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

LORD OF THE POLYWARPHISM

NURDAN BULUT
220502021

ZEYNEP KEDİKLİ
220501012

DERS SORUMLUSU:
DR. ÖĞR. ÜYESİ ERCAN ÖLÇER

TARİH:24.03.2023

1 GİRİŞ
1.1 Projenin amacı

Bu proje, çoklu oyuncuların 2 boyutlu bir dünyada stratejik savaşlar yapabileceği bir oyunun geliştirilmesini amaçlamaktadır. Oyuncular, belirlenen dünya boyutlarında savaşçılarını stratejik konumlara yerleştirerek diğer oyunculara karşı mücadele edeceklerdir. Tüm savaşçılar bir sınıftan türetilmelidir ve çok-biçimlilik (polymorphism) kullanılması amaçlanmaktadır. Nesneye yönelik programlama ve iyi yazılım geliştirme uygulamalarını benimseyerek yapılmalıdır.

1.2 Projede Beklenenler

- Kullanıcıların seçebileceği boyutlarda (16x16, 24x24, 32x32 veya kullanıcı tanımlı) bir oyun dünyası oluşturulacaktır.
- Oyunda en az 1 gerçek oyuncu olmak üzere maksimum 4 oyuncu bulunabilir.
- Her oyuncu bir el içinde en fazla 2 savaşçı oluşturabilir.
- Savaşçılar belirli kurallara göre yerleştirilir ve savaşçı seçimi yapılır.
- Oyuncular savaşçıları matris üzerinde belirli koordinatlara yerleştirir.
- Yeni savaşçıların yerleştirileceği yer boş olmalı veya aynı oyuncunun farklı bir savaşçısının yanında olmalıdır.
- Eğer yeni bir savaşçı eski bir savaşçının üzerine yerleştirilirse, eski savaşçı yok edilir ve bir kısmı hazineye aktarılır.
- Oyuncular, her elde belirli miktarlarda kaynak kazanırlar.
- Oyuncuların eli pas geçebilir ve bu isteğe bağlıdır.
- Oyuncular, dünyada kalan savaşçıların %60'ını ele geçiren veya dünyada sona kalan oyuncu olmak için rekabet ederler.
- Bir oyuncu oyunu kazanmak için belirli koşulları sağlamalıdır, aksi takdirde oyunu kaybeder.
- Dünya, 2 boyutlu bir standart vektör olarak oluşturulmalıdır.
- Matristeki durum oyuncuların her hamlesi sonunda ve tur sonunda gerçekleşen saldırılardan sonra ekranda gösterilecektir.
- 1. Oyuncu kırmızı, 2. Oyuncu mavi, 3. Oyuncu yeşil ve 4. Oyuncu sarı renk ile gösterilmelidir.
- Her ekran güncellemesinde ekran temizlenmeli ve tüm dünya baştan ekrana basılmalıdır.
- Yeni savaşçılar yerleştirilirken o an yerleştirilebilecek muhtemel yerler gösterilecek.
- Her oyuncunun savaşçılarını yerleştirebileceği alanlar, sırasıyla oyuncuların renkleriyle belirtilecek.
- Oyuncuların ikinci savaşçılarını yerleştirdiği durumda, bu yeni savaşçıların yerleştirilebileceği alanlar da önceden yerleştirilen savaşçıların konumları ve renk kodları dikkate alınarak gösterilecek.
- Tüm savaşçılar bir sınıftan türetilmelidir ve çok-biçimlilik (polymorphism) kullanılarak gerçeklenmelidir.
- Projenin tüm bölümlerinde nesneye yönelik programlama yaklaşımı benimsenmeli ve iyi yazılım geliştirme pratiklerine uyulmalıdır.

2 GEREKSİNİM ANALİZİ

2.1 Arayüz gereksinimleri

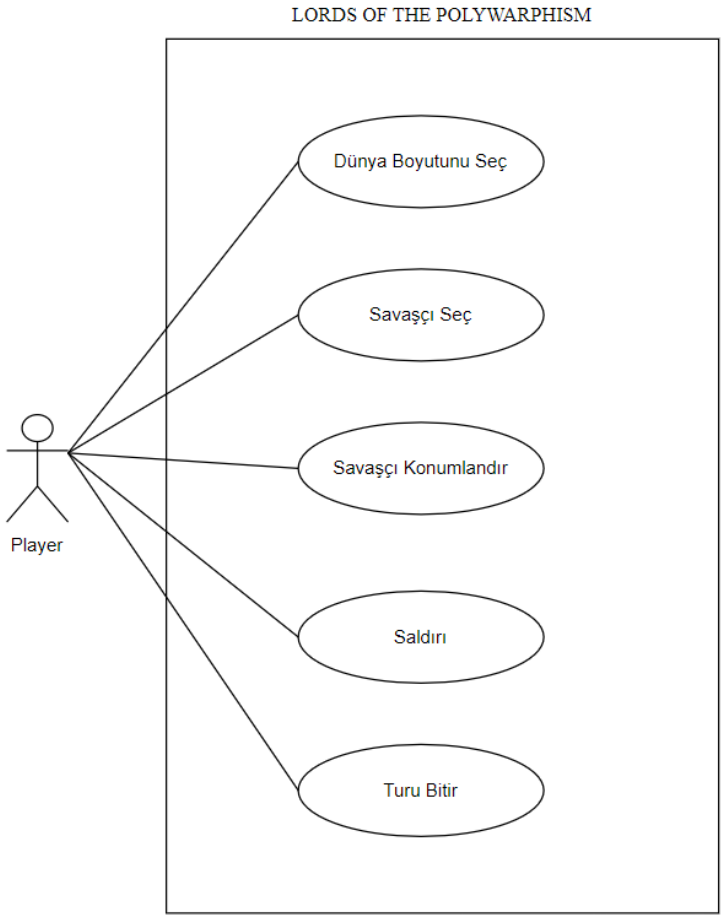
- Tahta Boyutu Seçimi:
  - Kullanıcı, oyun tahtasının boyutunu belirlemek için bir pencereye sahip olmalıdır.
  - Bu pencere, 8 ile 32 arasında bir değer girmesine izin verir.
  - Kullanıcı, belirlenen aralıkta bir değer girdiğinde, bu değeri onaylamak için bir düğme kullanabilir.
- Oyuncu Sayısının Belirlenmesi:
  - Kullanıcı, oyuncu sayısını belirlemek için bir pencereye sahip olmalıdır.
  - Bu pencere, 1 ile 4 arasında bir oyuncu sayısı girmesine izin verir.
  - Kullanıcı, belirlenen aralıkta bir değer girdiğinde, bu değeri onaylamak için bir düğme kullanabilir.
- Oyun Arayüzü:
  - Oyun başladığında, kullanıcıya bir oyun arayüzü sunulmalıdır.
  - Oyun arayüzü, oyuncuların sahip oldukları kaynakları, savaşçıları ve tahta durumunu gösterir.
  - Arayüz, savaşçı yerleştirme ve diğer oyun etkileşimlerini sağlamak için gerekli kontrolleri içermelidir.
- Savaşçı Yerleştirme:
  - Kullanıcı, oyuncu sırası kendisine geldiğinde savaşçıları tahtaya yerleştirebilmelidir.
  - Yerleştirme işlemi, kullanıcının bir hücreye tıklaması ve bir savaşçı türü seçmesiyle gerçekleşmelidir.
- Geçiş Yapma ve Eylemler:
  - Kullanıcı, sırası geldiğinde geçiş yapabilmeli ve sıradaki oyuncuya geçebilmelidir.
  - Oyuncu, sırası geldiğinde, bir tur içinde yapabileceği eylem sayısını sınırlayan belirli bir eylem limitine sahip olmalıdır.

2.2 Fonksiyonel gereksinimler

- Oyun Alanı Oluşturma:
  - Oyuncuların savaşçılarını yerleştirebilecekleri bir oyun tahtası oluşturulmalıdır.
  - Tahta, belirli bir boyuta sahip olmalıdır.
  - Oyun alanı, hücrelerden oluşmalıdır ve her hücre bir savaşçıyı veya boşluğu temsil etmelidir.
- Savaşçı Yerleştirme:
  - Oyuncular, savaşçılarını tahtaya yerleştirebilmelidir.
  - Her oyuncu, kendi savaşçılarını sadece belirli koşullarda belirli bir hücreye yerleştirebilmelidir.
  - Savaşçılar, oyuncunun sahip olduğu kaynaklara göre yerleştirilebilmelidir.
- Savaşçıların Hareketi ve Saldırısı:
  - Oyuncuların savaşçıları, belirli bir menzile sahip olmalıdır.

    - Savaşçılar, belirli bir menzildeki düşman savaşçılarına saldırabilmelidir.
    - Savaşçılar, belirli bir menzildeki dost savaşçılarına yardım edebilmelidir.
    - Savaşçılar, saldırılarını gerçekleştirdikten sonra düşman veya dost savaşçılarının sağlık puanlarını güncelleyebilmelidir.
- Oyun Akışı:
    - Oyun, tur tabanlı olmalıdır.
    - Her oyuncu, sırası geldiğinde belirli eylemleri gerçekleştirebilmelidir.
    - Bir oyuncu, sırası geldiğinde eylem yapmadan da turunu geçirebilmelidir.
- Oyuncu Kontrolleri ve Bilgilendirmeler:
    - Oyuncuların savaşçılarını yerleştirirken veya eylem yaparken hatalı girişler konusunda bilgilendirilmeleri sağlanmalıdır.
    - Oyuncuların sağlık puanları, kaynak miktarları ve sıra bilgileri görüntülenmelidir.
- Oyun Bitiş Koşulları:
    - Oyun, bir oyuncunun belirli bir miktarda alanı ele geçirmesiyle veya diğer oyuncuların eliminasyonuyla sona ermelidir.
    - Oyunun sona erdiğinde, kazanan oyuncu veya oyuncular ilan edilmelidir.

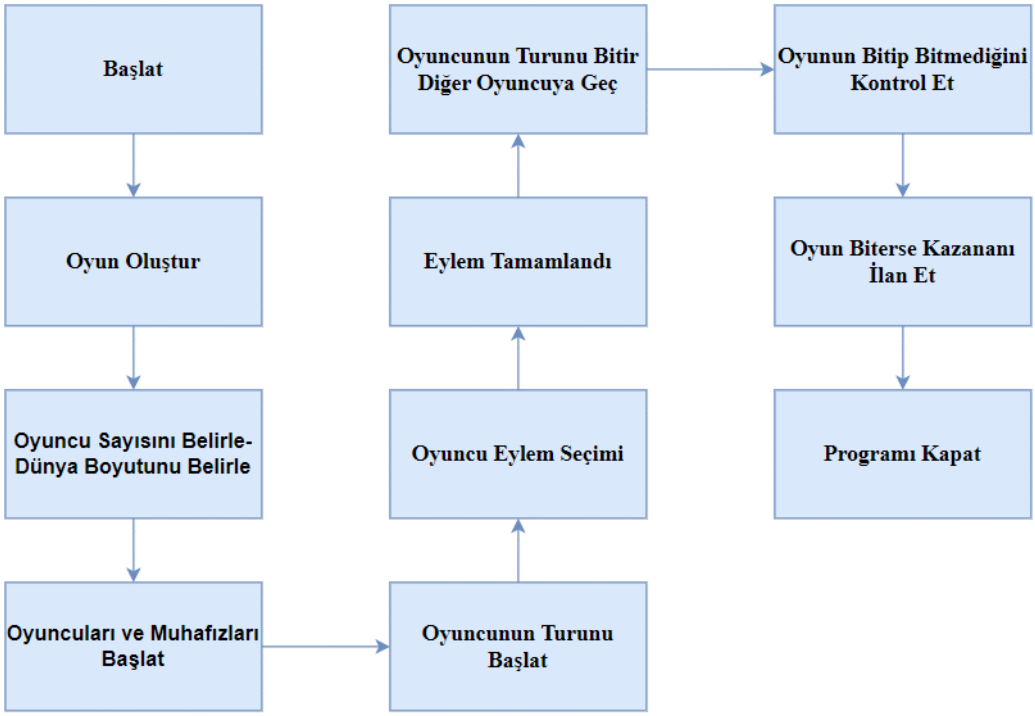
2.3 Use-Case diyagramı



3 TASARIM
3.1 Mimari tasarım

- Warrior Sınıfları:
    - Projede, farklı savaşçıları temsil etmek için Warrior sınıfları kullanılmıştır.
    - Her savaşçı, belirli bir saldırı gücü, sağlık, menzil ve maliyet değerlerine sahiptir.
    - Savaşçı sınıfları, Warrior sınıfından türetilmiş ve özelleştirilmiştir.
- Player ve Game Sınıfları:
    - Player sınıfı, oyuncuları temsil eder ve oyuncunun sahip olduğu kaynakları, sağlık durumunu ve savaşçılarını içerir.
    - Game sınıfı, oyunun temel mantığını ve oyun akışını yönetir.
    - Oyun durumu ve oyuncular arasındaki etkileşimleri kontrol eder.
- Kullanıcı Arayüzü (GUI) Sınıfları:
    - GameGUI sınıfı, tkinter kütüphanesi kullanılarak oluşturulmuş grafik kullanıcı arayüzünü (GUI) yönetir.
    - Kullanıcı arayüzü, oyuncuların oyun tahtasını görmesini, savaşçılarını yerleştirmesini ve tur kontrolünü gerçekleştirmesini sağlar.
- İşlem Yürütme Metodları:
    - Her savaşçı sınıfı, perform_action() metodunu uygular.
    - Bu metodlar, belirli bir savaşçının belirli bir eylemi gerçekleştirmesini sağlar (örneğin, saldırı yapma).
    - Eylemler, savaşçıların etkileşimlerini ve oyunun ilerleyişini yönetir.
- Oyun Tahtası ve Hücre Yönetimi:
    - Oyun tahtası, bir sözlük yapısı kullanılarak temsil edilir.
    - Sözlük, (x, y) koordinatlarını anahtar olarak alır ve o hücredeki savaşçı veya boş olduğunu belirtir.
    - Savaşçılar hareket ettirildiğinde veya yerleştirildiğinde, oyun tahtası güncellenir ve kullanıcı arayüzü yeniden çizilir.

- Modül diyagramı



3.2 Kullanılacak teknolojiler

- Proje Python programlama dili kullanılarak yazılmıştır.
- Projede kullanıcı arayüzü (GUI) uygulamaları oluşturmak için Tkinter harici kütüphanesi kullanılmıştır.
- Oyun Tahtası:
  - Oyun tahtası, bir ızgara olarak tasarlanmıştır.
  - Her bir hücre, tk.Button bileşenleri kullanılarak oluşturulmuştur.
  - Oyun tahtasındaki her hücre, oyun dünyasının bir parçasını temsil eder.
  - Oyuncular, savaşçılarını bu tahta üzerine yerleştirirler.
- Oyuncu Bilgi Bölümü:
  - Her oyuncunun bilgilerini göstermek için birer bölüm oluşturulmuştur.
  - Bu paneller, tk.Frame bileşenleri içinde tk.Label bileşenleri kullanılarak tasarlanmıştır.
  - Her oyuncunun sağlık durumu ve kaynak miktarı bu panellerde gösterilir.
- Savaşçı Seçim Penceresi:
  - Oyuncular, savaşçılarını yerleştirecekleri hücreyi seçmek için bir pencere kullanırlar.
  - Bu pencere, tk.Toplevel bileşeni kullanılarak oluşturulmuştur.
  - Kullanıcılar, bir savaşçı türü seçmek için bir açılır menüyü kullanabilirler (tk.OptionMenu).
  - Seçim yapıldıktan sonra, seçim penceresi kapatılarak kullanıcının seçimi işlenir.

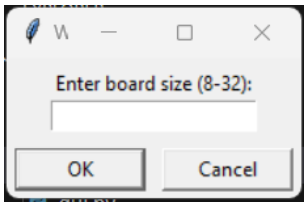
- Diğer Bileşenler:
    - Oyuncuların tur geçme ve oyunu sonlandırma işlemleri için düğmeler kullanılmıştır.
    - Tur geçme düğmesi, tk.Button bileşeni olarak tasarlanmıştır ve pass_turn işlevi ile ilişkilendirilmiştir.
    - Oyun tahtası ve oyuncu bilgi panelleri, tkinter bileşenlerini kullanarak düzenlenmiş ve görüntülenmiştir.

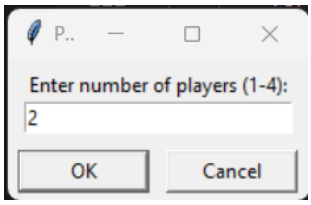
3.3 Veri tabanı tasarımı

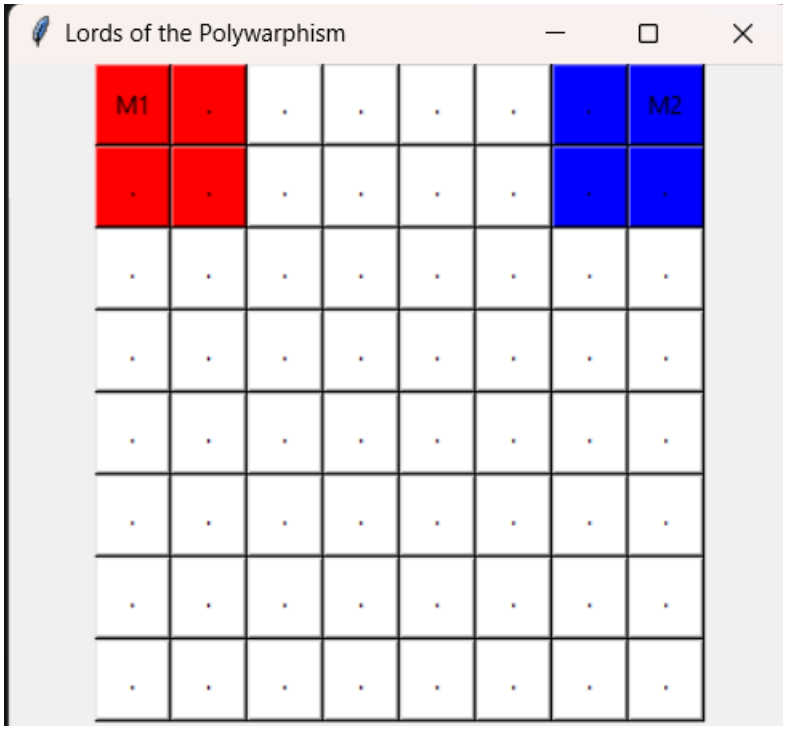
- Projede veri tabanı içermemektedir.

3.4 Kullanıcı arayüzü tasarımı

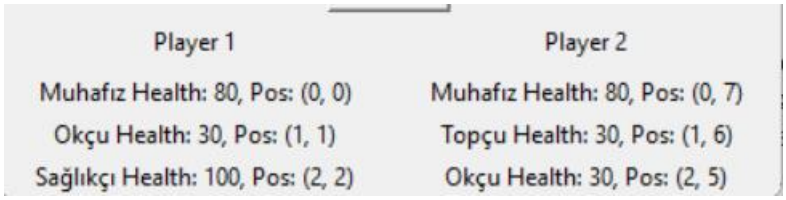
Oyun Tahtası: Oyun tahtası, bir grid yapısı kullanılarak oluşturuldu. Her hücre bir tkinter düğmesi (button) olarak temsil edilir ve oyun alanının boyutuna göre dinamik olarak oluşturulur. Oyuncular, bu tahtaya savaşçılarını yerleştirebilir.



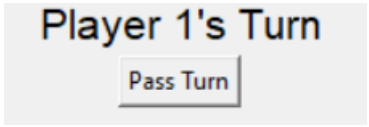




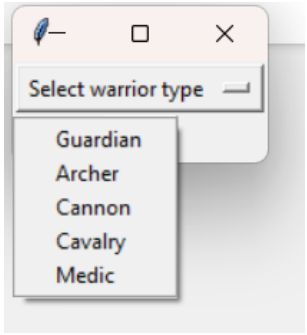
Oyuncu Bilgi Bölümü: Oyuncu bilgi bölümü, her oyuncunun sağlık durumu ve kaynak miktarını göstermek için oluşturulur. Her oyuncunun bilgileri yatay olarak sıralanır ve güncellenir.



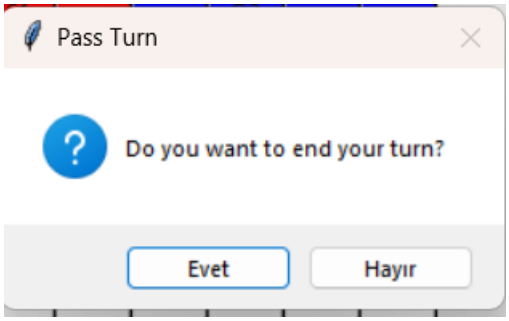
Tur Bilgilendirme Etiketi: Oyuncuların kimin sırasında olduğunu göstermek için bir etiket kullanılır. Her turun kimin sırasında olduğunu belirtmek için bu etiket güncellenir.



Savaşçı Seçim Penceresi: Oyuncuların savaşçılarını tahtaya yerleştirmek için savaşçı seçim penceresi kullanılır. Bu pencere, oyuncunun mevcut kaynaklarına bağlı olarak savaşçı seçimini sağlar.



Bilgilendirme Penceresi: Oyuncuların bilgilendirilmesi için mesaj kutuları kullanılır. Bu mesaj kutuları, oyuncuların eylemlerinin sonuçlarını ve önemli bilgileri iletmek için kullanılır.



Uygulamanın nasıl çalıştırılacağı ile ilgili açıklama: Dosyayı bir Python IDE sinde açıp kod çalıştırıldığında uygulama başlar, oyun tahtası ve oyuncu bilgi panelleri gösterilecek ve oyuncular savaşçılarını yerleştirmeye başlayabilir.

4 UYGULAMA

4.1 Kodlanan bileşenlerin açıklamaları

- Temel olarak 4 farklı bölüme ayrırdık .
- İlk sınıf player sınıfı oyuncuların savaşçıları oluşturmasına, savaşçılarına saldırmasına ve oyun tahtasına yerleştirmesine olanak tanır.
- **Player Sınıfı:**.

  __init__() metodu: Bir oyuncu oluşturulduğunda çağrılır. Oyuncunun kimliği (player_id), sağlığı (health), kaynakları (resources) başlangıç değerleriyle birlikte atanır. Ayrıca, oyuncunun sahip olduğu savaşçıları (warriors), geçen tur sayısını (passed_turns) ve oyundan elenip elenmediğini belirten bir bayrak (eliminated) tutar.

  create_warrior() metodu: Belirli bir savaşçı türündeki bir savaşçıyı yaratır ve oyun tahtasına yerleştirir. Parametreler olarak savaşçının türü (warrior_type), konumu (x ve y koordinatları) ve oyun tahtasını (game_board) alır. Yaratılan savaşçı, oyuncunun kaynaklarına göre yerleştirilir.

  add_warrior() metodu: Belirli bir savaşçı türündeki bir savaşçıyı oyuncunun ordusuna ekler. Yeterli kaynaklar varsa, savaşçı eklenir ve oyuncunun kaynakları azaltılır.

  display_warriors() metodu: Oyuncunun sahip olduğu savaşçıları ekrana yazdırır.

  attack_with_warrior() metodu: Belirli bir savaşçının belirli bir hedefe saldırmasını sağlar. Bu örnekte, saldırı sadece bir çıktı yazdırma işlemi ile temsil edilir. Gerçek bir oyun için, bu metot daha karmaşık bir saldırı mekanizması ile güncellenebilir.

- Bir diğeri sınıf savaşçı sınıfı. Bu kod parçası bir savaş simülasyonunda kullanılmak üzere savaşçı sınıflarını içerir. Temel olarak her savaşçı, isim, saldırı gücü, sağlık, saldırı menzili, maliyet ve oyuncu kimliği gibi özelliklere sahip bir sınıftan türetilir.

- **Warrior Sınıfı:**

  __init__() metodu: Bir savaşçı oluşturulduğunda çağrılır. Savaşçının özellikleri (isim, saldırı gücü, sağlık, saldırı menzili, maliyet ve oyuncu kimliği) başlangıç değerleriyle birlikte atanır.

  find_targets_in_range() metodu: Bir savaşçının belirli bir menzildeki hedefleri bulmasını sağlar. Savaşçının menziline göre, yatay, dikey ve çapraz menziller

dikkate alınarak hedefler bulunur. Bu metod, savaşçının saldırı stratejisini belirlemesine yardımcı olur.

perform_action() metodu: Bu metod, bir savaşçının belirli bir eylemi gerçekleştirmesini sağlar. Alt sınıflar bu metodu, savaşçı türlerine özgü saldırı stratejileriyle uygularlar. Örneğin, bir Muhafız belirli bir menzildeki düşmanlara saldırabilir, bir Okçu en yakındaki düşmanlara saldırabilir vb.

Alt Sınıflar (Guardian, Archer, Cannon, Cavalry, Medic). Her biri Warrior sınıfından türetilir.

Kendi perform_action() metotları vardır. Bu metotlar, savaşçının türüne özgü saldırı stratejilerini uygular. Örneğin, bir Muhafız yakındaki düşmanlara saldırırken, bir Okçu uzaktaki düşmanlara saldırabilir.

- **Gui sınıfı** (Grafiksel Kullanıcı Arayüzü) üzerinde bir strateji oyunu oynamak için bir platform sağlar. Oyun, savaşçıları yerleştirme, saldırı yapma ve oyuncuların elenmesi gibi temel oyun mekaniklerini içerir.

  GameGUI Sınıfı: Oyunun ana kontrolünü sağlar. Oyun tahtasını oluşturur, oyuncuları yönetir, sırayla oyunculara izin verir ve oyunun sona erip ermediğini kontrol eder.

  Oyun tahtası, oyuncuların savaşçılarını yerleştirmeleri için bir arayüz sağlar. Oyuncular, sırayla savaşçılarını yerleştirir ve ardından bu savaşçıları kullanarak rakip savaşçılarına saldırırlar.

- Son olarak **def main** fonksiyonunun kod bloğu ile oyun çalıştırılır ve test edilir.

4.2 Görev dağılımı

- Projenin tasarımı ve taslağı birlikte çıkarılmıştır. Yazılımın bileşenleri sınıflara ayrılmıştır. Herkes kendi sınıfını geliştirmiştir.
- Rapor hazırlanması sürecindeki görev dağılımı 1 ,2 ve 3 numaralı başlıklar Nurdan Bulut tarafından , 4 ve 5 numaralı başlıklar Zeynep Kedikli tarafından hazırlanmıştır.

4.3 Karşılaşılan zorluklar ve çözüm yöntemleri

- Karşılaşılan zorluklar ; ilk olarak oyunun tasarımı ve mantığını oluşturma sürecinde zorlandık. Daha önce oyun alanında bir tecrübemiz olmadığı için araştırmalar yaptık. Polimorfizm konusuyla ilgili bilgi edindik. Oyunun savaşma stratejilerindeki isterleri eklemekte zorlandık. Oyun tahtasındaki ızgara görünümünü koda adapte etekte zorlandık. GUI Programlama: Tkinter kütüphanesiyle birlikte grafiksel kullanıcı arayüzü oluşturma konusunda

deneyimimiz olmadığı için GUI bileşenlerini (pencere, düğme, etiket gibi) oluşturmak ve bunlar arasında etkileşim sağlamakta zorlandık.

4.4 Proje isterlerine göre eksik yönler

- Proje isterlerine göre eksik bir yön bulunmamaktadır.

5 TEST VE DOĞRULAMA

5.1 Yazılımın test süreci

- Yazılımı test eden kod parçası aşağıya görsel olarak eklenmiştir.

```
import tkinter as tk
from gui import GameGUI  #

def main():
    root = tk.Tk()
    root.withdraw()
    app = GameGUI(root)
    root.mainloop()

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

- Test eden kod bloğuyla yazılan oyunu çalıştırılır ve kodda yazılanların doğru çalışıp çalışmadığı oynanarak kontrol edilir.

5.2 Yazılımın doğrulanması

- Bu main fonksiyonu ile çalıştırıldıktan sonra elde edilen çıktının doğruluğu test edilmiş ve doğruluğu görülmüştür. Elde edilen çıktıda açılan pencerede tablo seçimi yapılıyor, oyuncu sayısı giriliyor ve daha sonrasında oyun oynanabiliyor.
- Main fonksiyonu ile yazılım tekrar tekrar test edilebilir.

NURDAN BULUT
https://github.com/nurdanbulut
ZEYNEP KEDİKLİ
https://github.com/Zeynepkedikli