T.C.

BANDIRMA ONYEDİ EYLÜL ÜNİVERSİTESİ

MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Proje Ödevi

2D DÖVÜŞ OYUNU

Şule Meşe

191502001

Emiray Saygın

201502034

Adem Türkeş

2111502220

1. Proje Tanımı

- - Fighter Game, bir dövüş oyunudur, oyuncuların birbirleriyle çevrimiçi veya yerel olarak karşı karşıya geldiği rekabetçi bir platform sağlar.
- - Oyuncular, farklı karakterlerin kontrolünü üstlenir ve rakiplerine karşı dövüşerek galip gelmeye çalışır.
- - Oyun, Python programlama dili ve Pygame oyun geliştirme kütüphanesi kullanılarak geliştirilmiştir.

2.Oyun Türü

- Bu oyun, Pygame kütüphanesi kullanılarak geliştirilmiş bir 2D dövüş oyunudur.
- Pygame, Python programlama dili için geliştirilmiş bir oyun geliştirme kütüphanesidir ve 2D grafikler, ses, giriş kontrolleri ve oyun mantığı gibi birçok oyun geliştirme özelliğini sağlar.
- Bu projede, Pygame'in sağladığı araçlar ve özellikler kullanılarak, iki oyuncunun birbirine karşı dövüştüğü, hızlı tempolu ve aksiyon dolu bir 2D dövüş oyunu geliştirilmiştir.
- Oyun, retro tarzı piksel grafikleri ve klasik dövüş oyunu mekanikleri ile oyunculara heyecanlı bir deneyim sunmayı hedefler.

3. Oyunun Hedef Kitlesi

- Bu oyun, dövüş oyunlarını seven ve retro tarzı grafiklerden hoşlanan geniş bir oyuncu kitlesine hitap etmeyi hedefler.
- Oyun, basit kontrolleri ve hızlı tempolu oynanışı ile hem yeni başlayanlar hem de deneyimli oyuncular için uygundur.
- Oyun, klasik dövüş oyunlarının temel unsurlarını içerirken, basit ve erişilebilir mekanikleri ile her yaştan oyuncuya hitap eder.
- Hızlı tempolu ve eğlenceli bir deneyim sunan oyun, oyunculara aksiyonlu bir atmosferde rekabet etme imkanı verir.
- Hedef kitlesi genellikle genç oyuncular ve dövüş oyunlarına ilgi duyan herkes olabilir.
- Basit kontrolleri ve eğlenceli oynanışıyla, hem çocuklar hem de yetişkinler için keyifli bir deneyim sunar.

4. Kullanılan Teknolojiler

Fighter Game'in geliştirilmesinde Python programlama dili ve Pygame oyun geliştirme kütüphanesi kullanılmıştır.

4.1 Python Programlama Dili

- Python, oyun geliştirme için oldukça yaygın bir dil olup, basit sözdizimi ve geniş kütüphane desteğiyle öne çıkar.
- Python, kullanımı kolay, yüksek seviyeli bir programlama dilidir.
- Temiz ve okunabilir sözdizimi sayesinde oyun geliştirme sürecini kolaylaştırır.
- Ayrıca, geniş bir topluluğa ve çevrimiçi kaynaklara sahiptir, bu da geliştiricilere destek sağlar.

4.2. Pygame Kütüphanesi

- Pygame, Python tabanlı bir oyun geliştirme kütüphanesidir.
- Grafiklerin işlenmesi, ses efektlerinin çalınması, oyun kontrollerinin işlenmesi ve diğer birçok oyun geliştirme işlemi için kullanılır.
- Pygame, oyun geliştirme sürecini kolaylaştıran bir dizi modül ve fonksiyon içerir.

5. Oyunun Oynanışı

Oyun, iki oyuncunun karşı karşıya geldiği ve birbirleriyle mücadele ettiği bir dövüş oyunudur. Oyuncular, klavye kontrollerini kullanarak kendi dövüşçülerini yönlendirir ve rakip dövüşçüye saldırır. İşte oyunun nasıl oynanacağına dair ayrıntılı bir rapor:

5.1 Başlangıç Ekranı

Oyun başladığında, oyunculara karşı karşıya gelmeye hazır olduklarını belirtmek için bir geri sayım ekranı görüntülenir. Bu geri sayım, oyunculara dövüşe hazır olmaları için bir süre verir.

5.2 Dövüş Sahnesi

Geri sayımın tamamlanmasının ardından, dövüş sahnesi başlar. Oyuncuların dövüşçüleri, ekranın zıt uçlarında karşılıklı olarak yer alır.

5.3 Dövüş Kontrolleri

Her oyuncu, kendi dövüşçüsünü kontrol etmek için belirli klavye tuşlarını kullanır. Örneğin:Oyuncu 1, A ve D tuşlarıyla sola ve sağa hareket edebilir, W tuşuyla zıplar, R ve T tuşlarıyla saldırır.Oyuncu 2, SOL OK ve SAĞ OK tuşlarıyla sola ve sağa hareket edebilir, YUKARI OK tuşuyla zıplar, ve KP1 ve KP2 tuşlarıyla saldırır.

5.4 Saldırılar

Her oyuncu, rakip dövüşçüye saldırmak için R veya T (Oyuncu 1 için) ve KP1 veya KP2 (Oyuncu 2 için) tuşlarına basar. Bu saldırılar, rakip dövüşçünün sağlık seviyesini azaltır.

5.5 Saldırı Animasyonları
Her saldırı, dövüşçünün animasyonunu tetikler. Bu animasyonlar, dövüşçünün belirli bir hareketi gerçekleştirdiği ve saldırı yaptığı görüntüsünü verir.

5.6 Sağlık Seviyeleri
Her dövüşçü, bir sağlık çubuğuyla temsil edilir. Herhangi bir saldırı, rakip dövüşçünün sağlık seviyesini azaltır. Sağlık seviyesi sıfıra ulaşan bir dövüşçü yenilir.

5.7 Oyunun Kazananı
Bir oyuncunun sağlık seviyesi sıfıra ulaştığında, diğer oyuncu kazanır.Kazanan, bir sonraki raunda geçer ve bir sonraki dövüşe başlar.

5.8 Oyunun Yeniden Başlatılması
Bir raundun sonunda, oyun bir süreliğine durur ve bir kazanan ilan edilir.Kazananın belirlenmesinden sonra, bir sonraki raund için bir geri sayım başlar ve dövüş sahnesi yeniden başlar.

Oyun, dövüş becerileri ve reflekslere dayalıdır. Oyuncular, rakiplerine karşı hızlı ve stratejik hamleler yaparak galip gelmeye çalışır. Her raund, oyuncuların dikkatini ve becerisini test eder, aynı zamanda animasyonlar ve ses efektleriyle etkileyici bir dövüş deneyimi sunar.

6. Proje Süreci

6.1. Analiz ve Planlama
Proje başlamadan önce, oyunun temel özellikleri belirlendi ve gereksinimler analiz edildi. Proje planı oluşturuldu ve iş akışı belirlendi.

6.2 Tasarım
Oyunun tasarımı yapıldı. Grafikler, karakterlerin ve arka planların tasarımı için Photoshop gibi araçlar kullanıldı. Oyun mekaniği ve kullanıcı arayüzü tasarlandı.

6.3 Geliştirme
Python ve Pygame kullanılarak oyunun kodlaması yapıldı. Karakter hareketleri, dövüş mekaniği, ses efektleri ve oyun kontrolleri uygulandı.

6.4 Test ve Hata Ayıklama
Oyunun farklı aşamalarında testler yapılarak hatalar tespit edildi ve düzeltildi. Oyunun performansı ve işlevselliği test edilerek geliştirme süreci yönlendirildi.

6.1 ANALİZ VE PLANLAMA

Fighter Game'in geliştirilme sürecine başlamadan önce, proje ekibi tarafından oyunun temel özellikleri belirlendi ve gereksinimler analiz edildi. Aşağıda, proje analizi ve planlamasına ilişkin detaylar yer almaktadır.

6.1.1 Temel Özelliklerin Belirlenmesi:

- Oyunun ana teması: Dövüş oyunu.
- Oyuncuların kontrol edeceği karakterler: Savaşçı (Warrior) ve Büyücü (Wizard).
- Karakterler arasında çeşitli dövüş hareketlerinin bulunması.
- Temel oyun mekaniği: Oyuncuların birbirleriyle dövüşerek rakiplerinin sağlık puanlarını azaltmak ve galip gelmek.

6.1.2 Gereksinimlerin Analizi:

- Grafiksel Gereksinimler: Oyun için arka planlar, karakterler, animasyonlar ve arayüz elemanları tasarlanmalıdır.
- Oyun Mekaniği Gereksinimleri: Karakter hareketleri, dövüş mekaniği, sağlık puanı sistemi ve oyun içi etkileşimler belirlenmelidir.
- Ses Gereksinimleri: Oyun için arka plan müziği, ses efektleri ve seslendirmeler sağlanmalıdır.
- Kodlama Gereksinimleri: Python ve Pygame kullanılarak oyunun kodlanması gerekmektedir.

6.2 TASARIM

Fighter Game'in tasarım sürecinde, grafiklerin, karakterlerin ve arka planların tasarımı için çeşitli çevrimiçi fotoğraf depolarından bulunan görseller kullanıldı ve oyun mekaniği ile kullanıcı arayüzü tasarlandı.

6.2.1 GRAFİK TASARIMI

- Oyunun arka planları ve diğer görsel öğeleri, çeşitli çevrimiçi fotoğraf depolarından bulunan görseller kullanılarak oluşturuldu.
- Bu görseller, dövüş alanlarını ve oyunun atmosferini yaratmak için seçildi ve düzenlendi.

6.2.2KARAKTERLERİN TASARIMI

- Ana karakterler olan Savaşçı (Warrior) ve Büyücü (Wizard) karakterleri, çevrimiçi fotoğraf depolarından bulunan uygun görseller kullanılarak tasarlandı.
- Bu görseller, karakterlerin genel görünümü, kostümleri ve silahları için referans olarak kullanıldı.
- Karakterlerin animasyonları ve dövüş hareketleri için, çevrimiçi bulunan çeşitli görsel öğeler bir araya getirilerek animasyonlar oluşturuldu.

6.2.3 ARKA PLAN TASARIMI

- Oyunun farklı seviyeleri için arka planlar, çevrimiçi fotoğraf depolarından seçilen uygun görseller kullanılarak tasarlandı.
- Bu görseller, farklı ortamları ve atmosferleri temsil etmek için dikkatlice seçildi ve düzenlendi.
- Arka planlar, oyuncuların dövüş alanlarını keşfederken gerçekçi bir deneyim yaşamalarını sağlamak için özenle seçildi.

6.2.4 KULLANICI ARAYÜZÜ TASARIMI

- Oyunun kullanıcı arayüzü kullanıcı dostu olması temel alınarak tasarlandı. Sağlık çubukları, puan tabloları ve diğer bilgilendirme elemanları pygame modülü ile kodlanarak oluşturuldu.
- Kullanıcı arayüzü, oyuncuların oyun içi bilgileri kolayca görmelerini ve oyunu yönetmelerini sağlayacak şekilde düzenlendi.

6.3 OYUNUN GELİŞTİRİLMESİ

Oyunun geliştirilmesi aşağıdaki aşamalardan oluşmaktadır ;

6.3.1 HAZIRLIK AŞAMASI

- Pygame kütüphanesinin yüklenmesi ve projeye entegrasyonu sağlandı.
- Gerekli dosyaların (resimler, sesler) proje dizinine eklenmesi yapıldı.

6.3.2 OYUN DÖNGÜSÜNÜN KURULMASI

1. Pygame ve Gerekli Modüllerin İmport Edilmesi
    - Bu bölümde pygame ve mixer modülleri import edilir. Bu modüller oyunun çalışması için gerekli olan pencere oluşturma, müzik çalma ve ses efektlerini işleme gibi işlevleri sağlar.

2. Oyun Penceresinin Oluşturulması ve Başlığının Ayarlanması
    - pygame.display.set_mode() fonksiyonu kullanılarak oyun penceresi oluşturulur ve boyutu SCREEN_WIDTH ve SCREEN_HEIGHT değişkenlerinden alınır.

- pygame.display.set_caption() fonksiyonu ile oyun penceresinin başlığı belirlenir.

3. Oyun Değişkenlerinin Tanımlanması:
    - Oyunun değişkenleri (intro_count, last_count_update, score, round_over, ROUND_OVER_COOLDOWN) tanımlanır. Bu değişkenler oyunun durumunu takip etmek için kullanılır.

4. Dövüşçü Değişkenlerinin Tanımlanması
    - WARRIOR_DATA ve WIZARD_DATA gibi dövüşçülerin özelliklerini tanımlayan değişkenler oluşturulur. Bu değişkenler, dövüşçülerin boyutu, ölçekleri ve konumlarını içerir.
    - warrior_sheet ve wizard_sheet gibi dövüşçülerin sprite levhaları yüklenir.

5. Müzik ve Ses Efektlerinin Yüklenmesi
    - Oyun için arka plan müziği ve ses efektleri yüklenir.

6. Arka Plan Görüntüsünün Yüklenmesi
    - Oyunun arka plan resmi bg_image değişkenine yüklenir.

7. Gerekli Animasyon Adımlarının Tanımlanması
    - Her dövüşçü için animasyon adımları (WARRIOR_ANIMATION_STEPS ve WIZARD_ANIMATION_STEPS) belirlenir.

8. Metin Yazdırma ve Arka Plan Çizme Fonksiyonlarının Tanımlanması
    - Metin yazdırma (draw_text) ve arka plan çizme (draw_bg) gibi yardımcı fonksiyonlar tanımlanır. Bu fonksiyonlar, oyun ekranındaki metinleri ve arka planı çizmek için kullanılır.

9. Dövüşçülerin Oluşturulması:
    - fighter_1 ve fighter_2 olmak üzere iki dövüşçü oluşturulur. Her bir dövüşçü için Fighter sınıfı kullanılır.
10. Oyun Döngüsü:
    - Ana oyun döngüsü while run: ile başlar. Bu döngü oyunun çalışmasını sağlar ve bir çıkış koşulu karşılanana kadar devam eder.
    - Her döngüde oyunun durumu güncellenir, dövüşçüler hareket ettirilir, animasyonlar güncellenir, ekran yenilenir ve olaylar işlenir.

11. Ekran Güncellemesi ve Pygame'in Kapatılması:
    - Oyun döngüsü sonlandığında, Pygame penceresi kapatılır ve program sonlanır.

6.3.3 KARAKTER SINIFININ OLUŞTURULMASI

1. Karakter Sınıfı Oluşturulması

Oyunda karakterlerle ilgili değişken ve fonksiyonları bir arada tutmak ve karakter nesnesi oluşturmak amacıyla Fighter.py" dosyası içinde "Fighter" sınıfı oluşturulmuştur. Bu sınıf karakterlerimizin kordinatları,zıplama,koşma,saldırı, can gibi özelliklerini ve metotlarını barındırmaktadır.

2. Karakter Nesnelerinin Oluşturulması

Fighter sınıfında karakter nesnelerinin oluşturulması ve başlatılması için "__init__()" yapıcı fonksiyonu oluşturulmuştur. Bu metot ana sınıfımızda çağırıldığında belirlenen parametrelerle karakterler oluşturulacaktır.

```
def __init__(self, player, x, y, flip, data, sprite_sheet, animation_steps, sound):
  self.player = player
  self.size = data[0]
  self.image_scale = data[1]
  self.offset = data[2]
  self.flip = flip
  self.animation_list = self.load_images(sprite_sheet, animation_steps)
  self.action = 0#0:idle #1:run #2:jump #3:attack1 #4: attack2 #5:hit #6:death
  self.frame_index = 0
  self.image = self.animation_list[self.action][self.frame_index]
  self.update_time = pygame.time.get_ticks()
  self.rect = pygame.Rect((x, y, 80, 180))
  self.vel_y = 0
  self.running = False
  self.jump = False
  self.attacking = False
  self.attack_type = 0
  self.attack_cooldown = 0
  self.attack_sound = sound
  self.hit = False
  self.health = 100
  self.alive = True
```

3. Karakterlerin Ekrana Çizdirilmesi

Fighter nesnelerinin görüntüsünü ekrana çizmek için "draw()" fonksiyonu oluşturulmuştur. Bu fonksiyon, karakterin doğru yönde yansıtılmış görüntüsünü doğru konuma yerleştirir.

```
def draw(self, surface):
  img = pygame.transform.flip(self.image, self.flip, False)
  surface.blit(img, (self.rect.x - (self.offset[0] * self.image_scale), self.rect.y - (self.offset[1] * self.image_scale)))
```

- Karakterin görüntüsünü yansıtmak: pygame.transform.flip fonksiyonu kullanılarak karakterin görüntüsü yatay eksende yansıtılır. Bu, karakterin sağa veya sola dönük olduğunu gösterir.

- Görüntünün ekrana yerleştirilmesi: surface.blit fonksiyonu, karakterin görüntüsünü belirtilen yere (koordinatlar, (self.rect.x - (self.offset[0] * self.image_scale), self.rect.y - (self.offset[1] * self.image_scale))) yükler. Bu koordinatlar, karakterin sol üst köşesinin yerini belirler.

4.Karakterlerin Hareket Etmesi ve Saldırması Kontrolleri

Fighter nesnesinin hareketini kontrol etmek amacıyla "move()" fonksiyonu oluşturulmuştur. Bu fonksiyon, karakterin klavye kontrollerine göre hareket etmesini, ekranda kalmasını, zıplamasını ve saldırı yapmasını sağlar.

4.1 Basılan Tuş bilgilerinin Kaydedilmesi

pygame.key.get_pressed() fonksiyonu, klavyeden basılan tuşları kontrol etmek için kullanılır. Bu fonksiyon, bir liste döndürür ve her tuşun bir durumunu içerir. Her bir tuşun durumu, o tuşa basılıp basılmadığını gösterir. Eğer bir tuşa basılmışsa, ilgili indeksteki değer True olur, aksi halde False olur.

```
#get keypresses
key = pygame.key.get_pressed()
```

4.2 Oyuncu 1 için Hareket ve Saldırma Kontrolü

Bu koddaki bloklar, karakterin hareket etmesi ve saldırması için kontrollerin yapılmasını sağlar. Bu kontroller, oyuncunun klavye girişlerine yanıt vermesini ve karakterin oyuncunun isteğine göre hareket etmesini ve saldırmasını sağlar.

1. **self.attacking == False and self.alive == True and round_over == False:**: Karakterin sadece saldırmadığı, hayatta olduğu ve raundun henüz bitmediği durumda diğer eylemleri gerçekleştirebilmesini sağlar.

2. **if self.player == 1:**: Bu koşul, karakterin oyuncu 1 tarafından kontrol edildiği durumu kontrol eder.

3. Hareket Kontrolleri:
    - **if key[pygame.K_a]:**: Sol ok tuşuna basıldığında karakterin sola hareket etmesini sağlar.
    - **if key[pygame.K_d]:**: Sağ ok tuşuna basıldığında karakterin sağa hareket etmesini sağlar.

- **if key[pygame.K_w] and self.jump == False:**: Yukarı ok tuşuna basıldığında ve karakter zıplama hareketini gerçekleştirmemişse, karakteri zıplatır.
- **if key[pygame.K_r] or key[pygame.K_t]:**: R veya T tuşlarına basıldığında karakterin saldırmasını sağlar. Hangi tuşa basıldığına göre saldırı türünü belirler.

4. Saldırı Kontrolleri:

- **if key[pygame.K_r]**: R tuşuna basıldığında karakterin birinci saldırı tipini gerçekleştirmesini sağlar.
- **if key[pygame.K_t]**: T tuşuna basıldığında karakterin ikinci saldırı tipini gerçekleştirmesini sağlar.

4.3. Oyuncu 2 için Hareket Ve Saldırma Kontrolü

Bu koddaki bloklar, ikinci oyuncunun hareket etmesi ve saldırması için kontrollerin yapılmasını sağlar.

1. **if self.player == 2:**: Bu koşul, karakterin oyuncu 2 tarafından kontrol edildiği durumu kontrol eder.

2. Hareket Kontrolleri:
   - **if key[pygame.K_LEFT]:**: Sol yön tuşuna basıldığında karakterin sola hareket etmesini sağlar.
   - **if key[pygame.K_RIGHT]:**: Sağ yön tuşuna basıldığında karakterin sağa hareket etmesini sağlar.
   - **if key[pygame.K_UP] and self.jump == False:**: Yukarı yön tuşuna basıldığında ve karakter zıplama hareketini gerçekleştirmemişse, karakteri zıplatır.

3. Saldırı Kontrolleri:
   - **if key[pygame.K_KP1]**: Klavyenin sayısal tuş takımında 1 tuşuna basıldığında karakterin birinci saldırı tipini gerçekleştirmesini sağlar.
   - **if key[pygame.K_KP2]**: Klavyenin sayısal tuş takımında 2 tuşuna basıldığında karakterin ikinci saldırı tipini gerçekleştirmesini sağlar.

Bu kontroller, ikinci oyuncunun klavye girişlerine yanıt vermesini ve karakterin oyuncunun isteğine göre hareket etmesini ve saldırmasını sağlar.

```
#check player 2 controls
if self.player == 2:
  #movement
  if key[pygame.K_LEFT]:
    dx = -SPEED
    self.running = True
  if key[pygame.K_RIGHT]:
    dx = SPEED
    self.running = True
  #jump
  if key[pygame.K_UP] and self.jump == False:
    self.vel_y = -30
    self.jump = True
  #attack
  if key[pygame.K_KP1] or key[pygame.K_KP2]:
    self.attack(target)
    #determine which attack type was used
    if key[pygame.K_KP1]:
      self.attack_type = 1
    if key[pygame.K_KP2]:
      self.attack_type = 2
```

4.4. Oyuncu 2 için Hareket Ve Saldırma Kontrolü

Bu koddaki bloklar, ikinci oyuncunun hareket etmesi ve saldırması için kontrollerin yapılmasını sağlar.

1. **if self.player == 2:**: Bu koşul, karakterin oyuncu 2 tarafından kontrol edildiği durumu kontrol eder.

2. Hareket Kontrolleri:
   - **if key[pygame.K_LEFT]:**: Sol yön tuşuna basıldığında karakterin sola hareket etmesini sağlar.
   - **if key[pygame.K_RIGHT]:**: Sağ yön tuşuna basıldığında karakterin sağa hareket etmesini sağlar.
   - **if key[pygame.K_UP] and self.jump == False:**: Yukarı yön tuşuna basıldığında ve karakter zıplama hareketini gerçekleştirmemişse, karakteri zıplatır.

3. Saldırı Kontrolleri:
   - **if key[pygame.K_KP1]**: Klavyenin sayısal tuş takımında 1 tuşuna basıldığında karakterin birinci saldırı tipini gerçekleştirmesini sağlar.

- **if key[pygame.K_KP2]**: Klavyenin sayısal tuş takımında 2 tuşuna basıldığında karakterin ikinci saldırı tipini gerçekleştirmesini sağlar.

Bu kontroller, ikinci oyuncunun klavye girişlerine yanıt vermesini ve karakterin oyuncunun isteğine göre hareket etmesini ve saldırmasını sağlar.

4.5 Oyuncunun Düşüş Hızı Ve Dikey Hareketinin Kontrolü

Bu iki satır, karakterin düşüş hızını hesaplayarak ve dikey konumunu güncelleyerek, yerçekimi etkisini simüle eder. Bu sayede, karakterin ekran boyunca düşmesi ve zıplaması sağlanır.

```
#apply gravity
self.vel_y += GRAVITY
dy += self.vel_y
```

1. **self.vel_y += GRAVITY**: Karakterin düşüş hızına yerçekimi etkisini ekler. **GRAVITY** sabiti, yerçekimi miktarını temsil eder. Her çerçeve yenilendiğinde, karakterin düşüş hızı **GRAVITY** sabiti kadar artar.

2. **dy += self.vel_y**: Karakterin dikey konumunu, her çerçeve için hesaplanan düşüş hızına göre günceller. Bu, karakterin her çerçeve yenilendiğinde yere düşmesini sağlar.

4.6 Oyuncunun Ekran Sınırları İçinde Kalmasının Kontrolü

Bu bölüm, karakterin ekran sınırları içinde kalmasını sağlamak için ekran sınırlarını kontrol eder ve gerektiğinde karakterin konumunu ayarlar. İşlevi şu şekildedir:

```
#ensure player stays on screen
if self.rect.left + dx < 0:
    dx = -self.rect.left
if self.rect.right + dx > screen_width:
    dx = screen_width - self.rect.right
if self.rect.bottom + dy > screen_height - 110:
    self.vel_y = 0
    self.jump = False
    dy = screen_height - 110 - self.rect.bottom
```

1. **if self.rect.left + dx < 0:**: Karakterin sol kenarı ekranın sol kenarından çıktıysa, karakterin x koordinatını ekranda kalmak için en sol pozisyona getirir. Bu durum, karakterin ekranın sol kenarına ulaşması durumunda kontrol edilir.
2. **if self.rect.right + dx > screen_width:**: Karakterin sağ kenarı ekranın sağ kenarından çıktıysa, karakterin x koordinatını ekranda kalmak için en sağ pozisyona getirir. Bu durum, karakterin ekranın sağ kenarına ulaşması durumunda kontrol edilir.
3. **if self.rect.bottom + dy > screen_height - 110:**: Karakterin alt kenarı (alt sınır), ekranın alt kenarından 110 piksel daha fazla bir yükseklikteyse, bu genellikle zeminin altında kalması anlamına gelir. Bu durumda, karakterin düşüş hızını sıfırlar, zıplama durumunu sıfırlar ve karakterin yüksekliğini ayarlar, böylece karakter yere değer

4.7. Oyuncunun Rakibine Dönük Yönde Olmasının Kontrolü

Bu kod parçası, her bir dövüşçünün rakibine karşı doğru yönde dönük olmasını sağlar. İki dövüşçünün birbirine doğru bakması, dövüşün daha gerçekçi ve doğal görünmesini sağlar.

```
#ensure players face each other
if target.rect.centerx > self.rect.centerx:
    self.flip = False
else:
    self.flip = True
```

1. **target.rect.centerx > self.rect.centerx**: Bu ifade, hedefin (rakibin) merkezinin, mevcut dövüşçünün (self) merkezinin sağında mı yoksa solunda mı olduğunu kontrol eder. **target.rect.centerx**, hedefin x koordinatının merkezini temsil eder. **self.rect.centerx** ise mevcut dövüşçünün x koordinatının merkezini temsil eder.

2. **self.flip = False**: Eğer hedefin merkezi mevcut dövüşçünün merkezinin sağında ise (**target.rect.centerx > self.rect.centerx** koşulu doğruysa), **self.flip** değişkeni **False** değerine atanır. Bu, mevcut dövüşçünün sprite'ının yatay olarak simetriksiz bir şekilde çizilmesini sağlar, yani mevcut dövüşçü sağa bakar.

3. **self.flip = True**: Eğer hedefin merkezi mevcut dövüşçünün merkezinin solunda ise (**target.rect.centerx > self.rect.centerx** koşulu yanlışsa), **self.flip** değişkeni **True** değerine atanır. Bu, mevcut dövüşçünün sprite'ının yatay olarak simetriksiz bir şekilde çizilmesini sağlar, yani mevcut dövüşçü sola bakar.

4.8 Oyuncuların Ardışık Saldırılarının Sınırlanması

Bu kod parçası, dövüşçülerin ardışık saldırılarını sınırlamak için kullanılır. Eğer bir dövüşçü bir saldırı gerçekleştirirse, saldırı soğuma süresi devreye girer ve bir sonraki saldırı için belirli bir süre beklemesi gerekecektir. Bu, oyun dengesini sağlamak ve ardışık saldırıları önlemek için önemlidir.

```
#apply attack cooldown
if self.attack_cooldown > 0:
  self.attack_cooldown -= 1
```

1. if self.attack_cooldown > 0:: Bu ifade, dövüşçünün (karakterin) saldırı soğuma süresinin sıfırdan büyük olup olmadığını kontrol eder. Eğer soğuma süresi sıfırdan büyükse, bu dövüşçünün henüz bir sonraki saldırıyı yapabilecek kadar süresinin olmadığını gösterir.

2. self.attack_cooldown -= 1: Bu satır, saldırı soğuma süresini bir azaltır. Her oyun döngüsünde (frame) veya zaman biriminde bu satır çalıştırıldığında, dövüşçünün saldırı soğuma süresi bir azalır. Yani, her çağrıda, dövüşçü bir saldırıyı bir adım daha yaklaşır.

4.8 Oyuncuların Pozisyonunun Güncellenmesi

Bu kod parçası, dövüş oyununda oyuncunun (dövüşçünün) konumunu güncellemek için kullanılır. Oyuncu hareket ettiğinde veya bir saldırı gerçekleştirdiğinde, bu satırlar çağrılır ve dövüşçünün pozisyonu güncellenir

```
#update player position
self.rect.x += dx
self.rect.y += dy
```

1. **self.rect.x += dx**: Bu satır, dövüşçünün x koordinatının güncellenmesini sağlar. **self.rect** bir dikdörtgen nesnesini temsil eder ve dövüşçünün konumunu ve boyutunu belirler. **dx** değişkeni, dövüşçünün x yönündeki hareket miktarını temsil eder. Bu satır, dövüşçünün x koordinatını **dx** kadar hareket ettirir.

2. **self.rect.y += dy**: Bu satır, dövüşçünün y koordinatının güncellenmesini sağlar. **self.rect** nesnesi dövüşçünün dikdörtgen alanını temsil eder ve **dy** değişkeni, dövüşçünün yönündeki hareket miktarını belirtir. Bu satır, dövüşçünün y koordinatını **dy** kadar hareket ettirir.

5. Oyuncunun Saldırı Yapması ve Rakibine Hasar Vermesi İşlemi

Bu kod parçası, bir dövüşçünün saldırı yapmasını ve hedefe zarar vermesini sağlar. Bu amaçla **attack()** adında bir fonksiyon oluşturulmuştur. Bu fonksiyon, dövüşçünün saldırı yapmasını ve hedefe zarar vermesini yönetir. Fonksiyon, bir hedef parametresi alır, bu da saldırının hedefini temsil eder.

```
def attack(self, target):
  if self.attack_cooldown == 0:
    #execute attack
    self.attacking = True
    self.attack_sound.play()
    attacking_rect = pygame.Rect(self.rect.centerx - (2 * self.rect.width * self.flip), self.rect.y, 2 * self.rect.width, self.rect.height)
    if attacking_rect.colliderect(target.rect):
      target.health -= 10
      target.hit = True
```

1. **def attack(self, target):**: Bu satır, **attack()** adında bir fonksiyon tanımlar. Bu fonksiyon, dövüşçünün saldırı yapmasını ve hedefe zarar vermesini yönetir. Fonksiyon, bir hedef parametresi alır, bu da saldırının hedefini temsil eder.

2. **if self.attack_cooldown == 0:**: Bu ifade, dövüşçünün saldırı soğuma süresinin sıfır olup olmadığını kontrol eder. Eğer saldırı soğuma süresi sıfırsa, dövüşçü saldırı yapmaya hazırdır.

3. **self.attacking = True**: Bu satır, dövüşçünün saldırı modunda olduğunu belirtir. Yani, dövüşçü saldırı yapmak üzere aktif durumdadır.

4. **self.attack_sound.play()**: Bu satır, saldırı sesini çalar. Oyun motorundan çekilen bir ses çalma fonksiyonunu çağırır.

5. **attacking_rect = pygame.Rect(self.rect.centerx - (2 * self.rect.width * self.flip), self.rect.y, 2 * self.rect.width, self.rect.height)**: Bu satır, saldırının etkileşime gireceği dikdörtgen alanını temsil eden bir **pygame.Rect** nesnesi oluşturur. Bu dikdörtgen, saldırıyı temsil eder ve saldırının çarpışma alanını belirler. **self.flip** değişkeni, dövüşçünün yüzünün hangi yöne dönük olduğunu belirler.

6. **if attacking_rect.colliderect(target.rect):**: Bu ifade, dövüşçünün saldırı alanının hedefle çarpışıp çarpışmadığını kontrol eder. Eğer saldırı alanı hedefle çarpışıyorsa, saldırı hedefe isabet etmiştir.

7. **target.health -= 10**: Bu satır, hedefin sağlığını 10 puan azaltır. Yani, saldırı hedefe zarar verir.
8. **target.hit = True**: Bu satır, hedefin vurulduğunu belirtir. Bu, hedefin bir sonraki çizim döngüsünde (frame) vurulduğunu göstermek için kullanılır.

6.Oyuncunun Animasyonlarının Yönetilmesi

Dövüşçünün animasyonlarını güncellemek ve oyun durumuna bağlı olarak farklı animasyonları oynatmak için **update()** fonksiyonu yazılmıştır. Fonksiyon 2 bölümde özetlenmiştir. Şimdi bu bölümleri açıklarsak ;

6.1 Dövüşçü Durumlarına Göre Animasyonların Atanması

Bu kısım, dövüşçünün durumuna göre hangi animasyonun oynatılacağını belirler. Eğer dövüşçü ölmüşse, vurulmuşsa, saldırı yapmakta ise, zıplamakta ise, koşmakta ise veya hiçbir eylem yapmıyorsa ilgili animasyon çağrılır.

```
#check what action the player is performing
if self.health <= 0:
    self.health = 0
    self.alive = False
    self.update_action(6)#6:death
elif self.hit == True:
    self.update_action(5)#5:hit
elif self.attacking == True:
    if self.attack_type == 1:
        self.update_action(3)#3:attack1
    elif self.attack_type == 2:
        self.update_action(4)#4:attack2
elif self.jump == True:
    self.update_action(2)#2:jump
elif self.running == True:
    self.update_action(1)#1:run
else:
    self.update_action(0)#0:idle
```

6.2 Dövüşçü Animasyonunun Güncellenmesi

Bu kısım, animasyonun her bir çerçevesini günceller. Belirli bir zaman aralığında animasyonun ilerlemesini sağlar ve animasyon tamamlandığında gerekli işlemleri gerçekleştirir.

```
animation_cooldown = 50
#update image
self.image = self.animation_list[self.action][self.frame_index]
#check if enough time has passed since the last update
if pygame.time.get_ticks() - self.update_time > animation_cooldown:
  self.frame_index += 1
  self.update_time = pygame.time.get_ticks()
#check if the animation has finished
if self.frame_index >= len(self.animation_list[self.action]):
  #if the player is dead then end the animation
  if self.alive == False:
    self.frame_index = len(self.animation_list[self.action]) - 1
  else:
    self.frame_index = 0
    #check if an attack was executed
    if self.action == 3 or self.action == 4:
      self.attacking = False
      self.attack_cooldown = 20
    #check if damage was taken
    if self.action == 5:
      self.hit = False
      #if the player was in the middle of an attack, then the attack is stopped
      self.attacking = False
      self.attack_cooldown = 20
```

1.animation_cooldown = 50: **animation_cooldown** değişkeni, her bir animasyon çerçevesi arasında geçen zaman aralığını belirtir. Bu değer, animasyonun ne kadar hızlı oynatılacağını kontrol eder.

2.self.image = self.animation_list[self.action][self.frame_index]: **self.image** değişkeni, dövüşçünün görüntüsünü, mevcut animasyon ve animasyon çerçevesine göre günceller. Bu, dövüşçünün ekranda gösterilmesini sağlar.

3.if pygame.time.get_ticks() - self.update_time > animation_cooldown:: Bu ifade, son animasyon güncellemesinden bu yana geçen zamanın, **animation_cooldown** değerinden büyük olup olmadığını kontrol eder. Eğer geçen zaman, **animation_cooldown** değerinden büyükse, yeni bir animasyon çerçevesine geçilir.

4.self.frame_index += 1: **self.frame_index** değişkeni, mevcut animasyon çerçevesini bir artırır, böylece bir sonraki çerçeve gösterilir.

5.self.update_time = pygame.time.get_ticks(): **self.update_time** değişkeni, son animasyon güncellemesinin zamanını günceller. Böylece, bir sonraki güncelleme için geçen zaman hesaplanabilir.

6.if self.frame_index >= len(self.animation_list[self.action]):: Bu ifade, mevcut animasyon çerçevesinin son çerçeve olup olmadığını kontrol eder. Eğer son çerçeveye ulaşıldıysa, gerekli kontroller yapılır.

7.if self.alive == False:: Eğer dövüşçü öldüyse, animasyon tamamlandıktan sonra son çerçeve sabitlenir ve bu animasyon artık oynatılmaz.

8.self.frame_index = len(self.animation_list[self.action]) - 1: Eğer dövüşçü öldüyse ve animasyon tamamlandıysa, son çerçeve olarak son animasyon çerçevesi atanır.

9.else:: Eğer dövüşçü hala hayattaysa, animasyon tekrar başa sarılır ve döngü baştan başlar.

10.if self.action == 3 or self.action == 4:: Eğer dövüşçü saldırı animasyonunu oynatıyorsa, saldırının tamamlanması için belirli bir süre beklenir.

11.self.attacking = False: Saldırı animasyonu tamamlandığında, **self.attacking** değişkeni **False** olarak ayarlanır.

12.self.attack_cooldown = 20: Saldırı animasyonunun tamamlanmasının ardından bir saldırı bekleme süresi başlatılır. Bu, ardışık saldırıların arasındaki zamanı belirler.

13.if self.action == 5:: Eğer dövüşçü vurulduysa, vuruş animasyonunun tamamlanması için belirli bir süre beklenir.

14.self.hit = False: Vuruş animasyonu tamamlandığında, **self.hit** değişkeni **False** olarak ayarlanır.

15.self.attacking = False: Eğer dövüşçü bir saldırı animasyonu içindeyse ve vurulduysa, saldırı animasyonu durdurulur.

16.self.attack_cooldown = 20: Vuruş animasyonunun tamamlanmasının ardından bir saldırı bekleme süresi başlatılır. Bu, ardışık saldırıların arasındaki zamanı belirler.

7. Oyuncunun Animasyonunun Güncellenmesi

Dövüşçünün animasyonunu güncellemek için **update_action()** fonksiyonu yazılmıştır. . Eğer dövüşçünün yeni bir animasyona geçmesi gerekiyorsa, bu fonksiyon çağrılır ve animasyon güncellenir. Bu, dövüş oyunlarında dövüşçünün durumunu ve davranışını görsel olarak temsil etmek için kullanılır.

```
def update_action(self, new_action):
  #check if the new action is different to the previous one
  if new_action != self.action:
    self.action = new_action
    #update the animation settings
    self.frame_index = 0
    self.update_time = pygame.time.get_ticks()
```

1. **def update_action(self, new_action):**: Bu satır, **update_action()** adında bir fonksiyon tanımlar. Bu fonksiyon, dövüşçünün mevcut animasyonunu günceller. Fonksiyon, bir **new_action** parametresi alır, bu da güncellenecek yeni animasyonun numarasını temsil eder.

2. **if new_action != self.action:**: Bu ifade, yeni animasyonun, dövüşçünün mevcut animasyonundan farklı olup olmadığını kontrol eder. Eğer yeni animasyon, mevcut animasyondan farklıysa, animasyon güncellenir.

3. **self.action = new_action**: Bu satır, dövüşçünün mevcut animasyonunu günceller. Yani, **new_action** değeri, dövüşçünün yeni animasyonu olur.

4. **self.frame_index = 0**: Bu satır, dövüşçünün animasyonunda kullanılan kare indeksini sıfırlar. Bu, dövüşçünün animasyonunun başlangıcına dönmesini sağlar.

5. **self.update_time = pygame.time.get_ticks()**: Bu satır, animasyon güncelleme zamanını günceller. **pygame.time.get_ticks()** fonksiyonu, oyunun başlangıcından bu yana geçen süreyi milisaniye cinsinden döndürür. Bu, animasyonların zamansal olarak senkronize olmasını sağlar.

8.Oyuncu Saldırı Animasyonlarının Yüklenmesi

Oyunda dövüşçünün animasyonlarını yüklemek ve listelemek için **load_images()** adında bir fonksiyon yazılmıştır. Bu fonksiyon, dövüşçünün animasyonlarını yüklemek için **sprite_sheet** ve **animation_steps** gibi bilgileri alır. **sprite_sheet**, dövüşçünün animasyonlarının bulunduğu sprite levhasını temsil ederken, **animation_steps** her bir animasyonun adım sayısını içeren bir liste veya dizi olarak sağlanır. Bu fonksiyon, sprite levhasından görüntüleri çıkarır, her bir çerçeveyi ölçekler ve bunları uygun bir şekilde animasyon listesine yerleştirir.

1. **def load_images(self, sprite_sheet, animation_steps):**: Bu satır, **load_images** adında bir fonksiyon tanımlar. Bu fonksiyon, sprite levhasından görüntülerin çıkarılması ve animasyon listesinin oluşturulmasından sorumludur. **sprite_sheet** parametresi, sprite levhasını, **animation_steps** parametresi ise her bir animasyonun adım sayısını içeren bir liste veya dizi olarak alır.
2. **animation_list = []**: Boş bir liste oluşturulur. Bu liste, tüm animasyonlar için alt listeler içerecek.
3. **for y, animation in enumerate(animation_steps):**: **animation_steps** üzerinde bir **for** döngüsü başlatılır. Bu döngü, animasyonların adım sayısını ve indeksini alır.
4. **temp_img_list = []**: Her animasyon için boş bir alt liste oluşturulur. Bu alt liste, animasyonun her bir çerçevesinin görüntülerini içerecek.
5. **for x in range(animation):**: Her animasyonun adım sayısı kadar bir **for** döngüsü başlatılır. Bu döngü, her bir animasyonun çerçevelerini alır.
6. **temp_img = sprite_sheet.subsurface(x * self.size, y * self.size, self.size, self.size)**: **sprite_sheet** üzerindeki belirli bir bölgeden (çerçeveden) görüntü alınır. **subsurface** yöntemi, belirtilen boyutlarda bir alt yüzey oluşturur ve bu alt yüzey, orijinal sprite levhasının belirli bir bölgesine işaret eder.
7. **temp_img_list.append(pygame.transform.scale(temp_img, (self.size * self.image_scale, self.size * self.image_scale)))**: Alınan görüntü, **pygame.transform.scale** yöntemiyle belirli bir ölçekte yeniden boyutlandırılır ve ardından **temp_img_list** listesine eklenir.
8. **animation_list.append(temp_img_list)**: Her animasyonun alt listesi, ana animasyon listesine (**animation_list**) eklenir.
9. **return animation_list**: Oluşturulan animasyon listesi döndürülür. Bu liste, her bir animasyonun alt listesini içerir ve her bir alt liste, o animasyonun çerçevelerini içerir

9. Oyuncular Tarafından Bir Tuşa Basınca Başlatılması

oyunun başlangıç ekranını çizen ve kullanıcıya oyunu başlatmak için bir tuşa basması gerektiğini bildiren bir fonksiyonu ve başlangıç ekranının gösterilip gösterilmediğini belirten bir değişkeni tanımlar

10. Kalan Zamanın Ekrana Yansıtılması

Pygame kullanarak oyunun süresini ekranda geri sayım şeklinde gösteren bir zamanlayıcı oluşturur.

11.Süre Sıfırlandığında Başlangıç Ekranına Dönülmesi

Bu kod parçası, Pygame kullanarak oyunun süresinin bitmesi durumunda bir "Game Over" mesajı görüntüleyen ve ardından oyunu başlatan bir işlemi gerçekleştirir.

12.Dövüşçünün Kalan Canının Renk Olarak Gösterimi

Bu kod parçası, savaşçıların sağlık çubuklarını çizmek için kullanılır. Sağlık çubuklarının rengi, savaşçının mevcut sağlık durumuna göre değişir