KOCAELİ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

PROGRAMLAMA LAB. 2 -1. PROJE

SUDE YAĞMUR GÖÇGÜN / 190202046@kocaeli.edu.tr

CANBERK TEZCAN / 190202033@kocaeli.edu.tr

ŞİRİNLER

1-ÖZET

Bu proje ile nesneye yönelik programlama ve veri yapıları algoritmalarını kullanarak Şirinler oyunu tasarlamamız beklenmektedir. Seçilen oyuncunun Labirent içerisinde puanlarını bitirmeden önce şirineye ulaşması gerekmektedir. Hedefe doğru giderken oyuncuyu birçok zorluk beklemektedir. Oyun içerisinde iki farklı oyuncu (farklı özellikleri olan) ve iki farklı düşman karakterlerimiz (farklı özellikleri olan) olacaktır. Kullanıcı oyun başlamadan önce seçtiği oyunculardan birini klavye yardımı ile kontrol edecek ve Şirine'ye ulaştırmaya çalışacaktır. Yine oyun başlamadan önce seçmiş olduğu düşman karakterlerden biri veya birden fazlası da onu durdurmaya çalışacaktır. Başlangıçta seçilen oyuncu 20 Puanla oyuna başlayacak ve oyuncu puanı 0 ya da 0'ın altına düştüğünde oyun oyuncunun başarısızlığı ile sonlanacaktır. Oyunu bir arayüz üzerinde gerçekleştirmemiz istenmiştir.

2-Giriş

Proje için Java programlama dili ve Netbeans geliştirme ortamını kullandık. Java programlama dili; açık kodlu, nesneye yönelik, zeminden bağımsız, yüksek verimli, çok işlevli, yüksek seviye, adım adım işletilen bir dildir. Netbeans platformu; Oracle tarafından geliştirilen bir java geliştirme ortamıdır (IDE). Özellikle kullanıcı arayüzü tasarımında sağladığı kolaylıklardan dolayı tercih edilmektedir. Bizim de Netbeans'i tercih etme sebeplerimizin başında GUI tarafında sağladığı kolaylıklardır.

3- Yöntemler ve Program Mimarisi

A-Dijkstra Algoritaması

Projede düşman karakterden oyuncu karaktere giden en kısa yolu hesaplamamız için dijkstra algoritması kullanmamız istenmişti. Harita üzerinde 1 değeri alan her noktayı (yani oyuncunun hareket edebileceği her nokta) bir düğüm olarak oluşturduk. Her bir düğümün de sağ sol üst ve alt konumlarından ilerleyebileceği noktaları kaynak noktaya komşu düğüm olarak ekledik. Böylece gidebilecek tüm düğümler ve komşularının bilgilerini tutmuş olduk. Algoritma başlangıçta bütün düğümlere henüz erişim olmadığını kabul ederek sonsuz değeri atar. Yani başlangıç durumunda henüz hiçbir düğüme gidemiyoruz. Ardından başlangıç düğümünün komşusu olan bütün düğümleri dolaşarak bu düğümlere ulaşım mesafesini

günceller. Baş düğümün komşuları güncellendikten bir sonraki adımda da komşularının komşularını güncelleye güncelleye ilerleyerek hedef düğüme kadar ulaşır. Ve sonuçta baş düğümden hedef düğüme olan en kısa mesafeyi bulmuş olur.

ZAMAN KARMAŞIKLIĞI

Algoritmamızda labirenti bitişiklik listesi şeklinde temsil ediyoruz ve Priorityqueue kullanıyoruz.

- Bitişik liste gösterimi ile, grafiğin tüm köşeleri O (V + E) zamanında gezilebilir.
- Çıkarma ve mesafe değerini azaltma gibi işlemler O (logV) süresini alır.
- Böylece, toplam zaman karmaşıklığı O (E + V) x O (logV) olur ve bu da O ((E + V) x logV) = O (ElogV) olur.

B-Temel Bilgiler

Projemiz Altin.java , Azman.java , DijkstraEnkisaYol.java , Dugum.java , Dusman.java, Gargamel.java , GozlukluSirin.java , Karakter.java , KomsuDugum.java ,Lokasyon.java, Mantar.java , Maze.java, Obje.java, Oyuncu.java,Puan.java,TembelSirin.java , GirisEkrani.java,BitisEkraniWin.java, BitisEkraniOver.java , Main.java olmak üzere 20 sınıftan oluşmaktadır.

Obje.java

Obje sınıfından türeyecek 2 adet sınıfımız olduğu için oluşturulan bir super class.

Altin.java

Obje sınıfından kalıtım alan altin.java kendine özel private değer ve adet değişkenini tutar. Private değişkenlerin her biri için ise set ve get methodlarını yazdık.Altın.java parametreli ve parametresiz olmak üzere 2 de constructora sahip.

Mantar.java

Obje sınıfından kalıtım alan mantar.java private olarak int değer değişkenine, bu değişkenin set ve get methodlarına , parametreli ve parametresiz olmak üzere 2 de constructora sahip.

Lokasyon.java

Oyuncuların ve düşman karakterlerin ilerleyeceği koordinatları tutacak x,y değişkenleri ve get set methodlarına sahip.

Karakter.java

Karakter sınıfımız hem oyuncu hem de düşman sınıflarının ortak özellikleri olan ID, tur , Ad değişkenlerini tutar. Bunların yanında karakterlerin ilerleyeceği x ve y koordinatlarını da lokasyon.java dan extend almakta. Private özelliklerimizin get ve set, public void EnKisaYol() methodu ve parametreli ve parametresiz constructora sahip.

Dusman.java

Dusman sınıfı tüm özelliklerini süper() ile karakter sınıfından extend almakta.

Gargamel.java

Gargamel.java Dusman.java'nın alt classı bu yüzden özelliklerinin hepsini düşman.java'dan kalıtım almakta.

Azman.java

Azman.java Dusman.java'nın alt classı bu yüzden özelliklerinin hepsini düşman.java'dan kalıtım almakta.

Oyuncu.java

Özelliklerini karakter.java dan extends alır. Kendine ait private skor değişkenine sahip. Skor değişkeninin get ve set methodları var.

Ayrıca oyuncunun puanını gösterecek olan public int PuaniGoster() methoduna sahip.

Puan.java

Oyuncu.java dan extend alır ve PuanıGoster() methodunu override ederek her oyuncuya göre özelleştirir.

TembelSirin.java

Özelliklerini oyuncu.java dan kalıtım alır.

GozlukluSirin.java

Özelliklerini oyuncu.java dan kalıtım alır.

GirisEkrani.java

Oyuncu seçimi sırasında görünecek ekranı oluşturduk.

BitisEkraniWin

Oyuncunun puanı bitmeden şirineye ulaştığında görünecek ekranı oluşturduk.

BitisEkraniOver.java

Oyuncunun düşmana yakalanıp puanının 0'landığında görünecek olan ekranı oluşturduk.

Dugum.java

Bir düğüm yapısı oluşturabilmek için yazdığımız bu sınıfta private String name; private List<KomsuDugum> adjacenciesList; private boolean visited;

private Dugum predecessor;

private double distance =Double.MAX_VALUE;

private int x,y; olmak üzere özelliklerimiz ve özelliklerimizin get ve set methodları var. Bunların yanışına parametresiz ve parametreli iki constructor yazdık.

```
public Dugum(String name,int x,int y) { this.name = name;
this.x=x;
this.y=y;
this.adjacenciesList = new ArrayList<>();}
```

KomsuDugum.java

Dijkstra algoritması için düğümlerimize komşu düğümler ekleyebilmemize yarayacak bu sınıfta

```
private double weight;
private Dugum startDugum;
private Dugum HedefDugum;
```

özelliklerimiz, özelliklerimizin get ve set methodları ve özellikleri parametre alan constructor var.

