Kocaeli Üniversitesi
Bilgisayar Mühendisliği Bölümü
Taş Kağıt Makas

Samet Ahmet Şahin, Nusret YILDIZ
210201041@kocaeli.edu.tr, 210201040@kocaeli.edu.tr
Bilgisayar Mühendisliği Bölümü, Kocaeli Üniversitesi Umuttepe Kampüsü, İzmit/KOCAELİ

Programlama Laboratuvarı I dersinin 2. projesi olan Taş Kağıt Makas; proje dökümanında yer alan isterler göz önünde bulundurularak geliştirilen, kullanıcının grafiksel arayüz aracılığıyla insan-bilgisayar oyunu ya da bilgisayar-bilgisayar oyunu başlatabildiği ve oyunun maksimum kaç tur süreceğini ayarlayabildiği, yine aynı grafiksel arayüz aracılığıyla bilgisayar-bilgisayar oyununda oyunun gidişatını görebildiği, ayrıca insan-bilgisayar oyununda bilgisayar-bilgisayar oyununun sunduğu özelliklere ek olarak oyun başlarken kendi nesne topluluğunu oluşturabildiği ve oyun sırasında bu nesne topluluğundan bir nesne seçimi yapmak suretiyle oyunun akışına katkı sağlayabildiği ve tüm oyun akışının bir ".log" dosyasına yazıldığı bir bilgisayar oyunu projesidir.

Taş Kağıt Makas projesinin oyun mekaniklerini ve web serverını halleden tarafı Java diliyle geliştirilmiş olup grafiksel arayüz için Godot Game Engine kullanılmış ve grafiksel arayüz için GDScript dili kullanılmıştır.

I. Giriş

Projenin oyun mekaniklerini halleden kısmı; oyun başlatıcıları (insan-bilgisayar oyunu başlatıcısı, bilgisayar-bilgisayar oyunu başlatıcısı ve bu başlatıcıları çağıran ana başlatıcı fonksiyonu), oyun akışının kaydedildiği dosyanın başlatıcısı, oyun akışını ilgili dosyaya yazan dosya yazıcısı, oyun durumunu kontrol eden bir denetleyici, oyuncuların elindeki tüm nesneleri kullanıp kullanmadığını kontrol eden ve eğer kullanmışlarsa aynı nesneleri tekrar kullanmalarını sağlayan bir denetleyici, oyunun bir tur oynaması için gerekli işlemleri yapan ve oyun akışını bir String'e kaydeden bir fonksiyon ve tüm bu fonksiyonları yöneten bir oynanış yöneticisi olmak üzere sekiz bölümden oluşmaktadır. Oyuncuların elindeki nesnelerin tutulması için ArrayList kullanılmıştır.

Projenin grafiksel arayüz kısmı bir Java web server aracılığıyla Godot Game Engine ile oyun mekaniklerinin bağlanması prensibiyle çalışmaktadır. Tüm kullanıcı girdileri bir bilgisayar faresi ile alınmaktadır. Gerisinin Samet tarafından anlatılması üzerinde mutabık kalınmıştır.

II. Yöntem

Oyun nesnelerinin ile oyuncuların bellekte düzenli bir şekilde tutulabilmesi ve rahat bir şekilde erişilebilmesi için nesneye yönelik programlama konsepti kullanılmış ve bu oyun nesneleri ile oyuncular birer "sınıf" olarak tanımlanmıştır. Bu sınıflar arasında kalıtımlar mevcuttur.

Oyun mekanikleri mikrofonksiyon mimarisi temel alınarak her mikrofonksiyonun sadece bir tane iş yapması ve bu mikrofonksiyonların toplu şekilde çağrılması prensibiyle hazırlanmıştır.

Aşağıda bu mikrofonksiyonlar detaylı bir şekilde açıklanmıştır.

