Programlama Labaratuvarı 2.Proje Raporu

Sadık Gölpek
sadikgolpek@gmail.com

Abdullah Önder
abdullahonder726@gmail.com

I.Özet

Bu projede, nesne yönelimli programlama (OOP) prensiplerine dayalı bir savaş araçları kart oyunu tasarlanmıştır. Oyuncu ve bilgisayar, başlangıçta belirli kategorilerde (hava, kara, deniz) savaş araçları kartları seçer ve sırayla kartlarını oynararak karşılaşmalar yapar. Oyunun amacı, oyuncunun stratejik seçimlerle kartlarını kullanarak rakibine karşı zafer kazanmasıdır. Java dilinde geliştirilen proje, kullanıcıya oyun sırasında kart seçimlerini yapma ve strateji oluşturma fırsatı sunmaktadır.

Proje, nesne yönelimli programlama (OOP) mantığıyla tasarlanmıştır. Bu sayede oyun elemanları (kartlar, oyuncular, oyun kuralları vb.) sınıflar ve nesneler aracılığıyla temsil edilmiştir.

Bu proje için **Java** dilinin tercih edilmesinin temel sebebi, Java'nın nesne yönelimli yapısının, büyük projelerde esneklik ve sürdürülebilirlik sağlamasıdır. Ayrıca, Java'nın açık kaynak kodlu yapısı, geniş kütüphane desteği ve platform bağımsız çalışabilmesi, yazılım geliştirme sürecinde önemli avantajlar sunar. Java, farklı işletim sistemlerinde çalışabilmesi ve taşınabilirliği ile öne çıkarken, geniş geliştirici topluluğu sayesinde de güçlü destek alabileceğiniz bir dil sunmaktadır.

Projede, nesne yönelimli programlamanın temel ilkelerinden biri olan **encapsulation** (kapsülleme) kullanılmıştır. Bu kapsamda, her sınıfın içindeki veriler özel getter ve setter metotları ile kontrol edilmiştir. **Getter** metodları, sınıfın özel verilerine dışarıdan erişimi sağlarken, **setter** metodları ise bu verilere dışarıdan değer atanmasını sağlamaktadır. Bu metodlar, veri güvenliğini sağlamak ve nesnelerin durumunu kontrol etmek amacıyla doğru bir şekilde kullanılmıştır.

Ayrıca, sınıflar arasındaki ilişkileri yönetmek için **method overriding** (metodun ezilmesi) teknikleri de kullanılmıştır. Bu sayede, alt sınıflar, üst sınıf metodlarını kendi ihtiyaçlarına göre yeniden tanımlayabilmektedir. Bu işlem, özellikle oyun kartlarının farklı türleri (hava, kara, deniz) gibi varyasyonlara sahip sınıflarda, her bir sınıf için işlevselliği sağlamak için tercih edilmiştir.

II.Giriş

Bu projede, oyuncuların stratejik kararlar alarak rakiplerine karşı savaşmalarını sağlayan bir savaş araçları kart oyunu tasarlanmıştır. Oyun, belirli kategorilerde (hava, kara, deniz) yer alan savaş araçları kartları üzerinden oynanır. Oyuncular, başlangıçta yalnızca belirli kartları seçebilirler; örneğin, uçak, obüs ve fırkateyn kartları seçilebilirken, siha, sida ve KFS gibi kartlar yalnızca oyuncu veya bilgisayar belirli bir puan seviyesine ulaştığında açılabilir. Bu sistem, oyunun ilerlemesiyle birlikte yeni stratejilerin ve kart seçeneklerinin devreye girmesini sağlar.

Oyun, temelde kart seçimi ve karşılaştırma aşamalarından oluşur. Oyuncu her hamlede, elindeki kartlardan üç tanesini seçer, ardından bilgisayar rastgele üç kart seçer. Bu kartlar karşılaştırılır ve kazanan belirlenir. Sonraki hamlede, her iki tarafa da yeni kartlar dağıtılır. Eğer oyuncu veya bilgisayar belirli bir seviyeye (örneğin 20 puan) ulaşırsa, oyun yeni kartları erişilebilir hale getirir. Oyun, ya belirli bir hamle sayısının tamamlanmasıyla ya da bir oyuncunun kartlarının tükenmesiyle sona erer.

