**Şirinler Oyunu***Samet Yavuz, Buğra Burak Önal*

Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi

190201066@kocaeli.edu.tr, 190201034@kocaeli.edu.tr

**Özet**

Bu proje ile nesneye yönelik programlama ve veri yapıları algoritmalarını kullanarak Şirinler oyunu tasarlamanız beklenmektedir. Seçilen oyuncunun Labirent içerisinde puanlarını bitirmeden önce Şirine’ye ulaşması gerekmektedir.

Hedefe doğru giderken oyuncuyu birçok zorluk beklemektedir. Oyun içerisinde iki farklı oyuncu ve iki farklı düşman karakterleriniz 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. Şekil 1’de, projede kullanacağınız harita verilmiştir. Mavi renkli kutu oyuncuların (yani kullanıcı tarafından kontrol edilecek olan karakter) başlama noktası olacaktır. Kırmızı oklar ise düşman karakterlerin labirente giriş yapabileceği kapılar olacaktır. Kullanıcının kontrol ettiği oyuncu puanları bitmeden önce Prensese ulaşmalıdır aksi halde oyuncu başarısız olur. Şekil 1: Projede kullanılacak harita Programlama Dili: Proje C++ veya Java dili kullanılarak gerçekleştirilecektir. En kısa yol algoritması: Sadece Dijisktra algoritması kullanılacaktır.

**Giriş**

Sizden bir arayüz tasarlamanız beklenmektedir. Şekil 1’de verilen haritayı oluşturmak için grafik kütüphanelerinden yararlanabilirsiniz. Bunun için graphics.h veya allegro.h gibi kütüphanelerinden faydalanabilirsiniz (Grafik için kütüphane sınırlaması yoktur). Başlangıçta seçilen oyuncu 20 Puanla oyuna başlayacak ve oyuncu puanı 0 ya da 0’ın altına düştüğünde oyun oyuncunun başarısızlığı ile sonlanacaktır.

**Yöntem**

Dosyadan alınan maze(labirent) diziye atılır.Daha sonra Neighbours metodu ile Hareket edilebilin kareler kom diziye eklenir.Kenarla

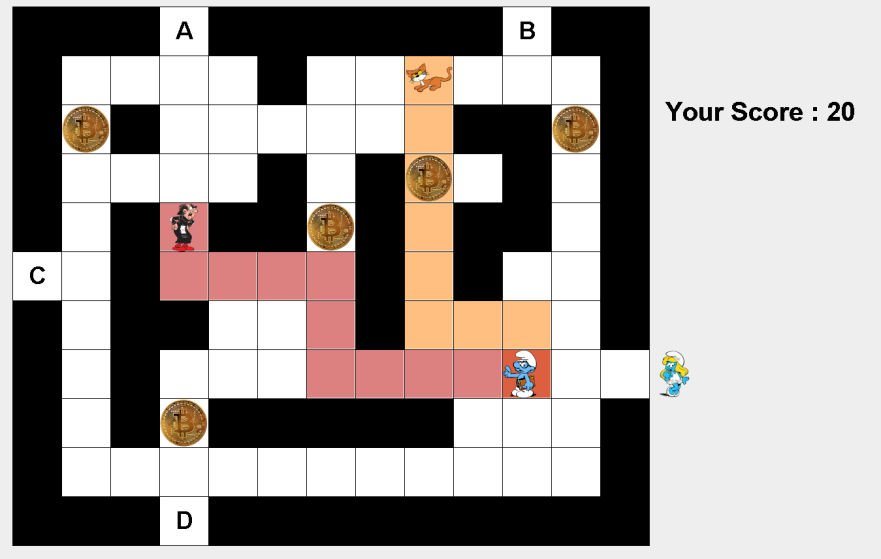
Dist, pred ve Visited adlı liste matrisin boyutu kadar eleman atanır.Tüm uzaklıklar maksimum değer olarak atanır.Ve başlangıcın uzaklığı 0 olarak atanır.Minvertex fonksiyonu çağrılır.Sonraki kare ziyaret edilmediyse ve uzaklık değeri x den küçük ise y iye eşitlenir ve x maksimum değeri küçülür.Döngüde ki bu olay sona erdiğinde min vertex fonksiyonu en yakın mesafeyi geri döndürür.Ve döndürülen değer ziyaret edildi olarak değiştirilir.Players daki neigbours dizisinin arrayi alır ve bir sonraki elemana uzaklığını hesaplatır ve pred geri döndürülür.

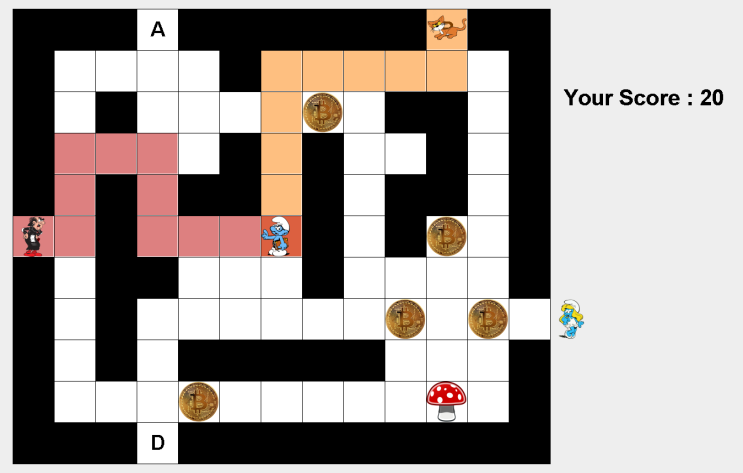
Daha sonra Players daki Players nesnesi oluşturulur ve okuma sağlanır.Sonra tüm labirent numaralandırılır ve komşular ve hareket edilebilen kareler birbirne çift yönlü bağlanır.Ve bu kareler t listesine eklenerek geri döndürülür.Daha sonra Djikstra sınfında bulunan printhpath() metoduna döndürülen argümanlarla tüm yollar denerek en kısa yol path isimli liste eklenir ve ters çevrilerek geri dönderilir.Artık elimizde yol vardır.

**Sonuç:**

Swing işleyişini daha iyi kavrayarak java grafik kütüphanesinde yararlandık.Farklı IDE deneme fırsatı bularak ufkumuzu genişlettik.Türlü hataları debug ederken bunun sayesinde araştırıp kendimizi geliştirdik.

