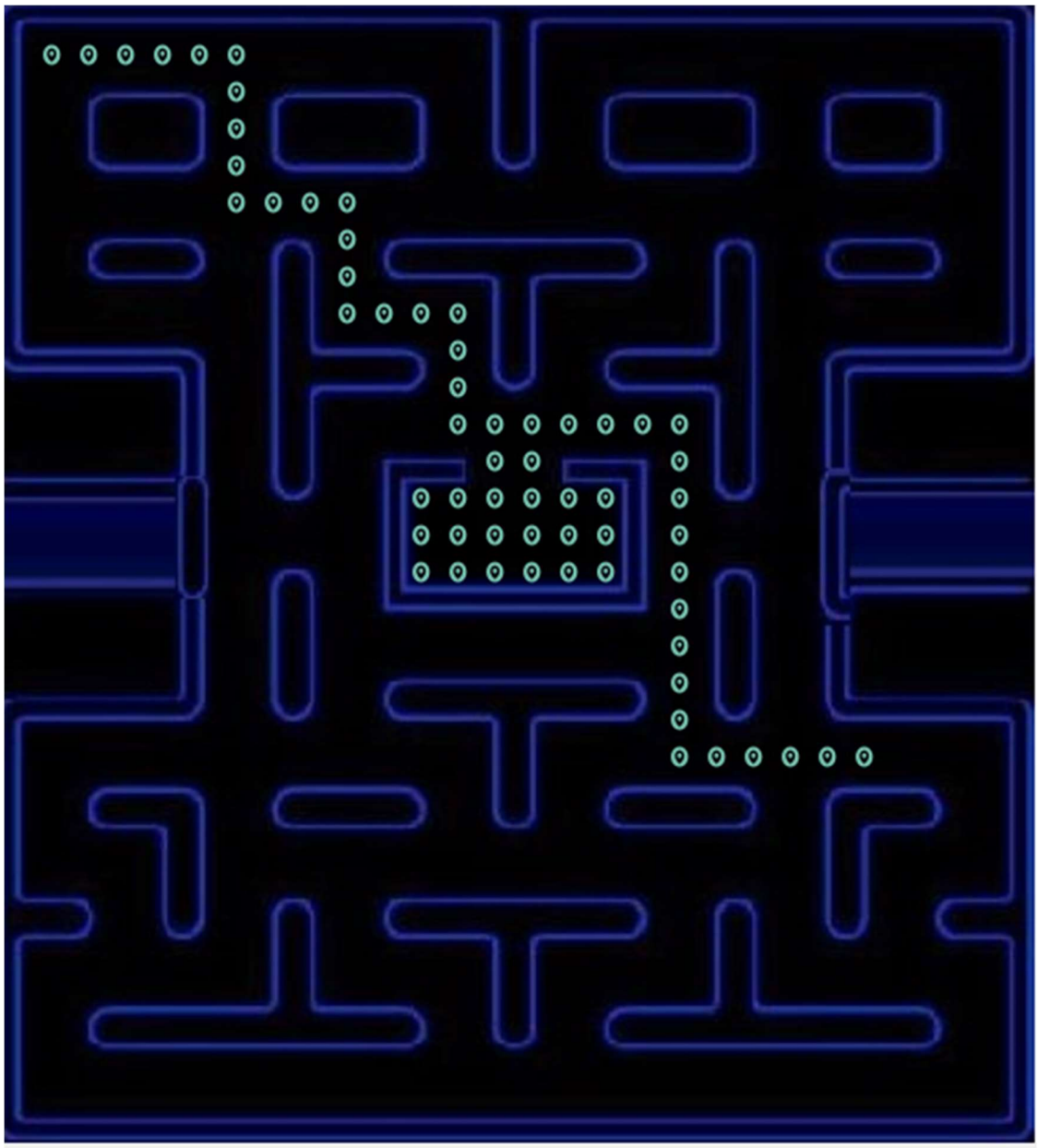


Yollara Eucledian maliyettten kaçmak isterken daha çok dolanabiliyor:

Çünkü maliyetli yolu seçmek toplam maliyeti büyük oranda etkiliyor. Eucledian ise kök işemi yaptığından dolayı root_maliyete göre etkisini azaltıyor.

