T.C
KOCAELİ SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR/YAZILIM MÜHENDİSLİĞİ

PROJE KONUSU:

POKEMON KART OYUNU

ÖĞRENCİ ADI:
NAZLI SU KETÇİ
AMİNE DERİN

ÖĞRENCİ NUMARASI:
220501007
220501004
Github:
https://github.com/nzlktc02
https://github.com/aminederin

TARİH: 04.07.2024

1.GİRİŞ

Projenin Amacı:

Bu projede basit bir Pokémon kart oyunu oluşturmayı amaçlamaktadır. Proje, oyuncuların rastgele dağıtılan Pokémon kartlarıyla karşılıklı olarak oynadığı ve kartların hasar puanlarına göre kazananın belirlendiği bir oyun sunar.

1. **Nesne Yönelimli Programlama Prensiplerini Kullanmak**: Bu projede OOP'nin dört temel prensibi olan Encapsulation (Kapsülleme), Inheritance (Kalıtım), Polymorphism (Çok Biçimlilik) ve Abstraction (Soyutlama) kavramları kullanılmaktadır.
2. **Grafik Kullanıcı Arayüzü (GUI) Oluşturmak**: Tkinter kullanılarak, kullanıcıların etkileşimde bulunabileceği basit ve kullanıcı dostu bir arayüz oluşturulmuştur.
3. **Kart Oyunu Mekaniklerini Uygulamak**: Kartların rastgele dağıtılması, oyuncuların kart seçimi, tur sonuçlarının belirlenmesi ve oyunun sonlandırılması gibi temel kart oyunu mekanikleri uygulanmıştır.

Projede Kullanılan OOP Prensipleri ve Yapılar

1. **Encapsulation (Kapsülleme)**:
   - Sınıfların içinde değişkenler (_name, _damage, _hand, _score) kapsüllenmiş ve bu değişkenlere erişim @property dekoratörleri ile sağlanmıştır.
2. **Inheritance (Kalıtım)**:
   - Card sınıfı genel kart özelliklerini temsil eder ve PokemonCard sınıfı bu sınıftan türetilmiştir.
   - Player sınıfı genel oyuncu özelliklerini temsil eder ve HumanPlayer ile ComputerPlayer sınıfları bu sınıftan türetilmiştir.
3. **Polymorphism (Çok Biçimlilik)**:
   - Player sınıfında tanımlanan draw_card ve play_card metotları, HumanPlayer ve ComputerPlayer sınıflarında aynı işlevselliği sağlar, bu da çok biçimlilik örneğidir.
4. **Abstraction (Soyutlama)**:
   - Card sınıfı ve Player sınıfı, belirli işlevselliği soyutlayarak genel bir yapı sunar. PokemonCard ve HumanPlayer/ComputerPlayer sınıfları, bu soyut yapıları özelleştirir.

Proje Gereksinimlerini Karşılamak

- **Yapıcı (Constructor) Metotlar**:
  - Card, PokemonCard, Player, HumanPlayer ve ComputerPlayer sınıflarında hem parametreli hem de parametresiz yapıcı metotlar bulunmaktadır.
- **Get ve Set Metotları**:
  - Card ve Player sınıflarında tüm özellikler için get (@property) ve set

(@property.setter) metotları tanımlanmıştır.

2.GEREKSİNİM ANALİZİ:

Gereksinim Analizi

Bu bölüm, Pokémon Kart Oyunu projesi için yapılacak gereksinim analizini kapsamaktadır. Gereksinim analizi, projenin gereksinimlerini belirlemek, analiz etmek ve dokümante etmek için yapılan bir süreçtir. Bu analiz, projenin doğru ve eksiksiz bir şekilde tamamlanabilmesi için gereklidir.

1. Fonksiyonel Gereksinimler

1.1. Oyun Başlangıcı

- **Kart Destesi Oluşturma**: Oyun başladığında, içinde 10 farklı Pokémon kartı bulunan bir deste oluşturulmalıdır.
- **Kartların Karıştırılması**: Deste oyun başlamadan önce rastgele karıştırılmalıdır.
- **Kartların Dağıtılması**: Hem oyuncuya hem de bilgisayara başlangıçta 3'er kart dağıtılmalıdır.

1.2. Oyun Mekaniği

- **Kart Çekme**: Her oyuncu sırası geldiğinde desteden kart çekebilmelidir.
- **Kart Oynama**: Her iki oyuncu da (oyuncu ve bilgisayar) tur başına bir kart oynayabilmelidir.
- **Tur Sonucu**: Oynanan kartların hasar puanlarına göre tur kazananı belirlenmeli ve puanlar güncellenmelidir.
- **Oyun Sonucu**: Destedeki tüm kartlar oynandığında oyun bitmeli ve toplam puanlara göre kazanan belirlenmelidir.

1.3. Grafik Kullanıcı Arayüzü (GUI)

- **Ana Pencere**: Oyunun oynanacağı ana pencere oluşturulmalıdır.
- **Kart Görselleri**: Her kart için görseller doğru bir şekilde gösterilmelidir.
- **Skor Takibi**: Oyuncu ve bilgisayarın skorları GUI üzerinden takip edilebilmelidir.
- **Tur ve Oyun Sonuçları**: Tur ve oyun sonuçları mesaj kutuları aracılığıyla oyuncuya bildirilmeli ve GUI güncellenmelidir.

2. Teknik Gereksinimler

2.1. Yazılım Gereksinimleri

- **Python 3.x**: Proje Python programlama dili ile yazılacaktır.
- **Tkinter**: GUI oluşturmak için Tkinter kullanılacaktır.
- **PIL (Pillow)**: Kart görsellerinin işlenmesi ve gösterilmesi için Pillow kütüphanesi kullanılacaktır.

2.2. Donanım Gereksinimleri

- Projenin çalıştırılabilmesi için temel bir bilgisayar gereklidir.

3. OOP (Nesne Yönelimli Programlama) Gereksinimleri

3.1. Encapsulation (Kapsülleme)

- **Veri Kapsülleme**: Sınıf içindeki veriler (örneğin, oyuncu adı, kart hasar puanı) kapsüllenmelidir. Bu verilere doğrudan erişim yerine get/set metotları kullanılmalıdır.

3.2. Inheritance (Kalıtım)

- **Sınıf Mirası**: `Card` sınıfı, genel kart özelliklerini kapsayacak ve `PokemonCard` sınıfı bu sınıftan türetilecektir. Benzer şekilde, `Player` sınıfı genel oyuncu özelliklerini kapsayacak ve `HumanPlayer` ile `ComputerPlayer` sınıfları bu sınıftan türetilecektir.

3.3. Polymorphism (Çok Biçimlilik)

- **Metot Geçersiz Kılma**: `Player` sınıfındaki `draw_card` ve `play_card` metotları `HumanPlayer` ve `ComputerPlayer` sınıflarında kullanılacak ve gerektiğinde özelleştirilecektir.

3.4. Abstraction (Soyutlama)

- **Soyutlama Kullanımı**: `Card` ve `Player` sınıfları, oyun içinde genel işlevselliği kapsayacak şekilde soyutlanacaktır. Bu sınıflar, daha spesifik sınıflar (PokemonCard, HumanPlayer, ComputerPlayer) tarafından özelleştirilecektir.

4. Diğer Gereksinimler

4.1. Kullanılabilirlik

Kullanıcı Dostu Arayüz: Bu bölüm, Pokémon Kart Oyunu projesi için kullanıcı arayüzü gereksinimlerini detaylandırmaktadır. Kullanıcı arayüzü (GUI), oyuncuların oyunla etkileşimde bulunduğu ve oyunun görsel öğelerini gördüğü bölümdür. Kullanıcı arayüzü gereksinimleri, oyunun kullanılabilirliğini ve kullanıcı deneyimini doğrudan etkiler.

