

A\*algoritmasıyla labirent cozum Raporu

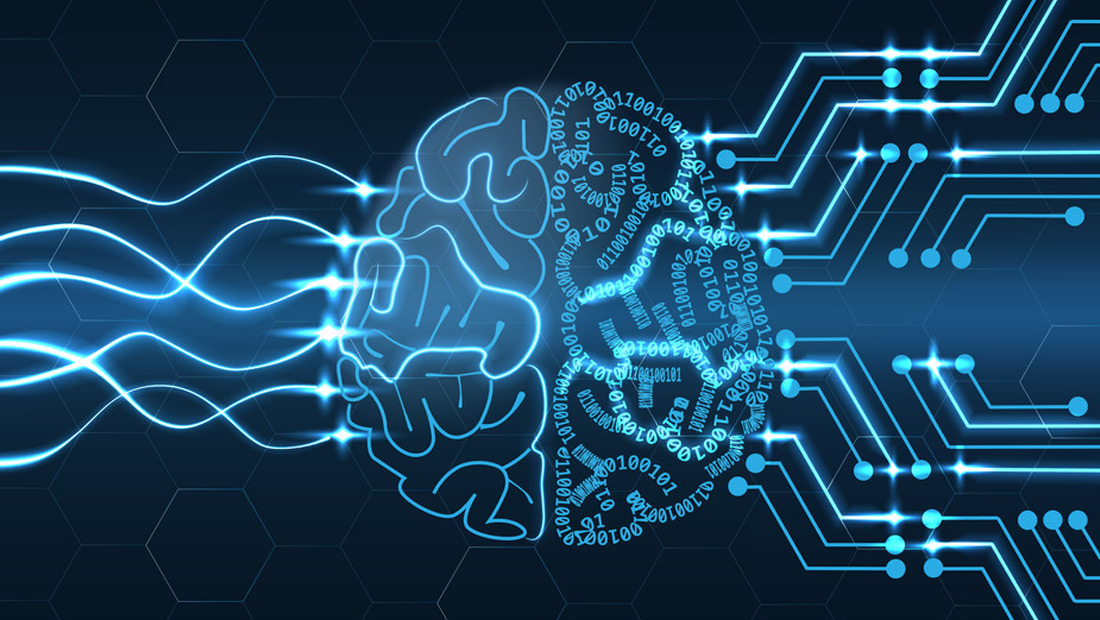
AD SOYAD

Nazli Dilara AKSOY

AD SOYAD

Sude Beyza CIFCI

Bilgisayar Muhendisligi



**PROJE OZETI :**

Projemizde ,labirenti A star algoritmasi kullanarak cozumlemeyi amaçlamaktayız.Labirent bir giriş ve bir cikisi olan ,engeller arasindan geçerek cikisi bulmayi amaçlayan bir oyundur.

**PROJEDE VERİLEN GÖREVİN ACİKLANMASİ, GERCEKLENME ADİMLARİ:**

-**Harita olusturulmasi :**

* Harita engeller ve hedefe ulaşabilecek bos alanlardan oluşur.
* Kullaniciya izgara sunulur (izgara liste ile düğümleri tutar) onun üzerinde fare ile gezinir.
* Fare tiklamalariyla baslangic,hedef ve bariyer düğümleri belirlenir.

-**Hareket:**

* Kullanici sag ve sol tıklamayla baslangic,hedef dugumunu belirledikten sonra klavyeden ‘space’tusuna basarak a\*algoritmasini calistirir.
* A\*algoritmasi baslangictan hedefe giden en kisa yol için kullanılır. Algoritma her adimda acik düğümleri değerlendirir, komsu düğümleri ziyaret eder maliyetleri hesaplayıp en iyi yol seçimini yapar.Bu işlem Kuyruk veri yapisiyla ilerler.
* Her adimda düğümler renklendirilir ve arayuz güncellenir.Boylelikle işlem adimlari gorsellestirilmis olur.
* En kisa yol bulunduktan sonra bu yolun düğümleri farkli renkle vurgulanir.

