**PROGRAMLAMA LABORATUVARI 1**

**3. PROJE**

Yunus Emre GÜL

Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi

[yunemregul@gmail.com](mailto:yunemregul@gmail.com)

**Özet**

Bu doküman Programlama Laboratuvarı 1 dersi 3. Projesi için çözümümü açıklamaya yönelik oluşturulmuştur. Dökümanda 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 projemi hazırlarken kullandığım kaynaklar bulunmaktadır.

**1. Proje Tanımı**

**2. Araştırmalar ve Yöntem**

Projeye proje tanımında anlatılan sınıfları oluşturmakla başladım. Sınıfları oluştururken düşündüren tek şey Oyuncu sınıfındaki hangi fonksiyonun *abstract* olması gerektiğine karar vermek oldu. Projede anlatırken de dikkat çekilmesi üzerine *kartSec* fonksiyonunun *abstract* olması gerektiğine karar verdim. Böylece bu fonksiyonun hem Kullanıcı sınıfında hem Bilgisayar sınıfında tanımlanması zorunlu olacak ve farklı şekilde çalışabileceklerdi.

Projede belirtilen Pokemon ana sınıfını ve altındaki on farklı sınıfı tanımlarken, kullanmamız zorunlu olan birçok nesneye yönelik programlama özelliğini hali hazırda kullanmış oluyoruz. Her alt sınıf Pokemon sınıfından oluşacağı için *inheritance* özelliğini, sınıflardaki bazı özellikleri *private* olarak tanımladığımndan ve dışarıdan *get, set* metodları ile ulaşım sağladığımdan *encapsulation* özelliğini kullanmış oluyorum. Pokemonların her alt sınıfını birlikte Pokemon tipinde bir *kartListesi* değişkeninde sakladığım için *polymorphism* özelliğini ve Oyuncu sınıfında *abstract* tip kullandığım için *abstraction* özelliğini kullanmış oluyorum. Böylece projede kullanılması zorunlu olan tüm özellikler sınıfları oluştururken kullanılmış oluyor.

Projede nasıl yapılması gerektiğini düşündüren bir diğer nokta da oyunun nasıl yönetileceğine karar vermekti. Bunun için en uygun ve gerçeği de temsil eden yöntemin Masa isminde bir sınıf oluşturmak olduğuna karar verdim. Masa sınıfı, kartları dağıtmak ve kendisinde oyuncuları bulundurmak gibi gerçek bir kart oyununda masa ne yapıyorsa onu yapacaktı. Bu yöntem işlerimi gayet kolaylaştırdı. Masa sınıfına oyunu yönetmeye dair temel fonksiyonları tanımladım. Oyunu yönettiği için kullanıcı arayüzünün de Masa sınıfında çizilmesinin uygun olduğuna karar verdim. Masa sınıfının işlevi ile ilgili detaylı bilgiye bölüm 4.2.’de ulaşılabilir.

Masa sınıfını kullanırken karşılaştığım bir sorun da masa sınıfı kendi kartListesi dizisinden oyunculara kart verirken kartın *alias* ını veriyor olmasıydı. Bu da beklenmedik sorunlara sebep oluyordu örneğin masadan oyuncuya kart verip masadaki kartı kullanılmış olarak işaretlediğimde oyuncudaki kartta kullanılmış olarak işaretleniyordu. Bu sorunu çözmek için bir süre uğraştım, nesne klonlama, nesne instantiation yöntemlerini araştırdım. En sonunda basit bir çözüm olarak masadan oyuncuya kart verirken *new* terimini çağırarak verdim. Bu çözümdeki sorun oyuncuya vereceğim kartın tipi 10 farklı pokemon tipi arasında değiştiğinden uzun bir *if else* ya da *switch* yapısı kullanmam gerekiyordu. Bunu da kullanıcıya verilen kart nesnesini *Class.forName* şeklinde bularak ve buradan da *newInstance* metodunu çağırıp yeni nesneyi oluşturarak çözdüm.

