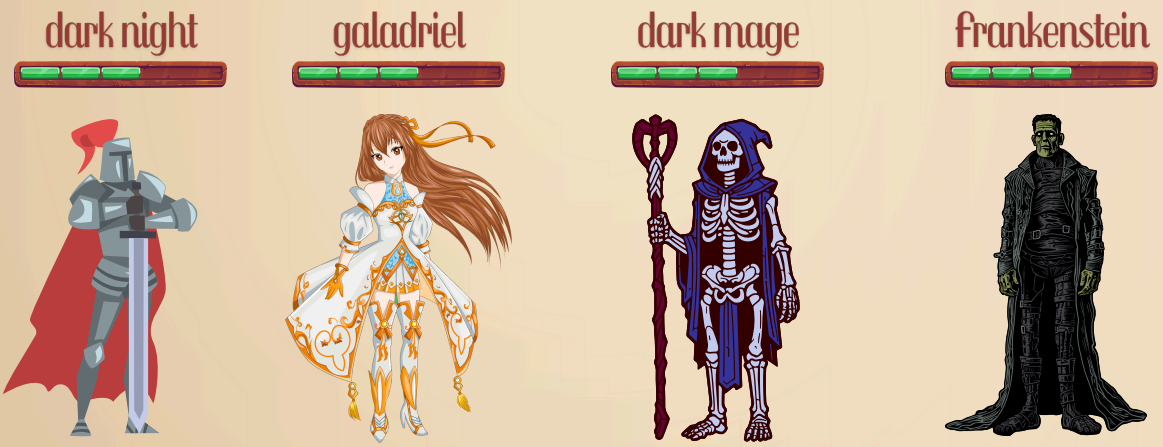
Java ile Connect 4 Oyunu Proje Raporu

4Loop Ladies

Connect4 programını GUI ile görsel hale getirmek



Renad Türkmani

Hatice Burcu

melike Beyazlı

Sümeyye Alp

İÇİNDEKİLER

İÇİNDEKİLER 2
1. Giriş 3
Proje Tanımı 3
Kullanılan Teknolojiler 3
Grup ve Görev Dağılımı 3
2. LibGDX Nedir? 4
3. Scene2D Nedir? 4
4. Oyun Tasarımı 5
4.1 Oyun Mekanikleri 5
4.2 Seviye Tasarımı 5
4.3 Grafik ve Ses Tasarımı 5
4.4 Arayüz ve Ekranlar 6
4.5 Oyun Ayarları ve Varsayılanlar 6
5. Uygulama Geliştirme 7
5.1 Geliştirme Ortamı 7
5.2 Kod Yapısı 7
6. Test ve Değerlendirme 42
Test Süreci 42
Kullanıcı Geri Bildirimi 43
Sonuçlar 43
7. Sonuç 43
8. Kaynaklar 43

1. Giriş

Proje Tanımı

Bu proje, Java programlama dersi kapsamında, popüler bir masa oyunu olan **Connect 4**'ün bilgisayar ortamında oynanabilen bir sürümünü geliştirmeyi amaçlamaktadır. Projenin temel amacı, kullanıcıların iki kişi olarak veya bilgisayara karşı oynayabileceği, görsel arayüzü ve etkileşimli yapısıyla eğlenceli bir oyun deneyimi sunmaktır. Connect 4, iki oyuncunun sırayla hamle yaparak kendi renklerindeki taşları 6 satır ve 7 sütundan oluşan bir tabloya bırakması ve yatay, dikey veya çapraz olarak dört taşı yan yana getirmeye çalıştığı strateji tabanlı bir oyundur. Projemiz, bu klasik oyunun dijitalleştirilmesiyle hem programlama becerilerimizi geliştirmeyi hem de kullanıcı dostu bir arayüz tasarlamayı hedeflemektedir.

Oyunun oynayış videosu:

https://drive.google.com/file/d/12iyVwTXtVRbh0sjO9gbaRTmQknC71RA2/view?usp=sharing

Kullanılan Teknolojiler

Projenin geliştirilmesinde, Java tabanlı açık kaynak bir oyun geliştirme çerçevesi olan **LibGDX** ve bu çerçevenin 2D oyunlar için sahne yönetimi ve kullanıcı arayüzü oluşturmayı kolaylaştıran **Scene2D** modülü kullanılmıştır. LibGDX, çoklu platform desteği ve yüksek performansıyla öne çıkarken, Scene2D ise oyun içi nesnelerin ve arayüz elemanlarının yönetimini kolaylaştırmaktadır. Bu teknolojiler sayesinde, oyunun hem masaüstü hem de mobil platformlarda çalışabilmesi ve modern bir kullanıcı deneyimi sunması hedeflenmiştir.

Grup ve Görev Dağılımı

Proje, dört kişilik bir ekip tarafından yürütülmüştür. Grup üyeleri ve görev dağılımları Tablo 1'de gösterildiği gibidir.

Tablo 1. Grup üyeleri ve görev dağılımları

İsim	Temsili Rol	Görevler
Sümeyye Alp	Lider	Oyunun geliştirilmesi ve arka planı
Melike Beyazlı	Büyücü	Arayüz tasarımı ve yazılımı
Hatice Burcu Bulduk	Son Dakika Harikalar Yaratan	Arayüz tasarımı ve yazılımı
Renad El İsmael El Türkmani	Takım Oyuncusu	Oyunun geliştirilmesi ve arka planı

Grup üyeleri, proje sürecinde iş birliği içinde çalışmış, görev paylaşımı ve iletişim konularında düzenli toplantılar yaparak süreci koordine etmiştir. Her üye, kendi sorumluluk alanında aktif rol almış ve projenin başarılı bir şekilde tamamlanmasına katkı sağlamıştır.

2. LibGDX Nedir?

LibGDX, Java tabanlı, açık kaynak kodlu bir oyun geliştirme çerçevesidir. Geliştiricilere, tek bir kod tabanı ile Android, iOS, masaüstü (Windows, Linux, MacOS) ve web (HTML5) platformlarında oyun geliştirme imkânı sunar. LibGDX'in en önemli özelliklerinden biri, çoklu platform desteği sayesinde yazılan kodun minimum değişiklikle farklı ortamlarda çalıştırılabilmesidir. Ayrıca, yüksek performans, esneklik ve genişletilebilirlik gibi avantajlarıyla hem 2D hem de 3D oyun projelerinde yaygın olarak tercih edilmektedir.

LibGDX, grafik, ses, fizik, giriş yönetimi ve dosya işlemleri gibi oyun geliştirme için gerekli olan temel bileşenleri kapsamlı bir şekilde sunar. Bu sayede geliştiriciler, düşük seviyeli detaylarla uğraşmadan oyunlarının mantığına ve tasarımına odaklanabilirler. Projemizde LibGDX'in tercih edilmesinin başlıca nedeni, Java diline olan hakimiyetimiz ve bu çerçevenin sunduğu kolaylıklar ile çoklu platform desteğidir.

3. Scene2D Nedir?

Scene2D, LibGDX çatısı altında yer alan ve özellikle 2D oyunlar için sahne grafiği ve kullanıcı arayüzü (UI) yönetimi sağlayan bir modüldür. Scene2D, oyun içi nesneleri ve arayüz elemanlarını "aktör" (actor) olarak adlandırılan nesneler şeklinde yönetir. Bu aktörler, bir "sahne" (stage) üzerinde konumlandırılır ve birbirleriyle etkileşime girebilirler.

Scene2D'nin temel bileşenleri şunlardır:

- **Aktörler (Actors):** Oyun içindeki görsel nesneler ve UI elemanlarıdır. Her aktör, konum, boyut, dönüş gibi özelliklere sahiptir ve sahne üzerinde hareket edebilir.
- **Sahne (Stage):** Tüm aktörlerin yer aldığı ve yönetildiği ana yapıdır. Kullanıcıdan gelen girişler (mouse, klavye, dokunmatik) sahneye iletilir ve ilgili aktörlere yönlendirilir.
- **Sahne Grafiği:** Aktörlerin hiyerarşik olarak düzenlenmesini ve birbirleriyle ilişkili olarak hareket etmelerini sağlar.

Scene2D, animasyonlar, butonlar, etkileşimli menüler ve diğer kullanıcı arayüzü bileşenlerinin kolayca oluşturulmasına olanak tanır. Projemizde, Connect 4 oyununun arayüzü ve kullanıcı etkileşimleri Scene2D kullanılarak tasarlanmış ve yönetilmiştir. Bu sayede, oyunun görsel bütünlüğü ve kullanıcı deneyimi üst düzeyde tutulmuştur.

4. Oyun Tasarımı

4.1 Oyun Mekanikleri

Connect 4 oyunu, iki oyuncunun sırayla hamle yaparak kendi renklerindeki taşları 6 satır ve 7 sütundan oluşan bir tabloya bırakması ve yatay, dikey veya çapraz olarak dört taşı yan yana getirmeye çalıştığı klasik bir strateji oyunudur. Oyunda, oyuncuların hamleleri sırasında taşlar, seçilen sütunun en altındaki boş alana yerleştirilir. Oyun, bir oyuncunun dört taşı yan yana getirmesiyle veya tüm alanlar dolduğunda berabere sonuçlanır. Projede, hem iki oyunculu (PvP) hem de bilgisayara karşı (PvB) oynanabilen modlar desteklenmektedir. Oyun, belirlenen tur sayısı kadar oynanır ve her turun sonunda skorlar güncellenir.

4.2 Seviye Tasarımı

Oyunda, kullanıcılar ayarlar ekranından (SettingScreen) oyun modunu (PvP veya PvB), zorluk seviyesini (easy, normal, hard), toplam tur sayısını ve oyuncu profillerini (isim, taş rengi, ikon) seçebilmektedir. PvB modunda, bilgisayarın zorluk seviyesi seçilerek farklı oyun deneyimleri sunulmaktadır. Her oyun turu sonunda, kazanan oyuncunun skoru artırılır ve toplam tur sayısı tamamlandığında genel kazanan belirlenir.

4.3 Grafik ve Ses Tasarımı

Oyun arayüzü, kullanıcı dostu ve modern bir tasarıma sahip olacak şekilde planlanmıştır. Her ekran için özel olarak hazırlanmış görseller kullanılmıştır. Örneğin, ana menü, ayarlar, oyun, sonuç ve alt menü ekranlarının her biri için ayrı arka plan ve buton görselleri tasarlanmıştır. Butonlar, basılı ve basılmamış hallerini göstermek için *_0 ve *_1 uzantılı iki farklı görsel ile desteklenmiştir. Bu sayede, kullanıcı etkileşimi sırasında görsel geri bildirim sağlanmıştır.

Taş renkleri, oyuncu ikonları ve diğer arayüz elemanları için de çeşitli PNG dosyaları kullanılmıştır. Tüm görseller, projenin **assets** klasöründe ilgili ekranlara göre kategorize edilmiştir (örneğin, **GameScreen/**, **MainMenuScreen/**, **SettingScreen/**).

Ses tasarımı için ise, https://opengameart.org/ sitesinden temin edilen ses efektleri kullanılmıştır. Taş bırakma, buton tıklama ve arka plan müziği gibi sesler, oyunun etkileşimli ve eğlenceli bir deneyim sunmasını sağlamaktadır. Ses ve müzik, ayarlar ekranından açılıp kapatılabilmektedir.

4.4 Arayüz ve Ekranlar

Oyun, Scene2D modülü kullanılarak çoklu ekran yapısında tasarlanmıştır. Her ekran için ayrı bir sınıf oluşturulmuş ve ilgili görseller ile işlevler bu sınıflarda yönetilmiştir. Ekranlar arası geçişler, kullanıcı etkileşimine göre dinamik olarak sağlanmaktadır. Örneğin:

- **MainMenuScreen:** Ana menü ekranı, oyun modları, ayarlar ve bilgi ekranına geçiş butonlarını içerir.
- **SettingScreen:** Oyun ayarlarının yapıldığı ekran. Oyuncu isimleri, taş renkleri, ikon seçimi, tur sayısı, zorluk seviyesi ve ses/müzik ayarları burada yapılır.
- **GameScreen:** Oyun tahtasının ve skorların gösterildiği ana oyun ekranı. Oyuncu hamleleri, taşların yerleştirilmesi ve turun ilerleyişi burada yönetilir.
- **SubMenuScreen:** Oyun sırasında açılan alt menü ekranı. Oyuna devam etme, tekrar başlatma veya ana menüye dönme seçenekleri sunar.
- **ResultScreen:** Oyun sonunda kazananı veya beraberliği gösteren ekran.
- **InfoScreen:** Oyun kurallarının ve oynanışın anlatıldığı bilgi ekranı.

Her ekran, ilgili görselleri ve sesleri kullanarak kullanıcıya zengin bir görsel ve işitsel deneyim sunar. Arayüzde, kullanıcı etkileşimini kolaylaştırmak için butonlar ve seçim kutuları gibi bileşenler özenle yerleştirilmiştir.

