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

Lords Of The Polywarphism

Hazırlayan

Bilgisayar Mühendisliği
Yavuz Selim GÜRSOY
220501003

Yazılım Mühendisliği
Talha TUNA
220502015

DERS SORUMLUSU:
PROF. DR. NEVCİHAN DURU

TARİH:
24/03/2024

1 GİRİŞ
1.1 Projenin amacı

- Bu projenin amacı, çok oyunculu bir savaş oyunu geliştirmek için gereken algoritmik ve programlama becerilerini uygulamaktır. Ayrıca nesne yönelimli programlama, çok biçimlilik, iyi yazılım geliştirme ilkelerine bağlılık ve görselleştirme gibi konuların pratik uygulamalarına değinilmektedir. Proje, yapay zeka bileşenlerinin entegrasyonunu içerdiğinden yapay zeka kavramlarını da içeriyor

- Gerçek oyuncuların etkileşimli bir şekilde oyunu oynamalarını sağlayacak bir arayüz geliştirmek.

- Oyuncuların stratejik kararlar almasını ve bu kararları savaşçı türleri ve yerleştirme stratejileri üzerinden uygulamalarını sağlayacak oyun mekaniklerini tasarlamak.

- Savaşçıların türleri arasında farklılıklar ve dengeyi sağlamak için uygun saldırı ve savunma mekaniklerini oluşturmak.

- Görselleştirmeyi sağlayarak oyunun daha etkileyici ve anlaşılır hale gelmesini sağlamak.

- Nesneye yönelik programlama prensiplerini tam olarak uygulamak ve kodun modüler, okunabilir ve sürdürülebilir olmasını sağlamak.

2 GEREKSİNİM ANALİZİ
2.1 Arayüz gereksinimleri

Oyunun Başlangıcı:
Kullanıcıya oyunun başlangıcında dünya boyutunu seçme seçeneği sunulmalıdır. Bu boyutlar 16x16, 24x24, 32x32 veya kullanıcı tanımlı bir kare dünyada olabilir.
Oyuncu sayısı seçimi yapılmalıdır. En az 1 gerçek oyuncu olmak üzere maksimum 4 oyuncu olabileceği belirtilmiştir.

Savaşçı Seçimi ve Yerleştirme:
Oyuncular sırayla savaşçılarını seçip yerleştirmelidir. Gerçek oyuncuların savaşçı seçimleri kullanıcı arayüzü üzerinden yapılmalıdır.
Kullanıcıya mevcut savaşçı türleri gösterilmeli ve tercihini yapması için bir menü sunulmalıdır.
Seçilen savaşçının dünyada hangi konuma yerleştirileceği için x ve y koordinatları girmesi istenmelidir.
Yeni savaşçının yerleştirilebileceği uygun alanlar görsel olarak belirtilmelidir.
Eğer bir savaşçı daha önce yerleştirilmiş bir savaşçının üzerine yerleştirilecekse, eski savaşçı yok edilir ve kullanılan kaynağın %80'i hazineye aktarılır.

Oyunun İlerlemesi:
Oyuncular sırayla savaşçılarını seçip yerleştirdikten sonra, saldırı başlar. Bu durumun kullanıcıya belirtilmesi gerekir.
Her elin sonunda, oyunculara sahip oldukları kaynak miktarı ve dünyada kalan savaşçı sayısı bilgisi sunulmalıdır.
Oyuncuların her elde kazandıkları kaynak miktarı ve bu kaynağın hesaplanması görsel olarak gösterilmelidir.

Oyunun Sonlanması:
Oyunun sona erdiği durumlar kullanıcıya bildirilmelidir. Örneğin, dünyada hiç savaşçı kalmadığında veya bir oyuncu üst üste 3 el pas geçtiğinde.
Oyun sona erdiğinde, kazanan oyuncu veya kaybeden oyuncu bilgisi kullanıcıya sunulmalıdır.

Görselleştirme:
Matristeki durum, her hamle ve tur sonunda gerçekleşen saldırılardan sonra ekranda gösterilmelidir.
Oyuncuların savaşçılarını yerleştirebilecekleri alanlar önceden yerleştirilen savaşçı konumları ve renk kodları dikkate alınarak gösterilmelidir.

2.2 Fonksiyonel gereksinimler

Fonksiyonel gereksinimler şu şekilde sıralanabilir:

Oyuncu Seçimi:
En az 1 gerçek oyuncu olmak üzere maksimum 4 oyuncu olmalıdır.
Gerçek oyuncuların yanı sıra yapay zeka oyuncusu da bulunmalıdır.
Oyuncular oyun başlamadan önce sırasıyla oyuncu isimlerini ve oyuncu türlerini seçmelidir.

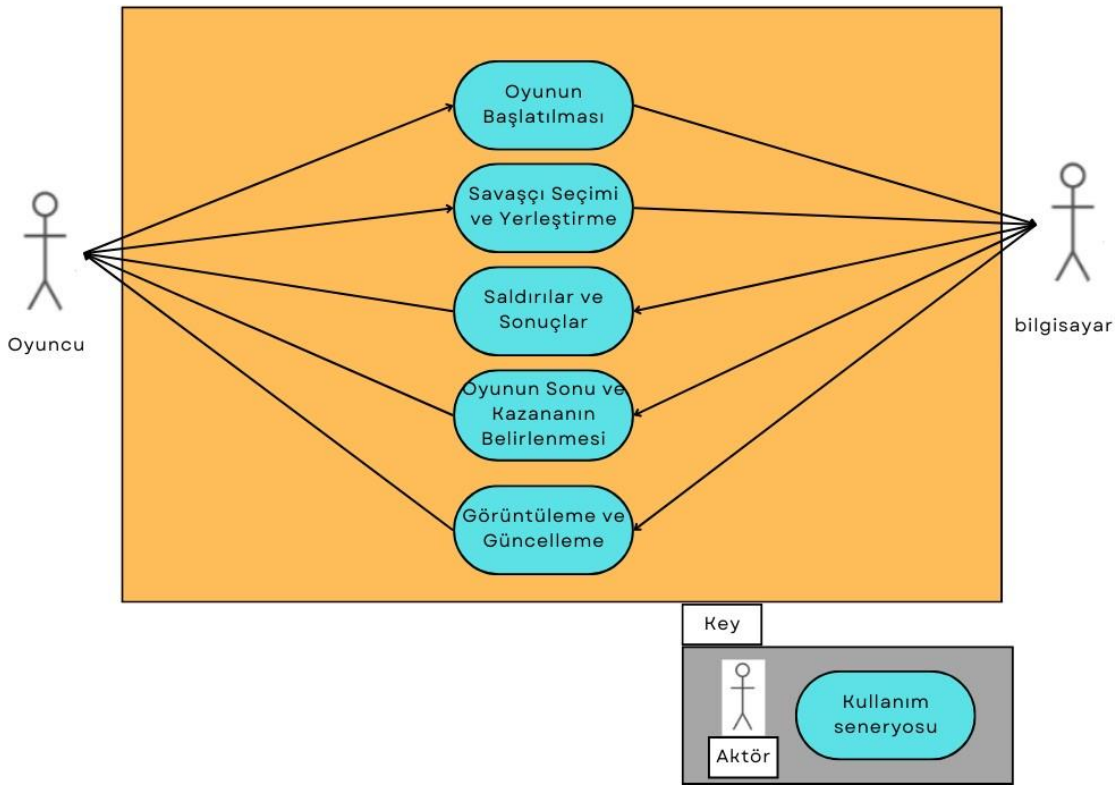
Oyun Dünyası Oluşturma:
Oyuncuların seçtiği dünya boyutuna göre 2-boyutlu bir dünya oluşturulmalıdır (16x16, 24x24, 32x32 veya kullanıcı tanımlı).
Dünyanın boyutları belirli aralıklarda olmalıdır (8x8'den küçük olamaz, 32x32'den büyük olamaz).
Dünya, savaşçıların yerleştirileceği bir matris şeklinde olmalıdır.

Savaşçı Seçimi ve Yerleştirme:
Oyuncular sırayla savaşçı türlerini seçmeli ve bu savaşçıları dünyaya yerleştirmelidir.
Her oyuncunun sırası geldiğinde, seçtiği savaşçı türünü ve yerleştirmek istediği konumu belirtmelidir.
Yeni savaşçılar, daha önce yerleştirilmiş bir savaşçıya komşu olacak şekilde yerleştirilmelidir.
Yeni savaşçılar, boş bir hücreye yerleştirilebilir veya aynı oyuncunun başka bir savaşçısının yanına yerleştirilebilir.
Yeni bir savaşçı daha önce yerleştirilmiş bir savaşçının üzerine yerleştirilirse, eski savaşçı yok edilir ve kaynaklarının %80'i hazineye aktarılır

Savaş ve Saldırı:
Tüm savaşçı yerleştirme işlemleri tamamlandıktan sonra savaş ve saldırı başlar.
Her turda, sahaya ilk yerleşen savaşçı en önce saldırır.
Savaşçılar, belirli menzillerdeki düşman savaşçıları hedef alarak saldırı gerçekleştirir.
Savaşçılar, belirli hasarlar verir ve belirli menzillerdeki düşmanlar üzerinde etki yapar.

Oyun Bitişi:
Oyun, dünyada sadece bir oyuncu kaldığında veya tüm dünyanın %60'ını ele geçiren oyuncu olduğunda sonlanır.
Bir oyuncu, savaşçıları olmadan veya üst üste 3 el pas geçerek oyunu kaybeder.

2.3 Use-Case diyagramı



3 TASARIM

3.1 Mimari tasarım

- Savaşçılar Hiyerarşisi ve Çok-Biçimlilik (Polymorphism): Projenin en temel yapısı savaşçılar hiyerarşisidir. Her savaşçı, genel bir sınıftan türetilmelidir. Bu genel sınıfta, tüm savaşçıların ortak özellikleri ve davranışları tanımlanır. Örneğin, savaşçıların canı, saldırı menzili, hasar verme yeteneği gibi özellikler bu genel sınıfta yer alabilir. Ardından, her savaşçı türü için ayrı bir alt sınıf oluşturulur. Alt sınıflar, genel sınıftan miras alarak özelleştirilmiş özelliklere ve davranışlara sahip olur. Çok-biçimlilik sayesinde, her savaşçı türü genel bir savaşçı olarak işlev görebilir, ancak türler arasında farklı davranışlar sergileyebilir.

- Dünya Temsili: Oyun dünyası, 2 boyutlu bir matris olarak temsil edilir. Bu matris, standart bir vektör kullanılarak oluşturulabilir. Her bir hücre, o bölgedeki savaşçıyı veya boş olduğunu belirten bir değer

içerebilir. Oyun dünyasının boyutu başlangıçta belirlenir ve maksimum boyutlar belirli kriterlere göre sınırlanır.

- Oyuncu Yönetimi: Oyuncular, genel bir oyuncu sınıfından türetilmelidir. Her oyuncu, kendi hazineyi, sahip olduğu kaynakları ve sahip olduğu savaşçıları içeren bir yapıya sahip olacaktır. Ayrıca, oyuncuların sırasını takip etmek ve hangi oyuncunun hamle yapacağını belirlemek için bir kontrol mekanizması olmalıdır.

- Savaşçı Yerleştirme ve Yerleştirme Kontrolleri: Yeni savaşçılar, oyuncuların belirlediği koordinatlara yerleştirilir. Ancak, bu yerleştirme işlemi belirli kurallara tabidir. Örneğin, yeni bir savaşçı yalnızca boş bir hücreye veya aynı oyuncunun başka bir savaşçısının yanına yerleştirilebilir. Bu kuralların uygulanması için gerekli kontroller sağlanmalıdır.

- Görselleştirme: Oyunun durumu ve her hamle sonrası saldırılar, görsel olarak kullanıcıya sunulmalıdır. Matris üzerindeki durum, renkler ve karakterlerle temsil edilir. Her güncelleme sonrasında ekran temizlenir ve güncel durum matris üzerinde gösterilir.

3.2 Kullanılacak teknolojiler

Bu kod Python dili tarafından yazılmıştır

```
import time
import random
import tkinter as tk
```

Time Kütüphanesi: Bu kütüphane, zamanla ilgili işlemleri gerçekleştirmek için kullanılır. Zamanı ölçmek, zaman aralıkları oluşturmak, programın belirli bir süre beklemesini sağlamak gibi işlemleri yapmak için kullanılır.

Random kütüphanesi: Rastgele sayı üretmek veya öğeleri karıştırmak gibi rastgelelikle ilgili işlemler için kullanılır. Özellikle oyun geliştirme veya rastgele veri oluşturma gibi senaryolarda yaygın olarak kullanılır.

Tkinter kütüphanesi: Bu kütüphane, Python'da masaüstü uygulamaları geliştirmek için kullanılan standart bir GUI (Grafiksel Kullanıcı Arayüzü) kütüphanesidir. Tkinter, kullanıcı arayüzü öğeleri (pencere, düğme, metin kutusu, menü vb.) oluşturmak ve yönetmek için kullanılır. Tkinter, basit ve orta düzeyde karmaşıklıktaki GUI uygulamalarını geliştirmek için idealdir.

3.3 Kullanıcı arayüzü tasarımı

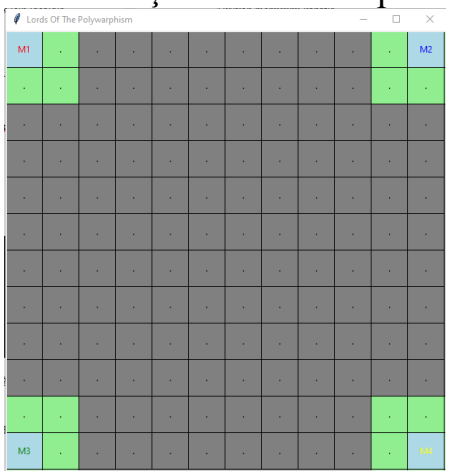
İlk olarak kodu çalıştırıyoruz kod çalıştıktan sonra şu şekil bir çıktı geliyor

```
Enter the board size:
```

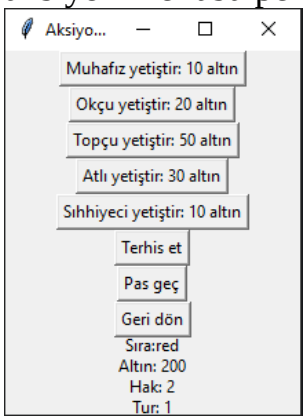
Tahtanın boyutunu giriyoruz (virgül ile) sonra

```
Enter the number of players:
```

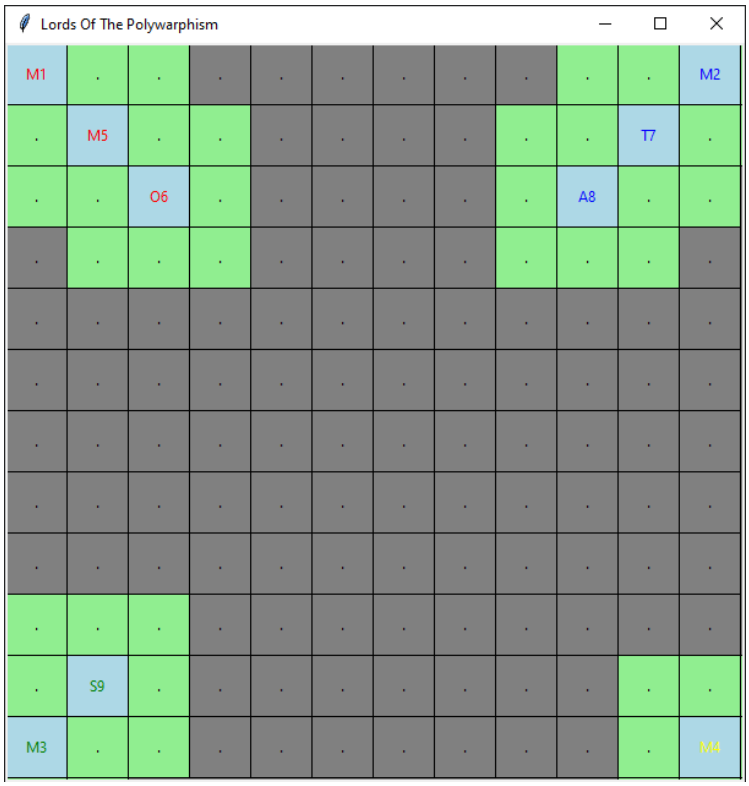
Bizden oyuncu sayısını istiyor oyuncu sayısını giriyoruz (max 4 oyuncu)
Sonra karşımıza bir tane pencere açılıyor



Sonra asker koymak için birinci oyuncu etrafındaki yeşil yere tıklaması gerekiyor tıkladıktan sonra bir tane daha pencere açılıyor o pencerede aksiyon menüsü penceresi



Bu butonlar sayesinde asker yerleştirebiliyoruz, terhis edebiliyoruz, pas geçebiliyoruz, geri dönebiliyoruz ve ayrıca sıradaki kişiyi, o kişinin altınını ve hakkını görebiliyoruz bir de turu gösteriyor



Buda askerlerin yerleştirilmiş bir örneği

4 UYGULAMA

4.1 Kodlanan bileşenlerin açıklamaları

def skip(self, window):

```
def skip(self, window):
    if self.currentPlayer.moveCount == 2:
        self.currentPlayer.skippedRounds.append(self.round)
        # Eğer oyuncu 1 hamle yapmayı tercih ederse, bu raundu pas geçilmiş raunt sayma.

    self.currentPlayer.moveCount = 0
    self._round_system()
    # Oyuncunun hakkını 0' a eşitle ve raunt sistemini çağır.

    window.destroy()
    # Aksiyon menüsünü kapatmaya zorla.
```

Bu skip fonksiyonu, oyuncunun bir raundu pas geçmesini sağlar.

Fonksiyonun işlevleri şunlardır:

Eğer oyuncunun henüz hiç hamlesi yoksa (yani moveCount değeri 2 ise), oyuncunun pas geçtiği bir rauntu işaretlemek için oyun durumunu güncellemek üzere skippedRounds listesine mevcut raund numarasını ekler.

Oyuncunun hamle hakkını 0'a (pas geçildiğini göstermek için) eşitleyerek oyun durumunu günceller.

Raunt sistemi, oyuncunun pas geçme işlemi sonrasında güncellemek için çağrılır.

Aksiyon menüsünü kapatır.

Bu fonksiyon, oyuncunun bir raundu pas geçmesini ve oyunun devam eden durumunu güncellemesini sağlar.

def demobilizate(self, window, clickedX, clickedY):

```
def demobilizate(self, window, clickedX, clickedY):
    player = self.currentPlayer

    if self.round == 1:
        new_window = tk.Toplevel(self.root)
        new_window.title("Uyarı!")
        label = tk.Label(new_window, text="İlk turda asker terhis edemezsin!")
        label.pack()
        window.destroy()
        # İlk rauntta asker terhis edilemez, edilirse oyuncu oyunu kaybeder.
        # Bu yüzden ilk rauntta terhise izin verme.

    if [clickedX, clickedY] in player.playerSoldiersList and self.round != 1:
        for soldier in soldiersList:
            if [soldier.xCoord, soldier.yCoord] == [clickedX, clickedY]:

                player.balance += soldier.cost * 80 / 100
                self._paint_grey_squares(soldier)
                # Askerin yetiştirilme maaliyetinin %80'ini oyuncuya geri ver.
                # Askerin etrafındaki yeşil karoları ve kendi karosunu griye boya.

                self._repaint()
                # Tahtayı yeniden boya

            self._round_system()
            del soldier
            window.destroy()
            # Çıkartılan askerin nesnesini sil, raunt sistemini çağır ve aksiyon menüsünü kapatmaya zorla.

    elif [clickedX, clickedY] not in player.playerSoldiersList:
        new_window = tk.Toplevel(self.root)
        new_window.title("Uyarı!")
        label = tk.Label(new_window, text="Burada terhis edilecek asker yok!")
        label.pack()
        window.destroy()
        # Karoda asker yoksa oyuncuyu uyar.
```

Bu demobilizate fonksiyonu, oyuncunun tıkladığı bir karedeki askeri terhis etmesini sağlar.

Fonksiyonun işlevleri şunlardır:

İlk rauntta asker terhis edilemez. Bu yüzden ilk rauntta terhis işlemi gerçekleştirilmeyecek ve oyuncuya bir uyarı mesajı gösterilecek.

İkinci raunttan itibaren, tıklanan karede oyuncuya ait bir asker varsa, bu askeri terhis eder. Askerin maliyetinin %80'ini oyuncuya geri verir, askerin etrafındaki yeşil kareleri ve askerin bulunduğu kareyi griye boyar. Son olarak, raunt sistemini günceller ve aksiyon menüsünü kapatır.

Eğer tıklanan karede oyuncuya ait bir asker yoksa, oyuncuyu buna göre uyarır.

Bu fonksiyon, oyuncunun belirli bir karedeki askeri terhis etmesini sağlar ve oyun durumunu günceller.

def recruit(self, window, soldier, clickedX, clickedY):

```
def recruit(self, window, soldier, clickedX, clickedY):
    player = self.currentPlayer

    self._search_green_squares(player)
    if player.balance < soldier.cost:
        new_window = tk.Toplevel(self.root)
        new_window.title("Uyarı!")
        label = tk.Label(new_window, text="Yetersiz altın!")
        label.pack()
        window.destroy()

    if player.moveCount == 0:
        new_window = tk.Toplevel(self.root)
        new_window.title("Uyarı!")
        label = tk.Label(new_window, text="Hakkın bitti!")
        label.pack()
        window.destroy()
        # Eğer olur da bir hata oluşup pencere kapanmazsa, oyuncunun işlem yapmasına izin verme.

    if [clickedX, clickedY] not in player.playerGreenSquares:
        new_window = tk.Toplevel(self.root)
        new_window.title("Uyarı!")
        label = tk.Label(new_window, text="Buraya erişimin yok!")
        label.pack()
        window.destroy()
        # Eğer oyuncu yeşil karelerin dışında erişmemesi gereken bir yere tıklarsa, oyuncuyu uyar.

    if [clickedX, clickedY] not in self.posList:
        new_window = tk.Toplevel(self.root)
        new_window.title("Uyarı!")
        label = tk.Label(new_window, text="Burada halihazırda bir asker var!")
        label.pack()
        window.destroy()
        # Eğer oyuncu yeşil karelerin dışında erişmemesi gereken bir yere tıklarsa, oyuncuyu uyar.

    if player.balance >= soldier.cost and player.moveCount > 0 and [clickedX, clickedY] in player.playerGreenSquares \
            and [clickedX, clickedY] in self.posList:
        player.balance -= soldier.cost
        player.moveCount -= 1
        player.playerSoldiersDict[soldier.name].append([clickedX, clickedY])
        player.playerSoldiersList.append([clickedX, clickedY])
```

```
    soldier.team = player.color
    soldier.xCoord = clickedX
    soldier.yCoord = clickedY

    soldiersList.append(soldier)
    allSoldiers.append(soldier)
    self.posList.remove([clickedX, clickedY])
    # Eğer oyuncunun askeri yetiştirmeye yetecek kadar parası, asker yetiştirme hakkı varsa
    # ve tıklanılan karo yeşil karo ise, bu karoda asker yok ise askeri karoya koy, maliyetini oyuncudan al,
    # oyuncunun hakkını 1 azalt, askerin konumunu ayarla, oyundaki asker listesine askeri ekle,
    # ve aktif pozisyonlar listesinden askerin koordinatlarını çıkart.

    for j in range(3, 0, -1):
        secondY1 = soldier.yCoord + 125 - (j * 50)
        secondY2 = soldier.yCoord + 75 - (j * 50)

        for k in range(3, 0, -1):
            secondX1 = soldier.xCoord + 125 - (k * 50)
            secondX2 = soldier.xCoord + 75 - (k * 50)

            self.canvas.create_rectangle(secondX1, secondY1, secondX2, secondY2, outline="black",
                                         fill="lightgreen")
            self.canvas.create_text(secondX2 + 25, secondY2 + 25, text=".")
    # Yeşile boyama işlemi. draw_board() fonksiyonunda kullanılan ile aynı.

    soldier.place = len(allSoldiers)

    for soldiers in soldiersList:
        soldierPosX = soldiers.xCoord
        soldierPosY = soldiers.yCoord

        self.canvas.create_rectangle(soldierPosX + 25, soldierPosY + 25, soldierPosX - 25,
                                     soldierPosY - 25,
                                     outline="black", fill="lightblue")
        self.canvas.create_text(soldierPosX, soldierPosY, text=f"{soldiers.name[0]}{soldiers.place}",
                                fill=soldiers.team)
        # Karolar yeşile boyandıktan sonra bazı askerlerin üstü kapanabilir.
        # Bunu önlemek için tüm askerlerin pozisyonlarını yeniden açık maviye boya
self._round_system()
window.destroy()
# Raunt sistemini çağır ve aksiyon menüsünü kapatmaya zorla.
```

Bu recruit fonksiyonu, oyuncunun tıkladığı bir kareye asker yerleştirmesini sağlar.