**Çıktılar**





T**İME COMPLEXİTY**

**public class Dijikstra {**

public static int [] dijkstra (Players G, int s) {

1 final int [] dist = new int [G.size()];

1 final int [] pred = new int [G.size()];

1 final boolean [] visited = new boolean [G.size()];

N for (int i=0; i<dist.length; i++) {

1 dist[i] = Integer.MAX\_VALUE; }

1 dist[s] = 0;

N for (int i=0; i<dist.length; i++) {

1 final int next = minVertex (dist, visited);

N if(next<=dist.length){

N visited[next] = true;

1 final int [] n = G.neighbors (next);

N for (int j=0; j<n.length; j++) {

1 final int v = n[j];

1 final int d = dist[next] + G.getWeight(next,v);

N if (dist[v] > d) {

N dist[v] = d;

N pred[v] = next;}}}}

N return pred; // (ignore pred[s]==0!)}

1 private static int minVertex (int [] dist, boolean [] v) {

1 int x = Integer.MAX\_VALUE;

1 int y = 0;

N for (int i=0; i<dist.length; i++) {

if (!v[i] && dist[i]<x) {y=i; x=dist[i];}}

1 return y;}

public static ArrayList printPath (Players G, int [] pred, int s, int e) {

1 final java.util.ArrayList path = new java.util.ArrayList();

1 int i=0;

N for(i=0;e!=s;i++) {

N path.add (G.getKomsu(e));

N e = pred[e];}

1 path.add (G.getKomsu(s));

1 Collections.reverse(path);

1 return path;}}

public class Players extends Characters{

1 private int[][] kenarlar;

1 private Object[] komsular;

public Players() {}

public void rF() {

try {

1 File file = new File("C:\\Users\\yavuz\\Desktop\\NetBeansProjects\\Proje 4\\src\\proje\\pkg4\\harita.txt");

1 Scanner reader = new Scanner(file);

1 int i = 0;

1 String p = reader.nextLine();

1 String e1 = reader.nextLine();

1 String e2 = reader.nextLine();

11 for (int row = 0; row < 11; row++)

11\*13 for (int col = 0; col < 13; col++) {

11 \*13 maze[row][col] = reader.nextInt();}}

} catch (FileNotFoundException ex) {

1 Logger.getLogger(Proje4.class.getName()).log(Level.SEVERE, null, ex);}}

public int size() {

1 return komsular.length;}

public void setKomsu(int vertex, Object label) {

1 komsular[vertex] = label;}

public Object getKomsu(int vertex) {

1 return komsular[vertex];}

public void kenarEkle(int source, int target) {

1 kenarlar[source][target] = 1;}

public boolean kenarı(int source, int target) {

1 return kenarlar[source][target] > 0;}

public void removeEdge(int source, int targ et) {

1 kenarlar[source][target] = 0;}

public int getWeight(int source, int target) {

1 return kenarlar[source][target];}

public int[] neighbors(int vertex) {

1 int count = 0;

N for (int i = 0; i < kenarlar[vertex].length; i++){

N if (kenarlar[vertex][i] > 0) {

N count++;}}

1 final int[] answer = new int[count];

1 count = 0;

N for (int i = 0; i < kenarlar[vertex].length; i++) {

N if (kenarlar[vertex][i] > 0) {

N answer[count++] = i;}}

return answer;}

public void print() {

N for (int j = 0; j < kenarlar.length; j++) {

N System.out.print(komsular[j] + "- ");

N for (int i = 0; i < kenarlar[j].length; i++) {

N if (kenarlar[j][i] > 0) {

N System.out.print(komsular[i] + ":" + kenarlar[j][i] + "");}}

N System.out.println();}}

public static Players enKisaYol() {

1 Players asd = new Players();

1 asd.rF();

1 int sayac = 0;

1 final Players t = new Players(144);

143 for (int i = 0; i < 143; i++) {

t.setKomsu(i, i);}

1 t.setKomsu(143, 143);

11 for (int i = 0; i < 11; i++) {

N\*N for (int j = 0; j < 13; j++) {

N\*N if ((j == 12) || (i == 10)) {

} else {

N if (maze[i][j] == 1) {

N if (maze[i][j + 1] == 1) {

N if(sayac==103){

N t.kenarEkle(103,143);

N t.kenarEkle(143,103);

}else{

N t.kenarEkle(sayac, sayac + 1);

N t.kenarEkle(sayac + 1, sayac);}}

if (maze[i + 1][j] == 1) {

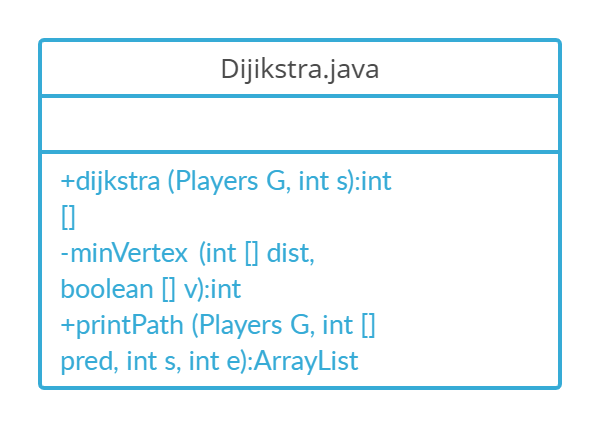
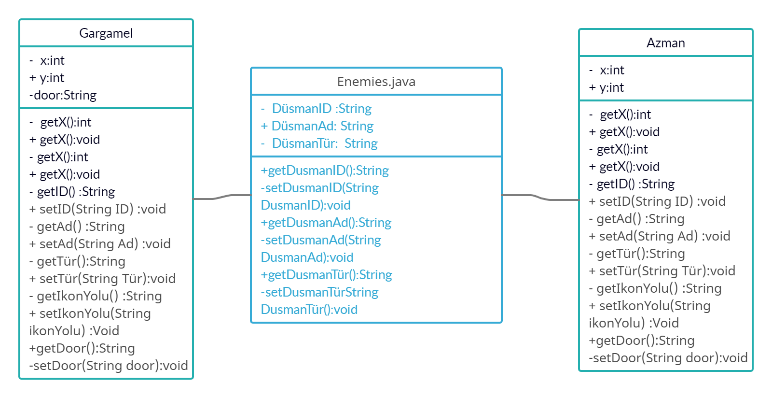
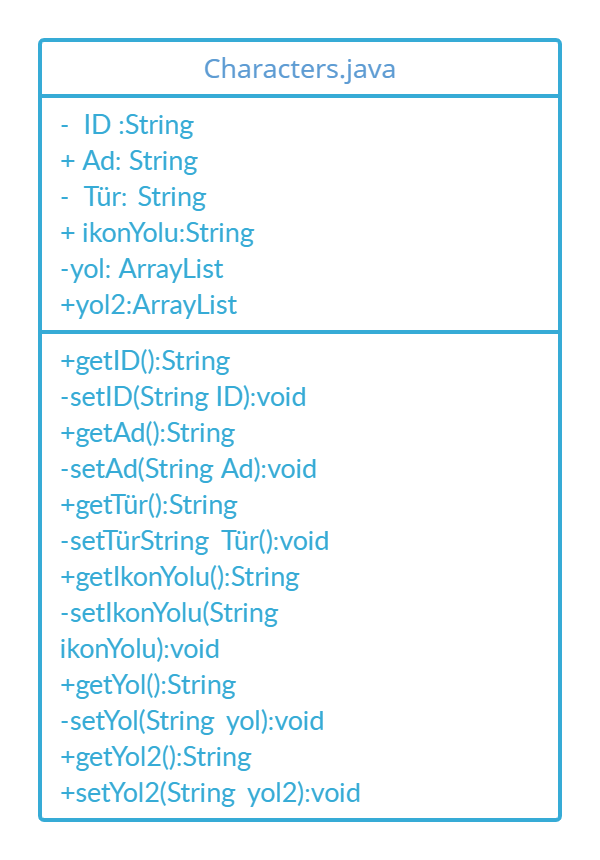
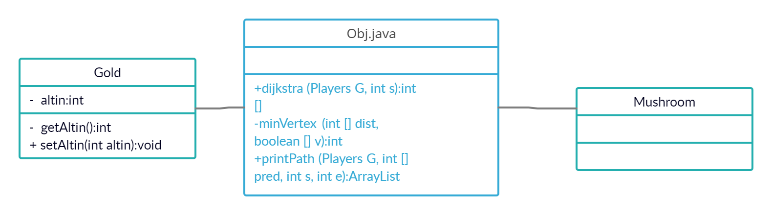
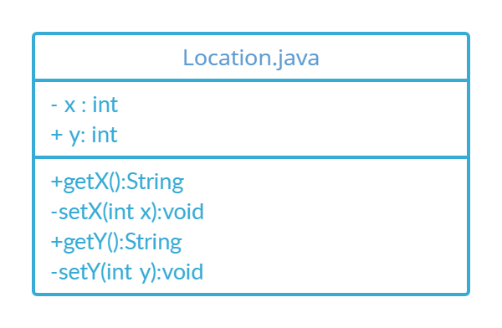
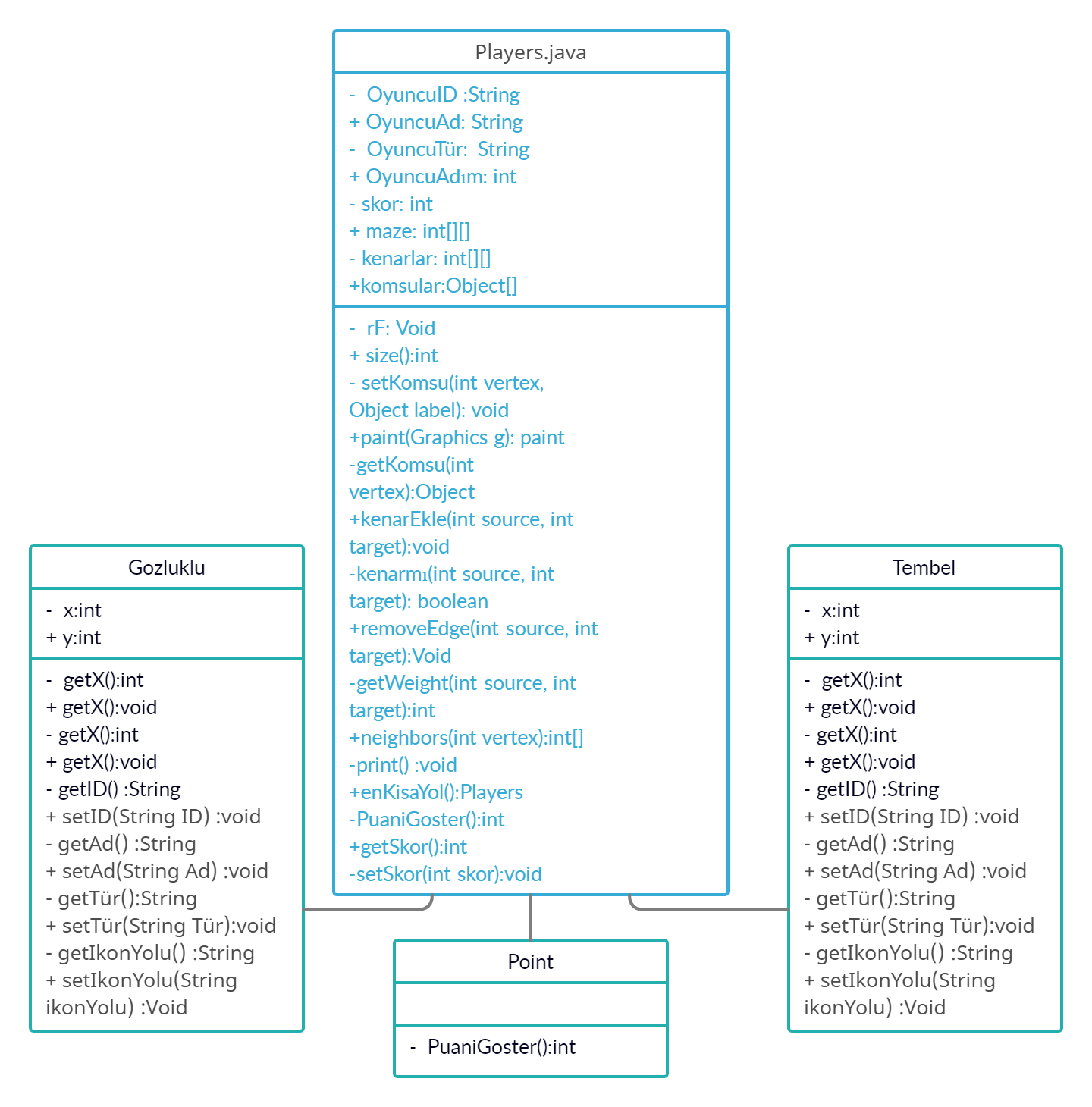
N if(sayac!=103){

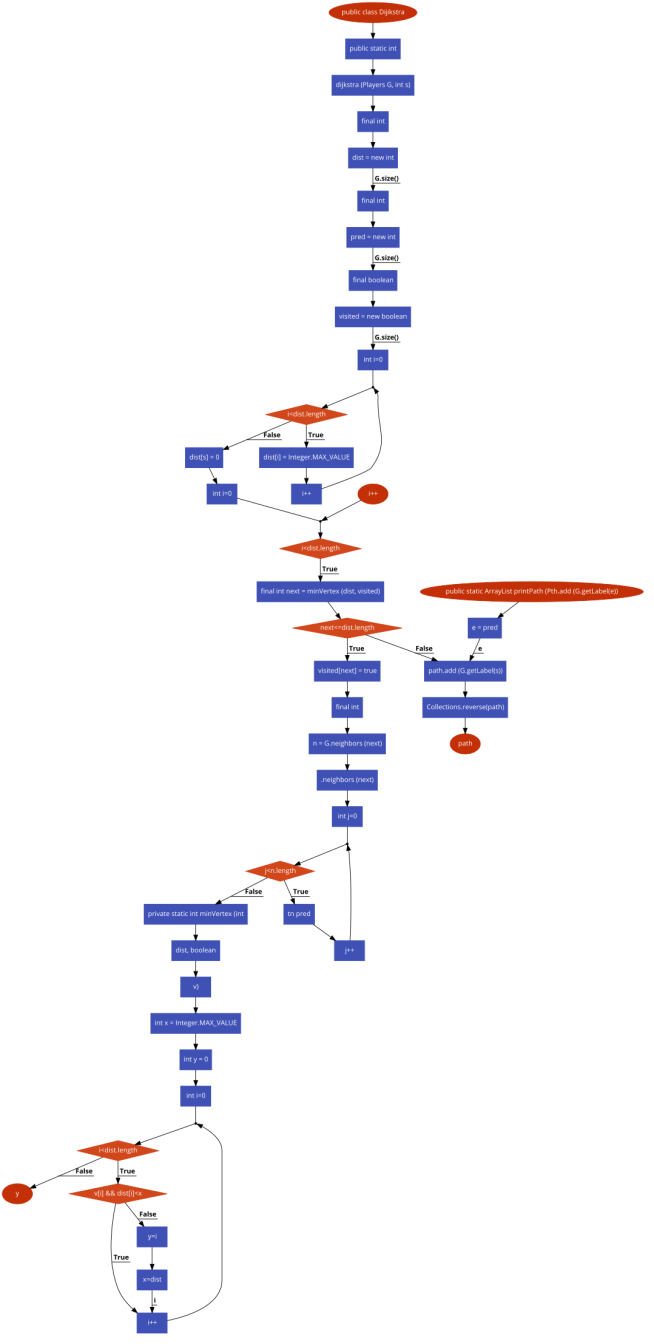
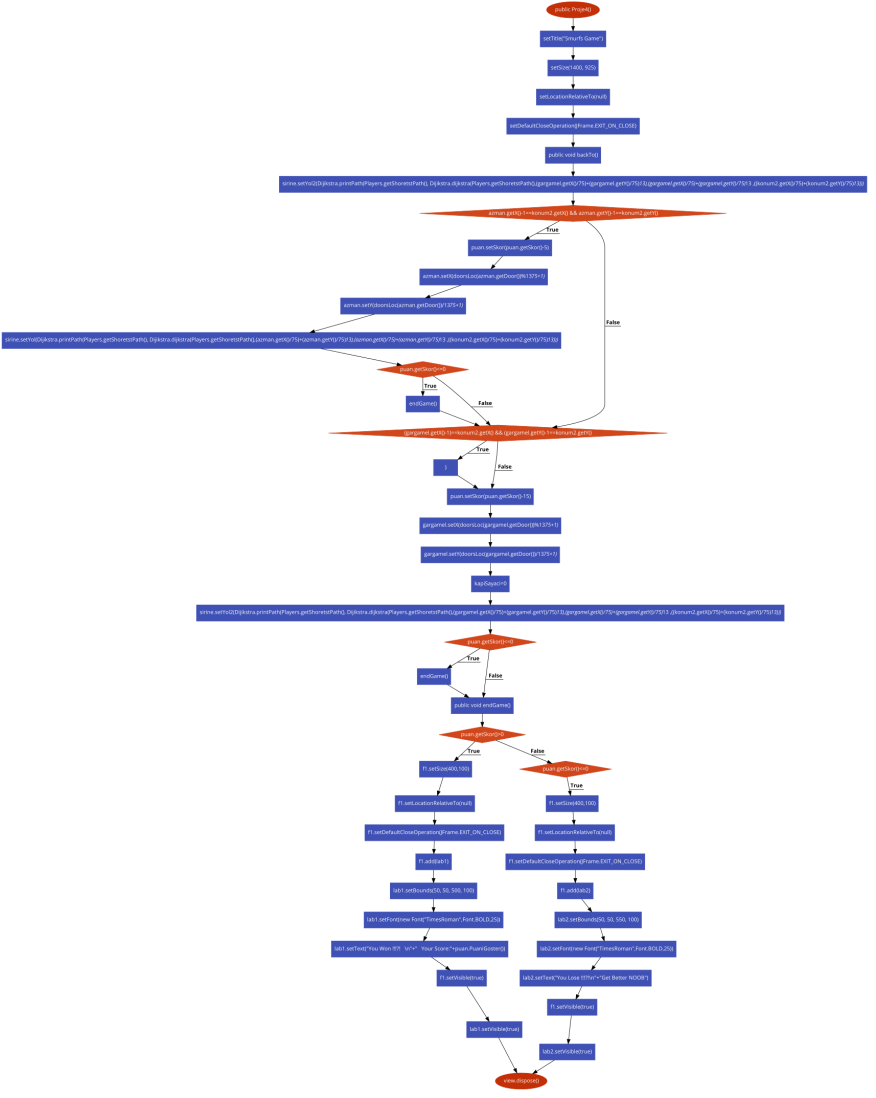
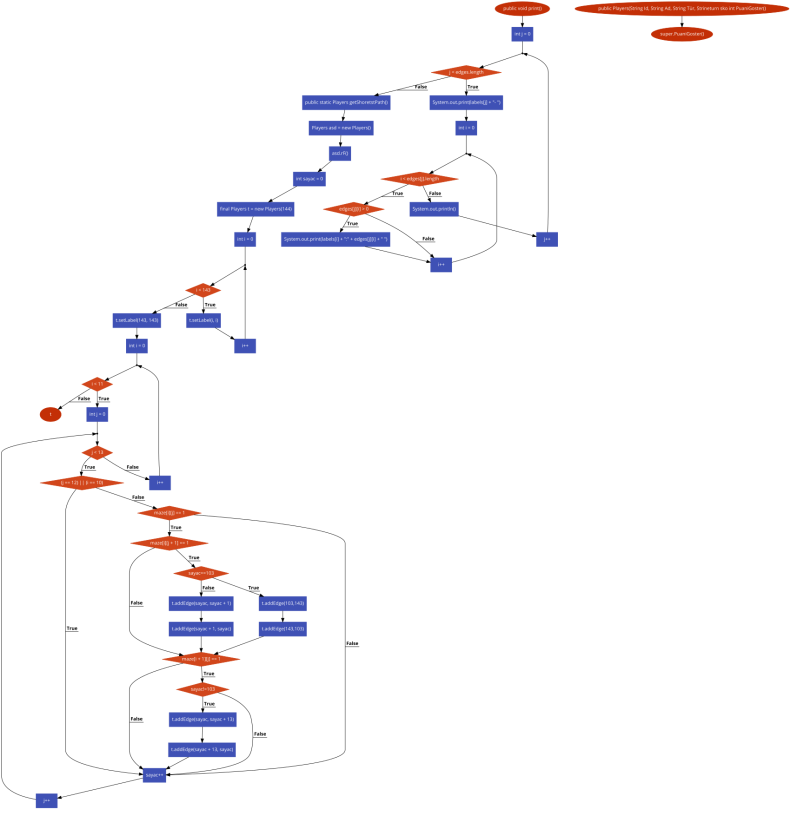
N t.kenarEkle(sayac, sayac + 13);}}}