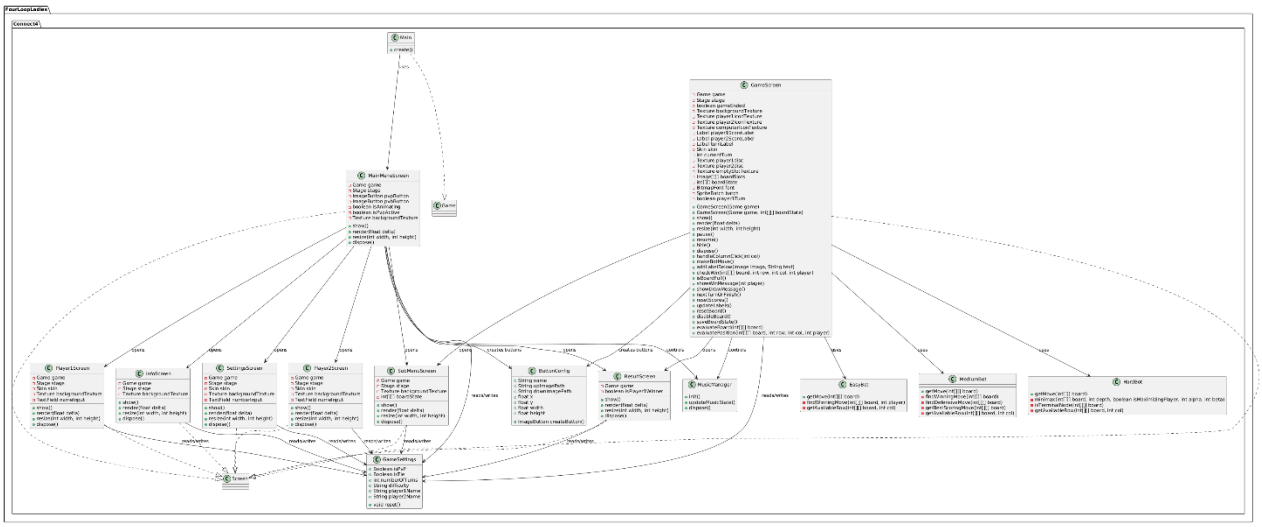
4.5 Oyun Ayarları ve Varsayılanlar

Oyun ayarları, merkezi bir yapı olan **GameSetting** sınıfında saklanmaktadır. Bu sınıf, oyun modunu, oyuncu isimlerini, taş renklerini, ikonları, skorları ve ses/müzik durumunu yönetir. Varsayılan değerler, kullanıcı herhangi bir seçim yapmadığında devreye girer ve oyunun sorunsuz başlamasını sağlar.Bu yapı sayesinde, oyun boyunca ayarlar ve skorlar merkezi olarak yönetilmekte ve ekranlar arası geçişlerde tutarlılık sağlanmaktadır.

5. Uygulama Geliştirme

5.1 Geliştirme Ortamı

Proje, Java programlama dili ile geliştirilmiş olup, LibGDX oyun motoru ve Scene2D arayüz kütüphanesi kullanılmıştır. Geliştirme ortamı olarak IntelliJ IDEA tercih edilmiş, proje yönetimi ve bağımlılıkların yönetimi için Gradle kullanılmıştır. Tüm görsel ve ses dosyaları, LibGDX'in **assets** klasöründe organize edilmiştir.



Şekil1. Genel UML diyagramı

5.2 Kod Yapısı

Proje, temel olarak ekranlar (Screen), oyun mantığı (Game Logic), ayarlar (Settings), yardımcı sınıflar (Utils) ve varlıklar (Assets) olmak üzere modüler bir yapıda tasarlanmıştır. Her ekran için ayrı bir sınıf oluşturulmuş ve LibGDX'in Scene2D kütüphanesinin Stage ve Actor yapısı kullanılarak arayüz elemanları yönetilmiştir. Bu yapı, oyunun farklı bölümlerinin (ana menü, ayarlar, oyun ekranı vb.) bağımsız olarak geliştirilmesine ve yönetilmesine olanak tanır.

```
public class SettingsScreen implements Screen {
    private final Game game;
    private Stage stage;
    private Skin skin;
    private Texture backgroundTexture;
    private TextField numberInput;
```

```
    private final String[] difficulties = {"easy", "normal", "hard"};
    private final float[] positionsY = {545.5f, 699.1f, 852.7f};

    public SettingsScreen(Game game) {
        this.game = game;
    }
.
.
.
```

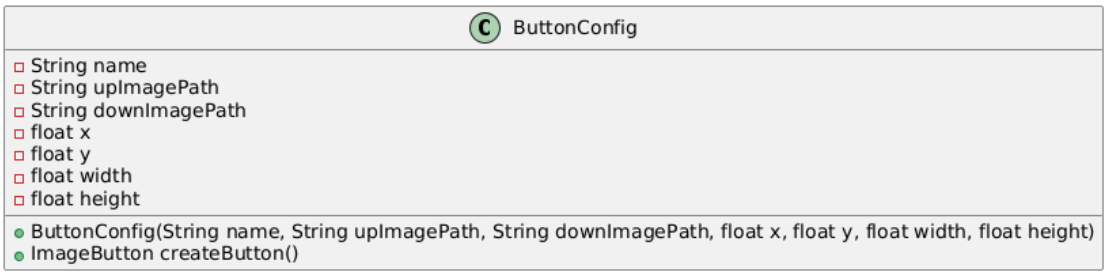
Oyun mantığı, **Game** sınıfı tarafından yönetilir. **Game** sınıfı, oyunun genel durumunu ve akışını kontrol eden ana sınıftır. Bu sınıf, farklı ekranlar arasında geçiş yapmayı sağlar ve oyunun başlangıç noktasıdır. **Game** sınıfı, oyunun başlatılması, ekranların yüklenmesi ve yönetilmesi gibi temel işlevleri üstlenir. Her ekran, **Screen** arayüzünü implement ederek, oyunun belirli bir durumunu temsil eder. Bu arayüz, ekranın yaşam döngüsünü yönetmek için gerekli olan **show**, **render**, **resize**, **pause**, **resume**, **hide** ve **dispose** gibi metodları içerir.

Her ekran, **Screen** arayüzünü implement ederek, oyunun belirli bir durumunu temsil eder. Bu arayüz, ekranın yaşam döngüsünü yönetmek için gerekli olan **show**, **render**, **resize**, **pause**, **resume**, **hide** ve **dispose** gibi metodları içerir. **show** metodu, ekran ilk kez gösterildiğinde çağrılır ve burada sahne oluşturulur, gerekli kaynaklar yüklenir ve kullanıcı arayüzü bileşenleri sahneye eklenir. **render** metodu, her çerçevede ekranın güncellenmesi ve çizilmesi için kullanılır; bu, kullanıcı etkileşimlerinin ve animasyonların akıcı bir şekilde gerçekleşmesini sağlar. **resize** metodu, ekran boyutu değiştiğinde sahnenin yeniden boyutlandırılmasını sağlar. **pause** ve **resume** metodları, oyunun duraklatılması ve devam ettirilmesi durumlarında gerekli işlemleri yönetir. **hide** metodu, ekran gizlendiğinde yapılması gereken işlemleri tanımlar. Son olarak, **dispose** metodu, ekran kapatıldığında veya kaynakların serbest bırakılması gerektiğinde çağrılır ve bellek yönetimi için önemlidir.

Bu modüler yapı, her ekranın kendi bileşenlerini ve işlevselliğini bağımsız olarak yönetmesine olanak tanırken, aynı zamanda oyunun genel akışını ve kullanıcı deneyimini de iyileştirir. Örneğin, ayarlar ekranında kullanıcı arayüzü bileşenleri (butonlar, metin alanları vb.) **Stage** üzerinde düzenlenirken, her bir bileşenin etkileşimleri ve görsel stilleri **Actor** sınıfı aracılığıyla yönetilir. Bu sayede, kullanıcıların oyun ayarlarını değiştirmesi, oyun modları arasında geçiş yapması ve genel olarak oyunun akışını kontrol etmesi kolaylaşır.

Buton Yönetimi: ButtonConfig Sınıfı

Arayüzde kullanılan butonların oluşturulması ve yönetilmesi için, tekrar eden kodları azaltmak ve tutarlılığı sağlamak amacıyla **ButtonConfig** adında yardımcı bir sınıf geliştirilmiştir. Bu sınıf, butonun ismini, görsel yollarını, konumunu ve boyutunu parametre olarak alır ve Scene2D'nin **ImageButton** nesnesini kolayca oluşturur. Sınıfta kullanılan başlıca yapılar ve işlevleri şunlardır:



Şekil 2. ButtonConfig UML yapısı

- **Texture**

LibGDX'in temel görsel (resim) veri yapısıdır. PNG, JPG gibi dosyalardan yüklenen ve ekranda gösterilebilen grafik verisini temsil eder. Butonun basılı (**downTex**) ve basılmamış (**upTex**) hallerini göstermek için iki farklı Texture nesnesi kullanılır.

```
Texture upTex = new Texture(Gdx.files.internal(upImagePath));
```

- **TextureRegion ve TextureRegionDrawable**

TextureRegion, Bir Texture'un tamamını veya bir bölümünü temsil eden yapıdır. Genellikle sprite sheet'lerde kullanılır. **TextureRegionDrawable ise** Scene2D arayüzünde, bir aktörün (örneğin butonun) görselini tanımlamak için kullanılır. TextureRegion'u Drawable arayüzüne dönüştürür.

```
style.imageUp = new TextureRegionDrawable(new TextureRegion(upTex));
```

Butonun farklı durumları için (basılı/basılmamış) uygun görseller atanır.

- **ImageButton ve ImageButton(style)**

ImageButton, Scene2D'nin hazır buton sınıfıdır. Görsel tabanlı butonlar oluşturmak için kullanılır.**ImageButton(style)** ise butonun farklı durumlarındaki (örneğin, normal, basılı) görsellerini ve stilini tanımlar.

```
ImageButton button = new ImageButton(style);
```

Oluşturulan TextureRegionDrawable nesneleri, butonun stiline atanır ve butonun ekranda nasıl görüneceği belirlenir.

- **Gdx.files.internal**

LibGDX'in dosya sistemi yönetimidir. Projenin **assets** klasöründen dosya yüklemek için kullanılır.

```
Texture downTex = new Texture(Gdx.files.internal(downImagePath));
```

Buton görsellerinin dosya yolunu belirtmek ve Texture nesnelerini oluşturmak için kullanılır.

Şekil 3. setting_button_0.png ve setting_button_1.png görselleri

- **Sahne Koordinatları ve Boyutlandırma**

setSize, setPosition, setOrigin; Scene2D'de aktörlerin (butonların) ekrandaki boyutunu, konumunu ve dönüş/ölçekleme merkezini ayarlamak için kullanılır. Butonun ekranda doğru yerde ve doğru boyutta görünmesi sağlanır.
Canva ve Java (LibGDX) arasındaki koordinat sistemleri farklılık gösterir. Canva'da y ekseni yukarıdan aşağıya doğru artarken, Java'da y ekseni aşağıdan yukarıya doğru artar. Bu nedenle, Java'da bir nesnenin y koordinatını belirlerken, Canva'daki y koordinatından arka plan yüksekliği ve nesnenin yüksekliği çıkarılarak hesaplanır:

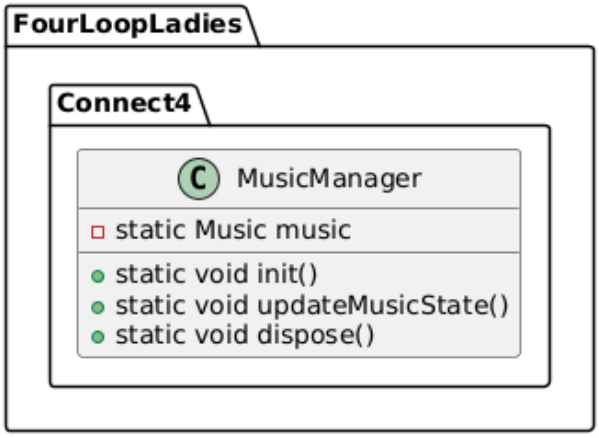
```
this.x = x;
this.y = 1080 - (y + height);
this.width = width;
this.height = height;

button.setSize(width, height);
button.setPosition(x, y);
button.setOrigin(button.getWidth() / 2, button.getHeight() / 2);
```

Bu sınıfın kullanılması ise şeklinde olmaktadır.

```
ButtonConfig settingsButtonConfig = new ButtonConfig(
                                                    "settings",
                                                    "MainMenuScreen/setting_button_1.png",
                                                    "MainMenuScreen/setting_button_0.png",
                                                    683.7f, 53f, 66.9f, 66.2f);
ImageButton settingsButton = settingsButtonConfig.createButton();
stage.addActor(settingsButton);
```

Müzik Yönetimi: MusicManager Sınıfı

Müzik nesnesi, oyun içinde yalnızca bir kez oluşturulan ve tekrar yüklenmeyen bir singleton yapısına sahiptir. Oyun başladığında veya müziğe ilk kez ihtiyaç duyulduğunda, ilgili müzik dosyası yüklenir ve döngüsel olarak çalınmaya başlar; bu süreç, müzik ayarlarının açık olması durumunda otomatik olarak gerçekleşir. Kullanıcı, ayarlarından müziği açıp kapattığında, müzik durumu güncellenir ve müzik ya oynatılır ya da duraklatılır. Oyun sona erdiğinde veya müziğe artık ihtiyaç kalmadığında, müzik nesnesi uygun bir şekilde serbest bırakılarak kaynak yönetimi sağlanır. Bu yapı, hem kullanıcı deneyimini iyileştirir hem de sistem kaynaklarının verimli bir şekilde kullanılmasını garanti eder. Sınıfın ana bileşenleri ve işlevleri şu şekildedir:



Şekil 4. MusicManager UML diyagramı

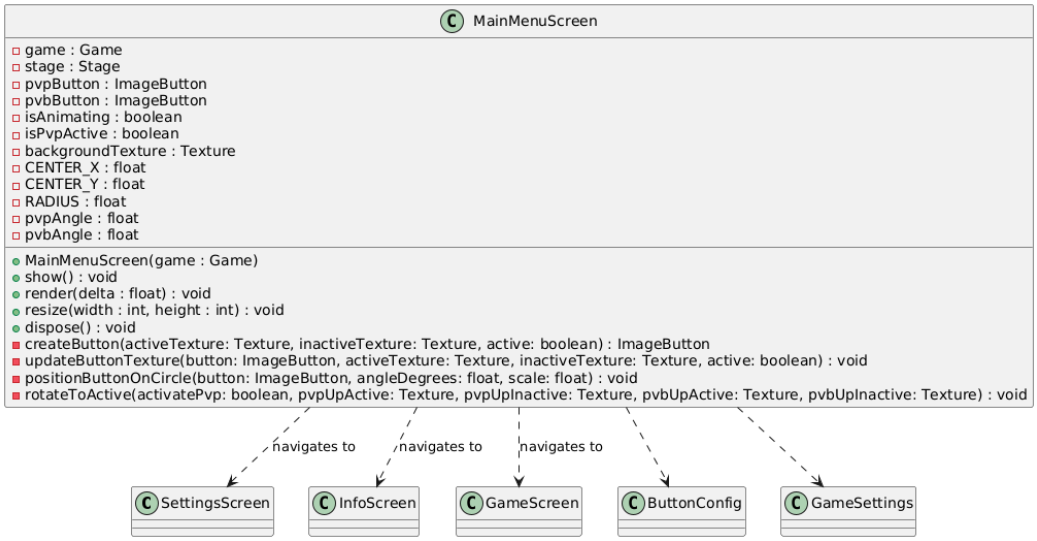
Sınıf, **init** metodu ile müzik dosyasını ilk kez yükler ve döngüsel olarak çalacak şekilde ayarlar; eğer oyun ayarlarında müzik açıksa, otomatik olarak çalmaya başlar. **updateMusicState** metodu, kullanıcı ayarlarından müziğin açılıp kapatılmasına göre müzik durumunu güncelleyerek, müzik açık olduğunda çalmıyorsa başlatır, kapalı olduğunda ise duraklatır. Oyun kapanırken veya müzik nesnesine ihtiyaç kalmadığında, **dispose** metodu çağrılarak müzik nesnesi uygun bir şekilde serbest bırakılır.

```
public class MusicManager {
  private static Music music;

  public static void init() {
    if (music == null) {
      music = Gdx.audio.newMusic(Gdx.files.internal("voices/forestismagic.mp3"));
      music.setLooping(true);
      if (GameSettings.isMusicOn) music.play();
    }
  }
  public static void updateMusicState() {
    if (music == null) return;
    if (GameSettings.isMusicOn) {
      if (!music.isPlaying()) music.play();
    } else {
      if (music.isPlaying()) music.pause(); // veya stop()
    }
  }
  public static void dispose() {
    if (music != null) music.dispose();
  }
}
```

Ana Menü Ekranı (MainMenuScreen.java)

Şekil 5. Ana menü ekranı arayüzü

Projenin ana menü ekranı Scene2D altyapısı ile tasarlanmış ve kullanıcı dostu bir arayüz sunacak şekilde yapılandırılmıştır. Ana menüde, oyun modları (PvP ve PvB), ayarlar, bilgi ve oyuna başla butonları yer almaktadır. Butonlar, görsel olarak aktif ve pasif durumlarını yansıtan farklı PNG dosyaları ile desteklenmiştir. Ayrıca, PvP ve PvB butonları dairesel bir düzende yer almakta ve seçime göre animasyonlu olarak öne gelmektedir. Sınıfın ana bileşenleri ve işlevleri şu şekildedir:



Şekil 6. MainMenuScreen UML diyagramı

positionButtonOnCircle metodu, belirtilen butonu, merkez noktası ve yarıçap kullanılarak hesaplanan dairesel bir yörünge üzerinde konumlandırır. Verilen açı derece cinsinden radyana çevrilir ve trigonometrik fonksiyonlarla butonun sahnedeki x ve y koordinatları belirlenir. Butonun genişliği ve yüksekliği göz önünde bulundurularak pozisyonu ayarlanır, ayrıca buton ölçeklendirilir ve dönüş noktası butonun merkezine ayarlanır; böylece animasyon ve dönüş hareketleri daha doğal olur.

```
private final float CENTER_X = 850 / 2f ;
private final float CENTER_Y = 821.7f;
private final float RADIUS = 60f;
private void positionButtonOnCircle(ImageButton button, float angleDegrees, float scale) {
   float rad = (float) Math.toRadians(angleDegrees);
   float x = CENTER_X + RADIUS * (float)Math.cos(rad) - button.getWidth() / 2;
   float y = 1080 - CENTER_Y + RADIUS * (float)Math.sin(rad) - button.getHeight();
   button.setPosition(x, y);
   button.setScale(scale);
   button.setOrigin(button.getWidth() / 2, button.getHeight() / 2);
}
```

rotateToActive metodu, PvP ve PvB modlarına ait butonların aktif durum değişimini animasyonlu şekilde gerçekleştirir. Bu metot, aktif olacak modu belirler ve her iki butonun hedef açılarını ve pozisyonlarını hesaplayarak belirlenen sürede butonları yeni konumlarına hareket ettirir. Aktif buton büyüyüp ön plana çıkarılırken (ölçek 1.2), pasif buton küçültülür (ölçek 0.75) ve arka plana alınır. Butonların görünümleri, aktif veya pasif durumlarına göre güncellenir. Animasyon tamamlandığında, **isAnimating** bayrağı false yapılarak yeni animasyonların başlatılmasına izin verilir.

```
private void rotateToActive(final boolean activatePvp,final Texture pvpUpActive, final Texture pvpUpInactive,final
Texture pvbUpActive, final Texture pvbUpInactive) {
  isAnimating = true;
  float targetPvpAngle = activatePvp ? 270f : 90f;
  float targetPvbAngle = activatePvp ? 90f : 270f;
  float duration = 0.4f;
  pvpButton.addAction(Actions.sequence(
    Actions.moveTo(
      CENTER_X + RADIUS * (float)Math.cos(Math.toRadians(targetPvpAngle)) - pvpButton.getWidth() / 2,
      1080 - CENTER_Y + RADIUS * (float)Math.sin(Math.toRadians(targetPvpAngle)) - pvpButton.getHeight(),
      duration
    ),
    Actions.run(() -> pvpAngle = targetPvpAngle)
  ));
  pvbButton.addAction(Actions.sequence(
    Actions.moveTo(
      CENTER_X + RADIUS * (float)Math.cos(Math.toRadians(targetPvbAngle)) - pvbButton.getWidth() / 2,
      1080 - CENTER_Y + RADIUS * (float)Math.sin(Math.toRadians(targetPvbAngle)) - pvbButton.getHeight(),
      duration
    ),
    Actions.run(() -> pvbAngle = targetPvbAngle)
  ));
  if (activatePvp) {
    pvpButton.addAction(Actions.scaleTo(1.2f, 1.2f, duration));
    pvbButton.addAction(Actions.scaleTo(0.75f, 0.75f, duration));
    pvpButton.toFront();
  } else {
    pvbButton.addAction(Actions.scaleTo(1.2f, 1.2f, duration));
    pvpButton.addAction(Actions.scaleTo(0.75f, 0.75f, duration));
    pvbButton.toFront();
  }
  updateButtonTexture(pvpButton, pvpUpActive, pvpUpInactive, activatePvp);
  updateButtonTexture(pvbButton, pvbUpActive, pvbUpInactive, !activatePvp);
  isPvpActive = activatePvp;
  stage.addAction(Actions.sequence(
    Actions.delay(duration),
    Actions.run(() -> isAnimating = false)
  ));
```

PvP ve PvB butonlarına tıklanıldığında önce tıklama sesi (default isVoiceOn:true) çalınır, ardından animasyon devam ediyorsa veya seçili mod zaten aktifse tıklama işleme alınmaz. Aksi durumda **rotateToActive** metodu çağrılarak ilgili mod aktif hâle getirilir ve **GameSettings.isPvP** değişkeni bu moda uygun olarak güncellenir. Böylece oyun moduna dair ayar merkezi olarak **GameSettings** sınıfında tutulur ve diğer bileşenler bu ayara göre davranabilir.

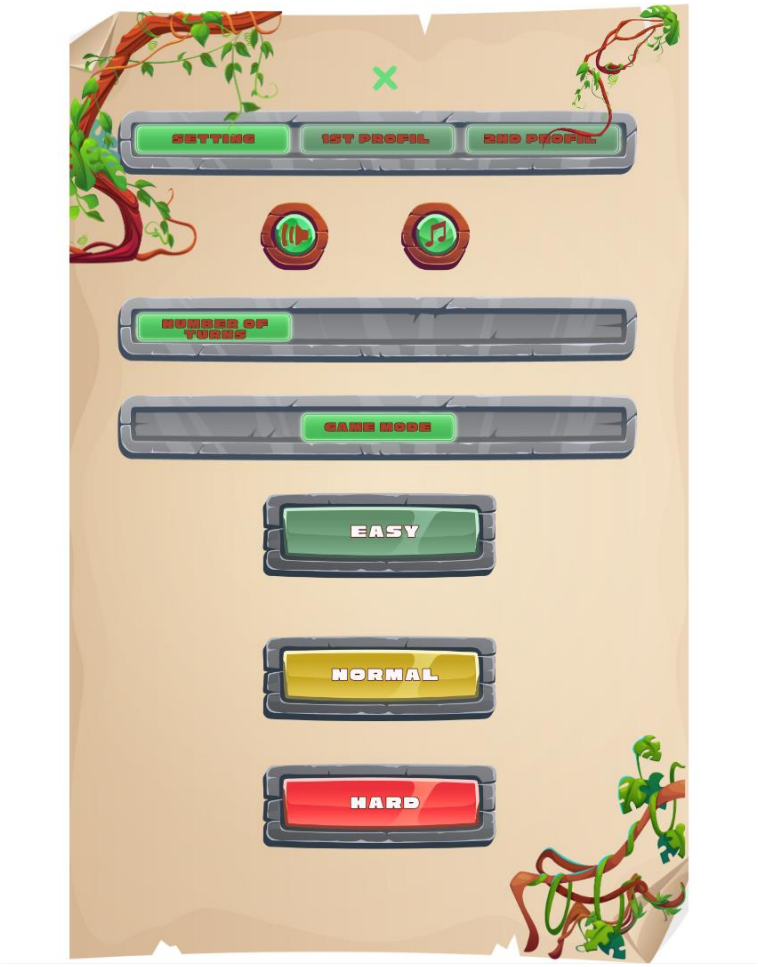
```
pvpButton.addListener(new ClickListener() {
  @Override
  public void clicked(InputEvent event, float x, float y) {
    if (GameSettings.isSoundOn) GameSettings.clickSound.play();
    if (isAnimating || isPvpActive) return;
    rotateToActive(true, pvpUpActive, pvpUpInactive, pvbUpActive, pvbUpInactive);
    GameSettings.isPvP = true; // PvP modunu aktif et
  }
});
```

Diğer menü butonlarında ise; "play" butonuna tıklandığında oyun ekranına geçiş yapılır (**GameScreen**), "settings" butonuna basıldığında ayar ekranı (**SettingsScreen**) açılır ve bu esnada **GameSettings.isPvP** güncel duruma göre set edilir, böylece ayarlar ekranı doğru mod bilgisiyle başlatılır. "info" butonuna tıklanınca ise oyun hakkında bilgi veren ekran (**InfoScreen**) gösterilir. Tüm bu butonlar da tıklama durumunda varsa ses efektini çalarak kullanıcının etkileşimini destekler.

```
infoButton.addListener(new ClickListener() {
  @Override
  public void clicked(InputEvent event, float x, float y) {
    if (GameSettings.isSoundOn) GameSettings.clickSound.play();
    game.setScreen(new InfoScreen(game));
  }
});
```

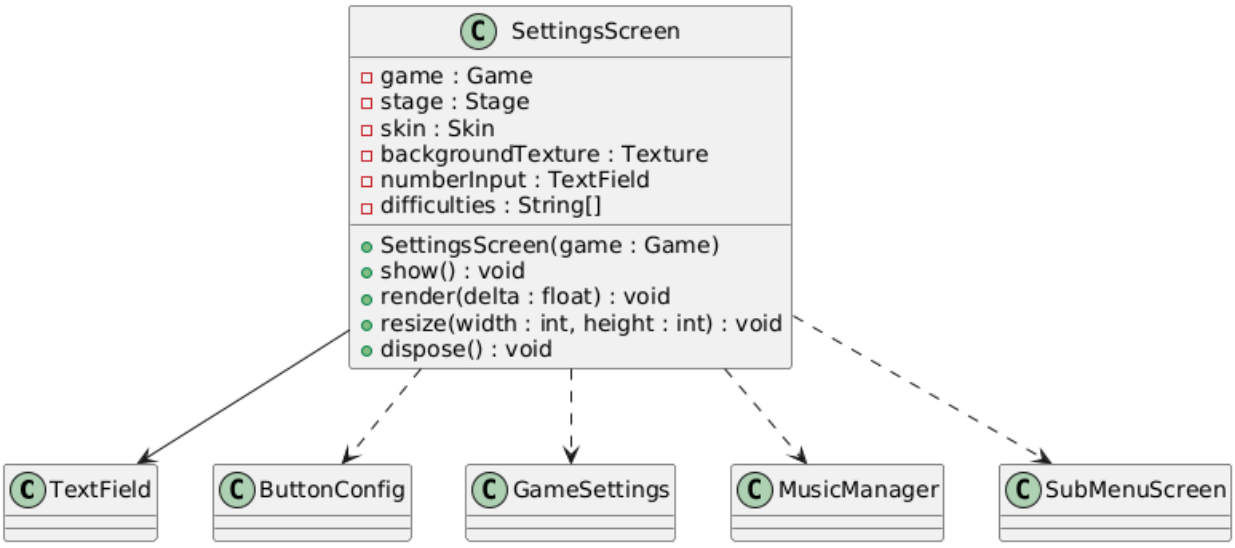
Butonlara tıklama olayları ve animasyonlar sayesinde kullanıcı deneyimi üst düzeyde tutulmuştur. Scene2D'nin sunduğu event ve animasyon yönetimi, kodun okunabilirliğini ve sürdürülebilirliğini artırmaktadır. Pokémonların aktif/pasif durumları ve animasyonlu geçişleri, kullanıcıya hangi modun seçili olduğunu açıkça göstermektedir.

Ayarlar Ekranı (SettingsScreen.java)



Şekil 7. Ayarlar ekranı arayüzü

Ayarlar ekranı, oyunun kişiselleştirilebilir parametrelerinin kullanıcı tarafından kolayca düzenlenebilmesi için Scene2D altyapısı ile tasarlanmıştır. Bu ekranda, oyun tur sayısı, zorluk seviyesi, ses ve müzik ayarları ile oyuncu profili seçimleri yapılabilmektedir. Tüm arayüz elemanları, görsel bütünlük ve kullanıcı deneyimi gözetilerek konumlandırılmıştır. Sınıfın ana bileşenleri ve işlevleri şu şekildedir:



Şekil 8. SettingScreen sınıf UML diyagramı

Sınıf, sahne oluşturma, arka plan resmi yükleme ve kullanıcı arayüzü bileşenlerinin stilini tanımlama işlemleri ile başlar. **TextField** bileşeni, kullanıcıdan dönüş sayısını almak için kullanılır ve bu alanın stili ayarlanır. Kullanıcı, 1 ile 20 arasında bir sayı girmelidir; bu sayı, **GameSettings.numberOfTurns** değişkenine atanır. Kullanıcı geçersiz bir giriş yaparsa, varsayılan değer 3 olarak ayarlanır.

```
skin = new Skin(Gdx.files.internal("uiskin.json"));
BitmapFont font = new BitmapFont();
TextField.TextFieldStyle textFieldStyle = new TextField.TextFieldStyle();
textFieldStyle.font = font;
textFieldStyle.font.getData().setScale(2f);
textFieldStyle.fontColor = Color.WHITE;
textFieldStyle.cursor = skin.getDrawable("cursor");
textFieldStyle.messageFont = font;
textFieldStyle.messageFontColor = Color.WHITE;
numberInput = new TextField(Integer.toString(GameSettings.numberOfTurns), textFieldStyle);
numberInput.setMessageText("Turns (1-20)");
numberInput.setPosition(331.8f, 1080 - 344.5f - 35.9f);
numberInput.setSize(369.6f, 35.9f);
numberInput.setTextFieldFilter(new TextField.TextFieldFilter.DigitsOnlyFilter());
numberInput.setAlignment(Align.center);
numberInput.addListener(new ChangeListener() {
  @Override
  public void changed(ChangeEvent event, Actor actor) {
    String text = numberInput.getText();
    if (text.isEmpty()) return;
    try {
      int val = Integer.parseInt(text);
      if (val < 1) val = 1;
      else if (val > 20) val = 20;
      numberInput.setText(Integer.toString(val));
```

```
            GameSettings.numberOfTurns = val;
        } catch (NumberFormatException e) {
            numberInput.setText("3");
            GameSettings.numberOfTurns = 3;
        }
    }
});
```

Ekranda ayrıca çeşitli butonlar bulunmaktadır. "exit" butonuna tıklandığında, kullanıcı ana menüye (**MainMenuScreen**) geri döner. "setting" butonu, ayarlar ekranını yeniden açar. "player1" ve "player2" butonları, sırasıyla birinci ve ikinci oyuncu ekranlarına geçiş yapar; ancak ikinci oyuncu butonu yalnızca PvP modunda aktif hale gelir.

```
player2Button.addListener(new ClickListener() {
    public void clicked(InputEvent event, float x, float y) {
        if (GameSettings.isSoundOn) GameSettings.clickSound.play();
        if (!GameSettings.isPvP) return;
        game.setScreen(new Player2Screen(game));
    }
});
```

"openVoice" ve "musicOn" butonları, ses ve müzik ayarlarını açıp kapatmak için kullanılır; bu butonlara tıklandığında, ilgili ayar güncellenir ve ses efektleri çalınır.

```
openVoiceButton.addListener(new ClickListener() {
    public void clicked(InputEvent event, float x, float y) {
        GameSettings.isSoundOn = GameSettings.isSoundOn? false: true;
        if (GameSettings.isSoundOn) GameSettings.clickSound.play();
        SubMenuScreen.updateVoiceButtons(openVoiceButton,"voice");
    }
});
```

Ayrıca, zorluk seviyelerini temsil eden butonlar da bulunmaktadır. Bu butonlar, "easy", "normal" ve "hard" seçeneklerini içerir ve kullanıcı bir zorluk seviyesi seçtiğinde, **GameSettings.difficulty** değişkeni güncellenir. **updateModeButtons** metodu, seçilen zorluk seviyesine göre butonların görünümlerini güncelleyerek kullanıcıya hangi seviyenin aktif olduğunu gösterir.

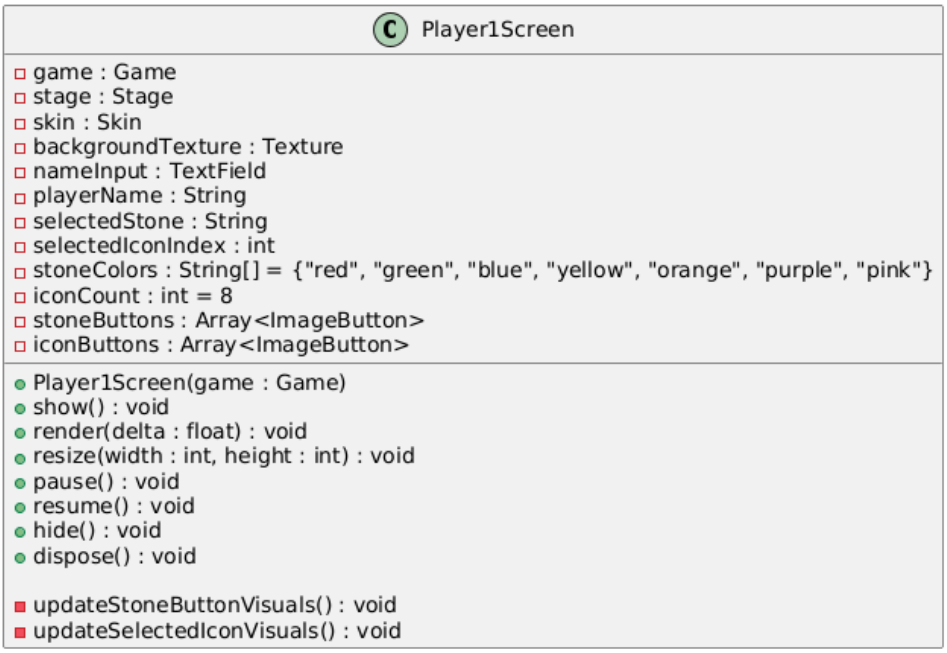
```
ImageButton[] difficultyButtons = new ImageButton[difficulties.length];
for (int i = 0; i < difficulties.length; i++) {
    final String level = difficulties[i];
    final float posY = positionsY[i];
    ButtonConfig config = new ButtonConfig(
        level,
        "SettingScreen/" + level + (level.equals(GameSettings.difficulty) ? "_button_1.png" : "_button_0.png"),
        "SettingScreen/" + level + (level.equals(GameSettings.difficulty) ? "_button_0.png" : "_button_1.png"),
        294.3f, posY, 262.7f, 95.7f
    );
```

```
    final ImageButton button = config.createButton();
    difficultyButtons[i] = button;
    stage.addActor(button);
    button.addListener(new ClickListener() {
        public void clicked(InputEvent event, float x, float y) {
            if (GameSettings.isSoundOn) GameSettings.clickSound.play();
            GameSettings.difficulty = level;
            updateModeButtons(difficultyButtons, level);
        }
    });
}
private void updateModeButtons(ImageButton[] buttons, String selectedLevel) {
    for (int i = 0; i < buttons.length; i++) {
        String level = difficulties[i];
        String suffix = level.equals(selectedLevel) ? "_button_1.png" : "_button_0.png";
        buttons[i].getStyle().imageUp = new TextureRegionDrawable(new TextureRegion(
            new Texture(Gdx.files.internal("SettingScreen/" + level + suffix))
        ));
    }
}
```

Oyuncu Profili Ayar Ekranları: Player1Screen ve Player2Screen

Şekil 9. Oyuncu Profili Ayar Ekranları

Oyuncu profili ayar ekranları, kullanıcıların oyun başlamadan önce kendi isimlerini, taş renklerini ve profil ikonlarını seçebilmeleri için Scene2D altyapısı ile tasarlanmıştır; bu ekranlar (Player1Screen ve Player2Screen) benzer mantıkla çalışmakta olup yalnızca varsayılan değerler ve seçilebilecek taş rengi/ikonlar farklılık göstermektedir. Ekran açıldığında, arka plan görseli ve sahne (Stage) oluşturulur, kullanıcıdan oyuncu ismini almak için bir **TextField** kullanılır ve girilmezse varsayılan isim atanır (örneğin, "Player 1"). Farklı renklerde taşlar için butonlar eklenmiş olup, seçilen taş rengi görsel olarak vurgulanır ve diğer oyuncunun seçtiği taş rengiyle aynı olamaz. Ayrıca, 8 farklı profil ikonu arasından seçim yapılabilir; seçilen ikon görsel olarak bulanıklaştırılır ve diğer oyuncunun seçtiği ikonla aynı olamaz. Çıkış, ayarlar ve oyuncu profili ekranları arasında geçiş için butonlar eklenmiştir. Tüm seçimler anında **GameSettings** üzerinden güncellenir ve diğer ekranlarda kullanılmak üzere saklanır, böylece kullanıcı etkileşimi sorunsuz bir şekilde sağlanır.



Şekil 10. Player1Screen UML diyagramı

```
public class Player1Screen implements Screen {
 private final Game game;
 private Stage stage;
 private Skin skin;
 private Texture backgroundTexture;
 private TextField nameInput;
 private String playerName;
 private String selectedStone;
 private int selectedIconIndex;
 private final String[] stoneColors = {"red", "green", "blue", "yellow", "orange", "purple", "pink"};
 private final int iconCount = 8;
 private Array<ImageButton> stoneButtons;
```

```
 private Array<ImageButton> iconButtons;
 public Player1Screen(Game game) {
 this.game = game;
}
```

Player1Screen sınıfı, **Screen** arayüzünü implement eder ve bir **Game** nesnesi alarak başlatılır. Bu, ekranın oyunla etkileşimde bulunmasını sağlar. Sınıf, sahne (**Stage**), görsel stiller (**Skin**), arka plan resmi (**Texture**), oyuncu adı girişi (**TextField**), seçilen taş rengi ve simge indeksi gibi değişkenleri tutar. Ayrıca, taş renkleri ve simgeleri için diziler tanımlanmıştır.

```
skin = new Skin(Gdx.files.internal("uiskin.json"));
BitmapFont font = new BitmapFont();
TextFieldStyle textFieldStyle = new TextFieldStyle();
textFieldStyle.font = font;
textFieldStyle.font.getData().setScale(2f);
textFieldStyle.fontColor = Color.WHITE;
textFieldStyle.cursor = skin.getDrawable("cursor");
textFieldStyle.messageFont = font;
textFieldStyle.messageFontColor = Color.LIGHT_GRAY;
nameInput = new TextField(playerName, textFieldStyle);2nameInput.setMessageText("Player 1 Name");
nameInput.setPosition(323.7f, 1080f - 39.3f - 257.1f);
nameInput.setSize(365.8f, 39.3f);
nameInput.setAlignment(Align.center);
stage.addActor(nameInput);
```

Kullanıcıdan oyuncu adını almak için bir **TextField** oluşturulur. **TextFieldStyle** ile metin alanının görünümü ayarlanır. Kullanıcı adı girişi, sahneye eklenir ve başlangıçta **playerName** ile doldurulur.

```
stoneButtons = new Array<>();
iconButtons = new Array<>();
for (int i = 0; i < stoneColors.length; i++) {
 String color = stoneColors[i];
 final String buttonPath = "SettingScreen/" + color + "_stone_button";
 ButtonConfig buttonConfig = new ButtonConfig(color + "_button", buttonPath + "_1.png", buttonPath + "_0.png",
338 + i * 50, 346.5f, 42.9f, 45f);
 ImageButton button = buttonConfig.createButton();
 final String selectedColor = color;
 button.addListener(new ClickListener() {
 public void clicked(InputEvent event, float x, float y) {
 if (GameSettings.isSoundOn) GameSettings.clickSound.play();
 selectedStone = selectedColor;
 if(selectedColor != GameSettings.player2StoneColor) {
 GameSettings.player1StoneColor = selectedColor;
 updateStoneButtonVisuals();
  }
 }
```

```
});
 stoneButtons.add(button);
 stage.addActor(button);
}
```

Taş renkleri için butonlar oluşturulur. Her bir renk için bir **ImageButton** oluşturulur ve sahneye eklenir. Butonlara tıklanıldığında, ses çalınır ve seçilen taş rengi güncellenir. Ayrıca, oyuncu 2'nin taş rengiyle çakışmaması durumunda, **GameSettings** güncellenir. Profil simgeleri için de benzer bir yapı ile butonlar oluşturulur. Her simge için bir **ImageButton** oluşturulur ve sahneye eklenir. Tıklama olayında, ses çalınır ve seçilen simge güncellenir.

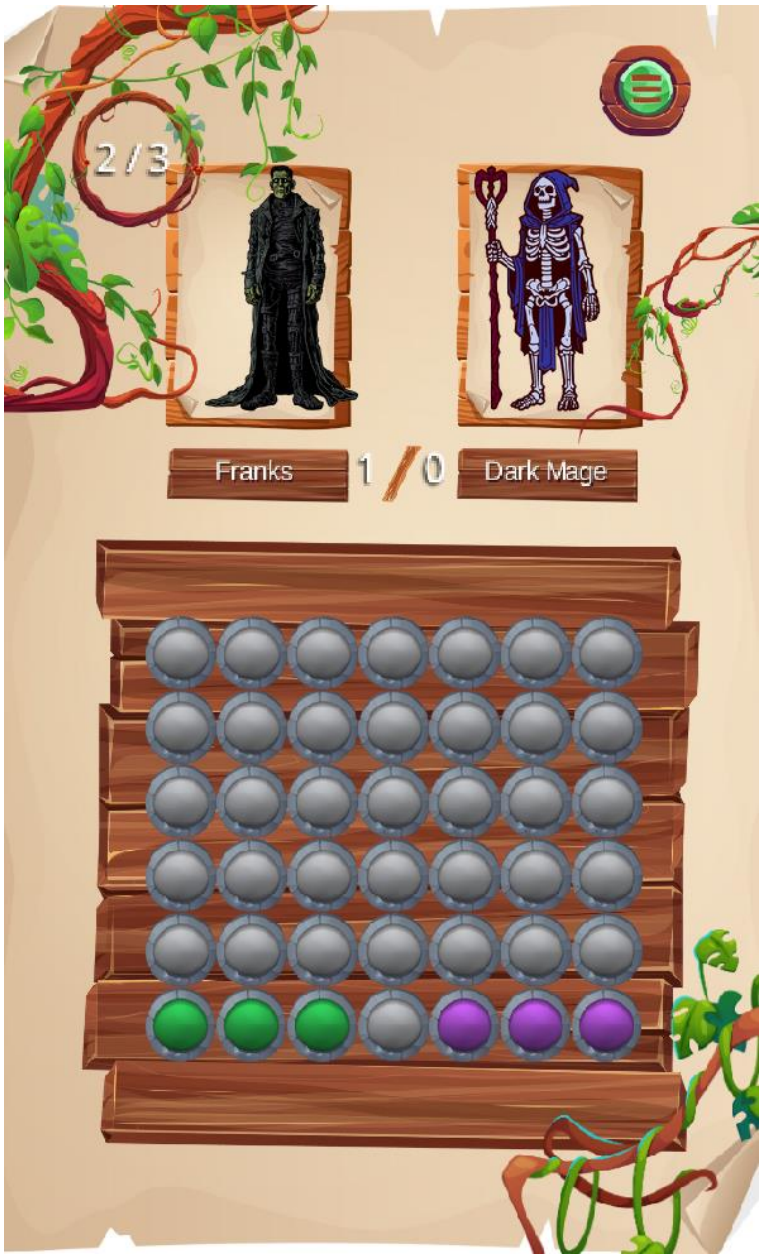
```
private void updateStoneButtonVisuals() {
 for (int i = 0; i < stoneButtons.size; i++) {
 ImageButton button = stoneButtons.get(i);
 String color = stoneColors[i];
 String suffix = color.equals(selectedStone) ? "_stone_button_1.png" : "_stone_button_0.png";
button.getStyle().imageUp = new TextureRegionDrawable(new  TextureRegion(new
Texture(Gdx.files.internal("SettingScreen/" + color + suffix))));
 }
}
```

updateStoneButtonVisuals metodu, taş butonlarının görsel durumunu günceller. Seçilen taş rengine göre butonların görseli değiştirilir.

Kaynak Yönetimi:
Ekran kapatıldığında, sahne, arka plan ve skin nesneleri uygun şekilde dispose edilerek bellek yönetimi sağlanır.

Bu yapı sayesinde, oyuncu profili ayar ekranları hem fonksiyonel hem de kullanıcı dostu bir deneyim sunmaktadır. Scene2D'nin sunduğu esnek arayüz yönetimi, kodun okunabilirliğini ve sürdürülebilirliğini artırmıştır. Tüm seçimler, merkezi bir yapı olan **GameSettings** üzerinden yönetilmekte ve ekranlar arası geçişlerde tutarlılık sağlanmaktadır.

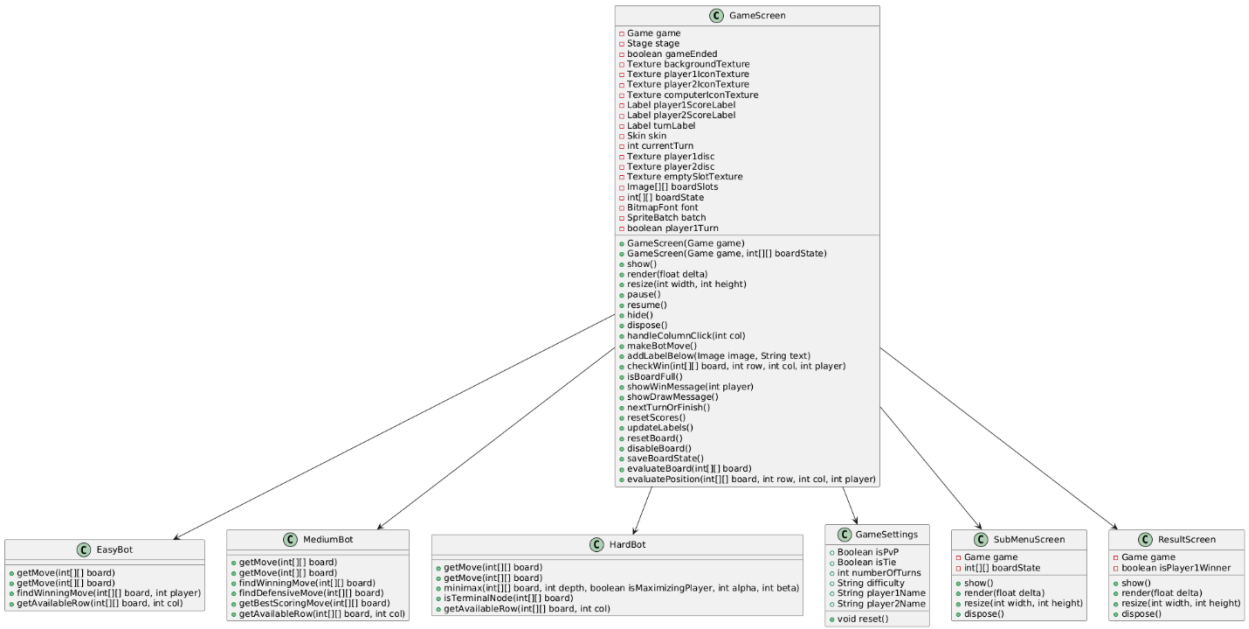
Oyun Ekranı ve Oyun Mantığı: GameScreen.java



Şekil 11. Oyun Ekranı arayüzü

Oyun ekranı, Connect 4 oyununun ana mantığının ve kullanıcı etkileşiminin gerçekleştiği temel sahnedir. Scene2D altyapısı ile tasarlanmış bu ekran, hem görsel hem de işlevsel olarak oyunun tüm dinamiklerini yönetir. Ekran açıldığında, arka plan görseli ile sahne oluşturulur ve oyuncuların ikonları, isimleri ile skorları ekranda gösterilir; ayrıca oyun kaçıncı turda olduğu ve toplam tur sayısı da kullanıcıya iletilir. 6 satır ve 7 sütunlu görsel oyun tahtası, her hücresi tıklanabilir şekilde hazırlanır ve oyuncular ya da bot sırayla

seçtikleri sütuna taşlarını bırakır; taşlar en alttaki boş hücreye yerleşir. Her hamleden sonra yatay, dikey ve çaprazda dörtlü dizilim kontrol edilerek kazanma ya da beraberlik durumu değerlendirilir; sonuçlara göre skorlar güncellenir ve yeni tura geçilir. PvB modunda bilgisayar, kolaydan zora üç farklı zorluk seviyesiyle (rastgele, kural tabanlı, minimax algoritması) hamle yapabilir. Ayrıca, oyun sırasında menüye dönme, oyunu duraklatma ve tur değiştirme gibi yönetim seçenekleri oyuncuya sunularak esnek ve kontrol edilebilir bir oyun deneyimi sağlanır.



Şekil 12. GameScreen UML diyagramı

```
private void handleColumnClick(int col) {
  if (gameEnded) return; // Oyun sona ermişse tıklamayı işlememeliyiz
  for (int row = ROWS - 1; row >= 0; row--) {
    if (boardState[row][col] == 0) {
      int currentPlayer = player1Turn ? 1 : 2;
      boardState[row][col] = currentPlayer;
      TextureRegionDrawable drawable = new TextureRegionDrawable(new TextureRegion(
        currentPlayer == 1 ? player1disc : player2disc));
      boardSlots[row][col].setDrawable(drawable);
      if (checkWin(boardState, row, col, currentPlayer)) {
        showWinMessage(currentPlayer);
        disableBoard();
        return;
      }
      if (isBoardFull()) {
        showDrawMessage();
        disableBoard();
        return;
      }
      player1Turn = !player1Turn; // sıra değiştir
```

```
        if (!GameSettings.isPvP && !player1Turn) {
            makeBotMove();
        }
        updateLabels();
        break;
    }
  }
}
```

Connect 4 oyununda, oyuncu hamlelerinin alınıp tahtaya işlenmesi **handleColumnClick** metoduyla başlar. Bu metod, tıklanan sütunda en alt boş satırı bulmak için 5'ten 0'a doğru yukarıya iterasyon yapar ve ilk boş hücreyi belirler. Daha sonra, sıradaki oyuncunun (player1Turn değişkeni ile takip edilir) taşını **boardState** matrisinde bu hücreye yazar ve aynı hücrenin görselini taş resmiyle günceller. Ardından, **checkWin** metodunu çağırarak bu hamlenin kazanma durumunu tetikleyip tetiklemediği kontrol edilir. Kazanma varsa oyuncunun skorunu artırır, oyunu sonlandırır, aksi halde berabere mi diye **isBoardFull** metodu ile tahta doluluk durumu sorgulanır. Berabere durumunda da oyunu bitirir. Eğer oyun devam edecekse, sıra **player1Turn** değişkeniyle değiştirilir. Eğer tek oyunculu moddaysa ve sıradaki hamle bilgisayarınsa, **makeBotMove** metodu çağrılır.

```
private static boolean checkWin(int[][] board, int row, int col, int player) {
  // 4 taş hizalama kontrolü - yatay, dikey, çapraz
  return (countContinuous(board, row, col, 0, 1, player) + countContinuous(board, row, col, 0, -1, player) - 1 >= 4) ||
      (countContinuous(board, row, col, 1, 0, player) + countContinuous(board, row, col, -1, 0, player) - 1 >= 4) ||
      (countContinuous(board, row, col, 1, 1, player) + countContinuous(board, row, col, -1, -1, player) - 1 >= 4) ||
      (countContinuous(board, row, col, 1, -1, player) + countContinuous(board, row, col, -1, 1, player) - 1 >= 4);
}
```

Kazanmaya yönelik ana denetim **checkWin** metodunda gerçekleşir. Bu metot, son bırakılan taş konumundaki yatay, dikey ve iki çapraz doğrultular boyunca aynı oyuncuya ait mevcut ardışık taş sayısını toplar. Bu sayıları hesaplamak için **countContinuous** metodu kullanılır. Bu metodun temel görevi verilen yönde kat edilen hücrelerde aynı oyuncunun taşlarının kaç tane ardışık olduğunu bulmaktır. Dört ve üzeri taşın yan yana gelmesi durumunda **checkWin** true döner, oyun kazanılmış sayılır.

```
static class EasyBot {
  private static final Random rand = new Random();
  static int getMove(int[][] board) {
    int winningMove = findWinningMove(board, 2);
    if (winningMove != -1) {
      return winningMove;
    }
    int blockingMove = findWinningMove(board, 1);
    if (blockingMove != -1) {
      return blockingMove;
    }
```

```
    int col;
    do {
      col = rand.nextInt(COLS);
    } while (board[0][col] != 0);
    return col;
  }
  private static int findWinningMove(int[][] board, int player) {
    for (int col = 0; col < COLS; col++) {
      int row = getAvailableRow(board, col);
      if (row != -1) {
        board[row][col] = player;
        if (checkWin(board, row, col, player)) {
          board[row][col] = 0;
          return col;
        }
        board[row][col] = 0;
      }
    }
    return -1;
  }
  private static int getAvailableRow(int[][] board, int col) {
    for (int row = ROWS - 1; row >= 0; row--) {
      if (board[row][col] == 0) {
        return row;
      }
    }
    return -1;
  }
}
```

Oyunun zorluk düzeyini belirleyen botların çalışma prensibi ise metodlar bazında farklılaşır. En basit bot olan **EasyBot**, ilk olarak mevcut tahta üzerinde kendi kazanma ihtimalini kontrol eder; **findWinningMove** metodu her sütunda olası bir taş bırakmayı deneyerek kazanma koşulunu eğer sağlıyorsa o hamleyi seçer. Kazanma yoksa, rakibin kazanmasını engellemek için aynı metodu rakip taşı gözüyle çağırır ve bir engelleme hamlesi yapar. Tüm bu kontroller başarısız olursa, rastgele sütun seçerek taş bırakır. Burada önemli olan, rastgele seçim öncesinde sütunun üst hücresinin boş olup olmadığına bakılmasıdır, aksi halde hatalı hamleye sebep olunabilir.

```
private static int getBestScoringMove(int[][] board) {
  int bestScore = Integer.MIN_VALUE;
  ArrayList<Integer> bestCols = new ArrayList<>();
  for (int col = 0; col < COLS; col++) {
    int row = getAvailableRow(board, col);
    if (row != -1) {
      board[row][col] = 2;
      int score = evaluateBoard(board);
```

```
            board[row][col] = 0;
            if (score > bestScore) {
                bestScore = score;
                bestCols.clear();
                bestCols.add(col);
            } else if (score == bestScore) {
                bestCols.add(col);
            }
        }
    }
    if (!bestCols.isEmpty()) {
        Random rand = new Random();
        return bestCols.get(rand.nextInt(bestCols.size()));
    }
    return -1;
}
```

Bir üst seviye bot olan **MediumBot** ise kazanma ve engelleme hamlelerinin ardından, **evaluateBoard** metodunu kullanarak tahtayı puanlar. Bu puanlama, her hücreyi oyuncunun ya da botun son taş koyduğu noktalardan başlayarak dört yönlü dizilimleri analiz eder. **evaluatePosition** metodu bu puanlamanın temelini oluşturur ve oyuncu taşlarının dizilimlerine göre pozitif y a da negatif ağırlıklar verir. Bu değerlere göre en yüksek puan getiren sütun belirlenir ve bot o sütuna hamlesini yapar. Böylece, **MediumBot** sadece anlık hamle değil, ilerleyen turlar için planlama da yaparak oyunu oynar.

```
private static int minimax(int[][] board, int depth, boolean isMaximizingPlayer, int alpha, int beta) {
    if (depth == 0 || isTerminalNode(board)) {
        return evaluateBoard(board);
    }
    if (isMaximizingPlayer) {
        int maxEval = Integer.MIN_VALUE;
        for (int col = 0; col < COLS; col++) {
            int row = getAvailableRow(board, col);
            if (row != -1) {
                board[row][col] = 2;
                int eval = minimax(board, depth - 1, false, alpha, beta);
                board[row][col] = 0;
                maxEval = Math.max(maxEval, eval);
                alpha = Math.max(alpha, eval);
                if (beta <= alpha) break;
            }
        }
        return maxEval;
    } else {
        int minEval = Integer.MAX_VALUE;
        for (int col = 0; col < COLS; col++) {
            int row = getAvailableRow(board, col);
```

```
        if (row != -1) {
            board[row][col] = 1;
            int eval = minimax(board, depth - 1, true, alpha, beta);
            board[row][col] = 0;
            minEval = Math.min(minEval, eval);
            beta = Math.min(beta, eval);
            if (beta <= alpha) break;
        }
      }
      return minEval;
    }
}
```

En karmaşık bot ise **HardBot** olup, Minimax algoritmasını alfa-beta budaması ile destekleyerek kullanır. **minimax** metodu, mevcut tahtada belirlenen derinlik kadar ileri hamleleri simüle eder, her durumda mevcut oyuncunun maksimum kazanımını hedefleyerek puanlama yapar. Bu metod, derinliği bittiğinde veya oyun sonlandığında **evaluateBoard** çağırarak durum değerini hesaplar. Minimax algoritması, oyuncu sırayla kendi hamlesini maksimize etmeye, rakibin hamlesini minimize etmeye çalışır. Alfa-beta budaması ise gereksiz dalların hesaplanmasını engelleyerek performansı artırır. Böylece **HardBot**, bütün hamleleri optimal şekilde değerlendirir; oyun boyunca mümkün olan en doğru hamleyi verir.

```
private static int evaluateBoard(int[][] board) {
 int score = 0;
 Random rand = new Random();
   for (int row = 0; row < ROWS; row++) {
      for (int col = 0; col < COLS; col++) {
         if (board[row][col] == 2) { // Bot
            score += evaluatePosition(board, row, col, 2);
         } else if (board[row][col] == 1) { // Rakip
            score -= evaluatePosition(board, row, col, 1);
         }
      }
   }
   score += rand.nextInt(31) - 15;
   return score;
}
private static int evaluatePosition(int[][] board, int row, int col, int player) {
   int score = 0;
   int[][] directions = {
      {0, 1},  // Yatay →
      {1, 0},  // Dikey ↓
      {1, 1},  // Çapraz □
      {1, -1}  // Çapraz □
   };
```

```
    for (int[] dir : directions) {
        int dr = dir[0];
        int dc = dir[1];
        for (int offset = -3; offset <= 0; offset++) {
            int countPlayer = 0;
            int countEmpty = 0;
            boolean valid = true;
            for (int i = 0; i < 4; i++) {
                int r = row + (offset + i) * dr;
                int c = col + (offset + i) * dc;
                if (r < 0 || r >= ROWS || c < 0 || c >= COLS) {
                    valid = false;
                    break;
                }
                if (board[r][c] == player) {
                    countPlayer++;
                } else if (board[r][c] == 0) {
                    countEmpty++;
                }
            }
            if (valid) {
                if (countPlayer == 4) {
                    score += 1000;
                } else if (countPlayer == 3 && countEmpty == 1) {
                    score += 50;
                } else if (countPlayer == 2 && countEmpty == 2) {
                    score += 10;
                }
            }
        }
    }
    return score;
}
```

Puanlama mantığı **evaluateBoard** ile başlar. Bu fonksiyon, tahtadaki her hücreyi inceler ve oyuncu 2'yi pozitif, oyuncu 1'i negatif puanlarla değerlendirir. **evaluatePosition** metodu, dört farklı yönde 4 taşlık dizilimleri kontrol ederek taş sayısı ve boşluk sayısına göre puanlar verir. Küçük bir rastgelelik eklenmesi ise botların hamlelerine çeşitlilik ve öngörülemezlik katmasını sağlamıştır.

```
private Image[][] boardSlots = new Image[ROWS][COLS];
```

Tüm süreçte, UI ile etkileşimde bulunan **boardSlots** matrisi güncellenerek her hamle görsel olarak doğru şekilde yansıtılır. Tahtanın etkileşimi **disableBoard** ve benzeri metodlarla oyun sonlarında kapatılır, oyuncuların yanlış hamle yapması engellenir. Bu tasarım, oyun mekaniğinin doğruluğunu sağlamak için mantıksal ve görsel veriyi senkronize tutar.

Oyun Akışı ve Tur Yönetimi:

- Her turun sonunda skorlar güncellenir, yeni tur başlatılır veya tüm turlar bittiğinde sonuç ekranına geçilir.
- Oyun sırasında menüye dönülebilir ve oyun duraklatılabilir.

Kaynak Yönetimi:

- Ekran kapatıldığında, sahne, batch, font ve tüm görseller uygun şekilde dispose edilerek bellek yönetimi sağlanır.

Bu yapı sayesinde, Connect 4 oyununun tüm mantığı ve kullanıcı etkileşimi tek bir ekranda yönetilebilmektedir. Scene2D'nin sunduğu esnek arayüz yönetimi, kodun okunabilirliğini ve sürdürülebilirliğini artırmıştır. Bot algoritmalarının farklı zorluk seviyelerinde uygulanması, oyunun tekrar oynanabilirliğini ve eğlencesini artırmaktadır.

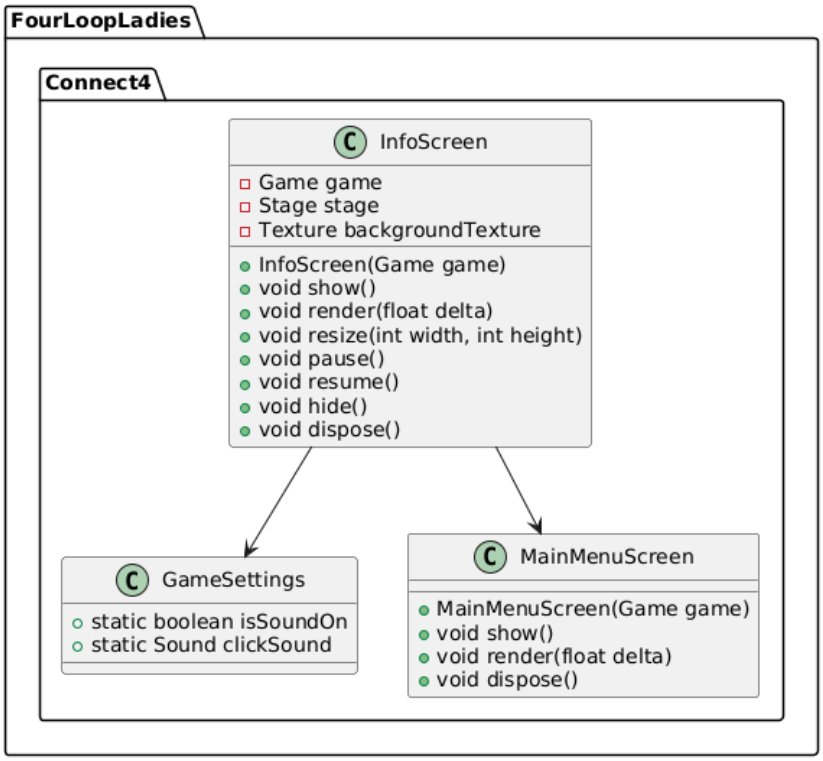
Bilgi Ekranı (InfoScreen.java)

Şekil 13. Bilgi Ekranı arayüzü

Bilgi ekranı, oyunun kurallarını ve oynanışını kullanıcıya açıklamak amacıyla tasarlanmıştır. Scene2D altyapısı ile oluşturulan bu ekranda, arka plan görseli ve ana menüye dönüş için bir çıkış butonu yer alır.

Temel Özellikler:

- **Arka Plan ve Sahne:** Ekran açıldığında, arka plan görseli ve sahne (Stage) oluşturulur.
- **Çıkış Butonu:** Kullanıcı, çıkış butonuna tıklayarak ana menüye dönebilir.
- **Kullanıcı Etkileşimi:** Butona tıklanıldığında ses efekti çalınır ve ekran geçişi sağlanır.



Şekil14. InfoScreen UML diyagramı

Bu sınıf, Game tipinde bir nesne alır ve bu nesne üzerinden ekran geçişlerini sağlar.

```
backgroundTexture = new Texture(Gdx.files.internal("InfoScreen/background.png"));
 Image background = new Image(backgroundTexture);
 background.setFillParent(true);
stage.addActor(background);
backgroundTexture = new Texture(Gdx.files.internal("InfoScreen/background.png"));
 Image background = new Image(backgroundTexture);
 background.setFillParent(true);
 stage.addActor(background);
```

- Çıkış Butonu oluşturulur.

```
backgroundTexture = new Texture(Gdx.files.internal("InfoScreen/background.png"));
Image background = new Image(backgroundTexture);
background.setFillParent(true);
stage.addActor(background);
```

- Çıkış butonuna tıklanırsa ana menüye geçilir

```
ButtonConfig exitConfig = new ButtonConfig("exit", "InfoScreen/exit_button_1.png",
"InfoScreen/exit_button_0.png", 605.9f, 480.5f, 34.2f, 34.2f);
   ImageButton exitButton = exitConfig.createButton();
   exitButton.addListener(new ClickListener() {
      @Override
      public void clicked(InputEvent event, float x, float y) {
         if (GameSettings.isSoundOn) GameSettings.clickSound.play();
         game.setScreen(new MainMenuScreen(game));
      }
   });
   stage.addActor(exitButton);
}
```

ButtonConfig sınıfının yeniden kullanılması, buton oluşturma işlemlerinde tekrarı azaltmış, kodun okunabilirliğini artırmıştır. Ayrıca ekran açılışında sahneye yapılan yüklemelerin temiz ve sistematik oluşu, LibGDX'in Scene2D yapısına uygun profesyonel bir kullanım göstermektedir.

Sonuç olarak, InfoScreen sınıfı; oyun deneyimini doğrudan etkilemeyen ancak dolaylı olarak kullanıcıyı yönlendiren ve bilinçli bir kullanıcı deneyimi sunmayı amaçlayan sade, işlevsel ve amaca yönelik bir ekran olarak başarılı bir şekilde kurgulanmıştır. Projenin geneline yayılan görsel ve etkileşimsel bütünlüğü koruyarak, kullanıcı dostu bir yapı ortaya koymuştur.

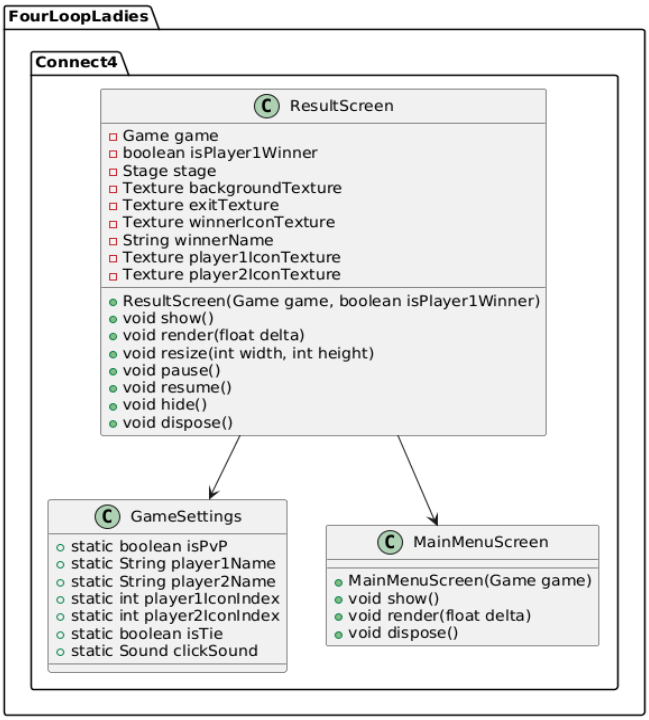
Sonuç Ekranı (ResultScreen.java)

Şekil 15. Sonuç Ekranı arayüzü

ResultScreen sınıfı, Connect4 oyununun tamamlandığı ekranı yönetmek amacıyla oluşturulmuştur. Bu ekran, oyunun sonucuna göre (beraberlik veya kazananın belirlenmesi) farklı içerikler göstermektedir. LibGDX kütüphanesi kullanılarak, ekran üzerine grafiksel öğeler (arka plan, oyuncu ikonları, yazılar ve çıkış butonu) yerleştirilir. Sınıf, Screen arayüzünü implement ederek oyun içinde ekranlar arası geçiş yapılmasına olanak sağlar.

Temel Özellikler:

- **Arka Plan ve Sahne:** Kazanan veya berabere duruma göre farklı arka plan görselleri kullanılır.
- **Kazanan/Beraberlik Bilgisi:** Kazanan oyuncunun veya iki oyuncunun ikonları ve isimleri ekranda büyük puntolarla gösterilir.
- **Çıkış Butonu:** Kullanıcı, çıkış butonuna tıklayarak ana menüye dönebilir.
- **Kullanıcı Etkileşimi:** Butona tıklanıldığında ses efekti çalınır ve ekran geçişi sağlanır.



Şekil 16. ResultScreen UML diyagramı

ResultScreen, oyunun **beraberlik** ya da **galibiyet** durumuna göre dinamik olarak farklı arka plan görselleri yükler:

```
backgroundTexture = new Texture( "ResultScreen/" + (GameSettings.isTie? "draw" : "win") + "_background.png");
```

Oyuncuların seçtiği profil ikonları, GameSettings üzerinden okunur ve kazanan ya da her iki oyuncuya göre sahneye yerleştirilir:

- PvP mi yoksa PvB mi oynandığına göre 2. oyuncunun ikonu manuel olarak "Computer" ikonuyla değiştirilir.

```
player1IconTexture = new Texture("GameScreen/profil_icon_" + (GameSettings.player1IconIndex + 1) + ".png");
  if (GameSettings.isPvP) {
    player2IconTexture = new Texture("GameScreen/profil_icon_" + (GameSettings.player2IconIndex + 1) +
".png");
  } else {
    player2IconTexture = new Texture("SettingScreen/computer_icon.png");
  }
}
```

show() metodu, ekran ilk defa gösterileceğinde çağrılır. Bu metodun işlevleri:

- Yeni bir Stage oluşturulur ve sahneye giriş için InputProcessor ayarlanır.
- Arka plan resmi sahneye eklenir.
- Çıkış butonu, ButtonConfig sınıfı ile oluşturulup sahneye eklenir.

Sonuç durumuna göre iki ayrı senaryo işlenir:

Beraberlik Durumu **(GameSettings.isTie == true)**

- Her iki oyuncunun ikonları sahneye yerleştirilir.
- Oyuncuların adları, beyaz renkte Label nesneleri ile ekrana yazılır.
- Çıkış butonuna tıklandığında, ana menüye dönüş sağlanır.
  Kazanan Belirli Durumu

- Sadece kazanan oyuncunun ikonu ve adı sahneye eklenir.
- Bu bilgiler isPlayer1Winner değerine göre belirlenir.
- winnerName alanına kazanan oyuncunun ismi atanır (ancak bu alan tanımlanmasına rağmen şu anki kodda değer ataması eksiktir; düzeltme gerekebilir).

```
public void show() {
   stage = new Stage(new FitViewport(850, 1080));
   Gdx.input.setInputProcessor(stage);
   BitmapFont font = new BitmapFont();
   LabelStyle labelStyle = new LabelStyle(font, Color.WHITE);
   Image background = new Image(backgroundTexture);
   background.setFillParent(true);
   stage.addActor(background);
   ButtonConfig exitConfig = new ButtonConfig("exit", "SettingScreen/exit_button_1.png",
"SettingScreen/exit_button_0.png", 416.8f, 69.6f, 30f, 30f);
   ImageButton exitButton = exitConfig.createButton();
   stage.addActor(exitButton);
   exitButton.addListener(new ClickListener() {
      public void clicked(InputEvent event, float x, float y) {
         if (GameSettings.isSoundOn) GameSettings.clickSound.play();
         game.setScreen(new MainMenuScreen(game));
      }
   });
```

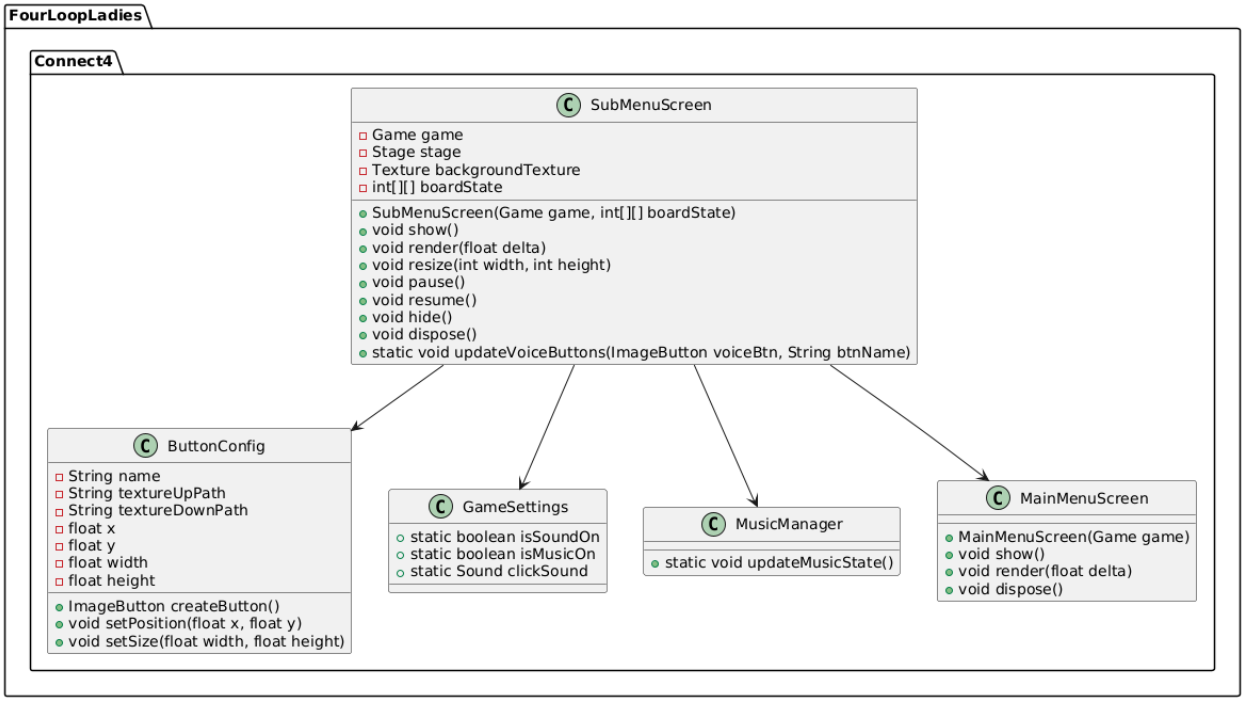
Alt Menü Ekranı (SubMenuScreen.java)

Şekil 17. Alt Menü Ekranı arayüzü

Connect4 oyunu içerisinde oyun duraklatılması durumunda açılan alt menü (pause menu) ekranını yönetmekle sorumludur. Menü ; "Yeniden Başlat", "Devam Et", "Ana Menüye Dön" ve "Ses Aç/Kapat" gibi işlevleri içerir.

Temel Özellikler:

- **Arka Plan ve Sahne:** Ekran açıldığında, arka plan görseli ve sahne (Stage) oluşturulur.
- **Devam Et Butonu:** Kullanıcı, oyuna kaldığı yerden devam edebilir. Oyun tahtasının mevcut durumu korunur.
- **Tekrar Oyna Butonu:** Oyun sıfırlanır ve yeni bir oyun başlatılır.
- **Çıkış Butonu:** Kullanıcı, ana menüye dönebilir.
- **Ses ve Müzik Ayarları:** Ses ve müzik aç/kapa butonları ile anında ayar değiştirilebilir.
- **Kullanıcı Etkileşimi:** Tüm butonlara tıklanıldığında ses efekti çalınır ve ilgili ekran geçişi veya ayar değişikliği sağlanır.



Şekil 18. SubMenuScreen UML diyagramı

Koddan Örnekler:

Alan Tanımları

```
public class SubMenuScreen implements Screen {
   private final Game game;
   private Stage stage;
   private Texture backgroundTexture;
   private int[][] boardState;
```

- game: Ana oyun nesnesidir. Ekranlar arasında geçiş yapılabilmesini sağlar.
- stage: LibGDX'in sahne sistemi ile UI bileşenlerinin bir araya getirildiği yapıdır.
- backgroundTexture: Menü arka plan görselidir.
- boardState: Oyunun duraklatıldığı andaki tahta durumunu (2D matris) temsil eder.

Yapıcı Metot(Constructor)

```
public SubMenuScreen(Game game, int[][] boardState) {
   this.game = game;
   this.boardState = boardState;
}
```

- Dışarıdan oyun durumu ve oyun nesnesi alınır. Bu bilgiler continue butonunun işlevini yerine getirmesinde kullanılacaktır.
- boardState parametresi sayesinde oyun durdurulduğu andaki tahta durumu korunur ve devam etmek istendiğinde aynı yerden sürdürülebilir.

Buton Konfigürasyonları:

Replay, GameScreen' e geçiş:

```
game.setScreen(new GameScreen(game));
```

Continue, boardstate son hali tutulur:

```
game.setScreen(new GameScreen(game, boardState));
```

Exit, ana menüye döner:

```
game.setScreen(new MainMenuScreen(game));
```

Alt menü ekranında yer alan **"Replay"** butonu oyunu sıfırdan başlatırken, **"Continue"** butonu mevcut oyun durumunu koruyarak kaldığı yerden devam etmenizi sağlar. **"Exit"** butonu ana menüye dönmenizi sağlar. Ayrıca, **"Ses"** ve **"Müzik"** butonları oyundaki ses efektleri ve arka plan müziğini anlık olarak açıp kapatmanıza olanak tanır.

Ses Aç/Kapat Butonları

- Butonlara tıklanıldığında boolean değerler ters çevrilerek ses veya müzik anında açılır veya kapatılır.
- Bu butonlar yeniden updateVoiceButtons() fonksiyonuyla görsel olarak güncellenir.

```
GameSettings.isSoundOn = GameSettings.isSoundOn? false: true;

GameSettings.isMusicOn = GameSettings.isMusicOn? false: true;
```

- Bu metod sayesinde ses ya da müzik butonlarının ikonları**,** kullanıcının yaptığı seçime göre dinamik olarak değişir.
- ButtonConfig dışı özel bir güncelleme mekanizmasıdır; çünkü stilin aktif durumda değiştirilmesini sağlar (runtime skin değiştirme).

```
public static void updateVoiceButtons(ImageButton voiceBtn, String btnName) {
    String base = "SettingScreen/"+btnName;
    if (btnName.equals("voice")) {
      voiceBtn.getStyle().imageUp = new TextureRegionDrawable(new TextureRegion(new
Texture(Gdx.files.internal(base + (GameSettings.isSoundOn ? "_on" : "_off") + "_button_1.png"))));
      voiceBtn.getStyle().imageDown = new TextureRegionDrawable(new TextureRegion(new
Texture(Gdx.files.internal(base + (GameSettings.isSoundOn ? "_on" : "_off") + "_button_0.png"))));
    }
    else if (btnName.equals("music")) {
      voiceBtn.getStyle().imageUp = new TextureRegionDrawable(new TextureRegion(new
Texture(Gdx.files.internal(base + (GameSettings.isMusicOn ? "_on" : "_off") + "_button_1.png"))));
      voiceBtn.getStyle().imageDown = new TextureRegionDrawable(new TextureRegion(new
Texture(Gdx.files.internal(base + (GameSettings.isMusicOn ? "_on" : "_off") + "_button_0.png"))));
    }
```

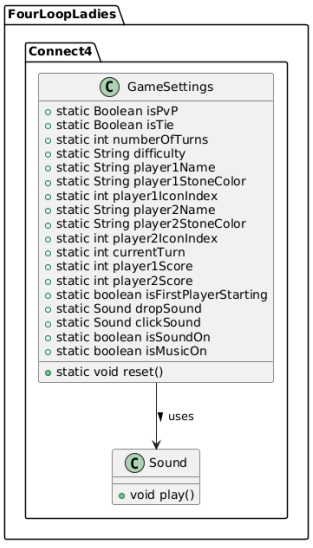
```
  }
}
```

SubMenuScreen sınıfı, oyun sırasında duraklama durumunda kullanıcıya sunulan bir kontrol ekranıdır. Bu ekranda oyuncu oyuna kaldığı yerden devam edebilir, oyunu yeniden başlatabilir veya ana menüye dönebilir. Ayrıca ses ve müzik ayarları anında değiştirilebilir. Kullanıcı etkileşimini artıran bu yapı, dinamik görseller ve oyun durumunu koruma yeteneğiyle dikkat çeker.Bu ekranlar sayesinde, oyunun kullanıcı deneyimi tamamlanmış ve profesyonel bir oyun akışı sağlanmıştır. Scene2D'nin sunduğu esnek arayüz yönetimi, kodun okunabilirliğini ve sürdürülebilirliğini artırmıştır. Tüm ekranlar, merkezi bir yapı olan **GameSettings** üzerinden yönetilmekte ve ekranlar arası geçişlerde tutarlılık sağlanmaktadır.

Oyun Ayarları ve Başlatıcı Sınıflar: GameSettings.java ve Main.java

1. **GameSettings.java**

Oyun boyunca tüm ayarların ve global değişkenlerin merkezi olarak yönetilmesi için **GameSettings** adında bir yardımcı sınıf kullanılmıştır; bu sınıf, oyunun farklı ekranlarında ve akışında ihtiyaç duyulan tüm ayarları ve geçici verileri (oyuncu isimleri, skorlar, ses/müzik durumu, oyun modu, zorluk seviyesi vb.) statik değişkenler olarak tutar. Temel özellikleri arasında PvP/PvB seçimi, zorluk seviyesi ve toplam tur sayısı gibi oyun modu ayarları, her iki oyuncunun adı, taş rengi, profil ikonu ve skorları gibi oyuncu bilgileri, efekt ve müzik için aç/kapa durumları ile ilgili ses dosyaları gibi ses ve müzik ayarları, ayrıca hangi oyuncunun başlayacağı, mevcut tur ve beraberlik durumu gibi oyun akışıyla ilgili değişkenler yer alır. Ayrıca, oyun başında veya sıfırlama gerektiğinde tüm ayarları varsayılan değerlere döndüren bir **reset()** fonksiyonu da bulunmaktadır.



Şekil 19. GameSettings UML diyagramı

```
public static Sound dropSound = Gdx.audio.newSound(Gdx.files.internal("voices/stone_dropping_voice.mp3"));
public static Sound clickSound = Gdx.audio.newSound(Gdx.files.internal("voices/click.wav"));
public static Sound switchSound = Gdx.audio.newSound(Gdx.files.internal("voices/button_switch.mp3"));
```

Bu üç satır, oyunda kullanılacak ses dosyalarını tanımlamak için LibGDX kütüphanesinin **Sound** sınıfını kullanmaktadır. Her bir ses dosyası, oyunun belirli olayları sırasında çalınmak üzere **Gdx.audio.newSound** metodu ile yüklenir. Aşağıda her bir ses dosyasının işlevi açıklanmıştır.

1. **dropSound**:Bu ses, oyuncunun bir taşını tahtaya bıraktığında çalacak olan ses dosyasını temsil eder.
2. **clickSound**:Bu ses, kullanıcı arayüzünde bir butona tıklandığında çalacak olan ses dosyasını temsil eder.
3. **switchSound**:Bu ses, kullanıcı arayüzünde bir butonun durumunun değiştiği (örneğin, bir seçenekten diğerine geçiş) durumlarda çalacak olan ses dosyasını temsil eder.

```
public static void reset() {
 isSoundOn = true;
 isPvP = true;
 numberOfTurns = 3;
 difficulty = "normal";
 player1Name = "Player 1";
 player1StoneColor = "red";
 player1IconIndex = 0;
 player2Name = "Player 2";
 player2StoneColor = "blue";
 player2IconIndex = 1;
 player1Score = 0;
 player2Score = 0;
 isFirstPlayerStarting = true;
}
```

reset metodu, Connect 4 oyununun ayarlarını varsayılan değerlere döndürmek için tasarlanmıştır. Bu metod, yeni bir oyun oturumu başlatıldığında veya mevcut oyunun sıfırlanması gerektiğinde çağrılır.

Metodun içeriği, ses ayarlarını açarak başlar, böylece oyunun başlangıcında seslerin çalınması sağlanır. Oyun modunu çok oyunculu olarak ayarlayarak, varsayılan olarak iki oyuncunun karşı karşıya geleceğini belirtir. Toplam tur sayısını 3 olarak belirler ve oyunun zorluk seviyesini "normal" olarak ayarlar.

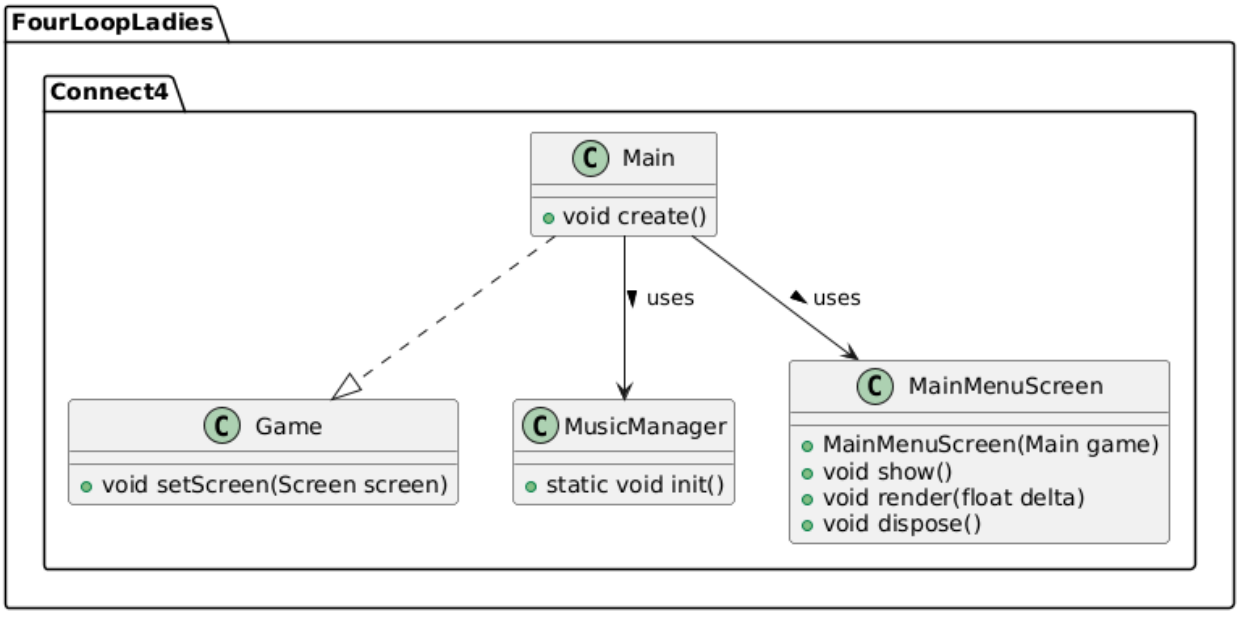
Oyuncu bilgileri de varsayılan değerlere döndürülür; oyuncu 1'in adı "Player 1", taş rengi "kırmızı" ve simge indeksi 0 olarak ayarlanırken, oyuncu 2'nin adı "Player 2", taş rengi "mavi" ve simge indeksi 1 olarak belirlenir. Ayrıca, her iki oyuncunun skoru sıfırlanır.

Son olarak, ilk oyuncunun oyuna başlayacağını belirtmek için **isFirstPlayerStarting** değişkeni **true** olarak ayarlanır. Bu yapı, oyunun her yeni oturumunda tutarlılığı sağlamak ve kullanıcı deneyimini iyileştirmek için kritik öneme sahiptir.

GameSettings sınıfı, Connect 4 oyununun temel ayarlarını ve durumunu yönetmek için kritik bir bileşendir. Oyun modunu, oyuncu bilgilerini, skorları, ses ve müzik ayarlarını tutarak oyunun genel işleyişini kontrol eder. Statik değişkenler sayesinde, bu ayarlara her yerden erişim sağlanabilir. **reset** metodu, oyunun başlangıç ayarlarını geri yükleyerek yeni oyun oturumları için temiz bir başlangıç sunar. Bu yapı, oyunun kullanıcı dostu olmasını ve farklı oyun senaryolarında esneklik sağlamasını mümkün kılar.

2. **Main.java**

Oyun uygulamasının giriş noktası olan **Main** sınıfı, LibGDX'in **Game** sınıfından türetilmiştir ve uygulama başlatıldığında arka plan müziği başlatılarak kullanıcı deneyimi zenginleştirilir; bu işlem, **MusicManager.init()** metodu ile gerçekleştirilir. Ardından, **setScreen(new MainMenuScreen(this));** ifadesi ile ana menü ekranı yüklenir, böylece oyuncular oyuna giriş yapabilir ve oyun seçeneklerini görebilirler. Bu yapı, oyunun başlangıç aşamasında kullanıcı etkileşimini sağlamak için kritik bir rol oynar.



Şekil 20. Main UML diyagramı

```
public class Main extends Game {
  @Override
  public void create() {
    MusicManager.init();
    setScreen(new MainMenuScreen(this));
  }
}
```

Main sınıfı, **Game** sınıfının soyut metodlarından biri olan **create()** metodunu override ederek oyun başlatıldığında yapılacak ilk işlemleri tanımlar. Bu metod içerisinde öncelikle **MusicManager.init()** çağrısı yapılır. Bu çağrı, oyunun müzik ve ses kaynaklarının yüklendiği ve oynatılmaya hazır hale getirildiği başlangıç konfigürasyon fonksiyonudur. Böylece oyun başlamadan önce tüm gerekli ses materyalleri hazır hale getirilir.

Daha sonra **setScreen(new MainMenuScreen(this))** ifadesi yer alır. LibGDX'te **Screen** arayüzü, oyun veya uygulama içinde farklı ekranları (menü, oyun ekranı, ayar ekranı vb.) yönetmek için kullanılır. Bu satırda, oyun başladığında geçilecek ilk ekran olarak **MainMenuScreen** nesnesi oluşturularak aktif ekran olarak atanır. **MainMenuScreen**'e **this** referansı geçirilmesi, oluşturulan ekranın ana oyun sınıfına (yani **Main**) erişebilmesini sağlar; bu, ekranlar arası geçişlerde ve oyun genel durumunun yönetiminde kullanılan yaygın bir desendir.
Bu yapı, uygulamanın başlatılması ve ilk ekranın yüklenmesi için gerekli olan temel akışı sağlar. Tüm ekranlar, bu ana sınıf üzerinden yönetilir ve ekranlar arası geçişler kolayca sağlanır.

Genel Değerlendirme

Bu iki sınıf, projenin temel yapı taşlarını oluşturur. **GameSettings** ile tüm oyun ayarları ve geçici veriler merkezi olarak yönetilirken, **Main** sınıfı ile uygulamanın başlatılması ve ekran yönetimi sağlanır. Bu yapı, projenin modülerliğini ve sürdürülebilirliğini artırır.

6. Test ve Değerlendirme

Test Süreci

Projenin test süreci, hem ekip içi hem de dış kullanıcılarla yapılan denemelerle yürütülmüştür. Ekip üyeleri, kod paylaşımı ve entegrasyonu sırasında oyunu defalarca çalıştırarak işlevselliği ve kullanıcı arayüzünü gözlemlemişlerdir. Ayrıca, oyun aile bireylerine ve okul arkadaşlarına oynatılarak gerçek kullanıcı deneyimi elde edilmiştir. Bu süreçte, kullanıcıların oyunu oynarken karşılaştıkları hatalar ve arayüzle ilgili öneriler

dikkate alınmış, gelen geri bildirimler doğrultusunda çeşitli hata düzeltmeleri ve iyileştirmeler yapılmıştır.

Kullanıcı Geri Bildirimi

Kullanıcılar, oyunun genel olarak eğlenceli ve anlaşılır olduğunu belirtmişlerdir. Özellikle arayüzün renkli ve sade olması, oyunun kolayca oynanabilmesini sağlamıştır. Ancak, oyun sırasında hangi oyuncunun sırasının olduğu bazen anlaşılamadığı için, bu konuda bir görsel veya isim vurgusu eklenmesi önerilmiştir. Ayrıca, kazanan oyuncunun hangi hamleyle oyunu kazandığını gösteren bir animasyon veya işaretleme yapılması da öneriler arasında yer almıştır.

Sonuçlar

Oyun, farklı bilgisayarlarda ve işletim sistemlerinde (Windows, Mac) sorunsuz şekilde çalışmıştır. Performans açısından herhangi bir takılma veya gecikme yaşanmamıştır. Kullanıcı deneyimi açısından, oyunun temel işlevleri ve arayüzü başarılı bulunmuş, alınan geri bildirimlerle daha da geliştirilebilir olduğu görülmüştür

7. Sonuç

Bu proje, ekip üyelerinin hem teknik hem de iş birliği becerilerini geliştirmelerine olanak sağlamıştır. Özellikle daha önce oyun programlama deneyimi olmayan ekip üyeleri için, LibGDX ve Scene2D kullanarak bir oyunun baştan sona nasıl tasarlanıp geliştirileceği konusunda önemli bir öğrenme fırsatı olmuştur. Proje sayesinde, oyun geliştirme konusundaki hedeflerimizden birini gerçekleştirmiş olduk.

Zaman kısıtı nedeniyle oyun iki boyutlu olarak tasarlanmış olsa da, ilerleyen dönemlerde üç boyutlu oyunlar geliştirmek ve mevcut oyuna yeni özellikler eklemek hedeflenmektedir. Özellikle, oyun sırasında sıranın kimde olduğunu daha belirgin göstermek ve kazanan hamleyi vurgulayan bir yapı eklemek planlarımız arasındadır. Ayrıca, yapay zekanın daha gelişmiş seviyelere çıkarılması ve çevrim içi çok oyunculu mod gibi yenilikler de gelecekte değerlendirilebilir.

8. Kaynaklar

Proje geliştirilirken aşağıdaki kaynaklardan yararlanılmıştır:

- OpenGameArt.org – Oyun için kullanılan ses ve bazı görsel materyaller
- LibGDX Resmi Dokümantasyonu ve Wiki
- LibGDX Ses Efektleri
- LibGDX Müzik Akışı
- Canva – Arayüz tasarımı ve görsel düzenlemeler için
- LibGDX forumları ve StackOverflow – Karşılaşılan teknik sorunların çözümü için

LibGDX'in resmi dökümantasyonu ve topluluk kaynakları, oyun tasarımı ve programlama sürecinde en çok başvurulan kaynaklar olmuştur.