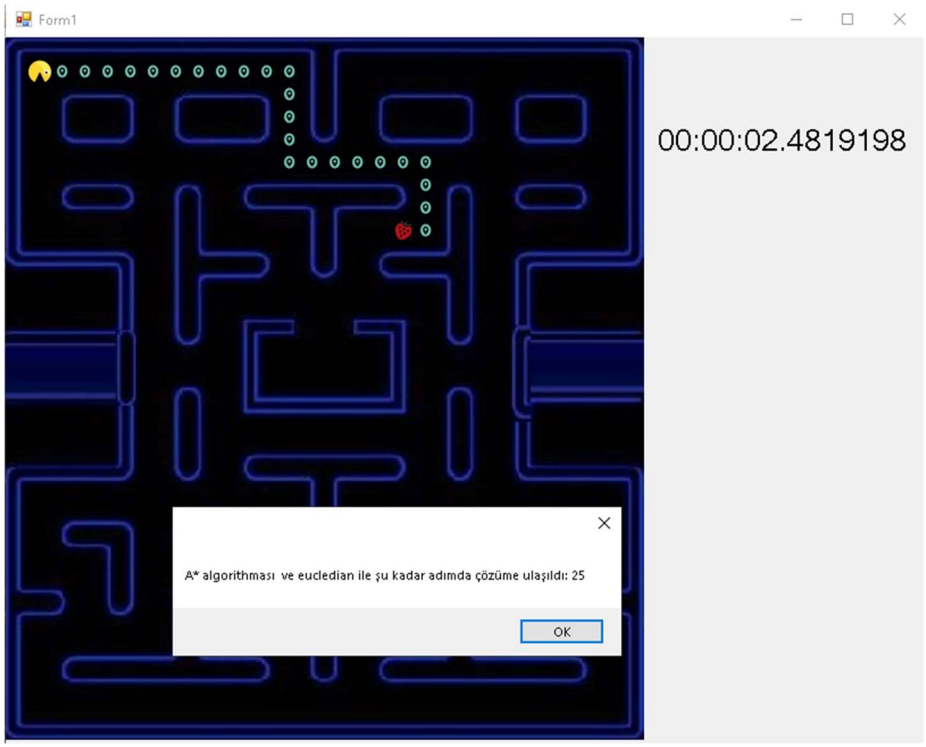
Bu sorun manhattanda daha nadir rastlanmaktadır.

- A* algoritması Yukarıdan baslayarak ilerleyen pacaman maliyeti daha iyi olan fakat ilerde daha çok maliite yol açacak yolu secebiliyor.

