Kocaeli Üniversitesi - Programlama Laboratuvarı 2 Proje 1 - Gezgin Robot

Göksel OKANDAN
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye
210201058@uzem.education

***Abstract*—Bu döküman Kocaeli Üniversitesi, Bilgisayar Mühendisliği Fakültesi, Programlama Laboratuvarı II dersinin 1. projesi olan Gezgin Robot projesinin raporudur. Bu dökümanda projenin özeti, tanımı, çözüme yönelik yapılan araştırmalar, kullanılan yöntemler ve algoritmalar, projenin hazırlanmasında kullanılan geliştirme ortamı hakkında bilgiler, karşılaşılan sorunlar gibi program hakkında bilgiler bulunmaktadır.**

***Index Terms*—labirent, çözüm, algoritma, arayüz, matris**

I. Özet

Bu döküman Kocaeli Üniversitesi, Bilgisayar Mühendisliği Fakültesi, Programlama Laboratuvarı II dersinin 1. projesi olan Gezgin Robot projesinin raporudur. Bu dökümanda projenin özeti, tanımı, çözüme yönelik yapılan araştırmalar, kullanılan yöntemler ve algoritmalar, projenin hazırlanmasında kullanılan geliştirme ortamı hakkında bilgiler, karşılaşılan sorunlar gibi program hakkında bilgiler bulunmaktadır.

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

Bu proje Java dili kullanılarak gerçekleştirilmiştir.

II. Giriş

A. Java Nedir? Ne Amaçla Kullanılır?

Java, 1995 yılında Sun Microsystems tarafından yayınlanmış bir hesaplama platformu ve programlama dilidir. Basit başlangıçlar ile ortaya çıkmıştır ve birçok hizmet ve uygulamanın oluşturulduğu güvenilir platformu sağladığı için bugünün dijital dünyasında büyük paya sahiptir. Gelecekte kullanmak üzere tasarlanmış yeni, inovatif ürünler ve dijital hizmetler de Java'yı temel almaya devam etmektedir. [1]

Kullanımı ücretsiz ve çok yönlü bir dil olması nedeniyle Java, yerelleştirilmiş ve dağıtılmış yazılımlar oluşturmada kullanılmaktadır. Mobil oyunlar ve bilgisayar oyunları dâhil birçok popüler video oyunu Java'da oluşturulmaktadır. Makine öğrenimi veya sanal gerçeklik gibi gelişmiş teknolojilerin kullanıldığı modern oyunlar bile Java teknolojisiyle oluşturulmaktadır. Aynı zamanda Java, WORA [Write Once and Run Anywhere (Bir Kez Yazın ve Her Yerde Çalıştırın)] felsefesine uygun yapısı sayesinde, merkezi olmayan bulut tabanlı uygulamalar için ideal seçimdir. Bulut sağlayıcıları, programlarını çok çeşitli platformlarda çalıştırmak için Java dilini seçmektedir. [2]

B. Labirent Nedir?

Yolların ya da geçitlerin çokluğu, karışıklığı yüzünden içinden kolay kolay çıkılamayan yerlere labirent veya dolambaç denir.

Yunan mitolojisinde, bu çeşit yerlerden ilkinin, mimar Daedalus tarafından Girit kralı Minos için yapıldığını anlatan bir hikâye vardır. Girit kralı Minos bir labirentte insan vücutlu, boğa başlı bir yaratık olan Minotor'u hapsetmişti. Bir gün Minos'un kızı Ariadne'nin sevgilisi Theseus, Minator'u öldürmek üzere Girit'e geldi. Ariadne ona, labirente girmeden önce, çıkarken yolunu kolay bulabilmesi için bir yumak verdi. Theseus bu labirente girerken bu yumağın ipliklerini yere salıverecek, çıkarken de iplikleri toplayarak aynı yoldan geçip dışarı çıkabilecekti. Böylece Theseus Minator'u öldürdükten sonra, yumak sayesinde yolunu bulup labirentten çıkmayı başardı.

Son yıllarda Girit'te yapılan kazılarda, Knossos yakınlarında bir dağda, bu hikâyede anlatılan labirente benzeyen bir yapı kalıntılarına rastlanmıştır. Son kazılarda Mısır'da da böyle bir labirentin kalıntıları ortaya çıkarılmıştır.

Aynı zamanda bazı dergilerde labirentin karışık, şaşırtıcı yollarına benzeyen bilmeceler ve bulmacalar yayınlanır. Psikoloji bilginleri bu çeşit bilmecelerin zihni geliştirmeye, zekayı işletmeye faydalı olduğunu ileri sürmektedirler. [3]

C. Projenin Basit Bir Özeti

Bu projede, bir robotun ızgara üzerinde verilen hedefe engellere takılmadan en kısa sürede ve en kısa yoldan, tüm ızgarayı değil, yalnızca gerekli yolları gezerek ulaştırılması amaçlanmıştır.

Bu uygulamada program, internetteki bir dosyadan alınmış veya kullanıcı tarafından boyutları girilerek oluşturulmuş bir ızgara matrisini görselleştirebilmekte, bir robotu bu ızgara matrisi üzerinde hareket ettirerek belirli bir giriş noktasından, belirli bir çıkış noktasına ulaştırmakta, ve aynı zamanda robotun yolunu da görselleştirmektedir.

Oluşturulmuş ızgara üzerinde engeller bulunmaktadır. Bu engellerin konumları internetten alınmış veya kullanıcı

tarafından boyutları verilerek oluşturulmuş bir ızgarada 1,2, veya 3 olarak belirtilmiştir. Bu rakamların her birisi farklı bir biçimdeki engeli temsil etmektedir.

Birinci problemde robotun başlangıç ve hedef noktaları, engel barındırmayacak şekilde olmak üzere, rastgele olarak belirlenmiştir. Ancak ikinci problemde bu hedef noktaları üstteki koşulların yanı sıra köşelerde de olmalıdır.

Tüm bilgilerin ışığında robotun en kısa sürede hedefe ulaştığı yol, bu yolu gittiğinde geçen zaman ve robotun kaç kare hareket ettiği kullanıcıya ekran üzerinde gösterilmektedir. Aynı zamanda robot, başlangıçta içinde bulunduğu labirentin çıkış noktasını bilmeden hareket etmektedir.

Izgara üzerindeki robot yalnızca sağa, sola, aşağıya veya yukarıya doğru hareket edebilmektedir. Robotun çapraz hareket etme kapasitesi bulunmamaktadır. Aynı zamanda robot yukarıda belirtildiği üzere labirent ızgarasını görememekte, yalnızca bir kare sonrasında ne olduğunu bilmektedir.

Izgara ise 1. problemde istenilen URL adresi üzerinden alınan bilgiler ile oluşturulurken, 2. problemde ise kullanıcıdan alınan boyut bilgileri ışığında rastgele ancak çözülebilir bir biçimde oluşturulmaktadır. 2. problemde çözülemez bir ızgara oluşmasını önleyecek metodlar kullanılmıştır.

1. problemin ızgarasında üç farklı çeşit engel olmasına karşılık, 2. problemde yalnızca bir çeşit engel bulunmaktadır.

1. problem ekranında 1. ve 2. URL ızgaralarının kullanıcıya gösterilmesi için iki ayrı buton bulunmakta, aynı zamanda robotun çalışması için bir de ”Çalıştır” butonu bulunmaktadır.

2. problem ekranında ise sadece ızgara oluşturabilmek için bir ”Izgara Oluştur” butonu ve çalıştırmak için ”Çalıştır” butonu bulunmaktadır.

Gösterilmekte olan labirentte sis efekti (ki bu projede siyah renk olarak kullanılmıştır) bulunmakta, bu sayede robotun gördüğü yollar kullanıcıya daha rahat bir şekilde gösterilebilmektedir.

Aynı zamanda projede istendiği gibi, programın bütün çıktıları bir output.txt (kaçıncı problem olduğuna göre değişmektedir) dosyasında saklanmaktadır. Bu sayede robotun hareketleri de okunabilmektedir.

III. Yöntem

A. Problem 1 Ekranı

Öncelikle Problem1 class’ından Problem adlı bir nesne türetildi. Problem1 class’ı JFrame’i extend etmektedir. Sonrasında Problem nesnesinin setup metodu çağrılarak, 1. problemin ekranı oluşturulmuştur.

Setup methodunda JFrame class’ından Problem adlı bir nesne türetilmiştir. Problem nesnesinin title’ı ”PROBLEM 1”dir. Boyutu her ne kadar 1600x900 olarak ayarlanmış olsa da ”Problem.setExtendedState(JFrame.MAXIMIZED_BOTH)” kod satırı kullanılarak pencerenin ekran boyutlarında olması sağlanmıştır.

Programın menüsü JMenuBar class’ından türetilmiş MenuBar nesnesine yerleştirilmiş, JMenu class’ından türetilmiş ProblemMenu nesnesi kullanılarak oluşturulmuştur. Bu menüye JMenuItem class’ından türetilmiş URL1, URL2, ve Run nesneleri ise istenen buton rolünü gören menü tuşlarıdır. ProblemMenu adlı JMenu nesnesi kullanıcı tarafından ”Problem Menüsü”, URL1, URL2 ve Run adlı JMenuItem nesneleri ise ”1. URL’den Izgara Oluştur”, ”2.URL’den Izgara Oluştur” ve ”Çalıştır” olarak görülmektedir.

URL1, URL2 ve Run nesnelerine eklenmiş ActionListener’lar sayesinde kullanıcının istediği JMenuItem’lara tıklandığında program kendisinden istenen işlemleri gerçekleştirebilmektedir.

B. Birinci Problemin Dosyalarının İnternetten Çekilmesi ve Izgaranın Oluşturulması

Setup methodunun içinde, ancak ActionListener methodlarının dışında (bu hareket fonksiyondan çıkıldığında nesnenin yok olmamasını sağlar), Map class’ına ait olmak üzere Grid1 (1. URL’den alınacak ızgara) ve Grid2 (2. URL’den alınacak ızgara) nesneleri türetilmiştir.

1. problemde istenen dosyalar sırasıyla ”http://bilgisayar.kocaeli.edu.tr/prolab2/url1.txt” ve ”http://bilgisayar.kocaeli.edu.tr/prolab2/url2.txt” sitelerinde bulunmaktadır. Bu sitelerden .txt dosyalarını çekebilmek için çeşitli Scanner’lar (RowScanner, ColumnScanner ve MatrixScanner) ve URL class’ından türetilmiş GridMatrixData nesneleri kullanılmıştır[4]. GridMatrixData nesneleri ile internete ulaşıldıktan sonra, Scanner’lar ile .txt dosyaları okunmuş, gerekli değişkenlere (RowCount, ColumnCount ve IntegerMatrix) .txt dosyalarında olan değerler atanmıştır. RowCount değişkenleri için dosyalardaki satırlar sayılırken[5], ColumnCount değişkenleri için dosyaların ilk satırlarının uzunluğu bulunmuştur[6]. Sonrasında ise .txt dosyalarındaki ızgara matrisinin uzun vadeli kullanımı için, .txt dosyalarındaki ızgara matrisleri Grid1 ve Grid2 nesnelerinin IntegerMatrix değişkenlerine kopyalanmıştır[7]. Başarılı olup olunmadığının testi için çıktı dosyalarında satır sayısı, sütun sayısı ve matrisin kendisi ekran çıktısında bulunmaktadır.

Sonrasında ise ızgaranın giriş çıkış noktaları, istendiği gibi, rastgeler olarak belirlenmiştir. Bunun için RandomizeStartEnd methodu kullanılmıştır. Bu methoda, kullanılan ızgaray göre, ızgaranın IntegerMatrix, RowCount ve ColumnCount değişkenleri gönderilmektedir. Random class’ından türetilmiş rand nesnesi kullanılarak StartY ve StartX rastgele bir biçimde belirlendikten sonra bu noktada bir engeli bulunup bulunmadığı kontrol edilir. Eğer engel bulunuyorsa, bir kez daha rastgele seçim yapılır. Bu durum engel olmayan bir yer bulununcaya kadar devam etmektedir. Engel olmayan bir yer bulunduktan sonra, bu noktanın değeri 0’dan 5 yapılmakta, ve boolean Start değişkeni true değerini almaktadır. Çıkış noktasının belirlenmesi için de aynı hareketler yapılsa da bu hareketlerde sadece engel değil başlangıç noktası da tekrar bir koordinat seçimini tetiklemektedir. Aynı zamanda çıkış bir 5 ile değil, 8 ile ifade edilmektedir. Uygun bir çıkış noktası bulunduktan sonra boolean End değişkeni de true olur ve fonksiyondan çıkılır. Doğruluğun kontrol edilmesi için matris yeniden ekran çıktısına kaydedilir.

Bu olanlardan sonra labirenti çözecek butonun hareket edebilmesi için bir obje matrisi oluşturulması hedeflenmiştir.

Bu obje matrisi ise CreateObjectMatrix methodu kullanılarak oluşturulmuştur. Bu methoda, kullanılan ızgaraya göre, ızgaranın IntegerMatrix, RowCount, ColumnCount ve matrisin kaydedilebilmesi için CellMatrix değişkenleri gönderilmektedir. Belirli bir koordinattaki IntegerMatrix değeri gridnumber değişkenine atanmakta, bu değişkenin değerine göre Cell sınıfından olmak üzere çeşitli nesneler atanmıştır. Bu nesneler değerine göre sırasıyla, 0 ise BlankCell, 1 ise Obstacle1, 2 ise Obstacle2, 3 ise Obstacle3, 5 ise StartCell ve 8 ise FinishCell'dir. Tüm nesnelerin x,y değerleri IntegerMatrix koordinatları ile aynı olsa da, Obstacle nesnelerinin IsObstacle boolean değişkenlerinin değeri true, StartCell nesnesinin IsStart boolean değişkeninin değeri true ve FinishCell nesnesinin IsFinish boolean değişkeninin değeri true olarak atanmıştır.

Tüm bu olanlardan sonra sıra artık ızgaranın görselleştirilmesine gelmiştir.

C. Birinci Problemin Izgarasının Görselleştirilmesi

Öncelikle tekrar tekrar oluşturulması durumunda birbirlerinin yanına eklenmemeleri için Problem JFrame'i temizlenmekte ve revalidate edilmektedir. Bu olaylar sonrasında kullanılacak olan ızgaranın boyutlarına göre bir GridLayout oluşturulduktan sonra, ızgara matrisindeki her bir rakam için bir for döngüsü içinde bir gridlabel nesnesi, CreateGridLabels methodu kullanılarak oluşturulmakta ve Problem JFrame'inin üzerine eklenmektedir.

CreateGridLabels fonksiyonu yalnızca IntegerMatrix'teki belirli bir koordinattaki rakamı almakta, buna göre bir JLabel oluşturmaktadır. Bu JLabel'ın HorizontalAlignment'ı "Center" olmakla birlikte PreferredSize'ı 30x30'dur. JLabel'ın rengi ise alınan rakama bağlıdır. Eğer rakam 0 ise siyah, 1, 2 veya 3 ise kırmızı, 5 ise yeşil, 8 ise sarı, 4 ise mavi, 7 ise turuncu olmaktadır. Her ne kadar bu method, ilk kullanıldığında ızgara matrisinde 4 veya 7 olmasa da daha sonraki durumlarda kullanıldığında gerekli olacaklardır. Bundan sonra JLabel opaklığı true olarak ayarlandıktan sonra, JLabel'ın sınırları da beyaz renk olarak ayarlanmıştır. Bundan sonra gridlabel adlı JLabel nesnesi geri döndürülmektedir[8].

Izgara oluşturulduktan sonra, gösterilen ızgaranın nesnesinin Display değişkeni true, gösterilmeyen ızgaranın nesnesinin Display değişkeni ise false değerini almaktadır. Bu ızgara labirenti çözülürken hangi ızgaranın çözüleceğinin anlaşılmasında rol oynayacaktır.

D. Birinci Problemin Çözülmesi

Ekranda gösterilen ızgaranın çözümü için menüdeki "Çalıştır" tuşuna basıldığında, MazeSolverStartTime adlı long değişken, Start alınan süreyi System.nanoTime() methodunu kullanarak alır. Bundan sonra MazeSolverRobot class'ından türetilmiş MSBot adlı nesne oluşturulduktan sonra bu robotun geçtiği kare sayısını sayacak MazeSolverPassedSquares değişkeni en başta değeri 0 olmak üzere tanımlanmıştır. Ancak robot hangi ızgarayı çözmesi gerektiğini bilmemektedir. Burada ise Map class'ından türetilmiş ve ızgara bilgilerini saklayan nesnelerin Display değişkenleri devreye girmektedir. Ekranda gösterilen ızgaranın nesnesinin Display değişkeni true değerinde olması nedeniyle, robot Display değişkeni doğru olan ızgarayı çözmektedir. Robotun ızgarayı çözmesi için robotun SolveMaze() methodu çağrıldıktan ve ızgara labirenti çözüldükten sonra, IntegerMatrix'teki değeri 5, 8, 4 veya 7 olan değişkenlere rastlandıkça MazeSolverPassedSquares 1 artmaktadır. Bunun da nedeni SolveMaze() methodunun (şu anda bir şey yapamıyor olsa da),IntegerMatrix'te en hızlı yolu 7, geçtiği yolları ise 4 yapmasıdır. Bundan sonra Problem JFrame'i bir kez daha temizlenmekte ve revalidate edilmekte, ve bu sefer çözülmüş durumda olan IntegerMatrix kullanılarak ızgara bir kez daha görselleştirilmektedir. Tüm bu olanların sonunda MazeSolverFinishTime adındaki long değişken, System.nanoTime() methodunu kullanarak, bitiş zamanını almaktadır. MazeSolverElapsedTime adındaki long değişken ise MazeSolverFinishTime'dan MazeSolverStartTime'ı çıkararak, sürecin ne kadar sürede gerçekleştiğini nanosaniye olarak bulmaktadır. Bu nanosaniye her ne kadar daha uygun saniye çeşitlerin çevirilmeye çalışıldıysa da bunda başarılı olunamamıştır. Bu olaylardan sonra, bir JOptionPane'de "Izgara Labirentinin Çözümü İçin Geçen Süre" ve "Izgara Labirentinin Çözümü İçin Geçilen Kare Sayısı" değişkenleri gösterilmektedir[9].

E. Problem 2 Ekranı

Öncelikle Problem2 class'ından Problem adlı bir nesne türetildi.Problem2 class'ı JFrame'i extend etmektedir. Sonrasında Problem nesnesinin Setup() methodu çağırılarak kullanılmıştır.

Setup methodunda JFrame class'ından Problem adındaki bir nesne türetilmiştir. Bu Problem nesnesinin title'ı "PROBLEM 2"dir. Aynı zamanda boyutu 1600x900 olmasına rağmen "Problem.setExtendedState(JFrame.MAXIMIZED_BOTH)" kod satırı kullanılarak pencerenin ekran boyutlarında olması sağlanmıştır.

Programın menüsü JMenuBar class'ından türetilmiş MenuBar nesnesine yerleştirilmiş, JMenu class'ından türetilmiş ProblemMenu nesnesi kullanılarak oluşturulmuştur. Bu menüye JMenuItem class'ından türetilmiş Generate ve Run nesneleri ise istenen buton rolünü gören menü tuşlarıdır. ProblemMenu adlı JMenu nesnesi kullanıcı tarafından "Problem Menüsü", Generate ve Run adlı JMenuItem nesneleri ise "Izgara Oluştur" ve "Çalıştır" olarak görülmektedir.

Generate ve Run nesnelerine eklenmiş ActionListener'lar sayesinde kullanıcının istediği JMenuItem'lara tıklandığında program kendisinden istenen işlemleri gerçekleştirebilmektedir.

F. İkinci Problemin Izgarasının Oluşturulması

Her şeyden önce Map class'ına ait GeneratedGrid adındaki bir nesne oluşturulur. Bu nesne ızgara hakkındaki bilgileri tutmaktadır. Generate tuşuna tıklanması ile kullanıcıya ızgaranın sütun sayısını girmesini isteyen bir JOptionPane açılır. Girilmiş olan değer col adlı bir String'te saklanır.

Sütun sayısı girildikten sonra, ızgaranın satır sayısının girilmesini sağlayan başka bir JOptionPane açılır. Girlimiş olan değer row adlı bir String'te saklanmaktadır. Satır sayısı da girildikten sonra, değerler String değişkenlerinden GeneratedGrid'in RowCount ve ColumnCount değişkenlerine aktarıldıktan sonra, 0'lardan ve 1'lerden oluşan rastgele bir matris oluşturulur[10]. Bu matris, GeneratedGrid'in IntegerMatrix'idir. Dah sonrasında RandomizeStartEnd() methodu kullanılarak ızgaranın giriş ve çıkış noktaları rastgele köşelere yerleştirilmektedir.

RandomizeStartEnd() methoduna, GeneratedGrid'in IntegerMatrix'i, Row Count'u ve ColumnCount'u gönderilmektedir. Random class'ında türetilmiş rand nesnesi sayesinde, StartYX ve EndYX değişkenleri 0, 1, 2 veya 3 değerini almaktadırlar. Bunun nedeni giriş ve çıkışların, 2. problem özelinde, sadece köşelerde olabilmesindendir. Eğer değişkenler birbirlerine eşitlerse ve 3 değillerse EndYX değişkenine 1 eklenmektedir. Eğer iki değişkenin de değerleri aynı ve 3 ise EndYX değişkeninden 1 azaltılmaktadır. Bu sayede giriş çıkış noktalarının çakışması önlenmektedir. Eğer değer 0 ise giriş veya çıkış sol üstte, 1 ise sağ üstte, 2 ise sol altta veya 3 ise sağ altta bulunmaktadır.

Sonrasında CanBeSolved(), CanBeSolvedIsSafe() ve CanBeSolvedIsPath() methodları kullanılarak, ızgaranın girişinden çıkışına bir yol olup olmadığı kontrol edilir. Eğer bir yol yok ise tekrardan rastgele bir matris oluşturularak ve giriş çıkışlar tekrar belirlenerek uygun bir matris bulunana kadar devam eder. Uygun bir matris oluşturulduğunda GeneratedGrid'in Display değişkeni true değerini alarak tüm bu işlemlerin tekrarını sağlayan while döngüsünden çıkışı sağlar.

CanBeSolved() methoduna, GeneratedGrid'in IntegerMatrix'i, Row Count'u ve ColumnCount'u gönderilmektedir. Çözüm olup olmadığı kontrol edilirken geçilen yerler ise visited adındaki bir boolean matrisinde saklanmaktadır. Aynı zamanda bir yol olup olmadığına göre pathexist adındaki bir boolean değişkenin durumu belirlenmektedir. Bu değişken sayesinde bir çözüm olup olmadığı program tarafından bilinmektedir. Daha sonra iki adet for döngüsü içinde giriş noktası bulunur ve bu start noktasından başlanarak bir yol aranması adına CanBeSolvedIsPath() methodu çağırılır[11].

CanBeSolvedIsPath() methoduna, CanBeSolved() methoduna gönderilmiş olan IntegerMatrix, giriş noktasının y ve x koordinatları ile boolean matris olan visited gönderilir. Öncelikle CanBeSolvedIsSafe() methodu kullanılarak sınırlar ve engeller tespit edilirken, visited boolean matrisi kulllanılarak daha önce geçilmemiş noktalar kontrol edilirler. Daha sonrasında üzerinde bulunulan nokta visited boolean matrisine eklenir. Ardından eğer nokta hedef nokta ise true değeri döndürülür. Eğer değilse yukarı, sola, aşağıya veya sağa hareket ederek yol aranır. Eğer bir yol varsa true, yoksa false değer döndürülür. Bu hareketler bir yol bulununcaya veya bir yol bulunamayacağı kesinleşene kadar devam eder[11].

Eğer bir yol varsa, pathexist değişkeni true olur ve program doğru IntegerMatrix ile çalışmaya devam eder[11].

Döngüden uygun bir matris ile çıkıldığı zaman, CreateObjectMatrix() methodu kullanılarak GeneratedGrid'in CellMatrix'i oluşturulur. CreateObejctMatrix() methoduna, GeneratedGrid'in IntegerMatrix, RowCount, ColumnCount ve obje matrisinin kaydedilebilmesi için CellMatrix değişkenleri gönderilmektedir. IntegerMatrix'in belirli bir koordinatındaki rakam gridnumber değişkenine atanmakta, bu değişkenin değerine göre Cell sınıfından bir nesne oluşturulmaktadır. Bu nesneler değerlerine göre sırasıyla, 0 ise BlankCell, 1 ise Obstacle, 5 ise StartCell ve 8 ise FinishCell'dir. Tüm nesnelerin x,y değişkenleri IntegerMatrix'teki rakamın x,y değerleriyle aynı olsa da, Obstacle nesnelerinin IsObstacle boolean değişkenlerinin değeri true, StartCell nesnesinin IsStart boolean değişkeni true ve FinishCell nesnesinin IsFinish boolean değişkeni true olarak atanmaktadır.

Tüm bu olanlardan sonra sıra ızgaranın görselleştirilmesine gelir.

G. İkinci Problemin Izgarasının Görselleştirilmesi

Öncelikle birden çok kez ızgara oluşturulması durumunda ızgaraların birbirlerinin yanına yerleşmesini engellemek adına Problem JFrame'inin ContentPane'inindeki her şey kaldırılmakta (bu üstteki menüyü içermemektedir), ve revalidate edilmektedir. Bundan sonra GeneratedGrid'in boyutlarına göre bir GridLayout oluşturulmakta, bunnlardan sonra GeneratedGrid'in IntegerMatrix'indeki her bir rakam değişkeni için gridlabel adındaki bir JLabel, CreateGridLabels() methodu kullanılarak oluşturulmakta ve bu gridlabel nesnesi Problem JFrame'inin üzerine eklenmektedir. CreateGridLabels fonksiyonu yalnızca IntegerMatrix'in belirli bir koordinatındaki sayıyı almakta ve buna göre bir JLabel oluşturmaktadır. Bu JLabel'ın HorizontalAlignment'ı "Center", PreferredSize'ı 30x30'dur. JLabel'ın rengi IntegerMatrix'ten alınan değişkenin değerine göre değişmektedir. Eğer değer 0 ise siyah, 1 ise kırmızı, 5 ise yeşil, 8 ise sarı, 4 ise mavi ve 7 ise turuncu renkte olmaktadır. Bu işlemlerden sonra JLabel'ın opaklığı true olarak ayarlanmakla birlikte, JLabel'ın sınırları beyaz renkte olmak üzere ayarlanmıştır. Bundan sonra gridlabel adlı JLabel nesnesi geri döndürülmektedir[8].

H. İkinci Problemin Çözülmesi

Ekranda gösterilmekte olan ızgaranın çözülmesi için menüdeki "Çalıştır" tuşuna basıldığında, MazeSolverStartTime adındaki long değişken, System.nanoTime() methodunu kullanarak Start alınan süreyi alır. Bundan sonra MazeSolverRobot adlı class'tan türetilmiş olan MSBot nesnesi oluşturulduktan sonra, bu robotun geçtiği kare sayısını sayacak olan MazeSolverPassedSquares değişkeni değeri en başta 0 olmak üzere tanımlanmıştır. Bu problemde yalnızca bir ızgara olsa da eğer GeneratedGrid'in Display değişkeni true ise MSBot SolveMaze() methodunu çağırarak, ızgara labirentini çözecektir. Çözüm gerçekleştirildikten sonra GeneratedGrid'in IntegerMatrix'indeki değerleri 5, 8, 4 veya 7 olan değişkenlere rastlandıkça MazeSolverPassedSquares 1 artmaktadır. Bunun nedeni birinci problemde de açıklandığı üzere SolveMaze() methodunun, IntegerMatrix'te en hızlı yolu 7, geçtiği tüm yolları ise 4 yapmasıdır. Bundan sonra Problem JFrame'i bir kez

daha temizlenmekte ve revalidate edilmekte, bunlardan sonra GeneratedGrid'in çözülmüş durumdaki IntegerMatrix'i kullanılarak, ızgara bir kez daha görselleştirilmektedir. Gereken işlemlerin hepsi gerçekleştikten sonra, MazeSolverFinishTime adındaki long değişken, System.nanoTime() methodunu kullanarak, bitiş zamanını almaktadır. MazeSolverElapsedTime adındaki long değişken ise MazeSolverFinishTime değişkeninden MazeSolverStartTime değişkenini çıkararak tüm işlemlerin gerçekleşme süresini nanosaniye olarak bulmaktadır. Her ne kadar nanosaniye diğer saniye birimlerine çevrilmek istense de bu konuda bşarılı olunamamıştır. Her şeyin sonunda bir JOptionPane'de "Izgara Labirentinin Çözümü İçin Geçen Süre" ve "Izgara Labirentinin Çözümü İçin Geçilen Kare Sayısı" ilgili değişkenleri ile gösterilmektedir[9].

IV. Deneysel Sonuçlar

- Birinci problemde menü sorunsuz bir biçimde çalışmakla birlikte tuşlar da tıklandığında doğru bir biçimde cevap vermektedir.
- Birinci problemde ızgara matrisleri doğru biçimde internetten alınıp görselleştirlebilmekte, aynı zamanda robotun hareket edebilmesi adına bir obje matrisi de oluşturulabilmektedir.
- Birinci problemde maalesef robotun SolveMaze metodu hazır olmadığından robot ızgara labirentini çözememektedir. Bu nedenle MazeSolverPassedSquares değişkeni her zaman 2 (bir tane giriş, bir tane çıkış) olarak kalmaktadır. Aynı zamanda geçen süre nanosaniyeden daha kolay anlaşılabilecek bir saniye birimine dönüştürülememiştir.
- İkinci problemde menü sorunsuz bir biçimde çalışmakla birlikte tuşlar da tıklandığında doğru bir biçimde cevap vermektedir.
- İkinci problemde oluşturulması gereken labirent matrisi doğru bir şekilde oluşturulabilmektedir. Doğru bir matris olduğunun doğrulanmasında CanBeSolved(), CanBeSolvedIsSafe() ve CanBeSolvedIsPath() methodları sayesinde her ne kadar robot çözebilecek fonksiyonlara sahip olamasa da bir çözüm gerçekleştirilebilmiştir.
- İkinci problemde ızgara görselleştirilmesi sorunsuz bir biçimde gerçekleştirilse de belirli bir boyut aşıldığında program donmaya başlamaktadır. O yüzden programın genelde hafif labirentler oluşturulmakta kullanılması tavsiye edilmektedir.
- İkinci problemde maalesef robotun SolveMaze metodu hazır olmadığından robot ızgara labirentini çözememektedir. Bu nedenle MazeSolverPassedSquares değişkeni her zaman 2 (bir tane giriş, bir tane çıkış) olarak kalmaktadır. Aynı zamanda geçen süre nanosaniyeden daha kolay anlaşılabilecek bir saniye birimine dönüştürülememiştir.
- Tüm ekran çıktıları başarılı bir biçimde bir .txt dosyasına kaydedilebilmektedir.

V. Sonuç

Bu projenin sonunda görülebileceği gibi Java programlama dili ve çeşitli labirent oluşturma ve çözme algoritmaları kullanılarak istenilen labirenti rastgele olmak üzere veya internetten aldığı bir .txt dosyası sayesinde oluşturabilen, oluşturduğu labirenti ızgara üzerinde görselleştirebilen, bu labirenti verilen kurallara uyarak çözebilen ve çözümünü aynı ızgarada görselleştirebilen bir program hazırlanabilmiştir.

VI. Kaynakça

References

[1] https://www.java.com/tr/download/help/whatis_java.html
[2] https://aws.amazon.com/tr/what-is/java/
[3] https://tr.wikipedia.org/wiki/Labirent
[4] https://stackoverflow.com/questions/6259339/how-to-read-a-text-file-directly-from-internet-using-java
[5] https://stackoverflow.com/questions/26448352/counting-the-number-of-lines-in-a-text-file-java
[6] https://stackoverflow.com/questions/55012354/how-to-read-the-number-of-columns-in-text-file-in-java
[7] https://stackoverflow.com/questions/49562415/read-a-matrix-from-a-txt-and-store-it-in-a-2d-array-in-java
[8] https://stackoverflow.com/questions/55973794/how-to-display-a-maze
[9] https://stackoverflow.com/questions/19342681/how-do-display-multiple-lines-in-joptionpane
[10] https://www.codespeedy.com/generate-random-matrix-in-java/
[11] https://www.geeksforgeeks.org/find-whether-path-two-cells-matrix/

VII. Projenin Ekran Görüntüleri

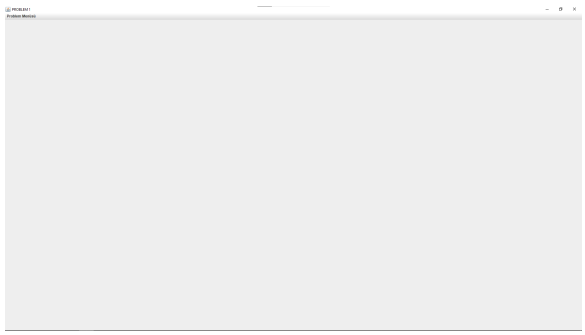

Fig. 1. Problem 1 Menü Tasarımı

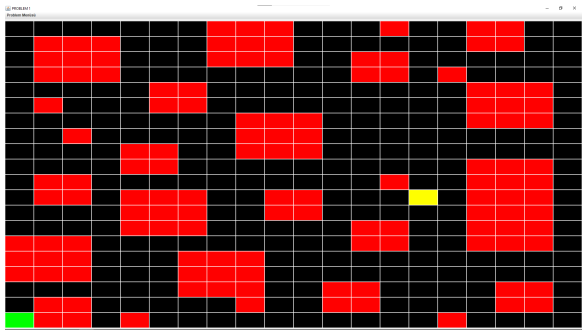

Fig. 2. Problem 1 1. URL Izgarası - 1

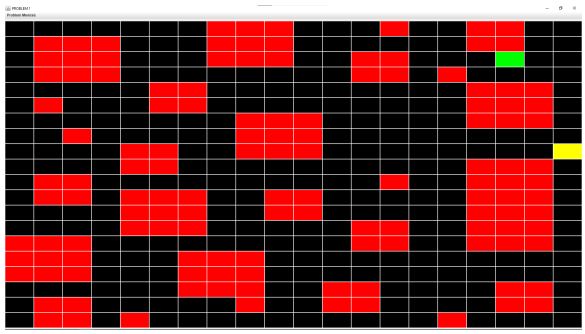

Fig. 3. Problem 1 1. URL Izgarası - 2

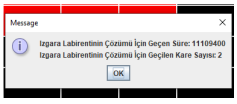

Fig. 4. Problem 1 1. URL Izgarası - Sonuç

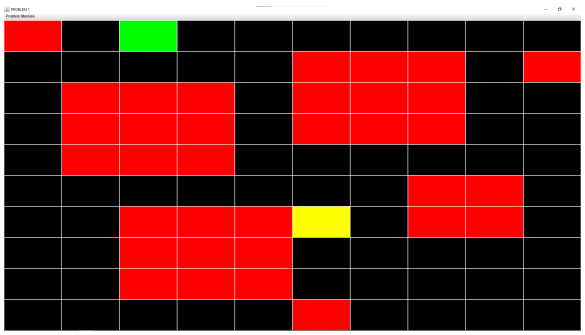

Fig. 5. Problem 1 2. URL Izgarası - 1

Fig. 6. Problem 1 2. URL Izgarası - 2

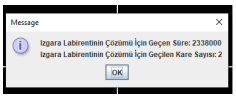

Fig. 7. Problem 1 2. URL Izgarası - Sonuç

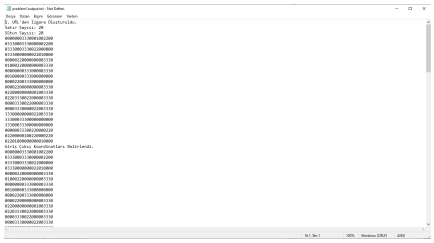

Fig. 8. Problem 1 Output Dosyası

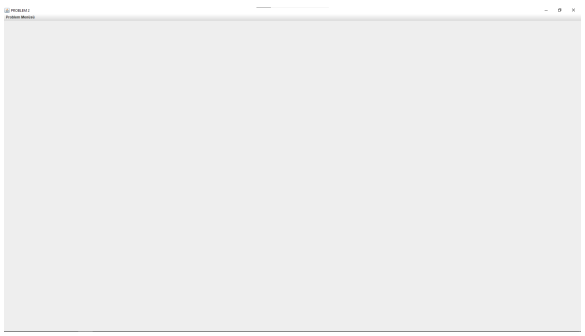

