Gezgin Robot Projesi

Aydın Can Altun
180202117

Barış Arslan
180202112

***Özet*—Gezgin Robot Projesi, uygulamaya verilen matris'I okuyarak yada kullanıcıdan alınan genişlik ve uzunluk bilgilerini kullanrak bir girişi ve çıkışı olan bir labirent oluşturan. Bu labirentin başlangıç noktasına bir gezgin robot koyan ve bu gezgin robotun, harita hakkında hiç bir bilgiye sahip olmadan haritanın çıkışını bulmasını amaçlayan bir projedir.**

Anahtar Kelimeler—labirent oluşturma, labirent içerisinde yol bulma, wall follower algoritması, prim's algoritması, nesneye yönelik programlama, soyut sınıflar, arayüzler, bağımlılık yönetimi

I. GIRIŞ

Gezgin Robot Projesi, verilen matris'i okuyarak yada kullanıcıdan alınan genişlik ve uzunluk bilgilerini bir girişi, bir çıkışı olan bir labirenti oluşturan bir haritaya bir robot yerleştirip bu robotun harita hakkında hiçbir bilgisi olmadan çıkışı bulmasını amaçlayan bir projedir. Robot, çıkışı bulduktan sonra harita içerisinde gezdiği tüm yollar üzerinden çıkışa giden en kısa yolu bulmak ile yükümlüdür. Harita oluşturma işlemi verilen bir matris üzerinden yapılıyor ise matris de birden fazla alan kaplayan engellerin bazılarını rastgele seçerek yürünebilir yol haline getirilmesi sağlanmıştır. Harita oluşturma işlemi bir uzunluk ve genişlik verilerek yapılıyor ise bir labirent oluşturulmalı ve labirentin bir girişi, bir çıkışı olmalıdır. Labirentin girişini ve çıkışını sağlayan tek bir yol olması ve kalan yollar çıkmaz sokak olması sağlanmıştır. Harita oluşturulduktan sonra haritanın başlangıç noktasına bir gezgin robot yerleştirilmiş ve bu robotun, harita hakkında bir bilgisi olmadan haritanın çıkışını bulması sağlanmıştır. Robot, çıkışı bulduktan sonra harita üzerinde gezdiği tüm yollar baz alınarak haritanın çıkışına ulaşabileceği en kısa yolu hesaplaması sağlanmıştır. Harita oluşturma ve robotun ilerlediği yolların izlenmesi görselleştirilmesi konsol kullanarak gerçekleştirilmiştir.

II. YÖNTEM

A. *Verilen Matrisi Kullanarak Harita Oluşturma*

Verilen matris'de 0 ile işaretlenmiş blokların yol 1, 2, 3 ile listelenmiş blokların engeller olduğu kabul edilir ve 2 boyutlu Blok dizisi oluşturulup matrisdeki veriler bu diziye aktarılır ve uygulama için bilinir hale gelir. Bir bloktan fazla yer kaplayan engeller oluşturulan haritanın sol en üstünden başlanılarak tek boyutlu bir diziye eklenir. Bu dizi bir döngü içerisinde dolaşılarak. Rastgele bloklar seçilir ve bu bloklar engel olmaktan çıkarak yürünebilir yol haline getirilir.

B. *Verilen Uzunluk ve Genişlik Bilgilerine Göre Labirent Oluşturma*

Kullanıcıdan alınan uzunluk ve genişlik bilgilerine göre tamamen engel bloklarından oluşan 2 boyutlu bir Blok sınıfı oluşturulur. Daha sonra bu harita üzerinde bir koordinat seçilerek bu koordinatın üstündeki, altındaki, sağındaki ve solundaki bloklar. Seçilerek tek boyutlu bir dizinin içine koyulur. Seçilen duvarlar dizisi içerisinde eleman kalmayana kadar çalışacak bir döngü başlatılır. Döngü başlangıçta seçilen duvarlar içerisinden rastgele bir duvar seçer ve bu duvarın etrafındaki aynı eksendeki blokların bir tanesi engel bir tanesinin yol olmasını kontrol eder. Eğer bu şart dikey veya yatay eksen sağlanıyor ise ve seçilmiş duvarın etrafında ikiden az yol var ise seçilen duvar, yol haline getirilir, seçilen duvarlar listesinden çıkartılır ve onun etrafındaki duvarlar seçilen duvarlar listesine eklenir. Eğer bu şart sağlanmıyor ise ilgili duvar seçilen duvarlar listesinden çıkartılır. Seçilen duvarlar listesinde herhangi bir duvar kalmadığı zaman labirent oluşturulmuş olur.

C. *Harita İçerisinde Başlangıç ve Bitiş Noktalarının Belirlenmesi*

Harita oluşturulduktan sonra başlangıç noktası, haritanın solundan başlayacak şekilde en üstten 2. Satırı dolaşmaya başlar ve bulduğu ilk yol bloğun bir üstündeki bloğu yol haline getirir. Bitiş noktası ise haritanın en alt satırının bir üstündeki satırda sağdan sola doğru dolaşıp bulduğu ilk yolun bir altındaki bloğu yol haline getirerek seçilir.

D. *Robotun Harita Hakkında Hiçbir Bilgiye Sahip Olmadan Haritanın Çıkışını Bulması*

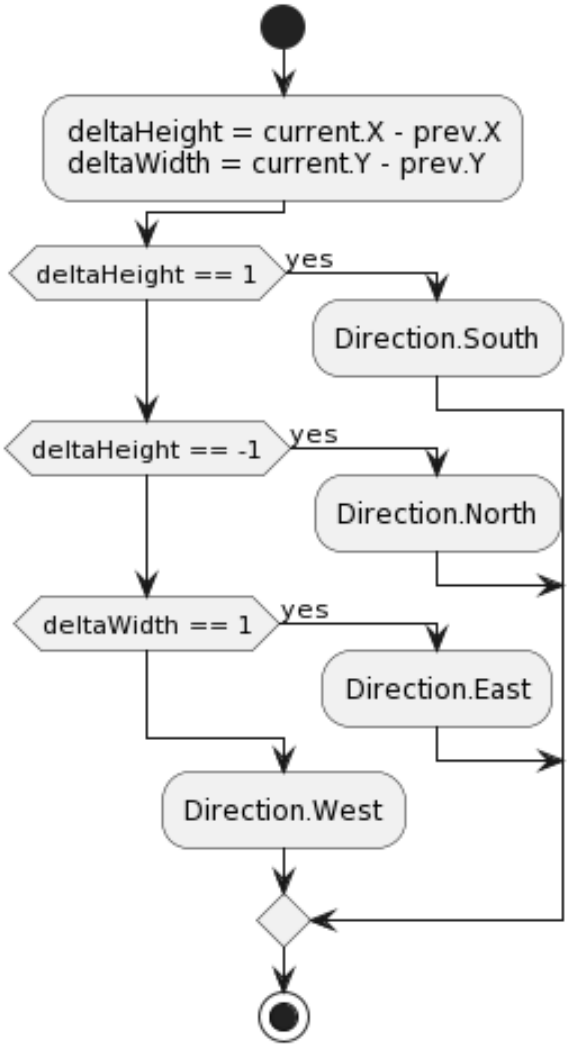
Robot, haritanın çıkışını ararken bulunduğu konuma göre elini sağındaki veya solundaki duvara yerleştirip bu duvarı takip ederek çıkışı bulur. Robot, haritaya ilk kez girdiğinde bir üstündeki, bir altındaki, bir solundaki ve bir sağındaki blokları kontrol eder ve verilen sıraya göre yürünebilir ilk yola doğru ilerleyerek adımını atar. İlk adımını attıktan sonra bir önceki adımını baz alarak şu anda hangi tarafa baktığını hesaplar. Eğer bir blok aşağıya indiyse güneye, bir blok yukarıya çıktıysa kuzeye, bir blok sola gittiyse batıya ve bir blok sağa gittiyse doğuya baktığını hesaplar. Bu hesaplamadan sonra takip ettiği duvara göre önceliklendirme kurallarına göre bir sonraki adımını atar.

