Programlama Laboratuvarı II
Proje 1

Furkan Demirsoy 210201052
Kocaeli Üniversitesi Bilgisayar Mühendisliği

I. ÖNSÖZ

Bu proje Kocaeli Universitesi Bilgisayar Mühendisliği Programlama Laboratuvarı II dersi 1. projesidir. Bu raporda bu projeyi nasıl yaptığımız, yaparken nelerden faydalandığımız ve karşılaştığımız hataları nasıl çözdüğümüz gibi birçok konuda bilgilendirme yapılıcaktır.

II. GİRİŞ

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

Proje gerçekleştirimi ile öğrencilerin nesneye yönelik programlama ve veri yapıları bilgisinin pekiştirilmesi ve problem çözme becerisinin gelişimi amaçlamaktadır.

III. ÖZET

A. Proje açıklaması

Proje bir Gezgin Robot projesidir ve bu projede bizden 2 adet problemin çözülmesi beklenmektedir

- 1.Problem: Bu problemde bizden robotu ızgara (grid) üzerinde verilen hedefe engellere takılmadan en kısa sürede ve en kısa yoldan ulaştırmamız beklenmektedir. Robotu tüm ızgarayı değil, yalnızca gerekli yolları gezerek hedefe ulaşmasını sağlamalıyız.

- 1.Problemde Bizden İstenenler:

  Projede daha önceden paylaşılan bir url adresindeki .txt uzantılı bir dosyadaki bilgiler alınarak bir ızgara(grid) oluşturulması istenmektedir Bu text dosyasında 0,1,2,3

  rakamları vardır ve bizim programımız çalıştığında bu text dosyasına erişerek 0 rakamını robotun geçebileceği bir yol olarak anlıyacaktır.1,2,3 rakamlarını engel olarak algılayacaktır. Robotun başlangıç ve hedef

  noktaları ızgara üzerindeki uygun (engel veya duvar içermeyen) karelere rastgele belirlenicektir. Robot başlangıçta tüm ızgara dünyasını bilmiycek, sadece bir adım sonraki kareleri görebilecektir. Her adımda robotun öğrenmediği kareler bulutlu (kapalı) olarak gösterilecek, öğrenilen kareler ise açılarak ilgili karelerde bulunan nesneye göre (engel, duvar, yol, vs.) belirtilecektir.

  Tüm bu bilgiler doğrultusunda, robotun hedefe en kısa sürede ulaşabileceği en kısa yol, adım adım ızgara üzerinde gösterilecektir. Robotun daha önce geçtiği yerler belli olacak şekilde her

adımda yol üzerinde iz bırakması gerekmektedir. Hedefe ulaşıldığında ise başlangıç noktasından hedef konuma giden robota göre en kısa yol ızgara üzerinde ayrıca çizdirilecektir. Geçen toplam süre (sn cinsinden) ve kaç kare üzerinden geçildiği bilgileri ekranda gösterilmesi beklenmektedir.

- 2.Problem

  Bu problemde bizden robotu labirentteki çıkış noktasına ulaştırmanız beklenmektedir.

- 2. Problemde Bizden İstenenler

  Kullanıcı tarafından istenilen boyutlarda bir ızgara oluşturmamız gerekmektedir. Izgara üzerine 1 nolu tipte engeller yerleştirilerek labirent oluşturulmalıdır. Labirent
  içerisinde mutlaka çıkışa ulaşamayan yollar bulunmalıdır.
  Labirentin giriş ve çıkış noktaları dörtgen ızgaranın herhangi çapraz 2 köşesi olarak belirlenmesi gerekmektedir. 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 gösterilmesi beklenmektedir. Her adımda robotun daha önce geçtiği yollar üzerinde iz bırakması gerekmektedir. Robot hedefe ulaştığında giriş noktasından çıkış noktasına giden yol ızgara üzerinde çizdirilmesi beklenmektedir. Geçen toplam süre (sn cinsinden), kaç kare üzerinden geçildiği bilgileri ekranda gösterilmesi beklenmektedir.

- Sınıf Tanımlamaları

- Robot Sınıfı

  Robot sadece yukarı-aşağı ya da sağ-sol doğrultusunda hareket edebilmeli, çapraz yönde hareket etmemelidir. Robot ızgara dünyasında bulunduğu konumdan sadece bir birim sonrası ile ilgili bilgileri görebilir. Haritanın tümünü görmemelidir.

- Izgara Sınıfı
  Bu sınıfta problem 1 için ızgara tasarımı verilen url adresindeki text dosyasına göre oluşturulurken problem 2 için ızgara kullanıcıdan alınacak boyut bilgisine göre oluşturulmalıdır.

- Engel Sınıfı: Engel sınıfında problem 1 için

  üç farklı tipteki nesneler kullanarak engellerin oluşturulması url adresindeki text dosyasına göre yapılırken problem 2 için labirent oluşumu 1 nolu tek bir nesne türü kullanılarak uygulama içerisinde rastgele oluşturulacaktır.

- Uygulama Sınıfı:
  Uygulama içerisinde robotun problem 1 ve problem 2 deki hedefe ulaşma süresi, kaç kare üzerinden geçildiği gibi bilgilerin tutulduğu ve ekranda gösterilmesi fonksiyonlarını sağlamalıdır

- Arayüz ve Görsellik ile İlgili İsterler

  Problem 1 ve problem 2 için uygulama ilk çalıştırıldığında başta robotun nasıl hareket ettiği simüle edilecek ancak sonrasında zaman kaybı olmaması “Sonuç Göster” butonu aracılığıyla sonuç kullanıcıya direk gösterilecektir. Problem 1 için arayüzde 3 buton yer alacak bunlar “URL Değiştir” butonu, “Çalıştır” Butonu ve “Sonuç Göster” butonları olacaktır. Proje ilk çalıştırıldığında 1. url adresindeki text dosyasına göre ızgara ve engel tasarımı yapılarak ekrana getirilecektir. Url Değiştir butonuna tıklandığında 2. url adresindeki text dosyasındaki veri kullanılarak ızgara ve engel yapısı tekrar oluşturulmalıdır. Butona her tıklandığında url adresi 1.ve 2. url adresleri arasında değiştirilerek tasarım yenilenecektir. Sonrasında Çalıştır butonuna tıklandığında tüm ızgara alanı başta bulutla kapatılarak engel ve yolların görünümü engellenmelidir. Robotun yavaş hareketi bu sırada gösterilmeli ve robot

hareket ettikçe gördüğü yerlerdeki (bir kare sonrası) bulutlar kaldırılacaktır. Sonuç Göster butonuna tıklandığında ise oyun sonuca ilerletilerek robotun sırasıyla gezdiği kareler ve bulduğu en kısa yol ekranda çizdirilecek ayrıca bir text dosyasına yazdırılarak kaydedilecektir. Yine kare sayıları ve ne kadar süre tuttuğu bilgileri ekranda gösterilecektir. Problem 2 için de problem 1'de olduğu gibi üç buton arayüzde yer alacak bunlardan ilki "Labirent Değiştir" butonu olması gerekmektedir. Bu butona her tıklandığında labirent tasarımı değiştirilecektir. Yine Çalıştır ve Sonuç Göster butonları problem 1 deki fonksiyonların aynısını yerine getirecektir.

IV. YÖNTEM

A. Araştırma

İlk olarak labirent içerisindeki en kısa yolun bulunması ve gösterilmesi için bir takım algoritmalar araştırdım ve robotun optimize hareketi için 2 seçenek olduğunu öğrendim en az kare geçerek hareket etmesi için A* algoritmasının ve bir Backtraking algoritması kullanılabileceğini öğrendim ancak her ne kadar A* algoritması daha kısa bir yol versede bu algoritma hedef noktasının koordinatlarını bilerek en kısa yolu bulduğu için ve bizim projemizde robot en başta hedef noktasının konumunu bilmediği için robotun en az kare geçerek hareket etmesi için kendim tarafından özelleştirilen bir Backtraking algoritması kullanmaya karar verdim.

Robotun hareketinin nasıl olacağına karar verdikten sonra robotun öğrendiği yollar içerisinden en kısa olan yolu bulması durumunu araştırmaya başladım ve bunun için Dijkstra en kısa yol algoritmasının en garanti sonucu vericeğini öğrendim.

Projede istenen robotun adım-adım kullanıcıya hareket ettiği yerleri göstermesi için programın yavaşlatılması gerektiğine ve bunun için de bir timer kullanmaya karar verdim.

Daha sonra 2. problemde labirentin mutlaka hedef noktasına bir yol ve çıkmaz yollar da bulunacak şekilde kullanıcıdan alınacak boyut bilgisine göre rastgele bir şekilde oluşması için bir Recursive-Backtraking algoritması kullanmaya karar verdim.

B. Arayüz

Araryüzün oluşturulması içi Swing kütüphanesi componentlerini kullandım ve bunun için toplamda 3 pencere açılmasını sağladım ilk pencere GUI isimli class ın içerisinde oluşturdum ve bu pencere sayesinde problem seçimi ve seçilen probleme göre url bilgisi ve boyut bilgilerinin contentpaneler ile kullanıcıdan alınmasını sağladım.

ilk pencere içerisindeki 1.problem butonuna basıldığında url seçimi yapıldıktan sonra verilen url adresindeki text dosyası içerisindeki bilgiler çekilerek dinamik bir şekilde ızgara ve labirent oluşturuldu ve bu pencerede de Çalıştır ve Sonuç Göster butonları eklendi Çalıştır butonuyla gölegeleme başlatılarak robotun hareket etmesi ve en sonda da en kısa yolu ve geçtiği kare sayısı ve zaman bilgileri kullanıcıya gösterildi ayrıca bu gösterimler swing kütüphanesindeki Graphics kütüphanesi ve paint fonksiyonuyla çizdirildi.

2.problem için yine 1. penceredeki butona basıldıktan sonra 2 adet contentpane ile genişlik ve uzunluk bilgileri pixel cinsinden alındıktan sonra istenilen boyutlarda ızgara yine Graphics kütüphanesinin paint fonksiyonu içerisinde oluşturuldu ve yine Çlışstır ve Sonuç göster butonlarıyla robotun hareketi,gölgeleme ve zaman kare bilgisi ve en kısa yol gösterildi ayrıca Labirent Değiştir butonuyla labirentin rastgele bir şekilde tekrar oluşturulması sağlandı.

C. Robotun Hareketi

Robotun en az sayıda kare geçerek hareket etmesi için bir Backtraking algoritması kullandım bu algoritma aşağıdaki şekilde çalışmaktadır:

- ziyaret edilmediyse ve engel değilse sola git
- yukarıdaki olmuyorsa ve ziyaret edilmediyse ve engel değilse sağa git
- yukarıdaki olmuyorsa ve ziyaret edilmediyse ve engel değilse yukarı git

- yukarıdaki olmuyorsa ve ziyaret edilmediyse ve engel değilse aşağı git
- yukarıdaki olmuyorsa ve engel değilse sola git
- yukarıdaki olmuyorsa ve engel değilse sağa git
- yukarıdaki olmuyorsa ve engel değilse yukarı git
- yukarıdaki olmuyorsa ve engel değilse aşağı git
- Yukarıdaki durumlardan hiçbiri geçerli değilse kaçış planı devreye girer bu kaçış planı için ziyaret ettiği düğümler Coordinate sınıfında ArrayListVisited isimli bir listede biriktirilir ve bu listeden en sondan başlayarak kareler ve ziyaret edilen karenin tüm komşuları çekilir.Robotun hareket etmesi için bu çekilen karelerin içinden ziyaret edilmeyen ilk komşusunun bulunduğu konuma hareket eder.

D. Izgara Oluşumu

Bilgiler url içerisindeki txt dosyasından çekildikten ya da 2. problem için rastgele bir şekilde oluşturulduktan sonra gelen bilgiler yeni satıra geçene kadar 0,0 koordinatından başlanarak x konumu 30 arttırılır ve g.drawReact fonksiyonuyla kare köşeleri,g.fillReact ile renkleri belirlenerek ızgaranın satırları oluşturulur yeni satıra geçildiğinde ise x koordinatı tekrar 0 getirilir ,y koordinatı 30 arttırılır ve tekrar yukarıda bahsedilen işlemler gerçekleşir ayrıca karelerin eklenme sırasına göre karelere bir id numarası verilir ve bütün bu x,y,id bilgileri koordinat sınıfında tutulur ayrıca kareler eklenirken kareler PassObjectXY gibi arraylistlerin içerisinde x,y,id bilgileri depolanır.

E. Gölgeleme

Gölgelemenin sağlanması için SeenList isimli coordinate sınıfındaki arraylist içerisindeki id bilgisi alınır ve bu sayede robotun hareket ettiği karelerin ve bu karelerin komuşusu olan karelerin id numaralarına göre çalıştır butonuna basılınca g.drawReact ve g.fillReact fonksiyonlarıyla görülen kareler engel ya da passobject olup olmamları durumuna göre çizilir.

F. En Kısa Yolun Bulunması

En kısa youl bulunması için seenList arraylisti içerisine robotun geçtiği ve etrafında komşu olan karelerin x,y,id numaralarını liste içerisine eklenir ve eklenen bu karenin id bilgileri bir graf veri yapısında tutulur ayrıca her 2 kare arasına da maaliyet olarak 10 sayısı eklenir daha sonra robot hedef noktasının koordinatlarını gördüğünde dijkstra algoritması çalıştırılır bunun için başlangıç noktasının id numarası startVertex olarak,hedef noktasının id numarası TargetVertex olarak veriilir ayrıca eklediğimiz graf veri yapısıda djkstraya eklenir bu algoritma içerisindeki fonksiyon içerisinden başlangıç noktasından hedef noktasına en kısa yoldan gidilmesini sağlıycak karelerin id bilgileri döndürülür ve hedef noktasına gidilmesi durumunda bu id numaralarının x,y koordinatları tespit edilerek bu kareler Color.Pink rengi ile boyanarak kullanıcıya gösterilir.

G. Rastgele Labirent Oluşumu

2.Problemde boyut bilgileri kullanıcıdan alındıktan sonra bu boyutlardaki labirentin rastgele oluşturulması için recursive-Backtraking algoritması kullanılmıştır bu algoritma için Obstacle sınıfındaki Configure fonksiyonu ile gerekli boyuttaki yol ve engel olabilceke noktalar belirlenir daha sonra yine aynı class içerisinde bulunan generate fonksiyonu içerisinde rastgele bir şekilde çıkış noktasına bir yol bulunacak şekilde aradaki bazı engeller yıkılarak rastgele labirent oluşturulur.

H. Deneysel Sonuçlar

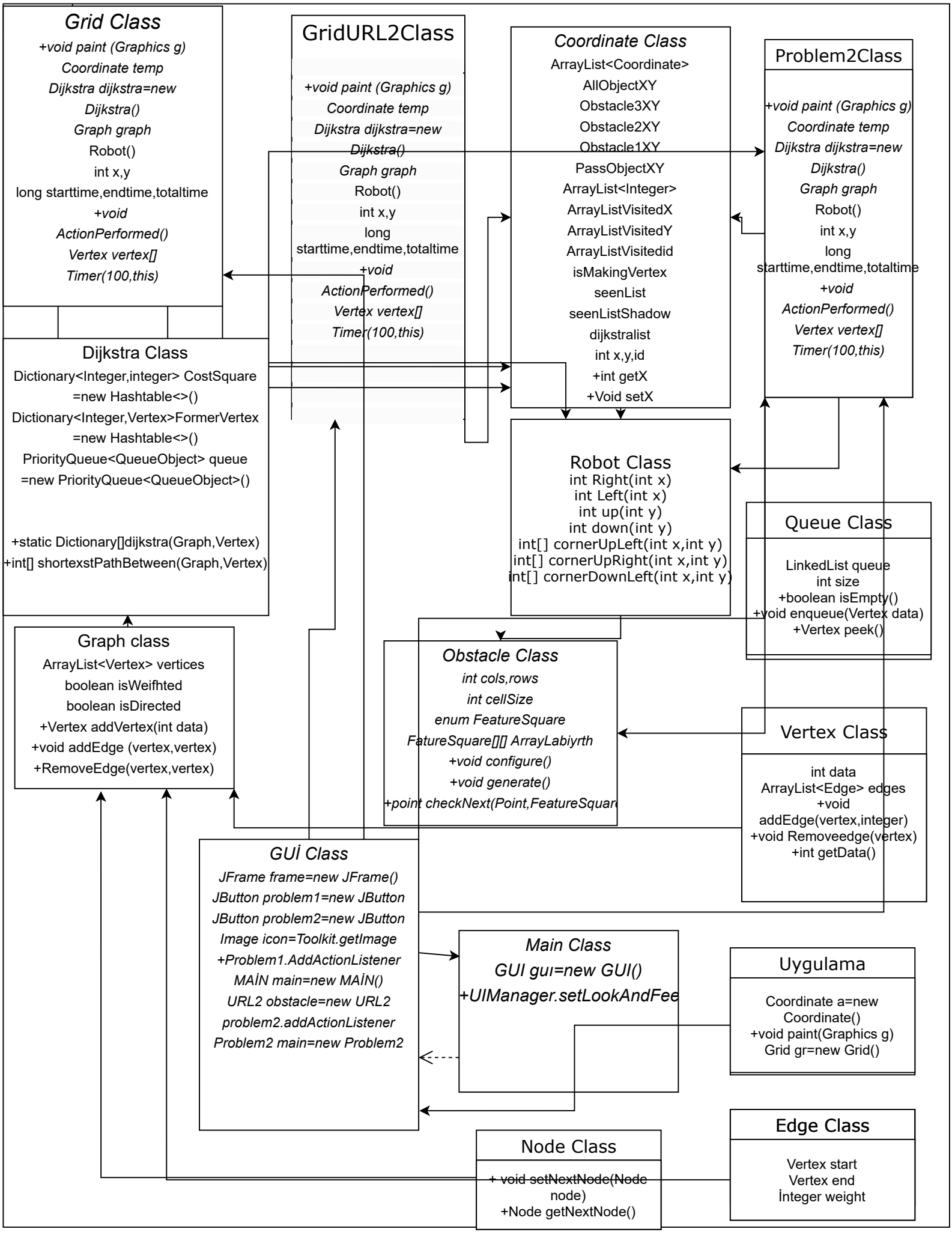
- Kullanıcıya adım-adım robotun hareketinin gösterilmesi için yapılan araştırma sonucunda timer kullanımı öğrenildi
- En kısa youln bulunması için kareler dinamik bir şekilde Vertex olarak eklenirken aynı kare id sinin Vertex olarak eklenmesi durumunda Dijkstra algoritmasının çalışmasında problem olduğu anlaşıldı bu sebepten aynı vertexin tekrar grafa eklenmesini önliycek bir if-else koşul durumu va algoritma geliştirimi öğrenildi

I. Sonuç

- Graf veri yapısının kurulması öğrenildi
- Djkstra ,A* gibi algoritmalar ve optimizasyon problemlerinin mantığı öğrenildi
- Swing kütüphanesinin içerisindeki Graphics kütüphane fonksiyonlarının kullanımı öğrenildi.
- Backtraking algoritması ve Recursive-Backtraking algoritması öğrenildi
- Thread ve timer mantığının ne olduğunun anlaşılması sağlandı
- Probelm çözme becerisi ve Veri Yapıları ve Algoritma yeteneğinin geliştirilmesi sağlandı.

J. Kaynakça

- https://stackoverflow.com/questions/4044726/how-to-set-a-timer-in-java
- https://bilgisayarkavramlari.com/2009/11/01/geri-izleme-algoritmasi-backtracking-algorithm/
- https://www.muhendisbeyinler.net/dijkstra-algoritmasi/
- https://stackabuse.com/graphs-in-java-a-star-algorithm/
- https://www.baeldung.com/cs/maze-generation


Grid Class
+void paint (Graphics g)
Coordinate temp
Dijkstra dijkstra=new
Dijkstra()
Graph graph
Robot()
int x,y
long starttime,endtime,totaltime
+void
ActionPerformed()
Vertex vertex[]
Timer(100,this)
GridURL2Class
+void paint (Graphics g)
Coordinate temp
Dijkstra dijkstra=new
Dijkstra()
Graph graph
Robot()
int x,y
long
starttime,endtime,totaltime
+void
ActionPerformed()
Vertex vertex[]
Timer(100,this)
Coordinate Class
ArrayList<Coordinate>
AllObjectXY
Obstacle3XY
Obstacle2XY
Obstacle1XY
PassObjectXY
ArrayList<Integer>
ArrayListVisitedX
ArrayListVisitedY
ArrayListVisitedid
isMakingVertex
seenList
seenListShadow
dijkstralist
int x,y,id
+int getX
+Void setX
Problem2Class
+void paint (Graphics g)
Coordinate temp
Dijkstra dijkstra=new
Dijkstra()
Graph graph
Robot()
int x,y
long
starttime,endtime,totaltime
+void
ActionPerformed()
Vertex vertex[]
Timer(100,this)
Dijkstra Class
Dictionary<Integer,integer> CostSquare
=new Hashtable<>()
Dictionary<Integer,Vertex>FormerVertex
=new Hashtable<>()
PriorityQueue<QueueObject> queue
=new PriorityQueue<QueueObject>()
+static Dictionary[]dijkstra(Graph,Vertex)
+int[] shortexstPathBetween(Graph,Vertex)
Robot Class
int Right(int x)
int Left(int x)
int up(int y)
int down(int y)
int[] cornerUpLeft(int x,int y)
int[] cornerUpRight(int x,int y)
int[] cornerDownLeft(int x,int y)
Queue Class
LinkedList queue
int size
+boolean isEmpty()
+void enqueue(Vertex data)
+Vertex peek()
Graph class
ArrayList<Vertex> vertices
boolean isWeifhted
boolean isDirected
+Vertex addVertex(int data)
+void addEdge (vertex,vertex)
+RemoveEdge(vertex,vertex)
Obstacle Class
int cols,rows
int cellSize
enum FeatureSquare
FatureSquare[][] ArrayLabiyrth
+void configure()
+void generate()
+point checkNext(Point,FeatureSquar
Vertex Class
int data
ArrayList<Edge> edges
+void
addEdge(vertex,integer)
+void Removeedge(vertex)
+int getData()
GUİ Class
JFrame frame=new JFrame()
JButton problem1=new JButton
JButton problem2=new JButton
Image icon=Toolkit.getImage
+Problem1.AddActionListener
MAİN main=new MAİN()
URL2 obstacle=new URL2
problem2.addActionListener
Problem2 main=new Problem2
Main Class
GUI gui=new GUI()
+UIManager.setLookAndFee
Uygulama
Coordinate a=new
Coordinate()
+void paint(Graphics g)
Grid gr=new Grid()
Edge Class
Vertex start
Vertex end
İnteger weight
Node Class
+ void setNextNode(Node
node)
+Node getNextNode()