DijkstraEnkisaYol

Dijkstra algoritmasının kodlandığı class. İşe önce **public void KisaYolHesaplama()** methoduna en kısa yolu hesaplayacağımız baş düğümü ve tüm düğümlerin bulunduğu listeyi göndermekle başladık. Methodumuzun içine girdiğimizde öncelikle kısayol hesabını tekrar tekrar farklı baş düğümler için kullanabilmemiz için düğüm listesindeki tüm düğümlerin bilgilerini sıfırladık.

```
for (sıfırdan düğümlistenin sonuna kadar) {
        if (dugumliste[l] boş değilse) {
          dugumliste[l] ziyaret edildi bilgisi=false
           dugumliste[l] mesafesi=sonsuz
           dugumliste[l] OncekiDugum(null);
        } }
```

Bu işlemden sonra algoritmanın işleme mantığında da olduğu gibi baş düğümün mesafesini = 0 ve ziyaret edildi mi bilgisini= true yaptıktan sonra bir PriorityQueue<Dugum> priorityQueue oluşturup methoda gelen baş düğümü kuyruka ekledik.

```
BasDugum.setMesafe(0);
    PriorityQueue<Dugum> priorityQueue = new
PriorityQueue<>();//öncelik sıralamalı kuyruk yapısı
    priorityQueue.add(BasDugum);
    BasDugum.setVisited(true);
```

Devamındaki işlemler, kuyruk boş değilse, kuyruktan çektiğimiz elemanın komşu düğümlerini gezerek tek tek mesafelerini ,ziyaret edildi bilgilerini güncellemek ve bu işlemi komşunun komşuları şeklinde hedef düğüme ulaşana kadar devam ettirmek şeklinde sürüyor. Kaba kod şeklinde anlatmak gerekirse:

while (priorityQueue boş değilken){

kaynakdugum = priorityQueue ilk elemanı çek ve kuyruktan sil

for (kaynakdugumun komşu listesini teker teker gez) {

Dugum v = komsuDugum.getHedefDugum();

if (v ziyaret edilmediyse){

double newMesafe = kaynakDugum.getMesafe() + komsuDugum.getWeight();

if (yeni mesafe küçükse v.getMesafe()) {

priorityQueue.remove(v);

v.setMesafe((int) newMesafe);

v.setOncekiDugum(kaynakDugum);

priorityQueue.add(v);

}}}

For döngüsü tamamlandığında kaynakdügümün tüm komşuları güncellendiğinden ziyaret edildi bilgisi true olur ve kod while dönerek kuyruka eklenen ikinci düğüm için aynı işlemleri tekrarlar.

}

Böylece artık gönderdiğimiz baş düğümden herhangi bir düğüme olan en kısa mesafe bilgisine artık sahibiz

Classtaki bir diğer **public List<Dugum> getEnKısaYol(Dugum hedefDugum) {}** methodunda ise başdüğümden gitmemizi istediğimiz hedef düğüm arasındaki en kısa yolu bir listeye sıralıyoruz .

List<Dugum> path = new ArrayList<>();

for (Dugum dugum = hedefDugum; dugum != null; dugum = dugum.getOncekiDugum()) {

path.add(dugum);

}

Maze.java

Oyunumuz genel anlamda main harici burada döndü. Gerekli olan değişkenleri tanımladıktan sonra Maze için 2 farklı constructor oluşturduk.

İf (2 düsman karakter varsa)

Public Maze(parametreler){

……..

}

Else

Public Maze(parametreler){

……..

}

public void paint(Graphics g) methodunda öncelikle txt den okunan haritaya göre labirenti ekrana çizdirdik.

for (0 dan maze matrisinin uzunluğuna kadar) {

for (0 dan sütun sonuna kadar) {

maze[satır][sütun]

eğer 1 ise kutuyu beyaz boya;

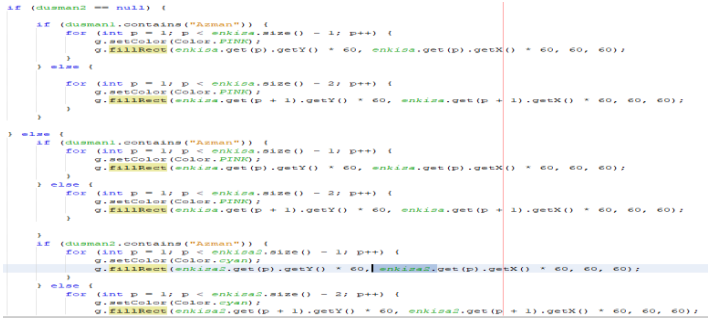
eğer 0 ise kutuyu gri boya;

}

Karakter seçiminin gözlüklü mü tembel mi olmasına göre ekrana koyduk.