Bu projede Java programlama dili kullanılmıştır ve nesne yönelimli programlama (OOP) prensiplerine dayalı olarak tasarlanmıştır. Oyun, 13 farklı sınıf ile modellenmiştir ve her bir sınıf oyun içindeki bir kavramı temsil etmektedir. Ana sınıf olarak Oyun.class belirlenmiş ve oyunun tüm akışı burada yönetilmektedir. Bu sınıf, oyun başladığında kartları dağıtır, hamleleri gerçekleştirir ve oyunun bitiş koşullarını kontrol eder.

Oyuncu.class, oyuncunun kartlarını ve özelliklerini tutan bir sınıftır.

SavasAraclari.class sınıfı, tüm savaş araçlarının temel sınıfıdır ve diğer sınıflar bu sınıftan kalıtım alarak özel araçlar oluşturur. KaraAraclari.class, DenizAraclari.class ve HavaAraclari.class sınıfları, savaş araçlarının her bir kategorisini temsil eder. Bu sınıflar, temel özellikler ve işlevler bakımından SavasAraclari.class sınıfını miras alır.

Bu sınıflardan KaraAraclari.class sınıfı, Obus.class ve KFS.class sınıflarını içermektedir. Obus ve KFS, kara araçlarının farklı türleridir ve her biri farklı özelliklere sahip olabilir. Benzer şekilde, HavaAraclari.class sınıfından türeyen Ucak.class ve Siha.class sınıfları, hava araçlarının farklı türlerini temsil eder. DenizAraclari.class sınıfı ise Firkateyn.class ve Sida.class gibi deniz araçlarını içerir.

Her bir sınıf, oyun kuralları doğrultusunda belirli özelliklere ve davranışlara sahiptir. Örneğin, her savaş aracı kartı dayanıklılık, saldırı gücü gibi özelliklere sahip olabilir ve bu özellikler kartların karşılaştırılmasında kullanılır. Ayrıca, getter ve setter metodları, her bir sınıfın özelliklerini güvenli bir şekilde dışarıya sunar ve değiştirilmesini sağlar. Override işlemleri ise alt sınıfların, üst sınıflardan miras aldıkları metodları kendi ihtiyaçlarına göre özelleştirmelerine olanak tanır.

Bu projedeki sınıfların ilişkisini ve yapısını daha iyi anlayabilmek için UML diyagramı hazırlanmıştır. UML diyagramı, sınıfların birbirleriyle olan ilişkilerini, kalıtımı ve metod çağrılarını görsel bir şekilde sunar. Bu diyagramı, projenin raporunun sonuna ekleyerek, oyunun yapısal tasarımını daha açık bir şekilde göstereceğiz.

III. Yöntem

Bu tasarımda, sınıflar ve nesneler aracılığıyla oyun öğeleri temsil edilmiş, ilişkiler ve davranışlar belirlenmiştir. OOP'nin temel özellikleri olan **kalıtım**, **encapsulation (kapsülleme)** ve **polymorphism (çok biçimlilik)** prensipleri, oyunun işleyişinde etkin bir şekilde kullanılmıştır.

Projede, **HavaAraclari.class**, **KaraAraclari.class**, **DenizAraclari.class** ve **SavasAraclari.class** gibi sınıflar, **abstract** yapıda tanımlanmıştır. Bu sınıfların **abstract** olarak tanımlanmasının temel nedeni, bu sınıfların doğrudan nesne örneği oluşturmayacak soyut sınıflar olmasıdır. Yani, bu sınıflar yalnızca ortak özellikleri ve davranışları içeren temel sınıflardır. **KaraAraclari**, **DenizAraclari** ve **HavaAraclari** gibi sınıflar ise, **SavasAraclari** sınıfından türetilmiş daha özel sınıflardır.

Abstract sınıf kullanmanın önemi, bir nesneye ait ortak özellik ve metotları bir arada tutarak, türeyen sınıflara bu özelliklerin miras verilmesini sağlamaktır. Böylece, her bir araç türü için tekrarı engelleyerek kodun daha sürdürülebilir ve esnek olmasını sağlarız. Örneğin, tüm savaş araçlarının sahip olması gereken **dayanıklılık**, **saldırı gücü** gibi özellikler **SavasAraclari** sınıfında tanımlanmış ve **KaraAraclari**, **HavaAraclari** ve **DenizAraclari** sınıfları bu özellikleri devralmıştır.