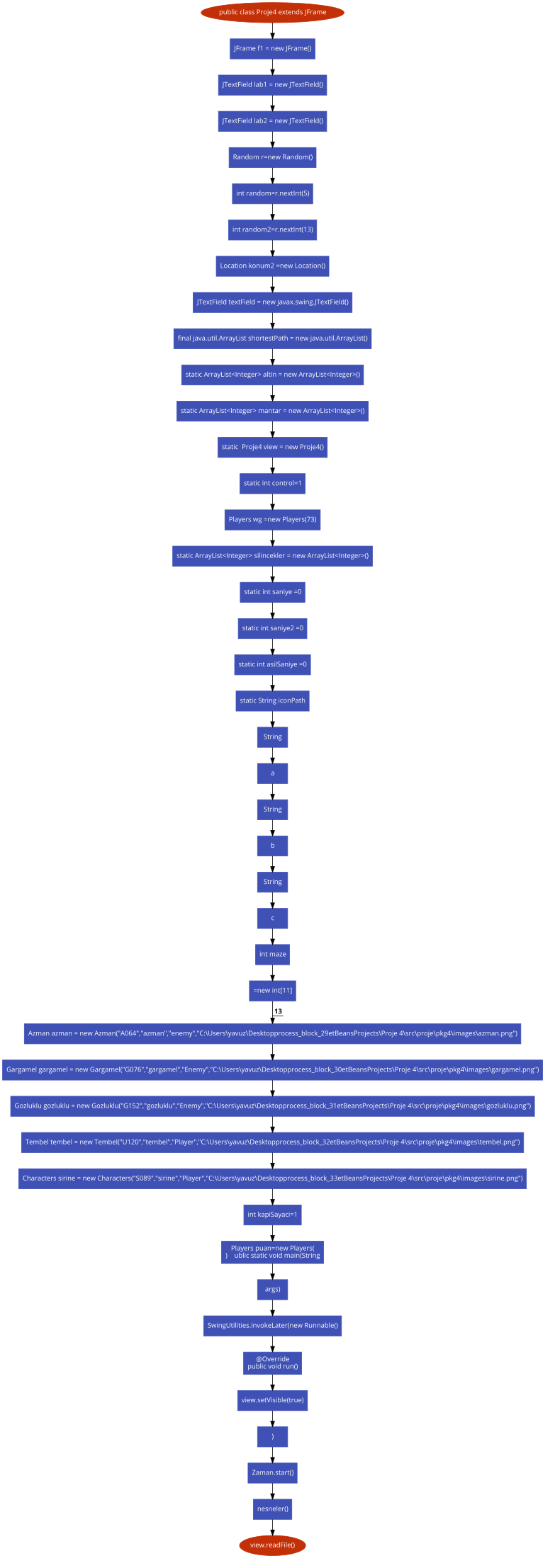
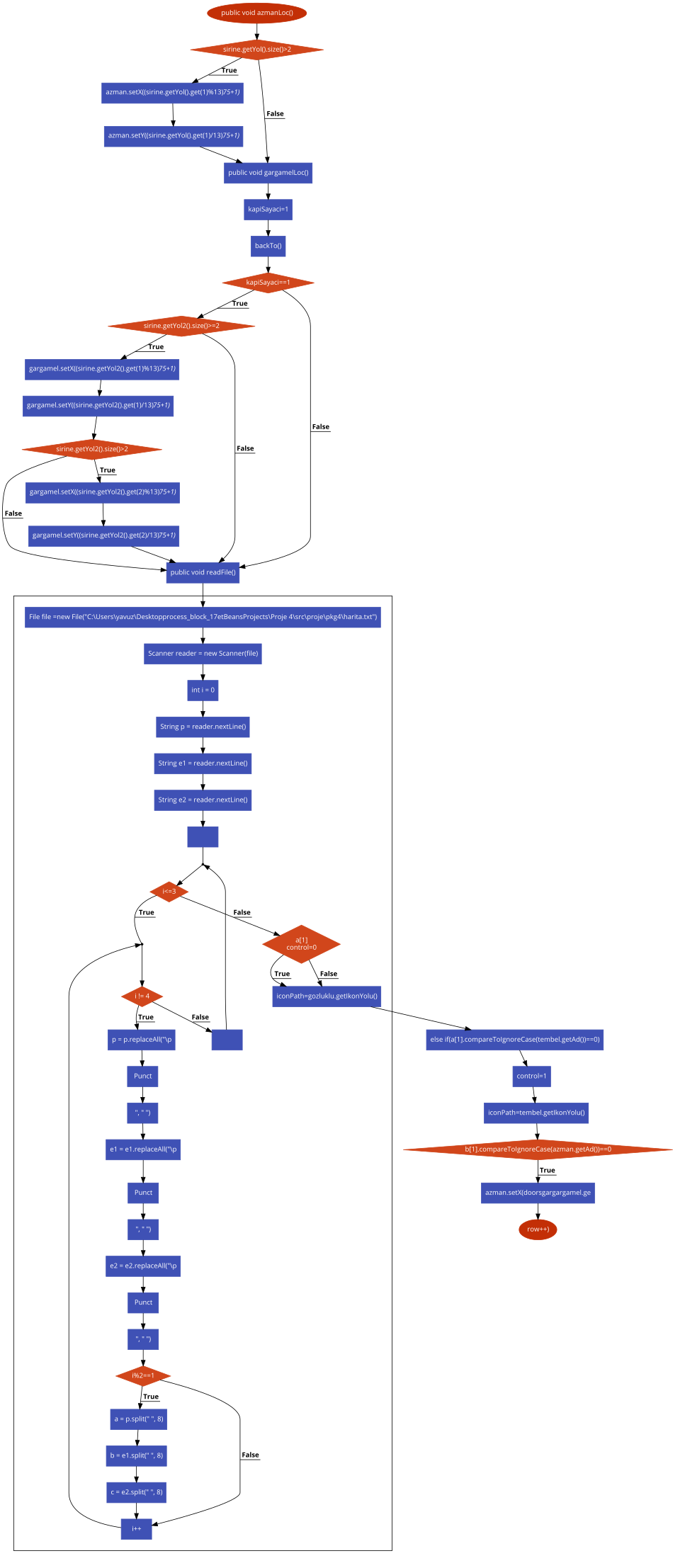
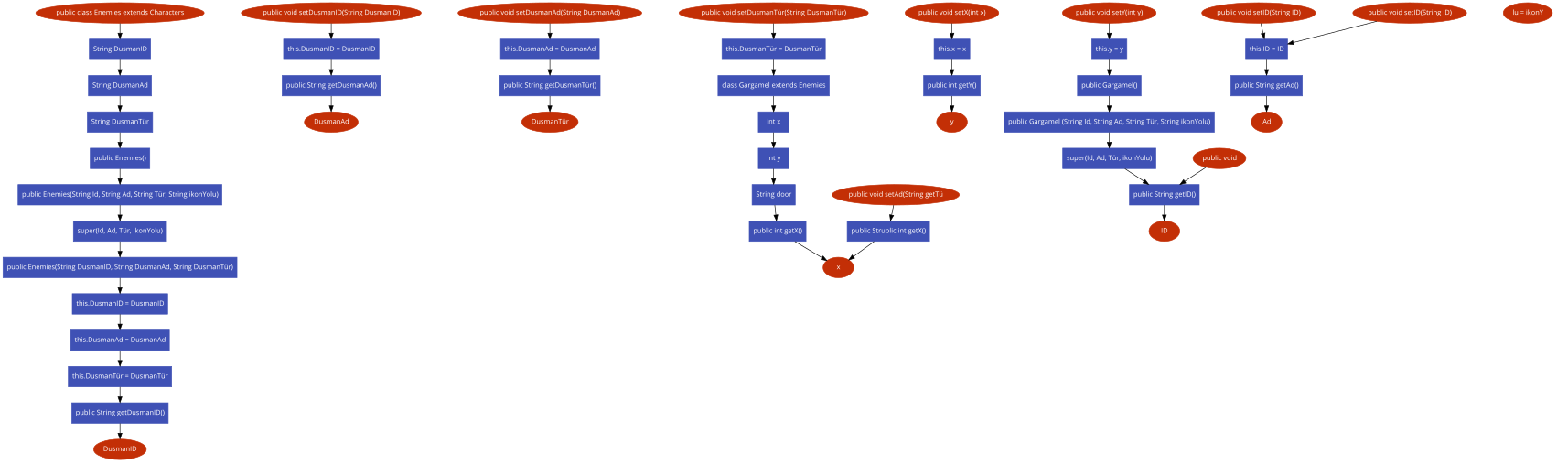
sayac++;}}

