**PROGRAMLAMA LABORATUVARI 2**

**1. PROJE**

Muhammet Ali BAKINÇ

Abdulhalik SARIÇİÇEK

Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi

**Özet**

Bu doküman Programlama Laboratuvarı 2 dersi 1. Projesi için çözümümüzü açıklamaya yönelik oluşturulmuştur. Dokümanda projenin tanımı, çözüme yönelik yapılan araştırmalar, kullanılan yöntemler, proje hazırlanırken kullanılan geliştirme ortamı ve kod bilgisi gibi programın oluşumunu açıklayan başlıklara yer verilmiştir. Doküman sonunda projemizi hazırlarken kullandığımız kaynaklar ve proje derlenirken dikkat edilmesi gereken hususlar bulunmaktadır.

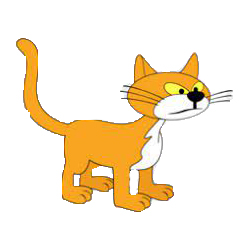
1. **Proje Tanımı**

Projede bizden istenen, C++ veya Java dillerinden biri kullanılarak bir labirent oyunu tasarlamamız beklenmektedir.

Tasarlayacağımız oyunda, iki farklı şirin karakterinden birini seçerek oyuna başlamamız ve düşman karakterler olan Gargamel ve Azman’dan kaçarak Şirine’ye ulaşmaya çalışacağız. Önceden belirlenen haritaya göre labirent oluşturulacak, düşmanlar haritada belirtilen kapılardan oyuna girecek ve oyuncuya en kısa yoldan ulaşmaya çalışacaktır. Oyuncu sağ, sol, yukarı ve aşağı gidebilecektir. Labirent içinde rastgele oluşacak mantar ve altınlar oyuncuya ek puan kazandırabilecektir. Oyuncunun puanı bitmesi halinde oyunu kaybedecektir. Bu oyunu nesneye yönelik programlama yöntemini ve en kısa yol algoritmalarını kullanarak yapmamız beklenmektedir.

Oyun işleyişi aşağıdaki gibi olmalıdır:

* Oyuncu ilk ekranda Tembel ve Gözlüklü karakterlerinden birini seçmelidir.
* Oluşan haritada düşman karakterlerin oyuncuya ulaşabileceği en kısa yol gösterilir.
* Oyuncu, karakterini yön tuşlarıyla hareket ettirir. Her hareketinin ardından düşman karakterler de oyuncuya doğru yaklaşır.
* Oyuncu düşmanlara yakalanması halinde belirli miktar puan kaybeder.
* Puanı bitmeden Şirine’ye ulaşmalıdır.
* Haritada oluşan mantar ve altınlar toplanarak puan toplanabilir.
* Oyuncu en iyi skorla oyunu tamamlamaya çalışır



(Şekil 1: Düşman karakterlerinin görünümü)

(Şekil 2: Oyuncu karakterlerinin görünümü)



(Şekil 3: Objelerin görünümü)



(Şekil 3: Şirine’nin görünümü)

Projede belirli sınıfların ve bu sınıflarda belirli özelliklerin kullanılması zorunlu kılınmıştır. Nesneye yönelik programlama yönteminde geçerli olan *Encapsulation*, *Inheritance*, *Polymorphism*, *Abstraction* yapılarının gerekli olanların kullanılması da beklenmektedir. Projede oluşturmamız gereken sınıflar ve özellikleri kabaca bölüm 1.1. de anlatılmıştır.

**1.1. Oluşturulması Gereken Sınıflar**

Aşağıda belirtilen tüm sınıflarda ortak olarak yapıcı (*constructor*) metotları, sınıflardaki tüm özellikler için *get*, *set* metotları tanımlanmalıdır.

**Karakter sınıfı**

Tüm oyuncu ve düşman karakterleri bu sınıftan türemiştir. Sınıfta *id*, *isim*, *tur* ve *lokasyon* özellikleri tutulmalıdır. Sınıf hiçbir sınıftan kalıtım almamaktadır.

**Oyuncu sınıfı**

Karakter sınıfından kalıtım alacaklardır. Oyuncu sınıfında *oyuncuID, oyuncuAdi* ve *oyuncuTur* ve *skor* özellikleri bulunmaktadır. Parametresiz bir *constructor* bulunmalıdır. Değişkenler için *getter* ve *setter* metotları oluşturulmalıdır.

