**Şirinler Oyunu**

*Kaan Kalaycı, Alperen Ünlü*

Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi

[ahmet.alper96@gmail.com.tr](mailto:ahmet.alper96@gmail.com.tr), [kaan.klyc1@hotmail.com](mailto:kaan.klyc1@hotmail.com)

**Özet:**

Programlama Laboratuvarı II projesi olarak bizden nesneye yönelik programlama ve veri yapıları algoritmalarını kullanarak Şirinler oyunu tasarlamamız istendi. Ayrıca bu oyunu grafiksel olarak sunmamız için bir arayüz yaratmamız istendi. Oyunda iki farklı oyuncu ve iki farklı düşman vardır. Bu oyuncu ve düşman sınıfları bir üst sınıf olan karakter sınıfından kalıtım alırlar. Aynı şekilde oyuncu karakter olan Tembel Şirin ve Gözlüklü Şirin oyuncu sınıfından; düşman karakter olan Gargamel ve Azman düşman sınıfından kalıtım alırlar. Tembel Şirin bir adım, Gözlüklü Şirin iki adım ilerler. Benzer şekilde Azman bir adım, Gargamel iki adım ilerler. Kullanıcı oyuncuyu klavye ile seçecek ancak düşmanlar bir .txt dosyasında önceden belirlenecek. Bu .txt dosyasında ayrıca düşmanların başlangıç noktaları olan kapılar ve oyunun işleyeceği harita bulunmaktadır. Bu .txt dosyasının içeriği oyun başlatılmadan önce değiştirilebilir. Oyunun amacı klavyeyi kullanarak hareket ederek Şirine’ye ulaşmaktır. Şirine’ye ulaşılırsa oyun kazanılır. Bu sırada düşmanlar oyuncuyu Dijkstra algoritmasını kullanan en kısa yollarla takip edecek ve oyuncu ile çarpıştıklarında oyuncunun puanını düşüreceklerdir. Gargamel 15 puan, Azman 5 puan düşürür. Eğer puan 0 veya 0’ın altına düşerse oyun kaybedilir.

**Giriş:**

Bu projeyi Java programlama dilini kullanarak yazdık. Geliştirme ortamı olarak Apache Netbeans’i kullandık. Arayüz tasarımı için Swing tasarım kütüphanesini kullandık.

**Yöntem:**

İlk önce karakterlerimizi oluşturmak için karakter sınıfını oluşturduk. Bu sınıfa ID, ad, tür ve lokasyon bilgisi özelliklerini verdik. Ayrıca bu bilgileri değiştirebilmek ve alabilmek için get-set ve constructor metodlarını oluşturduk. Ayrıca bir EnKısaYol metodu oluşturduk ancak kullanmamıza ihtiyacımız olmadı. Sonra karakter sınıfında kalıtım alacak oyuncu ve düşman sınıflarını oluşturduk. Oyuncu sınıfına skor özelliğini verdik. Düşman sınıfı sadece üst sınıfın özelliklerini taşıyor. Sonra aynı özelliklere sahip olsalar da karışmamaları için karakterlerin hepsine özel sınıflar oluşturduk. gözlüklüŞirin ve tembelŞirin oyuncu sınıfının alt sınıfı; gargamel ve azman düşman sınıfının alt sınıfıdır. Sonra bu karakterlerin koordinatlarını tutmak için lokasyon sınıfını ve get-set metodlarını oluşturduk. Düşman karakterlerin en kısa yol metodlarını oluşturmak için Dijkstra algoritmasında kullanılan edge(kenar)’leri kullanmak için Kenar sınıfını oluşturduk. Bu sınıfa x ve y koordinatı, kenarın uzunluğu, kenarın komşuları ve yol özelliklerini verdik. Sonra oyunun işleyeceği arayüzü oluşturmak için araYüz sınıfını yarattık. Burada karakterlerin lokasyon bilgilerinden yararlanarak çizim işlemlerini yaptık. Klavyeye göre yapılan hareket işlemlerini de burada gerçekleştirdik. Son olarak Main sınıfında oyunu çalıştırdık. Bunun için ilk olarak haritayı tutacak bir dizi ve düşmanları, düşmanların yollarını, düşmanların başlayacağı kapıları ve haritada hareket edilebilen kenarları tutan listeler oluşturduk. Sonra klavye girişiyle kullanıcıya şirinini seçtirdik. Sonra harita.txt dosyasını okuyarak oyunun haritasını, düşmanları, başladıkları kapıları oluşturduk. Aldığımız haritanın bilgilerine göre kenarları belirleyip listeye ekledik. Bu kenarlardan oyuncunun başlangıç noktasını hesap için kullanılan kenar olarak belirdik. Hesapladıktan sonra oluşan en kısa yolları düşmanlara ekledik ve arayüzü çalıştırdık. Her hareket ettiğimizde bu kenarlar ile yollar resetleniyor ve yeni konumlara göre tekrardan hesap yapılıyor. Bu şekilde oyuncunun skoru 0 veya altına inene kadar yada oyuncu Şirine’ye ulaşana kadar oyun devam ediyor.