Aynı işlemi dosyadan okunan gargamel veya azman veya her ikisi için de kapı bilgisine göre tekrarladık.

Sonrasında düşman karakteri her durum için kontrol ederek oyuncuya olan en kısa mesafeyi renkli şekilde ekranda gösterdik.



En son olarak da oyuncu skoru değişkenimizi kontrol ettik.

İf(oyuncuskor <=0 ise)

BitisEkraniOver çağır.

İf(oyuncu koordinatlar x ve y == şirine koordinatlar x ve y)

BitisEkraniWin çağır.

protected void processKeyEvent(KeyEvent key) oyun genel anlamda bu methodun içinde her şeyi belirleyecek olan oyuncunun sağ,sol,aşağı,yukarı hareketlerine göre gerçekleşti.

Sağ,sol,yukarı,aşağı caselerinin hepsi için oyun gidişatı aynıydı.

1) Bastığı tuşa göre oyuncunun x ve y sini güncelle
2) Oyuncunun gittiği yeni konumu DijkstraEnKisaYol methoduna gönder.
3) Dijkstradan dönen konuma göre düşmanı ilerlet
4) Repaint() ile işlemi tekrarla.

Main.java

Dosya okuma işlemlerini yapıp düşmanları çektik. Haritayı da okuyup bir matrise attık.

Matristeki haritayı dijsktrada kullanabilmek için düğüme dönüştürüp bir düğüm listesinde tutabilmemiz gerekiyordu.

for (0 dan başlayıp 11e kadar) {

for (0 dan 13e kadar) {

if (maze2[satır][sütun] == 1) {

bu noktayı düğüm listesine yeni bir düğüm olarak ekle }

} }

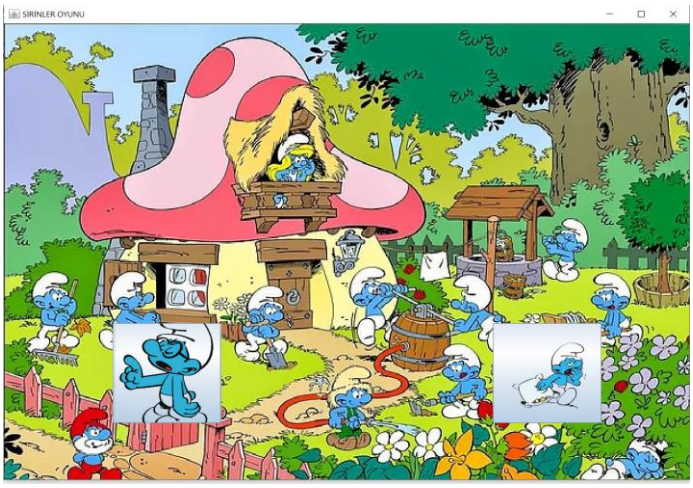
Arayüz ve ekrandan oyuncu seçimi ile ilgili işlemleri de tamamladıktan sonra tek bir işlem kaldı.

Her düğümün gidebileceği komşu düğümleri **KomsuDugumEkle()** methodu ile düğüm bilgisine ekledik.

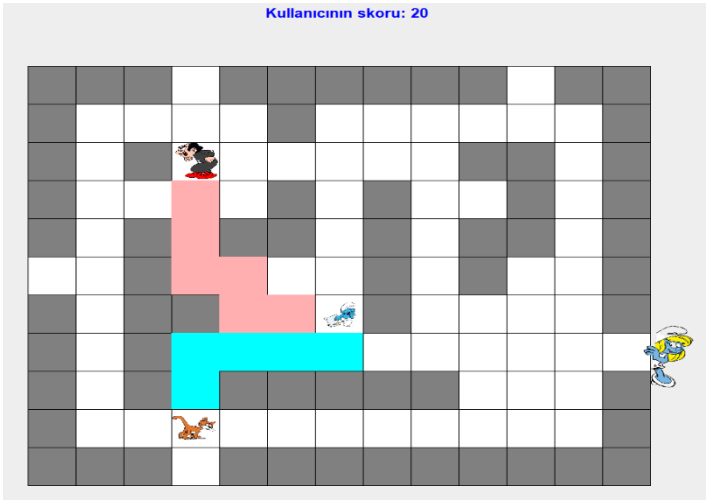
```
dugumliste[13].KomsuDugumEkle(new KomsuDugum(10, dugumliste[13], dugumliste[12]));
dugumliste[13].KomsuDugumEkle(new KomsuDugum(10, dugumliste[13], dugumliste[23]));
dugumliste[13].KomsuDugumEkle(new KomsuDugum(10, dugumliste[13], dugumliste[14]));
```

