

PROJE 1: GEZGİN ROBOT PROJESİ

1st Muhammet Rıdvan İNCE *Kocaeli Üniversitesi*

Bilgisayar Mühendisliği Bölümü

Kocaeli, Türkiye

Öğrenci No: 210201123

2nd Ömer YENER *Kocaeli Üniversitesi*

Bilgisayar Mühendisliği Bölümü

Kocaeli, Türkiye

Öğrenci No: 210201122

Abstract—Bu projenin amacı, Belirli kurallara göre hareket eden bir robotun önündeki engelleri aşarak istenen hedefe ulaşmasını sağlayan bir oyun tasarlanmasıdır. Oyunda iki adet problem çözülmektedir. Problemlerin çözümü için nesneye yönelik programlama ve veri yapıları bilgileri kullanılmıştır.

Index Terms—Nesneye Yönelik Programlama, Veri Yapıları, Algoritma, Python

I. GİRİŞ

Proje kendi içerisinde alt adımları olan 2 adet problemin çözülmesini gerektirmektedir. Birinci problemde robotun verili bir ızgara üzerinde engellere takılmadan en kısa süre içerisinde en kısa yolu kullanarak hedefe ulaşması gerekmektedir. Burada önemli olan husus robotun tüm ızgarayı değil yalnızca gerekli yolları gezerek hedefe ulaşmasının sağlanmasıdır. Birinci problemin alt adımları şu şekildedir:

- **İlk Adım:** Izgaranın oluşturulması
- **İkinci Adım:** Engellerin ızgaraya yerleştirilmesi
- **Üçüncü Adım:** Robot ve hedefin ızgara üzerinde uygun olan rastgele bir noktaya yerleştirilmesi
- **Son Adım:** Robotun hedefe ulaştırılması

Oluşturulan ızgaraların boyut bilgileri ile ızgaralar içerisindeki engellerin konum ve tip bilgilerine verili bir url adresindeki text dosyasından ulaşılmıştır. Izgarada kapladıkları alan büyüklüğüne göre 3 tip engel bulunmaktadır. Bunlardan 1 nolu engel 1 kare 2 nolu engel maksimum 4 kare 3 nolu

engel ise maksimum 9 kare kaplamaktadır. Burada her bir kare robot için bir adımı ya da bir hareketi ifade etmektedir. Robot bulunduğu konumdan sağ - sol - aşağı - yukarı olmak üzere 4 farklı noktaya hareket edebilmektedir.

Engeller ızgara üzerine yerleştirildikten sonra tüm ızgara duman ile kaplanmaktadır. Bunun amacı robotun hareketi esnasında sadece etrafındaki (sağ - sol - yukarı - aşağı) kareleri görebilmesi ve ızgaranın diğer noktaları hakkında herhangi bir bilgiye sahip olmamasıdır. Robot ızgaraya rastgele bir şekilde yerleştirilmiş olan hedef noktasında konumunu bilmemektedir. Bu doğrultuda robotun her bir birim hareketi neticesinde etrafındaki karelerin dumanı kaldırılmakta ve ne olduğu görülmektedir. Yani özetle robot sadece etrafında ne olduğunu bilmekte ızgara dünyasının geri kalanı hakkında bir bilgiye sahip bulunmamaktadır.

Bu doğrultuda her bir adımda etrafındaki karelere ilerleyen robot hedef noktaya ulaşması durumunda gezdiği tüm noktalar içerisinde başlangıç konumu ile hedef noktası arasındaki en kısa yolu tespit etmekte ve ızgara üzerinde bu yolu işaretlemektedir. Son olarak ise tüm bu süreçler hakkında bir rapor yazdırmaktadır.

Proje kapsamında çözülmesi gereken ikinci problem ise rastgele oluşturulan bir labirentin bir köşesinde bulunan robotun karşı çapraz köşede bulunan çıkış noktasına ulaştırılmasını kapsamaktadır. Problem 2'nin alt adımları ise şu şekildedir:

- **İlk Adım:** Kullanıcı tarafından verilecek boyutlarda bir ızgara oluşturulması
- **İkinci Adım:** Izgara üzerine problem 1'deki 1 nolu tipteki engelleri rastgele bir şekilde

yerleřtirmek

- **Üçüncü Adım:** Bařlangıç noktasından harekete bařlanması

Problem 2 kapsamında kullanılacak ızgaranın boyutları yani bir matris olarak düşünöldüğünde satır ve sütun sayıları kullanıcı tarafından belirlenmektedir. Iızgaranın oluşturulmasının akabinde problem 1'deki birinci tipteki engeller ızgara üzerine rastgele olarak yerleřtirilerek labirent oluşturulmuřtur. Labirent hedefe ulaşmayacak bir çok yolu içerecek řekilde tasarlanmıřtır.

Robotun bařlangıç noktası labirentin 4 köřesinden herhangi biri olabilir. Labirentten çıkıř noktası ise bařlangıç noktasının karřı çapraz köřesindeki nokta olarak belirlenmiřtir. Robot, bulunduğru noktadan Problem 1'dekine benzer řekilde sağ - sol - ařağı - yukarı olmak üzere 4 farklı noktaya hareket edebilmektedir. Robot, çıkıř olmayan bir yola girmesi durumunda ise geldiğru yoldan farklı yöne gidebileceğru ilk noktaya geri dönerek alternatif bir rotaya doğru hareket etmektedir. Robot geri dönerken iki kere geçtiğru noktalar hariç bir kere geçtiğru noktadan bir daha geçmemekte ve geri dönme esnasında geri döndüğru noktalara iřaret bırakmaktadır.

Robot bu řekilde hedef noktasına ulařtıktan sonra geçmiř olduğru tüm yollar içerisinde bařlangıç noktası ile çıkıř noktası arasındaki en kısa yolu belirleyerek bu yolları ızgara üzerinde göstermektedir.

II. YÖNTEM VE İÇERİK

Problemlerin çözümünde nesneye dayalı programa ve veri yapılarından faydalanılmıř olup Python programlama dili kullanılarak kodlanmıřtır. Arayüzün oluşturulmasında Tkinter ve Turtle kütüphanelerinden faydalanılmıřtır. Arayüzün nihai durumu Figür 1'de gösterilmektedir.

Proje kapsamında 14 adet sınıf tanımlanması yapılmıř olup ařağıdaki řekildedir. Ayrıca projeye ait UML diyagramı raporun sonuna eklenmiřtir.

- **Uygulama:** Bu sınıf projenin temel sınıfı olup, arayüz bu sınıf içerisinde oluşturulmuřtur. Ayrıca problemler kapsamında talep edilen temel görevlere ait metodlar bu sınıf içerisinde tanımlanmıřtır. Bu görevler problem 1 için; Iızgara oluşturup çeřitli tipteki engelleri bu ızgaraya yerleřtirmek (metod adı: URL getir), robot ve hedefi rastgele olarak ızgaraya yerleřtirip tüm ızgaranın dumanlanması



Fig. 1: Arayüz

ve robotun hareket bařlaması (metod adı: P1Calistir()) ve son olarak Robotun harekete bařlamasından sonra adım adım değıilde doğrudan sonuca gitmesi (metod adı: P1SonucGoster()) řeklindeyir. Problem 2 için ise rastgele labirentin oluşturulması (metod adı: LABDegistir()), robotun harekete bařlaması ve ařama ařama sonuca ilerlemesi (metod adı: P2Calistir()) ve son olarak robotun hızlı bir řekilde sonuca ilerlemesi (metod adı: P2SonucGoster()) řeklindeyir. Bu sınıf altında ayrıca robotun hızının kontrolü için 4 adet ilave metod eklenmiřtir. Uygulama sınıfı P1Fonksiyonlar ve P2Fonksiyonlar sınıflarının bir alt sınıfı olarak tasarlanmıřtır. Ayrıca Bu sınıf P1Izgara, P2Izgara, Robot, Duman, Hedef, Duvar ve EngelYerlestirici sınıfları ile güçlü iliřki içerisindeyir.

- **P1Izgara:** Bu sınıf içerisinde problem 1'e ait olan Iızgaranın özellikleri bulunmaktadır. Daha öncede ifade edildiğru üzere problem 1'deki ızgaraya ait bilgiler url adresi ile verilecek olan bir txt dosyasından çekilmektedir. Bu sebeple bu sınıfın bařlatılabilmesi için bir adet url bilgisine ihtiyaç bulunmakta olup yapıcı fonksiyon ařağıdaki řekilde tanımlanmıřtır.

```
class P1Izgara():
    def __init__(self, url):
        self.url = url
        .....
```

Bu url bilgisi kullanılarak oluşturulacak olan ızgaranın temel özellikleri de yine bu sınıf

içerisinde tanımlanmıştır. Bunlar ızgaranın satır ve sütun sayıları ile ızgaranın her bir karesinin piksel cinsinden boyutu ve ızgaranın her bir karesinin, engelli karelerin, engel olmayan karelerin ve labirentin sınırlarını belirleyen karelerin koordinatlarını gösteren listelerden oluşmaktadır.

- **P2Izgara:** Bu sınıf içerisinde problem 2'ye ait olan Izgaranın özellikleri bulunmaktadır. Temel olarak P1Izgara sınıfı ile aynı özellikleri taşımakta olup sınıfın başlatılabilmesi için bir url parametresine gerek bulunmamaktadır. Çünkü problem 2 kapsamındaki ızgara kullanıcının gireceği boyutlar doğrultusunda rastgele olarak oluşturulmaktadır. Rastgele labirentin oluşturulmasında Yığın veri yapısı kullanılmıştır. Bu sınıf P2Fonksiyonlar sınıfından kalıtım almaktadır. Çünkü bu sınıf içerisinde kullanılan Labirent P2Fonksiyonlar sınıfı içerisinde oluşturulmaktadır.
- **P1Fonksiyonlar:** Bu sınıfın içerisinde 7 adet metod bulunmakta olup Uygulama sınıfının üst sınıflarından birisidir. Ayrıca bu sınıf OrtakFonksiyonlar sınıfının bir alt sınıfı olarak tanımlanmıştır. Buradaki her bir metod Uygulama sınıfı içerisinde ilk problemin çözümünde kullanılmaktadır. Bu metodlardan ilki Labirent-Dolas() metodu olup içeriğine ait sözde kod aşağıdaki gibidir.

```
- GeriDonulen, ZiyaretEdilen, ogrenilen =
[RobotBaslangicKoordinat]
adimSayisi = 1
```

```
while True:
```

```
* Bulunduğu koordinatın sag sol alt ve üst
komsularını uygunKomsular isimli listede
tut. Bu komsular ZiyaretEdilen listesinde
olmayacak ya da labirentin engel (duvar) ve
sınır kısımlarında yer almayacak.
```

```
if (uygunKomsular)
```

```
-listeden bir komsuyu rastgele seç.
```

```
-Robotu seçilen koordinata götür.
```

```
-adimSayisi'ni bir artır
```

```
-Bulunduğu hücrenin ve komşu hücrelerin
dumanını temizle.
```

```
-Bulunduğu hücrenin korrdinatlarını
ZiyaretEdilen, GeriDonulen ve ogrenilen
listelerine ekle.
```

```
- ogrenilen listesine ayrıca bulunduğu hücrenin
komsularının da koordinatlarını ekle. Ancak bu
```

komsular ZiyaretEdilen listesinde olmayacak ya da labirentin engel (duvar) ve sınır kısımlarında yer almayacak.

- Komsuları kontrol et eğer hedef koordinat komsulardan birinde ise robotu oraya götür, bu hücrenin koordinatlarını geriDonulen listesine ekle ve döngüyü bitir.

else:

-geriDönülen listesini kullanarak bir önceki hücreye geri dön. Geri dönülen hücrenin komsularını kontrol et. Bu komsular içerisinde ZiyaretEdilen listesinde olmayan, ya da labirentin duvar ya da sınır kısımlarına denk gelmeyen bir komşusu yok ise bu hücreyi geriDonulen listesinden sil. Dongünün başına dön

-
- **P2Fonksiyonlar:** Bu sınıfın içerisinde 2 adet metod bulunmakta olup Uygulama sınıfının üst sınıflarından birisidir. Buradaki her bir metod Uygulama sınıfı içerisinde ikinci problemin çözümünde kullanılmaktadır. Bu sınıf ayrıca OrtakFonksiyonlar sınıfından miras almaktadır.
 - **OrtakFonksiyonlar:** Bu sınıfın içerisinde 1 adet metod bulunmakta olup P1Fonksiyonlar ve P2Fonksiyonlar sınıflarının üst sınıfı olarak tanımlanmıştır. Problem 1 ve 2'de kullanılan ortak operasyonlar için yazılan metodları içermektedir.
 - **Dugum:** Rastgele labirentin oluşturulmasında kullanılan veri yapısına ait sınıftır.
 - **Yigin:** Rastgele labirentin oluşturulmasında kullanılan veri yapısına ait sınıftır.
 - **Robot:** Robot sınıfı turtle.RawTurtle sınıfının bir alt sınıfı olarak tasarlanmış olup Izgara oluşturulduktan sonra bu ızgara üzerinde hareket edecek olan robotu temsil etmektedir.
 - **Duman:** Duman sınıfı turtle.RawTurtle sınıfının bir alt sınıfı olarak tasarlanmış olup Izgara oluşturulduktan sonra bu ızgaranın tamamen dumanla kaplanmasından sorumludur.
 - **Hedef:** hedef sınıfı turtle.RawTurtle sınıfının bir alt sınıfı olarak tasarlanmış olup Izgara oluşturulduktan sonra bu ızgara üzerine rastgele yerleştirilerek Robot tarafından tespit edilmeye çalışılan nesneyi temsil etmektedir.
 - **Duvar:** Duvar sınıfı turtle.RawTurtle sınıfının



Fig. 2: Izgara

bir alt sınıfı olarak tasarlanmış olup Izgara oluşturulurken engel yerleşecek olan yerleri işaretlemekten sorumludur.

- **EngelYerlestirici:** Bu sınıf turtle.RawTurtle sınıfının bir alt sınıfı olarak tasarlanmış olup Izgara oluşturulduktan sonra öncesinde duvar sınıfı tarafından işaretlenmiş olan bölgelere çeşitli kurallar çerçevesinde engelleri yerleştirmekten sorumludur.
- **Engel:** Bu sınıf Problem 1'de ızgara içerisine yerleştirilecek olan engellerin oluşturulduğu ve ızgaraya göre düzenlendiği sınıftır.

Her iki problemin çözüm sürecinde Izgaralar oluşturulurken turtle kütüphanesinden faydalanılmıştır. Bu kütüphaneden faydalanabilmesi için öncelikle turtle ve tkinter kütüphanelerinin birbirine bağlanması gerekmektedir. Bunun için öncelikle tkinter kütüphanesi kullanılarak bir ekran oluşturulmuş, bu ekrana yine tkinter kütüphanesi kullanılarak bir canvas eklenmiştir. Sonra turtle.TurtleScreen sınıfından üretilen ekran canvasın içine yerleştirilmiştir. Izgaralar bu ekran üzerine yerleştirilmiştir. Izgaraların ve ızgaraların üzerine de engellerin yerleştirilmesi aşamasında duvar, duman, engel ve engelYerlestirici sınıflarından faydalanılmıştır.