**Düşman sınıfı**

Karakter sınıfından kalıtım alacaklardır. Düşman sınıfında *dusmanID, dusmanAdi* ve *dusmanTur* özellikleri bulunmaktadır. Düşman karakterin oyuncuya ulaşması için hareket etmesi gereken yolu bulabilmesi için *enKisaYol* metodu yazılmıştır. Metot yolun başlangıç ve bitiş düğümlerini parametre olarak alır. Parametresiz bir *constructor* bulunmalıdır. Değişkenler için *getter* ve *setter* metotları oluşturulmalıdır.

**Gözlüklü sınıfı**

Oyuncu sınıfından türemiştir. Bu sınıfta sabit olarak *adim* özelliği bulunmaktadır. Gözlüklü karakteri 2 adım hareket etmektedir. Parametresiz *constructor* ile Oyuncu sınıfındaki *oyuncuAdi* değerine “Gözlüklü” ve *skor* değerine 20 atanmıştır.

**Tembel sınıfı**

Oyuncu sınıfından türemiştir. Bu sınıfta sabit olarak *adim* özelliği bulunmaktadır. Tembel karakteri 1 adım hareket etmektedir. Parametresiz *constructor* ile Oyuncu sınıfındaki *oyuncuAdi* değerine “Tembel” ve *skor* değerine 20 atanmıştır.

**Azman sınıfı**

Düşman sınıfından türemiştir. Bu sınıfta sabit olarak *adim* özelliği bulunmaktadır. Azman karakteri 1 adım hareket etmektedir. Parametresiz *constructor* ile Düşman sınıfındaki *dusmanAdi* değerine “Azman” atanmıştır.

**Gargamel sınıfı**

Düşman sınıfından türemiştir. Bu sınıfta sabit olarak *adim* özelliği bulunmaktadır. Gargamel karakteri 2 adım hareket etmektedir. Parametresiz *constructor* ile Düşman sınıfındaki *dusmanAdi* değerine “Gargamel” atanmıştır.

**Obje sınıfı**

Oyundaki toplanabilir objeleri temsil eden ana sınıftır. Bu sınıfta *konum* özelliği bulunmaktadır. Parametresiz bir *constructor* oluşturulmuştur.

**Altın sınıfı**

Obje sınıfından türemiştir. Oyundaki toplanabilir altınları temsil eden sınıftır. Bu sınıfta sabit olarak *sure, puan* ve *miktar* özellikleri bulunmaktadır. Altınları oluşturmak için *altinOlustur* metodu yazılmıştır. Parametresiz bir *constructor* oluşturulmuştur.

**Mantar sınıfı**

Obje sınıfından türemiştir. Oyundaki toplanabilir mantarları temsil eden sınıftır. Bu sınıfta sabit olarak *sure, puan* ve *miktar* özellikleri bulunmaktadır. Mantar oluşturmak için parametreli bir *constructor* oluşturulmuştur.

**Lokasyon sınıfı**

Oyundaki nesnelerin harita üzerindeki konumlarını tutmak için oluşturulan sınıftır. Bu sınıfta *satir* ve *sutun* özellikleri bulunmaktadır. Lokasyon oluşturmak için parametreli bir *constructor* oluşturulmuştur.

**Puan sınıfı**

Oyuncu sınıfından türemiştir. Oyuncunun puanını göstermek için oluşturulmuştur. Bu sınıfta Oyuncu sınıfındaki *puaniGoster* metodu override olmuştur.

**Node (Düğüm) sınıfı**

Oyuncunun ve düşmanın hareket edeceği alanları belirlemek için düğüm nesneleri oluşturulmuştur. Bu sınıfta *satir, sutun, deger, mesafe* ve boolean *gecildiMi* bilgileri tutulmaktadır. Ayrıca bir düğümün dört tarafındaki düğümler ile bir düğüme gelinen düğümün bilgisi de saklanmaktadır.

**2. Araştırmalar ve Yöntem**

Projeye proje tanımında anlatılan sınıfları oluşturmakla başladık. Projede belirtilen gerekli tüm sınıfları tanımlarken, kullanmamız zorunlu olan birçok nesneye yönelik programlama özelliğini hali hazırda kullanmış oluyoruz. Her alt sınıf bir üst sınıftan oluşacağı için *inheritance* özelliğini, sınıflardaki özelliklere dışarıdan *get, set* metotları ile ulaşım sağladığımızdan *encapsulation* özelliğini kullanmış oluyoruz. Grafik elemanlarının bir kısmını *oyuncuSecimEleman* listesinde tutarak, Object sınıfından türemiş sınıfları *polymorphism* özelliği ile saklamış oluyoruz. Böylece projede kullanılması tavsiye edilen özellikler sınıfları oluştururken kullanılmış oluyor.

Projede nasıl yapılması gerektiğini düşündüren en önemli nokta en kısa yol algoritması olan Dijkstra’yı çözmekti. Bunun için internette bulduğumuz makaleleri inceledik ve mantığını çözdüğümüz algoritmayı koda aktardık. Algoritmayı doğru kurabilmek projenin en temel problemiydi. Bunu doğru şekilde yapabildikten sonra geriye kalan kısımları daha rahat yapabildik. Main sınıfında oyunu yönetmeye dair temel fonksiyonları tanımladık. Oyunu yönettiği için kullanıcı ara yüzünün de Main sınıfında çizilmesinin uygun olduğuna karar verdik.

Takıldığımız bir diğer nokta ise Test sınıfında ara yüzü nasıl yapacağımıza karar vermekti. Öncelikle çizim işlemleri için JavaFX kütüphanesi kullanmaya karar verdik. Daha sonra çizimleri farklı fonksiyonlar içinde yazdık. Böylece Main’de yaptığımız tüm çizimler dinamik ve yönetimi kolay oldu. Bu yöntemdeki sorun hazır bir editör kullanamayacak olmamızdı, çizilen her element için koordinatları ve boyutu kendimiz bulmamız gerekiyordu ancak bizim için sorun olmadı.

**3. Geliştirme Ortamı**

Projemizi Windows sistemde, *Intellij Idea* üzerinde geliştirip derledik. *Java* dilini ve çizim yapabilmek için *JavaFX* kütüphanesini kullandık.

1. **Kod Bilgisi**

**4.1. UML Diyagramı ve Akış Şeması**

Son sayfadadır.

**4.2. Algoritma**

Bu kısımda projenin genel algoritmasına açıklık getireceğiz.

Oyun başladığında kullanıcıya oyunu başlatması için bir karakter seçim ekranı karşılıyor. Burada oyuncunun seçeceği karakter oyunun birtakım özelliklerinin kontrolünü de farklılaştırır. Oyunu yöneten ana sınıf Main sınıfı ve yönetime dair tüm fonksiyonlar (*karakterleriOlustur, oyuncuHareket, ...*) ve oyuncu nesneleri de bu sınıfta bulunuyor. Main sınıfının kodlarına bakıldığı zaman oyunun ana mantığı ve kuralları kolayca görülebilir.

Önce main metodu içerisinde grafik kütüphanemizin başlangıç metodu olan *launch* komutuyla ara yüz çizimlerini başlatmış olduk.

JavaFX kütüphanesinin çizimleri başlattığı *start* metodunu override ettik. Bu metotta öncelikle ekranı oluşturduk. Çizim nesnelerini bir *group* nesnesinde topladık. Öncelikle *haritaOku* metodu ile proje dosyaları içerisine önceden yerleştirilen *harita.txt* metin belgesinden harita bilgilerini okuduk. Satır satır gezdiğimiz belgede belli bir düzene göre girildiğini bildiğimiz bilgileri okuduk. İlk iki satırda düşman karakterlerinin hangi kapıdan gireceği bilgisi bulunmaktaydı. Bu bilgilere göre karakterlerin birer nesnesini oluşturduk ve sınıfın başında önceden tanımladığımız kapıların lokasyon bilgilerinden uygun olanını karakterin lokasyonuna atadık. Sonraki satırlarda ise 1 ve 0 değerleri ile belirtilmiş labirent bilgilerini aldık. Bunları da 11\*13’lük bir matrise atadık. Bu metodun bitiminde ise *dugumOlustur* metodunu çağırdık.

Bu metotta aldığımız harita bilgisine göre her bir kare alan için bir düğüm oluşturduk ve bunları birbirlerine bağladık. Bu işlemlerin bitmesinin ardından *start* fonksiyonundaki bir sonraki metot olan *oyuncuSecimEkrani* çalışmış oldu.

Bu fonksiyon başlangıç ekranını çizmek için oluşturuldu. İki oyuncu karakterinin resimlerini içeren dörtgenler ekrana çizildi ve bunlara tıklama olayı verildi. 1 ve 2 olarak belirlediğimiz karakterlerden biri kullanıcı tarafından seçildikten sonra bu bilgiyi de bir değişkende sakladık. Tıklanılan karakter *oyuncuAta* metoduna bu karakterin numarasını parametre olarak geçti. Bu metotta gelen numaraya göre karakterin bir nesnesi oluşturuldu ve tıpkı kapılarda olduğu gibi önceden belirlenen başlangıç noktası lokasyonu oyuncuya atandı. Ardından seçim ekranı için oluşturulan tüm nesneler ekrandan sildirildi. Ve *haritaCiz* metodu tetiklendi.