Fonksiyonun işlevleri şunlardır:

Oyuncunun bakiyesini ve hareket sayısını kontrol eder. Eğer yeterli bakiye ve hareket hakkı yoksa, uyarı mesajı gösterir ve işlemi sonlandırır.

Tıklanan karenin oyuncunun erişim alanında olup olmadığını kontrol eder. Eğer tıklanan kare oyuncunun erişim alanında değilse, uyarı mesajı gösterir ve işlemi sonlandırır.

Tıklanan karenin zaten bir asker tarafından işgal edilip edilmediğini kontrol eder. Eğer kare zaten bir asker tarafından işgal ediliyorsa, uyarı mesajı gösterir ve işlemi sonlandırır.

Yukarıdaki kontrolleri geçen durumda ise, askerin maliyetini oyuncunun bakiyesinden düşer, oyuncunun hareket sayısı bir azaltılır ve yeni asker tıklanan kareye yerleştirilir.

Yerleştirilen askerin özellikleri ve pozisyonu ayarlanır, askerin listelere eklenmesi sağlanır ve tıklanan kare artık mevcut pozisyon listesinden çıkarılır.

Yeni askerin yerleştirildiği karenin etrafı yeşile boyanır ve askerlerin listesindeki diğer askerlerin pozisyonları güncellenir.

Son olarak, raunt sistemi çağrılır ve aksiyon menüsü penceresi kapatılır.

Bu fonksiyon, oyuncunun belirli bir kareye asker yerleştirmesini sağlar ve oyunun devam etmesini sağlayacak şekilde oyun durumunu günceller.

def action_menu(self, message):

```
def action_menu(self, message):

    new_window = tk.Toplevel(self.root)
    new_window.title("Aksiyon Menüsü")

    label = tk.Label(new_window, text=message)

    button_text = "Muhafız yetiştir: 10 altın"
    button = tk.Button(new_window, text=button_text,
                       command=lambda text=button_text: self.recruit(new_window, Guardian(), self.clickedX,
                                                                     self.clickedY))
    button.pack()

    button_text = "Okçu yetiştir: 20 altın"
    button = tk.Button(new_window, text=button_text,
                       command=lambda text=button_text: self.recruit(new_window, Archer(), self.clickedX,
                                                                     self.clickedY))
    button.pack()

    button_text = "Topçu yetiştir: 50 altın"
    button = tk.Button(new_window, text=button_text,
                       command=lambda text=button_text: self.recruit(new_window, Artilleryman(), self.clickedX,
                                                                     self.clickedY))
    button.pack()

    button_text = "Atlı yetiştir: 30 altın"
    button = tk.Button(new_window, text=button_text,
                       command=lambda text=button_text: self.recruit(new_window, Cavalry(), self.clickedX,
                                                                     self.clickedY))
    button.pack()

    button_text = "Sıhhiyeci yetiştir: 10 altın"
    button = tk.Button(new_window, text=button_text,
                       command=lambda text=button_text: self.recruit(new_window, Medic(), self.clickedX,
                                                                     self.clickedY))
    button.pack()

    button_text = "Terhis et"
    button = tk.Button(new_window, text=button_text,
                       command=lambda text=button_text: self.demobilizate(new_window, self.clickedX, self.clickedY))
    button.pack()
button_text = "Pas geç"
button = tk.Button(new_window, text=button_text, command=lambda text=button_text: self.skip(new_window))
button.pack()

button_text = "Geri dön"
button = tk.Button(new_window, text=button_text, command=lambda text=button_text: new_window.destroy())
button.pack()

label.pack()
# Asker yetiştirme ve aksiyon menüsü.
```

Bu action_menu fonksiyonu, kullanıcıya oyun sırasında yapabileceği eylemleri seçmesine olanak tanıyan bir arayüz sağlar.

Bir Toplevel penceresi oluşturur ve başlığını "Aksiyon Menüsü" olarak ayarlar.

Belirli eylemleri gerçekleştirmek için kullanılacak düğmeler oluşturur. Bu eylemler şunlardır:

- Muhafız yetiştirme
- Okçu yetiştirme
- Topçu yetiştirme
- Atlı yetiştirme
- Sıhhiyeci yetiştirme
- Askeri terhis etme
- Pas geçme
- Menüden çıkma

Her bir eylem için bir düğme oluşturur ve her düğmeye ilgili eylemi gerçekleştirecek bir işlev atanır. Örneğin, "Muhafız yetiştir" düğmesine tıklandığında recruit fonksiyonu çağrılır ve Guardian sınıfından bir asker oluşturulur.

Oluşturulan düğmeleri pencereye yerleştirir.

Pencereye bir etiket ekler ve mesajı bu etikete yazar.

Bu fonksiyon, oyuncuların oyun sırasında yapabilecekleri eylemleri seçmelerini sağlar ve bu eylemleri gerçekleştirmek için uygun arayüzü sunar.

def draw_board(self):

```
def draw_board(self):
    for row in range(self.rows):
        for col in range(self.columns):
            x1 = col * 50
            y1 = row * 50
            x2 = x1 + 50
            y2 = y1 + 50

            self.posList.append([x2 - 25, y2 - 25])
            # Karoların orta noktalarını muhafız seçiminde aynı karoları kullanmamak için bir listede tut.

            self.canvas.create_rectangle(x1, y1, x2, y2, outline="black")
            self.canvas.create_text(x2 - 25, y2 - 25, text=".")
            # Izgara zemini oluştur ve karoların ortasına nokta ekle.

    for selectedPlayer in self.playerList:
        firstGuardian = Guardian()
        # Sıradan bir oyuncu al ve muhafız oluştur.

        guardianPos = random.choice(self.firstGuardiansPosList)
        guardianPosX = guardianPos[0]
        guardianPosY = guardianPos[1]

        firstGuardian.xCoord = guardianPosX
        firstGuardian.yCoord = guardianPosY
        firstGuardian.team = selectedPlayer.color
        firstGuardian.place = len(allSoldiers) + 1
        # Muhafız için konum belirle ve muhafızın özelliklerini tanımla

        selectedPlayer.playerSoldiersDict["Muhafız"].append(guardianPos)
        soldiersList.append(firstGuardian)
        allSoldiers.append(firstGuardian)
        # Muhafızın konumunu oyuncunun askerler sözlüğünün Muhafız kısmına ekle.

        self.firstGuardiansPosList.remove(guardianPos)
        selectedPlayer.playerSoldiersList.append([guardianPosX, guardianPosY])
        self.posList.remove(guardianPos)
        self._search_green_squares(selectedPlayer)
        # Kullanılabilir pozisyonlar listesinden muhafızın konumunu çıkar
```

```
selectedPlayer.firstGuardianPosX = guardianPosX
selectedPlayer.firstGuardianPosY = guardianPosY
# İlk muhafızlar için rastgele x ve y koordinatları seç ve bunları oyuncu nesnesinin ilk muhafızının
# x ve y koordinatlarına ata. Peşinden bu koordinatları koordinat listesinden sil.

# Eğer bunu yapmazsan muhafızlar üst üste binebilir.

for j in range(3, 0, -1):
    secondY1 = guardianPosY + 125 - (j * 50)
    secondY2 = guardianPosY + 75 - (j * 50)

    for k in range(3, 0, -1):
        secondX1 = guardianPosX + 125 - (k * 50)
        secondX2 = guardianPosX + 75 - (k * 50)

        self.canvas.create_rectangle(secondX1, secondY1, secondX2, secondY2, outline="black",
                                     fill="lightgreen")
        self.canvas.create_text(secondX2 + 25, secondY2 + 25, text=".")
        # Muhafızların yerleştirildiği konumların etrafındaki karoları açık yeşile boya ve ortalarına nokta ekle.

guardianPosX = selectedPlayer.firstGuardianPosX
guardianPosY = selectedPlayer.firstGuardianPosY

self.canvas.create_rectangle(guardianPosX + 25, guardianPosY + 25, guardianPosX - 25, guardianPosY - 25,
                             outline="black", fill="lightblue")
self.canvas.create_text(guardianPosX, guardianPosY, text=f"M{firstGuardian.place}",
                        fill=selectedPlayer.color)
# Muhafızların bulunduğu karoyu açık maviye boya ve isimlerini karoya yaz.
```

Bu draw_board fonksiyonu, oyun tahtasını oluşturur ve başlangıç pozisyonlarını belirler.

Fonksiyonun işlevleri şunlardır:

İlk olarak, oyun tahtasını oluşturmak için satır ve sütunlar boyunca bir döngü başlatır. Her bir karoyu 50x50 piksel boyutunda ve siyah bir kenarlıkla oluşturur. Karoların ortasına bir nokta ekler.

Ardından, her bir oyuncu için bir muhafız oluşturur. Muhafızların başlangıç pozisyonlarını rastgele seçer ve bu pozisyonları oyun tahtasına yerleştirir. Her muhafız için, çevresindeki 3x3'lük bir alanı yeşil renge boyar ve ortasına bir nokta ekler. Muhafızların bulunduğu kareyi de açık mavi renge boyar ve oyuncunun renginde muhafızın ismini yazar.

Son olarak, muhafızların yerleştirildiği konumları kullanılabilir pozisyonlar listesinden çıkarır ve muhafızların bulunduğu pozisyonları listelerine ekler.

Bu fonksiyon, oyunun başlangıcında oyuncuların muhafızlarını yerleştirmek ve oyun tahtasını oluşturmak için kullanılır.

def _round_system(self):

```
def _round_system(self):
    if self.currentPlayer is not None and len(self.currentPlayer.playerSoldiersList) == 0:
        self._decide_the_loser(self.currentPlayer)
        # Eğer oyuncunun askeri yoksa, oyuncuyu oyundan at.

    if len(self.playerList) == 1:
        self._decide_the_winner(self.playerList[0], option: 0)
        # Eğer oyuncu, oyuncu listesinde tek kaldıysa oyuncuyu galip ilan et.

    if self.currentPlayer is not None and len(self.currentPlayer.playerSoldiersList) >= (
            self.rows * self.columns * 60 // 100):
        self._decide_the_winner(self.currentPlayer, option: 1)
        # Eğer oyuncu tahtanın %60'ını ya da daha fazlasını ele geçirirse, oyuncuyu galip ilan et.

    if self.currentPlayer is None or self.currentPlayer.moveCount == 0:
        self.currentPlayer = self.playerList[0]
        self.playerList.remove(self.currentPlayer)
        self.playerList.append(self.currentPlayer)
        # Hakkı biten oyuncuyu listenin en sonuna at.

    if self._check_skips(self.currentPlayer.skippedRounds) is True:
        self._decide_the_loser(self.currentPlayer)
        # Eğer oyuncu 3 tur üst üste pas geçerse, oyuncuyu oyundan at.

    if self.currentPlayer.playerNum == 1 and self.currentPlayer.moveCount == 0:
        print(f"-----------------------------------------\n"
              f"{self.round}. Savaş")

        self.round += 1
        self.currentPlayer.balance += 10
        self.currentPlayer.balance += len(self.currentPlayer.playerSoldiersList)
        # Savaş aşaması başladığında konsola yazdır,
        # Raunt sayısını 1 arttır ve her oyuncuya verilmesi gereken altın miktarını ver.

        for player in self.playerList:
            player.moveCount = 2
            # Her oyuncunun asker yerleştirme hakkını 2'ye çıkart.
```

```
for soldier in soldiersList:
    if soldier.health > 0:
        if isinstance(soldier, Medic):
            soldier.heal()
            # Eğer asker listesindeki ilk asker bir Sıhhiyeciyse saldırı değil iyileştirme komutu kullan.

        else:
            soldier.attack()
            # Eğer asker Sıhhiyeci değilse, saldırı komutunu kullan.

    if soldier.killedTargets != 0:
        for target in soldier.killedTargets:
            if target in soldiersList:
                self._paint_grey_squares(target)
                # Eğer asker savaş esnasında birini öldürürse ve öldürülen hedef asker listesinden çıkartılmadıysa,
                # askeri listeden çıkart.

                self._repaint()
                # Oyun tahtasını hayatta kalan askerler için yeniden boya.

                del target
                # Ölen askerin nesnesini sil.
print("-----------------------------------------")
```

Bu _round_system fonksiyonu, oyunun her bir raundunun sonunda çağrılır ve oyunun durumunu kontrol eder, oyuncuların hamle haklarını yönetir ve gerekirse oyunun sonunu belirler.

Fonksiyonun işlevleri şunlardır:

İlk olarak, mevcut oyuncunun (self.currentPlayer) varlığını ve askerlerinin listesinin boş olup olmadığını kontrol eder. Eğer oyuncunun askeri yoksa, _decide_the_loser metodunu kullanarak oyuncuyu oyundan atar.

Ardından, oyuncu listesinde yalnızca bir oyuncunun kalıp kalmadığını kontrol eder. Eğer sadece bir oyuncu kaldıysa, bu oyuncuyu kazanan ilan eder.

Bir sonraki adımda, oyuncunun oyuncu tahtasının %60'ını veya daha fazlasını ele geçirip geçirmediğini kontrol eder. Eğer oyuncu bu koşulu sağlarsa, oyuncuyu kazanan ilan eder.

Sonraki kontrol, mevcut oyuncunun hamle hakkının olup olmadığını kontrol eder. Eğer oyuncunun hamle hakkı bitmişse, sırayı bir sonraki oyuncuya geçirir.

Bir sonraki kontrol, mevcut oyuncunun üst üste üç tur boyunca pas geçip geçmediğini kontrol eder. Eğer üç tur üst üste pas geçilmişse, oyuncuyu kaybeden ilan eder.

Son olarak, mevcut oyuncunun birinci oyuncu olup olmadığını ve hamle hakkının 0 olup olmadığını kontrol eder. Eğer bu koşullar sağlanıyorsa, bir sonraki savaş aşamasına geçiş için gerekli hazırlıkları yapar.

Bu _round_system fonksiyonu, her raundun sonunda oyunun durumunu kontrol eder ve gerektiğinde oyunun sonunu belirler.

def _check_skips(self, skippedRounds):

```python
def _check_skips(self, skippedRounds):
    if len(skippedRounds) >= 3:
        for i in range(len(skippedRounds)):
            if skippedRounds[i - 2] + 2 == skippedRounds[i - 1] + 1 == skippedRounds[i]:
                return True

        return False

    # Eğer sana gelen listede art arda bulunan üç raunt varsa True, yoksa False döndür.
```

Bu _check_skips fonksiyonu, belirli bir oyuncunun üç ardışık raunt boyunca hamle yapmadığını kontrol eder.

İlk olarak, skippedRounds adlı bir liste alır, bu liste önceki rauntlarda oyuncunun hamle yapmadığı rauntların indekslerini içerir.

Ardından, eğer skippedRounds listesinin uzunluğu 3 veya daha fazlaysa, yani oyuncu en az üç raunt boyunca hamle yapmamışsa, işlem devam eder.

Her bir önceki rauntun indeksi üzerinde döngü yaparak, eğer üç ardışık rauntta bulunan indeksler birbirlerine sıralı bir şekilde artarsa (örneğin, 2, 3, 4), bu durumda oyuncunun üç ardışık raunt boyunca hamle yapmadığı anlamına gelir ve fonksiyon True değeri döndürür.

Ancak, eğer üç ardışık rauntta bulunan indeksler birbirine sıralı değilse, yani ardışık değilse, fonksiyon False değeri döndürür.

Bu _check_skips fonksiyonu, oyuncunun üç ardışık raunt boyunca hamle yapmadığını kontrol eder ve buna göre bir değer döndürür. Bu, oyuncunun oyunu terk ettiği veya başka bir nedenle üç ardışık raunt boyunca hamle yapmadığı durumu belirlemek için kullanılır.

def _decide_the_loser(self, loser):

```
def _decide_the_loser(self, loser):
    self.playerList.remove(loser)
    self.currentPlayer = self.playerList[0]
    # Önce oyuncuyu oyuncu listesinden çıkar ve sırayı diğer oyuncuya ver.

    for soldier in allSoldiers:
        if soldier.team == loser.color and soldier in soldiersList:
            self._paint_grey_squares(soldier)
            # Oyuncunun tüm askerlerinin bulunduğu karoları ve yeşil karolarını griye boya.

            del soldier
            # Askeri sil.


    self._repaint()
    # Tahtayı yeniden boya.

    print(loser.color, "kaybetti!")

    del loser
    # Oyuncuyu sil.
```

Bu _decide_the_loser fonksiyonu, oyun sırasında bir oyuncunun kaybetmesini ve buna bağlı olarak oyun tahtasındaki durumun güncellenmesini sağlar.

İlk olarak, loser parametresi olarak belirtilen oyuncuyu oyuncu listesinden (playerList) çıkarır. Bu, oyuncunun oyun dışı kalmasını temsil eder. Ardından, sıranın diğer oyuncuya geçmesini sağlamak için currentPlayer değişkenini oyuncu listesinin ilk öğesine (self.playerList[0]) ayarlar. Bu, oyunun devam edeceği oyuncunun belirlenmesini sağlar.
Oyuncunun kaybeden takımına ait tüm askerlerin bulunduğu kareleri griye boyamak için _paint_grey_squares metodunu çağırır. Bu, kaybeden oyuncunun askerlerinin oyun tahtasındaki görünürlüğünü kaldırır.
Kaybeden oyuncunun tüm askerlerini (allSoldiers listesinde) ve soldiersList listesindeki askerlerini kaldırır.
Oyun tahtasını yeniden çizmek için _repaint metodunu çağırır. Bu, oyun tahtasındaki güncel durumu yeniden görselleştirir.
Son olarak, kaybeden oyuncunun rengini ekrana yazdırır ve loser parametresini silerek kaybeden oyuncunun bellekten kaldırılmasını sağlar.
Bu _decide_the_loser fonksiyonu, bir oyuncunun kaybetmesi durumunda oyun tahtasını ve oyun durumunu günceller. Bu, oyuncunun oyun dışı kalmasını ve oyunun devam eden oyuncular arasında devam etmesini sağlar.

def _decide_the_winner(self, winner, option):

```
def _decide_the_winner(self, winner, option):
    self.playerList.clear()
    if option == 0:
        print(winner.color, "tek başına hayatta kalarak kazandı!")
    # Eğer oyuncu tek başına hayatta kalarak kazandıysa bu bloğa gir.

    elif option == 1:
        print(winner.color, "oyun tahtasının %60'ından fazlasını ele geçirerek kazandı!")
    # Eğer oyuncu tek başına hayatta kalarak kazandıysa bu bloğa gir.

    print("Oyun tamamlandı.")
    exit(-1)
```

Bu _decide_the_winner fonksiyonu, oyunun sonunda kazananı belirlemek ve sonucu ekrana yazdırmak için kullanılır.

winner parametresi, kazanan oyuncuyu temsil eder.
option parametresi, kazanma seçeneğini belirler. Eğer option değeri 0 ise, kazananın tek başına hayatta kalarak kazandığı durumu ifade eder. Eğer option değeri 1 ise, kazananın oyun tahtasının %60'ından fazlasını ele geçirerek kazandığı durumu ifade eder.
Fonksiyonun işlevleri şunlardır:

İlk olarak, mevcut oyuncu listesini temizler. Bu, oyuncuların kazananı belirlemek için bir sonraki oyunda kullanılabileceği anlamına gelir.
Ardından, kazananın rengini ve kazanma seçeneğine göre kazananın nasıl kazandığını ekrana yazdırır. Bu, kazananın kazanma durumunu oyunculara ve diğer oyunculara bildirir.
Son olarak, oyunun tamamlandığını belirtmek için bir mesaj yazdırır ve programı sonlandırır (exit(-1)).
Bu _decide_the_winner fonksiyonu, oyunun sonunda kazananı belirleyerek ve sonucu ekrana yazdırarak oyunun sonlanmasını sağlar.

def _search_green_squares(self, player):

```
def _search_green_squares(self, player):
    for soldierPositions in player.playerSoldiersDict.values():
        for soldierPos in soldierPositions:
            player.playerGreenSquares = [pos for pos in player.playerGreenSquares if pos not in [
                [soldierPos[0] - 50, soldierPos[1] - 50],
                [soldierPos[0], soldierPos[1] - 50],
                [soldierPos[0] + 50, soldierPos[1] - 50],
                [soldierPos[0] - 50, soldierPos[1]],
                [soldierPos[0] + 50, soldierPos[1]],
                [soldierPos[0] - 50, soldierPos[1] + 50],
                [soldierPos[0], soldierPos[1] + 50],
                [soldierPos[0] + 50, soldierPos[1] + 50]
            ]]
    # Önce mevcut asker tipinin koordinatlarını oyuncunun yeşil kare listesinden çıkar.

    for soldierPositions in player.playerSoldiersDict.values():
        for soldierPos in soldierPositions:
            soldierPosX, soldierPosY = soldierPos
            player.playerGreenSquares.extend([
                [soldierPosX - 50, soldierPosY - 50],
                [soldierPosX, soldierPosY - 50],
                [soldierPosX + 50, soldierPosY - 50],
                [soldierPosX - 50, soldierPosY],
                [soldierPosX + 50, soldierPosY],
                [soldierPosX - 50, soldierPosY + 50],
                [soldierPosX, soldierPosY + 50],
                [soldierPosX + 50, soldierPosY + 50]
            ])
    # Ardından güncel asker tipinin pozisyonlarını ekle.
```

Bu _search_green_squares fonksiyonu, belirli bir oyuncunun askerlerinin etrafındaki yeşil kareleri bulur ve günceller.

İlk olarak, oyuncunun mevcut askerlerinin koordinatlarını içeren player.playerSoldiersDict sözlüğünün değerlerini (soldierPositions) döngüye alır. Bu döngü, her bir askerin pozisyonunu (soldierPos) alır.

Daha sonra, her bir askerin etrafındaki potansiyel yeşil karelerin koordinatlarını, mevcut yeşil kareler listesinden çıkarır. Bu, askerin etrafındaki karelerin güncellenmesini sağlar. Çıkarılacak koordinatlar, askerin etrafındaki dokuz kareyi temsil eder.

Ardından, her bir askerin pozisyonunu kullanarak etrafındaki dokuz karenin koordinatlarını hesaplar. Bu koordinatlar, askerin etrafındaki kareleri temsil eder. Bu koordinatlar, askerin etrafındaki kareleri güncellemek için kullanılacak.

Son olarak, hesaplanan koordinatları oyuncunun mevcut yeşil kareler listesine ekler. Bu, askerin etrafındaki karelerin güncellenmesini sağlar ve oyuncunun askerlerinin hareket edebileceği geçerli konumları temsil eder.

Bu _search_green_squares fonksiyonu, bir oyuncunun askerlerinin etrafındaki yeşil kareleri bulur ve günceller. Bu kareler, oyuncunun askerlerinin hareket edebileceği geçerli konumları gösterir.

def _click(self, event):

```
def _click(self, event):
    x, y = event.x, event.y
    # Oyuncunun tıkladığı pikselin x ve y koordinatlarını al.

    col = x // 50
    row = y // 50

    xOut, yOut = (col * 50) + 25, (row * 50) + 25
    self.clickedX, self.clickedY = xOut, yOut
    # Oyuncunun tıklayacağı pikselin tam olarak kutucuğun orta noktasına gelme ihtimali oldukça zor.
    # Bu yüzden önce kutucuğun alanını tanıttık.
    # Tıklanılan bölgeyle ilişkili kutucuğu işaretle.

    self.action_menu(
        f"Sıra:{self.currentPlayer.color}\nAltın: {self.currentPlayer.balance}\nHak: {self.currentPlayer.moveCount}\nTur: {self.round}")
    # Aksiyon menüsünü oluşturmak için gerekli fonksiyonu çağır ve oyuncunun bilgilerini bu menüye yazdır.
```

Bu _click fonksiyonu, oyuncunun oyun tahtasındaki bir kareye tıklamasını işler.

İlk olarak, event parametresinden tıklanan pikselin x ve y koordinatlarını alır.

Sonra, bu pikselin hangi satır ve sütuna denk geldiğini hesaplamak için tıklanan pikselin bölgesini bulur. Her bir kare 50x50 piksel olduğundan, tıklanan pikselin bölgesini 50'e böler ve tamsayı bölme operatörü // ile tıklanan pikselin tam olarak hangi kareye denk geldiğini bulur.

Ardından, tıklanan karenin tam olarak hangi pikselin merkezine denk geldiğini hesaplar. Bu, daha sonra kullanılacak olan self.clickedX ve self.clickedY değişkenlerini günceller.

Son olarak, action_menu metodunu çağırarak oyun arayüzünde bir eylem menüsü oluşturur. Bu menüde, mevcut oyuncunun rengi, altın miktarı, hareket hakkı ve tur numarası gibi bilgileri görüntüler. Bu bilgiler, oyuncunun mevcut durumunu gösterir ve oyuncunun yapabileceği eylemleri gözlemlemesine yardımcı olur.

Bu _click fonksiyonu, oyuncunun tıkladığı karenin konumunu belirleyerek ve oyuncunun mevcut durumunu göstererek kullanıcı etkileşimini işler.

def _paint_grey_squares(self, target):

```
def _paint_grey_squares(self, target):
    possibleGreenSquares = [[target.xCoord - 50, target.yCoord - 50],
                            [target.xCoord, target.yCoord - 50],
                            [target.xCoord + 50, target.yCoord - 50],
                            [target.xCoord - 50, target.yCoord],
                            [target.xCoord, target.yCoord],
                            [target.xCoord + 50, target.yCoord],
                            [target.xCoord - 50, target.yCoord + 50],
                            [target.xCoord, target.yCoord + 50],
                            [target.xCoord + 50, target.yCoord + 50]]
    # Ölen askerin etrafında bulunabilecek olası yeşil karo koordinatlarını tutan bir liste.
    for i in self.playerList:
        if [target.xCoord, target.yCoord] in i.playerSoldiersList:
            i.playerSoldiersList.remove([target.xCoord, target.yCoord])
            i.playerSoldiersDict[target.name].remove([target.xCoord, target.yCoord])

    self.posList.append([target.xCoord, target.yCoord])
    soldiersList.remove(target)
    # Ölen askeri kullanılan tüm listelerden çıkart.
    # Askerin koordinatlarını aktif koordinatlar listesine ekle.
    for i in soldiersList:
        if target in i.targetsInReach:
            i.targetsInReach.remove(target)
            # Oyun tahtasındaki her askerin çevresindeki askerlerin listesini tutan listelerine eriş,
            # Bu listeden ölen askeri çıkart.
    for possibleSquare in possibleGreenSquares:
        if possibleSquare in self.currentPlayer.playerGreenSquares:
            self.currentPlayer.playerGreenSquares.remove(possibleSquare)
            # Askerin etrafındaki karoları aktif yeşil karolar listesinden çıkart.
    self._search_green_squares(self.currentPlayer)
    # Aktif yeşil karolar için yeniden arama yap.

    for greySquare in possibleGreenSquares:
        posX = greySquare[0]
        posY = greySquare[1]

        self.canvas.create_rectangle(posX + 25, posY + 25, posX - 25, posY - 25,
                                     outline="black",
                                     fill="grey")
        self.canvas.create_text(posX, posY, text=".")
    # Çıkarılan askerin bulunduğu karoyla birlikte etrafındaki 9 karonun rengini de gri yap.
```

Bu _paint_grey_squares fonksiyonu, bir askerin öldürülmesi durumunda, ölen askerin etrafındaki kareleri gri renge boyar. Ayrıca, ölen askeri ve etrafındaki kareleri oyun tahtası ve oyuncu listelerinden kaldırır.

İlk olarak, ölen askerin etrafında olabilecek olası yeşil karelerin koordinatlarını içeren bir possibleGreenSquares listesi oluşturulur. Bu liste, ölen askerin etrafındaki dokuz kareyi temsil eder.

Daha sonra, ölen askeri oyuncu listelerinden (playerSoldiersList, playerSoldiersDict) ve oyun tahtası listelerinden (soldiersList) kaldırır. Ayrıca, ölen askerin koordinatlarını aktif pozisyonlar listesine ekler (posList).

Ölen askerin etrafındaki karelerden (yeşil kareler) etkilenen diğer askerlerin hedef listelerinden ölen askeri çıkarır.

Aktif oyuncunun yeşil kareler listesinden ölen askerin etrafındaki kareleri çıkarır ve yeniden yeşil kareleri aramak için _search_green_squares metodunu çağırır.

Son olarak, ölen askerin etrafındaki karelerin gri renkte dikdörtgenlerini çizer ve karelerin ortasına bir nokta koyar.

Bu fonksiyon, bir askerin öldürülmesi durumunda, oyun tahtasını güncellemek ve ölen askerin etrafındaki kareleri gri renge boyamak için kullanılır.

def _repaint(self):

```
def _repaint(self):
    for player in self.playerList:
        self._search_green_squares(player)
        # Fonksiyonu kullanarak askerlerin etrafındaki yeşil karoları belirle.

        for greenSquare in player.playerGreenSquares:
            posX = greenSquare[0]
            posY = greenSquare[1]

            self.canvas.create_rectangle(posX + 25, posY + 25, posX - 25, posY - 25, outline="black",
                                         fill="lightgreen")
            self.canvas.create_text(posX, posY, text=".")
            # Aramadan sonra elde ettiğin güncellenmiş yeşil karo listesini kullanarak yeşil karoları boya.

    for soldiers in soldiersList:
        soldierPosX = soldiers.xCoord
        soldierPosY = soldiers.yCoord

        self.canvas.create_rectangle(soldierPosX + 25, soldierPosY + 25, soldierPosX - 25,
                                     soldierPosY - 25, outline="black", fill="lightblue")
        self.canvas.create_text(soldierPosX, soldierPosY, text=f"{soldiers.name[0]}{soldiers.place}",
                                fill=soldiers.team)
```

Bu _repaint metodu, oyun tahtasını ve oyuncuların askerlerini yeniden çizer.

İlk olarak, her bir oyuncunun playerGreenSquares listesindeki yeşil kareleri bulmak için _search_green_squares metodunu çağırır. Bu yeşil kareler, oyuncunun askerlerinin hareket edebileceği geçerli konumları temsil eder.

Daha sonra, her bir oyuncunun yeşil karelerini döngüye alır. Her bir yeşil kare için, karenin koordinatlarını alır (pozX ve pozY) ve bu koordinatlar üzerinde bir dikdörtgen oluşturur. Bu dikdörtgeni lightgreen (açık yeşil) renkte çizer ve karenin merkezine bir nokta koyar (.). Böylece, oyuncunun askerlerinin hareket edebileceği geçerli konumları görsel olarak gösterir.

Ardından, her asker için soldiersList döngüsüne girer. Bu döngü, tüm askerleri tarar. Her asker için, askerin bulunduğu konumda bir dikdörtgen oluşturur. Dikdörtgeni lightblue (açık mavi) renkte çizer ve askerin adını ve numarasını içeren bir metin ekler. Askerin takım rengine göre metnin rengini ayarlar.

Bu _repaint metodu, oyun tahtasını ve askerlerin konumlarını güncellemek için kullanılır. Oyuncuların askerlerinin hareket etmesi veya saldırması gibi bir eylem gerçekleştikten sonra, bu metod çağrılarak oyun tahtası yeniden çizilir ve güncellenmiş konumlar görsel olarak görüntülenir.

Class GameBoard

```python
class GameBoard:
    def __init__(self, root, rows, columns, playerCount):
        self.root = root
        self.rows = rows
        self.columns = columns
        self.playerCount = playerCount
        self.playerList = []
        self.round = 1

        self.posList = []
        self.firstGuardiansPosList = [[25, 25],
                                      [25, (self.columns * 50) - 25],
                                      [(self.rows * 50) - 25, 25],
                                      [(self.rows * 50) - 25, (self.columns * 50) - 25]]

        self.clickedX = None
        self.clickedY = None
        self.currentPlayer = None

        for i in range(playerCount):
            player = Player(i + 1)
            self.playerList.append(player)

        self.canvas = tk.Canvas(root, width=columns * 50, height=rows * 50, bg="grey")
        self.canvas.pack()
        self.canvas.bind("<Button-1>", self._click)
        # Oyun tahtası tek bir butona sahip. Yerleştirmeler oyuncunun mouse'unun koordinatları alınarak yapılıyor.

        self.draw_board()
        self._round_system()
```

def__init__ metodu: Bu metod, GameBoard sınıfının yapıcı metodudur. Bu metod, root (Tkinter penceresi), rows (satır sayısı), columns (sütun sayısı) ve playerCount (oyuncu sayısı) gibi parametreler alır. Bu parametreler, oyun tahtasının ve oyuncuların özelliklerini belirlemek için kullanılır. Metodun işlevleri şunlardır:

root, rows, columns, playerCount ve diğer özelliklerin atamasını yapar.
Oyuncu listesini oluşturur ve her bir oyuncuyu belirtilen sayıda oluşturur.
Oyun tahtasını temsil etmek için bir Canvas bileşeni oluşturur ve görüntüyü root penceresine yerleştirir.
Fare tıklamalarını dinlemek için bir olay bağlayıcısı ekler (<Button-1>).
Oyun tahtasını ve oyunun başlangıç durumunu çizmek için _draw_board ve _round_system metodlarını çağırır.
_draw_board metodu: Bu metod, oyun tahtasının görünümünü oluşturur. Belirli bir satır ve sütun sayısına sahip bir ızgara çizerek oyun tahtasını oluşturur. Ayrıca, her oyuncunun başlangıç konumunu belirlemek için _set_starting_positions metodunu çağırır.

_round_system metodu: Bu metod, oyun turunun yönetimini sağlar. Sıranın hangi oyuncuda olduğunu belirler ve ilgili oyuncuya hareket yapma yetkisi verir. Oyunun dönemlerini izler ve oyuncuların sırasını yönetir.

Class Player

```python
class Player:
    def __init__(self, playerNum):
        self.playerNum = playerNum
        self.priority = playerNum
        self.balance = 99999

        self.firstGuardianPosX = None
        self.firstGuardianPosY = None

        self.playerSoldiersDict = {"Muhafız": [],
                                   "Okçu": [],
                                   "Topçu": [],
                                   "Atlı": [],
                                   "Sıhhiyeci": []}

        self.playerGreenSquares = []
        self.playerSoldiersList = []
        self.moveCount = 2
        self.skippedRounds = []

        if playerNum == 1:
            self.color = "red"

        elif playerNum == 2:
            self.color = "blue"

        elif playerNum == 3:
            self.color = "green"

        elif playerNum == 4:
            self.color = "yellow"
```

Bu sınıf, bir oyun içindeki bir oyuncuyu temsil etmek için kullanılır. def__init__ metodu, bir oyuncunun özelliklerini başlatır. Oyuncunun numarasını (playerNum), önceliğini (priority), bakiyesini (balance), ilk koruyucunun pozisyonunu (firstGuardianPosX ve firstGuardianPosY), sahip olduğu askerleri (playerSoldiersDict), yeşil karelerini (playerGreenSquares), asker listesini (playerSoldiersList), hareket sayısını (moveCount) ve atlanmış tur sayısını (skippedRounds) ayarlar.

Ayrıca, oyuncunun numarasına göre rengini (color) belirler. 1 numaralı oyuncu kırmızı, 2 numaralı oyuncu mavi, 3 numaralı oyuncu yeşil ve 4 numaralı oyuncu sarı renkte olacaktır.

playerSoldiersDict, oyuncunun sahip olduğu her bir asker tipinin bir listesini içeren bir sözlüktür. Bu listede, "Muhafız", "Okçu", "Topçu", "Atlı" ve "Sıhhiyeci" adlı asker tipleri bulunmaktadır.

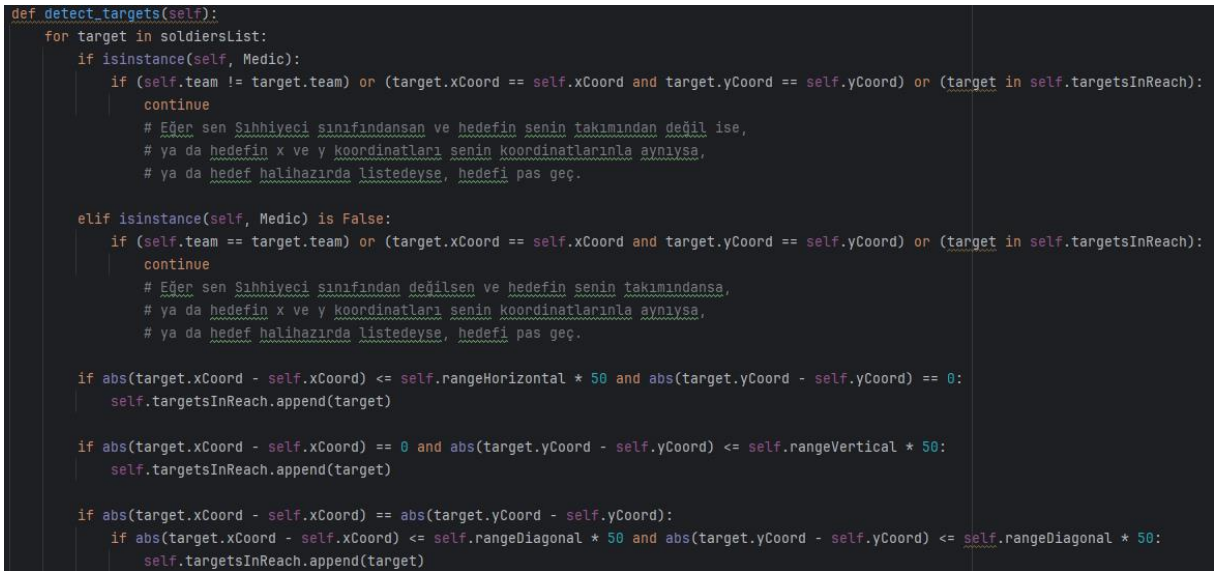
Class Soldier

```python
class Soldier:
    def __init__(self, name, cost, health, rangeHorizontal, rangeVertical, rangeDiagonal, dmg=None, dmgPercentage=None,
                 team=None, xCoord=None, yCoord=None):
        self.name = name
        self.cost = cost
        self.health = health
        self.rangeHorizontal = rangeHorizontal
        self.rangeVertical = rangeVertical
        self.rangeDiagonal = rangeDiagonal
        self.dmg = dmg
        self.dmgPercentage = dmgPercentage

        self.place = None
        self.team = team
        self.xCoord = xCoord
        self.yCoord = yCoord

        self.targetsInReach = []
        self.killedTargets = []
```

Bu kod, Soldier adında bir sınıf tanımlar. Bu sınıf, bir askeri karakteri temsil etmek için kullanılır.
def__init__ metodunda, bir asker nesnesinin özellikleri başlatılır. Bu özellikler arasında askerin adı (name), maliyeti (cost), sağlık puanı (health), yatay menzili (rangeHorizontal), dikey menzili (rangeVertical), çapraz menzili (rangeDiagonal), verdiği hasar (dmg), hasar yüzdesi (dmgPercentage), takımı (team), konumu (xCoord ve yCoord) bulunur.

Ayrıca, askerin mevcut pozisyonunu belirlemek için place, hedeflere erişilebilirlik listesi için targetsInReach ve öldürülen hedeflerin listesi için killedTargets öznitelikleri bulunur

Def detect_targets(self):



```
def detect_targets(self):
    for target in soldiersList:
        if isinstance(self, Medic):
            if (self.team != target.team) or (target.xCoord == self.xCoord and target.yCoord == self.yCoord) or (target in self.targetsInReach):
                continue
                # Eğer sen Sıhhiyeci sınıfındansan ve hedefin senin takımından değil ise,
                # ya da hedefin x ve y koordinatları senin koordinatlarınla aynıysa,
                # ya da hedef halihazırda listedeyse, hedefi pas geç.

        elif isinstance(self, Medic) is False:
            if (self.team == target.team) or (target.xCoord == self.xCoord and target.yCoord == self.yCoord) or (target in self.targetsInReach):
                continue
                # Eğer sen Sıhhiyeci sınıfından değilsen ve hedefin senin takımındansa,
                # ya da hedefin x ve y koordinatları senin koordinatlarınla aynıysa,
                # ya da hedef halihazırda listedeyse, hedefi pas geç.

        if abs(target.xCoord - self.xCoord) <= self.rangeHorizontal * 50 and abs(target.yCoord - self.yCoord) == 0:
            self.targetsInReach.append(target)

        if abs(target.xCoord - self.xCoord) == 0 and abs(target.yCoord - self.yCoord) <= self.rangeVertical * 50:
            self.targetsInReach.append(target)

        if abs(target.xCoord - self.xCoord) == abs(target.yCoord - self.yCoord):
            if abs(target.xCoord - self.xCoord) <= self.rangeDiagonal * 50 and abs(target.yCoord - self.yCoord) <= self.rangeDiagonal * 50:
                self.targetsInReach.append(target)
```

Bu fonksiyon, bir askerin ulaşabileceği hedefleri belirlemek için kullanılır. self ile belirtilen asker nesnesi için mevcut konumundan, menzil sınırları içindeki düşman askerleri tespit eder.

İlk olarak, Sıhhiyeci (Medic) sınıfından bir asker mi yoksa başka bir sınıftan bir asker mi olduğuna bakılır. Eğer asker bir Sıhhiyeci ise, hedefin kendi takımından olmaması ve hedefin mevcut pozisyonunun ya da hedefin zaten ulaşılabilir hedefler listesinde olmaması durumlarında hedef tespiti yapılır. Diğer asker sınıfları için ise, hedefin kendi takımından olması ve diğer koşulların kontrolü yapılır.

Daha sonra, hedefin konumu ile askerin menzili içindeki hedefleri tespit etmek için koordinatlar arasındaki mesafe kontrol edilir. Eğer hedefin x koordinatındaki fark, askerin yatay menzilinin katlarından biri ise ve y koordinatındaki fark sıfırsa, hedef yatay menzilde demektir. Benzer şekilde, eğer hedefin y koordinatındaki fark, askerin dikey menzilinin katlarından biri ise ve x koordinatındaki fark sıfırsa, hedef dikey menzilde demektir. Son olarak, eğer hedefin x ve y koordinatları arasındaki farklar eşit ise, hedefin çapraz menzilde olduğu kontrol edilir.

Her bir kontrol durumu için, hedef asker, self.targetsInReach listesine eklenir. Bu liste, askerin ulaşabileceği hedefleri saklamak için kullanılır. Bu fonksiyon, askerin etkili bir şekilde hedeflerini belirlemesine yardımcı olur ve oyun içinde stratejik kararlar almasına olanak tanır.

Def attack(self):

```
def attack(self):
    self.detect_targets()
    # Yukarıdaki fonksiyonu kullanarak etrafını tara.

    for target in self.targetsInReach:
        if target.team != self.team:
            if self.dmg is not None:
                target.health -= self.dmg
                # Eğer etrafındaki asker düşmanınsa ve yüzdelik hasar yerine mutlak hasar vurabiliyorsan,
                # düşmanın canını hasarın kadar azalt.

            elif self.dmgPercentage is not None:
                dmgDealt = target.health * self.dmgPercentage / 100
                target.health -= dmgDealt

        if target.health <= 0:
            print(f"{target.name}{target.place}, {self.name}{self.place} tarafından öldürüldü!")
            self.killedTargets.clear()
            # Askerin tek rauntta öldürdüğü askerleri bu listede tutmak istiyoruz.
            # Bu yüzden ekleme yapmadan önce listeyi boşalt.

            self.killedTargets.append(target)
```

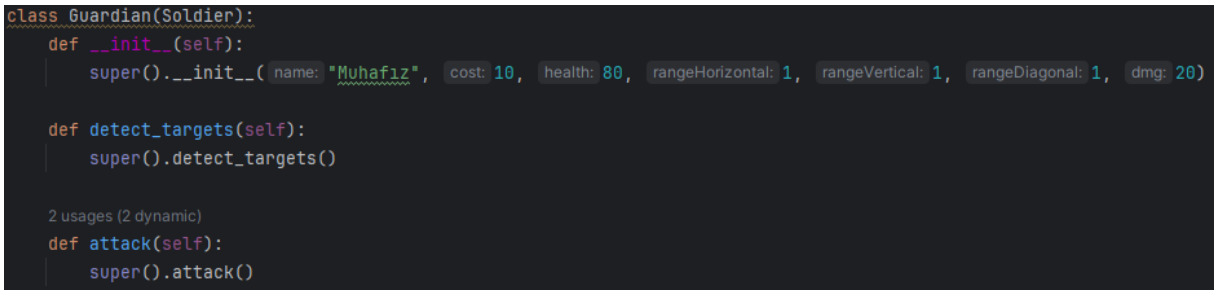
Bu fonksiyon, bir askerin saldırı yapmasını sağlar. İlk olarak, detect_targets() fonksiyonu çağrılarak askerin etrafındaki hedefler tespit edilir.

Ardından, askerin ulaşabileceği hedefler listesi (self.targetsInReach) üzerinde döngü başlatılır. Her bir hedef için:

Eğer hedef, askerin takımından farklı bir takıma aitse, asker saldırı yapar.
Eğer asker, hedefe mutlak hasar verebiliyorsa (self.dmg değeri None değilse), hedefin sağlık puanı, askerin hasarı kadar azaltılır.
Eğer asker, hedefe yüzdelik hasar verebiliyorsa (self.dmgPercentage değeri None değilse), hedefin mevcut sağlık puanının belirtilen yüzde kadarını azaltacak hasar hesaplanır ve hedefin sağlık puanı bu kadar azalır.
Her saldırı sonrasında hedefin sağlık puanı kontrol edilir. Eğer hedefin sağlık puanı 0 veya daha az ise, hedefin öldürüldüğü belirtilir ve bu bilgi ekrana yazdırılır. Ayrıca, öldürülen hedef self.killedTargets listesine eklenir.

Bu fonksiyon, bir askerin etrafındaki hedeflere saldırmasını, hasar vermesini ve öldürdüğü hedefleri takip etmesini sağlar. Oyun içinde askerler arasındaki savaş mekaniklerini yönetmek için kullanılır.

Class Guardian(Soldier):



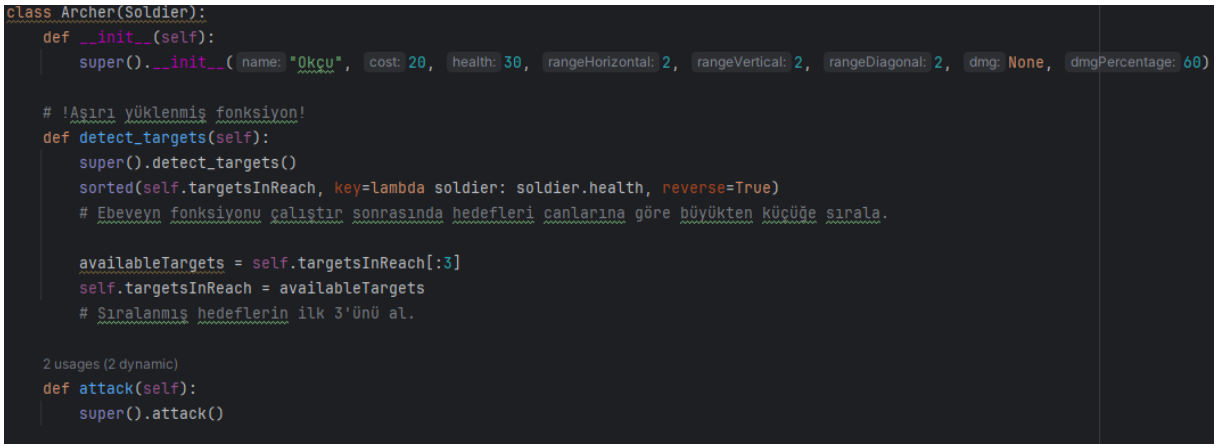
Bu sınıf, Soldier sınıfından türetilen Guardian adı verilen bir asker sınıfıdır. Soldier sınıfından türetilmesi, Guardian sınıfının özelliklerini ve davranışlarını miras aldığı anlamına gelir.

Guardian sınıfının __init__ yöntemi, nöbetçi askerin niteliklerini belirler. Muhafızın adı "Guardian", maliyeti 10, sağlığı 80, yatay menzili 1, dikey menzili 1, çapraz menzili 1 ve hasarı 20'dir.

Bu sınıf, Soldier sınıfından miras alınan dedektör_targetleri ve Saldırı yöntemlerini geçersiz kılar. Başka bir deyişle Guardian sınıfı, devralınan bu yöntemleri kendi ihtiyaçlarına uyacak şekilde yeniden tanımlıyor.

Detector_targets yöntemi, koruma etrafındaki hedefleri tespit etmek için kullanılır. Saldırı yöntemi koruma saldırılarına izin verir. Ancak bu yöntemler sınıf içinde yeniden tanımlanmaz ve yalnızca miras alınan Soldier sınıfındaki aynı isimdeki yöntemler çağrılır. Bu durumda Guardian'ın hedef tespit ve saldırı yetenekleri, devraldığı Soldier sınıfındaki yöntemlerin yeteneklerine bağlı olacaktır.

Class Archer(Soldier):



```
class Archer(Soldier):
    def __init__(self):
        super().__init__( name: "Okçu", cost: 20, health: 30, rangeHorizontal: 2, rangeVertical: 2, rangeDiagonal: 2, dmg: None, dmgPercentage: 60)

    # !Aşırı yüklenmiş fonksiyon!
    def detect_targets(self):
        super().detect_targets()
        sorted(self.targetsInReach, key=lambda soldier: soldier.health, reverse=True)
        # Ebeveyn fonksiyonu çalıştır sonrasında hedefleri canlarına göre büyükten küçüğe sırala.

        availableTargets = self.targetsInReach[:3]
        self.targetsInReach = availableTargets
        # Sıralanmış hedeflerin ilk 3'ünü al.

    2 usages (2 dynamic)
    def attack(self):
        super().attack()
```

Bu sınıf, Soldier sınıfından türetilen Archer adı verilen bir asker sınıfıdır. Soldier sınıfından türetildiği için bu, Archer sınıfının işlevselliğini ve davranışını devraldığı anlamına gelir.

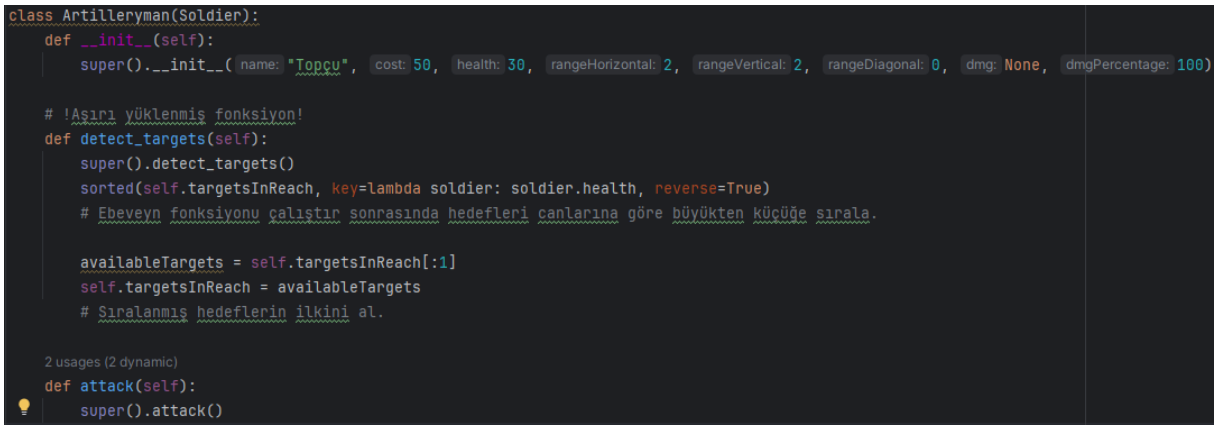
Archer sınıfının __init__ metodu okçu askerin niteliklerini belirler. Okçunun adı "Okçu" olup maliyeti 20, sağlığı 30, yatay menzili 2, dikey menzili 2 ve çapraz menzili 2'dir. Okçu hasarı yüzdeye dayalıdır (%60).

Soldier sınıfından miras alınan dedektör_targetleri ve Saldırı yöntemlerini geçersiz kılar. Okçunun etrafındaki hedefleri tespit etmek için dedektör_targets yöntemi kullanılır. Bu yöntem öncelikle miras alınan Soldier sınıfı üzerinde aynı isimli yöntemi çağırır ve daha sonra tespit edilen hedefleri sağlık değerlerine göre büyükten küçüğe doğru sıralar. Sıralanan hedeflerden ilk üçünü alarak self.targetsInReach listesini günceller.

Attack metodu ise, okçunun saldırı yapmasını sağlar. Ancak, bu metodlar sınıfın içinde yeniden tanımlanmamıştır, sadece miras alınan Soldier sınıfındaki aynı isimli metodlar çağrılmıştır. Bu durumda, okçunun hedef tespiti ve saldırı yapma yetenekleri, miras aldığı Soldier sınıfındaki metodların işlevselliğine bağlı olacaktır.

Bu sınıfın amacı, okçu askerinin temel özelliklerini ve davranışlarını tanımlamak ve gerektiğinde bu davranışları özelleştirmek için kullanılır. Örneğin, detect_targets metodunda hedefleri sağlık puanlarına göre sıralayarak, okçunun daha stratejik bir şekilde hedef seçmesi sağlanmıştır.

Class Artilleryman(Soldier):



```
class Artilleryman(Soldier):
    def __init__(self):
        super().__init__( name: "Topçu",  cost: 50,  health: 30,  rangeHorizontal: 2,  rangeVertical: 2,  rangeDiagonal: 0,  dmg: None,  dmgPercentage: 100)

    # !Aşırı yüklenmiş fonksiyon!
    def detect_targets(self):
        super().detect_targets()
        sorted(self.targetsInReach, key=lambda soldier: soldier.health, reverse=True)
        # Ebeveyn fonksiyonu çalıştır sonrasında hedefleri canlarına göre büyükten küçüğe sırala.

        availableTargets = self.targetsInReach[:1]
        self.targetsInReach = availableTargets
        # Sıralanmış hedeflerin ilkini al.

    2 usages (2 dynamic)
    def attack(self):
        super().attack()
```

Bu sınıf, Artilleryman adında bir asker sınıfıdır ve Soldier sınıfından türetilmiştir. Soldier sınıfından türetilmiş olması, Artilleryman sınıfının özelliklerini ve davranışlarını miras aldığı anlamına gelir.

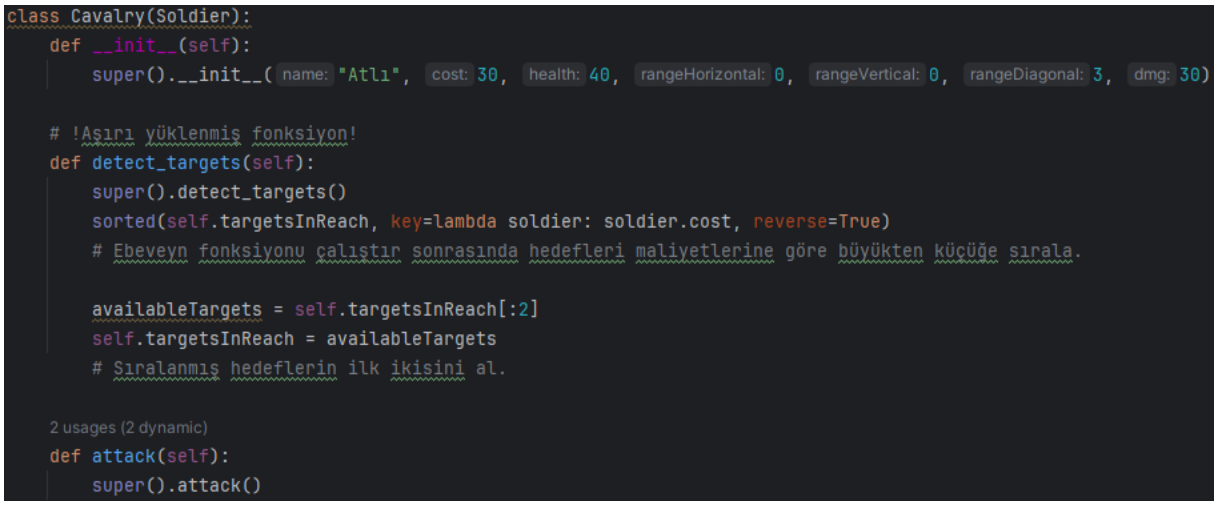
Artilleryman sınıfının __init__ metodu, bir topçu askerinin özelliklerini belirler. Topçunun adı "Topçu", maliyeti 50, sağlık puanı 30, yatay menzili 2, dikey menzili 2, çapraz menzili 0 olarak belirlenmiştir. Topçunun verdiği hasar, yüzdelik tabanlı olarak belirlenmiştir (%100).

Bu sınıf, Soldier sınıfından miras aldığı detect_targets ve attack metodlarını override etmektedir. detect_targets metodu, topçu askerinin etrafındaki hedefleri tespit etmek için kullanılır. Bu metod, önce miras alınan Soldier sınıfındaki aynı adlı metodu çağırır ve ardından tespit edilen hedefleri sağlık puanlarına göre büyükten küçüğe sıralar. Sıralanmış hedefler arasından ilkini alarak self.targetsInReach listesini günceller.

attack metodu ise, topçu askerinin saldırı yapmasını sağlar. Ancak, bu metodlar sınıfın içinde yeniden tanımlanmamıştır, sadece miras alınan Soldier sınıfındaki aynı isimli metodlar çağrılmıştır. Bu durumda, topçu askerinin hedef tespiti ve saldırı yapma yetenekleri, miras aldığı Soldier sınıfındaki metodların işlevselliğine bağlı olacaktır.

Bu sınıfın amacı, topçu askerinin temel özelliklerini ve davranışlarını tanımlamak ve gerektiğinde bu davranışları özelleştirmek için kullanılır. Örneğin, detect_targets metodunda hedefleri sağlık puanlarına göre sıralayarak, topçu askerinin daha stratejik bir şekilde hedef seçmesi sağlanmıştır. Bu durumda topçu, en az sağlık puanına sahip hedefe odaklanarak, düşmanın en zayıf noktalarına saldıracaktır.

Class Cavalry(Soldier):



Bu sınıf, Cavalry adında bir asker sınıfıdır ve Soldier sınıfından türetilmiştir. Soldier sınıfından türetilmiş olması, Cavalry sınıfının özelliklerini ve davranışlarını miras aldığı anlamına gelir.

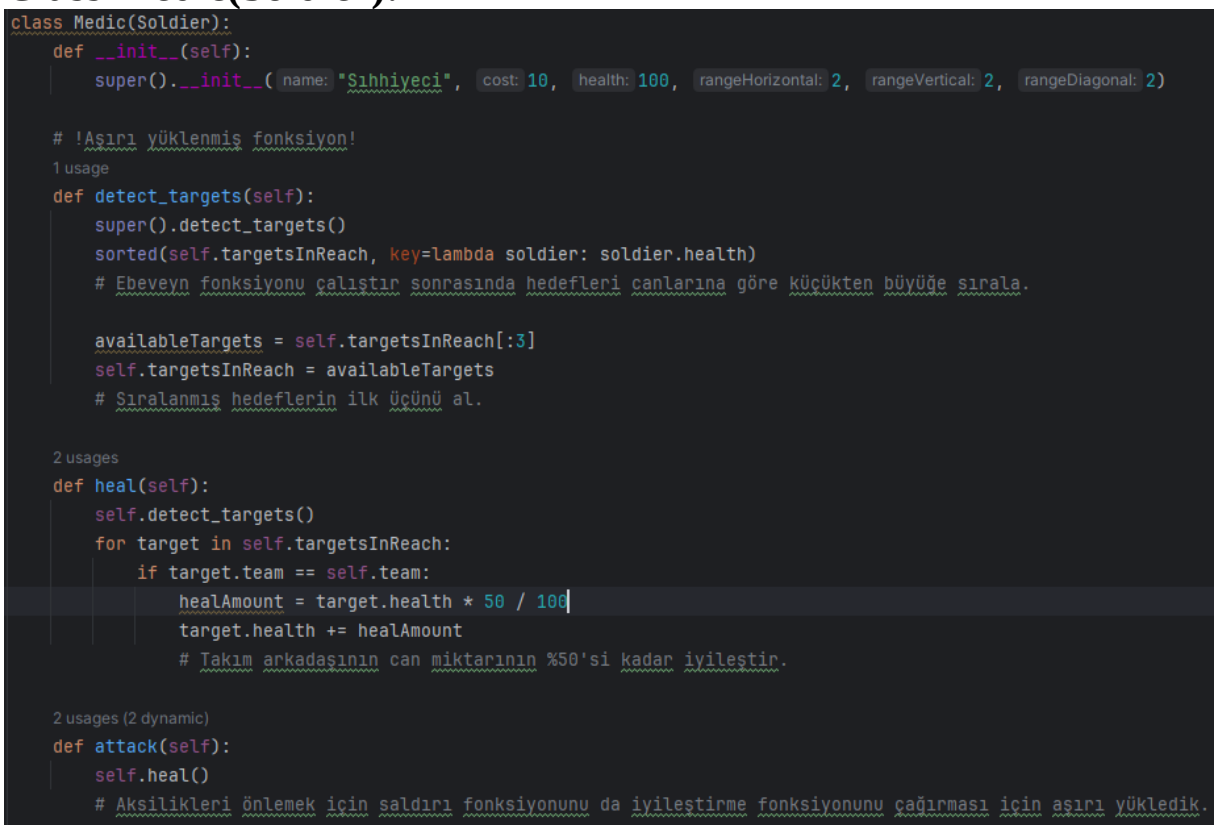
Cavalry sınıfının __init__ metodu, bir süvari askerinin özelliklerini belirler. Süvarinin adı "Atlı", maliyeti 30, sağlık puanı 40, yatay menzili 0, dikey menzili 0, çapraz menzili 3 olarak belirlenmiştir. Süvari askerinin verdiği hasar 30'dur.

Bu sınıf, Soldier sınıfından miras aldığı detect_targets ve attack metodlarını override etmektedir. detect_targets metodu, süvari askerinin etrafındaki hedefleri tespit etmek için kullanılır. Bu metod, önce miras alınan Soldier sınıfındaki aynı adlı metodu çağırır ve ardından tespit edilen hedefleri maliyetlerine göre büyükten küçüğe sıralar. Sıralanmış hedefler arasından ilk ikisini alarak self.targetsInReach listesini günceller.

attack metodu ise, süvari askerinin saldırı yapmasını sağlar. Ancak, bu metodlar sınıfın içinde yeniden tanımlanmamıştır, sadece miras alınan Soldier sınıfındaki aynı isimli metodlar çağrılmıştır. Bu durumda, süvari askerinin hedef tespiti ve saldırı yapma yetenekleri, miras aldığı Soldier sınıfındaki metodların işlevselliğine bağlı olacaktır.

Bu sınıfın amacı, süvari askerinin temel özelliklerini ve davranışlarını tanımlamak ve gerektiğinde bu davranışları özelleştirmek için kullanılır. Örneğin, detect_targets metodunda hedefleri maliyetlerine göre sıralayarak, süvari askerinin daha stratejik bir şekilde hedef seçmesi sağlanmıştır. Bu durumda süvari, düşmanın en değerli ve etkili birimlerine odaklanarak saldırı yapacaktır.

Class Medic(Soldier):



```
class Medic(Soldier):
    def __init__(self):
        super().__init__( name: "Sıhhiyeci",  cost: 10,  health: 100,  rangeHorizontal: 2,  rangeVertical: 2,  rangeDiagonal: 2)

    # !Aşırı yüklenmiş fonksiyon!
    1 usage
    def detect_targets(self):
        super().detect_targets()
        sorted(self.targetsInReach, key=lambda soldier: soldier.health)
        # Ebeveyn fonksiyonu çalıştır sonrasında hedefleri canlarına göre küçükten büyüğe sırala.

        availableTargets = self.targetsInReach[:3]
        self.targetsInReach = availableTargets
        # Sıralanmış hedeflerin ilk üçünü al.

    2 usages
    def heal(self):
        self.detect_targets()
        for target in self.targetsInReach:
            if target.team == self.team:
                healAmount = target.health * 50 / 100
                target.health += healAmount
                # Takım arkadaşının can miktarının %50'si kadar iyileştir.

    2 usages (2 dynamic)
    def attack(self):
        self.heal()
        # Aksilikleri önlemek için saldırı fonksiyonunu da iyileştirme fonksiyonunu çağırması için aşırı yükledik.
```

Bu sınıf, Medic adında bir asker sınıfıdır ve Soldier sınıfından türetilmiştir. Soldier sınıfından türetilmiş olması, Medic sınıfının özelliklerini ve davranışlarını miras aldığı anlamına gelir.

Medic sınıfının __init__ metodu, bir sıhhiyeci askerinin özelliklerini belirler. Sıhhiyecinin adı "Sıhhiyeci", maliyeti 10, sağlık puanı 100, yatay menzili 2, dikey menzili 2, çapraz menzili 2 olarak belirlenmiştir.

Bu sınıf, Soldier sınıfından miras aldığı detect_targets ve attack metodlarını override etmektedir. detect_targets metodu, sıhhiyeci askerinin etrafındaki hedefleri tespit etmek için kullanılır. Bu metod, önce miras alınan Soldier sınıfındaki aynı adlı metodu çağırır ve ardından tespit edilen hedefleri sağlık puanlarına göre küçükten büyüğe sıralar. Sıralanmış hedefler arasından ilk üçünü alarak self.targetsInReach listesini günceller.

heal metodu, sıhhiyeci askerin iyileştirme işlemini gerçekleştirir. Bu metod, sıhhiyecinin etrafındaki takım arkadaşlarının sağlık puanlarını iyileştirir. Sıhhiyeci, etrafındaki üç takım arkadaşının sağlık puanlarının %50'si kadarını iyileştirir.

attack metodu, sıhhiyeci askerin hem iyileştirme hem de saldırı yapmasını sağlar. Bu metod, heal metodunu çağırarak takım arkadaşlarını iyileştirir ve

böylelikle sıhhiyeci hem saldırı yapar hem de takım arkadaşlarını korur.

Bu sınıfın amacı, sıhhiyeci askerin temel özelliklerini ve davranışlarını tanımlamak ve gerektiğinde bu davranışları özelleştirmek için kullanılır. Örneğin, detect_targets metodunda hedefleri sağlık puanlarına göre sıralayarak, sıhhiyeci askerin daha etkili bir şekilde takım arkadaşlarını iyileştirmesi sağlanmıştır. Bu durumda sıhhiyeci, en az sağlık puanına sahip olan takım arkadaşlarını öncelikli olarak iyileştirir ve bu sayede takımın dayanıklılığını artırır.

def start():

```
def start():
    inputList = [0]
    boardSizeRows, boardSizeCols = 0, 0
    playerCount = 0

    while len(inputList) != 2 or (
            (boardSizeRows < 8 or boardSizeRows > 32) or (boardSizeCols < 8 or boardSizeCols > 32)):
        inputList = input("Enter the board size: ").split(",")
        boardSizeRows, boardSizeCols = int(inputList[0]), int(inputList[1])

    while playerCount < 2 or playerCount > 4:
        playerCount = int(input("Enter the number of players: "))

    root = tk.Tk()
    game_board = GameBoard(root, boardSizeRows, boardSizeCols, playerCount)
    root.mainloop()

start()
```

Bu start() fonksiyonu, oyunun başlatılmasını sağlar. Fonksiyon, kullanıcıdan gerekli girişleri alarak oyun tahtasını ve oyuncu sayısını belirler.
İlk olarak, bir inputList oluşturulur ve başlangıçta 0 değeri ile doldurulur. Bu liste, kullanıcının girdiği tahta boyutunu ve oyuncu sayısını saklamak için kullanılır. Daha sonra boardSizeRows, boardSizeCols ve playerCount değişkenleri sırasıyla tahta satır sayısını, tahta sütun sayısını ve oyuncu sayısını saklamak için tanımlanır.

İlk while döngüsü, kullanıcıdan geçerli bir tahta boyutu girmesini bekler. Kullanıcıdan alınan girdi, virgülle ayrılmış iki tam sayı olmalıdır. Girdi doğru formatta ve belirli bir aralıkta değilse (örneğin, en az 8x8 ve en fazla 32x32 olmalıdır), kullanıcıdan girdiyi tekrar girmesini isteyen bir döngü başlatılır.

İkinci while döngüsü, kullanıcıdan geçerli bir oyuncu sayısı girmesini bekler. Oyuncu sayısı en az 2 ve en fazla 4 olabilir. Geçerli bir oyuncu sayısı girilene kadar döngü devam eder.

Girilen tahta boyutu ve oyuncu sayısı geçerli olduğunda, tkinter modülü kullanılarak bir Tkinter penceresi oluşturulur. Bu pencere, oyun tahtası için bir arayüz sağlar. GameBoard sınıfından bir nesne oluşturulur ve bu nesne, Tkinter penceresi içinde görüntülenir.
Son olarak, root.mainloop() çağrısıyla Tkinter uygulaması başlatılır ve kullanıcı arayüzü görüntülenir. Kullanıcı bu arayüz üzerinden oyunu oynamaya başlayabilir.

Bu fonksiyon, oyunun başlatılmasını sağlayarak, oyuncuların ve tahta boyutunun belirlenmesini sağlar.

4.2 Görev dağılımı

Asker ve özellikler sınıfının oluşturulması Talha Tuna öğrencisine aittir. Oyun tahtası ve oyuncu sınıflarının oluşturulması Yavuz Selim Gürsoy öğrencisine aittir. Projenin geliştirilme sürecinde iki öğrenci de birbirinden yardım almıştır.

4.3 Karşılaşılan zorluklar ve çözüm yöntemleri

- Tkinter kütüphanesini bilmiyorduk. Çözüm olarak tarayıcıdan bakarak öğrendik.

4.4 Proje isterlerine göre eksik yönler

- Projede ele geçirilen karoların rengi oyuncunun rengi ile eşleştirilmeliydi ancak biz ele geçirilen karoların rengini yeşil renk yaptık.

5 TEST VE DOĞRULAMA

5.1 Yazılımın test süreci

Yazılımın test edilmesi için herhangi bir test programı yazmak yerine Python'da ki debugger uygulamasını kullandık. Bunun yanında projemizi geliştirirken her aşamadan sonra uygulamamızı çalıştırıp test ettik.

6 GİT HUB LİNKLERİ

Talha Tuna:
Yavuz Selim Gürsoy: