**STAR WARS PROJESİ**

*Hacı Bayram ÜNAL*

*180201006*

Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi

[hacibayram.unal@hotmail.com](mailto:hacibayram.unal@hotmail.com)

**Özet**

Bu projenin amacı, veri yapıları ve nesneye yönelik programlama konularının gerçekleştirilmesidir. Konuların uygulanması ve uygulamanın anlaşılması açısından, kullanıcı tarafından hareket ettirilen karakterin, labirent içerisinde yer alan düşman karakterlere yakalanmadan çıkışa ulaşması hedeflenmektedir. Çıkışa ulaşmaya çalışırken, düşman karakterler, oyuncu karakterine bulundukları konumdan itibaren en yakın konumu bularak (shortest path algorithm) ilerleyecekler ve her adımda bu bilgiyi güncelleyeceklerdir.

Proje Java programlama dili kullanılarak geliştirilmiştir. Program geliştirilirken, düşman karakterlerin kullanıcı karakterine ulaşması için izlemesi gereken en kısa yolu belirlemek için BFS (Breadth First Search)[1] algoritması temel alınmıştır. Her karakterin labirent üzerindeki hareketi karakter koordinatları güncellenerek sağlanır. Bu koordinatlar kullanılarak düşman karakter ile oyuncu karakter arasındaki en kısa yol bulmaya çalışılır ve düşman karakterin izlemesi gereken koordinatlar sırasıyla x ve y değerleri girilerek ArrayList yapısında tutulur. En kısa yol hesaplaması yapılırken de ne kadar süre geçtiği hesaplanıp bir değişkende tutulur.

Bu bilgilerin elde edilmesinin ardından, Java’nın grafik kütüphanesi Swing[2] kullanılarak bu veriler işlenir ve görsel bir arayüz sunulur, ardından programın kullanıcı tarafından kullanılması için ilgili tuşlara gerekli komutlar atanır.

**Anahtar kelimeler :** BFS, ArrayList, Swing, shortest-path-algorithms

# Giriş

Bu projede, ‘Harita.txt’ belgesindeki belirli düzene göre yazılmış karakter bilgilerinin dosya okuma işlemleri ile programatik olarak parçalara ayrılıp, bu parçaların düzeni ve içerdikleri bilgilere göre karakterlerin oluşturulması sağlanmıştır.

Projenin gerçekleştirilmesinde kullanılan senaryoda, düşman karakterler ve kullanıcının yönlendirebileceği iyi karakter bulunmaktadır. Bu karakterler ‘Karakter’ sınıfından türemektedir. Her karakterin bir koordinat bilgisi olması gerektiği için ‘Karakter’ sınıfı da, bu lokasyon bilgilerinin kalıtım yoluyla kendisine aktarıldığı ‘Lokasyon’ sınıfından türemektedir.

‘Harita.txt’ belgesinden veriler okunduktan sonra karakterler oluşturulur ve kullanıcıyla etkileşimde bulunmak ve daha anlaşılır bir yapı sunmak adına Java’nın grafik kütüphanesinden (Swing) yararlanılmıştır. ‘Harita.txt’ belgesinden okunan verilere göre karakterlerin varsayılan başlangıç x ve y koordinat değerleri atanır.

En kısa yol algoritmasını bulmak için BFS (Breadth First Search) algoritması temel alınmıştır. Bu algoritmanın işlenmesi sonucu elde edilen en kısa yol değerlerine göre ilgili düşman karakterden oyuncunun kontrol ettiği iyi karaktere giden en kısa yol renklendirilerek kullanıcı tarafından görülmesi sağlanır. Kullanıcı, iyi karakteri her hareket ettirdiğinde, kötü karakter de bu harekete karşılık karakterin kendi özelliğine göre bir birim hareket edecek ve bu hareketinden sonra iyi karaktere olan en kısa yol yeniden hesaplanıp grafik ekranında renklendirilerek gösterilecektir.

# Yöntem

Bu projede, ‘Harita.txt’ belgesindeki belirli düzene göre yazılmış karakter bilgilerinin dosya okuma işlemleri[3] ile programatik olarak parçalara ayrılıp, bu parçaların düzeni ve içerdikleri bilgilere göre karakterlerin oluşturulması sağlanmıştır.

Bu düzene göre program tarafında ‘,’ karakteri ile ayırma işlemi (split)[4] yapıldığında sıfırıncı indekste karakter bilgileri, birinci indekste kapı bilgileri yer almaktadır.

Ayrıştırılan bilgiler de kendi içlerinde ‘:’ karakteri ile ayrıldıkları için bu karakter baz alınarak ayrı bir ayırma işlemi daha yapılmaktadır. Bu ayırma işleminde sıfırıncı indeksten yeni oluşturulan iki elemanlı dizinin birinci indeksinde karakter ismi, ikinci indeksten yeni oluşturulan iki elemanlı dizinin birinci indeksinde ise kapı bilgisi yer almaktadır.

Bu bilgiler doğrultusunda karakterin ismine göre yeni bir nesne oluşturulur, kapı bilgisine göre de oluşturulan karakter nesnesine konum bilgisi ataması yapılır. Ardından bu nesneler bir dizide toplanır.

**Sözde kod :**

FileInputStream fr = new FileInputStream(“dosya yolu”);

InputStream ir = new InputStream(fr);

BufferedReader br = new BuferedReader(ir);

String line;

String[] split, split2;

While (line = br.readLine()) null olmadığı sürece

Eğer line (“,”) içeriyorsa

Split = line.split(“,”)

int i = 0

i = 0; i < split.length -1 olduğu sürece

split2 = split[i].split(“:”)

Karakter krk = karakterOlustur(split2[1])

Eğer krk null değilse

Eğer split2[1].equals(“A”) ise

Krk.setDeafultX(0)

Krk.setDefaultY(5)

Krk.setX(0)

Krk.setY(5)

Değilse eğer split2[1].equals(“B”) ise

Krk.setDefaultX(4)

Krk.setDeafultY(0)

Krk.setX(4)

Krk.setY(0)

Değilse eğer split2[1].equals(“C”) ise

Krk.setDefaultX(12)

Krk.setDefaultY(0)

Krk.setX(12)

Krk.setY(0)

Değilse eğer split2[1].equals(“D”) ise

Krk.setDefaultX(13)

Krk.setDefaultY(5)

Krk.setX(13)

Krk.setY(5)

Değilse eğer split2[1].equals(“E”) ise

Krk.setDefaultX(4)

Krk.setDefaultY(10)

Krk.setX(4)

Krk.setY(10)

Eğer okunan satır bilgisinde kapı bilgisi bulunmuyorsa, bu bilgi kullanıcının etkileşimde bulunacağı karakter anlamına gelmektedir. Bu karakterin ismine uygun olan nesne oluşturulur ve koordinat bilgisi ataması yapılır.