Takıldığım bir diğer nokta ise Masa sınıfında arayüzü nasıl yapacağıma karar vermekti. Önce her kart için farklı bir panel ya da buton komponenti eklemeyi düşündüm ancak bu, animasyon yapmak istersem, örneğin kartları hareket ettirmek, beni ve projeyi kısıtlayacaktı. Ayrıca her kart için farklı bir komponent kullanırsam her komponente ayrı ayrı ulaşmak da işleri uzatacaktı. Bu yöntem yerine tüm çizimleri *paintComponent* fonksiyonun içinde yaptım. Böylece Masa’da yaptığım tüm çizimlerim dinamik ve yönetimi kolay oldu. Bu yöntemdeki sorun hazır bir editör kullanamayacak olmamdı, çizilen her element için koordinatları ve boyutu kendim bulmam gerekiyordu ancak benim için sorun olmadı. Bu yöntemin getirdiği bir diğer sorun da buton gibi hazır bir yapı olmadığından çizimlerde tıklama işlemini kendim sağlamam gerekti. Bunun için de çizim yaptığım her karede kullanıcının mouse konumunu aldım, mouse konumunun eksenlerde belirli aralıklarda olup olmamasına göre nereye tıkladığına karar verdim.

Çizim yaparken karşılaştığım bir sorun da *JFrame* üzerinde *repaint* metodunu sürekli çağırdığımda takılan ve sürekli görünmez olup tekrar gelen bir *JFrame* elde etmem oldu. Bunun sebebi *JFrame* ana komponent olduğundan üzerinde *repaint* çağırmak bütün çerçeveyi tekrar çizdiriyordu bu da yavaş oluyordu. Bunu da çizimleri *JFrame* üzerinde değil de *JPanel* üzerinde yaparak çözdüm. Böylece *repaint* fonksiyonunu çağırdığımda bütün çerçeve yenilenmeden sadece *JPanel* elementi yenilendiği için hızlı ve stabil bir görüntü elde edebilmiş oldum.

Programın işlevsel kısmıyla alakalı herşey bittiğinde görüntüsünü iyileştirmek amacıyla masa ve menü tasarımı yaptım. Kartların savaşmaya başladığı ana göre kartları hareket ettirip çarpışma gibi gözüken bir animasyon elde ettim.

**3. Geliştirme Ortamı**

Projemi Linux sistemde, Intellij IDEA üzerinde geliştirdim. Projemin gelişimini ve versiyonlarını takip edebilmek için de Git versiyon kontrol sistemi kullandım.

**4. Kod Bilgisi**

**4.1. UML Diyagramı**

Kısım ektedir. [1](uml\_diyagrami.jpg)

**4.2. Akış Diyagramı**

Kısım ektedir. [2](akis\_diyagrami.jpg)

**4.3. Algoritma**

Bu kısımda projenin genel algoritmasına açıklık getireceğim.

Oyun başladığında kullanıcıya oyunun nasıl oynanmasını (Bilgisayar vs Bilgisayar veya Kullanıcı vs Bilgisayar) istediği sorulan bir seçim ekranı sunuluyor. Kullanıcı kendisi ve bilgisayarın savaşmasını seçti ise oyuncu ismi soruluyor. Oyuncu ismi de girildikten sonra geriye oyunun yönetilmesi kalıyor.

Oyunu yöneten ana sınıf Main sınıfı ancak yönetime dair tüm fonksiyonlar (*kartDagit, kartVer, ...*) ve oyuncu nesneleri Masa sınıfında bulunuyor. Bu da Main sınıfını kirletmeyen temiz bir kod sağlıyor, oyunun mantığı veya kuralları değişse bile değiştireceğim şeyler basit olacaktır. Main sınıfının kodlarına bakıldığı zaman oyunun ana mantığı ve kuralları kolayca görülebilir.

Masa sınıfında oluşturduğum *masaState* değişkeni oyunun hangi durumda olduğunu tutuyor. Örneğin oyun başlamadı ise *masaState*  değişkeni 0, oyun bitti ise 4 değerini tutuyor. Masanın durumuna kolayca ulaşabildiğim için yönetimi de kolay oluyor. Main sınıfında program bitene kadar çalışacak bir *while* döngüsü var. Bu döngü oyun boyunca Masada olacak şeyleri yönetiyor. Eğer oyuncular hazırsa ve kartlar dağıtılmayı bekliyorsa, bu *masaState* değerinin 1 olduğu anlamına gelir, Main sınıfı *masa.kartDagit* emrini veriyor.

Kartlar dağıtıldıktan sonra oyun durumu (*masaState*)değeri 2 olacaktır yani oyuncuların kart oynaması bekleniyordur. Bu durumda Main sınıfının tek kontrol ettiği şey eğer oyunun “Bilgisayar vs Bilgisayar” modunda olup olmadığıdır. Eğer oyun bu modda ise iki bilgisayarın da kart seçmesini emreder (*masa.oyuncular[0].kartSec*). Eğer oyun bu modda değil ise yani “Kullanıcı vs Bilgisayar” modunda ise Main sınıfı kullanıcıya kart seçmesini emredemeyeceği için kullanıcı kart seçimi yaptığında Masa sınıfı otomatik olarak karşıdaki oyuncunun da kart seçmesini emreder. Bu da oyunun devam etmesini sağlar.

Eğer masa kapışma durumunda ise Main sınıfı kapışan kartları değerlendirip değerlendirme sonucunda kazanan oyuncuya 5 skor ekler. Ardından masada ya da oyuncularda kart kalıp kalmamasına göre oyunun devam edip etmeyeceğine karar verir.

En sonda kartlar bittiğinde ve kazanan belli olduğunda Main sınıfı masa nesnesi üzerine kazananları kaydeder, masa nesnesi de buna göre oyunun bittiğini ve kazananları gösterir.

Masa sınıfındaki algoritmadan bahsedecek olursam, masa sınıfı yapıcı metodunda bir *kartListesi* alır. Girilen bu *kartListesi* masanın destesi olarak kabul edilir. Masa sınıfı üzerindeki *paintComponent* metodu ile kartların ve diğer çizimi yapılacak elementlerin çizimi yapılır. Masanın kart dağıtma özelliği sayesinde oyunculara kart verilebilir, verilen kart masada ‘kullanıldı’ olarak işaretlenir. Masadaki ‘rastgeleKart’ fonksiyonu ile masanın elindeki kullanılmamış kartlardan rastgele bir tanesi döndürülür.

**4.3. İstatistik**

Program kodu boşluksuz ve yorumsuz yaklaşık 937 satır ve birçok dosyadan oluşmaktadır. Kod düzenini sağlamak için yaklaşık 203 boş satır kullanılmıştır. Okuyucuya izlenim oluşturması için yaklaşık 138 yorum satırına yer verilmiştir.

Kullandığım hazır paketler ve ne için kullandığım kabaca aşağıdaki gibidir:

*javax.imageio.ImageIO*

Masada kullanılan imajları açmak için.

*javax.swing.\**

Masada çizim yapabilmek için.

*javax.awt.\**

Masada çizime yardımcı fonksiyonlar ve tıklama *event*ini yakalayabilmek için.

*java.util.Random*

Rastgele kart seçmek ve diğer rastgele işlemler için.

**Kaynakça**

1. *paintComponent* veçizime dair anlatımlar: <https://www.bogotobogo.com/Java/tutorials/javagraphics3.php>
2. *MouseListener*, klonlama, *new* vs *newInstance:* <https://www.geeksforgeeks.org/mouselistener-mousemotionlistener-java> <https://www.geeksforgeeks.org/clone-method-in-java-2/> https://www.geeksforgeeks.org/new-operator-vs-newinstance-method-java/
3. Karşılaştığım çeşitli problemler: <https://stackoverflow.com/>