Fig. 9. Problem 2 Menü Tasarımı

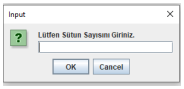

Fig. 10. Problem 2 Sütün Girdisi

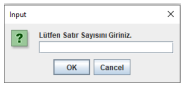

Fig. 11. Problem 2 Satır Girdisi

Fig. 12. Problem 2 Oluşturulmuş Izgara 1

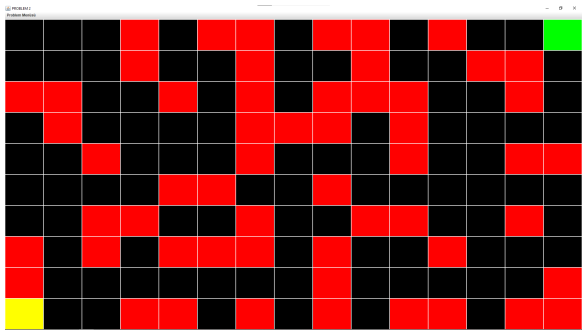

Fig. 13. Problem 2 Oluşturulmuş Izgara 2

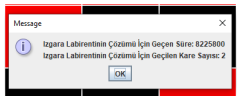

Fig. 14. Problem 2 Sonuç

Fig. 15. Problem 2 Ouput Dosyası

Fig. 16. Problem 1 Akış Şeması

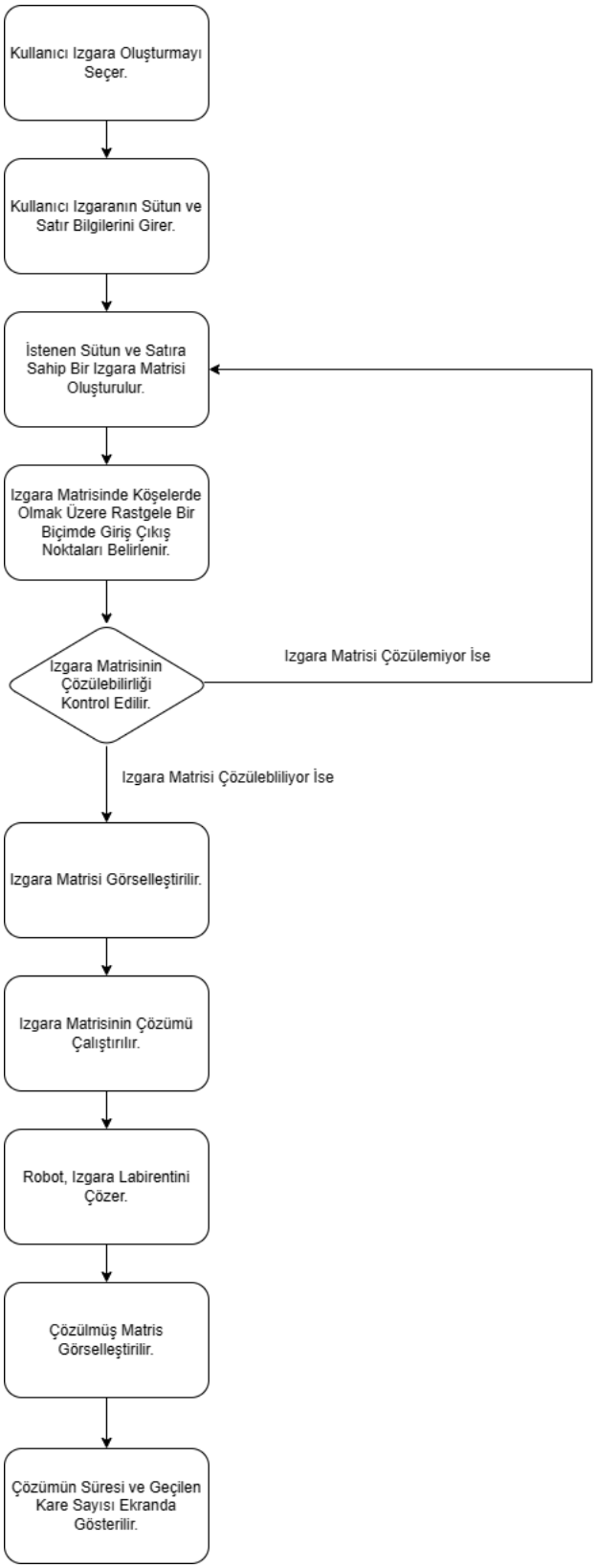

Fig. 17. Problem 2 Akış Şeması

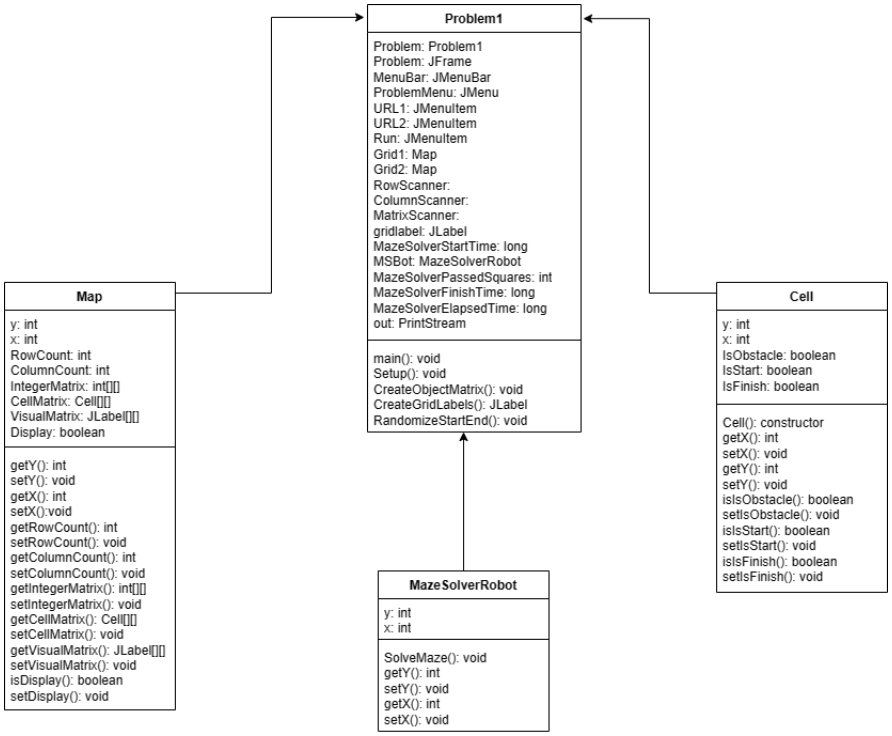

Fig. 18. Problem 1 UML Diyagramı

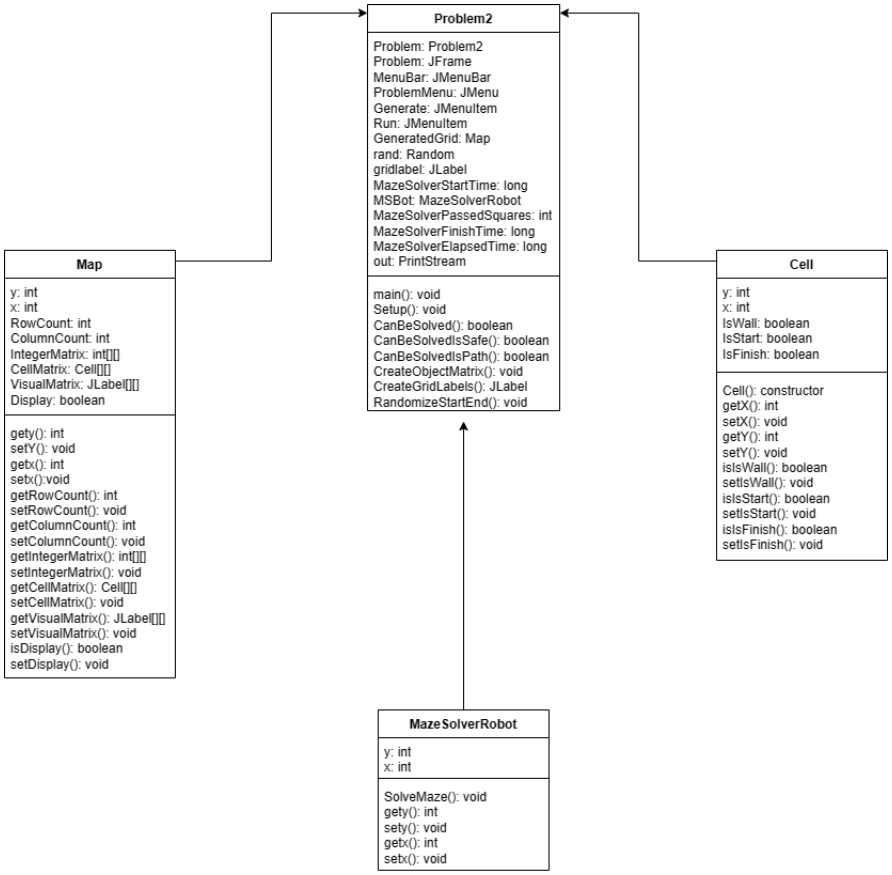

Fig. 19. Problem 2 UML Diyagramı