**-A\* Algoritmasinin uygulanmasi:**

* Baslangic ve hedef dugum belirlendikten sonra openset adında bir öncelikli kuyruk oluşturulur. Bu işlenecek düğümlerin tutulduğu yapıdır. Sonrasinda predecessor adında bir sözlük oluşturulur, bu sözlük düğümler arasındaki ilişkiyi kaydetmek için kullanılır.
* Gscore adında bir sözlük oluşturulur ve tüm düğümlere sonsuz bir değer atanır. Başlangıç düğümünden her düğüme olan gerçek maliyeti temsil eder.
* Fscore adında bir sözlük oluşturulur ve tüm düğümlere sonsuz bir değer atanır. Başlangıç düğümünden hedef düğüme olan tahmini maliyeti temsil eder. İlk olarak her düğüm için sonsuz değer atanır, daha sonra güncellenir.
* openSetHash adında bir küme oluşturulur ve başlangıç düğümü (startNode) bu küme içine eklenir. Bu küme, openSet içindeki düğümlerin tekrar tekrar eklenmesini önlemek için kullanılır.
* openSet boş olana kadar aşağıdaki adımlar tekrarlanır:
* openSet içinden en düşük tahmini maliyete (fScore) sahip düğüm (current) seçilir.
* Seçilen düğüm openSetHash kümesinden çıkarılır.
* Eğer seçilen düğüm hedef düğümse, en kısa yol bulunmuştur. drawPath fonksiyonu çağrılarak yol çizilir ve işlem sonlandırılır.
* Seçilen düğümün komşuları kontrol edilir. Her komşu için aşağıdaki adımlar gerçekleştirilir:
* Yeni geçiş maliyeti hesaplanır.
* Eğer bu geçiş maliyeti, komşu düğümün mevcut geçiş maliyetinden daha düşükse, geçiş yapılır ve aşağıdaki güncellemeler yapılır:
* predecessor sözlüğünde, komşu düğümün önceki düğüm olarak seçilen düğüm kaydedilir.
* gScore sözlüğünde, komşu düğümün yeni geçiş maliyeti güncellenir.
* fScore sözlüğünde, komşu düğümün yeni tahmini maliyeti güncellenir.
* Eğer komşu düğüm openSetHash kümesinde bulunmuyorsa, count değeri artırılarak öncelikli kuyruğa (openSet) eklenir ve openSetHash kümesine eklenir. Ayrıca, komşu düğüm makeOpen metoduyla açık olarak işaretlenir.
* İşlemler tamamlandıktan sonra, her iterasyonda pencerenin güncellenmesi için draw fonksiyonu çağrılır. Bu sayede işlem sırasında arayüzde güncellemeler görülebilir.
* Algoritma, hedef düğüme ulaşamadan openSet boşaldığında sonlanır ve False döner.

**KULLANİLAN ALGORİTMA:**

**A\*(STAR) ALGORITMASI:** A\* algoritması yapı olarak muteber sezgisel (admissable heuristic) bir algoritma olarak sınıflandırılabilir. Bunun sebebi algoritmasının mesafe hesaplamada kullandığı fonksiyondur:

f(n) = g(n) + h(n) denklemindeki

f(n) = hesaplama yapan sezgisel (heuristic) fonksiyon.

g(n) = Başlangıç düğümünden mevcut düğüme kadar gelmenin maliyeti

h(n) = Mevcut düğümden hedef düğüme varmak için tahmin edilen mesafe.

Dikkat edileceği üzere f(n) fonksiyonunun sezgisel olma sebebi, bu fonksiyon içerisinde bulunan ve tahmine dayalı olan h(n) sezgisel fonksiyonudur.