Okunan karakter bilgisindeki karakter oluşturma işlemi bir metot aracılığı ile gerçekleştirilmektedir. Bu metot String veri tipinde bir parametre alır. Bu parametreye göre karakter ismine uyuşan koşulda bir karakter nesnesi oluşturur ve geri döndürür.

**Sözde kod :**

Değilse eğer line.contains(“:”) ve line.startsWith(“Karakter”)

String[] oyuncuKarakteri = line.split(":")

Karakter krk = karakterOlustur(oyuncuKarakteri[1])

krk.setDefaultX(6)

krk.setDefaultY(5)

krk.setX(6)

krk.setY(5)

g.setOyuncuKarakteri(krk)

void karakterOlustur(String karakterAdi) {

Karakter karakterNesnesi

Eğer karakterAdi.equals(“Stromtrooper”) ise

karakterNesnesi = new Stormtrooper()

Değilse eğer karakterAdi.equals(“MasterYoda”) ise

karakterNesnesi = new MasterYoda()

Değilse eğer karakterAdi.equals(“LukeSkywalker”) ise

karakterNesnesi = new LukeSkywalker()

Değilse eğer karakterAdi.equals(“KyloRen”) ise

karakterNesnesi = new KyloRen()

Değilse eğer karakterAdi.equals(“DarthVader”) ise

karakterNesnesi = new DarthVader()

return karakterNesnesi

}

Karakterlerin hareketlerini ifade etmek, güncellemek ve grafik ekranında gösterebilmek için x ve y değerleri tutulması gerekmektedir. Bu özellikler ‘Lokasyon’ isimli bir sınıftan ‘Karakter’ sınıfına kalıtım olarak aktarılmaktadır.

Projenin senaryosuna göre farklı karakterler bulunmaktadır. Bu karakterlerin lokasyon bilgilerinin işlenmesi üzere lokasyon isimli sınıf türetilip karakter sınıfına kalıtılmıştır. Her karakterin koordinat bilgisinin yanı sıra diğer bilgilerine de ihtiyaç duyulmaktadır. Bu bilgileri ad, tür, can ve en kısa yolu her hesapladığında üzerinde işlem yapılması için ‘enKısaYol’ isminde bir dinamik liste yapısı bulunmaktadır.

Kötü karakterin, iyi karakteri yakalaması sonucunda karakterin canı azalacak ve oyun baştan başlamak üzere karakterler eski yerlerini alması gerekeceği için, ‘Lokasyon’ sınıfına karakterlerin başlangıçtaki konumlarını tutmak için ‘defaultX’ ve ‘defaultY’ isimli iki adet integer değişken tanımlanmıştır. Bu değişkenlere karakterlerin başladığı konumlar atanır ve üzerinde bir işlem yapılmaz. Karakterin hareketi için sadece ‘x’ ve ‘y’ isimli değişkenler kullanılmaktadır.

Kötü karakterlerin iyi karaktere ulaşmak için en kısa yolu hesaplaması gerekirken, iyi karakterin böyle bir işi yapmasına gerek yoktur. Bu nedenle karakter sınıfında tanımlanan metot, en kısa yolu hesaplaması istenen sınıflarda ‘override’ edilerek iyi karaktere giden en kısa yolu hesaplaması sağlanmıştır.

En kısa yolun hesaplanmasında kullanacak metot bazı parametrelere ihtiyaç duymaktadır. Bunlar; kötü karakterin konumu, iyi karakterin konumu ve karakter sınıfı tipindeki en kısa yolun hesaplanması istenen karakterin nesnesidir. Karakter nesnesine, karakterlerin türlerine göre hareketlerinin farklı olduğu için en kısa yolu bulurken nasıl davranması gerektiğini belirlemek için ihtiyaç duyulmaktadır.

Karakter en kısa yolu hesaplarken ne kadar süre geçtiğini tutmak için de ayrıca bir değişkene ihtiyaç duyulmaktadır. Bu işlem sonucunda ondalıklı sayı elde edildiği için bu değişkenin türü double olarak tanımlanmıştır.

**Sözde kod :**

public class Lokasyon {

protected int x, y

protected int defaultX, defaultY

}

public class Karakter extends Lokasyon {

protected String ad, tur

protected int can

protected ArrayList<Integer> enKisaYol

protected double yolBulmaSuresi

public void enKisaYol(int kotuX, int kotuY, int hedefX, int hedefY, Karakter k) { }

}

‘Harita.txt’ belgesinden okunan labirent bilgisine göre, projede bu bilgi ‘0’ rakamları ziyaret edilmiş, ‘1’ rakamları ziyaret edilmemiş olarak işlendi. Bu bilgiye göre en kısa yolu hesaplayacak olan karakterin sınıfında, yayılım öncelikli algoritmayı gerçekleştirmek için yardımcı bir sınıf kullanıldı.

Bu sınıf, en kısa yolu hesaplayacak olan karakterin koordinatlarını alır, kendisine ‘sıfır’ başlangıç seviyesini verir ve bu koordinatlara göre ‘Harita.txt’ den okunan labirent bilgisine göre, çevresinde ziyaret edilmemiş noktaları kendi alt düğümü olarak dinamik liste yapısına ekler, buraları ziyaret edildi olarak işaretler ve düğümlerin seviyesini arttırır. Ziyaret edilme bilgisi ‘Harita.txt’ belgesinden okunan ‘1’ ve ‘0’ rakamlarına göre ‘boolean’ matrisi oluşturulur, ziyaret edilen noktalar güncellenerek alt düğümlere aktarılır. Bir sonraki ilerleme işlemi en son gelinen ‘seviyenin’ bir arttırılması üzerinden devam eder. Bu işlem alt düğümlerin de alt düğümlerine uygulanarak iyi karaktere ulaşılana kadar devam etmektedir. İyi karaktere ulaşıldığı zaman, ulaşan düğümün üst düğümlerinin de üzerinden başlangıç konumuna varana dek önceki konumlarına geri döner. Böylece ilk ulaşan düğüm en kısa yolu vermiş olacaktır.

**Sözde kod :**

public class Node {

protected int x, y;

protected ArrayList<Node> altDugumler;

protected Node önceki;

protected boolean ziyaretEdildiMi = false;

int seviye;

boolean seviyeAramasi = false;

protected boolean[][] ziyaretEdilen;

}

En kısa yolu bulurken ilgili karakterin algoritmayı gerçekleştirme süresini de öğrenmek için Java’nın ‘Time Unit’[5] kütüphanesi kullanılmıştır. Bu kütüphanenin sağladığı komutların arasına çalışma süresi hesaplanması istenen komutlar yerleştirilir, hesaplanan süre ‘nanosaniye’ cinsindendir. Nanosaniye ‘1.000.000’ e bölündüğnde ‘milisaniye’, ‘1.000.000.000’ e bölündüğünde ‘saniye’ cinsinden bir ifade elde etmemiz mümkün olacaktır. Bu projede algoritmanın çalışma süresi ‘milisaniye’ cinsinden ifade edilmiştir.

**Sözde kod :**

long startTime = System.nanoTime()

. . . // çalışması istenen komutlar

. . . // çalışması istenen komutlar

long endTime = System.nanoTime()

long totalTime = endTime - startTime

this.yolBulmaSuresi = totalTime / 1000000.0;

Mevcut ele alınan karakter için en kısa yol algoritmasına giden parametreler doğrultusunda projede belirtilen veri yapısını gerçekleştirmek için ‘Node’ sınıfı tipinde bir nesne oluşturulur ve bu nesneye ilgili karakterin koordinat bilgisi aktarılır. Bu düğüm kendisinde hazır bulunan ziyaret listesine göre bulunduğu koordinattan sağına, soluna, arkasına ve ilerisine bakarak ziyaret edilmemiş noktaların koordinatlarını yeni düğümler oluşturup bu düğümlere atama yapar, düğümün koordinatını ziyaret listesinde ziyaret edildi olarak işaretler ve ve söz konusu düğümleri alt düğüm olarak ekler. Bu işlemi ‘ziyaretEdilmemisBul’ isimli metot aracılığı ile yapmaktadır. Bu metot ulaşılması istenen hedefin koordinatlarını ve söz konusu düğümü parametre olarak almaktadır.

**Sözde kod :**

public Node ziyaretEdilmemisBul(int hedefX, int hedefY, Node parent) {

boolean sol, sag, yukari, asagi

sol = sag = asagi = yukari = false

Eğer parent.x == hedefX ve parent.y == hedefY ise

return parent

Eğer parent.ziyaretEdilen[parent.x +1][parent.y] == false ise

sag = true

parent.ziyaretEdilen[parent.x+1][parent.y] = true

Eğer parent.ziyaretEdilen[parent.x – 1][parent.y] == false ise

sol = true

parent.ziyaretEdilen[parent.x-1][parent.y] = true

Eğer parent.ziyaretEdilen[parent.x][parent.y+1] == false ise

asagi = true

parent.ziyaretEdilen[parent.x][parent.y+1] = true

Eğer parent.ziyaretEdilen[parent.x][parent.y-1] == false ise

yukari = true

parent.ziyaretEdilen[parent.x][parent.y-1] = true

parent.altDugumler = new ArrayList<Integer>()

Eğer yukari == true ise

Node nodeYukari = new Node()

nodeYukari.x = parent.x

nodeYukari.y = parent.y

nodeYukari.onceki = parent

nodeyukari.seviye = parent.onceki.seviye + 1

nodeYukari.ziyaretEdilen=parent.ziyaretEdilen

parent.altDugumer.add(nodeYukari)

Eğer asagi == true ise

Node nodeAsagi = new Node()

nodeAsagi.x = parent.x

nodeAsagi.y = parent.y

nodeAsagi.onceki = parent

nodeAsagi.seviye = parent.onceki.seviye + 1

nodeAsagi.ziyaretEdilen = parent.ziyaretEdilen

parent.altDugumer.add(nodeAsagi)

Eğer sol == true ise

Node nodeSol = new Node()

nodeSol.x = parent.x

nodeSol.y = parent.y

nodeSol.onceki = parent

nodeSol.seviye = parent.onceki.seviye + 1

nodeSol.ziyaretEdilen = parent.ziyaretEdilen

parent.altDugumer.add(nodeSol)

Eğer sag == true ise

Node nodeSag = new Node()

nodeSag.x = parent.x

nodeSag.y = parent.y

nodeSag.onceki = parent

nodeSag.seviye = parent.onceki.seviye + 1

nodeSag.ziyaretEdilen = parent.ziyaretEdilen

parent.altDugumer.add(nodeSag)

}

İlk düğüm için bu işlemler uygulandıktan sonra ilk düğümün de alt düğümlerine aynı işlemleri uygulamak üzere, hedefe ulaşana kadar devam eden ve alt düğümlerin hepsini de kapsayacak bir döngü aracılığı ile ziyaret edilen noktalar arttırılacaktır. Mevcut ele alınan düğümün alt düğümlerine ulaşabilmek için güncel tutulan seviye sayısına göre bulunan düğümler bir listede toplanır ve bu düğümlere aynı işlemler uygulanmaya devam edilir.

Güncel seviye sayısına göre düğümleri getirmek için ‘seviyeyeGoreGetir’ isimli bir metot kullanılmıştır. Bu metot mevcut düğümü, bulunması istenen düğümlerin seviyesini ve içi doldurulmak üzere gönderilen bir dinamik liste yapısını parametre olarak almaktadır. Kök düğümden başlayarak bütün düğümleri dolaşır ve kendisine parametre olarak gelen seviyeye uygun olan düğümleri listeye ekler ve en sonunda listeyi geri döndürür.

public ArrayList<Node> seviyeyeGoreGetir(Node dugum, int seviye, ArrayList<Node> liste) {

Eğer dugum.altdugumler != null

Node geçici = dugum.altDugumler.get(0)

int sayac = 0

while geçici düğüme eşit olmayana kadar devam et

Eğer sayaç > 0

Eğer geçici,dugum.altDugumler.get(0)’a eşitse

boolean ziyaretEdilmeyenKaldiMi = false;

int i = 0;

i < gecici. altDugumler.size olduğu sürece

Eğer gecici.altDugumler.get(i).seviyeAramasi= false ise

Gecici. altDugumler.get(i).seviyeAramasi = true;

ziyaretEdilenler.add(temp.altDugumler.get(i))

gecici = gecici.altDugumler.get(i)

ziyaretEdilmemisKaldiMi = true;

break;

i’yi bir arttır başa dön

Eğer ziyaretEdilmemisKaldiMi = false ise

Gecici = gecici.onceki

int i = 0

i = 0; i < temp.altDugumler.size olduğu sürece

Eğer gecici.altDugumler.get(i).seviyeAramasi = false ise

Gecici. altDugumler.get(i).seviyeAramasi = true;

ziyaretEdilenler.add(temp.altDugumler.get(i))

gecici = gecici.altDugumler.get(i)

ziyaretEdilmemisKaldiMi = true

break;

i’yi bir arttır başa dön

Eğer gecici.seviye == seviye ise

Liste.add(gecici)

Gecici.seviyeAramasi = true

ziyaretEdilenler.add(gecici)

Eğer gecici.altDugumler == null ise

Eğer gecici.altDugumler.get(0).seviyeAramasi =false ise

Gecici = gecici.altDugumler.get(0)

Gecici.seviyeAramasi = true

Değilse

boolean ziyaretEdilmeyenKaldiMi = false;

int i = 0

i < gecici. altDugumler.size olduğu sürece

Eğer gecici.altDugumler.get(i).seviyeAramasi = false ise

Gecici. altDugumler.get(i).seviyeAramasi = true

ziyaretEdilenler.add(temp.altDugumler.get(i))

gecici = gecici.altDugumler.get(i)

ziyaretEdilmemisKaldiMi = true

break

i’yi bir arttır başa dön

Eğer ziyaretEdilmemisKaldiMi = false ise

Gecici = gecici.onceki

int i = 0

i = 0; i < temp.altDugumler.size olduğu sürece

Eğer gecici.altDugumler.get(i).seviyeAramasi = false ise

Gecici. altDugumler.get(i).seviyeAramasi = true

ziyaretEdilenler.add(temp.altDugumler.get(i))

gecici = gecici.altDugumler.get(i)

ziyaretEdilmemisKaldiMi = true

break

i’yi bir arttır başa dön

ziyaretEdilenler.add(gecici)

Değilse eğer gecici.altDugumler == null ise

Gecici = gecici.onceki

Boolean bulunduMu = false

int i = 0;

i = 0; i < temp.altDugumler.size olduğu sürece

Eğer gecici.altDugumler.get(i).seviyeAramasi = false ise

Gecici. altDugumler.get(i).seviyeAramasi = true

ziyaretEdilenler.add(temp.altDugumler.get(i))

gecici = gecici.altDugumler.get(i)

bulunduMu = true

break

i’yi bir arttır başa dön

Eğer bulunduMu == false ise

Eğer gecici == dugum ise

Break

Değilse

Gecici = gecici.onceki;

Boolean bulunduMu = false

int i = 0

i = 0; i < temp.altDugumler.size olduğu sürece

Eğer gecici.altDugumler.get(i).seviyeAramasi = false ise

Gecici. altDugumler.get(i).seviyeAramasi = true

ziyaretEdilenler.add(temp.altDugumler.get(i))

gecici = gecici.altDugumler.get(i)

bulunduMu = true

break;

i’yi bir arttır başa dön

Eğer ziyaretEdilmemisVarMi = false ise

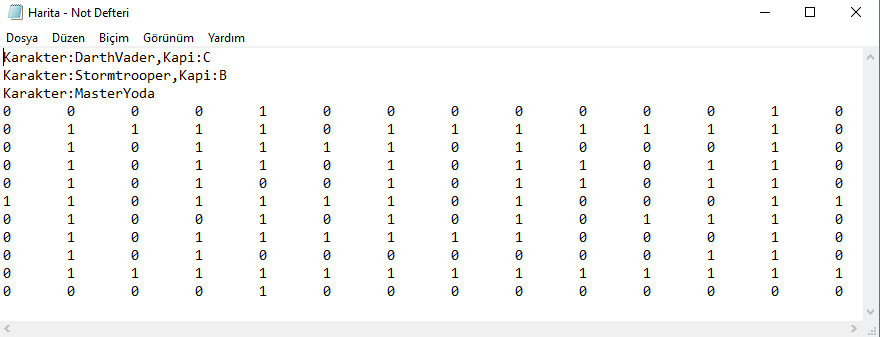
break

sayac++

}

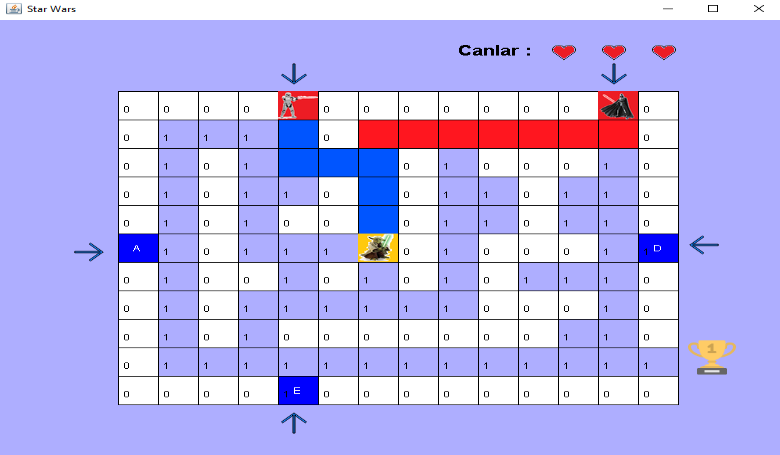
# Deneysel sonuçlar

1. Örnek ‘Harita.txt’ dosyası



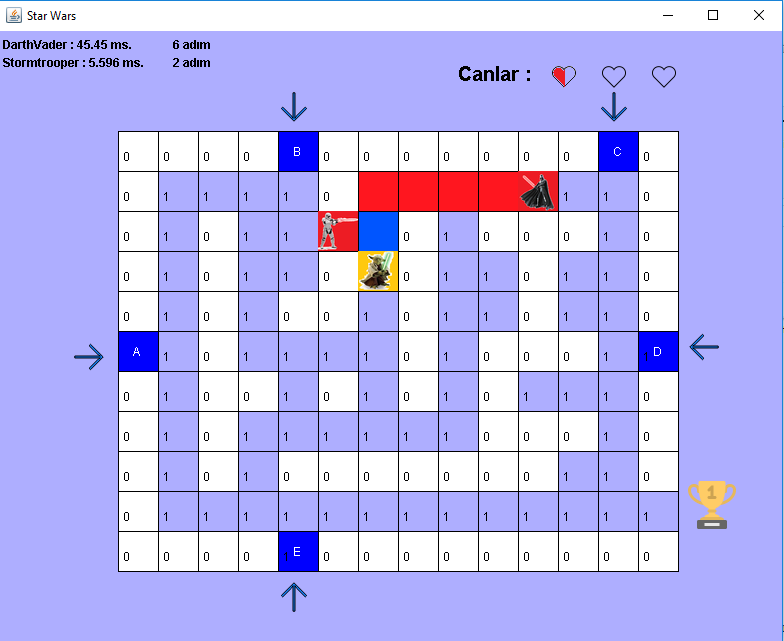
*Şekil 1 Örnek ‘Harita.txt’ dosyasının görüntüsü*

1. Örnek ‘Harita.txt’ dosyasına göre grafik ekranı başlangıcı



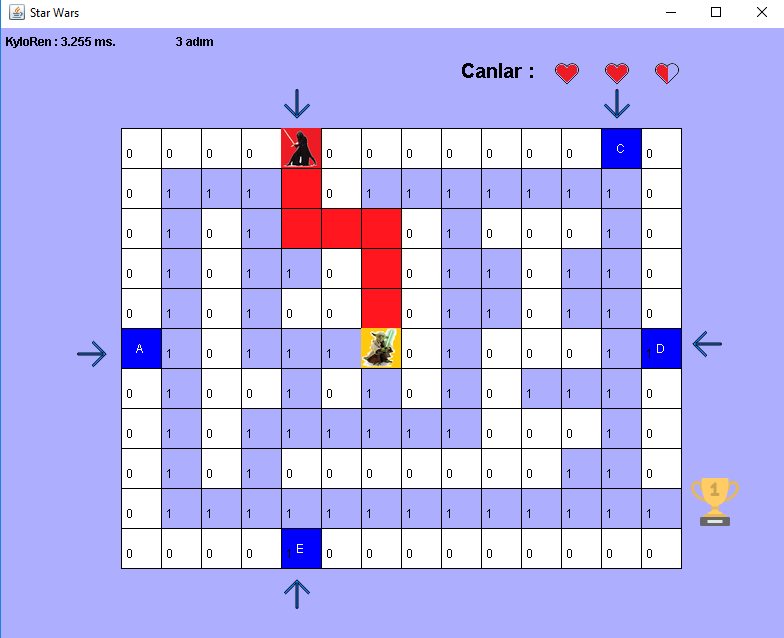
*Şekil 2 Örnek ‘Harita.txt’ dosyasına göre grafik ekranı başlangıcı*

1. Düşman karakterlere yakalanmadan önceki pozisyon



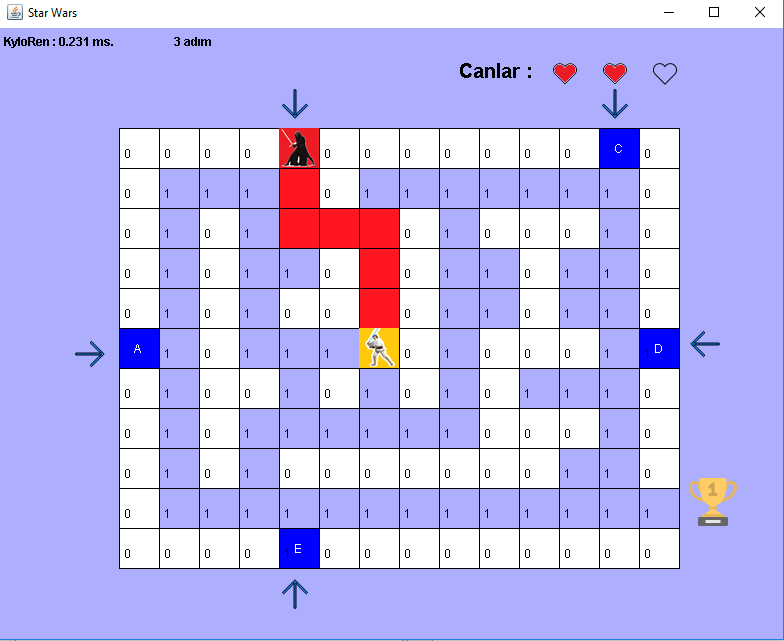
*Şekil 3 Oyuncu karakterinin düşman karaktere yakalanmadan önceki durumu*

1. Usta Yoda’nın düşman karaktere yakalandıktan sonraki can azalışı



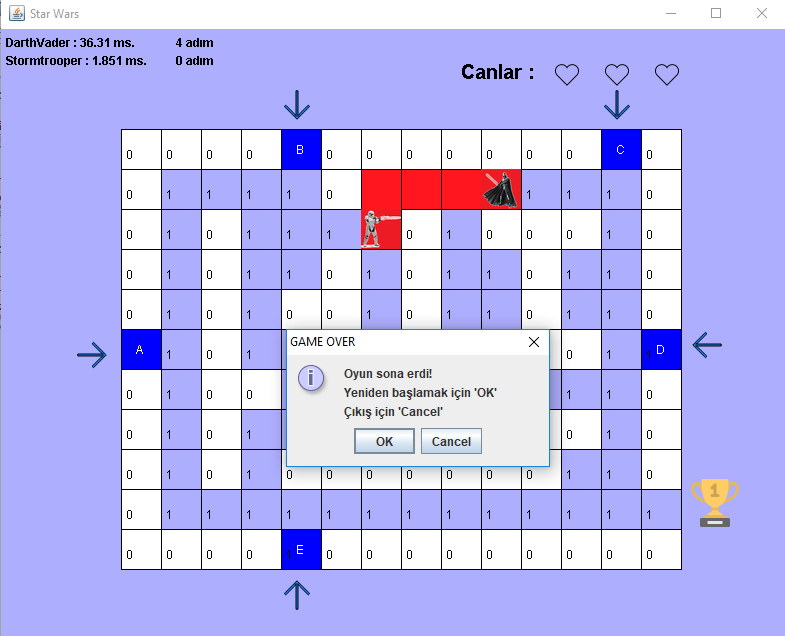
*Şekil 4 Usta Yoda’nın düşman karaktere yakalanması sonucu can azalışının gösterimi*

1. Luke Skywalker’ın düşman karaktere yakalandıktan sonraki can azalışı



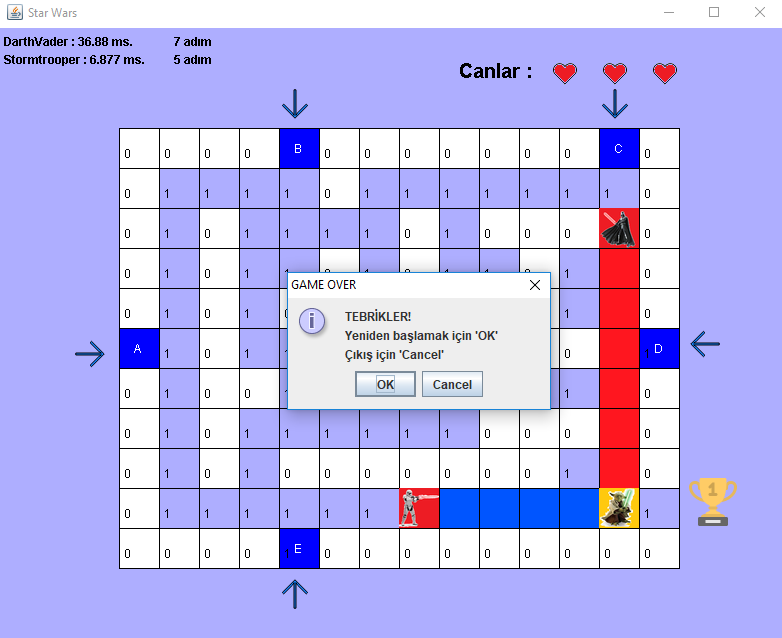
*Şekil 5 Luke Skywalker’ın düşman karaktere yakalandıktan sonraki can azalşının gösterimi*

1. Oyuncu karakterinin yakalanması sonucu canlarının tükenmesi



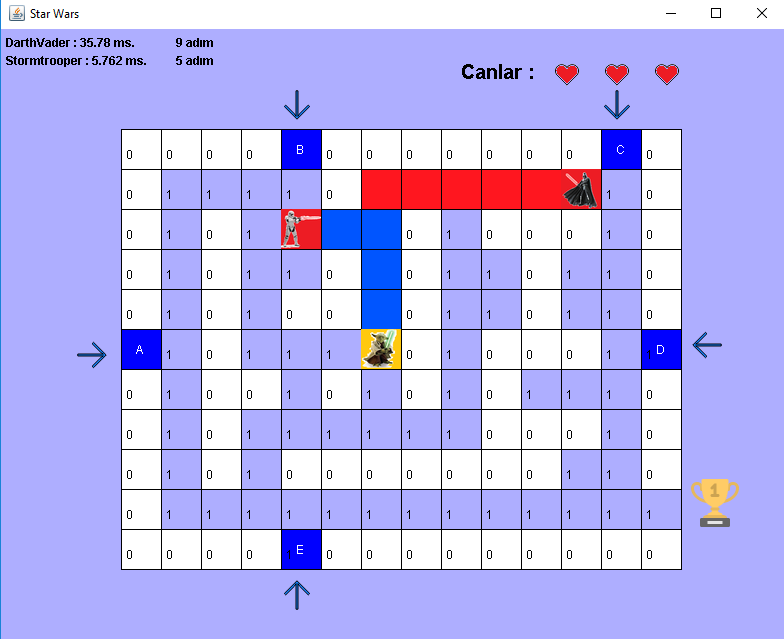
*Şekil 6 Oyuncu karakterinin yakalanması sonucu canlarının tükenmesinin gösterimi*

1. Oyuncu karakterinin çıkışa ulaşması



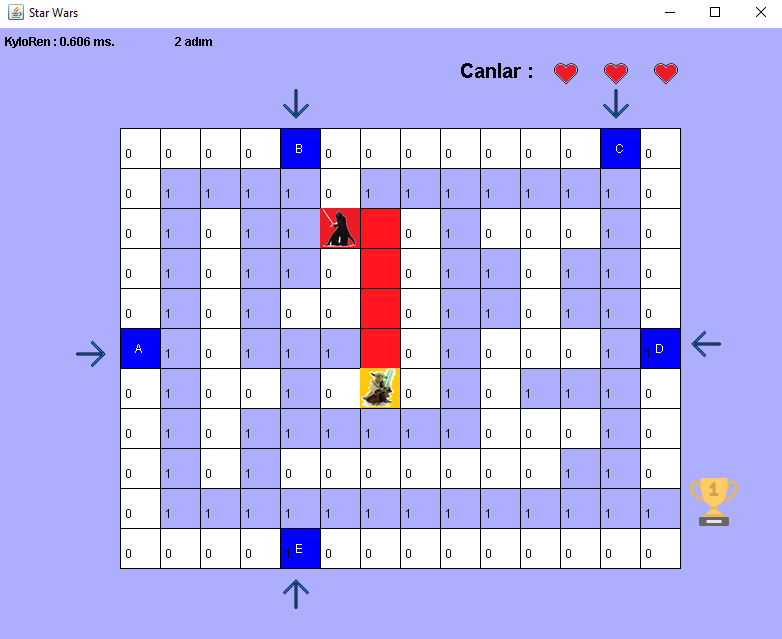
*Şekil 7 Oyuncu karakterinin düşman karakterlere yakalanmadan çıkışa ulaşması durumu*

1. Oyuncu karakterinin engele doğru hareket etmek istemesi



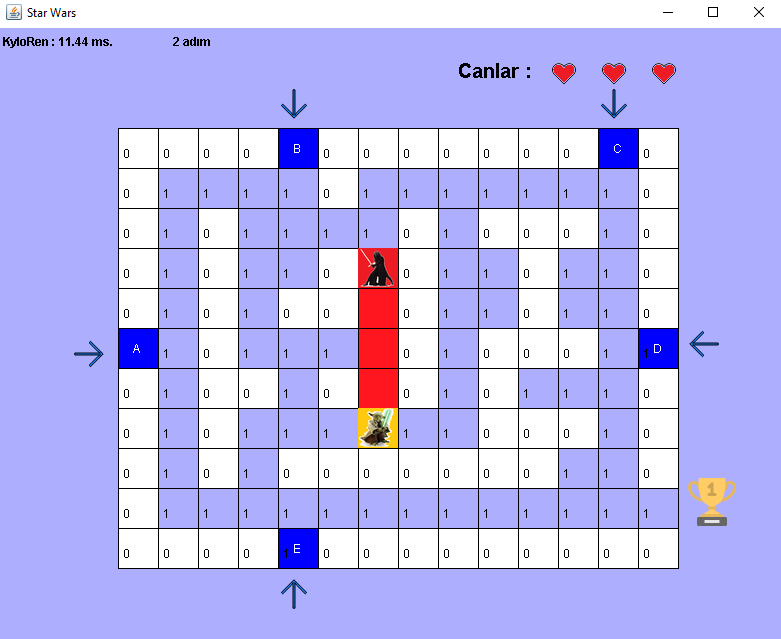
*Şekil 8 Oyuncu karakterinin engele doğru hareket etmek istemesi ve düşman karakterlerin bir birim hareket etmeleri*

1. Düşman birim olarak Kylo Ren’in kullanılması



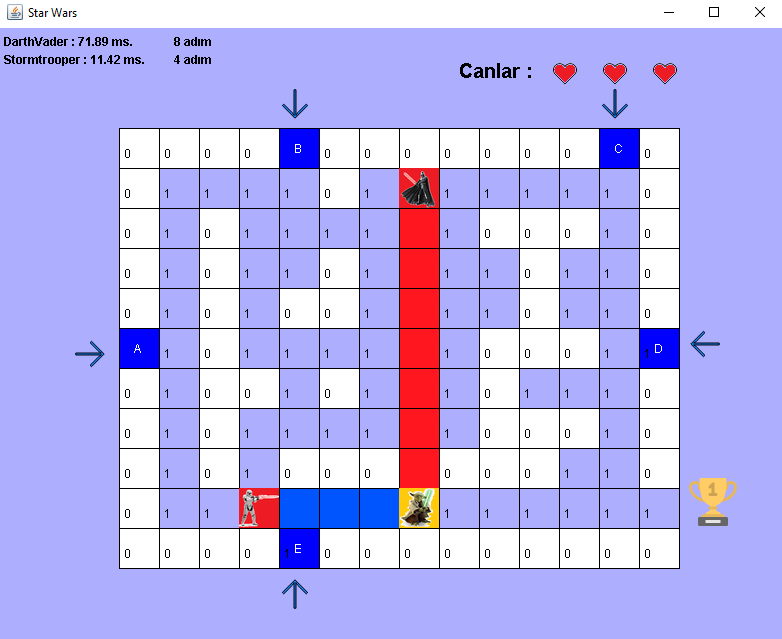
*Şekil 9 Düşman karakter Kylo Ren’in hareket etmeden önce*

1. Düşman karakter Kylo Ren’in hareketi



*Şekil 10 Düşman karakter Kylo Ren’in hareketini iki birim olarak gerçekleştirmesi*

1. Düşman karakter olarak Darth Vader’in kullanılması



*Şekil 11 Düşman karakter Darth Vader’in hareketini engellerin üzerinden geçerek gerçekleştirebilmesinin gösterimi*

1. **Karmaşıklık Analizi**

Projede kötü karakterlerin iyi karaktere doğru en kısa yolu bulurken kullandığı algoritmanın karmaşıklığına bakıldığında, en kısa yol metodunda hesaplamanın sürekli olarak işlediği döngü toplamda 154 kere çalışacaktır. Bunun nedeni labirentin 11x14 boyutlarında olmasıdır fakat bu sayı farklı labirentler hesaba katıldığında değişken olabileceği için ‘N’ defa olarak ifade edilmesi gerekmektedir.

‘N’ defa dönen kod parçasının içerisinde çeşitli metodlar çağırılmaktadır. Bu metotlar nesnelerin ziyaret edilme değerlerini sıfırlayan seviye arama sıfırlama metodu, seviye numarasına göre düğümleri liste olarak döndüren metot ve ziyaret edilmemiş düğümleri bulma metotlarıdır.

‘N’ defa çalışan döngü içerisinde seviye aramasını sıfırlama metodu çalıştığı için ve bu metot da kendisine gelen listedeki elemanların sayısı kadar bir döngüyü çalıştırıp ziyaret seviye arama değişkeni ve ziyaret edilme değerini ‘false’ olarak değiştirdikleri için buradan ‘N2+2’ karmaşıklık gelmektedir. Ardından algoritmada geçici bir düğüm değişkeni, bir liste ve metottan dönen değeri değişkene atama yaptığı için döngü her döndüğünde yeni bir atama yapıldığı ve döngü de ‘N’ defa çalıştığı için karmaşıklığa ‘+3N’ kadar bir karmaşıklık daha eklenmektedir.

Döngü içerisinde seviyeye göre arama yapılan metot çağırılır. Bu metodun karmaşıklığına bakılacak olursa; ilk adımda parametre olarak gelen düğümün alt düğümlerinin olup olmadığı kontrol edilir ve buradan ‘+1’ maliyet eklenileceği düşünülür. Bu kodun ‘N’ defa çalışacağını düşünürsek ‘+N’ eklenmelidir. En kötü ihtimalde bu kod çalışacağı için hemen alt satırlarında iki adet atama işlemi bulunmaktadır. Buradan da ‘+2’ maliyet eklenmesi gerekir fakat bu kodun da ‘N’ defa çalışacağını düşündüğümüzde ‘+2N’ maliyet eklenecektir. Bunun devamında bir ‘while’ döngüsü bulunmaktadır ve atama yapılan geçici değişken, metoda gelen parametreye eşit olmayana kadar devam etmesi koşuluyla çalışacaktır. Bu döngü çalışmaya başladığı vakit karmaşıklığımıza ‘N2’ seviyesinden devam etmemiz gerekir. Döngü içerisinde sayaç değişkenin sıfır dan büyük olması ve ayrıca bir kontrol daha yer almaktadır ve bir atama işlemi bulunuyor, bu sebeple maliyet ‘+3N2’ olarak arttırılmalıdır. Bu koşulların içerisinde alt düğümlerin sayısı sıfırdan büyük olması durumunda çalışması gereken bir kod parçası bulunuyor ve bunun sayısı bilinmediği için ‘N’ karmaşaya sahiptir. Bu kod parçasının içerisinde bir kontrol ve dört adet atama işlemi bulunmaktadır. Bu sebeple karmaşıklığımıza ‘N2+5N3’ maliyeti eklenir. Bu kod parçası çalıştıktan sonra değeri değiştirilen değişkene göre bir kontrol işlemi gerçekleşir ve kontrol sonucu sırası gelen kod parçası çalışacak olursa maliyet ‘N2+5N3+3N2’ artacaktır.