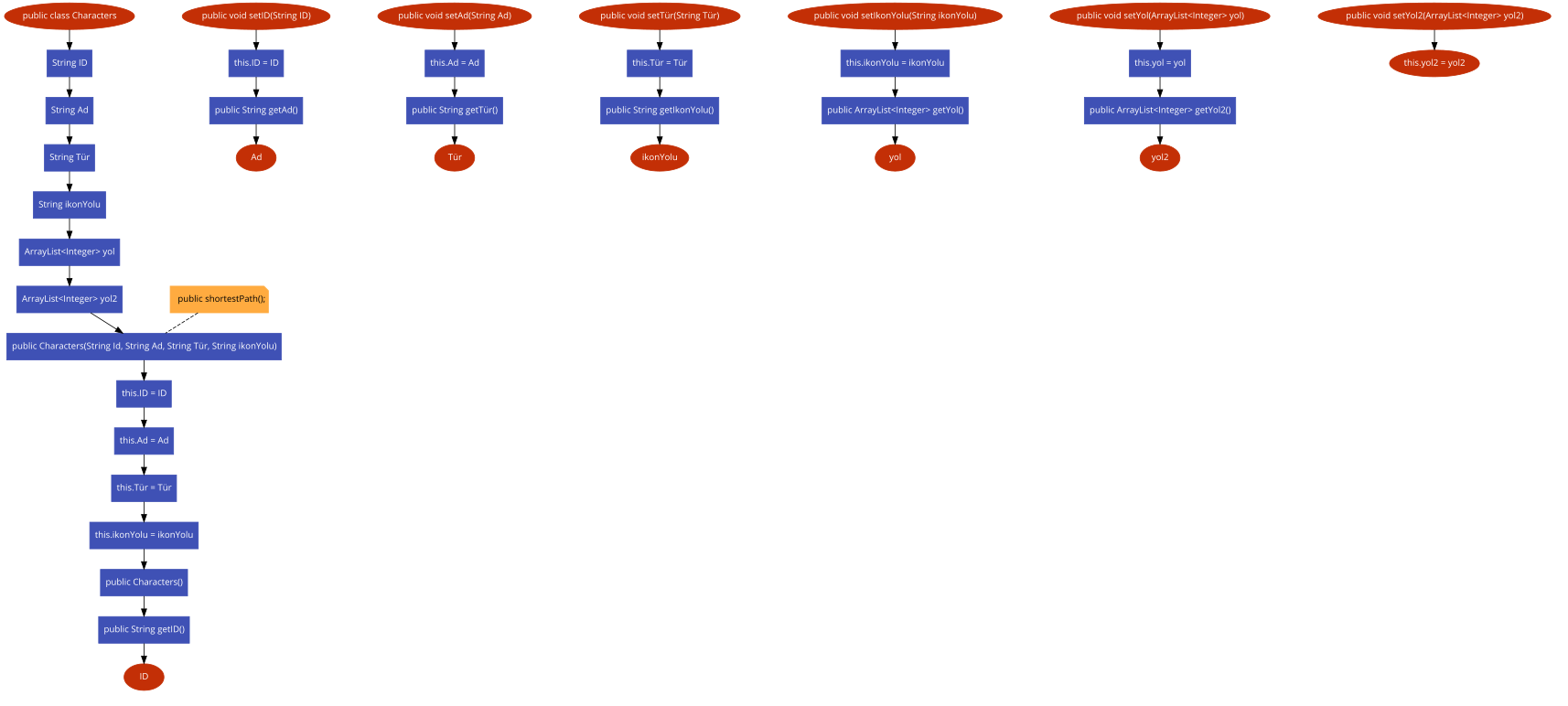
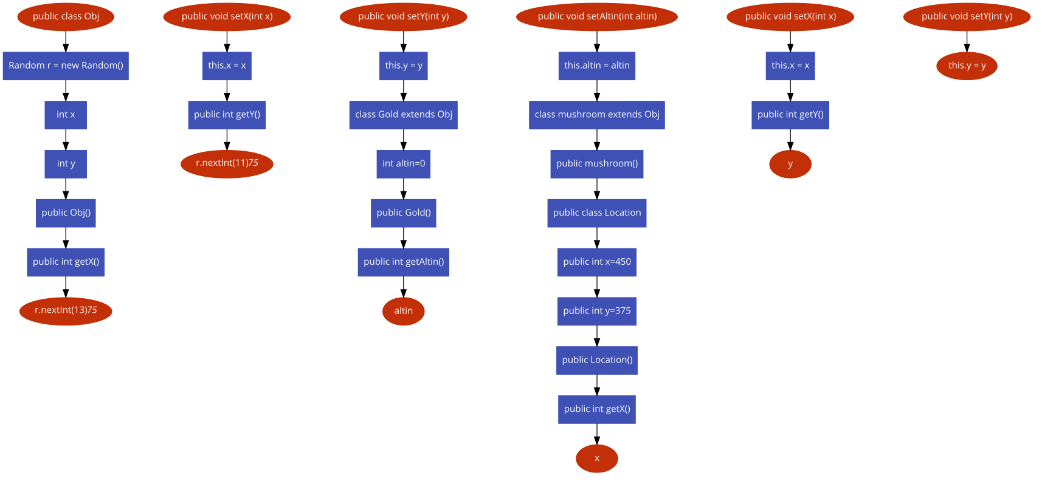
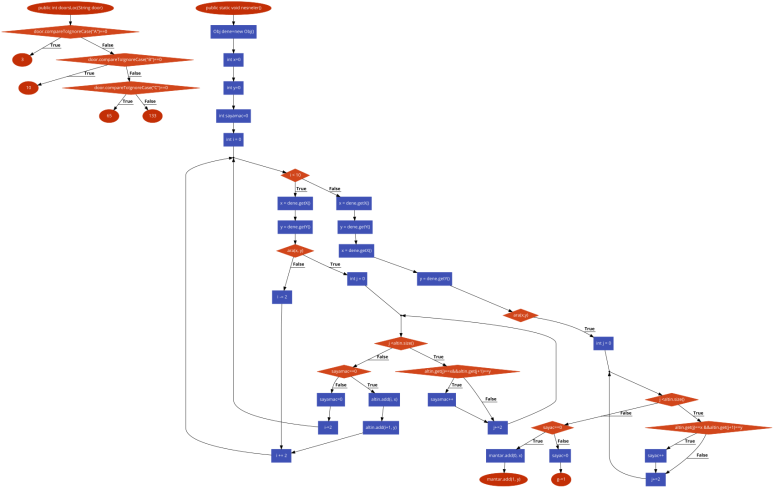