A. InitializeLogFile

Java'da bulunan java.io.File kütüphanesinin çalışma mantığı çerçevesinde bir try-catch yapısı oluşturulmuş ve try bölümünün içinde File türünde "logFile" isimli bir değişkene yeni bir File nesnesi atanmıştır. Bu nesnenin constructor'ına parametre olarak "RockPaperScissor [dd-MM-yyyy HH.mm.ss].log" String'i verilmiştir. Bu String dosya adını temsil etmektedir. Ardından createNewFile() fonksiyonu çağırılarak dosya oluşturulmuştur.

Dosyanın oluşturulması sırasında bir olumsuzluk meydana geldiği takdirde bu try-catch yapısının catch bölümü çalışmaktadır. catch bölümünde File nesnesinin verdiği hata konsol ekranına yazdırılmıştır.

B. WriteToLogFile

Bu mikrofonksiyon oyunun her turunun sonunda çağırılmaktadır ve amacı o ana kadar String türündeki logText isimli bir değişkene kaydedilen tüm oyun akışını ilgili dosyaya yazmaktır.

Java'da bulunan java.io.FileWriter kütüphanesinin çalışma mantığı çerçevesinde bir try-catch-finally yapısı oluşturulmuş ve try bölümünün içinde FileWriter türünde "fw" isimli değişkene yeni bir FileWrite nesnesi atanmıştır. Bu nesnenin

constructor'ına parametre olarak mevcut log dosyasının adı ve dosyaya yazılacak metnin dosyanın sonuna eklenmesi gerektiğini belirten bir boolean ifadesi verilmiştir. Bu nesnenin oluşturulmasının ardından logText String'inin içeriği bu nesne aracılığıyla ilgili dosyaya yazılmıştır.

String'in dosyaya yazılması sırasında bir olumsuzluk meydana geldiği takdirde bu try-catch-finally yapısının catch bölümü çalışmaktadır. catch bölümünde FileWriter nesnesinin verdiği hata konsol ekranına yazdırılmıştır.

try ve catch yapılarının içlerindeki işlemlerin sona ermesini takiben finally bölümü çalışmaktadır ve bu finally bölümünde logText String'inin içi boşaltılmaktadır.

C. InitializeGame

İki mikrofonksiyonun overload edilmesiyle oluşan bu mikrofonksiyon parametre almadığı zaman isHumanGame boolean'ına false değerini atayıp InitializeAiAiGame() mikrofonksiyonunu çağırmaktadır. Bu mikrofonksiyon int dizisi türünde ve startingDeck isimli bir parametre aldığı zaman ise isHumanGame boolean'ına true değerini atayıp InitializeHumanAiGame() mikrofonksiyonunu startingDeck parametresiyle çağırmaktadır.

D. InitializeAiAiGame

Bu mikrofonksiyonda Player türündeki player1 ve player2 değişkenlerine oluşturulan ComputerPlayer nesnelerinin ataması yapılmaktadır.

Öncelikle long türünde oluşturulan startID değişkenine değişkenin değeri 0 ya da 0'dan büyük olana kadar rastgele sayılar atanmaktadır. Gerekli şartın sağlanmasının ardından startID ve "Bilgisayar 1" String'i parametrelerine sahip bir ComputerPlayer nesnesi oluşturulmakta ve bu nesne player1 değişkenine atanmaktadır. Aynı işlemler player2 için de yapılmaktadır, tek fark ComputerPlayer nesnesine verilen parametrede "Bilgisayar 2" String'inin yer almasıdır.

E. InitializeHumanAiGame

Bu mikrofonksiyonda player1 değişkenine HumanPlayer, player2 değişkenine ComputerPlayer nesnelerinin ataması yapılmaktadır.

Öncelikle long türünde oluşturulan startID değişkenine değişkenin değeri 0 ya da 0'dan büyük olana kadar rastgele sayılar atanmaktadır. Gerekli şartın sağlanmasının ardından startID, "Sen" String'i ve startingDeck parametrelerine sahip bir HumanPlayer nesnesi oluşturulmakta ve bu nesne player1 değişkenine atanmaktadır. Aynı işlemler player2 için de yapılmaktadır, tek fark atanan nesnenin türünün HumanPlayer yerine ComputerPlayer olması ve ComputerPlayer nesnesine verilen parametrede "Bilgisayar" String'inin yer almasıdır.

F. CheckAndResetIsUsedFlags

Bu mikrofonksiyonun amacı oyuncuların ellerindeki nesnelerin tamamını kullanıp kullanmadıklarını kontrol etmek ve eğer hepsi kullanılmışsa her nesnenin isUsed boolean'ını false değerine sıfırlamaktır.

boolean türünde resetIsUsedFlag değişkeni oluşturulmuş ve başlangıç değeri olarak true atanmıştır. Önce player1 oyuncusunun elindeki tüm nesnelerde bulunan isUsed boolean'ı ile resetIsUsedFlag boolean'ı arasında mantıksal ve işlemi yapılmış ve bu işlemin sonucu resetIsUsedFlag boolean'ına atanmıştır. Eğer tüm nesneler dolaşıldıktan sonra resetIsUsedFlag boolean'ının değeri true ise player1 oyuncusunun elindeki tüm nesneleri kullandığı anlaşılmış ve ikinci bir döngüyle player1 oyuncusunun elindeki tüm nesnelerin isUsed boolean'ına false değeri atanmıştır.

Yukarıda yazılan işlemler player2 için de uygulanmış ve böylece gerekli kontroller tamamlanmıştır.

G. CheckGameStatus

int türünde currentRound parametresini alan bu mikrofonksiyonun amacı oyunun devam edip etmediğini, devam etmiyorsa hani oyuncunun kazandığını int türünde bir değer aracılığıyla geri döndürmektir.

İçinde bulunulan turun maksimum tur olup olmadığına göre oyun durumunun farklı şekillerde kontrol edilmesi gerektiği için currentRound değişkeni ile maxRound global değişkeni karşılaştırılmakta, eğer currentRound sayısı maxRound sayısından küçükse oyunun devam ettiği, aksi durumda ise oyunun sona ermesi gerektiği anlaşılmaktadır.

Oyun devam ediyorsa oyuncuların ellerindeki nesne sayısı karşılaştırılmaktadır. Eğer birinci oyuncunun nesne sayısı 0'dan büyük, ikinci oyuncunun nesne sayısı 0 ise birinci oyuncunun oyunu kazandığı anlaşılmakta ve 1 değeri return edilmektedir. Eğer birinci oyuncunun nesne sayısı 0, ikinci oyuncunun nesne sayısı 0'dan büyük ise ikinci oyuncunun oyunu kazandığı anlaşılmakta ve 2 değeri return edilmektedir. Eğer iki oyuncunun da elindeki nesne sayısı 0 ise oyunun berabere kaldığı anlaşılmakta ve 3 değeri return edilmektedir. Eğer iki oyuncunun da elindeki nesne sayısı 0'dan büyük ise oyunun hâlâ devam ettiği anlaşılmakta ve 0 değeri return edilmektedir.

Oyun bitmişse oyuncuların skorları karşılaştırılmaktadır. Eğer birinci oyuncunun skoru ikinci oyuncunun skorundan büyükse oyunu birinci oyuncunun kazandığı anlaşılmakta ve 1 değeri return edilmektedir. Eğer ikinci oyuncunun skoru birinci oyuncunun skorundan büyükse oyunu ikinci oyuncunun kazandığı anlaşılmakta ve 2 değeri return edilmektedir. Aksi durumda (iki oyuncunun skorlarının eşit olması durumu) oyunun berabere bittiği anlaşılmakta ve 3 değeri return edilmektedir.

H. PlayARound

Bu mikrofonksiyon oyunun bir turunu yönetmektedir. Oyun akışının ağırlıklı olarak bu mikrofonksiyonda bulunması nedeniyle bu mikrofonksiyonun büyük bir kısmı logText String'ine ve konsola oyun akışıyla ilgili bilgileri yazdıran kodlardan oluşmaktadır.

