T.C

KOCAELİ SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ

MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR/YAZILIM MÜHENDİSLİĞİ

PROJE KONUSU:

LORDS OF THE POLYWARPHISM

ÖĞRENCİ ADI:

AHMET CAN BOSTANCI

DİLA SERAY TEGÜN

ÖĞRENCİ NUMARASI:

220501031

220501022

GİTHUB LİNKLERİ : https://github.com/Bozokhalat https://github.com/dilaseray

DERS SORUMLUSU:

DR. ÖĞR. ÜYESİ NURBANU ALBAYRAK

TARİH: 24.03.2024

1. Giriş

1.1 Projenin amacı:

Bu projenin temel amacı, "Lords of the Polywarphism" adlı bir oyunun temelini oluşturuyor. Oyuncular farklı renklerde sahip oldukları bölgelerde askerler yerleştirir ve ardından sırayla hareket ederler. Oyun tahtası metin tabanlıdır ve bir dosyaya kaydedilir.

Kodun ana bileşenleri:

- **Oyun Döngüsü**: Oyun, bir ana oyun döngüsü içinde çalışır. Bu döngü, oyunun sürekli olarak güncellenmesini, kullanıcı girişlerini işlemesini ve ekrana çizim yapmasını sağlar.

- **Harita ve Karakterler**: Oyun, bir harita üzerinde karakterlerin hareket etmesini sağlar. Karakterler genellikle oyuncu karakteri ve düşmanlar olabilir. Harita, bir grid sistemi kullanarak karakterlerin hareket edebileceği bir alanı temsil eder.

- **Kullanıcı Girişleri**: Oyuncunun karakterini klavye girişleriyle kontrol edebilirsiniz. Örneğin, yön tuşlarıyla karakterinizi hareket ettirebilirsiniz.

- **Çizim ve Grafikler**: pygame kütüphanesi, ekran üzerine grafikler ve metinler çizmeyi sağlar. Bu, karakterlerin ve haritanın ekrana çizilmesini, güncellenmesini ve animasyonlarının oluşturulmasını sağlar.

- **Düşmanlar ve Çatışmalar**: Oyunda genellikle düşmanlar bulunur ve oyuncunun bu düşmanlardan kaçması veya onlarla savaşması gerekir. Çatışma durumları, karakterler arasındaki teması algılayarak ve çeşitli etkileşimlerle gerçekleşir.

- **Oyun Mantığı ve Kurallar**: Oyun genellikle belirli kurallara göre çalışır. Örneğin, karakterlerin belirli sınırlar içinde hareket etmesi veya belirli koşullar altında ölmesi gibi kurallar belirlenir.

- **Oyun Tahtası Oluşturma**: Belirlenen boyuttaki bir oyun tahtası oluşturulur. Bu tahta, satır ve sütunlardan oluşan bir matris şeklindedir. Her hücre bir numara ile belirlenir ve bir dosyada saklanır.

- **Oyuncu Sınıfı**: Her oyuncu kendi karakterlerini (Muhafız, Okçu, Topçu vb.) ve sahip oldukları bölgeleri temsil eder. Oyuncular sırayla hamle yaparlar.

- **Karakter Sınıfları**: Farklı karakter türlerinin (Muhafız, Okçu, Topçu, Atlı vb.) özelliklerini ve davranışlarını tanımlayan sınıflar bulunmaktadır. Her karakterin saldırma menzili, canı, yerleşme yeteneği ve saldırı yeteneği gibi özellikleri vardır.

- **Kodun Akışı**: Oyun sırasında, her oyuncu sırayla hamle yapar. Klavye girişleriyle oyuncular, sahip oldukları bölgelere karakter yerleştirebilirler. Ardından, karakterler belirli bir stratejiye göre hareket eder ve eylemler gerçekleştirirler.
  Bu kod, oyuncular arasında bir savaş stratejisi oyununu simüle etmek için tasarlanmış gibi görünmektedir.

1.2 Beklenen Gerçekleştirmeler:

- Oyuncuların, stratejik birimlerin konumlarını belirleyebilecekleri ve hareket ettirebilecekleri bir oyun alanı sağlanmıştır.

- Farklı karakter sınıflarını temsil eden birimlerin özelliklerini içeren detaylı bir veri tabanı oluşturulmuşur.

- Kullanıcı arayüzü, oyunun akışını yönetmek için gerekli kontrolleri sağlayacak ve oyuncuların rahat bir deneyim yaşamasını sağlayacaktır.

- Oyuncular arası etkileşim, stratejik planlama ve karar alma yeteneklerini test eden dinamik bir oyun ortamı oluşturulmuştur.

2. Gereksinim Analizi

2.1 Arayüz Gereksinimleri:

- Kullanıcı arayüzü, oyunun ana bileşenlerine (oyun alanı, birimlerin listesi, kontrol düğmeleri vb.) erişim sağlayacak şekilde tasarlanmıştır.

- Oyunun akışını yönetmek için kullanıcı arayüzü, kolayca anlaşılabilir ve kullanıcı dostudur.

2.2 Fonksiyonel Gereksinimler:

- Oyuncular, sırayla birimlerini yerleştirebilecek ve hareket ettirebileceklerdir.
- Her bir birimin, farklı yeteneklere ve özelliklere sahip olması, oyunun stratejik derinliğini artıracaktır.

3. Tasarım

3.1 Mimari Tasarım:

- Yazılım, modüler bir yapıda tasarlanacak ve her bir modül, belirli bir oyun özelliğini temsil edecektir. Bu sayede, yazılımın genişletilmesi ve bakımı daha kolay olacaktır.
- Ana bileşenler arasındaki ilişkiler ve veri akışı, açıkça tanımlanacaktır.

3.2 Kullanılacak Teknolojiler:

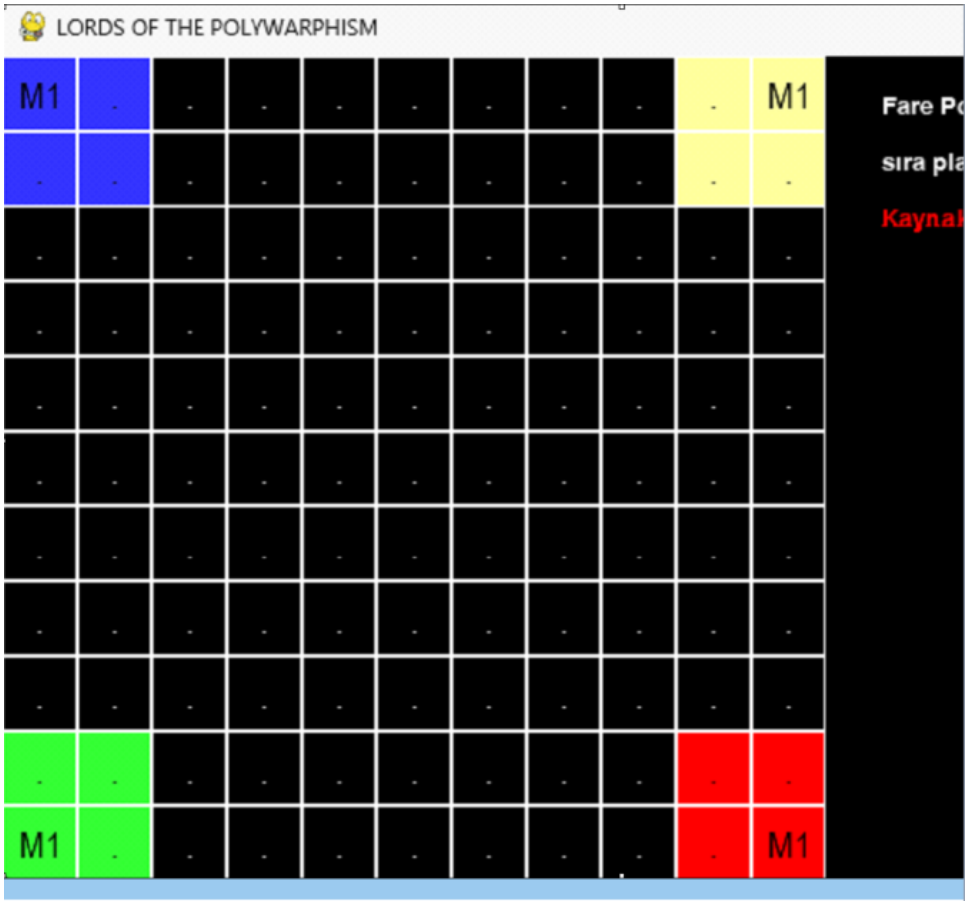
- Python programlama dili, oyunun ana kod tabanını oluşturmuş ve oyun mekaniklerini işletmiştir.
- Pygame kütüphanesi, grafiksel arayüz ve oyun motoru olarak kullanılmıştır. Bu kütüphane, oyunun görsel ve işlevsel yönlerini bir araya getirmiştir.

3.3 Veritabanı Tasarımı:

- Oyunun karakter ve birim verilerini depolamak için basit bir SQL veri tabanı kullanılmıştır. Bu veri tabanı, oyun içindeki karakterlerin özelliklerini ve yeteneklerini saklamıştır.

3.4 Kullanıcı Arayüzü Tasarımı:

- Kullanıcı arayüzü, Pygame kütüphanesi kullanılarak oluşturulacak ve oyunun ana bileşenlerini görsel olarak temsil edecektir. Kullanıcı arayüzü, oyuncuların oyunu kolayca kontrol etmelerini sağlayarak ve oyun deneyimini zenginleştirmiştir.



Kullanıcı arayüzü siyah bir ızgaradan ve sağ taraftaki durum takibinden oluşmaktadır. Üzerinde her farklı rengin başka bir takımı temsil ettiği renkler vardır. (Siyah takım değildir)
Sağ tarafta durum tablosunda hangi takımın sırası olduğu ve o takımın şuan ki kaynağı yer almaktadır.

4. Uygulama

4.1 Kodlanan Bileşenlerin Açıklamaları

```
import pygame

import sys

import random

num_kordi=dict()

kordi_num=dict()

renkler={"red":(255,0,0),"blue":(51,51,255),"green":(51,255,51),"yellow":(255,255,153),"black":(0,0,0),"white":(255,255,255)}
```

Bu kod, pygame kullanarak bir görsel uygulama veya oyun geliştirmek için gerekli olan temel yapıyı oluşturmak için başlangıç adımlarını içermektedir.

```
class Muhafız():

    def __init__(self,isim,numara,renk,player):
        self.can=80

        self.kaynak=10

        self.player=player

        self.numara=numara

        self.komsular = []

        self.saldırma_menzili=[]

        self.SetBolgeRenk(renk)

        karerenkdegistir(screen,renk,self.numara,isim,self.player,can=self.can,kaynak=self.kaynak)

        self.yerlesme_menzil()

        self.saldırma_menzil()
```

Bu kodda öncelikle bir sınıf tanımlanıyor. Daha sonra parametreleri tanımlayan teker teker fonksiyonlar çağırılıyor.

- **self.can=80**: Muhafızın canını temsil eden bir özellik oluşturuluyor ve başlangıç değeri olarak 80 atanıyor.
- **self.kaynak=10**: Muhafızın kaynak miktarını temsil eden bir özellik oluşturuluyor ve başlangıç değeri olarak 10 atanıyor.
- **self.player=player**: Muhafızın hangi oyuncuya ait olduğunu belirten bir özellik oluşturuluyor.
- **self.numara=numara**: Muhafızın numarasını belirten bir özellik oluşturuluyor.
- **self.komsular = []**: Muhafızın komşu muhafızlarını depolamak için bir liste oluşturuluyor.
- **self.saldırma_menzili=[]**: Muhafızın saldırma menzilini depolamak için bir liste oluşturuluyor.
- **self.SetBolgeRenk(renk)**: Belirli bir renge sahip olan bir bölgenin rengini ayarlamak için bir metod çağrılıyor.

```
def SetBolgeRenk(self,renk):

    self.renk=renk

def yerlesme_menzil(self):

    satir = (self.numara - 1) // cols

    sutun = (self.numara - 1) % rows
```

Bu kod parçaları Muhafız sınıfının iki metodunu tanımlar.

- **def SetBolgeRenk(self,renk):**: Bu metod, Muhafızın bulunduğu bölgenin rengini ayarlar. Parametre olarak **renk** alır ve bu rengi Muhafızın **renk** özelliğine atar.
- **def yerlesme_menzil(self):**: Bu metod, Muhafızın yerleşebileceği bölgeyi hesaplar. Öncelikle, Muhafızın numarasını kullanarak, bulunduğu bölgenin satır ve sütununu belirler. Bunun için Muhafızın numarasını satır ve sütun sayısına göre uygun bir şekilde dönüştürür. Bu hesaplama, bir dizi içindeki indislerin matris şeklinde nasıl düzenlendiğini anlamak için yapılır

```
for x in range(max(0, satir - 1), min(satir + 2, len(oyunmatrisi))):

    for y in range(max(0, sutun - 1), min(sutun + 2, len(oyunmatrisi[0]))):

        if (x, y) != (satir, sutun):

            satir=satirokuyucu("game.txt",oyunmatrisi[x][y]).split("*")

            if satir[2]=="." and satir[1]=="black" :

                self.komsular.append(oyunmatrisi[x][y])

for hucre in self.komsular:

    karerenkdegistir(screen,self.renk,hucre,".",self.player)

return self.komsular
```

Bu kod parçası, Muhafızın etrafındaki komşu bölgeleri bulur ve bu bölgeleri bir listeye ekler.

İki iç içe döngü kullanarak, Muhafızın bulunduğu bölgenin etrafındaki bölgeleri tarar. Bu döngüler, Muhafızın bulunduğu bölgenin birim uzaklığı içindeki (dahil) tüm bölgeleri gezerek işlem yapar.

- İlk döngü, **x** değişkeni üzerinde bir dizi indis değeriyle çalışır. Bu indis değerleri, Muhafızın bulunduğu bölgenin satır numarasından bir birim azdan bir birim fazlaya kadar (dahil) olabilir.
- İkinci döngü, **y** değişkeni üzerinde bir dizi indis değeriyle çalışır. Bu indis değerleri, Muhafızın bulunduğu bölgenin sütun numarasından bir birim azdan bir birim fazlaya kadar (dahil) olabilir.
- Her bir iterasyonda, taranan bölgenin koordinatları **(x, y)** olarak alınır ve Muhafızın bulunduğu bölgeye eşit olmaması kontrol edilir.

- Eğer taranan bölge Muhafızın bulunduğu bölgeye eşit değilse, bu bölge oyun matrisinde bulunur. Oyun matrisinden bu bölgenin özellikleri (**game.txt** dosyasından okunan özellikler) alınır.
- Bu özellikler incelenir ve eğer bölge boş (**.**) ve siyah renkte (**black**) ise, bu bölge Muhafızın komşuları listesine eklenir.

```
64      def imha(self):
65
66          karerenkdegistir(screen,"black",self.numara,".")
67
68          for hucre in self.komsular:
69
70              durum=satirokuyucu("game.txt",hucre).split("*")
71
72              if durum[2]==".":
73
74                  karerenkdegistir(screen,"black",hucre,".")
```

Bu kod parçası, Muhafızın kendisini ve komşu bölgelerini yok etmek için kullanılır.

İlk olarak, Muhafızın bulunduğu bölgeyi yok eder. Bu işlem, **karerenkdegistir** fonksiyonuyla gerçekleştirilir. Bu fonksiyon, ekran üzerinde belirli bir bölgenin rengini değiştirir ve bu durum "black" renk ile belirtilerek bölgenin yok olduğu gösterilir.

Daha sonra, Muhafızın komşu bölgeleri kontrol edilir. Muhafızın komşu bölgeleri, **self.komsular** listesinde saklanır. Her bir komşu bölge için aşağıdaki işlemler yapılır:

- Komşu bölgenin özellikleri **game.txt** dosyasından okunur.
- Eğer komşu bölge boş ise (**.**) ve üzerinde birim bulunmuyorsa, yani yok edilebilir durumdaysa, o bölge de yok edilir. Bu işlem yine **karerenkdegistir** fonksiyonu kullanılarak gerçekleştirilir.

```
76      def saldırma_menzil(self):
77
78          satir = (self.numara - 1) // rows
79
80          sutun = (self.numara - 1) % cols
81
82          for x in range(max(0, satir - 1), min(satir + 2, len(oyunmatrisi))):
83
84              for y in range(max(0, sutun - 1), min(sutun + 2, len(oyunmatrisi[0]))):
85
86                  if (x, y) != (satir, sutun):
87
88                      self.saldırma_menzili.append(oyunmatrisi[x][y])
```

Bu kod parçası, bir Muhafızın saldırma menzilini belirlemek için kullanılır. İlk olarak, Muhafızın bulunduğu satır ve sütun hesaplanır. Daha sonra, Muhafızın

bulunduğu hücrenin etrafındaki komşu hücreler taranır. Her bir komşu hücre, Muhafızın saldırma menziline eklenir. Bu şekilde, Muhafızın saldırabileceği tüm hücreler belirlenmiş olur.

```
    def bilgiguncelle(self):

        durum=satirokuyucu("game.txt",self.numara).split("*")

        self.can=float(durum[4])

    def saldir(self):

       for i in self.saldırma_menzili:

           dusman=satirokuyucu("game.txt",i).split("*")

           if dusman[1]!=self.renk and  dusman[2]!=".":

               bilgi=str(dusman[0])+"*"+dusman[1]+"*"+dusman[2]+"*"+dusman[3]+"*"+str(float(dusman[4])-20)

               satiryazici("game.txt",i,bilgi)
```

Bu kod, bir Muhafızın saldırı yapmasını ve saldırı sonrası oyun durumunu güncellemesini sağlar. **saldir** fonksiyonu, Muhafızın saldırma menzilindeki her bir hücreyi tarar. Eğer hedef hücrede bir düşman varsa ve bu düşman Muhafızın rengiyle farklı ise (yani düşman değilse) ve hücre doluysa, düşmanın can değeri 20 azaltılarak güncellenir. Bu durum, saldırı gerçekleştiğinde düşmanın canının azalmasını simgeler.

bilgiguncelle fonksiyonu ise Muhafızın bilgilerini günceller. Özellikle Muhafızın canını günceller. Bu bilgiler oyunun durumunu temsil eden bir dosyadan (**game.txt**) okunur ve Muhafızın can değeri güncellenir.

```
class Okcu(Muhafız):

    def saldırma_menzil(self):

        boyut = len(oyunmatrisi)

        satir = (self.numara - 1) // boyut

        sutun = (self.numara - 1) % boyut

        self.saldırma_menzili = []

        if satir+1<boyut and sutun+1<boyut:

            self.saldırma_menzili.append(oyunmatrisi[satir+1][sutun+1])

        if satir+2<boyut and sutun+2<boyut:

            self.saldırma_menzili.append(oyunmatrisi[satir+2][sutun+2])
```

Bu kod, **Okcu** sınıfının **Muhafız** sınıfını miras almasını sağlar ve **saldırma_menzil** metodunu özelleştirir. **saldırma_menzil** metodu, okçunun

saldırma menzilini hesaplar. Okçu sınıfı, Muhafız sınıfının davranışını devralır ancak saldırma menzili farklıdır.

boyut değişkeni, oyun matrisinin boyutunu belirtir. **satir** ve **sutun** değişkenleri, okçunun bulunduğu hücrenin satır ve sütun indislerini hesaplar. Ardından, okçunun saldırma menzilindeki hücreler belirlenir. Eğer okçu oyun matrisinin sınırlarında ise ve saldırma menzili içindeki hücreler oyun matrisinin boyutları içindeyse, bu hücreler saldırma menziline eklenir. Bu sayede okçunun saldırabileceği hücreler belirlenmiş olur.

```
124         if satir+2<boyut and sutun+2<boyut:
125 
126             self.saldırma_menzili.append(oyunmatrisi[satir+2][sutun+2])
127 
128         if satir-1>=0 and sutun+1<boyut:
129 
130             self.saldırma_menzili.append(oyunmatrisi[satir-1][sutun+1])
131 
132         if satir-2>=0 and sutun+2<boyut:
133 
134             self.saldırma_menzili.append(oyunmatrisi[satir-2][sutun+2])
135 
136         if satir-1>=0 and sutun-1>=0:
137 
138             self.saldırma_menzili.append(oyunmatrisi[satir-1][sutun-1])
139 
140         if satir+1<boyut and sutun-1>=0:
141 
142             self.saldırma_menzili.append(oyunmatrisi[satir+1][sutun-1])
143 
144         if satir-2>=0 and sutun-2>=0:
145 
146             self.saldırma_menzili.append(oyunmatrisi[satir-2][sutun-2])
```

Bu kod parçası, okçunun saldırma menzilini daha genişletir. Okçu, çapraz yönlere de saldırabilir hale getirilir.

Her bir **if** koşulu, okçunun saldırma menzilindeki hücreleri belirler. Örneğin, **if satir+2<boyut and sutun+2<boyut:** koşulu, sağ alt çapraz yöndeki hücreyi kontrol eder ve eğer oyun matrisi sınırları içinde ise, bu hücre saldırma menziline eklenir.

Diğer **if** koşulları da benzer şekilde farklı yönlerdeki hücreleri kontrol eder ve uygun olanları saldırma menziline ekler. Bu sayede okçunun saldırabileceği hücreler daha geniş bir alana yayılmış olur.

```
148         if satir+2<boyut and sutun-2>=0:
149 
150             self.saldırma_menzili.append(oyunmatrisi[satir+2][sutun-2])
151 
152         satir = (self.numara - 1) // boyut
153 
154         sutun = (self.numara - 1) % boyut
```

Bu kod parçası, okçunun saldırma menzilini genişletirken sol alt çapraz yöne de saldırmasını sağlar.

İlk **if** koşulu, **satir** değerini iki birim aşağısında ve **sutun** değerini iki birim solundaki hücreyi kontrol eder. Eğer bu hücre oyun matrisi sınırları içindeyse, bu hücre saldırma menziline eklenir.

Sonrasında, **satir** ve **sutun** değerleri tekrar hesaplanarak okçunun pozisyonunun oyun matrisindeki konumu güncellenir. Bu işlem, her hücrenin kontrol edilmesi ve uygun olanların saldırma menziline eklenmesi için döngüdeki bir sonraki adım için hazırlık yapar.

```
156         for x in range(max(0, satir - 2), min(boyut, satir + 3)):
157 
158             for y in range(max(0, sutun - 2), min(boyut, sutun + 3)):
159                     # Geçerli indekslerin sınırlarını kontrol et ve sadece sağ, sol, yukarı ve aşağı hücreleri al
160                 if (x, y) != (satir, sutun) and (x == satir or y == sutun):
161 
162                     self.saldırma_menzili.append(oyunmatrisi[x][y])
```

Bu kod parçası, okçunun saldırma menzilini genişletmek için daha karmaşık bir yaklaşım kullanır.

İç içe iki döngü kullanarak, okçunun bulunduğu hücrenin çevresindeki hücreleri kontrol eder. **`max`** ve **`min`** fonksiyonları, döngünün sınırlarını belirler ve bu sınırlar, oyun matrisinin boyutuyla sınırlanır.

Her bir hücrenin indisleri, okçunun bulunduğu hücreden en fazla iki birim uzaklıkta olabilir. Ancak sınırların dışına taşmamak için bu değerler **`max`** ve **`min`** ile kontrol edilir.

`if` koşulu, sadece sağa, sola, yukarıya veya aşağıya olan hücreleri kabul eder. Çapraz hücreler bu koşula uymaz. Bu şekilde, okçunun saldırma menzilindeki hücreler belirlenir ve **`self.saldırma_menzili`** listesine eklenir.

```
164     def SetBolgeRenk(self,renk):
165
166         self.can=30
167
168         self.kaynak=20
169
170         self.renk=renk
171
172     def saldir(self):
173
174         saldirilacak=[]
175
176         for i in self.saldırma_menzili:
177
178             dusman=satirokuyucu("game.txt",i).split("*")
179
180             if dusman[1]!=self.renk and dusman[2]!=".":
181
182                 saldirilacak.append(dusman)
183
184         saldirilacak=sorted(saldirilacak, key=lambda x: x[-2], reverse=True)
```

Bu kod parçası, bir savaşçının saldırma yöntemini tanımlar.

- **`SetBolgeRenk`** metodu: Savaşçının bölge rengini belirler. Aynı zamanda, savaşçının canını ve kaynağını da ayarlar. Bu değerler sabit olarak 30 ve 20 olarak atanmıştır.
- **`saldir`** metodu: Savaşçının saldırma işlemini gerçekleştirir. Öncelikle saldırılacak düşmanları depolamak için bir liste oluşturulur. Daha sonra, savaşçının saldırma menzilindeki her hücre için döngüye girilir. Her hücre için oyun matrisinden bilgiler alınır ve bu bilgiler düşmanların rengi ve durumu kontrol edilerek saldırıya uygun olanlar **`saldirilacak`** listesine eklenir.
  Ek olarak, saldırılacak düşmanlar **`sorted`** fonksiyonuyla sıralanır. Bu sıralama, düşmanların dayanıklılık değerine göre azalan şekilde yapılır (**`x[-2]`** ifadesi, düşmanın dayanıklılık değerine karşılık gelir). Böylece, saldırılacak düşmanlar dayanıklılık değerlerine göre önceliklendirilmiş olur.

```
188         for i in range(len(saldirilacak)):
189
190             can=float(saldirilacak[i][4])-float(float(saldirilacak[i][4])*(60/100))
191
192             bilgi=str(saldirilacak[i][0])+"*"+saldirilacak[i][1]+"*"+saldirilacak[i][2]+"*"+saldirilacak[i][3]+"*"+str(can)
193
194             satiryazici("game.txt",int(saldirilacak[i][0]),bilgi)
195
196     else:
197
198         for i in range(3):
199
200             can=float(saldirilacak[i][4])-float(float(saldirilacak[i][4])*(60/100))
201
202             bilgi=str(saldirilacak[i][0])+"*"+saldirilacak[i][1]+"*"+saldirilacak[i][2]+"*"+saldirilacak[i][3]+"*"+str(can)
203
204             satiryazici("game.txt",int(saldirilacak[i][0]),bilgi)
```

Bu kod parçası, saldırılacak düşmanları belirli bir sıraya göre sınırlar. Eğer saldırılacak düşman sayısı 3'ten azsa, tüm düşmanlara saldırı gerçekleştirilir. Ancak, 3'ten fazla düşman varsa, sadece ilk üç düşmana saldırılır.

- **`if len(saldirilacak) <= 3:`**: **`saldirilacak`** listesinin uzunluğu 3'ten azsa, yani saldırılacak düşman sayısı 3 veya daha azsa, tüm düşmanlara saldırı gerçekleştirilir. Bu durumda bir döngü oluşturulur ve her düşmana karşı saldırı yapılır.
- **`else:`**: **`saldirilacak`** listesinin uzunluğu 3'ten fazla ise, yani saldırılacak düşman sayısı 3'ten fazlaysa, sadece ilk üç düşmana saldırılır. Bu durumda bir döngü oluşturulur ve sadece ilk üç düşmana karşı saldırı yapılır.
  Her iki durumda da, her bir düşmanın dayanıklılık değeri saldırı sonrası %60 azaltılarak güncellenir ve bu bilgiler dosyaya yazılır. Bu işlem, düşmanların dayanıklılıklarının bir saldırı sonrası azalmasını simüle eder.

```
206 class Topcu(Muhafız):
207
208     def saldırma_menzil(self):
209
210         boyut = len(oyunmatrisi)
211         # Numaranın indekslerini bul
212         satir = (self.numara - 1) // boyut
213
214         sutun = (self.numara - 1) % boyut
215
216         for x in range(max(0, satir - 2), min(boyut, satir + 3)):
217
218             for y in range(max(0, sutun - 2), min(boyut, sutun + 3)):
219                 # Geçerli indekslerin sınırlarını kontrol et ve sadece sağ, sol, yukarı ve aşağı hücreleri al
220                 if (x, y) != (satir, sutun) and (x == satir or y == sutun):
221
222                     self.saldırma_menzili.append(oyunmatrisi[x][y])
```

Bu kod, **`Topcu`** sınıfının **`saldırma_menzil`** metodunu tanımlar. Bu metod, topçunun saldırabileceği hücreleri belirler.

İlk olarak, **`oyunmatrisi`**nin boyutunu temsil eden **`boyut`** değişkeni hesaplanır. Ardından, topçunun bulunduğu hücrenin indekslerini hesaplamak için **`satir`** ve **`sutun`** değişkenleri oluşturulur.

Daha sonra, iki döngü kullanarak topçunun saldırabileceği hücrelerin indekslerini belirler. Bu döngüler, topçunun bulunduğu hücrenin etrafındaki 3x3'lük bir bölgeyi tarar. Ancak, sadece bu bölgenin kenarındaki hücreler (yani sağ, sol, yukarı ve aşağıdaki hücreler) **`saldırma_menzili`** listesine eklenir.

Sonuç olarak, **`saldırma_menzili`** listesi, topçunun saldırabileceği hücrelerin indekslerini içerir. Bu indeksler, oyun matrisindeki hedef hücrelerin konumlarını temsil eder.

```
224     def SetBolgeRenk(self,renk):
225
226         self.can=30
227
228         self.kaynak=50
229
230         self.renk=renk
```

Bu kod, **SetBolgeRenk** adında bir metod tanımlar. Bu metod, bir **Muhafız** nesnesinin özelliklerini ayarlar.

- **renk** parametresi, muhafızın rengini belirtir.
- **self.can** ve **self.kaynak** özellikleri muhafızın canını ve kaynağını belirler.
- **self.renk** özelliği, muhafızın rengini saklar.
  Yani bu metod, muhafızın renk, can ve kaynak özelliklerini başlatır veya günceller.

```
232 def saldir(self):
233
234     saldirilacak=[]
235
236     for i in self.saldırma_menzili:
237
238         dusman=satirokuyucu("game.txt",i).split("*")
239
240         if dusman[1]!=self.renk and dusman[2]!=".":
241
242             saldirilacak.append(dusman)
243
244     saldirilacak=sorted(saldirilacak, key=lambda x: x[-2], reverse=True)
245
246     can=0
247
248     bilgi=str(saldirilacak[0][0])+"*"+saldirilacak[0][1]+"*"+saldirilacak[0][2]+"*"+saldirilacak[0][3]+"*"+str(can)
249
250     satiryazici("game.txt",int(saldirilacak[0][0]),bilgi)
```

Bu kod bir metod tanımlar: **`saldir`**. Bu metod, bir muhafızın saldırı işlemini gerçekleştirir.

- **`saldirilacak`** adında bir boş liste oluşturulur.
- **`self.saldırma_menzili`** içindeki hücreler üzerinde döngü yapılır.
- Her hücre için **`satirokuyucu`** fonksiyonu kullanılarak dosyadan veri okunur ve bu veri **`dusman`** değişkenine atanır.
- Eğer düşmanın rengi ve durumu belirli koşullara uygunsa (**`dusman[1]!=self.renk`** ve **`dusman[2]!="."`**) **`saldirilacak`** listesine eklenir.
- **`saldirilacak`** liste, listedeki 4. elemana göre büyükten küçüğe sıralanır.
- **`can`** değişkeni sıfırlanır.
- **`bilgi`** değişkenine, saldırılacak hedefin bilgileri ve sıfırlanan can değeri atanır.
- **`satiryazici`** fonksiyonu kullanılarak hedefin bilgileri güncellenir.

```
252 class Atlı(Muhafız):
253
254     def saldırma_menzil(self):
255
256         boyut = len(oyunmatrisi)
257
258         satir = (self.numara - 1) // boyut
259
260         sutun = (self.numara - 1) % boyut
261
262         if satir+1<boyut and sutun+1<boyut:
263
264             self.saldırma_menzili.append(oyunmatrisi[satir+1][sutun+1])
265
266         if satir+2<boyut and sutun+2<boyut:
267
268             self.saldırma_menzili.append(oyunmatrisi[satir+2][sutun+2])
269
270         if satir-1>=0 and sutun+1<boyut:
271
272             self.saldırma_menzili.append(oyunmatrisi[satir-1][sutun+1])
```

Bu kod, Atlı sınıfında bulunan **saldırma_menzil** metodunu tanımlar. Bu metod, Atlı birimlerinin saldırabileceği hücreleri belirler.

- Öncelikle oyun matrisinin boyutu **boyut** değişkenine atanır.
- Ardından, Atlı birimin bulunduğu hücrenin konumundan (indeks) satır ve sütun bilgileri (**satir** ve **sutun**) hesaplanır.
- Atlı birimin saldırabileceği hücrelerin koordinatları belirlenir:

  - Eğer Atlı birimin bulunduğu hücrenin bir alt ve bir sağında bulunan hücre matris sınırları içindeyse, bu hücre **saldırma_menzili** listesine eklenir.
  - Eğer Atlı birimin bulunduğu hücrenin iki alt ve iki sağında bulunan hücre matris sınırları içindeyse, bu hücre de **saldırma_menzili** listesine eklenir.
  - Eğer Atlı birimin bulunduğu hücrenin bir üst ve bir sağında bulunan hücre matris sınırları içindeyse, bu hücre de **saldırma_menzili** listesine eklenir.

```
310     def SetBolgeRenk(self,renk):
311
312         self.can=40
313
314         self.kaynak=30
315
316         self.renk=renk
317
318     def saldir(self):
319
320         saldirilacak=[]
321
322         for i in self.saldırma_menzili:
323
324             dusman=satirokuyucu("game.txt",i).split("*")
325
326             if dusman[1]!=self.renk and dusman[2]!=".":
327
328                 saldirilacak.append(dusman)
329
330         saldirilacak=sorted(saldirilacak, key=lambda x: x[-1], reverse=True)
```

Bu kod, Atlı sınıfında bulunan **SetBolgeRenk** ve **saldir** metodlarını tanımlar.

SetBolgeRenk metodunda:

- Atlı birimin bölgesinin rengi **renk** parametresiyle belirlenir.
- Atlı birimin can değeri **40** olarak atanır.
- Atlı birimin kaynak değeri **30** olarak atanır.
- Atlı birimin rengi **renk** olarak ayarlanır.

saldir metodunda ise:

- Atlı birimin saldırabileceği hücrelerdeki düşman birimleri bulunur ve bu bilgiler **saldirilacak** listesine eklenir.
- **saldirilacak** listesi düşman birimlerine göre sıralanır, en güçlü düşman birimi listenin başına gelir. Bu sıralamada, düşman birimlerinin can değerlerine (**x[-1]**) göre bir sıralama yapılır. Sıralama tersten yapılır, yani en yüksek can değeri en üstte olacak şekilde sıralanır.

```
332         if len(saldirilacak)<=3:
333
334             for i in range(len(saldirilacak)):
335
336                 can=float(saldirilacak[i][4])-float(30)
337
338                 bilgi=str(saldirilacak[i][0])+"*"+saldirilacak[i][1]+"*"+saldirilacak[i][2]+"*"+saldirilacak[i][3]+"*"+str(can)
339
340                 satiryazici("game.txt",int(saldirilacak[i][0]),bilgi)
341
342         else:
343
344             for i in range(3):
345
346                 can=float(saldirilacak[i][4])-float(30)
347
348                 bilgi=str(saldirilacak[i][0])+"*"+saldirilacak[i][1]+"*"+saldirilacak[i][2]+"*"+saldirilacak[i][3]+"*"+str(can)
349
350                 satiryazici("game.txt",int(saldirilacak[i][0]),bilgi)
```

Bu kod, Atlı birimin saldırı metodunda bulunan bir kontrol yapısını ve saldırı işlemini tanımlar.

Eğer **saldirilacak** listesinin uzunluğu 3 veya daha az ise:

- **for** döngüsüyle, **saldirilacak** listesindeki her düşman birimi için aşağıdaki işlemler yapılır:
  - Düşman biriminin can değeri 30 azaltılır.
  - Yeni can değeri ve diğer bilgiler birleştirilerek yeni bir bilgi oluşturulur.
  - Oluşturulan bilgi, **game.txt** dosyasındaki ilgili satıra yazılır.

Eğer **saldirilacak** listesinin uzunluğu 3'ten fazla ise:

- **for** döngüsüyle, en güçlü üç düşman birimi için aşağıdaki işlemler yapılır:
  - Düşman biriminin can değeri 30 azaltılır.
  - Yeni can değeri ve diğer bilgiler birleştirilerek yeni bir bilgi oluşturulur.
  - Oluşturulan bilgi, **game.txt** dosyasındaki ilgili satıra yazılır.

```
352 class Saglikci(Muhafiz):
353
354     def saldırma_menzil(self):
355
356         boyut = len(oyunmatrisi)
357
358         satir = (self.numara - 1) // boyut
359
360         sutun = (self.numara - 1) % boyut
361
362         self.saldırma_menzili = []
363
364         if satir+1<boyut and sutun+1<boyut:
365
366             self.saldırma_menzili.append(oyunmatrisi[satir+1][sutun+1])
```

Bu kod, **Saglıkcı** sınıfında bulunan **saldırma_menzil** metodu ile ilgili bilgileri içerir.

Bu metod, **Saglıkcı** sınıfına ait bir nesnenin saldırma menzilini belirlemek için kullanılır. İlgili oyun matrisindeki hücrelerin indekslerini hesaplar ve saldırma menzilini oluşturur.

Kodda şu adımlar gerçekleştirilir:

- **boyut** değişkenine oyun matrisinin boyutu atanır.
- Verilen birim numarasından satır ve sütun indeksleri hesaplanır.
- **saldırma_menzili** listesi başlangıçta boş olarak oluşturulur.
- Eğer bir sonraki satır ve sütun indeksleri oyun matrisinin boyutunu aşmıyorsa, bu hücre **saldırma_menzili** listesine eklenir.

Bu şekilde, **Saglıkcı** birimlerinin sadece kendilerine yakın olan bir hücreye saldırabilecekleri menzil belirlenmiş olur.

```
    def SetBolgeRenk(self,renk):

        self.can=100

        self.kaynak=10

        self.renk=renk

    def saldir(self):

        iyilestiricek=[]

        for i in self.saldırma_menzili:

            dost=satirokuyucu("game.txt",i).split("*")

            if dost[1]==self.renk and dost[2]!=".":

                iyilestiricek.append(dost)

        iyilestiricek=sorted(iyilestiricek, key=lambda x: x[-2], reverse=False)
```

Bu kod, **SetBolgeRenk** ve **saldir** metodlarını içeren **Saglıkcı** sınıfına aittir. Aşağıda bu iki metodun neyi açıkladığı belirtilmiştir:

- **SetBolgeRenk** Metodu:
  - Bu metod, bir **Saglıkcı** biriminin bölge rengini ayarlamak için kullanılır.
  - **renk** parametresiyle gelen değer **Saglıkcı** biriminin rengini belirler.
  - **self.can** ve **self.kaynak** değerleri belirli sabit değerlere atanır. Bu değerler, birimin canını ve kaynaklarını temsil eder.
- **saldir** Metodu:
  - Bu metod, bir **Saglıkcı** biriminin saldırma işlemini gerçekleştirir, ancak Saglıkcı birimi aslında iyileştirme işlemi yapmaktadır.
  - **iyilestiricek** adında bir boş liste oluşturulur.
  - **self.saldırma_menzili** içindeki her hücre için döngüye girilir.
  - Her hücre için, o hücrede bulunan dost birimler kontrol edilir.
  - Eğer dost birim bulunursa ve hedefte nokta (**.**) yoksa, bu dost birim **iyilestiricek** listesine eklenir.
  - **iyilestiricek** listesi, birimlerin canlarına göre küçükten büyüğe sıralanır.
  - En az cana sahip birime iyileştirme yapılacak şekilde, sıralı liste kullanılarak birimler iyileştirilir.

```
        if len(iyilestiricek)<=3:

            for i in range(len(iyilestiricek)):

                can=float(iyilestiricek[i][4])+float(float(iyilestiricek[i][4])*(50/100))

                bilgi=str(iyilestiricek[i][0])+"*"+iyilestiricek[i][1]+"*"+iyilestiricek[i][2]+"*"+iyilestiricek[i][3]+"*"+str(can)

                satiryazici("game.txt",int(iyilestiricek[i][0]),bilgi)

        else:

            for i in range(3):

                can=float(iyilestiricek[i][4])+float(float(iyilestiricek[i][4])*(50/100))

                bilgi=str(iyilestiricek[i][0])+"*"+iyilestiricek[i][1]+"*"+iyilestiricek[i][2]+"*"+iyilestiricek[i][3]+"*"+str(can)

                satiryazici("game.txt",int(iyilestiricek[i][0]),bilgi)
```

- Bu kod, **saldir** metodunun devamı niteliğindedir ve birimleri iyileştirmek için kullanılır. Aşağıda kodun açıklaması verilmiştir:

- Eğer **iyilestiricek** listesindeki birim sayısı 3 veya daha az ise:
  - **iyilestiricek** listesindeki her bir birim için bir döngü oluşturulur.
  - Her bir birimin canı artırılır. Bu artış yüzde 50'dir.
  - Birimin yeni bilgileri **bilgi** değişkenine atanır.
  - **satiryazici** fonksiyonu kullanılarak, **game.txt** dosyasındaki ilgili birim bilgileri güncellenir.
- Eğer **iyilestiricek** listesindeki birim sayısı 3'ten fazla ise:
  - Sadece ilk 3 birim iyileştirilir.
  - Yukarıdaki adımlar, sadece ilk üç birim için gerçekleştirilir.

```
hak=0
class Oyuncu():

    def __init__(self,oyuncu,renk,no):

        self.no=no

        self.devamdurumu=True

        self.anliksaldirikarakter=[]

        self.hazine=200

        self.oyuncu=oyuncu

        self.renk=renk

        self.oyunmatrisi=oyunmatrisi
```

Bu kod, **Oyuncu** sınıfını tanımlar. Bu sınıf, oyuncuların oyun içindeki temel özelliklerini ve durumlarını temsil etmek için kullanılır.

```
487     def cankontrol(self):
488
489         indx=0
490
491         for i in self.karakterlist:
492
493             i.bilgiguncelle()
494
495             if i.can<=0:
496
497                 i.imha()
498
499                 self.karakterlist.pop(indx)
500
501         indx+=1
```

Bu metod, oyuncunun kontrol ettiği karakterlerin can durumunu kontrol eder. İşlevi şu adımlarla gerçekleşir:

- **karakterlist** içindeki her karakter için bir döngü başlatılır.
- Her karakterin can durumu güncellenir (**bilgiguncelle()** metodunu çağırarak).
- Karakterin canı 0 veya daha az ise:
  - Karakter imha edilir (**imha()** metodunu çağırarak).
  - Karakter, **karakterlist** listesinden kaldırılır.
- Döngü bu şekilde tüm karakterler için tamamlandıktan sonra **indx** değişkeni artırılır.

Bu metod, oyuncunun kontrol ettiği karakterlerin can durumlarını izler ve canı sıfıra düşen karakterleri oyundan kaldırır.

```
503     def startgame(self):
504 
505         nesne=Muhafız("M1",self.no,self.renk,self.oyuncu)
506 
507         self.karakterlist.append(nesne)
508 
509     def toplam_alan(self):
510 
511         self.etkin_alan_numaraları=[]
512 
513         for i in range(1,cols*rows):
514 
515             hucre=satirokuyucu("game.txt",i).split("*")
516 
517             if  hucre[2]=="." and hucre[1]==self.renk:
518 
519                 self.etkin_alan_numaraları.append(int(hucre[0]))
```

Bu iki metod aşağıdaki işlevlere sahiptir:

- **startgame(self)**:
  - Bu metod, oyuncunun bir Muhafız nesnesi oluşturmasını sağlar.
  - Oluşturulan nesne, oyuncunun kontrol ettiği karakter listesine eklenir.
- **toplam_alan(self)**:
  - Bu metod, oyuncunun kontrol ettiği rengin sahip olduğu ve boş olan tüm alan numaralarını belirler.
  - Dosyadan okunan her satır kontrol edilir.
  - Eğer hücrenın içeriği boş ise (**hucre[2]=="."**) ve hücre rengi oyuncunun rengine eşitse (**hucre[1]==self.renk**), o zaman bu hücrenin numarası **etkin_alan_numaraları** listesine eklenir. Bu, oyuncunun kontrol ettiği karakterlerin yerleştirilebileceği alanları belirlemek için kullanılır.
  -

```
549         if karakterkodu=="1" and  (kordinatno in self.etkin_alan_numaraları):
550 
551             hak+=1
552 
553             self.atlısay+=1
554 
555             nesne=Atlı("A"+str(self.atlısay),kordinatno,self.renk,self.oyuncu)
556 
557             self.karakterlist.append(nesne)
558 
559             self.anliksaldirikarakter.append(nesne)
```

Bu kod parçası, belirli bir karakter koduna (**karakterkodu**) sahip bir nesnenin oluşturulması işlemini gerçekleştirir. Ayrıca, bu nesnenin oluşturulabileceği bir alanı kontrol eder.

- **karakterkodu == "1"** kontrolü, belirli bir karakter kodunun (**1**) olup olmadığını kontrol eder.
- **(kordinatno in self.etkin_alan_numaraları)** ifadesi, **kordinatno** adlı bir değişkenin, oyuncunun kontrol ettiği etkin alan numaraları listesinde (**self.etkin_alan_numaraları**) olup olmadığını kontrol eder. Bu, karakterin yerleştirilebileceği geçerli bir alana sahip olup olmadığını kontrol eder.
- Eğer her iki kontrol de sağlanıyorsa, bu karakterin yerleştirilebileceği bir alan vardır ve belirtilen koordinatlarda bir Atli nesnesi oluşturulur.
- Oluşturulan nesne, oyuncunun kontrol ettiği karakter listesine (**self.karakterlist**) ve anlık saldırı karakterleri listesine (**self.anliksaldirikarakter**) eklenir.
-

```
608  pygame.init() # Modül etkinleştirme
609
610  SQUARE_SIZE = 40 # Kare boyutu
611
612  nokta_font = pygame.font.SysFont("arial", 20) # Nokta yazı fontu
613
614  clock = pygame.time.Clock() # Oyun saati etkinleştirme
```

Bu kod parçası, bir Pygame uygulamasının başlatılmasını sağlar ve gerekli bileşenleri etkinleştirir:

- **`pygame.init()`**: Pygame modülünü etkinleştirir ve Pygame fonksiyonlarını kullanılabilir hale getirir.
- **`SQUARE_SIZE = 40`**: Kare boyutunu **`40`** olarak ayarlar. Bu değer, oyun alanındaki karelerin boyutunu belirler.
- **`nokta_font = pygame.font.SysFont("arial", 20)`**: "arial" adlı fonttan 20 boyutunda bir font oluşturur. Bu font, oyun ekranında noktaların yazılmasında kullanılabilir.
- **`clock = pygame.time.Clock()`**: Oyun saati için bir clock nesnesi oluşturur. Bu, oyunun belirli bir hızda çalışmasını sağlar.

```python
def matris_olustur(satir_sayisi, sutun_sayisi):

    matris = []

    sayac = 1

    for x1 in range(satir_sayisi):

        satir = []

        for x2 in range(sutun_sayisi):

            satir.append(sayac)

            sayac += 1

        matris.append(satir)

    return matris
```

Bu kod, belirli bir satır ve sütun sayısına sahip bir matris oluşturur. Her bir hücrenin değeri, matris içindeki konumuna göre artan bir sayıdır. Matris oluşturulduktan sonra oluşturulan matrisi döndürür.

```python
def draw_grid(screen, rows, cols,dosya):

    size=1

    for row in range(rows):

        for col in range(cols):

            xkord = col * SQUARE_SIZE

            ykord = row * SQUARE_SIZE

            noktatext = nokta_font.render(".", True, renkler["white"])

            text_rect = noktatext.get_rect(center=(xkord + SQUARE_SIZE // 2, ykord + SQUARE_SIZE // 2))

            pygame.draw.rect(screen, renkler["white"], (xkord, ykord, SQUARE_SIZE, SQUARE_SIZE),1)

            screen.blit(noktatext, text_rect)

            num_kordi[size]=(xkord,ykord)
            kordi_num[(xkord,ykord)]=size
            size+=1
```

Bu kod, bir oyun ekranında satır ve sütunlara göre bir ızgara oluşturur. Her hücre bir kareyi temsil eder. Her bir karenin ortasına bir nokta (".") yerleştirilir

ve bu noktalar oyunun temelini oluşturur. Ayrıca, her bir hücrenin koordinatlarını (x, y) içeren bir sözlük oluşturur. Bu sözlükler, her bir karenin numarasını koordinatlarına ve koordinatlarını numarasına eşler. Bu işlem oyun ekranının düzenini belirlemek için yapılmaktadır.

```
686     pygame.display.flip()  # Ekranı güncelle
687 
688     with open(dosya,"w",encoding="utf-8")as kayit:
689 
690         for konum in range(1,rows*cols+1):
691 
692             bilgi="{}*black*.*Unkown\n".format(konum)
693 
694             kayit.write(bilgi)
```

Bu kod, bir dosyaya oyun alanındaki her bir hücre için bir kayıt oluşturur. Her bir hücre için bir numara atanır ve varsayılan olarak bu hücrenin rengi siyah ("black") ve içeriği nokta ("."). Bilgi, her bir hücrenin numarası, rengi, içeriği ve tanımı (bilinmeyen, yani "Unknown") içeren bir dize olarak formatlanır ve dosyaya yazılır. Bu işlem, oyun başlamadan önce oyun alanını ve hücrelerin başlangıç durumunu belirlemek için yapılır.pygame.display.flip() ise oyun ekranını günceller.

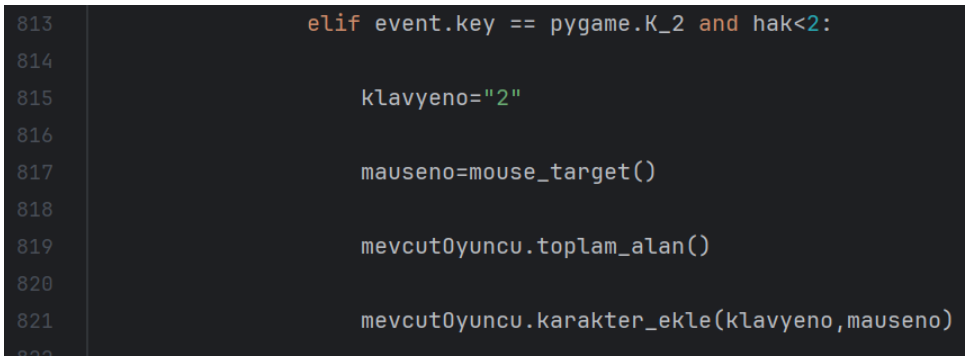
```
714 def mouse_target():
715 
716     for i in num_kordi.values():
717 
718         mouse_xx, mouse_yy = pygame.mouse.get_pos()
719 
720         if i[0]<mouse_xx<i[0]+40 and i[1]<mouse_yy <i[1]+40 :
721 
722             return kordi_num[(i[0],i[1])]
```

Bu fonksiyon, fare konumunu izleyerek fare imlecinin bulunduğu hücrenin numarasını belirler. Her bir hücrenin köşe koordinatları, fare imleci (mouse) koordinatları ile karşılaştırılır. Fare imleci bir hücrenin içindeyse, bu hücrenin numarası döndürülür. Bu sayede, fare imleci üzerinde bulunduğu hücre belirlenmiş olur.

```
while True:
    # Olayları işle
    mevcutOyuncu=sıra[index]

    if  not mevcutOyuncu=="saldiri":

        textsıra=font.render("sıra {}({})".format(mevcutOyuncu.oyuncu,mevcutOyuncu.renk),True,renkler["white"])

    else:
        textsıra=font.render("sıra {}".format("saldiri"),True,renkler["white"])

    pygame.draw.rect(screen,renkler["black"],(height,40,height+30,50))

    screen.blit(textsıra,(height+30, 50))
```

Bu kod, bir oyun döngüsünü temsil eder. **while True** döngüsü, oyunun sürekli olarak çalışmasını sağlar. Oyun döngüsü içinde, mevcut oyuncunun sırasını belirleyen bir koşul bulunur. Eğer mevcut oyuncu sırası saldırıya gelmişse, "saldiri" metnini ekrana yazdırır. Aksi halde, mevcut oyuncunun adını ve rengini ekrana yazdırır. Bu bilgiler, oyunun o anki durumu hakkında kullanıcıya bilgi verir.

Ek olarak, mevcut oyuncunun rengi ve adı, beyaz renkte bir arka plan üzerinde gösterilir. Bunun için **pygame.draw.rect** fonksiyonu kullanılarak arka plan oluşturulur ve **screen.blit** fonksiyonu ile metin ekrana yerleştirilir.



```
                elif event.key == pygame.K_2 and hak<2:

                    klavyeno="2"

                    mauseno=mouse_target()

                    mevcutOyuncu.toplam_alan()

                    mevcutOyuncu.karakter_ekle(klavyeno,mauseno)
```

Bu kod, kullanıcının klavyeden tuşa basmasıyla ilgili bir olayı işler. Eğer kullanıcı "2" tuşuna bastıysa ve belirli bir koşul sağlanıyorsa (hak değeri 2'den küçükse), oyun devam eder. Klavye tuşu "2" olarak atanmıştır (**pygame.K_2**). Ardından **mouse_target()** fonksiyonu çağrılarak fare imlecinin konumu alınır ve **mauseno** değişkenine atanır. Daha sonra mevcut oyuncunun **toplam_alan()** metodu çağrılarak etkin olan alanlar belirlenir ve **karakter_ekle()** metodu çağrılarak bir karakter eklenir.

```
873     mouse_x, mouse_y = pygame.mouse.get_pos()
874
875     mouse_click = pygame.mouse.get_pressed()
876
877     pygame.draw.rect(screen,renkler["black"],(height,20,height+20,20))
878     text = font.render(f"Fare Pozisyonu: ({mouse_x}, {mouse_y})", True, renkler["white"])
879
880     screen.blit(text, (height+30, 20))
881
882     pygame.display.flip()
883
884     clock.tick(15)
```

Bu kod, fare pozisyonunu takip eder ve ekranda bu pozisyonu görüntüler. **pygame.mouse.get_pos()** işlevi fare imlecinin anlık konumunu alır ve **mouse_x** ve **mouse_y** değişkenlerine atar. **pygame.mouse.get_pressed()** işlevi fare düğmelerinin durumunu alır. Kod, fare pozisyonunu ve sol düğmenin durumunu ekrana yazdırır. Son olarak, ekranı günceller (**pygame.display.flip()**) ve oyun saatini ayarlar (**clock.tick(15)**).

4.2 Görev Dağılımı:

- Görev dağılımını ikiye ayrılmıştır. Koddaki fonksiyon ve classlar bölüşüldü. Ödev raporu için de belirli kısımlar bölüşülerek yazılmıştır.

4.3 Karşılaşılan Zorluklar ve Çözüm Yöntemleri:

Uygun programlama dili seçmede zorluklar yaşanmıştır. Algoritmayı kurma ve classların birbiri arasındaki ilişkiyi kurmakta zorluklar yaşanmıştır. Bu zorlukları çözmek amacıyla yapay zekadan destek alınmıştır.

4.4 Proje İsterlerine Göre Eksik Yönler:

Mevcut birliğin üstüne yeni birliğin konuşlandırma özelliği olması gerekirken bu projede olmamasıdır.