1. Ana Pencere

- **Pencere Başlığı**: Pencere başlığı "Pokemon Kart Oyunu" olarak ayarlanmalıdır.
- **Pencere Boyutu**: Pencere boyutu oyunun tüm bileşenlerini rahatça gösterebilecek şekilde ayarlanmalıdır.
- **Pencere İkonu**: Mümkünse, pencere ikonu olarak Pokémon temalı bir ikon kullanılmalıdır.

2. Oyuncu ve Bilgisayar Panelleri

- **Oyuncu Paneli**: Oyuncunun kartlarını ve skorunu gösteren bir panel oluşturulmalıdır. Bu panel sol tarafta yer almalıdır.
    - **Oyuncu Kart Görselleri**: Oyuncunun elindeki kartların görselleri gösterilmelidir.
    - **Oyuncu Skoru**: Oyuncunun mevcut skorunu gösteren bir etiket (label) bulunmalıdır.
- **Bilgisayar Paneli**: Bilgisayarın kartlarını ve skorunu gösteren bir panel oluşturulmalıdır. Bu panel sağ tarafta yer almalıdır.
    - **Bilgisayar Kart Görselleri**: Bilgisayarın elindeki kartların görselleri gösterilmelidir (gizli olabilir).
    - **Bilgisayar Skoru**: Bilgisayarın mevcut skorunu gösteren bir etiket (label) bulunmalıdır.

3. Kart Görselleri

- **Kart Boyutları**: Kart görselleri standart bir boyutta (örneğin 200x300 piksel) olmalıdır.
- **Kart Dosyaları**: Kart görselleri `images` klasöründe saklanmalı ve kart isimlerine göre adlandırılmalıdır (örneğin, `bulbasaur.jpg`).

4. Tur ve Oyun Bilgilendirmesi

- **Tur Sonucu Mesaj Kutusu**: Her turun sonucunda, turun kazananını ve skor güncellemelerini gösteren bir mesaj kutusu açılmalıdır.
- **Oyun Sonucu Mesaj Kutusu**: Oyun bittiğinde, oyunun kazananını ve son skorları gösteren bir mesaj kutusu açılmalıdır.

5. Oyun Kontrolleri

- **Kart Seçimi**: Oyuncunun elindeki kartlardan birini seçebilmesi için tıklanabilir butonlar bulunmalıdır.
- **Tur Başlatma Butonu**: Seçilen kart ile turu başlatmak için bir buton (örn. "Raund Oyna" butonu) bulunmalıdır.
    - **Buton Durumu**: Oyuncu bir kart seçene kadar bu buton devre dışı olmalıdır.

- **Kart Çekme**: Oyuncunun yeni bir kart çekebilmesi için bir mekanizma olmalıdır (tur sonunda otomatik olarak çekilebilir).

6. Skor Takibi

- **Skor Etiketleri**: Hem oyuncunun hem de bilgisayarın skorları güncel olarak gösterilmelidir.
    - **Oyuncu Skoru Etiketi**: Oyuncunun skorunu gösteren bir etiket, sol panelde yer almalıdır.
    - **Bilgisayar Skoru Etiketi**: Bilgisayarın skorunu gösteren bir etiket, sağ panelde yer almalıdır.

7. Genel Gereksinimler

- **Kullanıcı Dostu Arayüz**: Arayüz, kullanıcıların oyunu kolayca anlamalarını ve oynamalarını sağlayacak şekilde basit ve kullanıcı dostu olmalıdır.
- **Hata Yönetimi**: Herhangi bir hata durumunda (örneğin, resim dosyası bulunamadığında), kullanıcıya anlamlı bir hata mesajı gösterilmelidir.
- **Performans**: Arayüz, hızlı yanıt vermeli ve kullanıcı girişlerine anında tepki vermelidir.

4.2. Performans

- **Hızlı Yanıt**: Oyuncu etkileşimlerine hızlı yanıt verilmelidir. Kart çekme, oynama ve skor güncellemeleri hızlı bir şekilde gerçekleştirilmelidir.

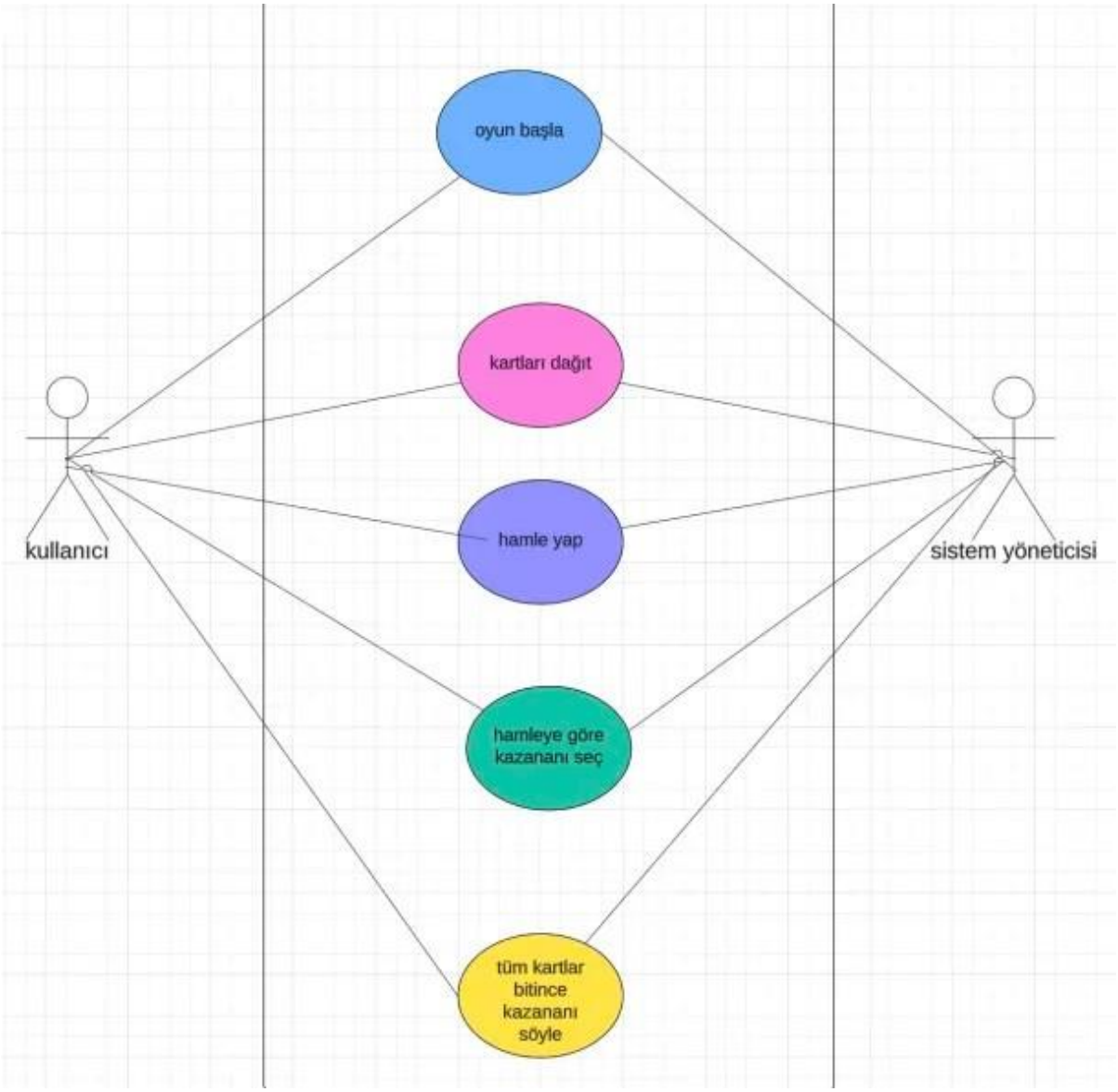
4.3. Güvenilirlik

- **Hata Yönetimi**: Kart görselleri yüklenirken veya dosya bulunamadığında hatalar uygun şekilde yönetilmelidir. Program çökmeden kullanıcıya bilgi verilmelidir.

Sonuç

Bu gereksinim analizi, Pokémon Kart Oyunu projesinin tüm fonksiyonel, teknik ve OOP gereksinimlerini kapsamaktadır. Proje bu gereksinimlere göre geliştirilerek, kullanıcıların keyif alabileceği basit bir kart oyunu sunmayı amaçlamaktadır. Gereksinimlerin doğru bir şekilde karşılanması, projenin başarısı için kritik öneme sahiptir.

Use- Case diyagramı:



3.MİMARİ TASARIM

Mimari Tasarım:

Bu bölümde, Pokémon Kart Oyunu projesinin mimari tasarımı detaylandırılmaktadır. Mimari tasarım, sistemin yapısal düzenini ve bileşenlerinin birbirleriyle nasıl etkileşime gireceğini tanımlar. Bu tasarım, projenin OOP (Nesne Yönelimli Programlama) prensiplerine dayalı olarak oluşturulmasını sağlar. 2. Sınıflar ve Metotlar

2.1. Card Sınıfı

- **Amacı**: Genel kart özelliklerini temsil eder.
- **Özellikler**:
  - `_name`: Kartın adı.
  - `_damage`: Kartın hasar puanı.
- **Metotlar**:
  - `__init__(self, name="Unknown", damage=0)`: Kartı oluşturur.
  - `__str__(self)`: Kartın adını ve hasar puanını döner.
  - `name` (getter ve setter): Kartın adı.
  - `damage` (getter ve setter): Kartın hasar puanı.
  - `get_image_path(self)`: Kartın görsel dosya yolunu döner.

2.2. PokemonCard Sınıfı

- **Amacı**: `Card` sınıfından türeyen ve Pokémon kartlarını temsil eden sınıf.
- **Özellikler**: `Card` sınıfından miras alınan özellikler.
- **Metotlar**: `Card` sınıfından miras alınan metotlar.

2.3. Player Sınıfı

- **Amacı**: Genel oyuncu özelliklerini temsil eder.
- **Özellikler**:
  - `_name`: Oyuncunun adı.
  - `_hand`: Oyuncunun elindeki kartlar.
  - `_score`: Oyuncunun skoru.
- **Metotlar**:
  - `__init__(self, name="Player")`: Oyuncuyu oluşturur.
  - `__str__(self)`: Oyuncunun adını ve skorunu döner.
  - `draw_card(self, deck)`: Desteden kart çeker.
  - `play_card(self, index)`: Belirtilen indeksteki kartı oynar.
  - `name` (getter ve setter): Oyuncunun adı.
  - `hand` (getter ve setter): Oyuncunun elindeki kartlar.
  - `score` (getter ve setter): Oyuncunun skoru.

2.4. HumanPlayer Sınıfı

- **Amacı**: İnsan oyuncuyu temsil eden sınıf.
- **Özellikler**: `Player` sınıfından miras alınan özellikler.
- **Metotlar**: `Player` sınıfından miras alınan metotlar.

2.5. ComputerPlayer Sınıfı

- **Amacı**: Bilgisayar oyuncusunu temsil eden sınıf.
- **Özellikler**: `Player` sınıfından miras alınan özellikler.

- **Metotlar**: `Player` sınıfından miras alınan metotlar.

2.6. Game Sınıfı

- **Amacı**: Oyunun genel akışını ve mantığını yönetir.
- **Özellikler**:
  - `deck`: Oyunun kart destesi.
  - `player`: İnsan oyuncu.
  - `computer`: Bilgisayar oyuncu.
  - `current_cards`: Oynanan kartlar.
  - `player_card_index`: Oyuncunun seçtiği kartın indeksi.
  - `window`: Tkinter ana penceresi.
- **Metotlar**:
  - `__init__(self)`: Oyunu başlatır ve gerekli bileşenleri oluşturur.
  - `create_deck(self)`: Kart destesini oluşturur.
  - `shuffle_deck(self)`: Kart destesini karıştırır.
  - `deal_cards(self)`: Kartları oyunculara dağıtır.
  - `play_round(self)`: Bir tur oynar.
  - `start_game(self)`: Oyunu başlatır.
  - `end_game(self)`: Oyunu sonlandırır ve kazananı ilan eder.
  - `create_widgets(self)`: GUI bileşenlerini oluşturur.
  - `update_hand(self)`: Oyuncunun elindeki kartları günceller.
  - `select_card(self, index)`: Oyuncunun seçtiği kartı belirler.
  - `update_card_images(self, player_card, computer_card)`: Oynanan kartların görsellerini günceller.
  - `update_scores(self)`: Skorları günceller.

3. Bileşenler Arası Etkileşim

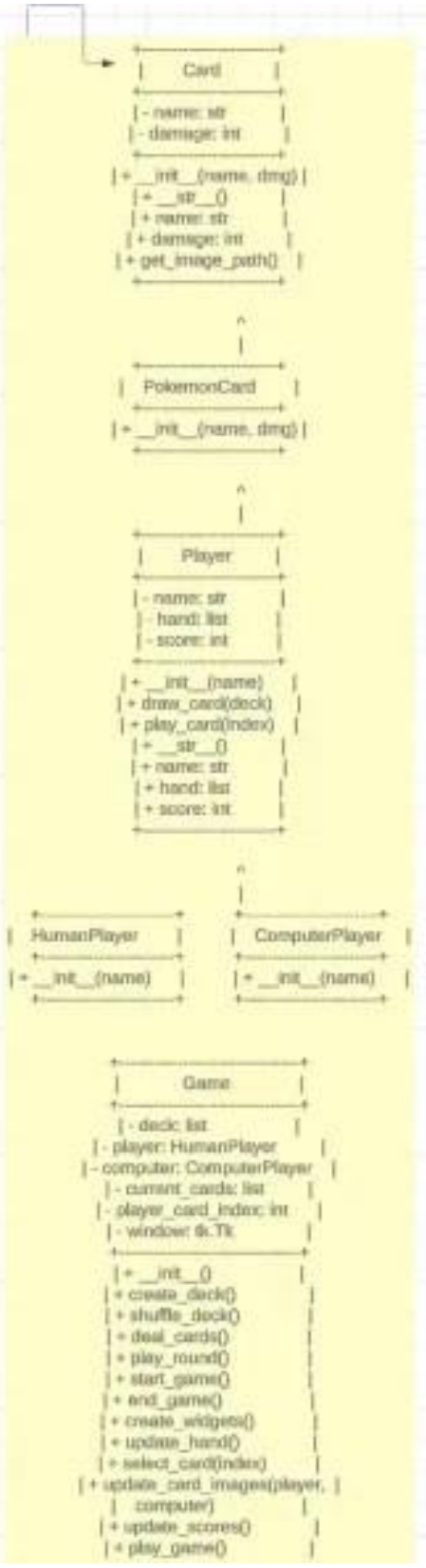
- **Oyun Başlangıcı**:
  1. `Game` sınıfı, `__init__` metodu ile başlatılır.
  2. `create_deck` metodu, `PokemonCard` nesnelerini içeren bir deste oluşturur.
  3. `shuffle_deck` metodu ile deste karıştırılır.
  4. `deal_cards` metodu, hem `HumanPlayer` hem de `ComputerPlayer` nesnelerine kartlar dağıtır.
  5. `create_widgets` metodu, GUI bileşenlerini oluşturur.
- **Oyun Akışı**:
  1. Oyuncu, GUI üzerinden bir kart seçer ve `select_card` metodu ile bu seçim kaydedilir.
  2. `play_round` metodu çağrıldığında, her iki oyuncu da birer kart oynar ve bu kartların hasar puanları karşılaştırılır.
  3. Tur sonucu `messagebox.showinfo` ile kullanıcıya bildirilir ve skorlar `update_scores` metodu ile güncellenir.
  4. Kartlar `update_card_images` metodu ile güncellenir.

5. Yeni kartlar `update_hand` metodu ile dağıtılır.
6. Destedeki kartlar bittiğinde `end_game` metodu çağrılır ve oyun sonucu ilan edilir.

4. Kullanıcı Arayüzü (GUI)

- **Ana Pencere**: Tkinter ana penceresi, oyunun tüm bileşenlerini barındırır.
- **Oyuncu Paneli**: Oyuncunun kartları ve skoru burada gösterilir.
- **Bilgisayar Paneli**: Bilgisayarın kartları ve skoru burada gösterilir.
- **Kart Görselleri**: Oyuncunun ve bilgisayarın oynadığı kartların görselleri güncellenir.
- **Butonlar**: Turu başlatmak ve kart seçimini yapmak için butonlar bulunur.

MODÜL DİYAGRAMI



Kullanılacak Teknolojiler

Pokémon Kart Oyunu projesi için kullanılacak teknolojiler ve kütüphaneler aşağıda detaylandırılmıştır. Bu teknolojiler, projenin geliştirilmesi ve çalıştırılması için gerekli olan tüm yazılım ve araçları içerir.

1. Programlama Dili

- **Python 3.x**: Proje, esnekliği, geniş kütüphane desteği ve kolay öğrenilebilirliği nedeniyle Python programlama dili kullanılarak geliştirilecektir. Python, nesne yönelimli programlama (OOP) yapısı sayesinde proje gereksinimlerini karşılamak için ideal bir seçimdir.

2. Grafik Kullanıcı Arayüzü (GUI)

- **Tkinter**: Python'un standart GUI kütüphanesi olan Tkinter, kullanıcı arayüzünü oluşturmak için kullanılacaktır. Tkinter, basit ve kullanıcı dostu bir arayüz oluşturmak için gerekli bileşenleri sağlar.

3. Görsel İşleme

- **Pillow (PIL)**: Python Imaging Library'nin (PIL) bir çatalı olan Pillow, kart görsellerini işlemek ve GUI'de göstermek için kullanılacaktır. Pillow, resim dosyalarını açma, yeniden boyutlandırma ve dönüştürme gibi işlemleri kolayca gerçekleştirmeyi sağlar.

4. Geliştirme Ortamı

- **IDE veya Metin Editörü**: Python kodunu yazmak ve düzenlemek için herhangi bir gelişmiş metin editörü veya Entegre Geliştirme Ortamı (IDE) kullanılabilir. Önerilen IDE'ler arasında PyCharm, Visual Studio Code ve Sublime Text bulunmaktadır.

5. Proje Yönetimi ve Versiyon Kontrolü

- **Git**: Proje kaynak kodlarının yönetimi ve sürüm kontrolü için Git kullanılacaktır. Git, kod değişikliklerini izlemek, işbirliğini kolaylaştırmak ve proje geçmişini yönetmek için ideal bir araçtır.
- **GitHub**: Proje depolama ve işbirliği için GitHub kullanılabilir. GitHub, kaynak kodlarını barındırmak, sürüm kontrolü yapmak ve ekip üyeleri arasında işbirliği sağlamak için yaygın olarak kullanılan bir platformdur.

6. Dokümantasyon

- **Markdown**: Proje dokümantasyonu, okunabilir ve yazımı kolay olan Markdown formatında yazılacaktır. README dosyaları, kullanım kılavuzları ve diğer dökümantasyonlar için Markdown kullanılabilir.

- **Sphinx**: Daha kapsamlı bir dokümantasyon oluşturmak için Sphinx kullanılabilir. Sphinx, Python projeleri için otomatik olarak API dokümantasyonu oluşturmayı sağlayan bir araçtır.

7. Grafik ve Resim Dosyaları

- **Resim Dosyaları**: Pokémon kartlarının görselleri için resim dosyaları kullanılacaktır. Bu resimler `images` adlı bir klasörde saklanacak ve kart isimlerine göre adlandırılacaktır (örneğin, `bulbasaur.jpg`).

Teknoloji Kullanımı

- **Python**: Kodun temel yapısı ve iş mantığı Python ile yazılacaktır. Sınıflar, metotlar ve oyun mekanikleri Python kullanılarak oluşturulacaktır.
- **Tkinter**: Kullanıcı arayüzü Tkinter ile oluşturulacak. Pencere yönetimi, düğmeler, etiketler ve kart görselleri Tkinter bileşenleri kullanılarak oluşturulacaktır.
- **Pillow (PIL)**: Kart görselleri, Pillow kullanılarak açılacak, yeniden boyutlandırılacak ve Tkinter'da görüntülenecektir.
- **Git ve GitHub**: Proje kodları ve dokümantasyonu Git kullanılarak versiyon kontrolü yapılacak ve GitHub üzerinde depolanacaktır.
- **Markdown ve Sphinx**: Proje dokümantasyonu, Markdown formatında yazılacak ve Sphinx kullanılarak API dokümantasyonu oluşturulacaktır.

Kullanıcı Arayüzü Tasarımı

1. Programlama Dili

- **Python 3.x**: Proje, esnekliği, geniş kütüphane desteği ve kolay öğrenilebilirliği nedeniyle Python programlama dili kullanılarak geliştirilecektir. Python, nesne yönelimli programlama (OOP) yapısı sayesinde proje gereksinimlerini karşılamak için ideal bir seçimdir.

2. Grafik Kullanıcı Arayüzü (GUI)

- **Tkinter**: Python'un standart GUI kütüphanesi olan Tkinter, kullanıcı arayüzünü oluşturmak için kullanılacaktır. Tkinter, basit ve kullanıcı dostu bir arayüz oluşturmak için gerekli bileşenleri sağlar.

3. Görsel İşleme

- **Pillow (PIL)**: Python Imaging Library'nin (PIL) bir çatalı olan Pillow, kart görsellerini işlemek ve GUI'de göstermek için kullanılacaktır. Pillow, resim dosyalarını açma, yeniden boyutlandırma ve dönüştürme gibi işlemleri kolayca gerçekleştirmeyi sağlar.

4. Geliştirme Ortamı

- **IDE veya Metin Editörü**: Python kodunu yazmak ve düzenlemek için herhangi bir gelişmiş metin editörü veya Entegre Geliştirme Ortamı (IDE) kullanılabilir. Önerilen IDE'ler arasında PyCharm, Visual Studio Code ve Sublime Text bulunmaktadır.

5. Proje Yönetimi ve Versiyon Kontrolü

- **Git**: Proje kaynak kodlarının yönetimi ve sürüm kontrolü için Git kullanılacaktır. Git, kod değişikliklerini izlemek, işbirliğini kolaylaştırmak ve proje geçmişini yönetmek için ideal bir araçtır.
- **GitHub**: Proje depolama ve işbirliği için GitHub kullanılabilir. GitHub, kaynak kodlarını barındırmak, sürüm kontrolü yapmak ve ekip üyeleri arasında işbirliği sağlamak için yaygın olarak kullanılan bir platformdur.

6. Dokümantasyon

- **Markdown**: Proje dokümantasyonu, okunabilir ve yazımı kolay olan Markdown formatında yazılacaktır. README dosyaları, kullanım kılavuzları ve diğer dökümantasyonlar için Markdown kullanılabilir.
- **Sphinx**: Daha kapsamlı bir dokümantasyon oluşturmak için Sphinx kullanılabilir. Sphinx, Python projeleri için otomatik olarak API dokümantasyonu oluşturmayı sağlayan bir araçtır.

7. Grafik ve Resim Dosyaları

- **Resim Dosyaları**: Pokémon kartlarının görselleri için resim dosyaları kullanılacaktır. Bu resimler `images` adlı bir klasörde saklanacak ve kart isimlerine göre adlandırılacaktır (örneğin, `bulbasaur.jpg`).

Teknoloji Kullanımı

- **Python**: Kodun temel yapısı ve iş mantığı Python ile yazılacaktır. Sınıflar, metotlar ve oyun mekanikleri Python kullanılarak oluşturulacaktır.
- **Tkinter**: Kullanıcı arayüzü Tkinter ile oluşturulacak. Pencere yönetimi, düğmeler, etiketler ve kart görselleri Tkinter bileşenleri kullanılarak oluşturulacaktır.
- **Pillow (PIL)**: Kart görselleri, Pillow kullanılarak açılacak, yeniden boyutlandırılacak ve Tkinter'da görüntülenecektir.
- **Git ve GitHub**: Proje kodları ve dokümantasyonu Git kullanılarak versiyon kontrolü yapılacak ve GitHub üzerinde depolanacaktır.
- **Markdown ve Sphinx**: Proje dokümantasyonu, Markdown formatında yazılacak ve Sphinx kullanılarak API dokümantasyonu oluşturulacaktır.

Özet

Bu projede kullanılacak teknolojiler ve araçlar, Pokémon Kart Oyunu'nun geliştirilmesi, çalıştırılması ve dokümantasyonunun oluşturulması için gerekli tüm bileşenleri kapsamaktadır. Python, Tkinter ve Pillow, oyunun temel işlevselliğini

ve kullanıcı arayüzünü oluşturmak için kullanılırken, Git ve GitHub kaynak kod yönetimi ve işbirliği için kullanılacaktır. Markdown ve Sphinx, projenin dokümantasyonunu oluşturmak için tercih edilecektir.

Kullanıcı Arayüzü Tasarımı
ChatGPT

Kullanıcı Arayüzü Tasarımı

Pokémon Kart Oyunu'nun kullanıcı arayüzü tasarımı, kullanıcı dostu ve etkileşimli bir oyun deneyimi sunmayı amaçlamaktadır. Aşağıda, oyunun kullanıcı arayüzü bileşenleri ve bunların işlevleri detaylı bir şekilde açıklanmıştır.

1. Ana Pencere

- **Pencere Başlığı**: "Pokemon Kart Oyunu" olarak ayarlanmalıdır.
- **Pencere Boyutu**: Tüm bileşenlerin rahatça sığabileceği bir boyutta olmalıdır. Örneğin, 800x600 piksel.

2. Oyuncu ve Bilgisayar Panelleri

- **Oyuncu Paneli**: Oyuncunun kartlarını ve skorunu gösteren bir panel oluşturulmalıdır. Bu panel sol tarafta yer alacaktır.
    - **Kart Görselleri**: Oyuncunun elindeki kartların görselleri bu panelde gösterilecektir.
    - **Oyuncu Skoru**: Oyuncunun mevcut skorunu gösteren bir etiket bu panelde yer alacaktır.
- **Bilgisayar Paneli**: Bilgisayarın kartlarını ve skorunu gösteren bir panel oluşturulmalıdır. Bu panel sağ tarafta yer alacaktır.
    - **Kart Görselleri**: Bilgisayarın elindeki kartların görselleri bu panelde gösterilecektir.
    - **Bilgisayar Skoru**: Bilgisayarın mevcut skorunu gösteren bir etiket bu panelde yer alacaktır.

3. Kart Görselleri

- **Kart Boyutları**: Kart görselleri 200x300 piksel boyutlarında olmalıdır.
- **Kart Dosyaları**: Kart görselleri `images` klasöründe saklanacak ve kart isimlerine göre adlandırılacaktır (örneğin, `bulbasaur.jpg`).

4. Oyun Kontrolleri

- **Kart Seçimi**: Oyuncunun elindeki kartlardan birini seçebilmesi için tıklanabilir butonlar yer almalıdır.
- **Tur Başlatma Butonu**: Seçilen kart ile turu başlatmak için bir buton (örn. "Raund Oyna" butonu) yer almalıdır.
    - **Buton Durumu**: Oyuncu bir kart seçene kadar bu buton devre dışı olmalıdır.

5. Tur ve Oyun Bilgilendirmesi

- **Tur Sonucu Mesaj Kutusu**: Her turun sonucunda, turun kazananını ve skor güncellemelerini gösteren bir mesaj kutusu açılmalıdır.
- **Oyun Sonucu Mesaj Kutusu**: Oyun bittiğinde, oyunun kazananını ve son skorları gösteren bir mesaj kutusu açılmalıdır.

6. Skor Takibi

- **Skor Etiketleri**: Hem oyuncunun hem de bilgisayarın skorları güncel olarak gösterilmelidir.
    - **Oyuncu Skoru Etiketi**: Oyuncunun skorunu gösteren bir etiket, sol panelde yer almalıdır.
    - **Bilgisayar Skoru Etiketi**: Bilgisayarın skorunu gösteren bir etiket, sağ panelde yer almalıdır.

UYGULAMA

```
import random
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
import os
```

Bu kod kısmı, Python'da çeşitli kütüphaneleri ve modülleri içe aktarıyor. Bu kütüphaneler ve modüller, Pokémon Kart Oyunu projesinde farklı işlevler için kullanılacaktır.

Açıklamalar

1. **random**:
   - `import random`
   - Bu kütüphane, rastgele sayı üretmek ve liste elemanlarını rastgele karıştırmak gibi işlevler için kullanılır. Örneğin, desteyi karıştırmak veya bilgisayarın rastgele bir kart seçmesini sağlamak için kullanılabilir.
2. **tkinter**:
   - `import tkinter as tk`
   - `from tkinter import messagebox`
   - Tkinter, Python'un standart GUI kütüphanesidir ve kullanıcı arayüzünü (GUI) oluşturmak için kullanılır. `tk` takma adıyla içe aktarılmıştır, bu da Tkinter bileşenlerine daha kısa adlarla erişimi sağlar. `messagebox`, Tkinter'da kullanıcıya mesaj kutuları göstermek için kullanılır (örneğin, tur veya oyun sonuçlarını göstermek için).
3. **PIL (Pillow)**:
   - `from PIL import Image, ImageTk`
   - Pillow, Python Imaging Library'nin (PIL) bir çatalıdır ve resim dosyalarını işlemek için kullanılır. Bu modüller, resim dosyalarını açmak, yeniden boyutlandırmak ve Tkinter'da göstermek için gereklidir. `Image` modülü resimleri açmak ve işlemek için kullanılırken, `ImageTk` modülü bu resimleri Tkinter ile kullanılabilir hale getirir.
4. **os**:
   - `import os`
   - Bu kütüphane, işletim sistemi ile etkileşime geçmek için kullanılır. Örneğin, dosya ve dizin yollarını kontrol etmek için kullanılır.

Projede, resim dosyalarının mevcut olup olmadığını kontrol etmek için kullanılabilir.

```
5. class Card:
       def __init__(self, name="Unknown", damage=0):
           self._name = name
           self._damage = damage

       def __str__(self):
           return f"{self._name} (Damage: {self._damage})"

       @property
       def name(self):
           return self._name

       @name.setter
       def name(self, value):
           self._name = value

       @property
       def damage(self):
           return self._damage

       @damage.setter
       def damage(self, value):
           self._damage = value

       def get_image_path(self):
           return f"images/{self._name.lower()}.jpg"

   class PokemonCard(Card):
       def __init__(self, name="Unknown", damage=0):
           super().__init__(name, damage)
```

Bu kod kısmı, Pokémon Kart Oyunu projesinde kartları temsil eden sınıfları tanımlamaktadır. Bu sınıflar, kartların özelliklerini ve davranışlarını kapsüller ve oyunun temel bileşenlerinden birini oluşturur.

Sınıfların Açıklamaları

1. **Card Sınıfı**:
   - Bu sınıf, genel bir kartı temsil eder ve kartın adını (`name`) ve hasar puanını (`damage`) içerir.

2. `__init__` Metodu (Constructor):

- Kart nesnesi oluşturulduğunda, bu metod çalışır ve kartın adını ve hasar puanını belirler.
- `name` ve `damage` varsayılan olarak "Unknown" ve 0 değerlerini alır, ancak kart oluşturulurken farklı değerler de verilebilir.

- `__str__` Metodu:
  - Bu metod, kartın adını ve hasar puanını içeren bir string döner.

- Örneğin, bir kartın adı "Bulbasaur" ve hasar puanı 70 ise, "`Bulbasaur (Damage: 70)`" döner.

- @property Dekoratörleri ve Getter/Setter Metotları:
  - Bu dekoratörler ve metotlar, kapsülleme (encapsulation) ilkesini kullanarak kartın adı ve hasar puanına kontrollü erişim sağlar.
  - `get_image_path` Metodu:
    - Bu metod, kartın görsel dosya yolunu döner.
    - Örneğin, kartın adı "Bulbasaur" ise, "`images/bulbasaur.jpg`" döner.
    - Bu, kart görsellerini dinamik olarak yüklemek için kullanılır.

2. **PokemonCard Sınıfı**:
   - Bu sınıf, `Card` sınıfından miras alır ve Pokémon kartlarını temsil eder.

`__init__` Metodu (Constructor):

- `Card` sınıfının `__init__` metodunu çağırarak (`super().__init__(name, damage)`), `PokemonCard` nesnesi oluşturulur.
- `name` ve `damage` varsayılan olarak "Unknown" ve 0 değerlerini alır, ancak kart oluşturulurken farklı değerler de verilebilir.

```
class Player:
    def __init__(self, name="Player"):
        self._name = name
        self._hand = []
        self._score = 0

    def draw_card(self, deck):
        if deck:
            self._hand.append(deck.pop())

    def play_card(self, index):
        if self._hand:
            return self._hand.pop(index)
        return None

    def __str__(self):
        return f"{self._name}: Score = {self._score}"

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def hand(self):
        return self._hand

    @hand.setter
```

```
        def hand(self, value):
            self._hand = value

        @property
        def score(self):
            return self._score

        @score.setter
        def score(self, value):
            self._score = value

    class HumanPlayer(Player):
        def __init__(self, name="Player"):
            super().__init__(name)

    class ComputerPlayer(Player):
        def __init__(self, name="Computer"):
            super().__init__(name)
```

Bu kod, Pokémon Kart Oyunu'ndaki oyuncuları temsil eden sınıfları tanımlar. `Player` sınıfı, oyuncunun adı, elindeki kartlar ve skor gibi temel oyuncu özelliklerini kapsar ve bu özelliklere güvenli erişim sağlar. `Player` sınıfının `__init__` metodu, oyuncu nesnesi oluşturulduğunda oyuncunun adını, elindeki kartları ve skorunu belirler. `draw_card` metodu, desteden bir kart çekerek oyuncunun eline eklerken, `play_card` metodu oyuncunun elindeki belirtilen indeksteki kartı oynar ve bu kartı elden çıkarır. `__str__` metodu, oyuncunun adını ve skorunu içeren bir string döner. Bu sınıfta ayrıca `name`, `hand` ve `score` özelliklerine doğrudan erişimi sağlayan getter ve setter metotları bulunur, bu da kapsülleme (encapsulation) ilkesine uygun olarak verilerin güvenliğini ve bütünlüğünü korur.

`HumanPlayer` ve `ComputerPlayer` sınıfları, `Player` sınıfından türetilmiştir ve sırasıyla insan ve bilgisayar oyuncularını temsil eder. Bu sınıflar, `Player` sınıfının tüm özelliklerini ve metodlarını miras alır. `HumanPlayer` sınıfı, insan oyuncu nesnelerini oluşturur ve `__init__` metodu aracılığıyla oyuncunun adını belirler. Benzer şekilde, `ComputerPlayer` sınıfı da bilgisayar oyuncu nesnelerini oluşturur ve `__init__` metodu aracılığıyla oyuncunun adını belirler.

Bu yapı, oyundaki oyuncuların yönetilmesi, kart çekme ve oynama işlemlerinin gerçekleştirilmesi için gerekli olan temel işlevselliği sağlar. Oyunun başında, oyuncuların desteden kart çekmesi ve her turda belirli bir kartı oynaması bu sınıflar aracılığıyla gerçekleştirilir. Ayrıca, oyuncuların skorları güncellenir ve oyun sonunda kazanan belirlenir. Bu sınıflar, oyun mantığının uygulanmasında önemli bir rol oynar ve oyuncuların oyunla etkileşime girmesini sağlar.

```
class Game:
    def __init__(self):
```

```
        self.deck = self.create_deck()
        self.player = HumanPlayer("Player")
        self.computer = ComputerPlayer("Computer")
        self.current_cards = []
        self.player_card_index = None
        self.window = tk.Tk()
        self.window.title("Pokemon Kart Oyunu")
        self.create_widgets()
        self.start_game()

    def create_deck(self):
        names = ["Bulbasaur", "Charmander", "Squirtle", "Pikachu",
"Jigglypuff",
                 "Meowth", "Psyduck", "Snorlax", "Gengar", "Eevee"]
        damages = [70, 50, 40, 200, 70, 40, 50, 90, 130, 40]
        deck = [PokemonCard(names[i], damages[i]) for i in range(10)]
        random.shuffle(deck)
        return deck

    def shuffle_deck(self):
        random.shuffle(self.deck)

    def deal_cards(self):
        for _ in range(3):
            self.player.draw_card(self.deck)
            self.computer.draw_card(self.deck)

    def play_round(self):
        player_card = self.player.play_card(self.player_card_index)
        computer_card = self.computer.play_card(random.randint(0,
len(self.computer.hand) - 1))
        self.current_cards = [player_card, computer_card]

        self.update_card_images(player_card, computer_card)

        if player_card.damage > computer_card.damage:
            self.player.score += 5
            result_text = "Oyuncu bu raundu kazandı!"
        elif player_card.damage < computer_card.damage:
            self.computer.score += 5
            result_text = "Bilgisayar bu raundu kazandı!"
        else:
            result_text = "Berabere!"

        messagebox.showinfo("Raund Sonucu", result_text)
        self.update_scores()
        self.update_hand()

        if not self.deck and not self.player.hand and not self.computer.hand:
            self.end_game()
        else:
            self.play_button.config(state=tk.DISABLED)

    def start_game(self):
        self.deal_cards()
        self.update_hand()
        self.update_scores()

    def end_game(self):
        if self.player.score > self.computer.score:
            winner = "Oyuncu oyunu kazandı!"
        elif self.player.score < self.computer.score:
            winner = "Bilgisayar oyunu kazandı!"
        else:
            winner = "Oyun berabere bitti!"

        messagebox.showinfo("Oyun Bitti", winner)
```

```
        self.window.quit()

    def create_widgets(self):
        self.player_frame = tk.LabelFrame(self.window, text="Oyuncu")
        self.player_frame.pack(side="left", padx=10, pady=10)

        self.computer_frame = tk.LabelFrame(self.window, text="Bilgisayar")
        self.computer_frame.pack(side="right", padx=10, pady=10)

        self.player_card_image = tk.Label(self.player_frame)
        self.player_card_image.pack(pady=20)

        self.computer_card_image = tk.Label(self.computer_frame)
        self.computer_card_image.pack(pady=20)

        self.play_button = tk.Button(self.window, text="Raund Oyna",
command=self.play_round, state=tk.DISABLED)
        self.play_button.pack(pady=20)

        self.player_score_label = tk.Label(self.window, text="Oyuncu Skoru:
0")
        self.player_score_label.pack(side="left", padx=10)

        self.computer_score_label = tk.Label(self.window, text="Bilgisayar
Skoru: 0")
        self.computer_score_label.pack(side="right", padx=10)

        self.player_hand_frame = tk.Frame(self.player_frame)
        self.player_hand_frame.pack()

    def update_hand(self):
        for widget in self.player_hand_frame.winfo_children():
            widget.destroy()

        for i, card in enumerate(self.player.hand):
            card_button = tk.Button(self.player_hand_frame, text=str(card),
command=lambda i=i: self.select_card(i))
            card_button.pack(side="left")

        if self.deck:
            self.player.draw_card(self.deck)
            self.computer.draw_card(self.deck)

    def select_card(self, index):
        self.player_card_index = index
        self.play_button.config(state=tk.NORMAL)

    def update_card_images(self, player_card, computer_card):
        player_img_path = player_card.get_image_path()
        computer_img_path = computer_card.get_image_path()
        fixed_size = (200, 300)  # Set the size for the images

        print(f"Checking paths:\nPlayer image path:
{player_img_path}\nComputer image path: {computer_img_path}")

        if not os.path.exists(player_img_path):
            print(f"File not found: {player_img_path}")
        if not os.path.exists(computer_img_path):
            print(f"File not found: {computer_img_path}")

        try:
            player_img = Image.open(player_img_path).resize(fixed_size,
Image.LANCZOS)
        except (FileNotFoundError, IOError) as e:
            messagebox.showerror("Hata", f"Player resmi yüklenemedi: {e}")
            self.window.quit()
```

```
        try:
            computer_img = Image.open(computer_img_path).resize(fixed_size,
Image.LANCZOS)
        except (FileNotFoundError, IOError) as e:
            messagebox.showerror("Hata", f"Computer resmi yüklenemedi: {e}")
            self.window.quit()

        player_img = ImageTk.PhotoImage(player_img)
        computer_img = ImageTk.PhotoImage(computer_img)

        self.player_card_image.configure(image=player_img)
        self.player_card_image.image = player_img

        self.computer_card_image.configure(image=computer_img)
        self.computer_card_image.image = computer_img

    def update_scores(self):
        self.player_score_label.config(text=f"Oyuncu Skoru:
{self.player.score}")
        self.computer_score_label.config(text=f"Bilgisayar Skoru:
{self.computer.score}")

    def play_game(self):
        self.window.mainloop()

if __name__ == "__main__":
    game = Game()
    game.play_game()
```

Bu kod, Pokémon Kart Oyunu'nu gerçekleştiren `Game` sınıfını tanımlar ve oyunun tüm işlevselliğini sağlar. `__init__` metodu, oyun destesi oluşturur (`create_deck`), insan oyuncu (`HumanPlayer`) ve bilgisayar oyuncu (`ComputerPlayer`) nesnelerini başlatır, oyun sırasında oynanan kartları saklamak için bir liste oluşturur, oyuncunun seçtiği kartın indeksini tutacak değişkeni başlatır, Tkinter ana penceresini (`tk.Tk`) ve başlığını oluşturur, GUI bileşenlerini oluşturur (`create_widgets`) ve oyunu başlatır (`start_game`). `create_deck` metodu, on farklı Pokémon kartını adları ve hasar puanları ile birlikte oluşturur ve desteyi karıştırır. `shuffle_deck` metodu, mevcut desteyi rastgele karıştırır, `deal_cards` metodu ise her oyuncuya başlangıçta üçer kart dağıtır.

`play_round` metodu, oyuncunun seçtiği kartı ve bilgisayarın rastgele seçtiği kartı oynar, bu kartları karşılaştırır ve hasar puanlarına göre tur sonucunu belirler. Oyuncunun kartı daha yüksek hasar puanına sahipse oyuncuya, bilgisayarın kartı daha yüksek hasar puanına sahipse bilgisayara puan ekler. Tur sonucunu bir mesaj kutusunda gösterir ve ardından skorları (`update_scores`) ve oyuncunun elindeki kartları (`update_hand`) günceller. Eğer destede ve oyuncuların ellerinde kart kalmadıysa oyunu sonlandırır (`end_game`), aksi takdirde tur oynama butonunu devre dışı bırakır.

`start_game` metodu, başlangıçta kartları dağıtır, el ve skorları günceller. `end_game` metodu, oyuncu ve bilgisayarın skorlarını karşılaştırarak

oyunun kazananını belirler ve bir mesaj kutusunda sonucu gösterir, ardından oyunu sonlandırır. `create_widgets` metodu, Tkinter kullanarak kullanıcı arayüzü bileşenlerini oluşturur; bunlar arasında oyuncu ve bilgisayar için ayrı paneller, kart görsellerini göstermek için etiketler, tur oynama butonu ve skor etiketleri yer alır. `update_hand` metodu, oyuncunun elindeki kartları günceller ve her kart için bir buton oluşturur, oyuncunun yeni bir kart çekmesini sağlar. `select_card` metodu, oyuncunun seçtiği kartın indeksini belirler ve tur oynama butonunu etkinleştirir.

`update_card_images` metodu, oynanan kartların görsellerini günceller. Kart görselleri ilgili dosya yolundan yüklenir ve boyutlandırılır. Eğer dosya bulunamazsa hata mesajı gösterilir ve program sonlandırılır. `update_scores` metodu, oyuncu ve bilgisayarın skor etiketlerini günceller. `play_game` metodu, Tkinter ana döngüsünü başlatarak kullanıcı arayüzünün etkileşimli olmasını sağlar. Programın ana kısmında (`if __name__ == "__main__":`), `Game` sınıfı bir nesne olarak oluşturulur ve `play_game` metodu çağrılarak oyun başlatılır. Bu yapı, oyunun başlatılmasını, oynanmasını ve tamamlanmasını sağlayan tüm işlevselliği kapsar ve oyunculara interaktif bir deneyim sunar.

Görev Dağılımı

- Kodun tasarım kısmını birlikte oluşturduk.Oyunun tasarım kısmının birazını Amine birazını Nazlı yaptı.Kullanılması gereken

fonksiyonları ve metodları Nazlı araştırdı. Arayüzü ve Pokemon kartlarını ortaklaşa düzenledik ve kodumuza entegre ettik.

- Raporun hazırlarken ortaklaşa çalıştık.Giriş fonksiyonel analiz kısmını Amine oluşturdu.Kalan kısımları da Nazlı yaptı.

Karşılaşılan Zorluklar ve Çözüm Yöntemleri

Zorluklar ve Çözüm Yöntemleri

GUI ile Oyun Mantığını Entegre Etmek:

- Zorluk: Tkinter gibi bir GUI kütüphanesi kullanarak oyun mantığını entegre etmek bazen karmaşık olabilir. Oyun mantığı ile GUI olaylarını senkronize etmek zor olabilir.
- Çözüm: GUI ile iş mantığını ayrı sınıflar ve metotlar kullanarak modüler hale getirin. Bu, her bileşenin (örneğin, kart çekme, puan güncelleme) kendi başına test edilebilmesini sağlar ve GUI olay işleyicilerinin iş mantığını çağırmasını kolaylaştırır.

Resim Dosyalarının Yüklenmesi:

- Zorluk: Resim dosyalarının doğru yolda bulunmaması veya yüklenememesi.
- Çözüm: Resim dosyalarının doğru yolda olduğundan emin olun. Ayrıca, dosya yollarının doğruluğunu kontrol eden ve hata durumunda kullanıcıya bilgi veren hata işleme mekanizmaları ekleyin. Kodda aşağıdaki gibi hata ayıklama mesajları kullanın:

python
Kodu kopyala
```
if not os.path.exists(player_img_path):
    print(f"File not found: {player_img_path}")
```

- GUI Öğelerinin Güncellenmesi:

- Zorluk: Oyuncu kartları ve skor gibi GUI öğelerini dinamik olarak güncellemek karmaşık olabilir.
- Çözüm: Tkinter widget'larını güncellemeye yönelik yardımcı metotlar oluşturun. Örneğin, update_hand ve update_scores gibi metotlar, GUI öğelerini güncellemek için merkezi bir yer sağlar.

Oyun Durumunun Yönetimi:

- Zorluk: Oyun durumunu (oyuncuların elindeki kartlar, skorlar, destede kalan kartlar) yönetmek ve güncellemek zor olabilir.
- Çözüm: Oyun durumunu yönetmek için merkezi bir Game sınıfı kullanın ve bu sınıfın sorumluluğunda olacak şekilde durumu güncelleyen metotlar yazın. Bu sınıf, oyun durumunu tutarlı ve yönetilebilir kılmak için merkezi bir kontrol noktası sağlar.

Kullanıcı Etkileşimlerinin Doğru İşlenmesi:

- Zorluk: Kullanıcı etkileşimlerini doğru şekilde işlemek ve kullanıcı hatalarını önlemek.
- Çözüm: Kullanıcı etkileşimlerini sınırlamak ve doğrulamak için GUI bileşenlerinin durumlarını kontrol eden (örneğin, play_buttonun durumunu yönetmek) mantık ekleyin. Bu, kullanıcıların yanlış zamanda yanlış işlemler yapmasını engeller.

Unit Testlerin GUI İle Çalışması:

- Zorluk: Tkinter gibi GUI kütüphaneleriyle çalışan kodun unit test edilmesi zordur.
- Çözüm: İş mantığını ve GUI mantığını ayırarak iş mantığını bağımsız olarak test edin. Ayrıca, GUI bileşenlerini test etmek için mock kütüphanelerini veya GUI otomasyon araçlarını kullanabilirsiniz.

Özet
Bu zorluklarla başa çıkmak için yazılım geliştirme sürecinde iyi bir tasarım uygulamak ve modüler kod yazmak çok önemlidir. İş mantığını ve GUI bileşenlerini ayrı sınıflar ve modüller halinde düzenlemek, kodun okunabilirliğini ve test edilebilirliğini artırır. Ayrıca, hataları önlemek ve çözmek için uygun hata işleme ve kullanıcı doğrulama tekniklerini kullanmak da kritik öneme sahiptir.

Proje isterlerine göre eksik yönler

Eksiklik: Kartların kapalı bir şekilde konulup sonra açılması gerekmektedir. Şu anda kartlar hemen gösteriliyor. Öneri: Kapalı kart görselleri eklenmeli ve kartlar açıldığında değiştirilmeli. Destede Ortada Kalan 4 Kart:

Eksiklik: İlk kart dağıtımından sonra ortada 4 kartın kalması gerekmektedir. Şu anda bu durum kodda yönetilmemektedir. Öneri: Kartlar dağıtıldıktan sonra ortada kalan 4 kartın tutulduğu bir yapı eklenmelidir.,

Eksiklik: Oyun döngüsü tamamlanmamış. Kartlar ortaya konduktan sonra tekrar desteden kart çekilmesi ve oyun devamı eksik. Öneri: Her raund sonunda oyuncuların ve bilgisayarın ortada kalan kartlardan rastgele bir kart çekmesini sağlayan kod eklenmelidir.

TEST VE DOĞRULAMA
Yazılımın test süreci:

Test Senaryoları
Kart Oluşturma Testi:

Kart nesnelerinin doğru şekilde oluşturulduğunu ve özelliklerinin doğru ayarlandığını doğrulayın.
Oyuncu Testleri:

Oyuncuların (kullanıcı ve bilgisayar) kart çekme ve oynama işlemlerini doğru şekilde gerçekleştirdiğini doğrulayın.
Oyun Başlangıç Testi:

Oyunun doğru şekilde başladığını ve kartların dağıtıldığını doğrulayın.
Raund Oynama Testi:

Raund oynama işleminin doğru şekilde gerçekleştirildiğini doğrulayın, skorların güncellendiğini ve kazananın doğru belirlendiğini kontrol edin.
Oyun Sonu Testi:

Oyun sonlandığında doğru kazananın belirlendiğini ve doğru mesajın gösterildiğini doğrulayın.
Test Kodları
python
Kodu kopyala

```
import unittest
from game import Card, PokemonCard, HumanPlayer, ComputerPlayer, Game

class TestPokemonGame(unittest.TestCase):
    def setUp(self):
        self.deck = [PokemonCard("Bulbasaur", 70), PokemonCard("Charmander", 
50), PokemonCard("Squirtle", 40)]
        self.player = HumanPlayer("Player")
        self.computer = ComputerPlayer("Computer")
        self.game = Game()

    def test_card_creation(self):
        card = PokemonCard("Pikachu", 200)
        self.assertEqual(card.name, "Pikachu")
        self.assertEqual(card.damage, 200)
        self.assertEqual(card.get_image_path(), "images/pikachu.jpg")

    def test_player_creation(self):
        self.assertEqual(self.player.name, "Player")
        self.assertEqual(len(self.player.hand), 0)
        self.assertEqual(self.player.score, 0)

    def test_draw_card(self):
        self.player.draw_card(self.deck)
        self.assertEqual(len(self.player.hand), 1)
```

```
        self.assertEqual(len(self.deck), 2)

    def test_play_card(self):
        self.player.draw_card(self.deck)
        card = self.player.play_card(0)
        self.assertEqual(len(self.player.hand), 0)
        self.assertEqual(card.name, "Bulbasaur")

    def test_deck_creation(self):
        deck = self.game.create_deck()
        self.assertEqual(len(deck), 10)

    def test_shuffle_deck(self):
        original_deck = self.game.create_deck()
        self.game.shuffle_deck()
        self.assertNotEqual(self.game.deck, original_deck)

    def test_deal_cards(self):
        self.game.deal_cards()
        self.assertEqual(len(self.game.player.hand), 3)
        self.assertEqual(len(self.game.computer.hand), 3)

    def test_play_round(self):
        self.game.mode = "PlayerVsComputer"
        self.game.deal_cards()
        self.game.player_card_index = 0
        self.game.play_round()
        self.assertEqual(len(self.game.player.hand), 3)
        self.assertEqual(len(self.game.computer.hand), 3)
        self.assertTrue(self.game.player.score >= 0)
        self.assertTrue(self.game.computer.score >= 0)

    def test_end_game(self):
        self.game.player.score = 10
        self.game.computer.score = 5
        with self.assertRaises(SystemExit):  # Expect the program to exit after
showing the end game messagebox
            self.game.end_game()

if __name__ == "__main__":
    unittest.main()
```

Açıklamalar

Test Senaryoları: Her bir test senaryosu için ilgili test metodları oluşturulmuştur (test_card_creation, test_player_creation, test_draw_card, test_play_card, test_deck_creation, test_shuffle_deck, test_deal_cards, test_play_round, test_end_game).

Setup: setUp metodu, her bir testten önce çalıştırılır ve gerekli nesnelerin (desteler, oyuncular, oyun) oluşturulmasını sağlar.
Unit Testler: Her bir test, ilgili fonksiyonların ve sınıfların beklenen şekilde çalışıp çalışmadığını doğrular.

Bu test kodlarını çalıştırarak, projenizdeki hataları erken tespit edebilir ve düzeltmeler yapabilirsiniz. Unutmayın ki GUI ile ilgili kısımları test etmek daha karmaşık olabilir ve genellikle unittest yerine daha özel test araçları gerektirebilir. Bu örnekte temel iş mantığını test etmeye odaklandık.

KAYNAKÇA:

ChatGPT

https://kamer-dev.medium.com/oopnin-4-ana-prensibi-encapsulation-inheritence-abstraction-polymorphism-712ed2fbac7e

https://bidb.itu.edu.tr/seyir-defteri/blog/2019/02/05/object-oriented-programming