TABLE I. SAĞ EL KURALI

Baktığı Yön	**İlk Tercih**	**İkinci Tercih**	**Üçüncü Tercih**	**Dördüncü Tercih**
Kuzey	Doğu	Kuzey	Batı	Güney
Güney	Batı	Güney	Doğu	Kuzey
Doğu	Güney	Doğu	Kuzey	Batı
Batı	Kuzey	Batı	Güney	Doğu

TABLE II. SOL EL KURALI

Baktığı Yön	İlk Tercih	İkinci Tercih	Üçüncü Tercih	Dördüncü Tercih
Kuzey	Batı	Kuzey	Doğu	Güney
Güney	Doğu	Güney	Batı	Kuzey
Doğu	Kuzey	Doğu	Güney	Batı
Batı	Güney	Batı	Kuzey	Doğu

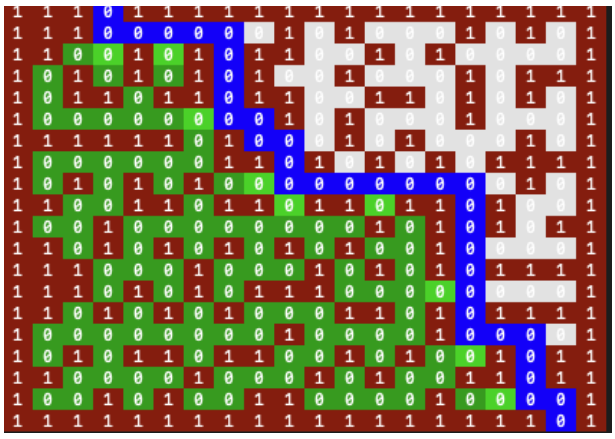


E. Robotun Haritanın Sonuna Ulaşmasından sonra Gezdiği Yollar İçerisindeki En Kısa Yolun Bulunması

Robot, her adım atmadan önce mevcut konumunu dolaştığı konumlar listesine ekler. Bu listedeki bilgiler kullanılarak robot bitiş noktası X koordinatına A ve yatay düzlemde gittiği en derin noktaya B denildiği zaman A x (B+1) uzunluğunda bir harita oluşturur. Oluşturulan haritada, robotun gezdiği her nokta yürünebilir nokta ve gezmediği her nokta engel olarak kabul edilir ve queue yapısı kullanılarak BFS algoritması çalıştırılarak robot gezdiği noktalar arasında bitiş noktasına ulaşan en kısa yolu hesaplar.

F. Oluşturulan Haritanın ve Robotun Hareketlerinin Görselleştirilmesi

Uygulama içerisinde haritanın görüntüsü ve robotların hareketlerinin izlenmesi konsol uygulamasında yansıtarak gerçekleştirilmiştir. Oyun haritası üzerinde, robot için görülmüş yürünebilir yollar yeşil arka plan rengi ile, robot tarafından görünmeyen yürünebilir yollar koyu yeşil renk ile, başlangıç noktası ve bitiş noktaları sarı arka plan rengi ile, üzerinden yürünemez engeller koyu kırmızı, yürünebilir engeller kırmızı renk ile, robotun aktif pozisyonu eflatun arka plan rengi ile, robotun daha önceden dolaştığı yollar beyaz arka plan rengi ile ve robotun gezdiği yollar arasında çıkış noktasına ulaşabileceği en kısa yol mavi arka plan rengi ile görselleştirilmiştir.



G. Projei Yapısı

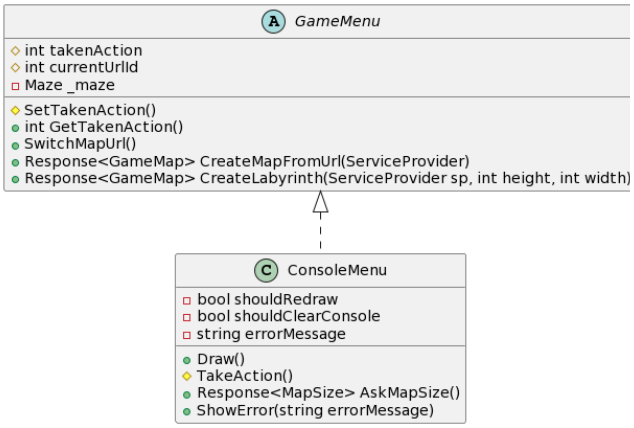
Proje .NET 7 Framework'ü kullanılarak geliştirilmiştir. Proje içerisinde içerisinde soyut sınıfları barındıran Abstractions, programın belli özelliklerini daha yönetilebilir olması için oluşturulmuş olan sınıflar için Entity, belirli özellikleri genelde kullanılmasını sağlayan sınıfları içinde barındıran Helpers, bağımlılıkları yönetmek için hazırlanmış olan arayüzleri Interfaces ve soyut ve arayüzleri uygulayan sınıfların bulunduğu Implementations klasörlerinden oluşmaktadır. Projenin gerçekleştirilmesi konsol üzerinden sağlanmış olsa bile Abstactions ve Interfaces klasörleri içerisindeki hazırlanan yapılara uyan yeni sınıflar yazılması takdirde uygulama minimum geliştirme eforu ile bir grafik kütüphanesi ile çalışabilir hale getirile bilinir. Benzer şekilde robotun yol bulma yöntemi hine bu yapılar kullanılarak, duvar takip etmesi yerine farklı yöntemlere çevrile bilinir.

III. NESNEYE YÖNELIK PROGRAMLAMAYA YAKLAŞIM

Çözülmesi gereken problemlerin görselliği farklı geliştiriciler tarafından farklı şekillerde gerçekleştirile bilineceği gibi harita hakkında hiçbir fikri olmayan bir robotun çıkış yolunu bulabilmesi farklı geliştiriciler tarafından farklı yöntemler kullanılarak gerçekleştirile bilinir. Bu sebepten ötürü haritayı çizen, robotu hareket ettiren ve bir kullanıcı uygulamayı kullanırken navigasyon görevi görecek menü soyut sınıflar olarak kurgulanmıştır. Bu soyut sınıfları uygulayan sınıfların, miras aldığı sınıftan daha fazla özellik bekleyebilmesi durumları da göz önünde bulundurularak iki adet arayüz sınıfı tanımlanmıştır.

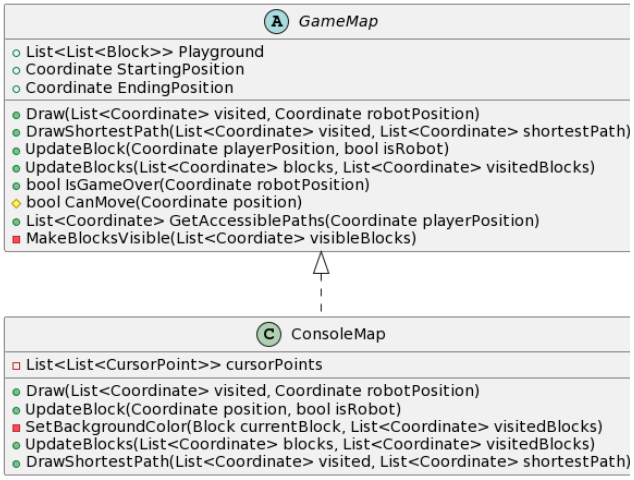
A. *GameMenu Soyut Sınıfı*

Uygulama içerisinde ne yapmak istediğini algılayan ve yapılmak istenen aksiyona göre Url'den matris okuyarak harita oluşturan ya da kullanıcıdan uzunluk ve genişlik bilgilerini alarak bir labirent oluşturan sınıftır. ConsoleMenu sınıfı bu sınıftan türemiştir ve kullanıcıdan aksiyonları konsol üzerinden alıp gerekli aksiyonları yerine getirir.



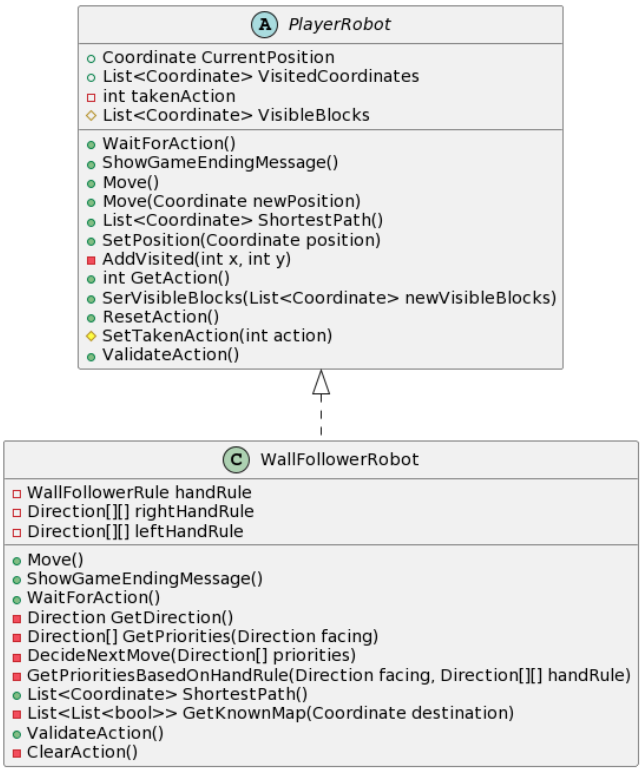
B. *GameMap Soyut Sınıfı*

Menüde alınan aksiyonlar sonrası oluşturulan haritayı kullanıcıya ve robotun harita üzerinde aldığı aksiyonları gösterir. ConsoleMap sınıfı bu sınıftan türemiştir ve haritayı ve haritada alınan aksiyonları konsol üzerinde gösterir. Robot çıkışı bulduktan sonra robotun hesapladığı en kısa yolu çizme görevini üstlenir.



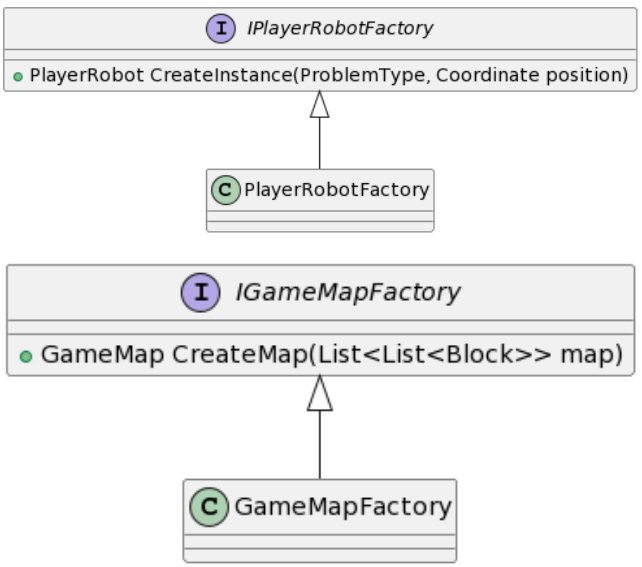
C. *PlayerRobot Soyut Sınıfı*

Harita oluşturulduktan ve kullanıcıya gösterildikten sonra harita hakkında bilgiye sahip olmadan çıkış yolunu bulmak ile yükümlü olan sınıftır. WallFollowerRobot sınıfı bu sınıftan türemiştir ve WallFollower algoritmasını kullanarak haritanın çıkışını arar. Haritanın çıkışını bulduktan sonra BFS algoritması ile robotun gezdiği yollar içerisinden çıkışa giden en kısa yolu bulur.

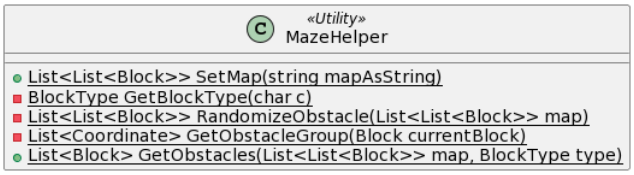


D. *Arayüzler*

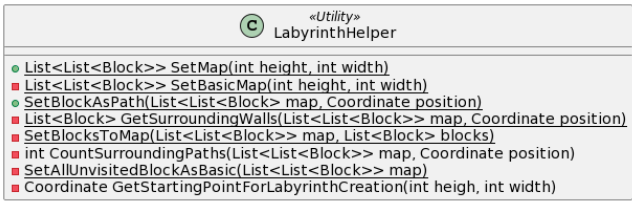
Soyut sınıfları uygulayan sınıfların oluşturulurken farklı parametrelere ihtiyaç olmasından dolayı. PlayerRobot ve GameMap soyut sınıflarını uygulayan sınıfları üretilmesi için kullanılacak iki adet arayüz oluşturulmuştur. PlayerRobotFactory, WallFollowerRobot sınıflarını üretirken url'den okunan bir matris'de yolu bulacaksa sol el kuralını, verilen uzunluk ve genişliğe göre rastgele oluşturulan labirentler için sağ el kuralını kullanacak şekilde oluşturulur.



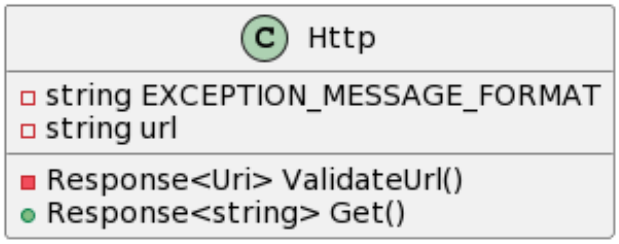
E. *Yardımcı Sınıflar*



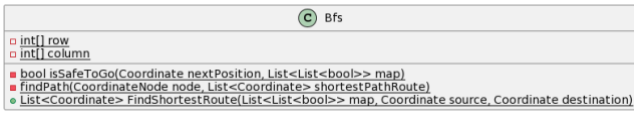
MazeHelper sınıfı, urlden okunan matrisin haritaya çevrilmesinde görevli statik sınıftır.



LabyrinthHelper sınıfı, verilen uzunluk ve genişlik bilgilerine göre bir girişi ve bir çıkışı olan rastgele bir labirent üretiminde görevli statik sınıftır.

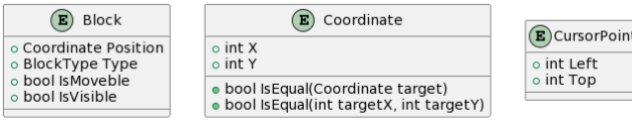


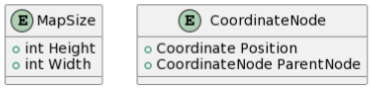
Http sınıfı, verilen url üzerindeki datayı okumakla görevli utility sınıfıdır.

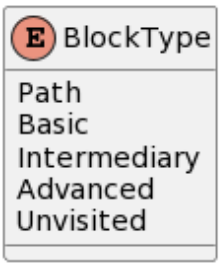


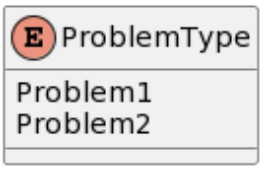
Bfs sınıfı, robotun dolaştığı yollar üzerinden hesapladığı haritayı alarak başlangıç noktasından, bitiş noktasına giden en kısa yolu bulur. PlayerRobot sınıfı uygulayan sınıf kendi içinde bir en kısa yolu bulma algoritması barındırmadığı durumlarda bu statik sınıf kullanılabilir.