4- Programın Tasarımı

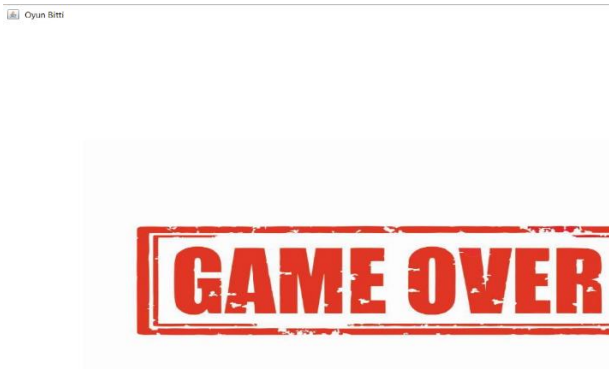
Program temelde 4 ekrandan oluşuyor.



Giriş ve oyuncu karakter seçim ekranı



En kısa yolların gösterildiği oyun ekranı.



Kullanıcı şirineye ulaşmadan puanı sıfırlandığında görünen ekran

Kullanıcı puanı bitmeden şirineye ulaştığında görünen ekran.

5-Deneysel Sonuçlar

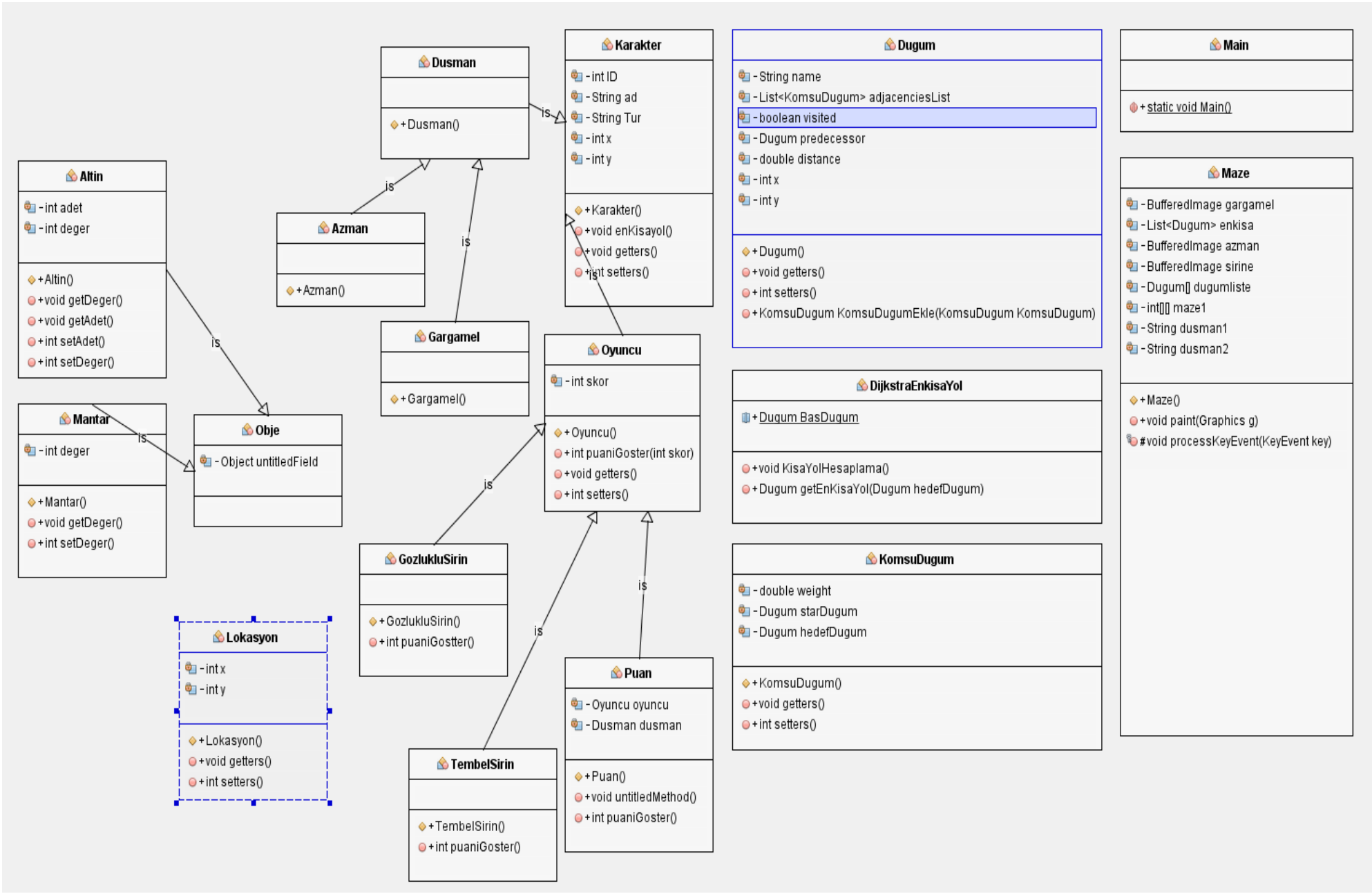
Projeyi gerçekleştirmeden önce dijkstra algoritması konusunda uzun araştırmalar ve denemeler yaparak bu konuda çok fazla tecrübe edinmiş olduk.Dinamik arayüz elementleri hakkında da oldukça çeşitli bilgi sahibi olma fırsatı bulduk.

5-Sonuç

Bu projeyi gerçekleştirerek en kısa yol algoritmaları hakkında oldukça önemli bilgiler ve tecrübeler edindik. Veri yapıları ve nesne yönelimli programlama hakkında sahip olduğumuz teorik bilgileri uygulamaya dökme fırsatı bulduk.

6-Kaynakça

1) https://www.baeldung.com/java-dijkstra
2) https://stackabuse.com/graphs-in-java-dijkstras-algorithm/
3) https://www.javatpoint.com/java-swing
4) https://github.com/danallen1ny/Maze-game/blob/master/GUI


Dusman
+Dusman()
Karakter
-int ID
-String ad
-String Tur
-int x
-int y
+Karakter()
+void enKisayol()
+void getters()
+int setters()
Dugum
-String name
-List<KomsuDugum> adjacenciesList
-boolean visited
-Dugum predecessor
-double distance
-int x
-int y
+Dugum()
+void getters()
+int setters()
+KomsuDugum KomsuDugumEkle(KomsuDugum KomsuDugum)
Main
+static void Main()
Altin
-int adet
-int deger
+Altin()
+void getDeger()
+void getAdet()
+int setAdet()
+int setDeger()
Azman
+Azman()
Gargamel
+Gargamel()
Maze
-BufferedImage gargamel
-List<Dugum> enkisa
-BufferedImage azman
-BufferedImage sirine
-Dugum[] dugumliste
-int[][] maze1
-String dusman1
-String dusman2
+Maze()
+void paint(Graphics g)
#void processKeyEvent(KeyEvent key)
Mantar
-int deger
+Mantar()
+void getDeger()
+int setDeger()
Obje
-Object untitledField
Oyuncu
-int skor
+Oyuncu()
+int puaniGoster(int skor)
+void getters()
+int setters()
DijkstraEnkisaYol
+Dugum BasDugum
+void KisaYolHesaplama()
+Dugum getEnKisaYol(Dugum hedefDugum)
GozlukluSirin
+GozlukluSirin()
+int puaniGostter()
KomsuDugum
-double weight
-Dugum starDugum
-Dugum hedefDugum
+KomsuDugum()
+void getters()
+int setters()
Lokasyon
-int x
-int y
+Lokasyon()
+void getters()
+int setters()
Puan
-Oyuncu oyuncu
-Dusman dusman
+Puan()
+void untitledMethod()
+int puaniGoster()
TembelSirin
+TembelSirin()
+int puaniGoster()
is