Arama işleminde gelen parametreye eşit seviyede bir düğüm bulunursa bu düğümün listeye eklenmesi ve ziyaret edildi olarak işaretlenmesi için ‘+3N2’ maliyet eklenmedir.

Düğüm eklendikten sonra ziyaret edilmeyen elemanın kalmadığının kontrolü için koşullamalar yapılmıştır. Buradaki kontrol if-else if yapısı ile sağlanmıştır. İkisinin de karmaşıklığına bakacak olursak:

Eğer ilk koşul sağlanırsa; alt iki koşul bulunmaktadır. Bunlardan birincisi ‘+3N2’ karmaşıklık getirirken diğerinde, bir atama işlemi ve ‘N’ defa çalışan bir döngü bulunuyor. Bu döngü içerisinde de dört atama işlemi bulunduğu için karmaşıklığa ‘N2+5N3+N2’ eklenmelidir. Bu döngüden elde edilen değere göre bir koşul daha bulunduğu için bu koşulda iki adet atama ve bir adet N defa çalışan döngü bulunuyor bu koşula göre algoritmanın karmaşıklık maliyeti ‘N2+2N2+5N3’ artacaktır.

Eğer ikinci koşul sağlanacak olursa; ilk olarak iki adet atama işlemi ve bir adet ‘N’ defa çalışan döngü ve bu döngü içerisinde bir koşul sağlandığı vakit dört adet atama işlemi gerçekleşecektir. En kötü ihtimalle bu koşul ‘N’ defa sağlanıp atama işlemleri de ‘N’ defa gerçekleşeceği için maliyete ‘5N3+2N2+N2’ eklenmelidir. Bu döngü çalıştıktan sonra elde edilen kontrol değişkeninin değerine göre son olarak bir kontrol daha yapılmaktadır. Bu kontrolde de yapılan koşullamada bir adet atama, bir kontrol, tekrar bir atama ‘N’ defa çalışan döngü içerisindeki ‘if’ koşulunda dört adet atama işlemi gerçekleşir ve en kötü ihtimalde de maliyete ‘4N2+5N3’ eklenecektir. İkinci koşulun maliyeti daha büyük olduğu için (7N3+10N2) ikinci koşul en kötü ihtimal olarak seçilmelidir.

Son olarak sayaç değişkeninin bir arttırılması gerketiği için maliyete ‘+N2’ eklenir.

Sonuç olarak projede kullanılan algoritmanın karmaşıklık analizini yaparken toplam karmaşıklık maliyeti ‘17N3+ 21N2+7N’ olarak bulunur ve ‘Big O Notasyonu[6]’na göre O(17N3+ 21N2+7N) = O(N3) bulunacaktır.

Zaman karmaşası olarak da T(n) = (17N3+ 21N2+7N) olacaktır.

# Sonuçlar

Proje, ‘Harita.txt’ dosyasındaki karakter bilgilerini okuyup ilgili düşman ve oyuncu karakter yapılarını oluşturup harita üzerinde nereye yerleştirilmeleri gerektiklerini nesnelere koordinat bilgileri ile bildirir.

‘Harita.txt’ dosyasında yer alan ‘1’ ve ’0’ rakamlarına göre ‘0’ların engel, ‘1’lerin hareket edilebilecek yol olduğunu işlenir ve buna göre oluşturulan matrisi grafik ekranının oluşturulduğu sınıfa gönderilir.

Grafik ekranı kendi içerisinde tanımlanan karakter ve harita oluşturmak için tanımlanan bilgilerin ‘main’ sınıf tarafından değer ataması yapıldıktan sonra grafik ekranını hazırlar ve kullanıcıya sunar. Kullanıcı

belirlenen tuşları kullanarak oyuncu karakteri ile etkileşimde bulunur ve düşman karakterler de oyuncu karakterine giden en kısa yolu tespit ederek o yolda karaktere özel olarak tanımlanmış hareket özelliğine göre hareketini gerçekleştirir ve ardından oyuncu karakterin son konumuna göre en kısa yol hesaplamasını tekrar gerçekleştirir.

1. **Kaynakça**

[1] “BilgisayarKavramlari”

Sığ Öncelikli Arama (Breadth First Search , BFS) – Bilgisayar Kavramları 13.11.2008

<http://bilgisayarkavramlari.sadievrenseker.com/2008/11/13/sig-oncelikli-arama-breadth-first> search/

28.02.2019

[2] “BilgisayarKavramlari”

Swing Kütüphanesi – Bilgisayar Kavramları

01.06.2011

<http://bilgisayarkavramlari.sadievrenseker.com/>2011/06/01/swing-kutuphanesi/

02.03.2019

[3] “TURKCELL | Geleceği Yazanlar”

Java’da Dosya İşlemleri | Geleceği Yazanlar

28.04.2016

http://theoryofprogramming.com/2018/01/14/n-ary- tree-k-way-tree-data-structure/

02.03.2019

[4] “Stack Overflow”

How to split a string in Java – Stack Overflow 10.08.2010

https://stackoverflow.com/questions/3481828/how-to-split-a-string-in-java

03.03.2019

[5] “Stack Overflow”

Java – How to calculate the running time of my program? – Stack Overflow

05.03.2011

https://stackoverflow.com/questions/5204051/how-to-calculate-the-running-time-of-my-program

05.03.2019

[6] “Yazılıma Giriş”

The Big O Notation – Yazılıma Giriş

30.03.2017

<http://yazilimagiris.com/2017/03/o-notation/>

10.03.2019