LABİRENT PROJESİ

1st Yalçın Dağbaşı
Kocaeli üniversitesi

Kocaeli/Türkiye
210202040

2nd Emir Çınar
Kocaeli üniversitesi

Kocaeli/Turkiye
210202014

I. ÖZET

Veri Yapılarının temelleri ve nesne yönelimli programlama kullanılarak bir labirent oyunu tasarlanması istenmiş ve belli başlı problemlere çözüm bulunması istenmiştir.

II. PROJE TANIMI

Belirli kurallara göre hareket eden bir robotun önündeki engelleri aşarak istenen hedefe ulaşmasını sağlayan bir oyun tasarlanması beklenmektedir. Oyunda iki adet problemin çözülmesi gerekmektedir. Problemlerin çözümü için nesneye yönelik programlama ve veri yapıları bilgilerinin kullanılması beklenmektedir.

A. PROBLEM 1

Bu problemde sizden robotu ızgara (grid) üzerinde verilen hedefe engellere takılmadan en kısa sürede ve en kısa yoldan ulaştırmanız beklenmektedir. Robotu tüm ızgarayı değil, yalnızca gerekli yolları gezerek hedefe ulaşması sağlanmalıdır.Izgara üzerine engeller ve duvarlar yerleştirilmelidir. Izgara boyutu, engel sayısı ve engellerin konum bilgileri içeriği matris biçimindeki bir text dosyasından alınacaktır.Robotun hedefe en kısa sürede ulaşabileceği en kısa yol, adım adım ızgara üzerinde gösterilmelidir. Robotun daha önce geçtiği yerler belli olacak şekilde her adımda yol üzerinde iz bırakması gerekmektedir

B. PROBLEM 2

Bu problemde robotu labirentteki çıkış noktasına ulaştırmanız beklenmektedir. 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.Tüm bu bilgiler doğrultusunda, robotun çıkışa ulaşmak için izlediği yol adım adım ızgara üzerinde gösterilmelidir. Her adımda robotun daha önce geçtiği yollar üzerinde iz bırakması gerekmektedir. Robot hedefe ulaştığında giriş noktasından çıkış noktasına giden yol ızgara üzerinde çizilmelidir. Geçen toplam süre (sn cinsinden), kaç kare üzerinden geçildiği bilgileri ekranda gösterilmelidir.

III. IZGARA SINIFI

Bu sınıfta labirent için oluşturulması gereken ızgara alanı oluşturulması amaçlanmıştır. Problem birde ifade edilen url değiştir butonu için kullanılan özellikler bu sınıfın özellikleri olarak kullanılmıştır. Bu sınıfta url'den veri okumak için de string özellikler tanımlanmıştır. Bu sınıfın kurucu metodunda bu ve diğer sınıflarda kullanılmak üzere url adresinden satır bilgisi okunmuştur. Ayrıca bu kurucu metodda ızgara alanının matris halindeki hali ve JavaSwing görsel uygulamalarda gösterilmek üzere JPanel matris hali de tanımlanmıştır.

A. Public static void dosyaOku()

Bu metotta problem 1 için verilecek url adresinden veri okuma işlemi yapılmıştır. Okunan veriler labirent adlı iki boyutlu dizinin içine atılmıştır. Böylece görsel uygulamalar ve gerekli işlemler için kullanılacak veriler bu boyutta kayıt altına alınmıştır.

B. Public static int[][] etrafinaDuvarCek()

Problem 1 için yol ve engellerin bulunduğu url adresi paylaşılırken labirent ekranının etrafına bir duvar çekilmesi gerektiği not düşülmüştü. Dosyaoku() metodunda verileri okuyup verileri bir matris içine attıktan sonra sıra etrafına duvar çekme işlemine geldi. Duvar işlemi için ekstra 2 satır ve 2 sütun çekme işlemi altta verilen matriskopyala metodunda yapılmıştır. Gerekli indexlere 1 verisi verilerek etrafına duvar çekme işlemi yapılmıştır. Bu metod adından da anlaşılacağı üzere geri değer döndüren bir metottur.

C. Public static int[][] matrisKopyala()

Hemen yukarıda bahsedilen etrafına duvar çekme işlemi yapılmadan önce geçici bir matris burda oluşturulmuş ve duvar çekilecek yerleri boşta kalacak şekilde kalan labirent verileri burda geçici matrise kopyalanmış; üzerine duvar eklenmesi işlemi için etrafinaDuvarCek() metoduna yollanmıştır.

IV. KARELIEKRAN SINIFI

Swing işleminin bu sınıfta yapılması için özellik olarak frame eklenmiştir. Ayrıca Izgara sınıfında kullanılan labirent adlı iki boyutlu dizi de bu sınıfta tanımlanmıştır. Bu sınıfın kurucu metodunda JavaSwing işlemleri gerçekleştirilmiş ve bu sınıftan bir nesne oluşturulunca diğer verilerin otomatik olarak

yüklenmesi için Izgara sınıfında oluşturulan dosyaoku() , matriskopyala() metodları ve biraz sonra aşağıda okuyacağınız kareOlustur() metodu burada çağrılmıştır.

A. public static void doldur()

url adresinden okunan veriler labirent adlı matrise kayıt edilmişti. Yol ve duvarların renklerini belirleyip gerekli atamalar burda yapılmış; rengi belirlenen her JPanel frame'ye eklenmiştir. Ayrıca bu metotta her karenin etrafına çizgi çekilip kare şeklinin oluşması için BorderFactory.createLineBorder() metodu kullanılmıştır.

B. Public static void KareOlustur()

Bu metotta swing işlemleri için frame boyutu ve frame içerisine eklenecek Gridlayout tanımlanmıştır. Gridlayout kullanma sebebimiz,girilen boyuta göre kendiliğinden ızgara alanını oluşturmasıdır. Bu metotta doldur() metodu çağrılmıştır.

C. Public static void BaslangicBelirle(int sayi)

Problem 1'de başlangıç ve hedef noktasının rastgele olarak belirlenmesi koşulu belirtilmiştir. Bu metotta başlangıç yeri belirlenmiştir. Bu ve alttaki metot oluşturulurken başlangıç ve bitiş noktalarının url'den okunan engel veya duvarlara denk gelmemesine dikkat edilmiştir. Başlangıç yeri siyah renk ile boyanmıştır

D. Public static void bitisBelirle()

Yukarıda belirtilen başlangıç belirle ile aynı mantığa sahiptir. Bitiş noktası magenta rengi ile boyanmıştır

E. LabirentBFS sınıfı

Bu sınıfın kurucu metodunda KareliEkran sınıfından bir nesne oluşturulmuştur. Ayrıca tüm panellerin görünürlükleri setVisible metoduyla kapatılmıştır.

F. Public static ArrayList BFS()

Bu metotta başlangıç ve hedef noktası arasındaki en kısa yolu bulmak için BFS Algoritması kullanılmıştır. BFS Algoritması bir grafikteki düğümleri ve bağlantıları ziyaret etmek için kullanılan bir arama algoritmasıdır. Bu algoritma, bir grafikte belirli bir başlangıç düğümünden başlar ve grafikteki tüm düğümleri ve bunlar arasındaki bağlantıları keşfeder. BFS Algoritması Kuyruk veri yapısını kullanır. İlk giren İlk çıkan (FIFO) mantığına göre çalışır. Bu metotta başlangıç noktası kuyruk içerisine offer() metodu ile atılarak işlemlere başlandı. Kuyruk boş kalana kadar, her düğümün komşuları ziyaret edilerek hedef noktası bulunmaya çalışılır. Kuyruktan çıkarılan her düğüm için hedef düğüm ile aynı mı kontrolü yapılır. Aynı değil ise o düğümün yol olan tüm komşu düğümleri kuyruğa eklenir. Komşularına bakıldıktan sonra o düğüm kuyruktan çıkarılır. Gezilen noktalar bir arrayliste eklenir.

G. Public static boolean bitisGorunduMu()

Robot her ilerlediğinde bir adım sonraki kutucuklar açılacaktır. Bitiş yeri göründüğünde daha fazla dolaşmaması adına bu metot oluşturulmuştur. Boolean ifadesi döndüren bu metot farklı metotlarda kullanılmak üzere çağrılmıştır.

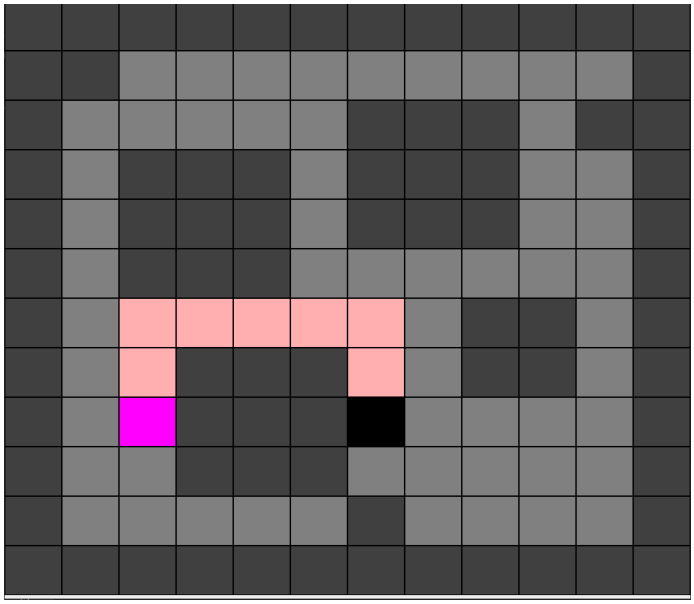

Fig. 1. En Kısa Yolun BFS Algoritması ile bulunması

H. Public static void bitiseGit()

Robotun her adımında bir sonraki adım ihtimallerinin olduğu yerlerinin görüneceğini söylemiştik. Bu metot, bitiş noktası göründüğünde robotun direkt olarak hedefe ulaşması için oluşturulmuş bu sayede daha az adımda işlemin bitirilmesi öngörülmüştür. Sırayla sağ,sol,yukarı ve aşağı olmak üzere 4 farklı ihtimal bu metot içerisinde tanımlanmış ve gerekli işlemler yapılmıştır.

I. Public static void etrafiniBoya()

Daha önce de bahsettiğimiz gibi ekran en başta tamamen bulutlu ve robot hareket ettikçe gitme ihtimalinin bulunduğu paneller teker teker görünür vaziyete getirilmiştir. İşte sürekli bahsettiğimiz bu işlemin yapıldığı metot etrafiniBoya() metotudur. Bu metot ilgili konum adresini parametre olarak alıp o konumun sağı,solu,yukarı ve aşağı panelini görünür hale getirmiştir. Ayrıca bu metotta enYakinNoktaBul() metodunda kullanılmak üzere görünür hale getirilen her panelin konumu da bir liste kaydedilmiştir.

J. Private Static void EkrandaYerGoster()

Bu metot görsel açıdan hangi aşamada hangi panelin boyanacağını gösteren metottur. Bu metotta Thread.sleep() metodu kullanılarak işlemler arasında zaman bırakıp işlemlerin daha görünür hale getirilmesi amaçlanmıştır.

K. Public static void BaslangicBoya()

Bu metotta Başlangıç paneli siyah renk ile boyanmış ve görünürlüğü 'false' haline getirilmiştir. Public static void baslangicBitisBelirle() Problem 1 için başlangıç ve bitiş renkleri rastgele olacak şekilde olcağını konuşmuştuk. KareliEkran sınıfındaki ilgili metotlara erişim bu metot ile gerçekleştirilmiştir

L. Public static void bitisBoya()

Robotun geçtiği tüm noktalar mavi boya ile boyanmıştır. Robot bitiş hedefini ararken başlangıç noktasından geriye doğru tekrar geçebilme ihtimaline karşın başlangıç noktasının siyah renkten mavi renge dönme sorunu ortaya çıkmıştır. Bu sorunu ortadan kaldırmak için tüm işlemler bittiğinde başlangıç ve bitiş renkleri bu metotta eski hallerine döndürülmüştür.

M. Public static boya()

BFS metotunda en kısa yol pembe renge boyanmıştır. Pembe renge boyanacak her hücre bu metota yollanmış ve rengi pembeye boyanmıştır.

N. Public static void tumEkraniGoster()

Tüm işlemler bittikten bir süre sonra labirentin tamamını göstermek için bu panel oluşturulmuştur.

O. Public static void sifirla()

Görsel arayüz için oluşturulan menüde oyuna başla butonuna birden fazla tıklandığında oyunun önceki verileri silinmeyip yeni oyun verileri eski oyunun üzerine eklenip sonuç olarak istenmeyen sonuç ve başarısız bir yürütme ortaya çıkarmıştır. Bu sorunun önüne geçmek için menünün oluşturulduğu sınıfta oyunabaşla butonuna her basışta ortaya çıkan aksiyonların en başına bu metot eklenerek gerekli işlemler sıfırlanmış böylece yeni oyun verileri oluştururken eski oyun verilerinin programı bozması engellenmiştir.

P. Public static int[] enYakinNoktaBul()

Bu metot robotun etrafını gezilmiş noktalar sardığında gezilmemiş en yakın noktayı bulması için oluşturulmuştur. etrafiniBoya() metodunda görünürlüğü açılan her noktanın bir liste kaydedildiğini söylemiştik. Bu metotta da bu listenin içinde dolaşarak o anki konuma en yakın gri renkli(gezilmemiş) noktanın bulması amaçlanmıştır. En yakın noktanın bulunması işlemi analitik geometride kullanılan iki nokta arasındaki uzaklık formülünden yararlanmıştır. Bu metot en yakın noktanın koordinatlarını döndüren bir dizi şeklinde tanımlanmış ve kullanılması üzerine rastgeleDolas() metoduna yollanmıştır.

Q. Public static void rastgeledolas()

Robotun hareket ederken hedef noktasını bilmemesi ve rastgele bir şekilde kareleri dolaşarak hedefi bulması istenmiştir. Bu işlem gerçekleştirilirken hedefi bulana kadar işlem gerçekleştiren bir döngü kurulmuştur. Bu metodun çalışma mantığı,robotun bulunduğu hedefin bir sonraki adımında gideceği yerler arasında gezilmemiş nokta varsa önce oraya hareket etmesi, gezilmemiş birden fazla nokta varsa rastgele bir şekilde hareket etmesi amaçlanmıştır. Robot bir noktaya geldiğinde hemen bir sonraki adımında ziyaret etmemiş bir noktanın olmama olasılığı vardır. Bu sorunu çözmek için de robotun etrafında gezilmemiş bir nokta olmaması halinde robotun o anki bulunduğu konuma en yakın gezilmemiş noktanın koordinatlarını veren enYakinNoktaBul() metodu kullanılmıştır. Bu şekilde robotun doğru hedefi bulma noktasındaki sürenin aza indirgenmesi ve doğru hedefi bulma olasılığının artması amaçlanmıştır.

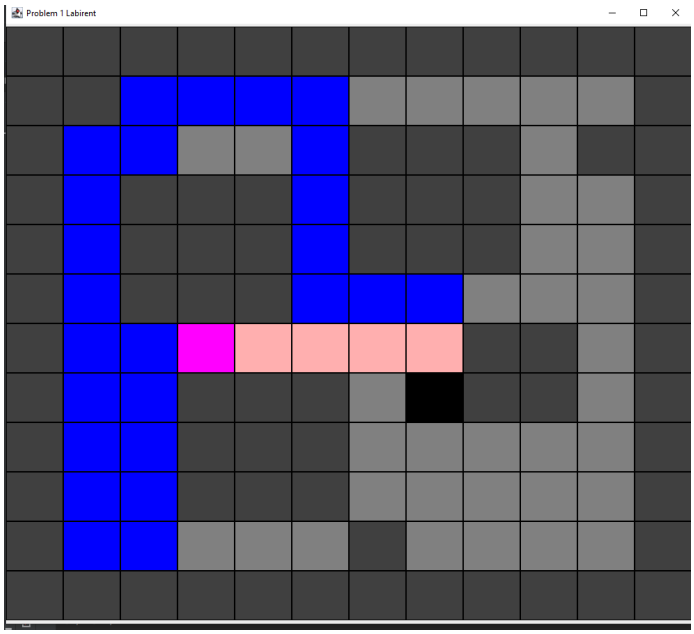

Fig. 2. Robotun rastgele dolaşarak hedefi araması

V. NewAnaEkran ve oyun sinifi

Oyun sınıfı herhangi bir nesne oluşturmak için değil oyunabasla butonundaki aksiyonların oluşturulması için sadece kurucu metoda sahip bir sınıftır. Bu sınıfta menü oluşumu ve ilgili butonlara basıldığında raporda açıklamasının yapıldığı sırada belirtilen yerlerdeki metotlar çalıştırılmıştır.

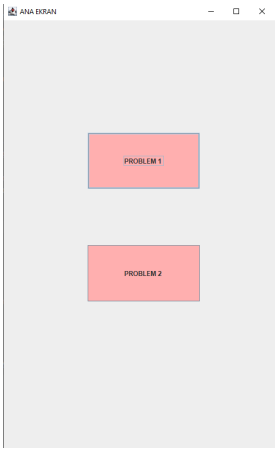

Fig. 3. Problem Seçme Menüsü

VI. MazeCreator Sinifi

Bu sınıf problem2'de labirent oluşturulması için kullanılmıştır

A. Private void generateMaze()

Bu metot, bir labirent oluşturmak için kullanılmıştır. Başlangıçta rastgele bir hücre seçilir ve o hücre ”0” olarak işaretlenir. Ardından, bir yığın oluşturulur ve başlangıç hücresi yığına eklenir. Yığın boş olmadığı sürece, yığının en üstündeki hücre alınır ve bu hücrenin henüz ziyaret edilmemiş komşuları bulunur. Eğer en az bir komşusu varsa, rastgele bir komşu seçilir ve seçilen komşunun ve mevcut hücrenin arasındaki duvar kaldırılır ve komşu hücre ”0” olarak işaretlenir. Seçilen komşu yığına eklenir ve işlem tekrarlanır. Eğer mevcut hücrenin hiç ziyaret edilmemiş komşusu yoksa, mevcut hücre yığından çıkarılır ve işlem yine tekrarlanır. Bu işlem yığında kalan hücre kalmayıncaya kadar devam eder.

B. Private list¡int¿ ziyaretEdilmeyenKomsuBul()

Bu metot, verilen koordinatlarda bulunan bir hücrenin ziyaret edilmemiş komşularını bulmak için oluşturulmuştur. Komşular,projede verildiği gibi kuzey, güney, batı ve doğu yönlerinde bulunabilir. Metot, komşuların bulunduğu koordinatları içeren bir liste döndürür. Hücrenin ziyaret edilmemiş bir komşusu yoksa, boş bir liste döndürür. Ayrıca, hücrenin sınırda olması durumunda, komşuların olup olmadığını kontrol etmek için uygun sınır koşulları da kontrol edilir.

VII. Deneme Sinifi

Bu sınıfın kurucu metodu bir labirent (maze) oluşturur. Metot, bir genişlik ve yükseklik parametresi alır ve bu değerlere göre bir iki boyutlu tamsayı dizisi oluşturur. Dizi elemanlarına ilk olarak 1 değeri atanır. Daha sonra generateMaze() adlı başka bir metot çağrılır ve bu metot, rastgele olarak seçilen duvarları kırarak labirenti oluşturur. Son olarak, diziye bir çerçeve eklenir, böylece labirentin dış duvarları tamamlanmış olur.

A. Private void generateMaze()

Bu metod, bir labirentin içindeki duvarları kırarak rastgele bir şekilde labirenti oluşturur. Bu metot bir DFS algoritmasını kullanarak labirenti oluşturur. Başlangıç noktası rastgele seçilir ve bir yığına eklenir. Yığındaki her eleman, içinde bulunduğu hücrenin komşularından birini rastgele seçer ve aradaki duvarı kırarak labirenti oluşturmaya devam eder. Bu işlem, yığın boşalana kadar devam eder. Sonuç olarak, labirentin içindeki duvarlar kırılarak, bir çözüm yolunu bulmak için kullanılabilecek bir labirent oluşturulur.

B. Private void getVisitedNeighbors()

Bu metod, verilen x ve y koordinatlarındaki bir hücrenin, henüz ziyaret edilmemiş olan komşularını döndürür. Metot, bir Liste nesnesi içinde komşuların koordinatlarını tutar. Bu metot, generateMaze() metodunun bir parçasıdır ve bu metot, DFS algoritmasının bir parçası olarak kullanılır. DFS algoritması, ziyaret edilmemiş hücreleri keşfederken bu metodu çağırarak, bir hücrenin komşularını kontrol eder. Koordinatlar, labirentin dışına çıkmayacak şekilde sınırlandırır.

VIII. Arayüz Sinifi

Bu sınıf genel itibarı ile problem 1 için arayüz oluşturur aynı zamanda problem 1 için yol bulma metodu olan yolbul() metodunu içerir.

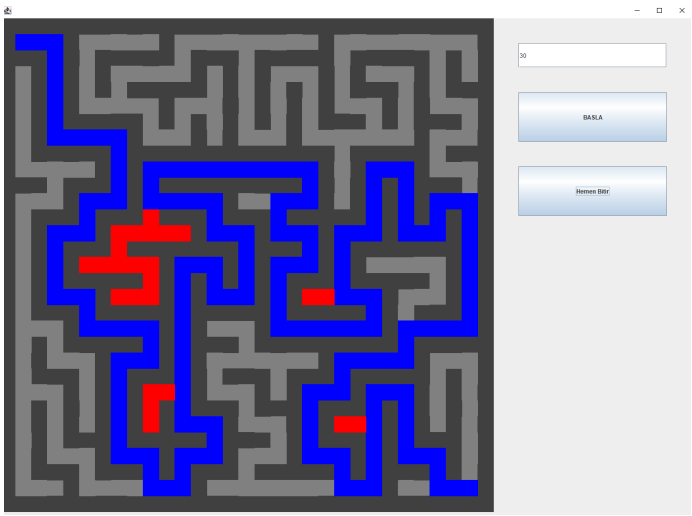

Fig. 4. Problem1’de hedefin aranması yolu

IX. Public void yolBul()

Bu metot, bir labirentin çözümü için derinlik öncelikli arama algoritmasını (DFS) kullanarak bir yol bulur. İki parametre alır: satirkonum ve sutunkonum, hangi hücreden başlayacağını belirtir. Bu metot, bir hücreyi ziyaret eder ve eğer labirentin sonuna ulaşmışsa işlemi sonlandırır. Aksi takdirde, labirentin sağ, sol, üst ve alt yönlerindeki komşu hücrelerini kontrol eder ve ziyaret edilmemiş her bir hücreyi tekrar ziyaret etmek üzere yolBul metotunu çağırır. Bu işlem, bir yol bulunana kadar devam eder ve bulunan yolu mavi hücrelerle işaretler.

X. Sonuçlar

- BFS VE DFS Algoritmaları hakkında daha somut bilgiler elde edip nerelerde kullanmamız gerektiğini anladık.
- A* gibi sezgiye yönelik en yakın yol bulma algoritmaları hakkında bilgi edindik
- Url’den veri okunurken karşımıza çıkan sorunlar hakkında bilgi sahibi olduk

XI. KAYNAKÇA

- https://bilgisayarkavramlari.com/2008/11/13/derin-oncelikli-arama-depth-first-search/
- https://bilgisayarkavramlari.com/2008/11/13/sig-oncelikli-arama-breadth-first-search/
- https://stackoverflow.com/questions/21815839/simple-java-2d-array-maze-sample
- https://www.javatpoint.com/java-get-data-from-url

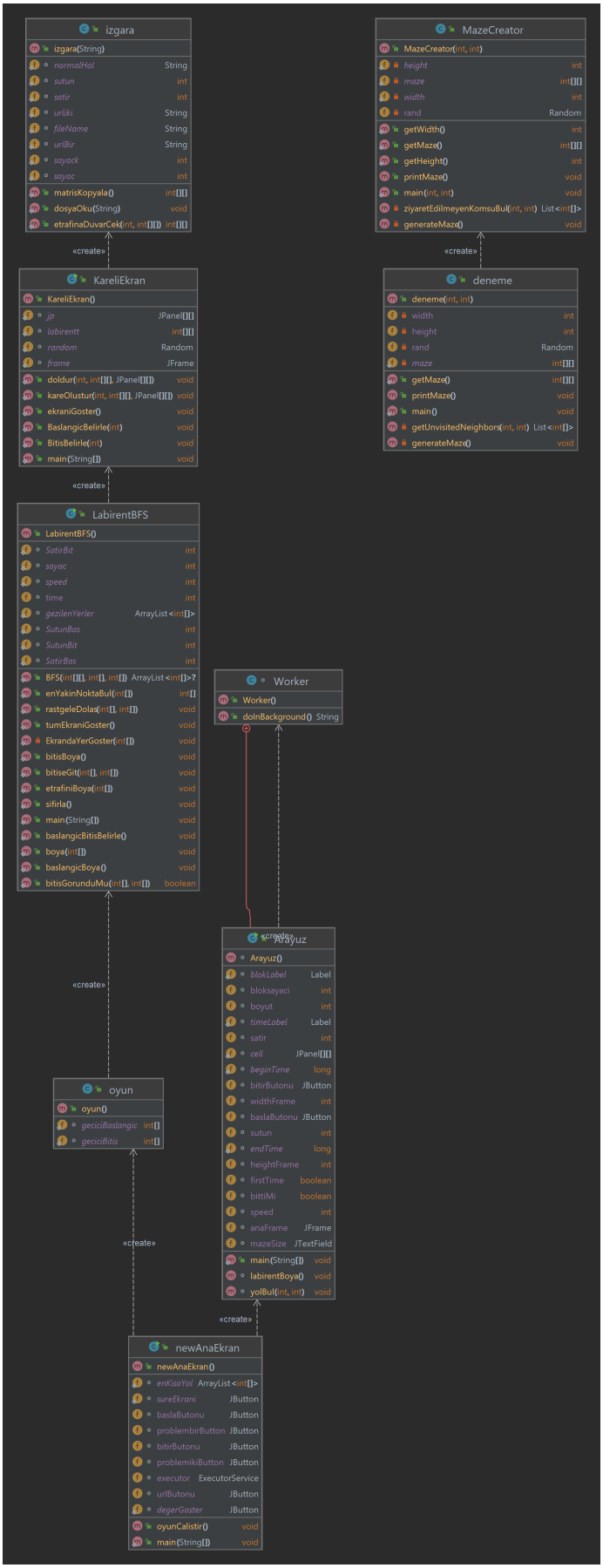

Fig. 5. UML DİYAGRAMI