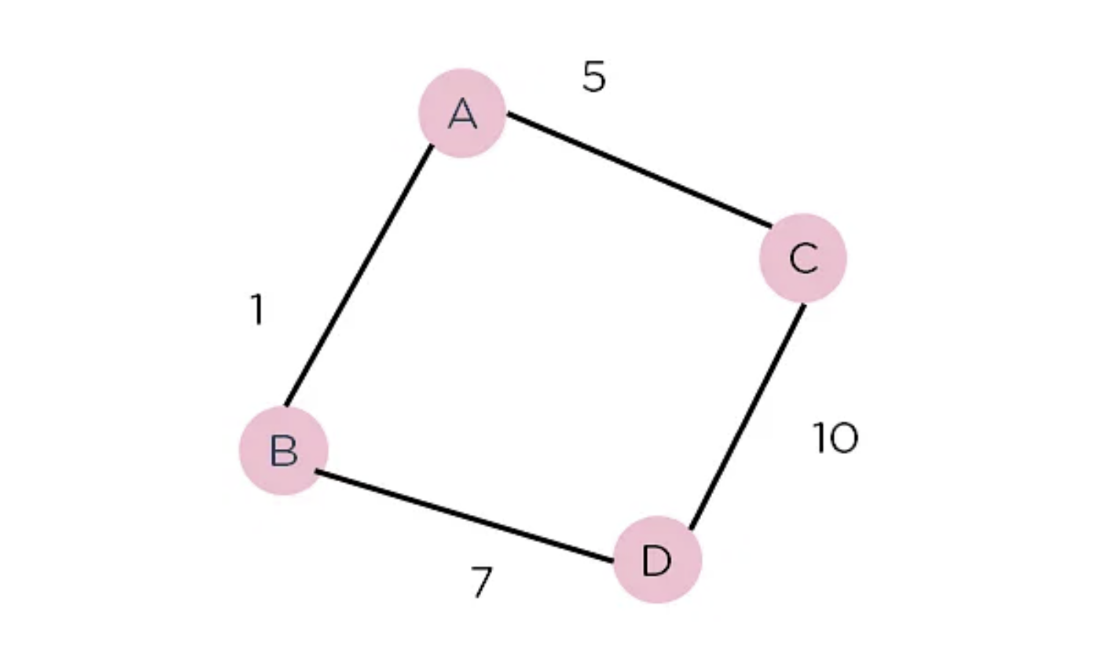
**Algoritmanın çalışması:**

Algoritma yukarıdaki toplama işlemini kullanan oldukça basit bir yapıya sahiptir. Veri yapısı olarak bir öncelik sırası (priority queue) kullanan algoritmada en öncelikli olan düğüm f(n) değeri en düşük olan düğümdür.

Algoritma her adımda en düşük değeri (Ve dolayısıyla en önemli) düğümü alır (yani bu düğüme gider) ve düğümü sıradan (queue) çıkarır.

Gidilen bu düğüme göre komşu olan bütün düğümlerin değerleri güncellenir (artık bu düğüme gelmenin bir maliyeti vardır ve dikkat edilirse f(n) fonksiyonu içerisinde bu değer yer almaktadır.)

Algoritma yukarıdaki adımları hedefe varana kadar (yani hedef düğümü öncelik sırasında (priority queue) en öne gelene kadar) veya sırada (queue) düğüm kalmayana kadar tekrarlar.



Yukarıda gösterilen ağırlıklı grafiği ele alalım. Bu grafikte düğümler ve aralarındaki mesafeler bulunmaktadır. Diyelim ki A'dan başlayarak D'ye gitmeniz gerekiyor.

Başlangıç noktası A olduğu için başlangıçta bazı başlangıç heuristic değerine sahip olacaktır. Sonuçlar şu şekildedir:

f(A) = g(A) + h(A) f(A) = 0 + 6 = 6

Sonra diğer komşu düğümlere geçilir:

f(A-B) = 1 + 4

f(A-C) = 5 + 2

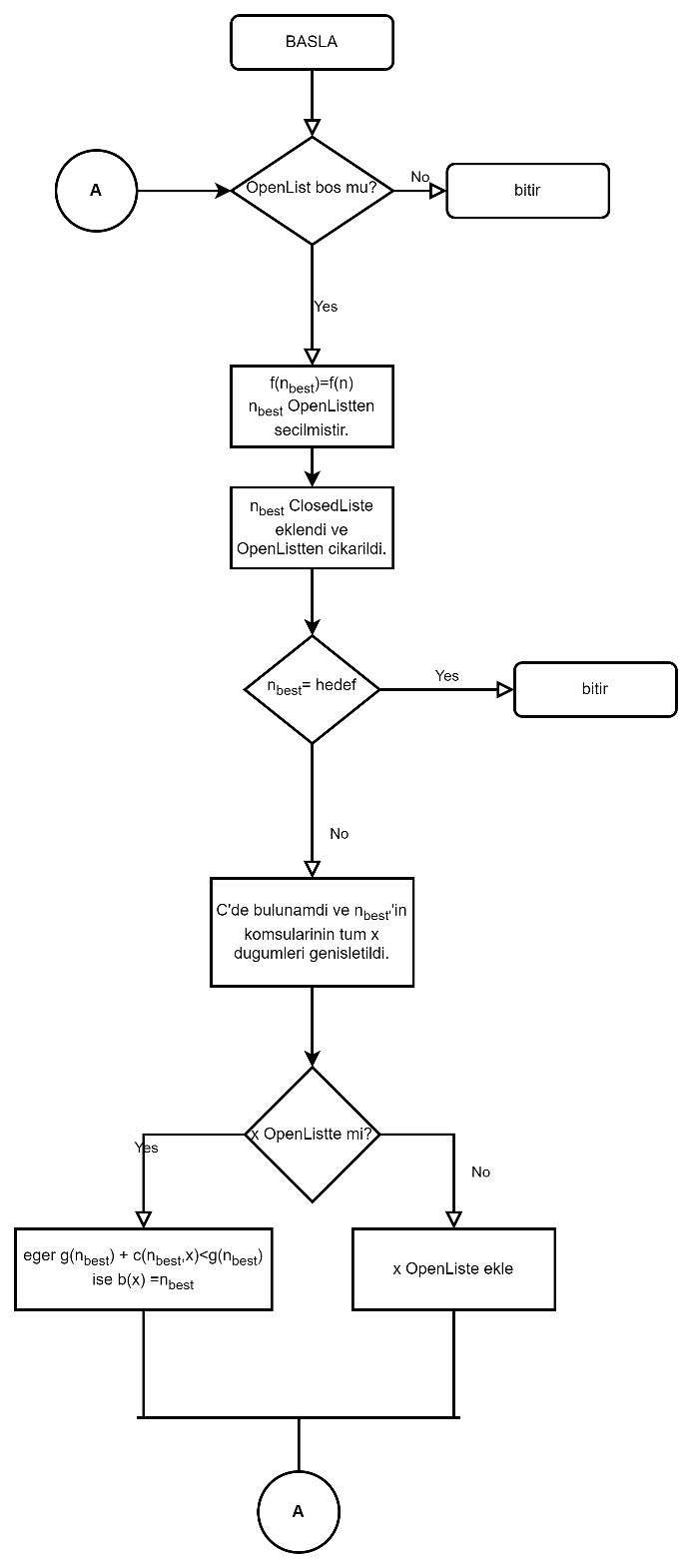
Şimdi bu düğümlerden hedefe giden yol alınır ve ağırlıklar hesaplanır:

f(A-B-D) = (1+ 7) + 0

f(A-C-D) = (5 + 10) + 0

Açıktır ki düğüm B en iyi yolu vermektedir, bu nedenle hedefe ulaşmak için B düğümünü seçmelisiniz.

**A\*(YILDIZ) ALGORİTMASI GENEL AKIŞ SEMASI:**



**A\*(YILDIZ) İLE LABİRENT ÇÖZÜMÜ GENEL ALGORİTMA:**

1. Gerekli kütüphaneler ve renkler içeri aktarılır.

2. Labirentin çizileceği ızgaranın satır sayısı, genişliği ayarlanır.

3. Sınıflar ve gerekli fonksiyonlar oluşturulur.

4. main'de başlangıç (startNode) ve bitiş (endNode) düğümü boş olarak tanımlanır.

5. while döngüsüyle kodun devamlılığı sağlanır.

6. Açılan pencerede izgarayı çizmek için draw() isimli fonksiyon çağırılır.

7. pygame.event.get() fonksiyonuyla pygame etkinlikleri alınır ve bu etkinlikler üzerinde kontrol yapılır.

8. Eğer kullanıcı çıkış yapmak için pencereyi kapattıysa (pygame.QUIT etkinliği), döngüyü sonlandırarak program kapatılır. Değilse

9. Eğer sol fare düğmesine tıklanırsa (pygame.mouse.get\_pressed()[0]), tıklanan konumun ızgara üzerindeki konumu alınır.

10. Eğer başlangıç düğümü (startNode) henüz ayarlanmamışsa ve tıklanan düğüm bitiş düğümü (endNode) değilse, tıklanan düğümü başlangıç düğümü olarak ayarlar ve görsel olarak başlangıç düğümü işaretlenir.

11. Eğer bitiş düğümü (endNode) henüz ayarlanmamışsa ve tıklanan düğüm başlangıç düğümü (startNode) değilse, tıklanan düğümü bitiş düğümü olarak ayarlar ve görsel olarak bitiş düğümü işaretlenir.

12. Eğer tıklanan düğüm ne başlangıç düğümü ne de bitiş düğümü ise, tıklanan düğümü bariyer düğümü olarak ayarlar ve görsel olarak bariyer düğümü işaretlenir.

Değilse buraya yönelir

13. Eğer sağ fare düğmesine tıklanırsa (pygame.mouse.get\_pressed()[2]), tıklanan konumun izgara üzerindeki konumu alınır.

14. İlgili ızgara düğümüne (node) erişir ve düğümü sıfırlar (reset() metodunu çağırır). Böylece düğümün durumu sıfırlanır.

15. Eğer sıfırlanan düğüm başlangıç düğümü (startNode) ise, başlangıç düğümünü None olarak ayarlanır.

16. Eğer sıfırlanan düğüm bitiş düğümü (endNode) ise, bitiş düğümü None olarak ayarlanır.

17. Bir tuşa basılırsa, tuşun hangi tuş olduğu fonksiyon tarafından kontrol edilir.

18. Eğer basılan tuş boşluk tuşu ise, başlangıç düğümü (startNode) ve bitiş düğümü (endNode) ayarlanmışsa, tüm izgara düğümleri için gerekli fonksiyonu çağırarak komşularını günceller ve A\* algoritmasını başlatır (AStarAlgorithm() fonksiyonunu çağırır). [A\* algoritmasının uygulanması ayrıntılı olarak bir başlık altında açıklanmıştır.]

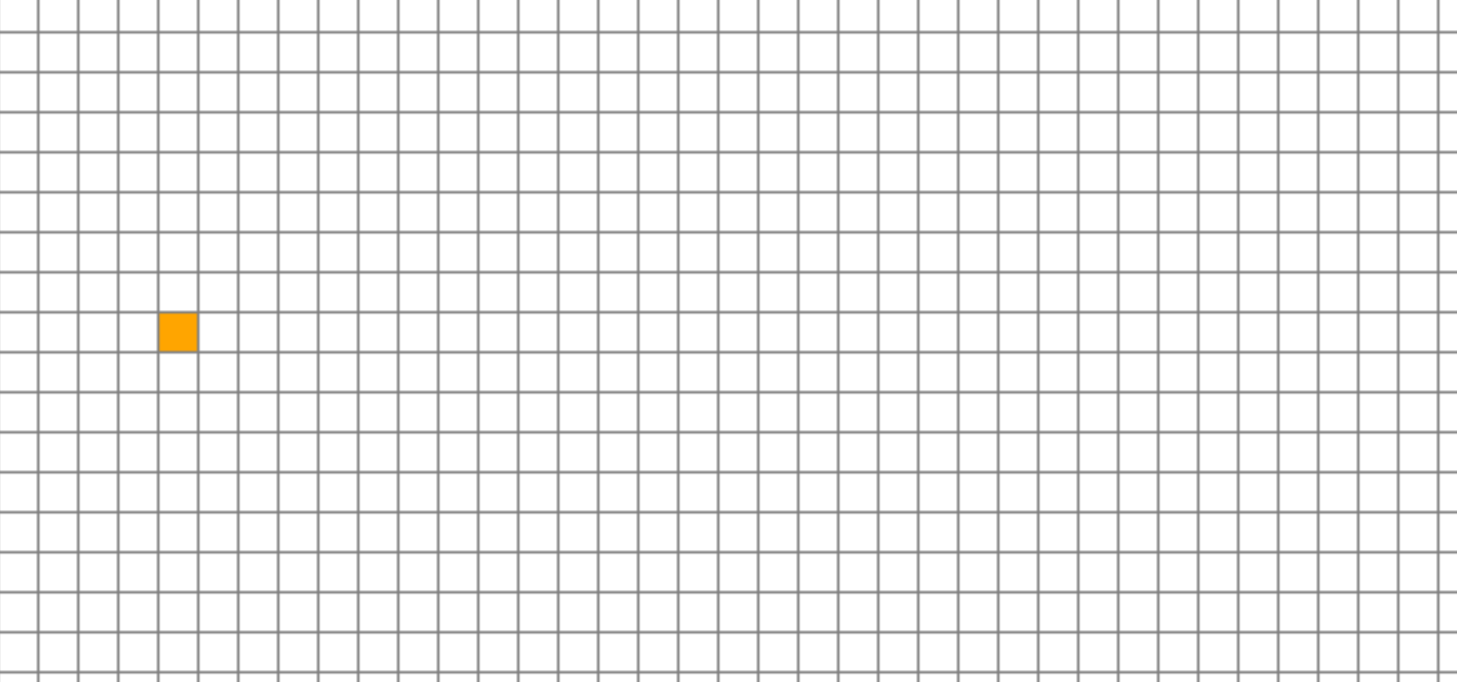
19. Eğer basılan tuş "c" tuşu ise, başlangıç düğümünü ve bitiş düğümünü None olarak ayarlar ve yeni bir izgara oluşturur.