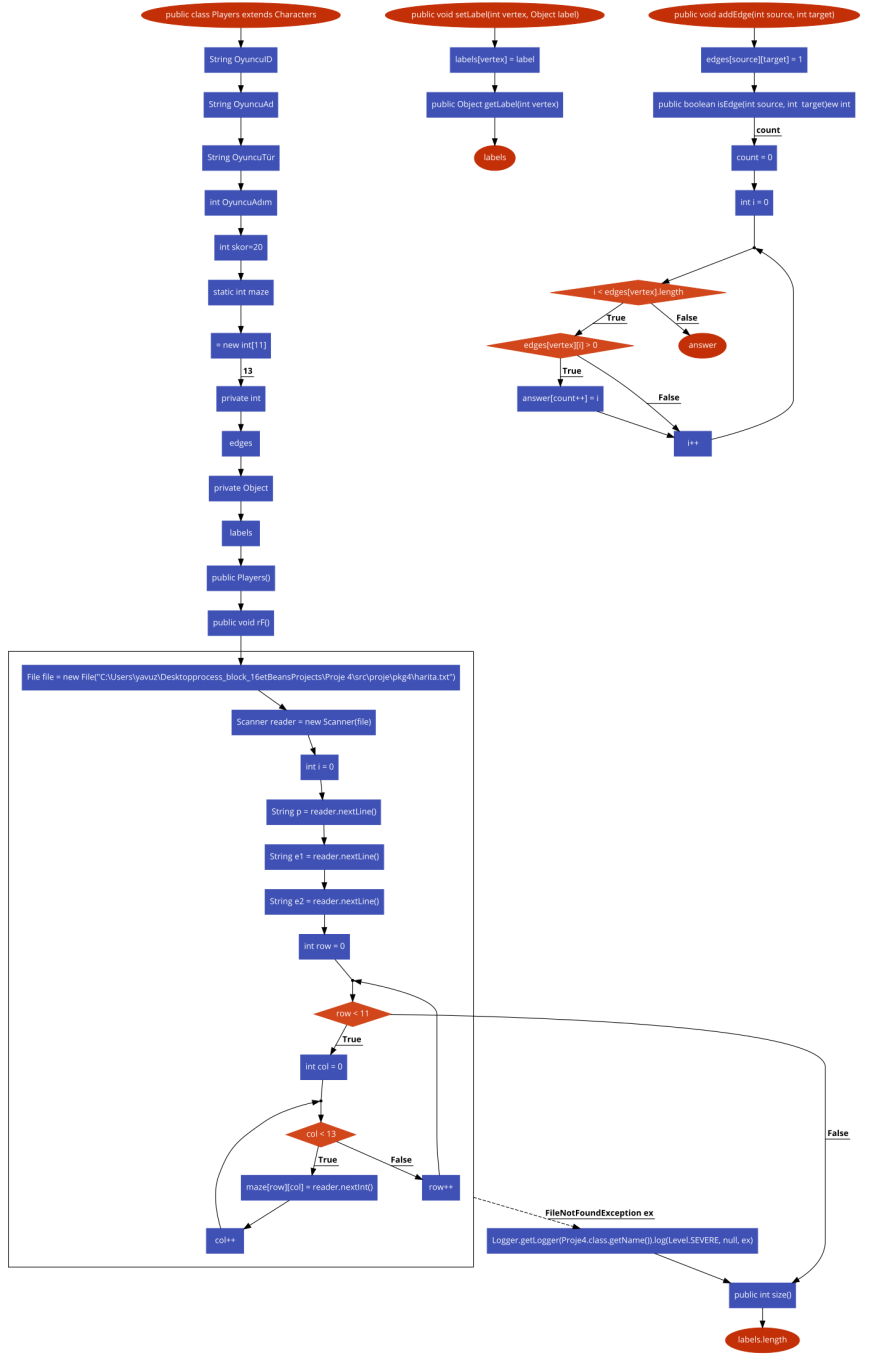
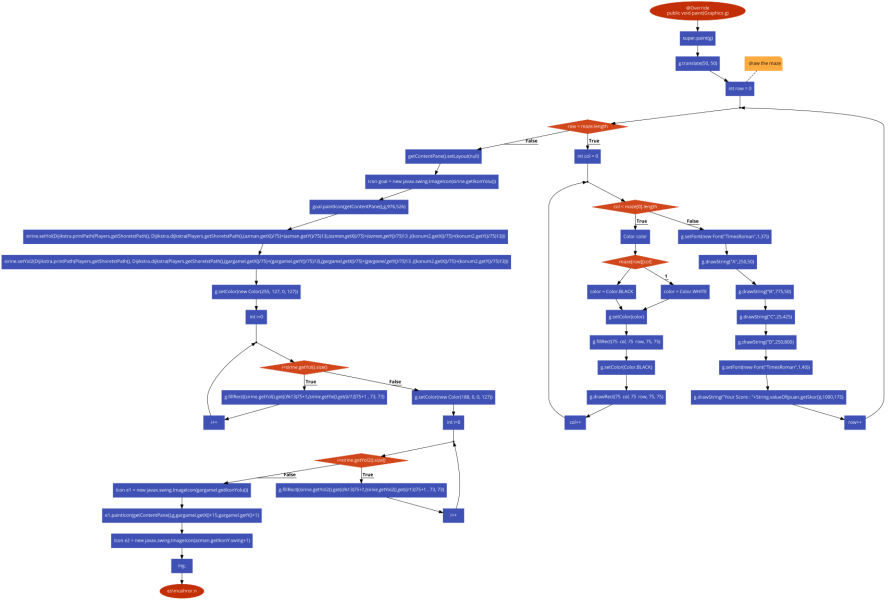
N return t;}

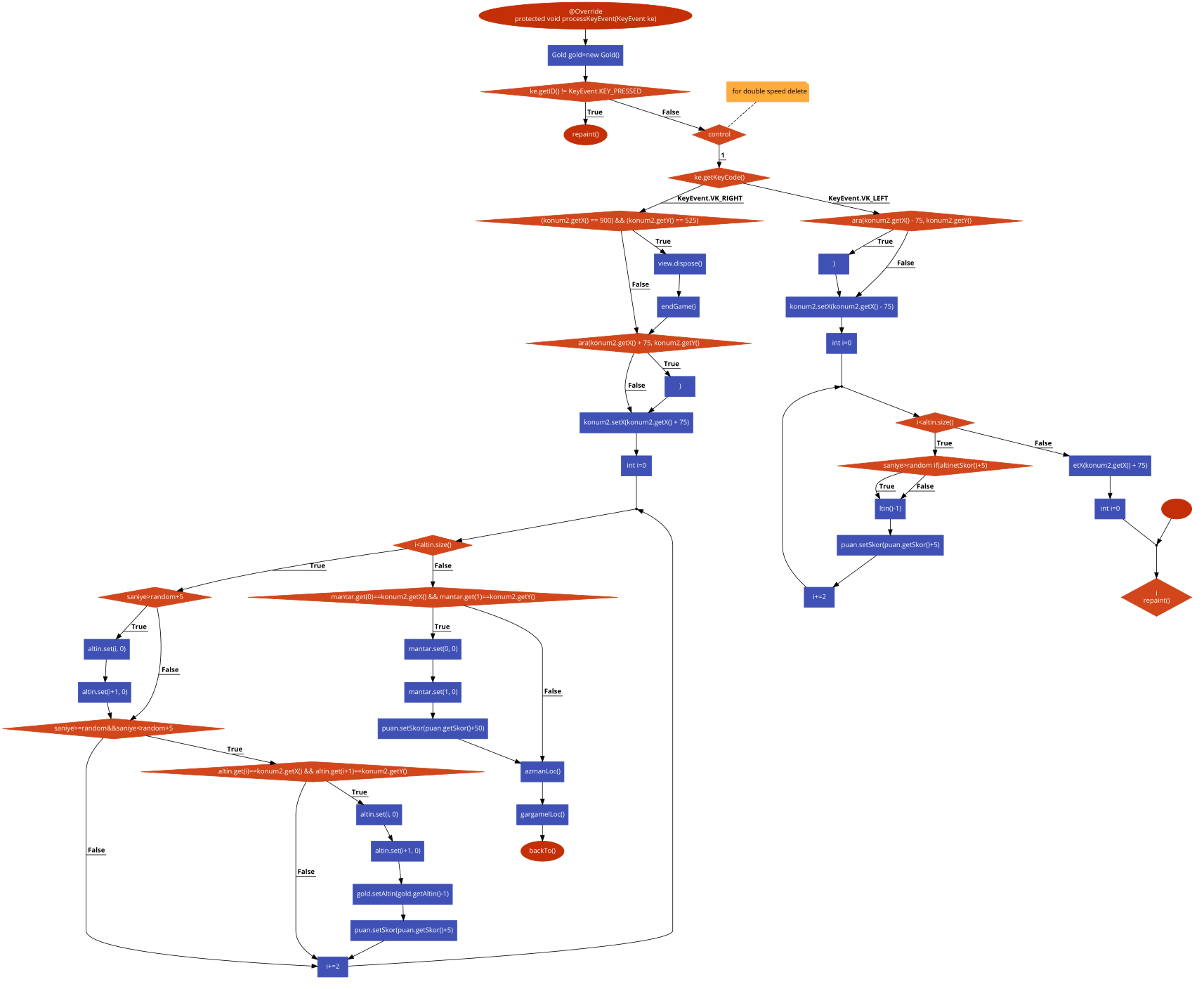
**Djikstranın zaman Karmaşıklığı = O(N^2)**

**UML Diyagramları**









**Kaynakça**

<https://www.javatpoint.com/java-jbutton>

<https://www.javatpoint.com/java-jlabel>

https://www.javatpoint.com/java-jframe

<https://www.javatpoint.com/java-actionlistener>

https://www.w3schools.com/java/java\_abstract.asp

https://www.tutorialspoint.com/swing/swing\_imagei

<https://www.youtube.com/watch?v=eVV6p0axFYc>

<https://www.youtube.com/watch?v=jT3c45XkPTg>

<https://www.youtube.com/watch?v=SN-FYlvtwOE>

<https://www.youtube.com/watch?v=ZtBMVxyD8A0>

<https://ensarkarabudak.com/dijkstra-en-kisa-yol-algoritmasi/>

<https://www.youtube.com/watch?v=GazC3A4OQTE>

<https://www.youtube.com/watch?v=XB4MIexjvY0>

https://www.youtube.com/watch?v=f8nmipbEhBU