Oyuncuların bu turda seçtiği nesneler Player.DeckItem türündeki deckItem1 ve deckItem2 değişkenlerine atanmaktadır. Ardından seçilen nesnelerin birbirlerine karşı olan etki değerleri hesaplanmakta ve birer değişkende tutulmaktadır. Hesaplanan etki değerleri rakip nesnelerin dayanıklılıklarından UpdateStats() fonksiyonuyla eksiltilmekte, fonksiyonların döndürdüğü değer birer değişkende tutulmakta ve nesnelerdeki isUsed boolean'larına true değeri atanmaktadır. UpdateStats() fonksiyonlarının döndürdüğü değerler kontrol edilmekte, nesne eleme ve nesne yükseltme işlemlerinden gerekli olanlar yapılmaktadır.

Yukarıda yazan tüm işlemler ilgili log dosyasına yazılmak üzere logText String'ine kaydedilmektedir.

I. HandleGameplay

Bu fonksiyonun amacı yukarıda detaylarıyla anlatılan mikrofonksiyonları belli bir düzende çağırarak oyun akışını yönetmektir.

Bu fonksiyon sırasıyla PlayARound() mikrofonksiyonunu çağırarak oyunu bir tur ilerletmekte, CheckAndResetIsUsedFlags() mikrofonksiyonuyla oyuncuların ellerindeki nesnelerin tamamını kullanıp kullanmadığını kontrol etmekte, CheckGameStatus() mikrofonksiyonunu çağırarak oyunun durumunu bir global değişkene atamakta ve oyun akışına dair elde edilen tüm bilgileri ilgili dosyaya yazmaktadır.

III. Deneysel Sonuçlar

Projenin geliştirilmesi esnasında bazı beklenmedik davranışlar ve hatalarla karşılaşılmış, birtakım beyin fırtınası ve hata ayıklama işlemi sonucu bu istenmeyen davranışlar ortadan kaldırılmış ve proje kusursuz bir hâle getirilmiştir.

Bu beklenmedik davranışlar ve hatalar aşağıda detaylarıyla açıklanmıştır.

A. ComputerPlayer.java - SelectItem

Bu fonksiyonda bilgisayar nesne seçimini nesnenin önceden kullanılıp kullanılmadığını kontrol etmeden yapmaktaydı. Bu da bilgisayarın tüm nesneleri kullanmasına gerek kalmadan aynı nesneyi seçebilmesine sebep olmaktaydı. Bir do-while döngüsü eklenip bilgisayarın seçtiği nesnenin isUsed boolean'ı false olana kadar tekrar tekrar nesne seçmesi sağlanmıştır.

Bu fonksiyonda yer alan başka bir hata ise Random.nextInt() çağrısında üst limit olarak 5 sayısı kullanılmasıydı. Bilgisayarın elindeki nesnelerden en az biri oyundan elendiğinde bilgisayarın elindeki nesne sayısından daha büyük bir sayı seçme ihtimali vardı ve bunun sonucunda java.lang.ArrayIndexOutOfBoundsException hatası verebilmekteydi. Random.nextInt() çağrısında üst limit olarak 5 sayısı yerine super.GetItemDeck().size() yazılarak bu hata giderilmiştir.

B. Game.java - CheckAndResetIsUsedFlags

Bu fonksiyonda nesnelerin isUsed boolean'larının sıfırlanması resetIsUsedFlag boolean'ının false olarak başlatılması ve nesnelerin isUsed boolean'larıyla resetIsUsedFlag boolean'ının mantıksal veya işlemine tabii tutulmasıyla yapılmaktaydı. Bu mantık nesnelerde isNotUsed tarzı bir boolean kullanılsaydı doğru olabilmekteydi ancak isUsed boolean'ları için yanlıştır. Bu hata resetIsUsedFlag boolean'ının true olarak başlatılması ve nesnelerin isUsed boolean'larıyla resetIsUsedFlag boolean'ının mantıksal ve işlemine tabii tutulmasıyla düzeltilmiştir.

C. Player.java - Constructors

Bu sınıfın constructor'larında InitializeDeck() fonksiyonu çağırılıyordu. Aynı zamanda ComputerPlayer ve HumanPlayer sınıflarının constructor'larında da InitializeDeck() fonksiyonu çağırılıyordu. Bu oyuncuların nesne dizilerinin iki defa başlatılmasına ve oyuncuların oyuna 5 yerine 10 tane nesneyle başlamasına sebebiyet vermekteydi. Player sınıfının constructor'larındaki InitializeDeck() çağrıları yorum satırına alınarak bu hata giderilmiştir.

IV. Grup Üyelerinin Katkıları

Nesnelere ve oyunculara ait tüm sınıflar ve oyun mekaniklerinin tamamı Nusret YILDIZ tarafından yapılmıştır.

Grafiksel arayüz ve web serverla ilgili her şey Samet Ahmet Şahin tarafından yapılmıştır.

V. Sonuç

Proje dökümanında yer alan isterler doğrultusunda bir Taş Kağıt Makas oyunu başarıyla oluşturulmuştur. Bu proje sayesinde nesneye yönelik programlama konsepti uygulamalı olarak öğrenilmiş ve bu konsepte ait bilgiler daha iyi kavranmıştır. Ayrıca Godot Game Engine'de grafiksel arayüz tasarımı, Java'da web server oluşturma ve kullanmaya dair tecrübeler edinilmiştir.

REFERENCES

[1] https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/

[2] https://www.geeksforgeeks.org/abstraction-in-java-2/

[3] https://www.geeksforgeeks.org/overriding-in-java/

[4] https://www.geeksforgeeks.org/inheritance-in-java/

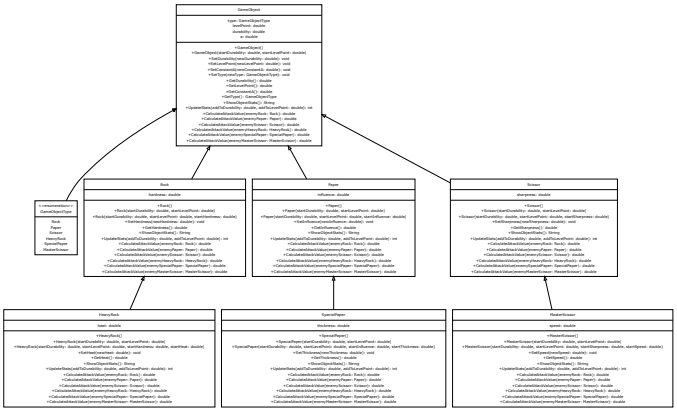
[5] https://www.w3schools.com/java/java_files_create.asp

[6] https://www.w3schools.com/java/java_date.asp

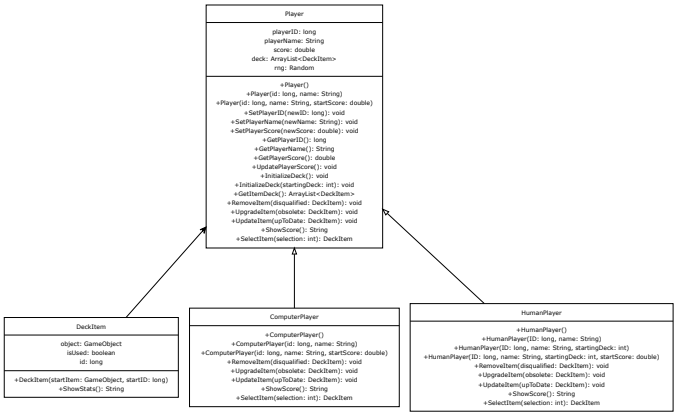
VI. UML DIYAGRAMLARI

A. Projenin Genel UML Diyagramı

B. Oyun Nesnelerinin UML Diyagramı



C. Oyuncuların UML Diyagramı



VI. UML DIYAGRAMLARI

A. Projenin Genel UML Diyagramı

B. Oyun Nesnelerinin UML Diyagramı

C. Oyuncuların UML Diyagramı