Burada 11\*13’lük harita için kare şekiller oluşturuldu ve harita matrisindeki değerlere göre 1 numaralı kareler beyaz, 0 numaralı kareler gri ve kapı noktaları pembe renklendirildi. Daha sonra düzenlenebilmesi için şekiller de bir matriste tutuldu. Sonunda ise *karakterleriOlustur* metodu çağrıldı.

Bu metotta düşman, oyuncu ve Şirine karakterleri ilgili konumlara çizdirildi. Yön tuşlarıyla hareket edilebilmesi için bir tetikleyici oluşturuldu. İlgili butona basıldığında algılanan yön *oyuncuHareket* metoduna parametre olarak geçildi. Son kısımda ise *objeOlusturucu* ve *kisaYoluCiz* fonksiyonları sırasıyla çağrıldı.

Altın ve mantar oluşumunu yöneten *objeOlusturucu* metodu iki adet zamanlayıcıyı başlatır. Bunlar 7 ve 10 sn.’lik aralıklarla *altinOlustur* ve *mantarOlustur* metotlarını çağırırlar. Altınlar *altinOlustur* ile Altın sınıfındaki *altinOlustur* metodu ile oluşturulur. Farklı konumlarda oluşmalarını sağlamak için *altinKontrolEt* fonksiyonuna gönderilirler. Burada altınların konumları kontrol edilir ve bir boolean değer döndürülür. Daha sonra altınlar ekrana çizilir. Mantar da aynı şekilde mantarOlustur ile oluşturulur ve konumuna çizilir.

En kısa yolu çizmeye yarayan *kisaYoluCiz* fonksiyonu bir *dugumOlustur* metodunu tetikler ve bir düşman nesnesinin *enKisaYol* fonksiyonuna düşmanın ve oyuncunun bulunduğu düğümleri parametre olarak geçer. Bu fonksiyon Dijkstra ile en kısa yolu hesaplar ve sınıftaki bir değişkene atar. Daha sonra bu yola göre ekrandaki kareler açık mavi veya açık yeşil renklendirilerek yol belirtilir. Bu işlem ikinci düşman için de tekrarlanır.

Oyuncunun hareket edeceği yönü alan *oyuncuHareket* fonksiyonu, ilgili yönü oyuncu karakterinin özelliklerine göre kontrol eder. Uygun olması halinde oyuncuyu yeni konuma taşır ve çizdirir. Ardından *dusmanHareket* fonksiyonun çağırır.

Düşmanın hareketini kontrol eden fonksiyon önce *kisaYoluTemizle* ile eski yolu temizler. Ardından *kisaYoluCiz* ile yeni yolu tespit eder. Bu yeni yol üzerinden düşman karakterinin özelliklerine göre ilerler. Eğer düşman karakter ile oyuncu aynı konuma gelmiş ise düşman karakter tekrar kapısına döndürülür. Oyuncu puan kaybeder. Sonunda ise *puanGuncelle* ile yeni puan hesaplanır ve oyunun durumu belirlenir.

Puan sıfırın altına inmiş ise oyun sonlandırılır. Ekran *haritaSil* ile temizlenir ve oyun tamamlanır. Oyuncu çıkış kapısına ulaşmış ise işlemler tekrarlanır ve oyun kazanılmış olur.

Oyuncu hareketinden sonra tetiklenen *objeToplama* fonksiyonu ile de oyuncunun bir objeyle aynı konuma gelmesi halinde puanı arttırılır.

**4.3. İstatistik**

Program kodu boşluklar dahil yaklaşık 950 satırdan oluşmaktadır. Kullandığımız kütüphaneler ve ne için kullandığımız kabaca aşağıdaki gibidir:

*javafx*

Çizim yapmak için kullandığımız kütüphanedir. Yaklaşık 12 alt kütüphanesi import edilmiştir.

*java.io*

Dosya okuma işlemleri için kullanıldı.

*java.util*

Random, arrayList, Timer gibi temel sınıflar için kullanıldı.

**4.4. Programın Derlenmesi**

Programın kaynak kodu toplam 14 sınıf, 1 metin belgesi ve 7 resim dosyasından oluşmaktadır. Bu projeyi *Intellij Idea*, *Netbeans* veya farklı bir Java IDE’si kullanarak derleyebilirsiniz. Derlerken dikkat edilmesi gereken mesele ise kullandığınız IDE’ye *JavaFX* kütüphanesinin kurulmuş olması gereklidir.

**Kaynakça**

**1.** JavaFX kütüphanesi kullanım rehberi,

<https://www.javatpoint.com/javafx-tutorial>

**2.** Dijkstra algoritmasının kullanımı,

<https://www.happycoders.eu/algorithms/dijkstras-algorithm-java/>

**3.** Çeşitli sorunları çözmek için,

<https://stackoverflow.com/>