**Deneysel Sonuçlar:**

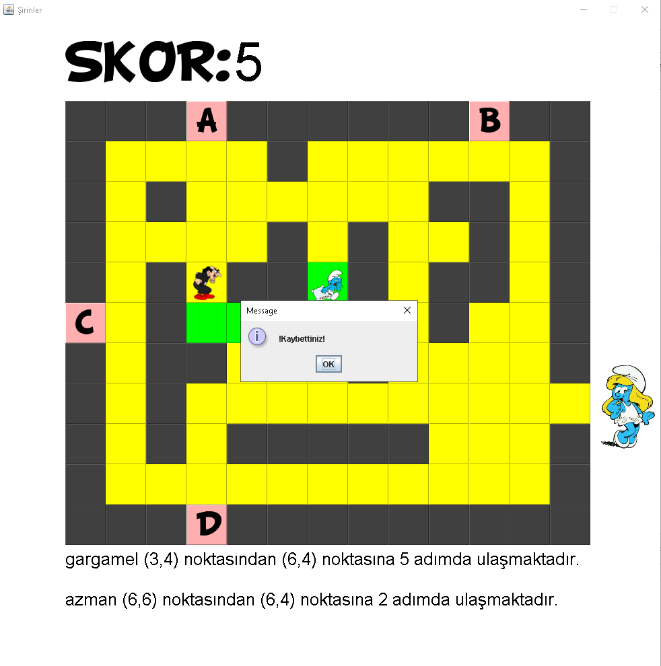
Gargamel’in A kapısında Azman’ın B kapısında başladığı karakterin Tembel Şirin seçildiği başlangıç durumu:



Gargamel’in C kapısında Azman’ın D kapısında başladığı karakterin Gözlüklü Şirin seçildiği başlangıç durumu:



Skorun 0’a düştüğünde oyunu kaybetme durumu:



Şirine’ye ulaşıldığında oyunu kazanma durumu:



**Karmaşıklık:**

Bir algoritmanın büyük-O karmaşıklık analizini yaparken en büyük artışı yöneten terim alınır, diğerleri göz ardı edilir çünkü input size sonsuza giderken bu büyümeyi yönetecek olan en büyük değişkenli terimdir.



Karmaşıklık hesabı için bu metodu ele alırız. Yukarıdaki metod, başlangıç noktası olan kenarın komşularının bir yola eklenip eklenmediğini kontrol eder. Eklenmedilerse kenarın komşusunun yoluna başlangıç kenarının yolunu ekler. For döngüsü başlangıç kenarının yolundaki kenar sayısı kadar çalışır. Sonra metod kenarın komşu için çalışır. Eğer komşu bir yola eklenmişse komşunun yolunun uzunluğunu kontrol eder. Uzunluk önceki kenarın uzunluğunun bir fazlasından fazlaysa yeni bir yol oluşturur, bu yola önceki kenarı ekler ve yeni yol komşunun yolu olur. Komşular kalmayana kadar bu kod recursive olarak çalışmaya devam eder. Kenarların hepsini ziyaret etmek için O(n) zaman gereklidir. Aynı şekilde komşu kenarların uzunluğunu güncellemek için O(n) zaman gereklidir. Buradan zaman karmaşıklığı O(n\*n)=O(n^2) bulunur. Kenarların uzunluğunu ve eklenip eklenilmediğini kontrol etmek ve önceki kenarın uzunluğunu tutmak için n sayısına kenar sayısı dersek O(n) bellek gereklidir. Buradan bellek karmaşıklığı O(n)+O(n)+O(n)=O(n) bulunur.

**Yalancı Kod:**

1-Başla.

2-Arayüz penceresini tanımla.

3-Oyuncu şirini tanımla.

4-Düşmanı tanımla.

5-Düşmanlar, kapılar, yollar, kenarlar listelerini tanımla.

6-Klavyeden şirin seç.

7-harita.txt dosyasını oku.

8-Dosyanın içeriğine göre düşman karakterlerin oluştur.

9- Düşman karakterlerin başlangıç noktalarını oluştur.

10-Arayüz için oyun haritasını oluştur.

11-Haritaya göre kenarları belirle.

12-Kenarların komşularını belirle.

13-Oyuncunun başlangıç noktasına göre kenar belirle.

14-Bu kenara göre kısa yol hesabını yap.

15-Düşmanların başlangıç noktalarına göre kenarlar belirle.

16-Düşmanların kenarlarının en kısa yollarını kısa yol listesini ekle.

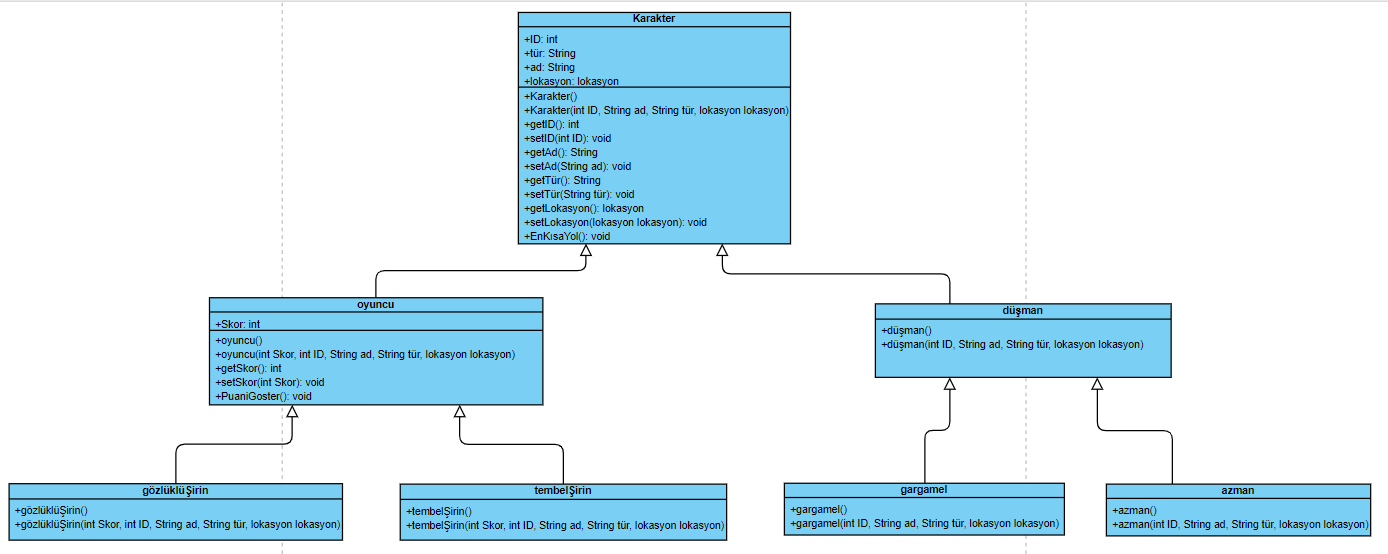
17-Arayüzü çalıştır.

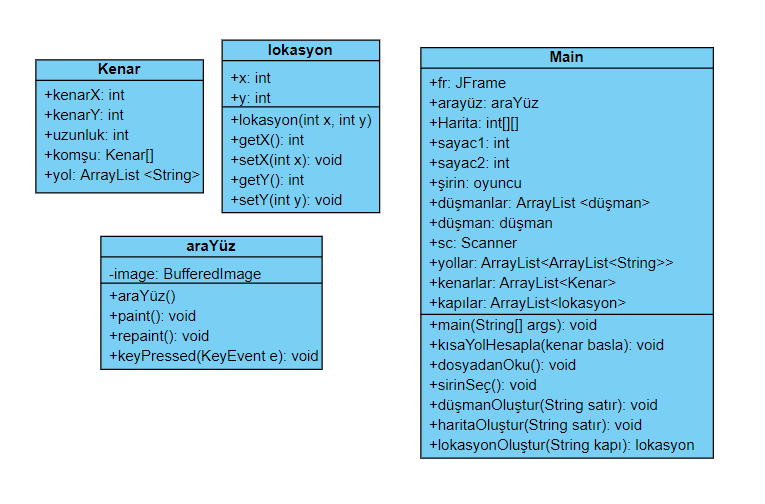
18-Her klavye hareketinde kenarları ve yollar tekrardan hesapla.

19-Skor 0 veya altına düşerse oyunu kaybet.

20-Şirineye ulaşılırsa oyunu kazan.

21-Bitir.

**UML Sınıf Diyagramı: **

****

**Sonuç:**

Sonuç olarak bizden istenen obje sınıfının işlemleri dışında oyunu başarıyla yarattık. Kodda işlem sırasının ne kadar önemli olduğunu anladık. Yol bulma algoritmaları hakkında daha çok bilgi edindik. Java’da dosya işlemleri konusunda deneyim kazandık. Zamanlayıcı kullanan işlemler konusunda kendimizi daha çok geliştirmemiz gerektiğini fark ettik.

**Kaynakça:**

<https://stackoverflow.com/questions/1006611/java-swing-timer>

<https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>

<https://www.geeksforgeeks.org/dijkstras-algorithm-for-adjacency-list-representation-greedy-algo-8/>

<https://www.softwaretestinghelp.com/dijkstras-algorithm-in-java/>

<https://www.bogotobogo.com/Java/tutorials/javagraphics3.php>

<https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html>

<https://youtube.com/playlist?list=PLIHume2cwmHcGWl_h31xBh1zSlqmxfFRQ>

<https://zetcode.com/javagames/animation/>

<https://www.javatpoint.com/java-string-split>

<https://www.dijitalders.com/icerik/68/1947/java_da_goruntu_haraketleri.html>