

**T.C**

**KOCAELİ SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ BİLGİSAYAR/YAZILIM MÜHENDİSLİĞİ**

**PROJE KONUSU: 2-boyutlu matris formunda bir dünyada iki oyunculu bir savaş oyunu simülasyonu**

**ÖĞRENCİ ADI: Aybüke Türidi Elif Beyza BEYAZ ÖĞRENCİ NUMARASI: 220502005 220502033**

**Github Linkleri**

**Aybüke Türidi :** [**https://github.com/220502005**](https://github.com/220502005) **Elif Beyza Beyaz :** [**https://github.com/beyzabeyaz0**](https://github.com/beyzabeyaz0)

**DERS SORUMLUSU:**

**PROF. DR./DR. ÖĞR. ÜYESİ Nevcihan Duru TARİH: 24.03.2024**

# GİRİŞ

## Projenin amacı

Bu projede temel amaç 2 boyutlu matris formunda bir dünyada savaş simülasyonu gerçekleştirmektir.

* + - Oyunun ilerlemesi için konsol girişleri kullanılmalıdır.
    - Dolu kareler arasındaki savaşları yönetmek için bir saldırı sistemi bulunmalıdır. Bu saldırı sistemi oyunun sonlanması için gereklidir.
    - Oyuncuların üç kez pas geçmesi durumunda veya bir oyuncunun belirli bir alanı kaplaması durumunda oyun sonlanmalıdır.

**Ek kazanma durumu:** Ana kazanma şartlarına ek olarak, yan yana üç kareyi kaplayan oyuncunun kazanmasını sağlayan bir ek kazanma durumu da eklenir. Bu, oyunun heyecanını artırabilir ve farklı stratejilerin kullanılmasına olanak tanır.

# GEREKSİNİM ANALİZİ

## Arayüz gereksinimleri

Bir 2-boyutlu savaş oyunu için kullanıcı arayüzü aşağıdaki özelliklere sahip olmalıdır:

* Oyuncuların oyun tahtasını görmelerini sağlayacak grafiksel bir arayüze sahip olmalıdır. Bu arayüz, 2-boyutlu matris formundaki dünyayı görsel olarak temsil etmelidir.
* Oyun dünyasında, oyuncularını ayırt edebilmesi için her oyuncunun savaşçılarının renkleri ayrı olmalıdır.
* Oyun tahtası, kullanıcıların mevcut durumu anlamalarına yardımcı olacak şekilde gösterilmelidir. Dolu kareler, boş kareler net bir şekilde görünür olmalıdır.
* Kullanıcılar, terminal girişleri veya arayüz üzerinden hamlelerini belirlemek için bir yol sunulmalıdır. Bu, kullanıcıların savaşçılarını hareket ettirmek veya saldırı yapmak gibi eylemleri gerçekleştirmelerine olanak tanımalıdır.
* Kullanıcıların oyunun durumu hakkında bilgi sahibi olmalarını sağlayacak durum göstergeleri olmalıdır. Örneğin hangi oyuncunun sırasının geldiği veya oyunun sona erip ermediği gibi bilgiler gösterilebilir.

oyuncu

Arayüz

* Oyunun sona ermesi durumunda, kazanan oyuncunun bildirildiği bir mesaj veya ekran gösterilmelidir. Bu, kullanıcıların oyunun sonucunu hızlı bir şekilde öğrenmelerini sağlar.
* Arayüz, kullanıcıların oyunu kolayca anlamalarına ve kullanmalarına olanak tanıyan basit ve kullanıcı dostu olmalıdır. Aşırı karmaşık veya karışık bir arayüz, kullanıcı deneyimini olumsuz etkileyebilir.

## Fonksiyonel gereksinimler

* + - Kullanıcı istediği boyutta bir matriste oyunu oynayabilmelidir.
    - Oyuncu, konsol aracılığıyla hamlelerini yapabilmelidir.
    - Oyuncular, savaşçılarını boş kutucuklara yerleştirebilmelidir. Dolu kareler üzerine yerleştirme yapılamamalıdır.
    - Herhangi bir oyuncu, hamle yapamadığı durumda pas geçebilmelidir. Bir oyuncu üç kez pas geçerse, oyunu kaybeder.
    - Oyunun sona erip ermediği sürekli olarak kontrol edilmelidir. Bir oyuncu tüm alanın %60'ını kapladığında veya yan yana üç kareyi kapladığında oyun sona ermeli ve kazanan oyuncu belirlenmelidir.
  1. **Use-Case diyagramı**

|  |  |  |
| --- | --- | --- |
|  | Savaş oyunu |  |
| Savaşçı ve koordinat seç  Matris boyutu seç | | |

# TASARIM

## Mimari tasarım

### Mimari Tasarım:

Oyuncu Sınıfı (Oyuncu):

* + - Oyuncuların temel özelliklerini ve davranışlarını tanımlar.
    - Oyuncuların isimlerini, sahip oldukları savaşçıları ve renklerini içerir.
    - Oyuncuların matris alanına yerleştirilmesini ve savaşçı seçimini yönetir.

Savaşçı Sınıfları (Okcu, Muhafiz, Atli, Saglikci, Topcu):

* + - Farklı savaşçı türlerini temsil eder.
    - Her bir savaşçının adı, canı, saldırı gücü ve rengi gibi özellikleri bulunur.

### Mimari İşleyiş:

Oyuncu sınıfı, oyun alanına oyuncuları yerleştirir ve savaşçılarını yönetir.

Savaşçı sınıfları, savaşçıların özelliklerini tanımlar.

main fonksiyonu, oyunun ana işleyişini sağlar. Sırayla oyunculara hamle yapma fırsatı verir ve oyunun kazananını belirler.

|  |
| --- |
| Oyuncu |
| -name: str  -savascilar: list  -renk: tuple |
| + \_ \_ init\_ \_()  + savasci\_ekleme()  + savasci\_secimi() |

+\_ \_ init \_ \_ ()

-ad: str

-can: int

-renk: tuple

Savasci

+ \_ \_ init \_ \_ ( )

Okcu

+ \_ \_ init \_ \_ ( )

Muhafız

+ \_ \_ init \_ \_ ( )

Atli

+ \_ \_ init \_ \_ ( )

Saglikci

+ \_ \_ init \_ \_ ()

Topçu

## Kullanılacak teknolojiler

* 1. **Veri tabanı tasarımı**

Bu projede veri tabanı kullanılmamıştır.

## Kullanıcı arayüzü tasarımı

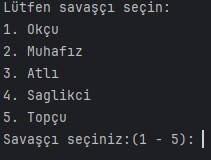
Bu Python kod örneği, Pygame kütüphanesi kullanılarak basit bir oyunu uygulamaktadır. Kullanıcı arayüzü, terminal üzerinden etkileşimli bir şekilde gerçekleştirilmektedir. Kullanıcı, oyunu başlatmak için terminal üzerinde çalıştırdıktan sonra oyunun gerektirdiği adımları takip eder.

Kullanıcı arayüzüyle ilgili bazı önemli noktalar:

* + - Kullanıcı, başlangıçta bir matris boyutu seçer. Bu boyut, oyun alanının satır ve sütun sayısını belirler.



* + - Oyuncular sırasıyla savaşçılarını seçer. Oyuncu 1 savaşçılarını seçtikten sonra, oyuncu 2'nin savaşçıları rastgele seçilir.



* + - Oyuncu, savaşçılarını oyun alanındaki kendi seçtiği koordinatlara yerleştirir. Koordinat girişi yapılırken, oyuncuların pas geçme seçeneği de bulunmaktadır.

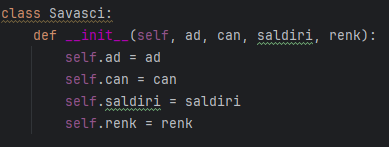


* + - Her oyuncu hamle yaptıktan sonra, oyun alanı güncellenir ve oyuncuların mevcut durumu gösterilir.
    - Oyun alanının belirli bir yüzdesi dolu olduğunda veya belirli bir koşul sağlandığında bir oyuncu kazanır ve oyun sona erer.

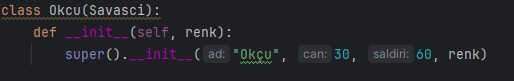
# UYGULAMA

## Kodlanan bileşenlerin açıklamaları

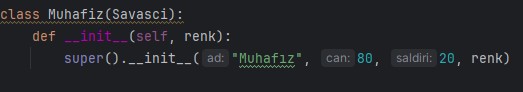
Savaşçı sınıfının oluşturulması ve savaşçının özelliklerinin eklenmesi.

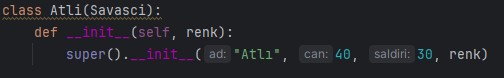


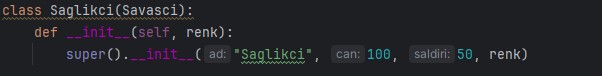
“Okcu” adında bir sınıf oluşturulmuştur ve bu sınıf “Savasci” sınıfından türetilmiştir. (Savasci sınıfının alt sınıfıdır). Yani “Okcu” sınıfı, “Savasci” sınıfının özelliklerini (ad, can, saldırı, renk gibi) miras alır ve üzerine ek özellikler ekleyebilir. Bu yöntem diğer savaşçılar içinde kullanılmıştır.



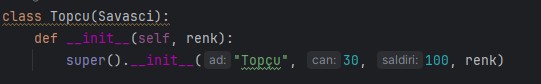
Muhafız Sınıfı:

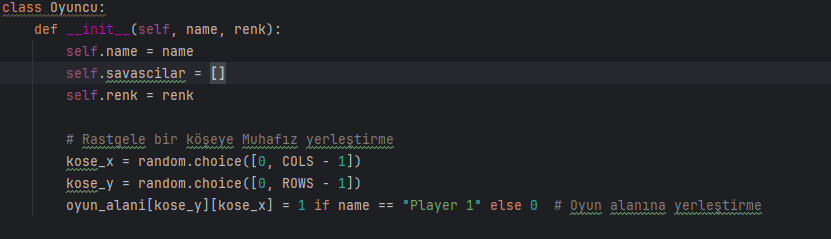


Atlı Sınıfı:

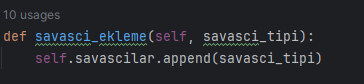
Sağlıkçı Sınıfı:

Topçu Sınıfı:

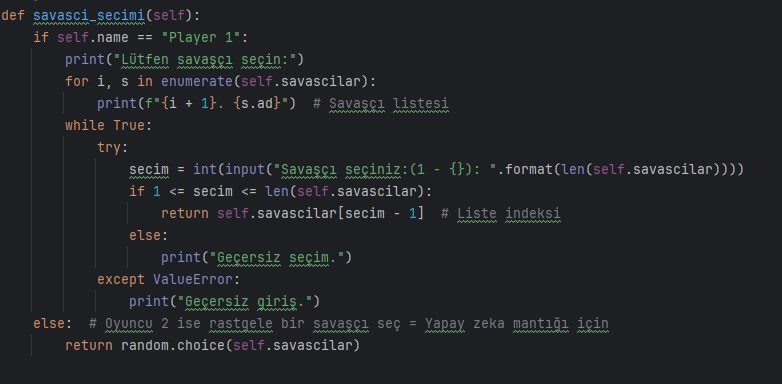


Oyunu sınıfının oluşturulması ve köşelere muhafızların yerleştirilmesi:

Savaşçı eklemek için kullanılacak metodun eklenmesi:

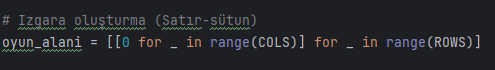


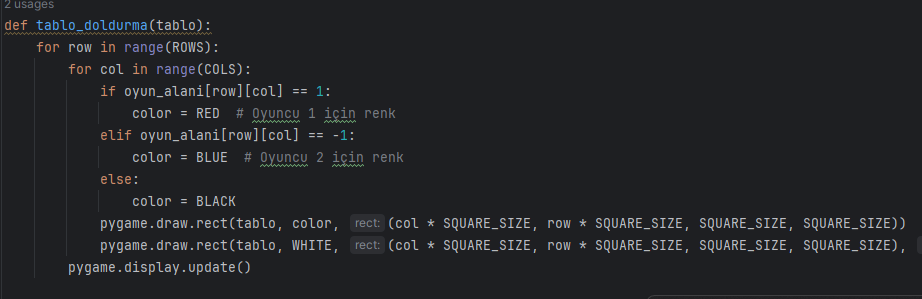
Savaşçı seçimi gerçekleştirmek için kullanılacak metot:



Savaşçı seçimi yapılırken 1.oyuncu menüden istediği savaşçıyı seçer, 2.oyuncu için “random” ile rastgele seçim yapılır.

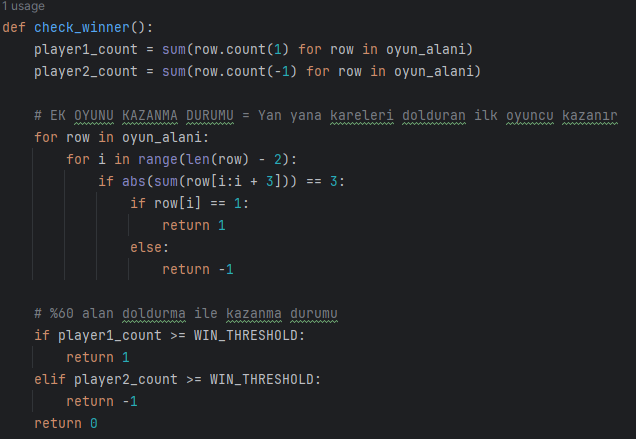
Matrisin kare şeklinde gözükmesi için kullandığımız ızgara, satır-sütun arası çizgiler için kullanılan metot:



Oyuncuların savaşçı yerleştirdikleri karelerin boyanması için kullanılan metot:

1.oyuncu için kırmız 2.oyuncu için mavi renk kullanılır. Boyamak için pygame modülünden “draw” metodu kullanılmıştır.

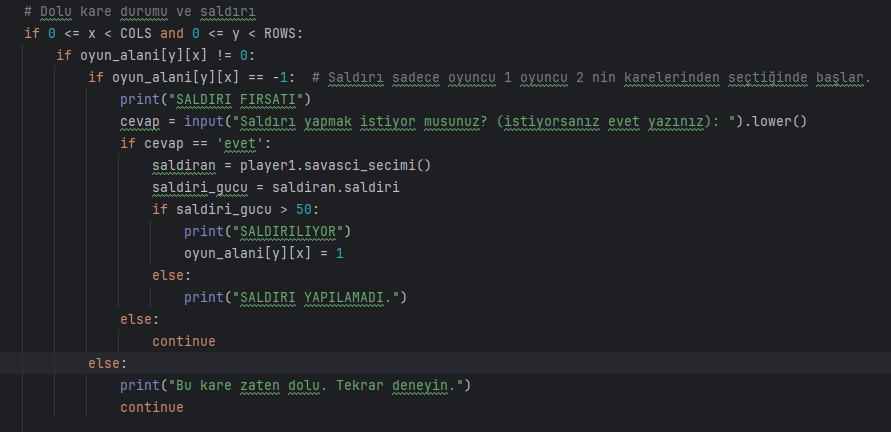
Oyunu hangi oyuncunun kazandığını anlamak için kullanılan metot:



Yan yana 3 tane aynı rengi olan oyuncu kazanır ya da oyun alanının %60’ını alan oyuncu kazanır.

**Main fonksiyonu** içerisinde oyunun işlenişi gerçekleştirilmiş ve bu fonksiyonun çağrılması durumunda oyun başlayacaktır. Aynı zamanda pygame ile oluşturulacak ekranın fonksiyonları da main fonksiyonun içerisinde bulunmaktır.

Aşağıdaki kod parçası, oyuncunun seçtiği karede saldırı yapılıp yapılamayacağını kontrol eder. Eğer kare doluysa ve düşman oyuncuya aitse, oyuncuya saldırı yapma fırsatı verilir. Oyuncu saldırı yapmak isterse, savaşçı seçilir ve saldırı gücü hesaplanır. Eğer saldırı gücü belirli bir eşiği aşıyorsa, saldırı gerçekleşir. Aksi takdirde, saldırı yapılamaz ve oyuncuya yeniden bir kare seçmesi söylenir.



## Görev dağılımı

Kodlama aşaması:

Aybüke Türidi: Oyun sistemi(Kullanıcı seçimleri, oyun ayarları, tablo oluşturma ve düzenleme)

Elif Beyza Beyaz: Sistem gereksinimleri (Sınıf türleri ve özellikleri)

Rapor hazırlanma aşaması:

Aybüke Türidi: Raporun uygulama kısmı

Elif Beyza Beyaz: Diyagram oluşturma, gereksinimler ve tasarım.

## Karşılaşılan zorluklar ve çözüm yöntemleri

### Zorluklar:

* + - Oyuncuların el değişimin gerçekleştirilmesi

### Çözüm Yöntemleri:

* + - Oyuncu el değişimi için “turn = 3 – turn” kod parçası kullanıldı. Oyuncu 1 için turn değeri 1 iken, oyuncu 2 için 2'dir. Bu şekilde el değişimi sağlanır.

## Proje isterlerine göre eksik yönler

* + - Oyuncu sayısı seçimi
    - Menzil durumlarına göre hareket etme
    - Oyuncunu seçtiği savaşçının baş harfi gözükmemesi
    - Savaşmanın eksik olması

# TEST VE DOĞRULAMA

## Yazılımın test süreci

Yazılım için ayrı bir test kodu yazılmadı fakat konsoldan olabilecek durumlar için girişler yapıldı.

## Yazılımın doğrulanması

* Matrise köşelere muhafızların yerleşmesi için test yapıldı bunun sonucunda köşelere muhafızlar yerleşiyor.