Duvar sınıfından türetilen duvar isimli turtle başlangıçta canvas üzerinde yerleştirilen ekran üzerinde (0,0) konumunda bulunmaktadır (Figür - 2). Başlangıçta belirlenen oyun alanı büyüklüğüne göre her bir karenin alanı toplam oyun alanının satır ya da sütun sayısına bölünmesi ile elde edilmektedir. Satır ya da sütundan hangisinin seçileceğine ise büyüklüklerine göre karar verilmektedir. Satır ya da sütun sayısından hangisi daha büyük ise her

```

31 self.all_coordinates = []
32 self.road_coordinates = []
33 self.fridge_coordinates = []
34 self.obstacles_coordinates = []
35
36 for y in range(len(self.maze)):
37     for x in range(len(self.maze[0])):
38         character = self.maze[y][x]
39         x_coordinate = -1*self.game_space / 2 + x*self.wall_size_pixel + self.wall_size_pixel/2
40         y_coordinate = self.game_space / 2 - y*self.wall_size_pixel - self.wall_size_pixel/2
41
42         self.all_coordinates.append((x_coordinate, y_coordinate))
43         self.obstacles_coordinates.append((x_coordinate, y_coordinate), character))
44
45         if character == "S":
46             self.fridge_coordinates.append((x_coordinate, y_coordinate))
47         if character == "O":
48             self.road_coordinates.append((x_coordinate, y_coordinate))
49         else:
50             self.obstacles_coordinates.append((x_coordinate, y_coordinate), character))
51
52 self.wall_coordinates = [coordinates for coordinates in self.all_coordinates if coordinates not in self.road_coordinates]

```

Fig. 3: Koordinat Belirleme

bir karenin büyüklüğünde o kareye göre belirlenmektedir. Oyun alanı içerisindeki her bir karenin büyüklüğü bu şekilde belirlendikten sonra her bir kareye ait koordinatlar Figure - 3'te gösterilen kod aracılığıyla tespit edilmektedir.

Problem 1 için oluşturulacak olan ızgaraya ait bilgiler belirli bir URL adresindeki txt dosyasından çekilmektedir. Bu txt dosyasında 0 olan rakamlar yolları, 1, 2 ve 3 rakamları ise engel çeşitlerini göstermektedir.

III. DENEYSEL SONUÇLAR

Proje kapsamında talep edilen birinci ve ikinci probleme ait çözüm yolları tümüyle tespit edilmiş olup, çeşitli simülasyonlar yapılmıştır. Problem 1 için kullanılan iki labirent tipinde de labirente rastgele olarak yerleştirilen robotun yine labirente rastgele olarak yerleştirilen hedefi bulması ve öğrendiği kareler içerisindeki en kısa yolu tespit etmesi başarılı bir şekilde simüle edilmiştir. Burada robot başlangıçta hedef koordinatları bilmediğinden etrafındaki karelerden uygun olana rastgele olarak gitmekte ve hedef koordinatları tespit ettiğinde durmaktadır. Sonrasında ise öğrencidği kareler içerisinde A* en kısa yol algoritmasını kullanarak en kısa yolu tespit etmektedir.

Problem 2'de ise başlangıç ve hedef koordinatları sırasıyla labirentin kuzeybatı ve güneydoğu köşeleri seçilmiştir. İstendiği takdirde bu seçimler değiştirilebilmektedir. Robot burada da hedef koordinatları bilmediğinden rastgele olarak ilerlemekte ve hedef koordinatı tespit ettiğinde A* en kısa yol algoritmasını kullanarak en kısa yolu tespit etmektedir.

SONUÇ

Bu projede bir ızgara üzerine yerleştirilen robotun engellere ve duvarlara takılmadan hedef ko-

ordinatları tespit ederek başlangıç ve hedef koordinatlar arasındaki en kısa yolu tespit eden bir uygulama geliştirilmiştir. Robot burada en kısa yolu tespit ederken öğrendiği kareleri kullanmaktadır. Projede python yazılım dili kullanılmıştır. Arayüz için ise tkinter ve turtle kütüphanelerinden faydalanılmıştır. Proje kapsamında talep edilen tüm problem çözümleri elde edilmiştir.

REFERENCES

- [1] stackoverflow.com
- [2] [geeksforgeeks.org](https://www.geeksforgeeks.org)
- [3] Kocaeli Üniversitesi Bilgisayar Mühendisliği Bilgisayar Labaratuarı Ders Notları

GEZGİN ROBOT PROJESİ UML DİYAGRAMI