20. Bitiş

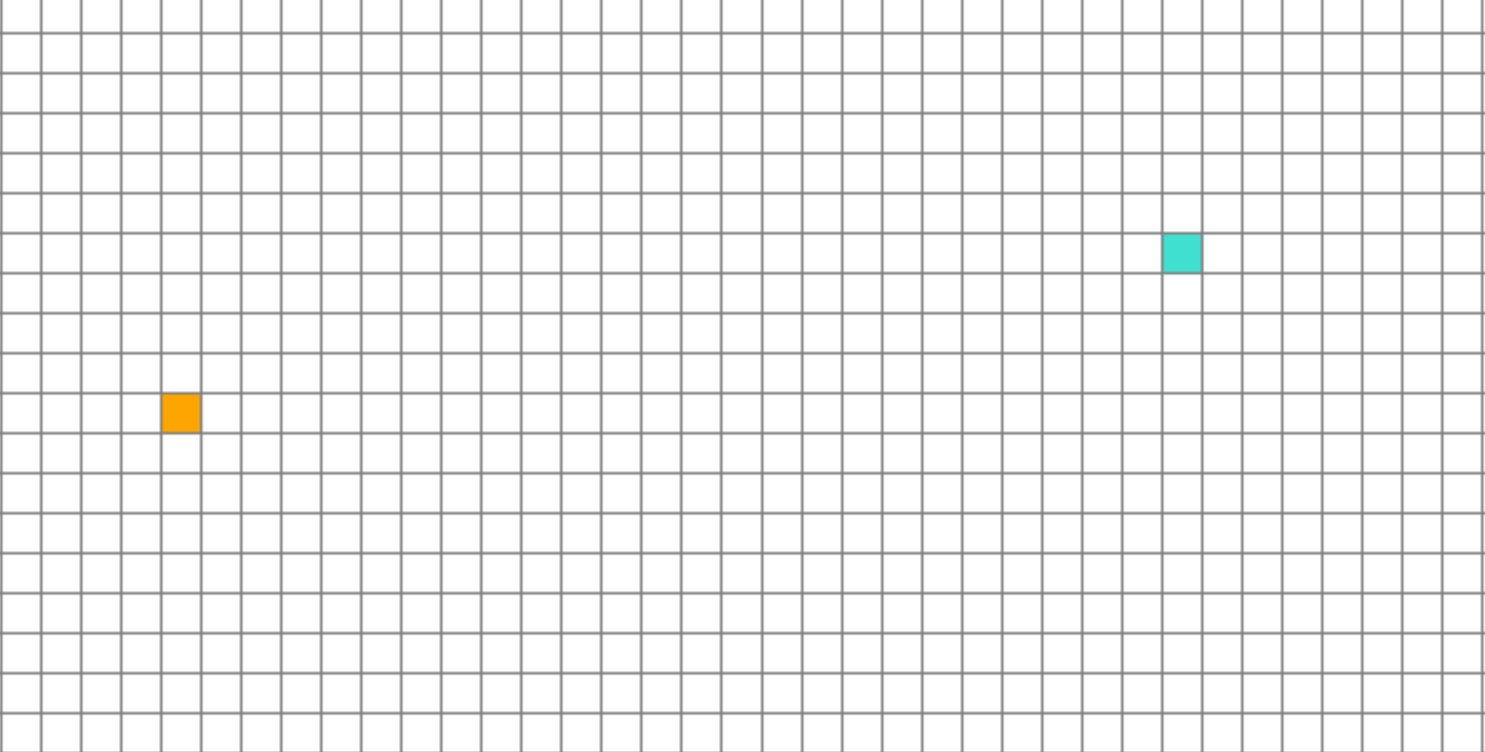
AKIS SEMASI:

**PROJE CIKTILARI:**

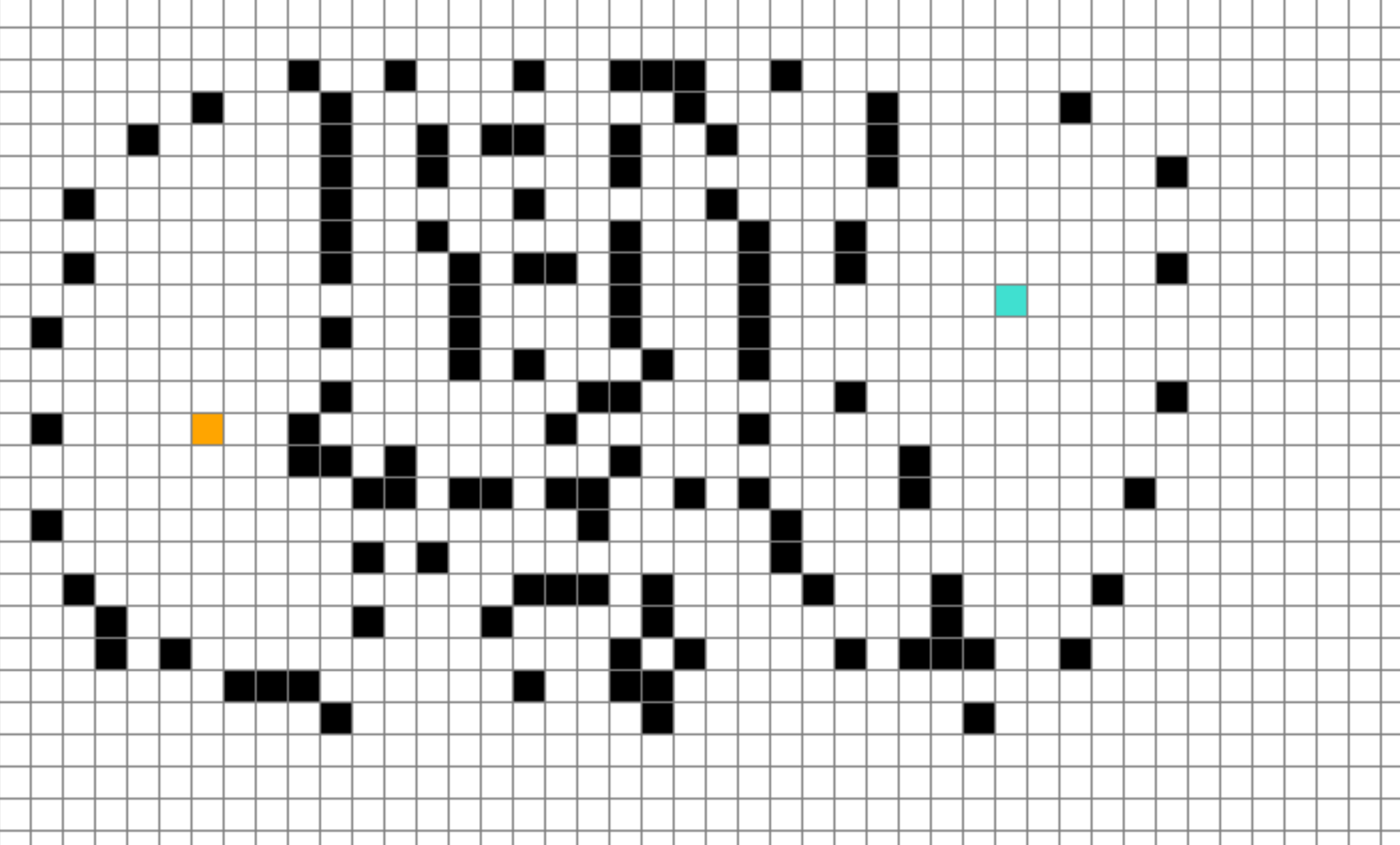
1. Baslangic noktasi belirlendi.



1. Bitis noktasi belirlendi.



1. Duvarlar belirlendi.



**4.** A\* algoritmasi kullanilarak labirent cozuldu.