F. *Özellik Sınıfları ve Enumlar*

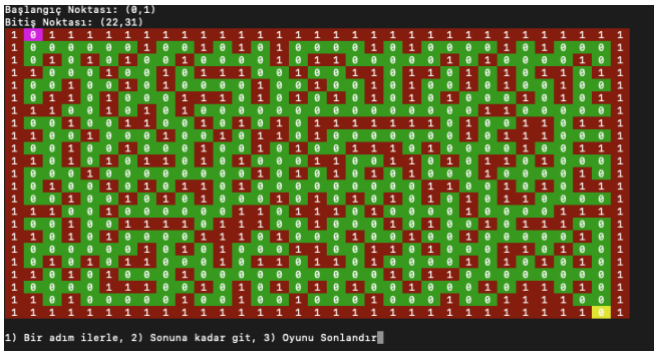


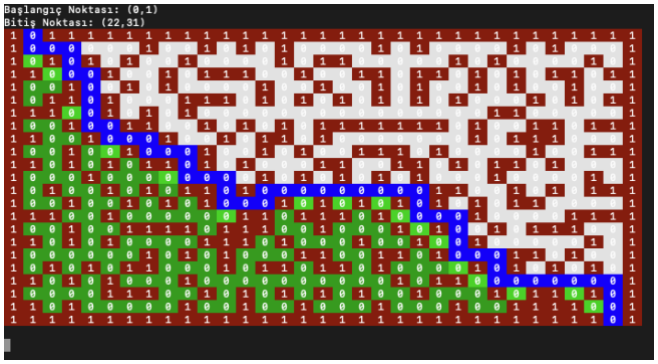


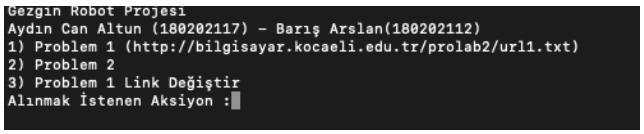


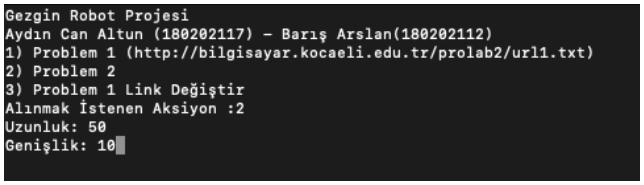


IV. Deneysel Sonuçlar









V. SONUÇ

Verilen bir matris üzerinden veya kullanıcıdan alınan uzunluk ve genişlik bilgilerine göre rastgele labirentler oluşturulması sağlanmıştır. Harita hakkında hiçbir bilgisi olmayan bir robotun haritanın çıkışını duvarları takip ederek ulaşması sağlanmıştır. Proje farklı görselleştirmeler veya farklı algoritmaların kullanılabilinmesi için soyut sınıflar ve arayüzler yardımıyla minimum efor ile geliştirilmeye açık bir hale getirilmiştir. Uygulamanın mevcut versiyonu sadece konsol üzerinde çalıştığı için ve konsolda sürekli olarak farklı arka plan renklerinde harita çizdirmek zaman maliyeti yüksek bir işlem olduğu ve kullanıcı dostu bir çözüm olmadığı için konsoldaki ilgili satırın, ilgili sütununda güncelleme yapılması sağlanarak sistemin optimize çalışması sağlanmıştır. Bu durum soyut sınıfların bir özelliği olduğu için sistemi, grafik kütüphaneleri kullanarak değiştirmek isteyen biri için optimal bir çözüm olacaktır.

KAYNAKLAR

[1] https://www.researchgate.net/publication/343187805_Comparison_of_Hand_Follower_and_Dead-End_Filler_Algorithm_in_Solving_Perfect_Mazes

[2] https://www.allaboutcircuits.com/projects/how-to-build-a-robot-follow-walls/

[3] https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

[4] https://www.geeksforgeeks.org/shortest-path-in-a-binary-maze/

[5] https://bradleycarey.com/posts/2012-08-15-maze-solving-algorithms-wall-follower/

[6] https://medium.com/swlh/fun-with-python-1-maze-generator-931639b4fb7e

[7] https://www.techiedelight.com/find-shortest-path-source-destination-matrix-satisfies-given-constraints/