Projenin önemli bir parçası olarak **super** metodunu kullandık. **Super**, alt sınıfların üst sınıfın constructor'ını (yapıcı metodunu) veya metodlarını çağırmasını sağlayan bir anahtar kelimedir. Bu, alt sınıfların, üst sınıflarından miras aldığı özellikleri kullanabilmesini sağlar ve **kalıtım** kavramını destekler. Örneğin, bir **KaraAraclari** sınıfı, **SavasAraclari** sınıfından miras aldığı **dayanıklılık** özelliğini kullanırken, **super** kelimesi aracılığıyla üst sınıfın **constructor** metodunu çağırarak bu değeri alabilir.

Super kullanımı, nesne yönelimli programlamada sınıflar arasında hiyerarşik bir ilişki kurar. Alt sınıflar, üst sınıflarındaki fonksiyonları ve verileri kullanarak daha karmaşık davranışlar oluşturabilir. Bu, yazılımın yeniden kullanılabilirliğini artırır ve aynı işlevsellik için her seferinde sıfırdan yazılım geliştirmek yerine var olan yapıyı kullanma imkanı sağlar.

Oyuncular, her hamlede 3 kart seçerler ve bu kartlar karşılaştırılarak kazanan belirlenir. Bilgisayarın kart seçimini **random** (rastgele) olarak yapıyoruz. Bilgisayar, kendi elindeki kartlardan 3 kartı rastgele seçer ve bu seçimde, stratejik kararlar yerine tamamen şansa dayalı bir mekanizma işler. Bu, oyuna farklı bir dinamik katmakta ve oyuncunun her hamlede neyle karşılaşacağını kestiremeyeceği bir yapı oluşturmakta faydalıdır.

Oyun boyunca her iki oyuncu da kart seçer ve bunlar karşılaştırılır. Oyunun sonunda, **toplam skor değeri yüksek olan taraf kazanır**. Eğer skorlar eşitse, bu durumda tarafların elindeki **elenmemiş kartların toplam dayanıklılık** miktarı karşılaştırılır. **Toplam dayanıklılık** değerleri arasında fark, kazananın skoruna eklenir. Bu, yalnızca şansa dayalı bir oyun oynamaktan ziyade, stratejik kararların ve kartların dayanıklılığının da oyunun sonucunu etkilediği bir oyun yapısı oluşturur.

Nesne yönelimli programlamayı bir organ yapısına

benzetmek mümkündür. Her bir sınıf, bir organı temsil eder ve bu organlar arasındaki etkileşimler ile oyun süreci işler. Nesneler, birbirleriyle hiyerarşik bir biçimde iletişim kurar ve her bir sınıf, alt sınıflarına farklı özellikler ve işlevler aktarır. Tıpkı bir organizmanın farklı organlarının belirli işlevleri yerine getirmesi gibi, burada da her sınıf belirli bir işlevi yerine getirir.

Nesne yönelimli programlama, **kapsülleme** sayesinde, her sınıfın yalnızca gerekli olan verileri dışarıya sunmasını sağlar. Bu, yazılımın sürdürülebilirliğini ve esnekliğini artırır. Ayrıca, **polymorphism (çok biçimlilik)** ve **inheritance (kalıtım)** gibi prensipler sayesinde, farklı sınıflar birbirleriyle etkili bir şekilde çalışabilir ve birbirinden bağımsız olarak genişletilebilir.

Projede, nesnelerin birbiriyle benzersiz olması ve her bir nesnenin doğru bir şekilde kopyalanabilmesi için **clone** metodunu kullandık. **Clone** metodu, nesnelerin kendilerini çoğaltarak yeni bir nesne örneği oluşturmamıza olanak tanır. Bu sayede, bir nesnenin kopyası oluşturulabilir ve her iki nesne birbirinden bağımsız çalışabilir. Bu yaklaşım, nesnelerin doğru şekilde yönetilmesi ve oyun içerisindeki her kartın benzersiz bir şekilde temsil edilmesi açısından önemli olmuştur.

IV. Sonuç

Bu projede, nesne yönelimli programlamanın (OOP) avantajlarından tam anlamıyla faydalanılmış ve yazılım tasarımında bu prensipler etkin bir şekilde kullanılmıştır. Savaş araçları kart oyunu, sınıfların ve nesnelerin doğru bir şekilde organize edilmesiyle tasarlanmış, her bir sınıf belirli bir işlevi yerine getirecek şekilde yapılandırılmıştır. Kalıtım, polimorfizm ve encapsulation (kapsülleme) gibi temel OOP ilkeleri, oyun içerisindeki farklı kart türlerinin birbirleriyle etkileşimde bulunmasını ve doğru bir oyun akışının sağlanmasını mümkün kılmıştır.

Ancak, projede görselleştirme aşamasına yeterli süre ve kaynak ayıramadık. Oyun görsel bir kullanıcı arayüzü (GUI) ile zenginleştirilebilirdi, fakat optimizasyon ve performans iyileştirmeleri için daha fazla zamana ihtiyaç duyulmuş ve görselleştirme kısmı yapılmamıştır. Başlangıçta görselleştirme için **Swing** kütüphanesi ile bir arayüz geliştirmeyi planlamıştık. Ancak, bu kütüphaneyle çalışma aşamasına yeterince zaman ayıramadık ve bunun yerine oyunun işleyişini tamamen **terminal üzerinden yönetilen bir oyun** olarak tasarladık. Bu şekilde, oyuncuların hamleleri metin tabanlı bir arayüzle gerçekleştirebilmesi sağlandı. Bu, kullanıcı etkileşimini sağlamak için yeterli olmuş olsa da, görsel bir arayüzle oyun deneyimi çok daha zenginleştirilebilirdi.

Projede her adım, oyun sırasında gerçekleşen işlemler ve hesaplamalar **txt dosyasına kaydedilmiştir**. Bu, her oyun seansı sonrası oyunun ilerleyişinin ve her adımın kaydını tutmamıza olanak sağlamıştır. Bu sayede, oyuncu hamlelerinin ve karşılaştırmalarının izlendiği ayrıntılı bir kayıt dosyası oluşturulmuştur. Ayrıca, projenin **tüm kaynak kodları** da ayrı bir txt dosyasına yazdırılmıştır, böylece projeyi daha sonra incelemek isteyen kişilerin kodları takip etmeleri kolaylaşmıştır. Bu, projenin paylaşılabilirliğini artırarak yazılımın geliştirilmesi ve bakımı açısından önemli bir adım olmuştur.

Sonuç olarak, projenin geliştirilmesinde nesne yönelimli programlama yaklaşımları başarıyla uygulanmış ve oyun mantığı düzgün bir şekilde çalışmaktadır. Görselleştirme aşamasına geçilememiş olsa da, terminal üzerinden yönetilebilen oyun işlevsel ve işleyiş açısından güçlüdür. Gelecekte, zaman ve kaynak imkanları doğrultusunda görselleştirme aşamasına geçilebilir ve oyunun kullanıcı deneyimi daha da geliştirilerek zenginleştirilebilir.

V. Yazar Katkıları

Bu proje, iki kişilik bir ekip tarafından geliştirilmiştir ve her bir ekip üyesi belirli alanlarda katkı sağlamıştır. Projenin savaş mekaniği tasarımı, her iki ekip üyesinin ortak çalışmasıyla oluşturulmuştur. Abdullah

Önder, savaş araçları kartlarının sınıf tanımlamaları ve sınıflar arasındaki ilişkilerin kurulması üzerinde yoğunlaşmış, ayrıca savaş mekaniği ile ilgili temel yapıları kurmuştur. Sadık Gölpek ise, savaş mekaniği ve dosya yazdırma işlemlerini geliştirerek, oyunun mantığını daha sağlam ve işlevsel hale getirmiştir. Sadık ayrıca, dosya kayıtlarının doğru bir şekilde yapılmasını ve her adımın düzgün bir şekilde takip edilmesini sağlayarak oyunun işleyişine katkı sağlamıştır.

Bu proje, nesne yönelimli programlamanın (OOP) kurallarına dayanarak geliştirilmiştir ve bu süreçte nesne yönelimli tasarımın oldukça kuralcı bir yapıya sahip olduğunu öğrendik. Özellikle kalıtım, kapsülleme, polimorfizm gibi OOP ilkelerinin doğru bir şekilde uygulanması gerektiğini öğrendik.

Bu, bizim ikinci projemizdi ve nesne yönelimli programlama prensiplerine dayalı bir oyun tasarımında edindiğimiz deneyimler, yazılım geliştirme becerilerimizi önemli ölçüde geliştirdi. Proje boyunca öğrendiğimiz nesne yönelimli yazılım tasarımı ve uygulama süreçleri, gelecekteki projelerimizde daha verimli ve sürdürülebilir yazılımlar üretmemizi sağlayacak temelleri attı.

VI. Kaynakça:

Java ve Nesne Yönelimli Felsefesini anlamak için:

https://docs.oracle.com/javase/tutorial/

ChatGPT. Aldığımız bazı hatalarının çözümü için kullanılan bilgiler:

https://chatgpt.com/

VII.Tablo

UML diyagramımız bu şekildedir: