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

PROJE KONUSU:
LORDS OF THE POLYWARPHISM

ÖĞRENCİ ADI:
ÖĞRENCİ NUMARASI:
Senem ADALAN - 220502045
Sema Su YILMAZ - 220502016

DERS SORUMLUSU:
PROF. DR. NEVCİHAN DURU

TARİH:
24 MART 2024

İÇİNDEKİLER

1. GİRİŞ

1.1 Projenin Amacı

Bu projenin amacı 2-boyutlu matris formunda bir dünyada çok oyunculu bir savaş oyunu gerçekleştirmektir.Projede gerçekleştirilmesi beklenen işlemler şu şekildedir;

- Boyut seçimi ve kontrolü
- Oyuncu sayısının belirlenmesi ve kontrolü
- Oyuncu sırası ve hamle yapma
- Savaşçı üretme ve yerleştirme
- Saldırıların gerçekleştirilmesi
- Muhafızların yerleştirilmesi
- Savaşçıların etkileşimi ve kontrolü
- Kaynak ve can yönetimi
- Oyun bitiş koşulları
- Kazananın belirlenmesi
- Nesne yönelimli programlama (OOP) prensiplerinin uygulanması

2. GEREKSİNİM ANALİZİ

2.1 Arayüz Gereksinimleri

- **Oyun Başlangıç Ekranı:** Oyunun başladığı ana ekrandır.Bu ekranda dünya boyutu seçme seçeneği bulunmalıdır.Ayrıca oyuncu sayısını belirleme seçeneği de yer almalıdır.

- **Savaşçı Seçim Ekranı:** Gerçek oyuncuların veya yapay zekanın savaşçılarını seçtikleri alandır.Savaşçıların isimleri ve sıraları bu ekranda yer almalıdır.

- **Savaşçı Yerleştirme Ekranı:** Yeni savaşçıların yerleştirilmesi için tasarlanan işlemlerdir.Savaşçı türü seçimi yapılmalıdır.Yerleştirilebilecek uygun hücreler gösterilmelidir.Koordinat seçimi yapılmalı ve yerleştirme işlemi tamamlandıktan sonra oyuna devam edilmelidir.

- **Oyun Alanı Ekranı:** Oyunun gerçekleştiği ana ekrandır.2 boyutlu vektör kullanılarak düzenlenmiş alandır.Her oyuncunun sahip olduğu savaşçılar, boş hücreler, her oyuncunun yerleşim yapabileceği muhtemel alanları ve oyuncuların kaynak bilgilerinin modellenmesi gereken kısımdır.

2.2 Fonksiyonel Gereksinimler

Oyun Başlangıç Ekranı (İG-1)

- Yeni Oyun Başlatma Yeteneği: Kullanıcı yeni bir oyun başlatabilmelidir.
- Dünya Boyutu Seçme: Kullanıcı oyun başlangıcında dünya boyutunu (16x16, 24x24, 32x32 veya özel boyut) seçebilmelidir.
- Oyuncu Sayısını Belirleme: Kullanıcı oyun başlangıcında oyuncu sayısını (1-4 arası) belirleyebilmelidir.

Savaşçı Seçim Ekranı (İG-2)

- Savaşçı Seçimi: Gerçek oyuncular veya yapak zeka savaşçılarını seçebilmelidir.
- Savaşçı Türlerinin Listesi: Kullanıcılar tüm savaşçı türlerinin listesine erişebilmelidir.
- Savaşçı Yerleştirme Koordinatları: Kullanıcılar seçtikleri savaşçıların yerleştirme koordinatlarını belirleyebilmelidir.

Savaşçı Yerleştirme Ekranı (İG-3)

- Savaşçı Türü Seçimi: Kullanıcılar yeni savaşçı eklerken tür seçimi yapabilmelidir.
- Uygun Yerlerin Gösterilmesi: Yeni savaşçı yerleştirilirken uygun hücrelerin gösterilmesi gerekmektedir.
- Koordinat Seçimi: Kullanıcılar savaşçıları yerleştirecekleri koordinatları seçebilmelidir.

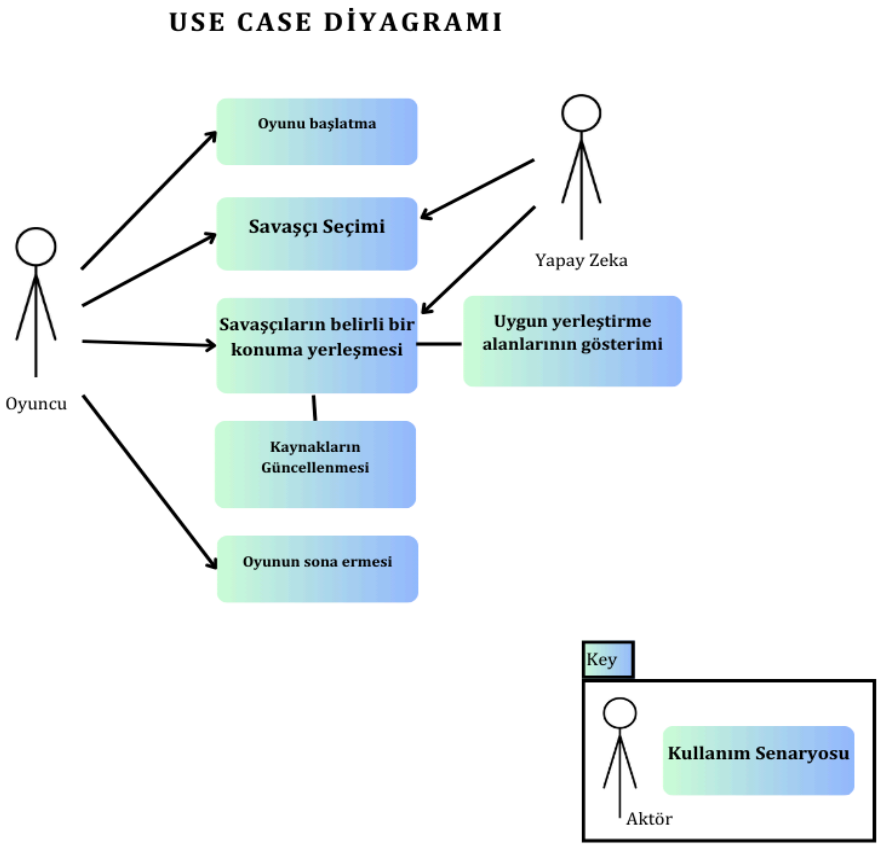
Oyun Alanı Ekranı (İG-4)

- Oyun Alanının Görüntülenmesi: 2 boyutlu vektör dünyası ekran üzerinde gösterilmelidir.
- Oyuncu Bilgileri: Oyuncuların sahip olduğu kaynaklar ve canlar ekran üzerinde görüntülenmelidir.

Oyun Sonu Ekranı (İG-5)

- Oyun Sonucunun Gösterilmesi: Kazanan veya kaybeden oyuncu ekranda belirtilmelidir.
- Yeniden Başlatma veya Ana Menüye Dönme Seçenekleri: Kullanıcı oyunu tekrar başlatma veya ana menüye dönme seçeneklerine erişebilmelidir.

2.3 Use-Case Diyagramı



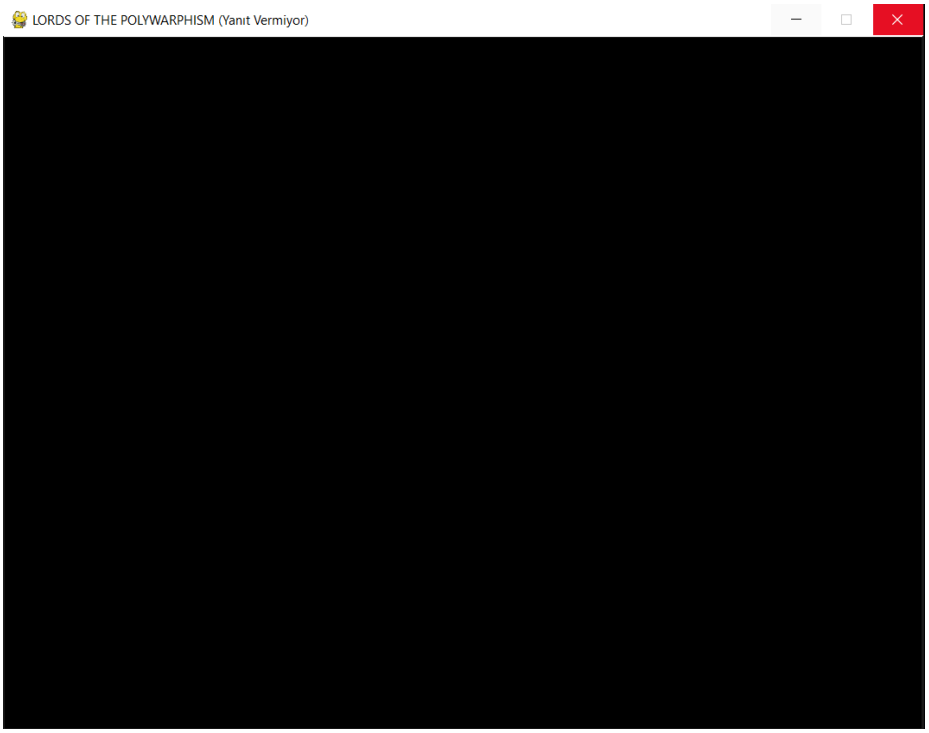
3. TASARIM
3.1 Mimari Tasarım
- Nesne Yönelimli Tasarım:
    - Proje, nesne yönelimli programlama prensiplerine dayalı olarak tasarlanmalıdır.
    - Her savaşçı türü için ayrı bir sınıf oluşturulmalıdır.
    - Savaşçılar arasında ortak özellikler ve davranışlar tanımlanmalıdır.
- Oyun Dünyası Tasarımı:
    - Oyun dünyası 2 boyutlu bir matris olarak temsil edilmelidir.
    - Matris, standart vektör veri yapısı kullanılarak oluşturulmalıdır.
    - Her hücre, boş veya bir savaşçıya ait olabilir.
- Kullanıcı Tasarımı:
    - Kullanıcıların savaşçı seçimi ve yerleştirme işlemlerini yapabilecekleri bir arayüz tasarlanmalıdır.
    - Kullanıcılar, seçimlerini menüler aracılığıyla yapmalı ve savaşçıları matris üzerine yerleştirmek için koordinatları girmelidir.
- Yapay Zeka Tasarımı:
    - Yapay zeka, insan oyuncular gibi savaşçı seçimi ve yerleştirme işlemlerini yapabilmelidir.
    - Yapay zeka, stratejik kararlar alarak hamlelerini gerçekleştirmelidir.

3.2 Kullanılacak Teknolojiler
- Bu projede Python programlama dili kullanılmıştır.
- Pygame modülü kullanılarak oyunun grafik arayüzü oluşturulmuştur. 2 boyutlu matrisin tasarlanması ile oyunun modellenmesi aşamalarında Pygame modülünden yararlanılmıştır.
- Sys modülü kullanıcının oyunu sonlandırma veya sonraki turlarda oyuna devam etme işlemlerinde kullanılmıştır. Kullanıcıya sunulan seçeneklere bağlı olarak, sys.exit() fonksiyonu çağrılarak program derhal sonlandırılmış veya oyunun devamı sağlanmıştır.
- Random modülü, oyunun başlangıç kısmında köşelere muhafızları rastgele yerleştirme işleminde kullanılmıştır. Bu sayede, her oyun başladığında farklı bir başlangıç durumu elde edilmiştir.
- Projede ABC (Abstract Base Class) kullanıldığı için ABC modülü de kullanılmıştır. Savasci sınıfı soyut bir temel sınıf olarak tanımlanmış ve bu sınıftan türetilen alt sınıfların metotlarını ve özelliklerini tanımlamak için ABC modülünden yararlanılmıştır.

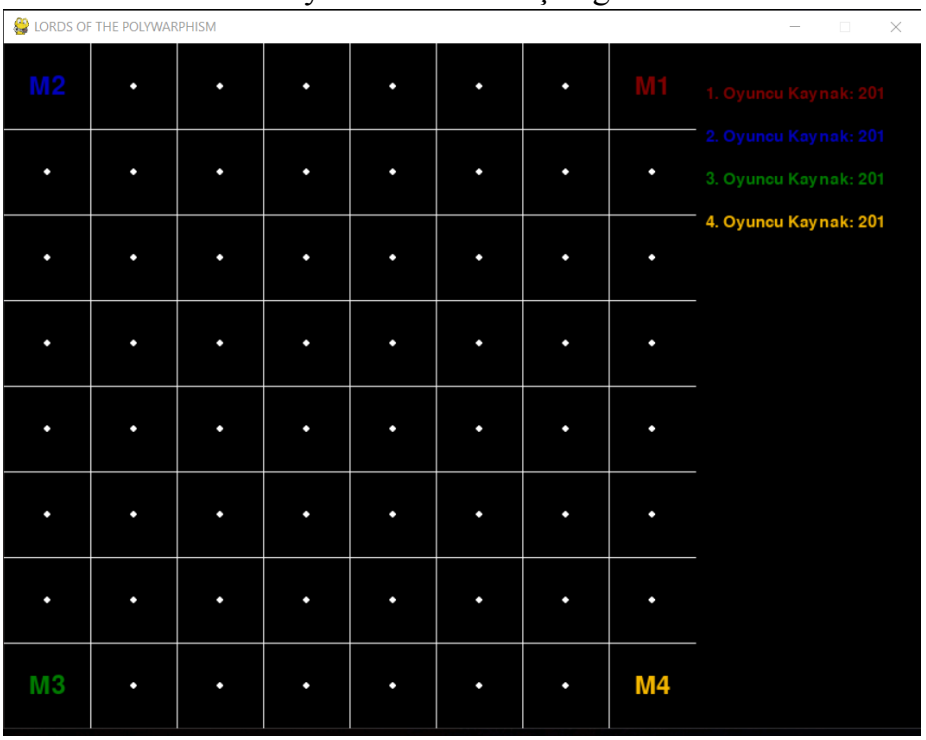
3.3 Kullanıcı Arayüzü Tasarımı

- Savaşçı Seçimi:Kullanıcı arayüzü, kullanıcılara mevcut savaşçı türleri ve özellikleri hakkında bilgi sunmalıdır.
- Savaşçı Yerleştirme:Kullanıcılar, seçtikleri savaşçıları oyun dünyasının belirli bir konumuna yerleştirmelidir.Kullanıcı arayüzü, kullanıcılara oyun dünyasının boyutunu göstermeli ve savaşçıları yerleştirebilecekleri uygun konumları işaretlemelidir. Kullanıcılar, savaşçıları yerleştirmek için matris koordinatlarını girmelidir.
- Oyun Durumu Gösterimi:Kullanıcı arayüzü, oyuncuların sırasını, sahip oldukları kaynak miktarını göstermelidir.

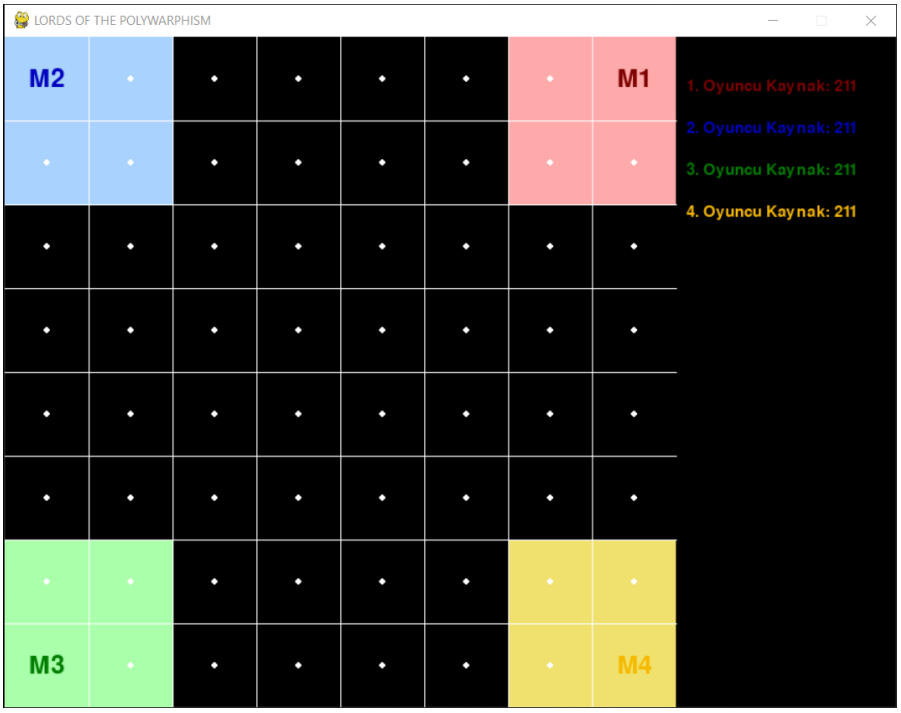
- Görselleştirme:Kullanıcı arayüzü, oyun dünyasının görsel olarak temsil edilmesini sağlamalıdır.Oyun dünyasındaki boş hücreler ve savaşçılar kullanıcı arayüzünde belirgin bir şekilde görüntülenmelidir.Görsel unsurlar, oyuncuların oyun durumunu daha kolay anlamalarına yardımcı olmalıdır.
- Hata Yönetimi:Kullanıcı arayüzü, kullanıcıların hatalı girişler yaptığında veya uygunsuz işlemler gerçekleştirdiğinde uygun hata mesajları göstermelidir.



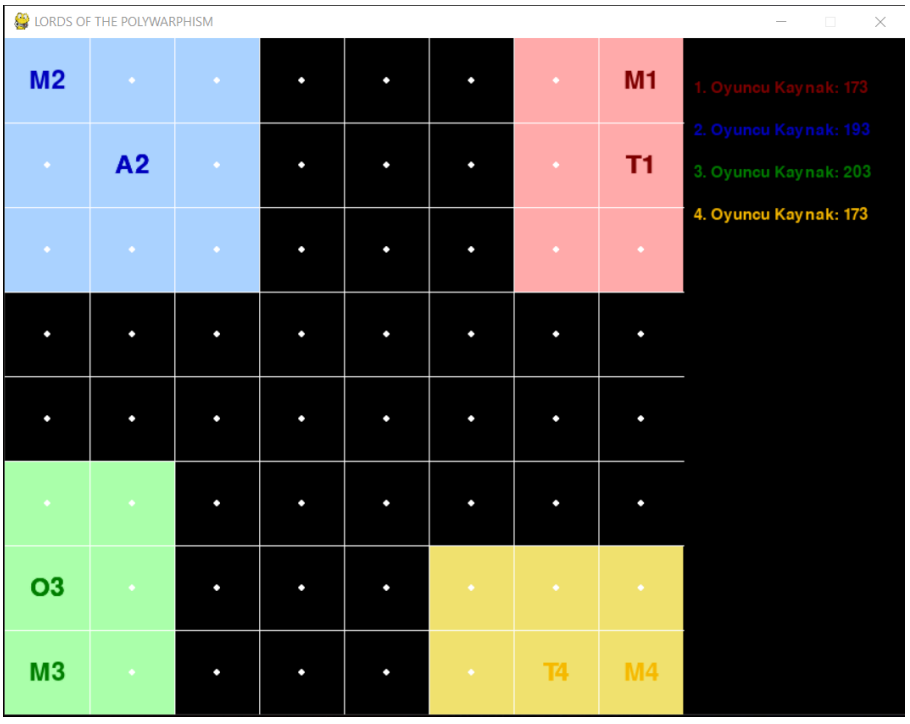
Oyun ekranının başlangıcı



Izgaranın oluşturulması, oyuncuların ve kaynakların yerleştirilmesi



Savaşçıların yerleşeceği muhtemel alanların belirlenmesi



İlk turun sonunda savaşçıların yerleşimi

- Ana Ekran ve Menüler:
    - Oyunun başlangıcında kullanıcıları karşılayan ana ekran bulunmalıdır. Ana ekran, yeni oyun başlatma gibi seçenekleri içermelidir.
    - Menüler, kullanıcıların savaşçı seçimi yapmalarını ve oyunu kontrol etmelerini sağlamalıdır.
- Savaşçı Seçimi ve Yerleştirme:
    - Kullanıcılar, mevcut savaşçı türleri arasından seçim yapabilmelidir.
    - Seçilen savaşçılar, matris üzerinde belirli koordinatlara yerleştirilmelidir. Kullanıcılar, yerleştirme işlemini klavye aracılığıyla gerçekleştirebilirler.
- Oyun Alanı Görselleştirmesi:
    - Oyun alanı, kullanıcıların savaşçıları yerleştirebileceği bir matris olarak görselleştirilmelidir. Her hücre, boş veya bir savaşçıya ait olabilir ve kullanıcılar bu durumu görsel olarak görmelidir.
    - Savaşçılar, belirli semboller veya renklerle temsil edilerek kolayca tanınabilir hale getirilmelidir.
- Hata ve Uyarı Mesajları:
    - Kullanıcı hataları veya uygun olmayan işlemler durumunda uygun hata ve uyarı mesajları gösterilmelidir. Bu mesajlar, kullanıcıların neyin yanlış gittiğini anlamalarına yardımcı olmalıdır.
- Kullanıcı Deneyimi Düşünülerek Tasarım:
    - Kullanıcı arayüzü, kullanıcıların oyunu rahatlıkla oynayabilmeleri ve keyif alabilmeleri için kullanıcı deneyimi tasarımı prensiplerine uygun olmalıdır. Menülerin ve kontrollerin düzeni, kullanıcıların akıcı bir şekilde gezinmelerini sağlamalıdır.

4. UYGULAMA
4.1 Kodlanan Bileşenlerin Açıklamaları

import pygame: Pygame kütüphanesini projeye ekler.import sys: Python sistem fonksiyonlarını sağlar.import random: Rastgele sayılar oluşturmak için gereken kütüphaneyi içeri aktarır.from abc import ABC, abstractmethod: ABC (Abstract Base Class) ve abstractmethod, soyut sınıflar ve soyut yöntemler oluşturmak için kullanılır.get_world_size() fonksiyonu: Oyun dünyasının boyutunu kullanıcıdan alır. Kullanıcıya istenilen formatta giriş yapması için talimatlar verir ve doğru bir giriş alıncaya kadar döngü içinde kalır.draw_grid() fonksiyonu: Pygame penceresine oyun alanının ızgarasını çizer. Parametre olarak pencere yüzeyini (window), satır ve sütun sayısını (rows ve cols), pencere boyutunu (window_size), oyuncuları (players) ve kullanılabilir pozisyonları (available_positions) alır. Izgaranın her satırı ve sütunu için çizgileri çizer.

```python
import pygame
import sys
import random
from abc import ABC, abstractmethod

def get_world_size():
    while True:
        try:
            input_str = input("Oyun boyutunu girin (örneğin, 16x16 veya 8x8): ")
            rows, cols = map(int, input_str.split('x'))

            if 8 <= rows <= 32 and 8 <= cols <= 32:
                return rows, cols
            else:
                print("Geçersiz dünya boyutu. Lütfen 8 ile 32 arasında bir değer girin.")
        except ValueError:
            print("Geçersiz bir değer girdiniz. Lütfen iki sayıyı 'x' işareti ile ayırarak girin.")


def draw_grid(window, rows, cols, window_size, players, available_positions):
    square_size = window_size // rows
    line_color = (255, 255, 255)  # Çizgileri beyaz renkte çiz

    # Satır çizgilerini çiz
    for i in range(1, rows):
        pygame.draw.line(window, line_color, (0, i * square_size), (window_size, i * square_size))
```

Sütun çizgilerini çizme: Bu döngü, oyun alanının her sütununda bir çizgi çizer. Bu çizgiler oyun alanını sütunlara böler.Karelerin ortasına beyaz nokta ekleme: Bu döngü, her kare için orta noktasına bir beyaz nokta ekler. Ancak, eğer bir oyuncunun muhafızı ya da savaşçısı bu karedeyse, bu noktayı boş bırakır. Savaşçı sembollerini çizme: Bu döngü, her oyuncunun savaşçısının sembolünü çizer. Oyuncunun rengi ve savaşçının adı ile birlikte savaşçının sembolünü çizer. Bu sembollerin konumları oyuncu bilgilerinde saklanır ve draw_warrior() işlevi kullanılarak çizilir.

```python
# Sütun çizgilerini çiz
for i in range(1, cols):
    pygame.draw.line(window, line_color, (i * square_size, 0), (i * square_size, window_size))

# Karelerin ortasına beyaz nokta ekle
for i in range(rows):
    for j in range(cols):
        if any(player['guard_position'] == (i, j) for player in players.values()):
            continue  # Bu nokta bir muhafızın altında, boş bırak
        if any(player['warrior_position'] == (i, j) for player in players.values()):
            continue
        pygame.draw.circle(window, line_color,(i * square_size + square_size // 2, j
                                               * square_size + square_size // 2), 3)

# Savaşçı sembollerini çiz
for player_id, player_info in players.items():
    warrior_position = player_info['warrior_position']
    if warrior_position:
        warrior_name = player_info['warrior_name']
        draw_warrior(window, warrior_position, square_size, player_info['color'],
                     f"{warrior_name}{player_id}")
```

place_guards(players, rows, cols): Bu işlev, oyuncuların muhafızlarını oyun alanının köşelerine yerleştirir. Her bir oyuncunun muhafızı rastgele bir köşeye yerleştirilir. corners adlı bir liste oluşturulur ve her bir oyuncu için bir köşe seçilir. Seçilen köşe oyuncunun muhafızının yerleştirileceği köşe olarak atanır ve kullanılan köşe listeden çıkarılır.ask_for_fighter_choice(player_id): Bu işlev, belirli bir oyuncuya savaşçı seçimi yapma fırsatı verir. Kullanıcıya istenilen formatta bir seçenek girmesi için talimatlar verir ve doğru bir giriş alınana kadar döngü içinde kalır. Geçerli bir seçim yapıldığında, seçilen savaşçının numarası döndürülür.

```
def place_guards(players, rows, cols):
    corners = [(0, 0), (0, cols - 1), (rows - 1, 0), (rows - 1, cols - 1)]

    for player in players:
        random_corner = random.choice(corners)
        players[player]['guard_position'] = random_corner  # Köşeden rastgele bir muhafız yerleştir
        corners.remove(random_corner)  # Aynı köşeyi bir daha seçilmemesi için listeden çıkar


def ask_for_fighter_choice(player_id):
    while True:
        try:
            choice = int(
                input(f"{player_id}. oyuncu, savaşçı seçin (1: Muhafız, 2: Okçu, 3: Topçu, 4: Atlı, 5: Sağlıkçı): "))
            if 1 <= choice <= 5:
                return choice
            else:
                print("Geçersiz savaşçı seçimi. Lütfen 1 ile 5 arasında bir değer girin.")
        except ValueError:
            print("Geçersiz bir değer girdiniz. Lütfen bir sayı girin.")
```

ask_for_fighter_count(player_id): Bu işlev, belirli bir oyuncuya kaç savaşçı seçeceğini sorar. Kullanıcıya istenilen formatta bir sayı girmesi için talimatlar verir ve doğru bir giriş alınana kadar döngü içinde kalır. Geçerli bir sayı girildiğinde, seçilen savaşçı sayısı döndürülür.ask_for_next_round(): Bu işlev, bir sonraki turun geçilip geçilmeyeceğini sorar. Kullanıcıya "evet" veya "hayır" olarak yanıt vermesi için talimatlar verir ve doğru bir yanıt alınana kadar döngü içinde kalır. Eğer kullanıcı "evet" derse True değeri döndürülür ve bir sonraki tur başlatılır. "hayır" dendiğinde ise program sonlanır.

```
def ask_for_fighter_count(player_id):
    while True:
        try:
            count = int(input(f"{player_id}. oyuncu, kaç savaşçı seçeceksiniz? (0-2): "))
            if 0 <= count <= 2:
                return count
            elif count == 0:
                print()
                return count
            else:
                print("Geçersiz savaşçı sayısı. Lütfen 0 ile 2 arasında bir değer girin.")
        except ValueError:
            print("Geçersiz bir değer girdiniz. Lütfen bir sayı girin.")


def ask_for_next_round():
    while True:
        user_input = input("Bir sonraki tura geçilsin mi? (evet/hayır): ").lower()
        if user_input == 'evet':
            return True
        elif user_input == 'hayır':
            sys.exit()
        else:
            print("Geçersiz giriş. Lütfen 'evet' veya 'hayır' olarak yanıt verin.")
```

draw_warrior(screen, position, square_size, color, warrior_name): Belirtilen konuma savaşçının adını çizer. Kullanıcıya savaşçının adını ve rengini belirlemesi için bir ekran, konum, kare boyutu ve renk sağlanır. Savaşçının adı ve rengi belirlendikten sonra adı belirtilen konuma çizer. place_potential_guards (players, rows, cols): Bu işlev oyuncuların muhafızlarının potansiyel olarak yerleştirilebileceği konumları belirler. Her oyuncu için mevcut muhafız konumunu alır ve bu konumun etrafındaki 8 potansiyel konumu hesaplar. Bu potansiyel konumlar oyun alanının sınırları içinde olmalıdır. Belirlenen potansiyel konumlar bir liste olarak döndürülür.

```python
def draw_warrior(screen, position, square_size, color, warrior_name):
    font = pygame.font.Font(None, 36)
    text = font.render(warrior_name, True, color)
    text_rect = text.get_rect(center=(position[0] * square_size + square_size // 2,
    position[1] * square_size + square_size // 2))
    screen.blit(text, text_rect)


def place_potential_guards(players, rows, cols):
    potential_positions = []
    for player_id, player in players.items():
        guard_position = player['guard_position']
        for i in range(-1, 2):
            for j in range(-1, 2):
                if (i, j) != (0, 0):
                    new_pos = (guard_position[0] + i, guard_position[1] + j, player_id)
                    if 0 <= new_pos[0] < rows and 0 <= new_pos[1] < cols:
                        potential_positions.append(new_pos)
    return potential_positions
```

get_valid_placement_guard_positions(player_id, rows, cols, guard_position): Belirli bir oyuncunun muhafızı için geçerli yerleştirme konumlarını hesaplar. Her bir potansiyel konum, mevcut muhafızın konumu etrafındaki komşu konumlarıdır. Bu potansiyel konumlar, oyun alanının sınırları içinde olmalıdır. Belirlenen geçerli yerleştirme konumları bir liste olarak döndürülür. get_valid_ placement_warrior_positions(player_id, rows, cols, warrior_position): Belirli bir oyuncunun savaşçısı için geçerli yerleştirme konumlarını hesaplar. Her bir potansiyel konum, mevcut savaşçının konumu etrafındaki komşu konumlarıdır. Bu potansiyel konumlar, oyun alanının sınırları içinde olmalıdır. Belirlenen geçerli yerleştirme konumları bir liste olarak döndürülür.

```python
def get_valid_placement_guard_positions(player_id, rows, cols, guard_position):
    potential_positions = []
    for i in range(-1, 2):
        for j in range(-1, 2):
            if (i, j) != (0, 0):
                new_pos = (guard_position[0] + i, guard_position[1] + j)
                if 0 <= new_pos[0] < rows and 0 <= new_pos[1] < cols:
                    potential_positions.append(new_pos)
    return potential_positions


def get_valid_placement_warrior_positions(player_id, rows, cols, warrior_position):
    potential_warrior_positions = []
    for i in range(-1, 2):
        for j in range(-1, 2):
            if (i, j) != (0, 0):
                new_pos = (warrior_position[1] + i, warrior_position[0] + j)
                if 0 <= new_pos[0] < rows and 0 <= new_pos[1] < cols:
                    potential_warrior_positions.append(new_pos)
    return potential_warrior_positions
```

place_potential_warriors(players, rows, cols): Bu işlev, oyuncuların savaşçılarının potansiyel olarak yerleştirilebileceği konumları belirler. Her bir oyuncu için mevcut savaşçı konumunu alır ve bu konumun etrafındaki 8 potansiyel konumu hesaplar. Bu potansiyel konumlar, oyun alanının sınırları içinde olmalıdır. Belirlenen potansiyel konumlar bir liste olarak döndürülür.

```python
def place_potential_warriors(players, rows, cols):
    potential_warrior_positions = []
    for player_id, player in players.items():
        warrior_position = player['warrior_position']
        warrior_color = player['color']
        for i in range(-1, 2):
            for j in range(-1, 2):
                if (i, j) != (0, 0):
                    if warrior_position:
                        new_pos = (warrior_position[1] + i, warrior_position[0] + j, player_id)
                        if 0 <= new_pos[0] < rows and 0 <= new_pos[1] < cols:
                            potential_warrior_positions.append(new_pos)
    return potential_warrior_positions
```

update_and_draw_potential_positions(screen, square_size, players, rows, cols): Bu işlev, potansiyel savaşçı pozisyonlarını günceller ve ekrana çizer. Öncelikle, place_potential_warriors() işlevi kullanılarak potansiyel savaşçı pozisyonları alınır. Ardından, her bir oyuncunun potansiyel savaşçı pozisyonunu ekrana çizmek için potential_warrior_positions listesi üzerinde döngü yapılır. Son olarak, her bir oyuncunun mevcut muhafız pozisyonunu ve bu pozisyona ilişkin metni ekrana çizer.

```
def update_and_draw_potential_positions(screen, square_size, players, rows, cols):
    potential_warrior_positions = place_potential_warriors(players, rows, cols)

    # Potansiyel savaşçı pozisyonlarını ekrana çiz
    for player in players:
        for pos in potential_warrior_positions:
            player_id = pos[2]
            back_color = {1: (255, 170, 170), 2: (170, 212, 255), 3: (170, 255, 170), 4: (242, 229, 111)}
            pygame.draw.rect(screen, back_color[player_id],(pos[1] * square_size, pos[0] * square_size, square_size, square_size))

    for player in players:
        guard_position = players[player]['guard_position']
        square_size = 640 // rows
        pygame.draw.rect(screen, players[player]['back_color'],(guard_position[1] * square_size, guard_position[0] * square_size,
        square_size, square_size))
        font = pygame.font.Font(None, 36)
        text = font.render(f"M{player}", True, players[player]['color'])
        screen.blit(text, (guard_position[1] * square_size + square_size // 2 - text.get_width() // 2,guard_position[0] *
        square_size + square_size // 2 - text.get_height() // 2))
```

draw_resource_management(screen, players, rows): Bu işlev, oyuncuların kaynak yönetimini ekrana çizer. Her oyuncunun kaynak sayısını ve oyuncu numarasını görüntüler. Oyuncu numarası, oyuncu rengi ve kaynak sayısı ile birlikte görüntülenir. update_resources(players, total_warriors): Bu işlev, oyuncuların kaynaklarını günceller. Her bir oyuncunun başlangıçta 10 kaynağı vardır. Ardından, her oyuncunun sahip olduğu savaşçı sayısına bağlı olarak ek kaynaklar eklenir. Her savaşçı için bir kaynak eklenir. Bu, oyuncuların kaynaklarını güncellemek için kullanılır.

```
def draw_resource_management(screen, players, rows):
    square_size = 640 // rows
    for i, player in enumerate(players, start=1):
        font = pygame.font.Font(None, 22)
        text = font.render(f"{i}. Oyuncu Kaynak: {players[player]['resources']}", True, players[player]['color'])
        source_position = (rows * square_size + 10, i * 40)
        screen.blit(text, source_position)


def update_resources(players, total_warriors):
    for player_id, player_info in players.items():  # Tur başlarında her oyuncunun kaynağına 10 kaynak ekle
        player_info['resources'] += 10

    for player_id, warrior_count in total_warriors.items():  # Her savaşçı için ek olarak 1 kaynak ekle
        players[player_id]['resources'] += 1
        players[player_id]['resources'] += warrior_count
```

count_warriors(warriors_matrix): Oyun alanındaki savaşçı sayısını hesaplar. Verilen savaşçı matrisi üzerinde döngü yapar ve her bir savaşçı için oyuncu kimliğini alır. Her oyuncunun sahip olduğu savaşçı sayısını toplar ve bir sözlük olarak döndürür. place_warrior_manually(player_id, players, warriors_matrix, available_positions): Bir oyuncunun savaşçısını manuel olarak yerleştirmesini sağlar. Oyuncudan bir konum girmesini ister ve belirtilen koordinatlara savaşçı yerleştirmek için uygunluğunu kontrol eder. Oyuncudan savaşçı seçimini (muhafız, okçu, topçu, atlı veya sağlıkçı) alır ve savaşçıyı yerleştirir. Yerleştirilen savaşçının bilgilerini günceller ve oyun alanı matrisini de günceller.

```
def count_warriors(warriors_matrix):
    total_warriors = {}
    for row in warriors_matrix:
        for warrior in row:
            if warrior:
                player_id = warrior['player_id']
                total_warriors[player_id] = total_warriors.get(player_id, 0) + 1
    return total_warriors


def place_warrior_manually(player_id, players, warriors_matrix, available_positions):
    placed = False
    while not placed:
        try:
            y, x = map(int, input(f"{player_id}. oyuncu, savaşçıyı yerleştirmek istediğiniz konumu seçin (x y): ").split())
            x -= 1
            y -= 1
```

```
if (x, y) in available_positions and warriors_matrix[x][y] is None:
    warrior_choice = ask_for_fighter_choice(player_id)
    if warrior_choice == 1:
        warrior_name = "M"
        player_info = players[player_id]
        if player_info['resources'] >= 10:
            players[player_id]['resources'] -= 10
        else:
            print("Yeterli kaynağınız yok. Muhafız yerleştiremezsiniz.")
            continue
    elif warrior_choice == 2:
        warrior_name = "O"
        player_info = players[player_id]
        if player_info['resources'] >= 20:
            players[player_id]['resources'] -= 20
        else:
            print("Yeterli kaynağınız yok. Okçu yerleştiremezsiniz.")
            continue
    elif warrior_choice == 3:
        warrior_name = "T"
        player_info = players[player_id]
        if player_info['resources'] >= 50:
            players[player_id]['resources'] -= 50
        else:
            print("Yeterli kaynağınız yok. Topçu yerleştiremezsiniz.")
            continue
```

```
        elif warrior_choice == 4:
            warrior_name = "A"
            player_info = players[player_id]
            if player_info['resources'] >= 30:
                players[player_id]['resources'] -= 30
            else:
                print("Yeterli kaynağınız yok. Atlı yerleştiremezsiniz.")
                continue
        else:
            warrior_name = "S"
            player_info = players[player_id]
            if player_info['resources'] >= 10:
                players[player_id]['resources'] -= 10
            else:
                print("Yeterli kaynağınız yok. Sağlıkçı yerleştiremezsiniz.")
                continue
        players[player_id]['warrior_name'] = warrior_name
        players[player_id]['warrior_position'] = (x, y)
        warriors_matrix[x][y] = {'player_id': player_id,'player_name': players[player_id]['name'],'warrior_name':
        warrior_name,'color': players[player_id]['color']}
        available_positions.pop((x, y))
        placed = True
    else:
        print(
            "Bu konum geçersiz veya başka bir savaşçı tarafından işgal edilmiş. Lütfen başka bir konum seçin.")
except ValueError:
    print("Geçersiz giriş. Lütfen x ve y koordinatlarını boşlukla ayırarak girin.")
```

ai_place_warriors(player_id, players, rows, cols, valid_positions, warriors_matrix): Bu işlev, yapay zeka tarafından savaşçıların yerleştirilmesini yönetir. İlk olarak, muhtemel savaşçı pozisyonları belirlenir. Ardından, mevcut savaşçı pozisyonları güncel savaşçı pozisyonlarından filtrelenir. Son olarak, uygun pozisyonlardan biri seçilir ve savaşçı (Topçu) bu konuma yerleştirilir. Yerleştirme başarılıysa True değeri döndürülür, aksi halde False döndürülür.ask_for_ai_decision(player_id, players): Bu işlev, belirli bir oyuncunun yapay zeka kullanıp kullanmayacağına karar vermesini sağlar. Eğer oyuncu yapay zeka kullanmayı seçerse ve yeterli kaynağa sahipse, kaynaklarından 50'yi harcar ve True değeri döndürür. Aksi halde False değeri döndürür.

```
def ai_place_warriors(player_id, players, rows, cols, valid_positions, warriors_matrix):
    # Muhtemel savaşçı pozisyonları
    potential_warrior_positions = []
    for row in range(rows):
        for col in range(cols):
            if (row, col) in valid_positions:
                potential_warrior_positions.append((row, col))

    # Savaşçıyı yerleştirilecek uygun pozisyonlar listesinden geçerli savaşçı pozisyonları
    current_warrior_positions = [player['warrior_position'] for player in players.values()]

    # Uygun pozisyonları güncel savaşçı pozisyonlarından filtreleme
    available_positions = [pos for pos in potential_warrior_positions if pos not in current_warrior_positions]

    # Rastgele bir savaşçı pozisyonu seç
    if available_positions:
        for pos in available_positions:
            x, y = pos
            adjacent_cells = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
            for adj_pos in adjacent_cells:
                adj_x, adj_y = adj_pos
                if 0 <= adj_x < rows and 0 <= adj_y < cols:
                    adj_warrior = warriors_matrix[adj_x][adj_y]
                    if adj_warrior and adj_warrior['player_id'] == player_id:
                        players[player_id]['warrior_position'] = (x, y)
                        players[player_id]['warrior_name'] = ("T")
                        warriors_matrix[x][y] = {'player_id': player_id, 'player_name': players[player_id]['name'],
                        'warrior_name': players[player_id]['warrior_name'], 'color': players[player_id]['color']}
                        return True
```

```
        random_position = random.choice(available_positions)
        y, x = random_position
        players[player_id]['warrior_name'] = "T"
        players[player_id]['warrior_position'] = (x, y)
        warriors_matrix[x][y] = {'player_id': player_id,
        'player_name': players[player_id]['name'],
        'warrior_name': "T", 'color': players[player_id]['color']}
        return True
    else:
        return False


def ask_for_ai_decision(player_id,players):
    decision = input(f"{player_id}. oyuncu, yapay zeka kullanılsın mı? (E/H): ")
    player_info = players[player_id]
    if decision == "E":
        if player_info['resources'] >= 50:
            players[player_id]['resources'] -= 50
        else:
            print("Yeterli kaynağınız yok. Sağlıkçı yerleştiremezsiniz.")
    return decision.upper() == "E"
```

Soyut bir savaşçı sınıfı olan Savasci sınıfını tanımlanır.__init__(self, player_id, color, guard_position): Bu yöntem, Savasci sınıfının örneklerinin oluşturulması sırasında çağrılır. Bu yöntemde savaşçının oyuncu kimliği (player_id), rengi (color) ve muhafız konumu (guard_position) gibi temel özellikleri tanımlanır. Ayrıca, savaşçının diğer özellikleri için yer tutucu değerler atanır.@abstractmethod draw(self, screen, square_size): Bu yöntem, savaşçının ekrana çizilmesi için soyut bir yöntemdir. attack(self, enemies): Bu yöntem, savaşçının saldırı yapmasını sağlar. Alt sınıflar bu yöntemi uygulayabilir ve kendi saldırı mantıklarını tanımlayabilirler.

```python
class Savasci(ABC):
    def __init__(self, player_id, color, guard_position):
        self.player_id = player_id
        self.color = color
        self.guard_name = None
        self.guard_position = guard_position
        self.resource_cost = None
        self.health = None
        self.target_enemies = 3
        self.damage = None
        self.range_horizontal = None
        self.range_vertical = None
        self.range_diagonal = None

    @abstractmethod
    def draw(self, screen, square_size):
        pass

    def attack(self, enemies):
        pass
```

Muhafiz sınıfı, Savasci sınıfından türeterek oluşturur.__init__(self, player_id, color, guard_position): Bu metot, Muhafiz sınıfının örneklerinin oluşturulması sırasında çağrılır. Savasci sınıfının __init__() metodunu çağırarak temel özellikleri (oyuncu kimliği, renk ve muhafız pozisyonu) tanımlar. Ayrıca, muhafızın adı, kaynak maliyeti, sağlık, hasar, menzil ve diğer özellikler atanır.draw(self, screen, square_size, position): Bu metot, muhafızın ekrana çizilmesinden sorumludur. Verilen ekrana (screen), kare boyutuna (square_size) ve pozisyona göre muhafızı çizer.attack(self, enemies, players): Bu metot, muhafızın saldırısını yönetir. Verilen düşmanlar listesi (enemies) ve oyuncular sözlüğü (players) üzerinde döngü yaparak saldırı gerçekleştirir. Muhafızın saldırı menziline giren düşmanların sağlık değerlerini azaltır ve gerekirse düşmanı oyundan çıkarır.

```python
class Muhafiz(Savasci):
    def __init__(self, player_id, color, guard_position):
        super().__init__(player_id, color, guard_position)
        self.guard_name = "Muhafız"
        self.guard_position_symbol = "M"
        self.resource_cost = 10
        self.health = 80
        self.damage = 20
        self.range_horizontal = 1
        self.range_vertical = 1
        self.range_diagonal = 1

    def draw(self, screen, square_size, position):
        pygame.draw.rect(screen, self.color,
                         (position[0] * square_size, position[1] * square_size, square_size, square_size))
        draw_warrior(screen, position, square_size, self.color, self.guard_position_symbol)

    def attack(self, enemies, players):
        for enemy in enemies:
            enemy_x, enemy_y, enemy_player_id = enemy
            guard_x, guard_y = self.position

            if (abs(enemy_x - guard_x) <= self.range_horizontal and
                    abs(enemy_y - guard_y) <= self.range_vertical):
                players[enemy_player_id]['health'] += self.damage
                if players[enemy_player_id]['health'] <= 0:
                    del players[enemy_player_id]
```

Okcu sınıfı, Savasci sınıfından türeterek oluşturur.__init__(self, player_id, color, guard_position): Bu metot, Okcu sınıfının örneklerinin oluşturulması sırasında çağrılır. Savasci sınıfının __init__() metodunu çağırarak temel özellikleri (oyuncu kimliği, renk ve muhafız pozisyonu) tanımlar. Ayrıca, okçunun adı, kaynak maliyeti, sağlık, hasar, menzil ve diğer özellikler atanır.draw(self, screen, square_size, position): Bu metot, okçunun ekrana çizilmesinden sorumludur. Verilen ekrana (screen), kare boyutuna (square_size) ve pozisyona göre okçuyu çizer.attack(self, enemies, players): Bu metot, okçunun saldırısını yönetir. Verilen düşmanlar listesi (enemies) ve oyuncular sözlüğü (players) üzerinde döngü yaparak saldırı gerçekleştirir. Okçu, her bir saldırıda en sağlıklı düşmana saldırır ve saldırı menziline giren düşmanların sağlık değerlerini azaltır. Saldırıları üç kez tekrarlar.

```python
class Okcu(Savasci):
    def __init__(self, player_id, color, guard_position):
        super().__init__(player_id, color, guard_position)
        self.guard_name = "Okçu"
        self.guard_position_symbol = "O"
        self.resource_cost = 20
        self.health = 30
        self.damage = 0.6
        self.range_horizontal = 2
        self.range_vertical = 2
        self.range_diagonal = 2

    def draw(self, screen, square_size, position):
        pygame.draw.rect(screen, self.color,(position[0] * square_size, position[1] * square_size, square_size, square_size))
        draw_warrior(screen, position, square_size, self.color, self.guard_position_symbol)

    def attack(self, enemies, players):
        for _ in range(3):
            max_health_enemy = max(enemies, key=lambda x: players[x[2]]['health'], default=None)
            if max_health_enemy:
                enemy_x, enemy_y, enemy_player_id = max_health_enemy
                archer_x, archer_y = self.position

                if (abs(enemy_x - archer_x) <= self.range_horizontal and abs(enemy_y - archer_y) <= self.range_vertical):
                    players[enemy_player_id]['health'] *= (1 + self.damage_percentage)
                    if players[enemy_player_id]['health'] <= 0:
                        del players[enemy_player_id]
                    enemies.remove(max_health_enemy)
                else:
```

Topcu sınıfı, Savasci sınıfından türeterek oluşturur.__init__(self, player_id, color, guard_position): Bu metot, Topcu sınıfının örneklerinin oluşturulması sırasında çağrılır. Savasci sınıfının __init__() metodunu çağırarak temel özellikleri (oyuncu kimliği, renk ve muhafız pozisyonu) tanımlar. Ayrıca, topçu savaşçısının adı, kaynak maliyeti, sağlık, hasar, menzil ve diğer özellikler atanır.draw(self, screen, square_size, position): Bu metot, topçu savaşçısının ekrana çizilmesinden sorumludur. Verilen ekrana (screen), kare boyutuna (square_size) ve pozisyona göre topçu savaşçısını çizer.attack(self, enemies, players): Bu metot, topçu savaşçısının saldırısını yönetir. Verilen düşmanlar listesi (enemies) ve oyuncular sözlüğü (players) üzerinde döngü yaparak saldırı gerçekleştirir. Topçu savaşçısı, en sağlıklı düşmana saldırır ve saldırı menziline giren düşmanların sağlık değerlerini büyük miktarda azaltır.

```python
class Topcu(Savasci):
    def __init__(self, player_id, color, guard_position):
        super().__init__(player_id, color, guard_position)
        self.guard_name = "Topçu"
        self.guard_position_symbol = "T"
        self.resource_cost = 50
        self.health = 30
        self.damage = 100
        self.range_horizontal = 2
        self.range_vertical = 2
        self.range_diagonal = 0

    def draw(self, screen, square_size, position):
        pygame.draw.rect(screen, self.color,(position[0] * square_size, position[1] * square_size, square_size, square_size))
        draw_warrior(screen, position, square_size, self.color, self.guard_position_symbol)

    def attack(self, enemies, players):
        max_health_enemy = max(enemies, key=lambda x: players[x[2]]['health'], default=None)
        if max_health_enemy:
            enemy_x, enemy_y, enemy_player_id = max_health_enemy
            artillery_x, artillery_y = self.position

            if (abs(enemy_x - artillery_x) <= self.range_horizontal and abs(enemy_y - artillery_y) <= self.range_vertical):
                players[enemy_player_id]['health'] *= (1 + self.damage_percentage)
                if players[enemy_player_id]['health'] <= 0:
                    del players[enemy_player_id]
                enemies.remove(max_health_enemy)
```

Atli sınıfı, Savasci sınıfından türeterek oluşturur.__init__(self, player_id, color, guard_position): Bu metot, Atli sınıfının örneklerinin oluşturulması sırasında çağrılır. Savasci sınıfının __init__() metodunu çağırarak temel özellikleri (oyuncu kimliği, renk ve muhafız pozisyonu) tanımlar. Ayrıca, atlı savaşçısının adı, kaynak maliyeti, sağlık, hasar, menzil ve diğer özellikler atanır.draw(self, screen, square_size, position): Bu metot, atlı savaşçısının ekrana çizilmesinden sorumludur. Verilen ekrana (screen), kare boyutuna (square_size) ve pozisyona göre atlı savaşçısını çizer.attack(self, enemies, players): Bu metot, atlı savaşçısının saldırısını yönetir. Verilen düşmanlar listesi (enemies) ve oyuncular sözlüğü (players) üzerinde döngü yaparak saldırı gerçekleştirir. Atlı savaşçısı, en yüksek kaynağa sahip iki düşmanı hedef alır ve saldırı menziline giren düşmanların sağlık değerlerini arttırır.

```
class Atli(Savasci):
    def __init__(self, player_id, color, guard_position):
        super().__init__(player_id, color, guard_position)
        self.guard_name = "Atlı"
        self.guard_position_symbol = "A"
        self.resource_cost = 30
        self.health = 40
        self.damage = 30
        self.range_horizontal = 0
        self.range_vertical = 0
        self.range_diagonal = 3

    def draw(self, screen, square_size, position):
        pygame.draw.rect(screen, self.color,(position[0] * square_size, position[1] * square_size, square_size, square_size))
        draw_warrior(screen, position, square_size, self.color, self.guard_position_symbol)

    def attack(self, enemies, players):
        if len(enemies) >= 2:
            # Düşmanları kaynaklarına göre sırala
            enemies.sort(key=lambda x: players[x[2]]['resource'], reverse=True)

            # İlk iki düşmanı hedef al
            for enemy in enemies[:2]:
                enemy_x, enemy_y, enemy_player_id = enemy
                horseman_x, horseman_y = self.position
```

Saglikci sınıfı,Savasci sınıfından türeterek oluşturur.__init__(self, player_id, color, guard_position): Bu metot, Saglikci sınıfının örneklerinin oluşturulması sırasında çağrılır. Savasci sınıfının __init__() metodunu çağırarak temel özellikleri (oyuncu kimliği, renk ve muhafız pozisyonu) tanımlar. Ayrıca, sağlıkçı savaşçısının adı, kaynak maliyeti, sağlık, hasar, menzil ve diğer özellikler atanır.draw(self, screen, square_size, position): Bu metot, sağlıkçı savaşçısının ekrana çizilmesinden sorumludur. Verilen ekrana (screen), kare boyutuna (square_size) ve pozisyona göre sağlıkçı savaşçısını çizer.attack(self, allies, players): Bu metot, sağlıkçı savaşçısının saldırısını yönetir. Verilen müttefikler listesi (allies) ve oyuncular sözlüğü (players) üzerinde döngü yaparak saldırı gerçekleştirir. Sağlıkçı savaşçısı, en az sağlığa sahip müttefik birliğine can ekler.

```
                # Çapraz menzilde mi kontrol et
                if (abs(enemy_x - horseman_x) <= self.range_diagonal and abs(enemy_y - horseman_y) <= self.range_diagonal and
                    (enemy_x != horseman_x or enemy_y != horseman_y)):
                    players[enemy_player_id]['health'] += self.damage
                    if players[enemy_player_id]['health'] <= 0:
                        del players[enemy_player_id]
                    enemies.remove(enemy)


class Saglikci(Savasci):
    def __init__(self, player_id, color, guard_position):
        super().__init__(player_id, color, guard_position)
        self.guard_name = "Sağlıkçı"
        self.guard_position_symbol = "S"
        self.resource_cost = 10
        self.health = 100
        self.damage = -50  # Can eklemek için negatif hasar kullanıyoruz
        self.range_horizontal = 0
        self.range_vertical = 2
        self.range_diagonal = 2

    def draw(self, screen, square_size, position):
        pygame.draw.rect(screen, self.color,(position[0] * square_size, position[1] * square_size, square_size, square_size))
        draw_warrior(screen, position, square_size, self.color, self.guard_position_symbol)
```

pygame.init(): Pygame kütüphanesini başlatır.width_height = (850, 640): Oyun ekranının genişliği ve yüksekliğini belirten bir demet oluşturulur.screen = pygame.display. set_mode ((width_height)): Belirlenen genişlik ve yükseklikte bir oyun ekranı oluşturulur.pygame. display.set_caption("LORDS OF THE POLYWARPHISM"): Oyun ekranının başlığını ayarlar.

```
    def attack(self, allies, players):
        if allies:
            # En az cana sahip olan dost birliği bul
            min_health_ally = min(allies, key=lambda x: players[x[2]]['health'])
            ally_x, ally_y, ally_player_id = min_health_ally
            healer_x, healer_y = self.position

            # Menzilde mi kontrol et
            if (abs(ally_x - healer_x) <= self.range_horizontal and abs(ally_y - healer_y) <= self.range_vertical and
                (ally_x != healer_x or ally_y != healer_y)):
                players[ally_player_id]['health'] += self.damage


# Oyun ekranını oluştur
pygame.init()
width_height = (850, 640)
screen = pygame.display.set_mode((width_height))
pygame.display.set_caption("LORDS OF THE POLYWARPHISM")
```

pygame.init(): Pygame kütüphanesini başlatır.first_round = True: İlk turun oynanıp oynanmadığını takip etmek için bir bayrak oluşturulur.while True:: Sonsuz bir döngü başlatılır. Bu döngü, oyunun ana döngüsünü oluşturur. Oyun, oyuncular belirlenene kadar ve dünya boyutu alınıncaya kadar devam eder.İç içe bir while döngüsü, kullanıcıdan geçerli bir oyuncu sayısı girmesini bekler (num_players). Kullanıcı, 1 ile 4 arasında bir sayı girene kadar döngü devam eder. Girilen sayı, num_players değişkenine atanır.get_world_size() fonksiyonu, oyun dünyasının boyutunu alır (satır ve sütun sayıları). Bu fonksiyon, geçerli bir dünya boyutu alana kadar kullanıcıya sorular sorar ve boyutları döndürür.

```
def main():
    pygame.init()
    first_round = True  # İlk turu izlemek için bir bayrak

    while True:
        while True:
            try:
                num_players = int(input("Oyuncu sayısını girin (1-4): "))
                if 1 <= num_players <= 4:
                    break
                else:
                    print("Geçersiz oyuncu sayısı. Lütfen 1 ile 4 arasında bir değer girin.")
            except ValueError:
                print("Geçersiz bir değer girdiniz. Lütfen bir sayı girin.")

        # Dünya boyutunu al
        rows, cols = get_world_size()
```

players = {}: Oyuncuları depolamak için boş bir sözlük oluşturulur.Bir for döngüsü kullanarak, her bir oyuncunun renklerini (player_color ve player_second), adını ('Oyuncu 1', 'Oyuncu 2' gibi), muhafız pozisyonunu ('guard_name' ve 'guard_position'), kaynaklarını ('resources') ve diğer özelliklerini belirler. Her oyuncu için bir sözlük oluşturulur ve players sözlüğüne eklenir.place_guards(players, rows, cols): Oyuncu muhafızlarını yerleştirir. Bu fonksiyon, her oyuncunun muhafızını rastgele bir köşeye yerleştirir.available_positions: Boş konumları belirlemek için bir sözlük oluşturur. Bu sözlük, (satır, sütun) koordinat çiftleriyle doldurulur ve başlangıçta hepsi boştur.warriors_matrix: Oyuncu savaşçılarını saklamak için bir matris oluşturur. Bu matris, oyun alanının her bir karesini temsil eder ve başlangıçta hepsi boştur.potential_positions: Her bir karenin muhtemel bir yerleşim alanı olup olmadığını gösteren bir iki boyutlu bir liste oluşturur. Bu liste, her kare için bir Boolean değeri içerir ve başlangıçta tüm değerler False olarak ayarlanır.

```
# Oyuncuları oluştur
players = {}
for i in range(1, num_players + 1):
    background_color = {1: (255, 170, 170), 2: (170, 212, 255), 3: (170, 255, 170), 4: (242, 229, 111)}
    player_color = background_color[i]

    font_color = {1: (127, 0, 0), 2: (0, 0, 191), 3: (0, 127, 0), 4: (249, 187, 0)}
    player_second = font_color[i]

    players[i] = {'id': i, 'color': player_second, 'back_color': player_color, 'name': f"Oyuncu {i}", 'warrior_position':
    None, 'guard_name': f"M{i}", 'guard_position': None, 'resources': 201}

place_guards(players, rows, cols)  # Oyuncu muhafızlarını yerleştir
available_positions = {(x, y): None for x in range(rows) for y in range(cols)}  # Boş konumları belirlemek için bir liste
oluştur
warriors_matrix = [[None for _ in range(cols)] for _ in range(rows)]  # Oyuncu savaşçılarını saklamak için bir matris
oluştur
potential_positions = [[False for _ in range(cols)] for _ in range(rows)]  # Her bir karenin muhtemel bir yerleşim alanı
olup olmadığını gösteren bir iki boyutlu liste oluştur
```

İç içe geçmiş bir while döngüsü, oyunun çalışmasını sağlar. İçteki döngü, kullanıcının oyun penceresini kapatarak çıkış yapmasına kadar devam eder (running bayrağı False olana kadar).for event in pygame.event.get(): döngüsü, pygame olaylarını yakalar. Bu döngü, kullanıcının pencereyi kapamasını (pygame.QUIT olayı) kontrol eder ve running bayrağını False olarak ayarlar.screen.fill((0, 0, 0)): Ekranı siyah renkle temizler.draw_grid(screen, rows, cols, 640, players, available_positions.keys()): Oyun ızgarasını çizer. draw_grid fonksiyonu, oyun alanının satır ve sütun sayılarına göre bir ızgara çizer. Ayrıca, oyuncu muhafızlarını ve boş konumları çizmek için players ve available_positions.keys() argümanlarını kullanır.Oyuncu muhafızlarını çizmek için bir döngü oluşturulur. Her oyuncu için muhafız pozisyonu alınır ve bu pozisyonda bir metin (muhafız sembolü) çizilir. Çizilen metin, muhafızın hangi oyuncuya ait olduğunu belirtir.

```
# Oyun döngüsü
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((0, 0, 0))  # Ekranı temizle
    draw_grid(screen, rows, cols, 640, players, available_positions.keys())  # Izgara çizimini yap

    # Oyuncu muhafızlarını çiz
    for player in players:
        guard_position = players[player]['guard_position']
        square_size = 640 // rows
        font = pygame.font.Font(None, 36)
        text = font.render(f"M{player}", True, players[player]['color'])
        screen.blit(text, (guard_position[1] * square_size + square_size // 2 - text.get_width() // 2, guard_position[0]
        * square_size + square_size // 2 - text.get_height() // 2))
```

İlk tur, first_round bayrağı ile kontrol edilir. Eğer first_round True ise, bu oyunun henüz ilk turunda olduğumuzu belirtir.İlk turda, oyunculara bir sonraki tura geçip geçmeme seçeneği sunulur. Bu, ask_for_next_round() fonksiyonu aracılığıyla gerçekleştirilir.Eğer kullanıcı bir sonraki tura geçmek isterse, first_round bayrağı False olarak ayarlanır, böylece bir sonraki turlarda bu koşul sağlanmaz.İlk turda oyuncu muhafızları çizilir ve her muhafızın üzerinde oyuncunun adı belirtilir. Bu, oyun alanının üzerindeki oyuncuların muhafızlarının belirlenmesine yardımcı olur.

```
# İlk turda ise, kullanıcıya bir sonraki tura geçilsin mi sor
if first_round:
    draw_resource_management(screen, players, rows)
    pygame.display.flip()
    if ask_for_next_round():
        first_round = False  # İlk tur bitti
        for player in players:
            guard_position = players[player]['guard_position']
            square_size = 640 // rows
            pygame.draw.rect(screen, players[player]['back_color'], (
            guard_position[1] * square_size, guard_position[0] * square_size, square_size, square_size))
            font = pygame.font.Font(None, 36)
            text = font.render(f"M{player}", True, players[player]['color'])
            screen.blit(text, (guard_position[1] * square_size + square_size // 2 - text.get_width() // 2,
                               guard_position[0] * square_size + square_size // 2 - text.get_height() // 2))
```

İlk turdan sonra, oyuncuların muhafızlarının olduğu karelerin arka planını boyar.Potansiyel yeni muhafız konumlarını belirler ve uygun arka plan renkleriyle boyar.Her oyuncunun muhafızının bulunduğu karelerin arka plan rengini değiştirir ve muhafızın üzerine oyuncunun adını yazar.Bu adımlar, ikinci turun tüm oyuncuları için gerçekleştirilir.

```
        # Potansiyel muhafız konumlarını al ve uygun arka plan rengiyle boyayın
        potential_guard_positions = place_potential_guards(players, rows, cols)
        for pos in potential_guard_positions:
            player_id = pos[2]
            back_color = {1: (255, 170, 170), 2: (170, 212, 255), 3: (170, 255, 170),4: (242, 229, 111)}
            pygame.draw.rect(screen, back_color[player_id], (pos[1] * square_size, pos[0] * square_size, square_size,
            square_size))

        draw_grid(screen, rows, cols, 640, players, available_positions.keys())
        pygame.display.flip()
    else:
        pygame.quit()
        sys.exit()
else:
    # İkinci turda, muhafızların olduğu karelerin arkaplanını boyadıktan sonra potansiyel yeni muhafız konumlarını
    belirleyin ve uygun arka plan renkleriyle boyayın
    for player in players:
        guard_position = players[player]['guard_position']
        square_size = 640 // rows
        pygame.draw.rect(screen, players[player]['back_color'], (
        guard_position[1] * square_size, guard_position[0] * square_size, square_size,
        square_size))  # Muhtemel muhafız yerleştirildiğinde bu arka planı düzeltilcek
        font = pygame.font.Font(None, 36)
        text = font.render(f"M{player}", True, players[player]['color'])  # M'lerin rengi burdan güncellenecek
        screen.blit(text, (guard_position[1] * square_size + square_size // 2 - text.get_width() // 2,
                           guard_position[0] * square_size + square_size // 2 - text.get_height() // 2))
```

place_potential_warriors fonksiyonu çağrılarak her oyuncu için potansiyel savaşçı pozisyonları alınır.Potansiyel muhafız ve savaşçı pozisyonları uygun arka plan rengiyle boyanır.Potansiyel yerleşim alanları, yani savaşçının etrafındaki kareler, çizilir.Her oyuncunun savaşçısının bulunduğu karenin arka planı boyanır.

```
# Potansiyel muhafız konumlarını al ve uygun arka plan rengiyle boyayın.
potential_warrior_positions = place_potential_warriors(players, rows, cols)
for pos in potential_guard_positions:
    player_id = pos[2]
    back_color = {1: (255, 170, 170), 2: (170, 212, 255), 3: (170, 255, 170), 4: (242, 229, 111)}
    pygame.draw.rect(screen, back_color[player_id], (pos[1] * square_size, pos[0] * square_size, square_size,
    square_size))

# Muhtemel yerleşim alanlarını çiz.Yani savaşçının çevresini çiziyor.
for x in range(rows):
    for y in range(cols):
        player_id = pos[2]
        back_color = {1: (255, 170, 170), 2: (170, 212, 255), 3: (170, 255, 170), 4: (242, 229, 111)}
        if potential_positions[x][y]:
            pygame.draw.rect(screen, ((back_color[player_id])), (
            y * square_size, x * square_size, square_size, square_size))  # 255,255,255

# Savaşçının bulunduğu karenin arka planını boyayın
for player in players:
    warrior_position = players[player]['warrior_position']  # Savaşçının pozisyonunu al
    square_size = 640 // rows
    pygame.draw.rect(screen, players[player]['back_color'], (warrior_position[0] * square_size, warrior_position
    [1] * square_size, square_size, square_size))
```

place_potential_warriors fonksiyonu kullanılarak her oyuncu için potansiyel savaşçı pozisyonları belirlenir.Bu pozisyonlar, uygun arka plan rengiyle ekrana çizilir.Her savaşçının üzerinde oyuncu kimliği ve savaşçı tipi bilgileriyle birlikte gösterilir.Ardından her oyuncunun muhafızının pozisyonu alınarak ekrana çizilir.

```
# Potansiyel savaşçı pozisyonlarını al ve ekrana çiz
potential_warrior_positions = place_potential_warriors(players, rows, cols)
for pos in potential_warrior_positions:
    player_id = pos[2]
    back_color = {1: (255, 170, 170), 2: (170, 212, 255), 3: (170, 255, 170), 4: (242, 229, 111)}
    pygame.draw.rect(screen, back_color[player_id],(pos[1] * square_size, pos[0] * square_size, square_size,
    square_size))
    warrior_name = players[player_id]['warrior_name']
    draw_warrior(screen, (pos[1], pos[0]), square_size, players[player_id]['color'],f"{warrior_name}{player_id}")

for player in players:
    guard_position = players[player]['guard_position']
    square_size = 640 // rows
    pygame.draw.rect(screen, players[player]['back_color'], (
    guard_position[1] * square_size, guard_position[0] * square_size, square_size, square_size))
    font = pygame.font.Font(None, 36)
    text = font.render(f"M{player}", True, players[player]['color'])
    screen.blit(text, (guard_position[1] * square_size + square_size // 2 - text.get_width() // 2,
                       guard_position[0] * square_size + square_size // 2 - text.get_height() // 2))
```

Oyuncuların sahip olduğu savaşçı pozisyonları alınır ve bu pozisyonlar ekrana çizilir. Ardından savaşçı pozisyonları matrise yerleştirilir.Son olarak potansiyel pozisyonlar güncellenir ve ekrana çizilir.

```
# Oyuncuların savaşçılarını ekrana çiz
for x in range(rows):
    for y in range(cols):
        if warriors_matrix[x][y]:
            player_id = warriors_matrix[x][y]['player_id']
            color = warriors_matrix[x][y]['color']
            warrior_name = warriors_matrix[x][y]['warrior_name']
            pygame.draw.rect(screen, players[player_id]['back_color'],(x * square_size, y * square_size, square_size,
            square_size))
            draw_warrior(screen, (x, y), 640 // rows, color, f"{warrior_name}{player_id}")

# Savaşçıları matrise yerleştir
for player_id, player_info in players.items():
    warrior_position = player_info['warrior_position']
    if warrior_position:
        x, y = warrior_position
        warriors_matrix[x][y] = {'player_id': player_id, 'player_name': player_info['name'], 'warrior_name':
        player_info['warrior_name'], 'color': player_info['color']}

# Potansiyel pozisyonları güncelle ve ekrana çiz
update_and_draw_potential_positions(screen, square_size, players, rows, cols)
```

Savaşçıların matristeki pozisyonları taranır ve ekrana çizilir.Potansiyel pozisyonlar güncellenir ve ekrana çizilir.Kaynak yönetimi alanı belirlenir ve ekranda siyah bir dikdörtgenle temizlenir.Savaşçı sayısı her tur başında hesaplanır ve kaynaklar güncellenir. Kaynak yönetimi ekranına kaynak bilgileri çizilir.Izgara çizimi yapılır ve ekran güncellenir.

```
# Savaşçıları ekrana çiz
for x in range(rows):
    for y in range(cols):
        if warriors_matrix[x][y]:
            warrior_info = warriors_matrix[x][y]
            draw_warrior(screen, (x, y), square_size, warrior_info['color'], f"{warrior_info['warrior_name']}
            {warrior_info['player_id']}")

update_and_draw_potential_positions(screen, square_size, players, rows, cols)  # Potansiyel pozisyonları güncelle ve
ekrana çiz

resource_management_area = (
rows * square_size, 0, 210, rows * square_size)  # Kaynak yönetimi alanını belirle
pygame.draw.rect(screen, (0, 0, 0),
                 resource_management_area)  # Kaynak bilgilerinin üst üste gelmemesi için siyaha boya

total_warriors = count_warriors(warriors_matrix)  # Her tur başında savaşçı sayısını hesapla
update_resources(players, total_warriors)  # Her tur başında kaynakları güncelle
draw_resource_management(screen, players, rows)

draw_grid(screen, rows, cols, 640, players, available_positions.keys())
pygame.display.flip()
```

Kullanıcıya bir sonraki tura geçilsin mi sorulur.Eğer bir sonraki tura geçilecekse, her oyuncu için savaşçı sayısı belirlenir ve potansiyel savaşçı pozisyonları güncellenir.Eğer bir sonraki tur yapay zeka tarafından yönetilecekse, yapay zeka kararı alınır ve uygun savaşçı pozisyonları belirlenir.Eğer bir sonraki tur manuel olarak yapılacaksa, kullanıcıdan savaşçıların konumları alınır.Son olarak, pygame.quit() çağrısı ile oyun kapatılır.Ana program main() fonksiyonunu çağırarak başlatılır.

```
            # Kullanıcıya bir sonraki tura geçilsin mi sor
            if ask_for_next_round():
                for player_id in players:
                    fighter_count = ask_for_fighter_count(player_id)
                    # Potansiyel savaşçı konumlarını al ve uygun renkte çiz
                    potential_warrior_positions = place_potential_warriors(players, rows, cols)
                    for pos in potential_warrior_positions:
                        x, y, current_player_id = pos
                        potential_positions[x][y] = True
                    pygame.display.flip()
                    if ask_for_ai_decision(player_id, players):
                        valid_positions = get_valid_placement_guard_positions(player_id, rows, cols,players[player_id]
                        ['guard_position'])
                        ai_place_warriors(player_id, players, rows, cols, valid_positions, warriors_matrix)
                    else:
                        for _ in range(fighter_count):
                            place_warrior_manually(player_id, players, warriors_matrix, available_positions)

        pygame.quit()

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

4.2 Görev Dağılımı

- Projenin Kodlanması: Senem ADALAN
- Projeye Yapay Zeka İşlemlerinin Eklenmesi: Sema Su YILMAZ
- Raporun Hazırlanması: Senem ADALAN, Sema Su YILMAZ

4.3 Karşılaşılan Zorluklar ve Çözüm Yöntemleri

Bu projeyi geliştirirken birçok farklı alanda zorlandığımızı söyleyebiliriz. Ödevi hangi programlama dili ile yazacağımız konusunda karar veremedik. Bu problemin çözümü için bizde önce ödeve başlayan arkadaşlarımızdan, öğretmenlerimizden fikirlerini aldık.Bu sayede projeyi Python programlama dilinde geliştirmeye karar verdik.Python programlama dilini seçtiğimizdeyse oyunu hangi modülü kullanarak tasarlayacağımız konusunda kararsız kaldık.Birkaç gün yaptığımız araştırmalar dahilinde Pygame modülü ile oyunu tasarlamaya başladık.Pygame modülünü kullanırken Türkçe kaynakların oldukça yetersiz olduğunu gördük.Bu problemi çözebilmek için de farklı dillerde yazılan makaleleri, videoları, kitapları araştırdık.Daha önce oyun tasarımı deneyimimiz olmadığı için projenin yazılması kısmında bazı problemlerle, bilgi eksiklikleriyle karşılaştık.İnternette yer alan açık kaynak kodlu ve Python programlama dilinde yazılmış basit düzeyde oyunları inceledik.Yeterli olmayan kısımlarda ise internetten araştırmalarımıza devam ettik.

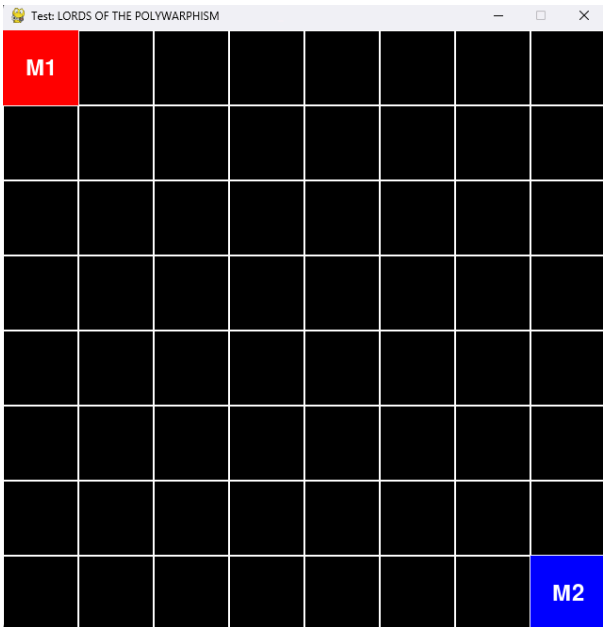
4.4 Proje İsterlerine Göre Eksik Yönler

Saldırı ve Savunma Mekanizması: Oyuncular arasındaki savaş ve saldırı mekanizmaları henüz uygulanmadı. Savaş sırasında savaşçıların birbirleriyle etkileşimleri ve savaş sonucunda kaybedilen veya kazanılan kaynak miktarları gibi faktörler göz önünde bulundurulmalıydı.

Oyun Bitiş Koşulları: Oyunun ne zaman biteceği ve bir oyuncunun kazanma koşulları belirlenmedi. Oyunun sona ermesi için gerekli olan şartlar ve bir oyuncunun galip gelmesi için gereken koşullar tanımlanmalıydı

5. TEST VE DOĞRULAMA

5.1 Yazılımın Test Süreci



Öncelikle pygame kütüphanesi başlatılır ve oyun ekranı oluşturulur.Oyun için varsayılan olarak 2 oyuncu ve 8x8 boyutunda bir oyun alanı belirlenir.Her bir oyuncu için özellik atanır ve oyuncu muhafızlarının başlangıç pozisyonları belirlenir.Ardından oyun ekranında bir ızgara çizilir. Bu ızgara, oyun alanını ve oyuncuların hareket edeceği kareleri temsil eder.Her oyuncunun muhafızı, belirlenen başlangıç pozisyonlarına göre ekrana yerleştirilir.Son olarak oyun ekranı güncellenir ve ekranda çizilen her şey görüntülenir.Bu test kodu, oyunun başlangıç durumunu kontrol etmek ve gerekli bileşenlerin doğru bir şekilde yerleştirilip yerleştirilmediğini görmek için kullanılabilir.

5.2 Yazılımın Doğrulanması

- Oyuncu Sayısının Alınması: Test uygulaması, kullanıcıdan doğru bir şekilde oyuncu sayısını girdi olarak alıyor. Geçersiz girdiler kabul edilmiyor ve kullanıcıya geçerli bir giriş yapması isteniyor.
- Oyuncu Muhafızlarının Konumlandırılması: Her oyuncunun muhafızı, doğru bir şekilde belirlenen başlangıç pozisyonlarına yerleştiriliyor. Bu, ızgarada görsel olarak doğru bir şekilde temsil ediliyor.
- Oyun Ekranının İlk Durumu: Test uygulaması, oyunun ilk durumunu görsel olarak doğru bir şekilde oluşturuyor. Oyuncu muhafızları belirlenen renklerde ve pozisyonlarda görünüyor.
- Oyun İçi Etkileşimlerin Eksikliği: Test uygulaması, şu anda kullanıcı ile etkileşim sağlayacak mekanizmalar içermiyor. Kullanıcıdan herhangi bir giriş alınmadığı için oyun dinamiklerinin test edilmesi mümkün değil.
- Tam Oyun Akışının Sağlanmaması: Test uygulaması, yalnızca oyunun başlangıç durumunu kontrol ediyor ve ardından kapatılıyor. Oyunun tam akışı, oyuncu hareketleri, savaşçı yerleştirmeleri ve kaynak yönetimi gibi diğer temel işlevleri içermiyor.
- Hata Kontrol Mekanizmalarının Eksikliği: Uygulama, kullanıcının hatalı girişler yapmasına izin veriyor ancak bu durumlar için özel bir geri bildirim veya düzeltme mekanizması içermiyor. Kullanıcı yanlış girdi yaptığında, uygulama sadece yeni geçerli bir giriş yapmasını istiyor.

6. GİTHUB LİNKLERİ

7. KAYNAKÇA

https://www.youtube.com/watch?v=ALPGHnJF06s&list=PLgSuixojMiGipAKRR8ZhpyyTJOZXyIi5V

https://www.youtube.com/watch?v=2lrTWbxtlPA&list=PL2rbrnYCWV9wnoDjA60bqU7hsNBDtoYke

https://www.youtube.com/watch?v=rSyDZy9lgZQ&t=601s

https://www.youtube.com/watch?v=Nh7CLz_Z4dU&t=1439s

https://1kodum.com/pythonda-polimorfizm/

https://www.obenseven.com.tr/yazilim/python/nesne-tabanli-programlama/python-cok-bicimlilik-polymorphism/

https://www.obenseven.com.tr/yazilim/python/nesne-tabanli-programlama/python-cok-bicimlilik-polymorphism/

https://sahinbolukbasi.medium.com/python-oop-polymorphism-2d52f8432d22

https://www.sadikturan.com/python-nesne-tabanli-programlama/kalitim/1402

https://duygubulut.wordpress.com/2017/12/02/polimorfizm-ve-sanalvirtual-fonksiyonlar/
https://chat.openai.com/

https://www.youtube.com/watch?v=R8rrR2T9UX8

https://github.com/Samuelczhu/Space_War

https://github.com/techwithtim/PygameForBeginners