Gezgin Robot Proje Raporu

Taha Furkan Genç
Öğrenci No :210201077

Emir Muharrem Acar
Öğrenci No: 210201059

Özet

Bu rapor 2022-2023 öğretim yılı Kocaeli Üniversitesi 2. sınıf öğrencileri Taha Furkan Genç ve Emir Muharrem Acar tarafından Programlama Labaratuvarı 2 dersinin 1. projesi için hazırlanmıştır. İçerikte projeden genel anlamıyla bahsedilmiş olup gerçekleştirilen eylemler, verimlilik, sunum, analiz ve projenin hedefteki istenilen amaca hizmet etme şekli ve miktarından bahsedilmiştir.

I. Giriş

A. Projenin Genel Tanımı

Gezgin Robot projesi bilinmeyen engellerin olduğu bir yolda hedefe ulaşan ve gezdiği yolların en kısa mesafesini hesaplayan bir robotun olduğu projedir. Robotun çalıştığı alanların iki farklı modu vardır. Bunlardan birincisi : daha öncesinde oluşturulmuş bir url içerisindeki txt formatlı engel template'ini alan ve içerisinde tanımlanmış olan alanlara özel engelleri oluşturan bir moddur. İkincisi ise kullanıcıdan alınan boyutlara uygun bir şekilde rastgele labirent oluşturan bir moddur.

B. Giriş

Bu kısımda projede istenilenler ve bize verilen bilgiler doğrultusunda adımların belirlenmesi sağlanacaktır. Adımlar şöyle belirlenmiştir:

- Labirentin istenilen moda göre oluşturmak.
- Labirentin arayüze aktarılmasını sağlamak.
- Labirentin başlangıc ve bitiş noktasını belirlemek.
- Labirentte yapılacak olan kesif turunu oluşturmak.
- Labirentte en kısa yol algoritmasını çalıştırmak.
- Sonuçların arayüzde göşterilmesi ve text dosyasına kaydedilmesi.

II. Yöntem

A. Alan Modu Belirlemek

Programın çalıştırılacağı ana ekrandan önce , alanın modunun seçilebilmesi için yeni bir ekran açılır (Fig. 1). Bu ekranda 3 adet tuş bulunmaktadır.Kullanıcının seçebileceği 2 farklı URL adresi vardır. Bu üç tuşun biri , rastgele labirent oluşturur. Diğer ikisi ise URL'den alınan bilgilere göre rastgele engeller ile bir alan oluşturmak içindir. Programın çalıştırılacağı ana ekrandan önce , alanın modunun seçilebilmesi için yeni bir ekran açılır. Bu ekranda 3 adet tuş bulunmaktadır.Bu üç tuşun biri rastgele labirent oluşturur. Diğer ikisi ise URL'den alınan bilgilere göre rastgele engeller ile bir alan oluşturmak içindir.Kullanıcının seçebileceği 2 farklı URL adresi vardır. Bunlardan birincisi 10x10 formatında bir alan verirken diğeri 20x20 formatında bir alan verir.

```
JFrame secimframe = new JFrame( title: "Seçim ekranı");
secimframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
secimframe.setLayout(new GridLayout( rows: 3, cols: 1, hgap: 10, vgap: 10));
JButton url1button = new JButton( text: "URL1");
url1button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            int maze[][]=urldenLabirentYap( txturl: "http://bilgisayar.kocaeli.edu.tr/prolab2/url1.txt");
            labirentmain(maze, urldenmi: true, labirentboyutu: 30);
        } catch (Exception ex) {
            throw new RuntimeException(ex);
        }

    }
});
JButton url2button = new JButton( text: "URL2");
url2button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            int maze[][]=urldenLabirentYap( txturl: "http://bilgisayar.kocaeli.edu.tr/prolab2/url2.txt");
            labirentmain(maze, urldenmi: true, labirentboyutu: 30);
        } catch (Exception ex) {
            throw new RuntimeException(ex);
        }

    }
});
JButton randommazebutton = new JButton( text: "RANDOM");
randommazebutton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        int labirentboyutu=25;
        do {
            String labirentboyutstr = JOptionPane.showInputDialog(secimframe,  message: "Labirent boyutu giriniz (5-100)");
            labirentboyutu = Integer.parseInt(labirentboyutstr);
        }while (!(labirentboyutu>5 && labirentboyutu<100));
        int labirentboyutupxl=750/labirentboyutu;
        Mazemaker mazemaker = new Mazemaker(labirentboyutu, labirentboyutu);
        mazemaker.doldur( baslangicX: 1, baslangicY: 1);
```

Fig. 1. Alan modu belirleme kodu

B. Engellerin Yerleştirilmesi

Eğer seçilen mod URL'den alınan alan üzerinden işlem yapılması ise programın işleyişi şu şekilde olur : URL'nin içindeki txt dosyası okunurken bir yandan da programa kaydedilir(Fig. 2). Verilen alanda yazılan 2x2 lik alanlar (Fig. 3) ve 3x3 lük alanlar (Fig. 4), programın engel koyması için belirtilmiş alanlardır. Bu alanlara koyulacak engeller sadece o formattaki alanlara özel olmalıdır. Örneğin 3x3 lük formatta oluşturulan engeller 2x2 lik alanlara da uyumlu olmamalıdır. Eğer seçilen mod rastgele labirent oluşturma ise programın işleyişi şu şekilde olur : Labirentin oluşturulmasından önce kullanıcıya labirent boyutunu belirtmesi için bir ekran verilir. Eni ve boyu, kullanıcıdan alınan değerle aynı olacak şekilde labirent oluşturulur.
URL'den alanı alıp engel yerleştirmek
URL içindeki labirentin boyutlarını al
URL içindeki Txt'yi satır satır al;
her satırdaki karakterleri tek tek al;
bu karakterleri Integer'a çevir;
Integer matrise kaydet;
matrisi sırası ile oku;
her 2 görülen yerde olan 2x2'lik alana, oluşturulan rastgele nesneyi yerleştir;

her 3 görülen yerde olan 3x3'lük alana, oluşturulan rastgele nesneyi yerleştir;

```java
public static int[][] urldenLabirentYap(String txturl) throws Exception {
    Random random = new Random();

    int mazeboundsI=0;
    int mazeboundsJ=0;
    URL url = new URL(txturl);
    URLConnection conn = url.openConnection();
    BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    String inputLine;
    inputLine = reader.readLine();
    mazeboundsJ++;
    mazeboundsI=inputLine.length();
    System.out.println("MAZEBOUNDSI :"+mazeboundsI);
    while ((inputLine = reader.readLine()) != null) {
        // System.out.println(inputLine);
        mazeboundsJ++;
    }
    System.out.println("MAZEBOUNDSI: "+mazeboundsI+" MAZEBOUNDSJ: "+mazeboundsJ);
    reader.close();

    int[][] tmpmaze = new int[mazeboundsI][mazeboundsJ];

    URLConnection conn2 = url.openConnection();
    BufferedReader reader2 = new BufferedReader(new InputStreamReader(conn2.getInputStream()));
    int s1=0;
    while ((inputLine = reader2.readLine()) != null) {
        System.out.println(inputLine);
        for (int s2 = 0; s2 < inputLine.length() ; s2++) {
            tmpmaze[s1][s2]=Integer.parseInt(String.valueOf(inputLine.charAt(s2)));
        }
        s1++;
    }
    reader2.close();

    for (int i = 0; i < tmpmaze.length; i++) {
        for (int j = 0; j < tmpmaze[0].length; j++) {
            System.out.print(tmpmaze[i][j]+" ");
        }
        System.out.println();
    }
```

Fig. 2. Engellerin yerleştirilmesi kod örneği

```java
for (int i = 0; i < tmpmaze.length; i++) {
    for (int j = 0; j < tmpmaze[0].length; j++) {
        if(tmpmaze[i][j]==2)
        {
            int rastgelesifirla=random.nextInt( bound: 5);
            switch (rastgelesifirla){
                case 0:
                    tmpmaze[i][j]=0;
                    tmpmaze[i][j+1]=1;
                    tmpmaze[i+1][j]=1;
                    tmpmaze[i+1][j+1]=1;
                    break;
                case 1:
                    tmpmaze[i][j]=1;
                    tmpmaze[i][j+1]=0;
                    tmpmaze[i+1][j]=1;
                    tmpmaze[i+1][j+1]=1;
                    break;
                case 2:
                    tmpmaze[i][j]=1;
                    tmpmaze[i][j+1]=1;
                    tmpmaze[i+1][j]=0;
                    tmpmaze[i+1][j+1]=1;
                    break;
                case 3:
                    tmpmaze[i][j]=1;
                    tmpmaze[i][j+1]=1;
                    tmpmaze[i+1][j]=1;
                    tmpmaze[i+1][j+1]=0;
                    break;
                case 4:
                    tmpmaze[i][j]=1;
                    tmpmaze[i][j+1]=1;
                    tmpmaze[i+1][j]=1;
                    tmpmaze[i+1][j+1]=1;
                    break;
            }
        }
    }
```

Fig. 3. 2X2 engellerin yerleştirilmesi

```java
if (tmpmaze[i][j]==3){

    tmpmaze[i][j]=   1; tmpmaze[i+1][j]=   1; tmpmaze[i+2][j]=   1;
    tmpmaze[i][j+1]= 1; tmpmaze[i+1][j+1]= 1; tmpmaze[i+2][j+1]= 1;
    tmpmaze[i][j+2]= 1; tmpmaze[i+1][j+2]= 1; tmpmaze[i+2][j+2]= 1;

    int rastgelesifirla=random.nextInt( bound: 4);
    switch (rastgelesifirla){
        case 0:
            tmpmaze[i+ random.nextInt( bound: 3)][j+random.nextInt( bound: 3)]=0;
            tmpmaze[i+1][j+1]=1;
            break;
        case 1:
            tmpmaze[i+1][j+1]=0;
            int rastgelesifirla2=random.nextInt( bound: 4);
            switch (rastgelesifirla2){
                case 0:
                    tmpmaze[i+1][j]=0;
                    break;
                case 1:
                    tmpmaze[i+1][j+2]=0;
                    break;
                case 2:
                    tmpmaze[i][j+1]=0;
                    break;
                case 3:
                    tmpmaze[i+2][j+1]=0;
                    break;
            }
            break;
        case 2:
            tmpmaze[i+1][j]=0;
            tmpmaze[i+1][j+2]= 0;
            break;
        case 3:
            tmpmaze[i][j+1]=0;
            tmpmaze[i+2][j+1]= 0;
            break;
    }

}
```

Fig. 4. 3X3 engellerin yerleştirilmesi

C. Labirentin Arayüze Aktarılması

Tüm işlemlerden sonra aldığımız labirentin içindeki değerlere göre sınıf oluşturulur (Fig. 5). Program her bir kare için iki farklı sınıftan nesne oluşturur. Bu sınıflar Engel ve Yol sınıflarıdır. Her bir sınıf, programın arayüzünde bir panel oluşturur. Bu paneller yardımı ile labirentimiz görselleştirilir.

D. Başlangıç Ve Bitiş Noktarını Belirlemek

Url'den alınan alanlarda başlangıç ve bitiş noktası belirlenirken harita dörde bölünür. Sol üst kısımda rastgele bir yerde başlangıç yeri belirlenir. Başlangıç koordinatları, robot sınıfından nesne oluşturulurken parametre olarak verilir (Fig. 6).Sağ alt köşede ise rastgele bir yerde bitiş yeri belirlenir. Bitiş koordinatları ise labirentin oluşturulduğu metotta , integer olarak verilir. Labirentin tutulduğu integer matrisinde "9" olarak tutulur (Fig. 7). Eğer seçilen mod rastgele labirent oluşturma ise başlangıç noktası her zaman 1,1 noktası olur. Başlangıç noktasından itibaren labirent oluşturulur. Labirent oluşturulduktan sonra oluşturulan matris sondan başa doğru taranır ve görülen ilk yol hedef olarak belirlenir. Genellikle hedef sağ alt köşede olur.

```
public static void labirentmain(int maze[][], boolean urldenmi,int labirentboyutu){
    Random random =new Random();
    JFrame frame = new JFrame( title: "Labirent");
    //frame.setSize(1000, 1000);
    //frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setBackground(Color.BLACK);

    Izgara[][] izgaralar = new Izgara[maze.length][maze[0].length];
    for (int i=0;i<maze.length;i++){
        for(int j=0;j<maze[0].length;j++){
            if(maze[i][j]==1)
            {
                izgaralar[i][j]=new Engel( x: labirentboyutu*j, y: labirentboyutu*i,labirentboyutu,labirentboyutu,frame);
            }
            if(maze[i][j]==0)
            {
                izgaralar[i][j]=new Yol( x: labirentboyutu*j, y: labirentboyutu*i,labirentboyutu,labirentboyutu,frame);
            }
            if(maze[i][j]==9)
            {
                izgaralar[i][j]=new Yol( x: labirentboyutu*j, y: labirentboyutu*i,labirentboyutu,labirentboyutu,frame);
                JLabel bitisisareti = new JLabel( text: "Fnsh");
                bitisisareti.setVerticalAlignment(JLabel.CENTER);
                bitisisareti.setHorizontalAlignment(JLabel.CENTER);
                izgaralar[i][j].panel.add(bitisisareti);
                izgaralar[i][j].panel.setBackground(Color.ORANGE);
            }
        }
    }

    frame.setSize( width: labirentboyutu*maze[0].length+250, height: labirentboyutu*maze.length+250);
```

Fig. 5. Labirentin arayüze aktarılması

```
    Robot robot;
    int baslangicI;
    int baslangicJ;
    if(urldenmi==true) {

        int acikliksayisi;
        do {
            acikliksayisi = 0;
            baslangicI = (random.nextInt( bound: maze.length / 2) + 1);
            baslangicJ = (random.nextInt( bound: maze[0].length / 2) + 1);
            if (maze[baslangicI + 1][baslangicJ] == 0)
                acikliksayisi++;
            if (maze[baslangicI][baslangicJ + 1] == 0)
                acikliksayisi++;
            if (maze[baslangicI - 1][baslangicJ] == 0)
                acikliksayisi++;
            if (maze[baslangicI][baslangicJ - 1] == 0)
                acikliksayisi++;
        } while (acikliksayisi < 2 || maze[baslangicI][baslangicJ]!=0);
        //izgaralar[baslangicI][baslangicJ].panel.setBackground(Color.ORANGE);
        //if(acikliksayisi>1 && maze[baslangicI][baslangicJ]==0)
        robot = new Robot(baslangicI, baslangicJ, maze, izgaralar);
    }
    else{
        baslangicI=1;
        baslangicJ=1;
        robot = new Robot(baslangicI, baslangicJ, maze, izgaralar);
    }
    int graf[][]= robot.grafOlustur(robot.getRobothafiza(),izgaralar);
    int baslangicdugumnumarasi=izgaralar[baslangicI][baslangicJ].dugumnumarasi;
```

Fig. 6. Başlangıç noktasının belirlenmesi

E. Labirente Yapılacak Olan Kesif Turunu Olusturmak

Robot başladığı zaman sadece kendi olduğu nokta ve çevresindeki dört nokta olmak üzere sadece 5 noktayı bilmektedir.Robotun daha önce görmediği , hafızasına almadığı , noktalar arayüzde sis ile gösterilir. Robot sisli blokları bilemez.Robot çözüme başladığında büyük olasılık dahilinde hedefi de göremeyeceğinden dolayı en kısa yolu bulabilmek için öncelikle bir keşif turu atmalı ve hedefi bulmalıdır. Bu keşif turunda robotumuzun mantığı şu şekildedir: robot sırayla alt blok, sağ blok , üst blok , ve sol blok kontrolü yapar. Bu noktalara sırayla bakarken eğer bir engel veya önceden gezilmiş bir yol görmediği sürece o yöne doğru ilerler ve algoritmasını tekrar çalıştırır.Robot

```
int acikliksayisi=0;
int randmazeI;
int randmazeJ;
do {
    acikliksayisi=0;
    randmazeI= (random.nextInt( bound: tmpmaze.length / 2) + tmpmaze.length / 2)-1;
    randmazeJ= (random.nextInt( bound: tmpmaze[0].length / 2) + tmpmaze[0].length / 2)-1;
    if(tmpmaze[randmazeI+1][randmazeJ]==0)
        acikliksayisi++;
    if(tmpmaze[randmazeI][randmazeJ+1]==0)
        acikliksayisi++;
    if(tmpmaze[randmazeI-1][randmazeJ]==0)
        acikliksayisi++;
    if(tmpmaze[randmazeI][randmazeJ-1]==0)
        acikliksayisi++;
}while (acikliksayisi<2 || tmpmaze[randmazeI][randmazeJ]!=0);//degistirdim

tmpmaze[randmazeI][randmazeJ]=9;
```

Fig. 7. Bitiş noktasının belirlenmesi

her ilerlediği bloğu ve olduğu bloğun çevresindeki 4 bloğu hafızasına ekler ve oradaki sis kalkar. Labirentte yanlış girilen bir yol algılandığında robotun doğru olarak tespit ettiği en son konuma giderek buradan itibaren yol aramaya devam etmesi gerekmektedir. Bu yüzden bu algoritmada özyineleme tekniği kullanılmıştır (Fig. 8).
robot olduğu adımı boyar;
robot ,çevresindeki 4 bloğu açar ve hafızasına alır;
eğer robootun çevresinde bitiş bloğu varsa oraya yönlenir ve program biter;
eğer robotun çevresinde bitiş bloğu yoksa önce aşağıya bakar. Eğer orada yol varsa bu fonksiyon tekrar çağırılır;
Eğer orada yol yoksa sağa bakar. Eğer sağda yol varsa bu fonksiyon tekrar çağırılır;
Eğer orada yol yoksa yukarı bakar. Eğer yukarıda yol varsa bu fonksiyon tekrar çağırılır;
Eğer orada yol yoksa sola bakar. Eğer solda yol varsa bu fonksiyon tekrar çağırılır;

```
//asagi
boolean returndeger=robothareket(maze, izgaralar, robotunIkoordinati: robotunIkoordinati+1, robotunJkoordinati,robothafiza);
//sol
if(!returndeger){
    returndeger=robothareket(maze, izgaralar, robotunIkoordinati, robotunJkoordinati: robotunJkoordinati+1,robothafiza);
}
//yukari
if(!returndeger){
    returndeger=robothareket(maze, izgaralar, robotunIkoordinati: robotunIkoordinati-1, robotunJkoordinati,robothafiza);
}
//sag
if(!returndeger){
    returndeger=robothareket(maze, izgaralar, robotunIkoordinati, robotunJkoordinati: robotunJkoordinati-1,robothafiza);
}

if (returndeger){
    izgaralar[robotunIkoordinati][robotunJkoordinati].panel.setBackground(Color.MAGENTA);
}

return returndeger;
```

Fig. 8. Recursive ile keşif turu

F. Labirentte En Kısa Yol Algoritmasını Çalıştırmak

Programın nihai amacı olan kısa yol algoritmasını çalıştırmak için öncelikle bir graf matrisi oluşturmamız gerekliydi. Bu kısımda daha öncesinde birkaç düğüm belirleme algoritması oluşturmuştuk. Fakat her oluşturduğumuz algoritmada ihtimali küçük de olsa graf bağlanmama sorunu oluyordu. Bu yüzden düğüm belirleme algoritması oluşturmak

yerine robotun açtığı tüm yolları bir düğüm olarak tanımlayıp ağırlıksız bir graf oluşturduk. Bu şekilde grafta kopukluk oluşma riskini sıfıra indirdik (Fig. 9). Daha öncesinde ağırlıklı graflar üzerinde çalışacağımızı düşündüğümüz için ağırlıklı graflar üzerinde çalışan bir dijkstra algoritması yazmıştık. Fikrimizi değiştirdiğimizde yazdığımız dijkstra algoritmasının ağırlıksız graflarda da sorunsuz çalışabildiğini gördüğümüzden bu algoritmayı değiştirmemeye karar verdik. Algoritmamızda başlangıç noktamız olan düğümümüzun en kısa yol uzaklığı 0 olarak ayarlanmıştır. Daha sonrasında bu düğümlerin komşularında olan düğümlerin ağırlıkları girilir. En kısa olan düğümden devam edilerek dijkstra tablosu düğümlere bağlı olarak oluşturulur. Her düğüm için "bir önceki düğüm" değişkeni tutulur. Bu sayede en kısa yolun güzergahı olan düğümler elde edilebilir.

```
//sağ kontrol
for(int sag=i+1;sag<robothafiza.length;sag++){
    if(robothafiza[sag][j]==2 || robothafiza[sag][j]==1) {
        break;
    }
    if(izgaras[sag][j].dugumnumarasi!=-1){
        graph[izgaras[sag][j].dugumnumarasi][izgaras[i][j].dugumnumarasi]=sag-i;
        break;
    }
}

//sol kontrol
for(int sol=i-1;sol>0;sol--){
    if(robothafiza[sol][j]==2 || robothafiza[sol][j]==1) {
        break;
    }
    if(izgaras[sol][j].dugumnumarasi!=-1){
        graph[izgaras[sol][j].dugumnumarasi][izgaras[i][j].dugumnumarasi]=i-sol;
        break;
    }
}

//aşağı kontrol
for(int asagi=j+1;asagi<robothafiza[0].length;asagi++){
    if(robothafiza[i][asagi]==2 || robothafiza[i][asagi]==1) {
        break;
    }
    if(izgaras[i][asagi].dugumnumarasi!=-1){
        graph[izgaras[i][asagi].dugumnumarasi][izgaras[i][j].dugumnumarasi]=asagi-j;
        break;
    }
}

//yukari kontrol
for(int yukari=j-1;yukari>0;yukari--){
    if(robothafiza[i][yukari]==2 || robothafiza[i][yukari]==1) {
        break;
    }
    if(izgaras[i][yukari].dugumnumarasi!=-1){
        graph[izgaras[i][yukari].dugumnumarasi][izgaras[i][j].dugumnumarasi]=j-yukari;
        break;
```

Fig. 9. Dügümleri bağlama

G. Sonuçların Arayüzde Gösterilmesi ve Text Dosyasına Kaydedilmesi

Labirentte oluşan her bloğun bir düğüm numarası vardır. Bu düğüm numaraları graf oluşturma kısmında güncellenir. Güncellenmeyen düğüm numaraları "-1" olarak tutulur. Dijkstra algoritması bize bu düğüm numaralarını sırayla verir (Fig. 10). Bu düğümlerin sırayla verilmesi bu kısımda da animasyon yapabilmemiz için olanak sağlayacağından yöntemimizin avantajlı olduğunu düşündük. Fakat daha sonrasında gelişen komplikasyonlardan dolayı herhangi bir animasyon yapamadık. Buradaki temel problemin tuş aktivasyonunun olduğu alanda çağırılan fonksiyonun içindeki Thread.sleep() fonksiyonundan kaynaklı olduğunu düşünüyoruz. Fonksiyonun tamamı bitmediğinden dolayı animasyon kısımlarını göstermek yerine fonksiyonun bitmesini bekliyor ve animasyonu göremiyoruz. Tuş olmadan başlayan prototiplerimizde animasyonu da aktif olarak görebiliyoruz. Program animasyon olmasa dahi robot hareketlerini ve dijkstra yolunu başlangıçtan bitişe sıra ile oluşturur . "Dijkstra yap" butonu bu olayı gerçekleştirir. "Dijkstra yap" butonunun diğer bir özelliği ise robotun sonuçlarının gösterildiği bir Frame oluşturmaktır. "Dijkstra yap" butonunun diğer bir özelliği ise açılan Frame'de yazan sonuçları ve en kısa yolun tutulduğu bir txt formatlı labirenti, programın dosyasında oluşturulan bir txt dosyasına yazdırmaktır.Animasyon yapmak yerine alternatif bir yöntem olarak klavye tuşlarını kullandık. "yolu göster" butonunun görevi bu özelliği çalıştırmaktır. Sağ ok tuşuna bastıkça robotun hareket ettiği alan sırasıyla boyanır. Her boyanmada gidilen karenin sırası bloğun üstüne yazılır. İsteğe bağlı olarak bu işlem sol ok tuşu ile geriye çekilebilir. .

```
public static ArrayList<Integer> dijkstra(int [][] graf, int baslangicindis, int bitisindis){
    int[] uzaklik = new int[graf.length]; // En kısa mesafeleri tutar
    boolean[] ziyaretedildimi = new boolean[graf.length]; // Ziyaret edilen düğümleri tutar
    int[] oncekidugum = new int[graf.length]; //(yolu göstermek için) önceki düğümleri tutar

    // Diğer tüm düğümlerin mesafeleri sonsuz olarak ayarlanır
    //ziyaretedildimi false ayarlanır
    for (int i = 0; i < graf.length; i++) {
        uzaklik[i]=Integer.MAX_VALUE;
        ziyaretedildimi[i]=false;
    }


    // Başlangıç düğümüne mesafe 0 olarak ayarlanır
    uzaklik[baslangicindis]=0;

    // Tüm düğümleri ziyaret edene kadar döngü
    for (int i = 0; i < graf.length-1; i++) {
        // Henüz ziyaret edilmemiş en kısa mesafeli düğümü bul
        int minimumindex = minimumDugumBul(uzaklik, ziyaretedildimi);

        // Düğümü ziyaret et
        ziyaretedildimi[minimumindex] = true;

        // Düğümün tüm komşularını kontrol et
        for (int j = 0; j < graf.length; j++) {
            // Henüz ziyaret edilmemiş ve minimumindex'den j'ye olan mesafe pozitif ise
            if (!ziyaretedildimi[j] && graf[minimumindex][j] > 0) {
                int denenecekuzaklik = uzaklik[minimumindex] + graf[minimumindex][j];
                // Yeni mesafe, mevcut mesafeden daha kısa ise güncelle
                if (denenecekuzaklik < uzaklik[j]) {
                    uzaklik[j] = denenecekuzaklik;
                    oncekidugum[j] = minimumindex;
                }
            }
        }
    }
```

Fig. 10. Dijkstra

III. SONUÇ

A. Programın Yapabildikleri ve İşlevsellikler Hakkında

Programın yöntemler ve kod ifadeleriyle de anltıldığı üzere yerine getirilmesi istenen işlevlerin büyük kısmını yapabilir olup bunlar : , kullanıcıdan alınan boyutlara göre çözümü olan bir labirent oluşturabilmek, matris formatında verilen alanı arayüze aktarabilmek, arayüze sis ekleyebilmek, haritayı öğrenebilen bir robot oluşturmak, keşif turu yapabilen bir algoritma geliştirmek, açılan yollardan yola çıkarak en kısa yolu

bulabilmek, en kısa yolu arayüzde ve programın oluşturduğu txt dosyasında gösterebilmektir.

IV. Deneysel Çıktılar

A. Deneysel Çıktıların Açıklamalarıyla Verilmiş Halleri

Fig. 11. Seçim ekranı

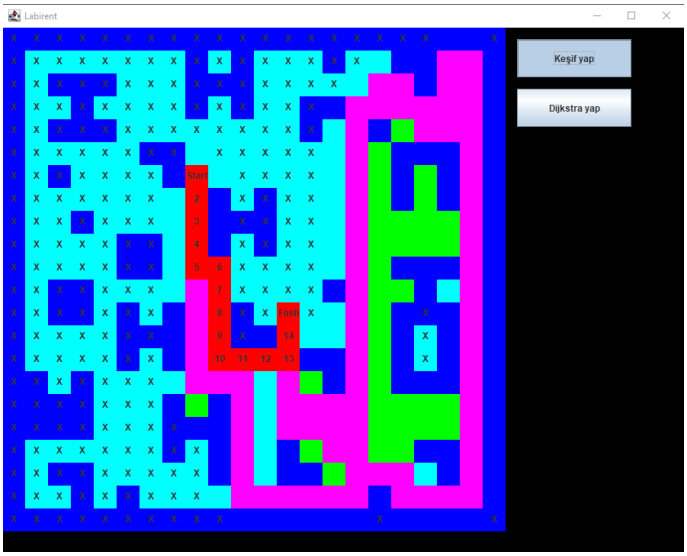


Fig. 12. URL1'den oluşan alanın dijkstra ile çözümü .

V. Kapanış ve Kaynakça

Çıktılar yüklenen dosya üzerindeki işlemlerden alınmıştır.

A. Kaynakça

https://www.youtube.com/watch?v=dyrvXiMumXc
https://www.youtube.com/watch?v=rop0W4QDOUI
https://www.youtube.com/watch?v=jT3c45XkPTg
https://www.youtube.com/watch?v=Kmgo00avvEw
https://www.dropbox.com/s/fn7tbiwv2zx46ga/Maze.java?dl=0
https://algs4.cs.princeton.edu/41graph/Maze.java.html

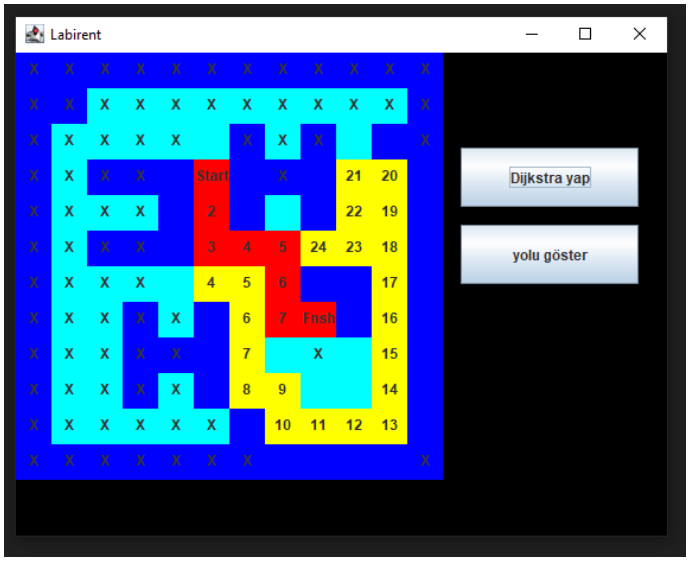


Fig. 13. URL2'den oluşan alanın dijkstra ve robot hareketi.

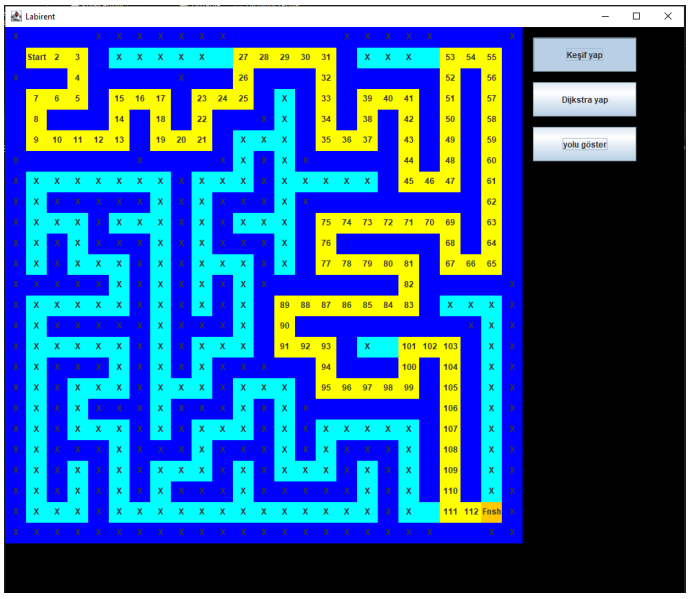


Fig. 14. Rastgele oluşturulan labirentin çözümü ve robot hareketi.

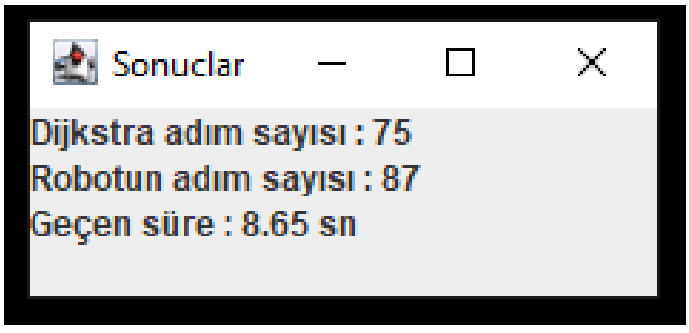


Fig. 15. Sonuç penceresi.

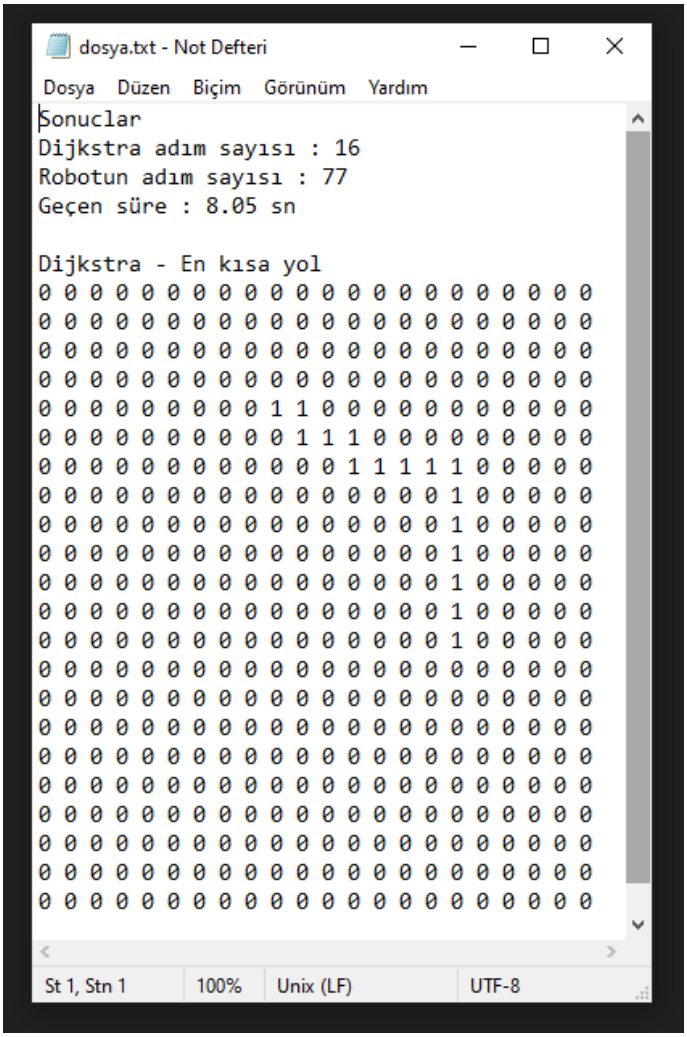
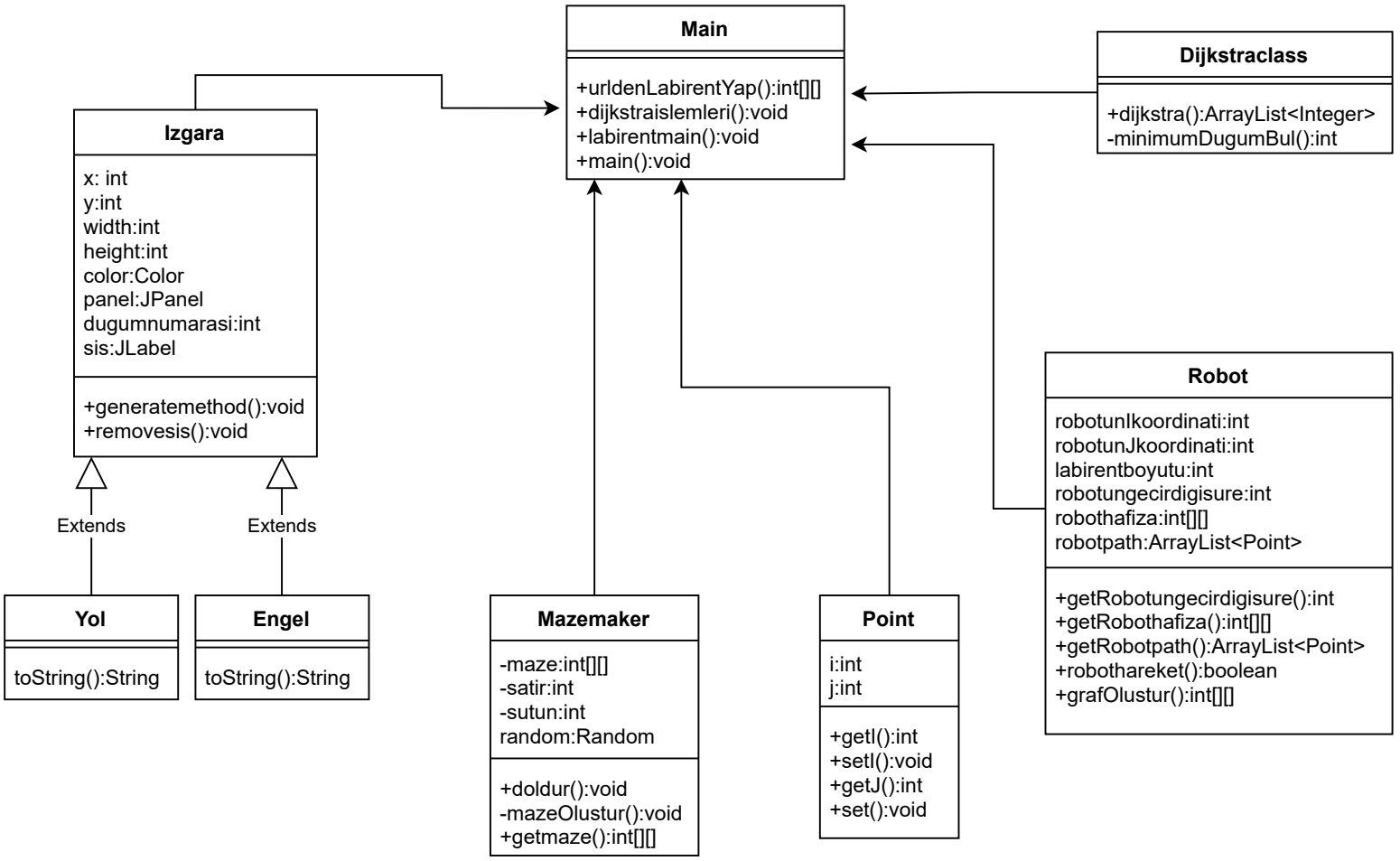


Fig. 16. Sonuçların text dosyasına yazdırılması.

https://coderanch.com/t/343686/java/setLocationRelativeTo
https://www.baeldung.com/java-solve-maze
https://github.com/armin-reichert/mazes/tree/master/mazes-algorithms/src/test/java/de/amr/maze/tests
https://docs.oracle.com/javase/tutorial/uiswing/components/button.html
ChatGPT
Veri Yapıları ve Algoritmalar dersinin slaytları (Bazı kaynakların URL'sinde LaTeX komutlarını çalıştıran karakterler olduğundan dolayı rapora eklenememiştir.)


Main
+urldenLabirentYap():int[][]
+dijkstraislemleri():void
+labirentmain():void
+main():void
Dijkstraclass
+dijkstra():ArrayList<Integer>
-minimumDugumBul():int
Izgara
x: int
y:int
width:int
height:int
color:Color
panel:JPanel
dugumnumarasi:int
sis:JLabel
+generatemethod():void
+removesis():void
Extends
Extends
Robot
robotunIkoordinati:int
robotunJkoordinati:int
labirentboyutu:int
robotungecirdigisure:int
robothafiza:int[][]
robotpath:ArrayList<Point>
+getRobotungecirdigisure():int
+getRobothafiza():int[][]
+getRobotpath():ArrayList<Point>
+robothareket():boolean
+grafOlustur():int[][]
Yol
toString():String
Engel
toString():String
Mazemaker
-maze:int[][]
-satir:int
-sutun:int
random:Random
+doldur():void
-mazeOlustur():void
+getmaze():int[][]
Point
i:int
j:int
+getI():int
+setI():void
+getJ():int
+set():void