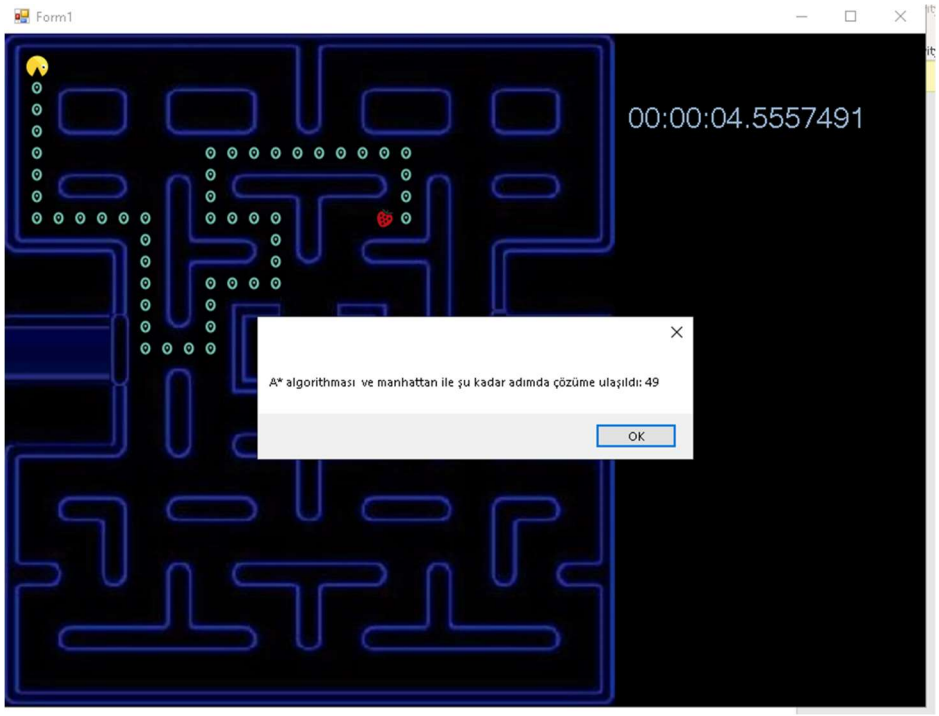


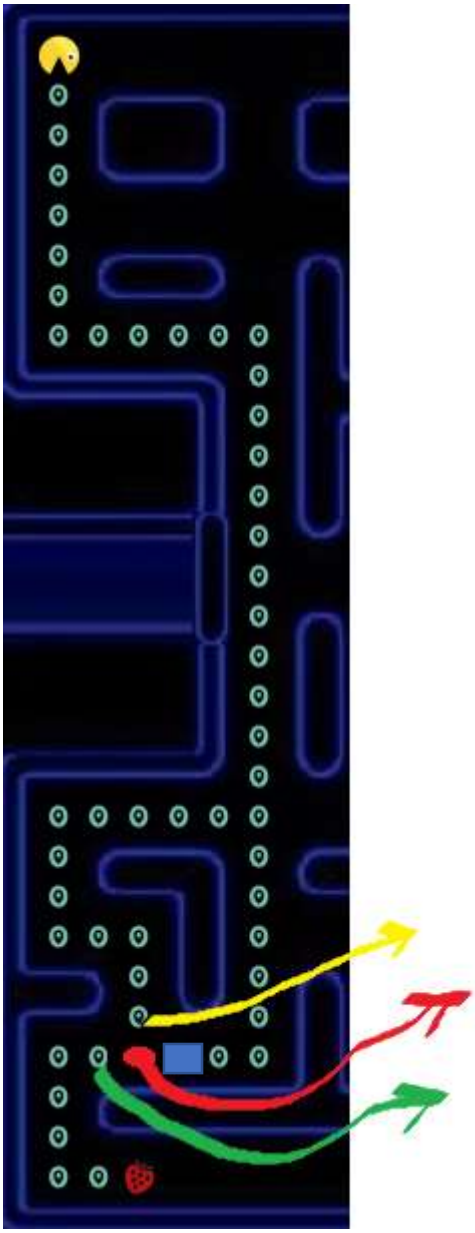
- Pacaman için Manhattan daha kötü bir sezgisellik sunuyor. Karsılastırması aşağıdadır:

Eucledian 25 Adım:




Manhattan 49 Adım:





Not: Mavi kare önceki pozisyon.

Kırmızı şimdi pozisyon ve burada karar aşamasında.

Kırmızı Location: (27,4)

Cilek Location: (30,4)

Gidebileceği yerler:

Sarı: (26,4)

Yesil (27,3)

Bu yerlerin roota uzaklıkları da aynı olduğu için x diyelim;

Manhattan sarı: |26-30|+|4-4| = 4

Manhattan yesil: |27-30| + |4-3|= 4

Toplam maliyeleri eşit olduğu için ½ oranla yanlış yolu seçti.

---

## *Aynı durumu Eucledian ile baktığımızda:*

Eucledian sarı: KÖK[ (26-30)^2 +(4-4)^2 ] = 4

Eucledian yesil: KÖK[ (27-30)^2 + (4-3)^2 ] = 3,1

Yeşil yol toplam maliyeti daha karlı olduğu için yeşil yolu devam edecekti.

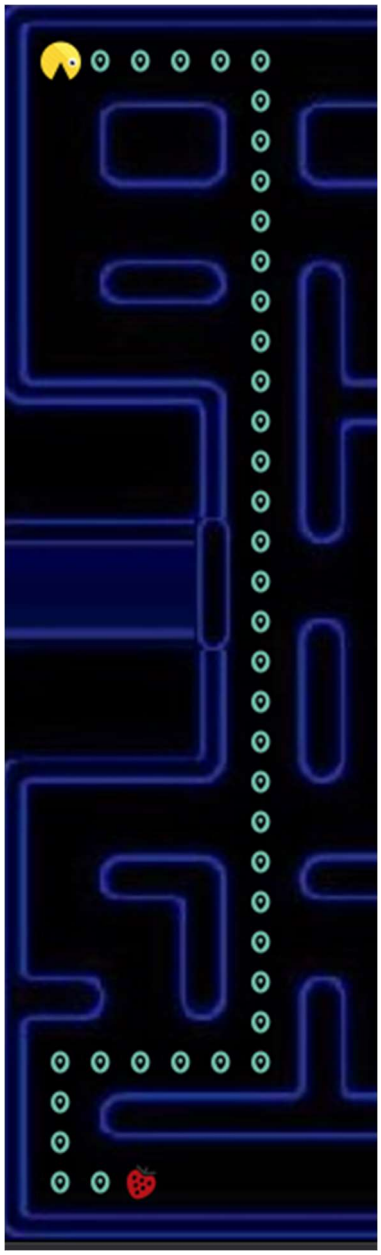
---

## ÇÖZÜM

Manhattan problem çözümü ise öncelik vermek. Pacman başlangıç pozisyounu yukarıda olduğu için tercih sırası aşağıdaki gibidir:

1. Sağ
2. Sol
3. Aşağı
4. Yukarı

Bu sıralamayı ise komsularını eklerken yaptım, bu sekilde doğru sırayla maliyet kontrolü yapıyor ve karsılastırıyor.



## Bölüm 5: KAYNAKLAR

Bu projeyi yaparken farklı kaynaklardan kod almadım. Tüm kodlar ve arayüz oluşturma ve yönlendirmeyi kendim yaptım.