KOCAELİ ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
PROGRAMLAMA LABORATUVARI

Tuğba ÇEVİKER
210202056

Samethan KAZANBAŞ
210202043

I. ÖZET

Bu döküman Programlama Laboratuvarı II dersinin 1. Projesi olan Gezgin Robot Projesi'nin çözümünü açıklamaya yönelik oluşturmuluştur. Dökümanda Giriş, Yöntem, Deneysel Sonuçlar , Sonuç , Kaynakça ve Akış Diyagramı kısımlarına yer verilmiştir.

II. GİRİŞ

Projede bizden istenen C++,C,Java veya Python dilini kullanarak belirli kurallara göre hareket eden robotun engelleri aşarak istenen hedefe ulaşmasını sağlayan bir oyun tasarlanması beklenmektedir. Oyunda iki adet problem bulunmaktadır. Problemlerin çözümü için nesneye yönelik programlama ve veri yapıları bilgilerinin kullanılması beklenmektedir.

PROBLEM 1 Birinci problemde sizden robotu ızgara (grid) üzerinde verilen hedefe engelleri aşarak en kısa sürede ve en kısa yoldan ulaştırmanız beklenmektedir. Robotu tüm ızgarayı değil,yalnızca gerekli yolları gezerek hedefe ulaşmasını sağlamalısınız.

Adım 1:Verilen url okunup ordan çekilen matrisin boyutlarında karesel bir ızgara alanı oluşturmanız gerekmektedir.

Adım 2: Verilen url okunup ordan çekilen matrise göre ızgara üzerine engeller ve duvarlar yerleştirilmelidir. Engeller birbirinden farklı tipteki nesnelerden oluşabilir. (Verilecek text dosyasındaki 0 değeri engelsiz yollara; 1, 2, 3 değerleri ise üç farklı tipteki nesne için engelleri temsil edecektir. Birbirinden farklı sayıda karesel alan işgal eden bu üç engel nesnesinden 1 değerli nesne yalnızca 1 karelik alan 2 değerine sahip nesneler yanyana 2 kare içeren maksimum 2x2 lik; 3 değerine sahip nesneler ise yan yana 3 kare içeren maksimum 3x3 lük kare alana Şekil 1' deki gibi yerleştirilecektir.)

Adım 3: Robotun başlangıç ve hedef noktaları ızgara üzerindeki uygun (engel veya duvar içermeyen) karelere rastgele belirlenmelidir. Robot başlangıçta tüm ızgara dünyasını bilmemelidir, sadece bir adım sonraki kareleri görebilmelidir. Her adımda robotun öğrenmediği kareler bulutlu olarak gösterilmeli, öğrenilen kareler ise açılarak ilgili karelerde bulunan nesneye göre atanmalıdır. Adım 4: Tüm bu bilgiler doğrultusunda, robotun hedefe en kısa sürede ulaşabileceği en kısa yol, adım adım ızgara üzerinde gösterilmelidir. Robotun daha önce geçtiği yerler belli olacak şekilde boyanmalıdır. Hedefe ulaşıldığında ise başlangıç noktasından hedef konuma giden robota göre en kısa yol ızgara üzerinde ayrıca çizdirilmelidir. Geçen toplam süre ve kaç kare üzerinden geçildiği ekranda gösterilmelidir.

PROBLEM 2 Adım 1: Kullanıcıdan alınan boyutlarda bir ızgara oluşturmanız gerekmektedir. Adım 2: Izgara üzerine 1 nolu tipte engeller yerleştirilerek labirent oluşturulmalıdır. Labirent içerisinde mutlaka çıkışa ulaşamayan yollar bulunmalıdır. Adım 3: Labirentin giriş ve çıkış noktaları dörtgen ızgaranın herhangi çapraz 2 köşesi olarak belirlenmelidir. Robot başlangıçta labirenti bilmemelidir. Labirentte yanlış girilen bir yol algılandığında robotun doğru olarak tespit ettiği en son konuma giderek buradan itibaren yol aramaya devam etmesi gerekmektedir. Adım 4: Tüm bu bilgiler doğrultusunda, robotun çıkışa ulaşmak için izlediği yol adım adım ızgara üzerinde gösterilmelidir. Her adımda robotun daha önce geçtiği yollar boyanarak gösterilmelidir. Robot hedefe ulaştığında giriş noktasından çıkış noktasına giden yol ızgara üzerinde çizilmelidir. Geçen toplam süre , kaç kare üzerinden geçildiği ekranda gösterilmelidir.

III. YÖNTEM

Uygulamayı çalıştırdığımızda oluşturduğumuz 'oyunarayuz' adlı classımızı çağırarak nesnemizi oluşturuyoruz ve JFrameden kalıtılan bu nesnemizin setVisibleını 'true' yapıyoruz. Böylece ilk ara yüzümüz ekrana geliyor.Bu ara yüzde iki tane buton bulunmaktadır.Bunlar : Problem 1 ve Problem 2 butonlarıdır.Eğer kullanıcı problem 1 butonuna basarsa oluşturuduğumuz 'oyunarayuz1' adlı classımızı çağırarak nesnemizi oluşturuyoruz ve JFrameden kalıtılan bu nesnemizin setVisibleını 'true' yapıyoruz.Ardından oyunarayuz1 ekranımız açılıyor.Bu ekranda da iki buton bulunmaktadır .Bunlar: URL 1 ve URL 2 butonlarıdır.Eğer kullanıcı URL 1 butonuna tıklarsa main methodumuzun bulunduğu uygulama classındaki okuma1 methodu çağırılır.Eğer kullanıcı URL 2 butonuna tıklarsa main methodumuzun bulunduğu uygulama classındaki okuma2 methodu çağırılır.

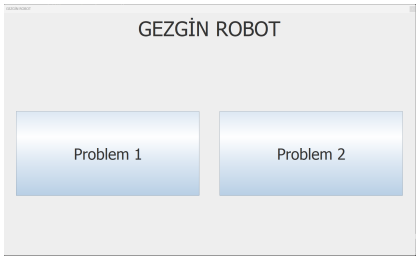

Fig. 1. Örnek Bulutlu Labirent

Okuma1 ve Okuma2 methodları: Bu iki methodda da bize verilen url adresleri string türünde kesintisiz okunduktan sonra matris boyutu kadar bölünerek bir matris integer türünden doldurulur.Ardından labirentin etrafına duvar atayabilmek için oluşturulan bu matrisin etrafını birlerle çevirdik. Başlangıç ve bitiş noktalarının atamasını yaptık.Ardından oluşturduğumuz bu matrisi,boyutunu ,false değerini ve okuma1 methodu için bir ,okuma2 methodu için 2 değerini ızgara nesnesi oluşturup kurucu methoduna bu değerleri gönderdik.

Kullanıcı Problem 2 methoduna basarsa main methodumuzun bulunduğu uygulama classındaki okuma3 methodu çağırılır.

Okuma3 methodu: Bu methoda ilk olarak kullanıcıdan labirent boyutunu istedik.Ardından bu boyutlarda bir matris oluşturuduk.Oluşturduğumuz bu matrisin bütün indislerine bir değerini atadık.Ardından matrisi ve boyutunu GenerateMaze methoduna gönderdik. GenerateMaze methodu: Yazdığımız bu method yanlış çıkışlarıda olan bir labirenti oluşturan özyinelemeli bir methodtur.Methodun mantısı ziyaret edilmemiş komşu hücrelerin yani duvarların listesini oluşturur ve özyinelemeli olarak dört eksende gezerek listeyi boişaltmaya çalışır ve yolları birbirine bağlar.Böylece elimizde birlerin duvarları sıfırların yolları temsil ettiği bir matris bulunur. GenerateMaze methodundan gelen matirisin etrafını duvarlarla kaplamak için bu matrisin etrafı birlerle çevirilir. Ardından oluşturduğumuz bu matrisi,boyutunu ,true değerini ve üç değerini ızgara nesnesi oluşturup kurucu methoduna bu değerleri gönderdik.

Izgara classı: Izgara nesnesi çağırıldığında bu classsın kurucu methodunda bir tane Frame oluşturulur.Bu Frame üç farklı panelden oluşuyor bunlar:labirentin gözükeceği gridpanel , butonların gözükeceği butonpanel ve bu iki paneli bir arada tutan mainpanel. Izgara adına GridLayout türünden bir nesne daha oluşturulur ve bu nesneye boyutlar gönderilir.Panel türünden nesneler tutan bir matris oluşturulur ve boyutları belirlenir.Urlden okuduğumuz ya da rastgele oluşturduğumuz matrisin bütün indislerini gezebileceğimiz iç içe for göngüsü oluşturulur. Her döngüde yeni bir panel nesnesi oluşturulur.Eğer matirisin indisindeki değer 1 ise burası duvardır ve panelin arka planına gri atanır.Panel türünden nesneleri tutan matrisimizin bu indisine panel eşitlenir ve bu panel GridPanele add methoduyla eklenir. Eğer matirisin indisindeki değer 2 veya 3 ise bu indislere rastgele duvar ataması gerekmektedir.Random methodu sayedinde bir veya iki değeri atanır.Bir atandıysa burası duvar iki atandıysa yol olarak tanımlanıp maviye boyanır.Eğer matirisin indisindeki değer 0 ise burası yoldur ve panelin arka planına mavi atanır.Panel türünden nesneleri tutan matrisimizin bu indisine panel eşitlenir ve bu panel GridPanele add methoduyla eklenir.Eğer matirisin indisindeki değer 4 ise burası başlangıç noktasıdır ve panelin arka planına kırmızı atanır.Panel türünden nesneleri tutan matrisimizin bu indisine panel eşitlenir ve bu panel GridPanele add methoduyla eklenir.Eğer matirisin indisindeki değer 9 ise burası bitiş noktasıdır ve panelin arka planına pembe atanır.Panel türünden nesneleri tutan matrisimizin bu indisine panel eşitlenir ve bu panel GridPanele add methoduyla eklenir.Labirentin setVisiblei true yapılır ve labirent ekranda gözükür.Bir robot nesnesi oluşturulur ve robotun gezdiği her bir indisi bize veren search.path methodu çağırılır.Bu methoda başlangıç noktamızı , boş bir ArrayList ve matris gönderilir. Özyinelemeli olan bu method ilk önce bitiş noktasında gelip gelmediğini kontrol eder.Eğer gelmemişse robotumuz bizim belirlediğimiz sırada dört eksen boyunca önündeki indis yolsa hareket eder ve bu indisleri pathe ada methoduyla ekler.Bu metodun batığı depth first search algoritmasıyla bağlıdır.Search path methodundan gelen ve robotumuzun izlediği yolları elinde tutan ArrayList oluşturulmuştur.Oluşturduğumuz GridPanel nesnesi add methoduyla MainPanele eklenir.Üç yeni buton tanımlanır. Bunlar: çalıştır , sonuç göster ve labirent değiş butonlarıdır.Eğer kullanıcı çalıştır butonuna basarsa ilk önce tüm labirent beyaza boyanır.Ardından thread nesnesi sayesinde search path methodundan gelen indisler turuncuya etrafındaki indisler ise duvarsa griye yolsa maviye adım adım boyanır.Eğer kullanıcı sonuç göster butonuna basarsa labirent ilk halinde boyanır sadece search pathten gelen indisler turuncuya boyanır.Böylece bulutsuz labirent görüntüsü elde edilir.Ayrıca labirent çözme süresi ve geçilen kare sayısı hesaplanır ve ekrana yazdırılır.

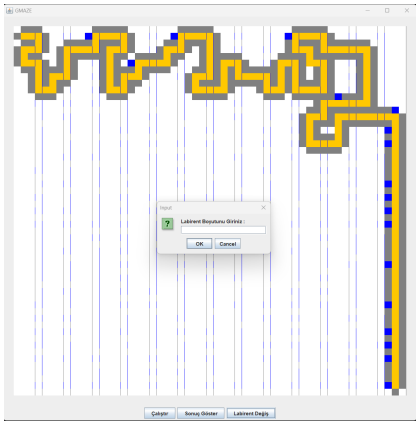

Fig. 2. Bulutlu Labirent ve labirent değiş butonu

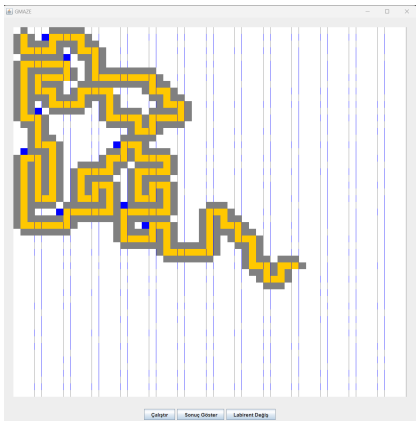

Fig. 3. Örnek Bulutlu Labirent

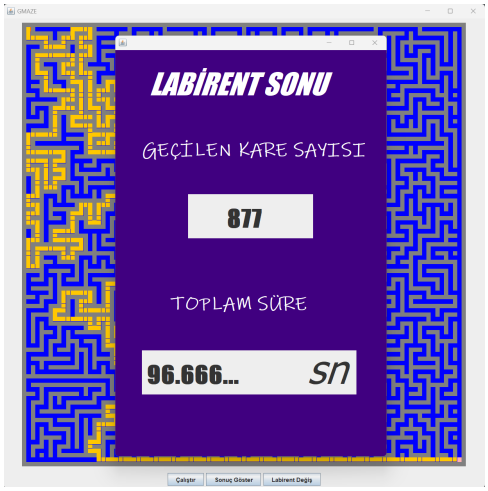

Fig. 4. Sonuç Ekranı

https://www.overleaf.com/project/6419b1abecbe4f721793c6df

IV. DENEYSEL SONUÇLAR

Bize verilen url uzantılarını okuyabildik ve bir matrise atayabildik.

Bu matrisin de yardımıyla indisinde 0 bulunan yerleri yol 1 olan yerleri duvar olarak atayabildik.

Problem 1 için indisinde 2 ve 3 olan yerlerde Random methodunu kullanarak rastgele duvar ataması atayabildik.

Problem 2 için kullanıcı tarafından girilen boyut değerinde bir matris oluşturabildik ve bu matrisi yazdığımız algoritma sayesinde bitiş noktasına ulaşmayan yollar da ekleyerek oluşturbildik.

Her iki problem içinde oluşturduğumuz robot hareket algoritması sayesinde robotumuz matriste her zaman yol olan indisleri tercih ederek bitiş noktasına ulaşmayı başardı.

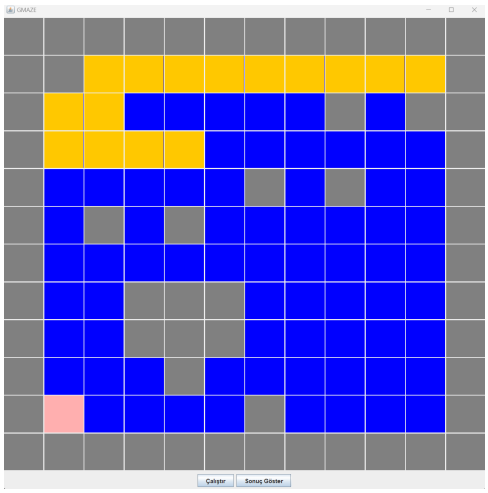

Fig. 5. Problem 1-Labirent

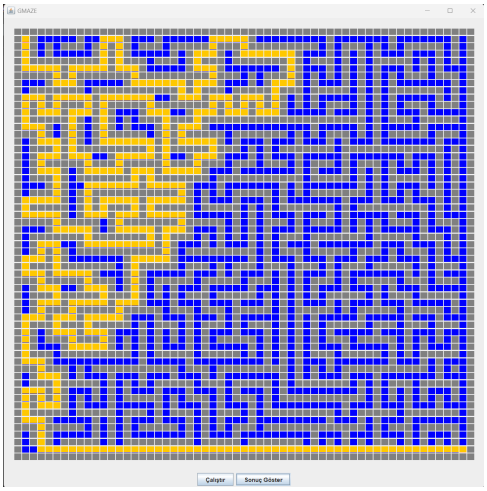

Fig. 6. Problem 2-Labirent

V. SONUÇ

Sonuç olarak bu proje sonunda, GridLayout yapısını,JFrame yapısını,Nesneye Yönelik Programlama mantığını daha iyi kavradık. Bunun yanı sıra labirent oluşturma kurallarını,özyinelemeli methodların mantığını ve Depth First Search Algoritmasının mantığını daha iyi anladık.

VI. KAYNAKÇA

https://stackoverflow.com/questions/36161146/maze-path-finder-in-java

https://iq.opengenus.org/maze-generator-in-java/

https://www.youtube.com/watch?v=dyrvXiMumXc

https://www.geeksforgeeks.org/different-ways-reading-text-file-java/

https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html

https://www.geeksforgeeks.org/java-program-for-rat-in-a-maze-backtracking-2/

https://www.youtube.com/watch?v=$_K EzW50Nhyw$

https://www.mehmetkirazli.com/java-random-sinifi/

VII. UML SINIF DİYAGRAMI

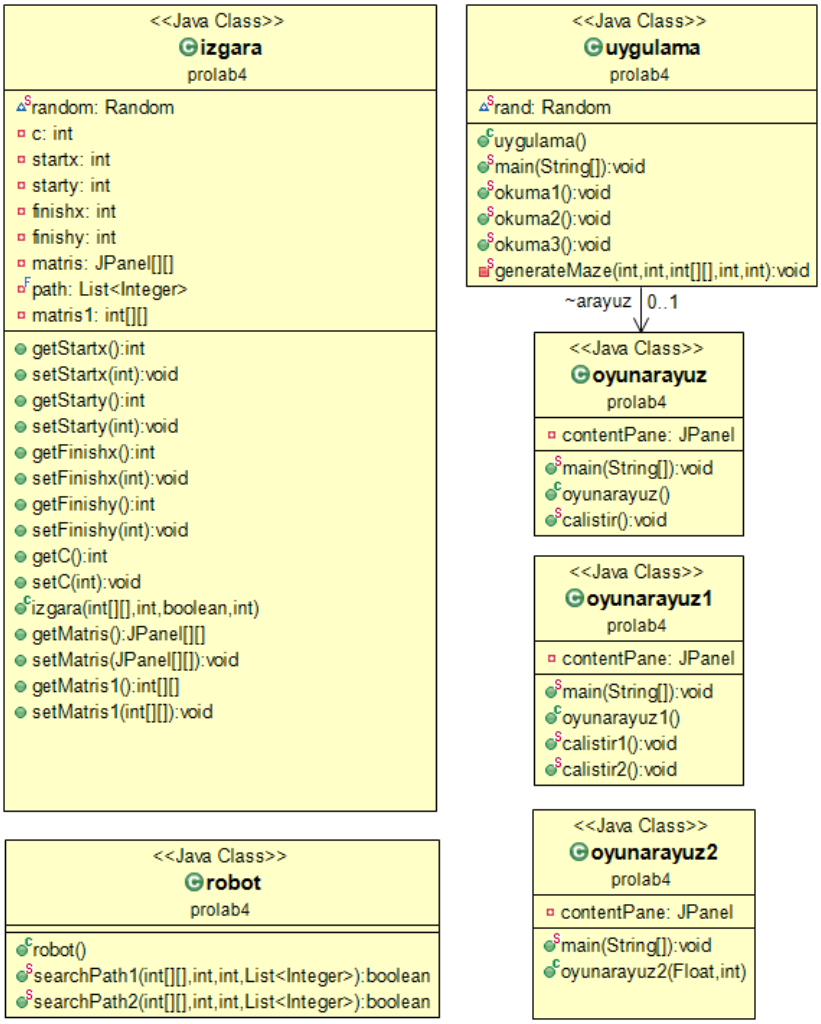

Fig. 7. UML Sınıf Diyagramı