BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

OYUN PROGRAMLAMA DERSİ PROJESİ

GİZEMNUR TETİK - 191522009

2023-2024 BAHAR DÖNEMİ

1 GENEL BİLGİLER

1.1 Oyun Programlama Nedir?

Oyun programlama, video oyunlarının tasarlanması ve geliştirilmesinde kullanılan bir yazılım disiplinidir. Oyun programlama, bilgisayar oyunları veya mobil oyunlar gibi çeşitli oyun türlerini içerebilir. Bu süreçte, geliştiriciler, oyun dünyasını oluşturmak, oyun mekaniğini uygulamak ve oyuncuların etkileşimde bulunabileceği bir deneyim sunmak için yazılım kodları kullanır. Oyun programlaması, grafiklerin, ses efektlerinin, yapay zekanın, kullanıcı arabirimlerinin ve diğer oyun bileşenlerinin oluşturulmasını da içerebilir. Programcılar, oyun motorları veya oyun geliştirme araçları gibi yardımcı teknolojileri kullanarak, oyunun tüm yönlerini kontrol etmek ve gerçekleştirmek için programlama dilleri ve algoritmaları kullanır. Oyun programlaması, yaratıcı bir süreç olarak kabul edilirken, aynı zamanda teknik zorluklar ve optimizasyon gereksinimleri de içerir. Oyun programcıları, eğlenceli ve sürükleyici oyun deneyimleri sağlamak için sanat ve bilimi bir araya getirerek, oyun endüstrisine yenilikçi ve ilgi çekici oyunlar sunmada önemli bir rol oynarlar.

1.2 Oyun Programlama Tarihçesi

Oyun programlama tarihi, bilgisayar ve video oyunlarının gelişimiyle birlikte şekillenmiştir. İlk oyunlar basit grafikler ve sınırlı işlevsellikle başladı. 1950'lerdeki bilgisayarlar, oyunları çalıştırmak için yeterli güce sahip değildi ve genellikle basit matematiksel problemlerin çözümüne odaklanıyordu. 1960'ların sonlarında ve 1970'lerin başlarında, arcade makineleri ve ev bilgisayarları popülerlik kazandı.

Bu dönemde Spacewar!, Pong ve Atari gibi oyunlar, oyun programlamasının temel taşları oldu. Oyunlar genellikle donanımın sınırlamalarına uygun olarak programlanıyordu ve görsel ve işitsel öğeler oldukça basitti.

Şekil 1: *PDP-1 bilgisayarında "Spacewar!" isimli video oyunu*

1980'ler ve 1990'lar, oyun programlamasının büyük bir ilerleme kaydettiği dönemlerdi. Ev bilgisayarları, video oyun konsolları ve kişisel bilgisayarlar oyun endüstrisinin büyümesini tetikledi. Bu dönemde Super Mario Bros., Pac-Man, The Legend of Zelda gibi ikonik oyunlar ortaya çıktı.

Şekil 2: *"Super Mario Bros" oyunundan bir ekran görüntüsü*

2000'ler ve sonrasında ise oyun programlaması büyük bir devrim geçirdi. Oyun motorları ve geliştirme araçları daha gelişmiş hale geldi.

Unreal Engine ve Unity gibi oyun motorları, geliştiricilere güçlü araçlar ve hızlı prototipleme imkanı sağladı. Yapay zeka, daha sofistike ve gerçekçi oyun karakterleri ve düşmanlarının geliştirilmesinde önemli bir rol oynamaya başladı.

Şekil 3: *Unity oyun motoru logosu*

Günümüzde oyun programlaması, yüksek performanslı grafikler, karmaşık fizik simülasyonları, etkileyici yapay zeka ve çok oyunculu oyun deneyimleri gibi birçok gelişmiş özelliği içeren büyük ve karmaşık projeleri kapsamaktadır. Sanal gerçeklik, artırılmış gerçeklik ve bulut tabanlı oyunlar gibi yeni teknolojiler, oyun programlamasının geleceğini şekillendiren alanlardır.

Oyun programlama tarihçesi, teknolojik ilerlemeler ve yaratıcı yeniliklerle birlikte sürekli olarak evrim geçirmektedir. Bugün, oyun programlaması büyük bir endüstri haline gelmiş ve oyunlar milyonlarca oyuncuya ulaşan popüler ürünler haline gelmiştir.

1.3 Oyun Programlamada Kullanılan Teknolojiler

Oyun programlamasında kullanılan teknolojiler, oyunların geliştirilmesi ve işlevsel bir şekilde çalışması için önemlidir. Bu teknolojiler oyun türüne, platformuna ve geliştirici tercihlerine göre değişiklik gösterebilir. Aşağıda bahsedilen teknolojiler, genel olarak oyun programlamasında yaygın olarak kullanılan ve temel öneme sahip olanlardır.

1.3.1 Oyun Motorları

Oyun motorları, oyunların geliştirilmesinde temel yapıları sağlayan yazılımlardır. Popüler oyun motorları arasında Unity, Unreal Engine ve CryEngine bulunur.

Oyun motorları, grafiklerin, fizik simülasyonunun, sesin, animasyonların ve kullanıcı girişinin yönetimi gibi temel işlevleri sunar. Bu motorlar, oyun geliştirme araçları ve programlama dilleriyle entegre çalışarak geliştiricilere kolaylık sağlar.

Şekil 4: Unreal Engine oyun motoru logosu

1.3.2 Programlama Dilleri

Oyun programlamasında birçok farklı programlama dili kullanılır. C++ ve C# gibi diller, performans ve verimlilik sağlama açısından tercih edilen dillerdir. Python, JavaScript, Lua gibi diller de oyun programlamasında kullanılan diğer seçenekler arasında yer alır. Bu diller, oyunun belirli bölümlerini veya oyun senaryosunu yönetmek için kullanılabilir.

VOID

Yukarıda oyun programlamanın ne olduğuna ve tarihçesine kısa bir göz attık. Dersin içeriğini kavradık ve şimdi tasarımını ve içeriğini kendimizin oluşturduğu arcade tür oyunumuzdan bahsedelim. Öncelikle arcade oyun nedir açıklayalım. Arcade oyun ; hızlı tempolu oynanışı ve retro tarzı grafikleriyle dikkat çeker. Oyuncular, labirent benzeri seviyelerde ilerlerken engelleri aşmak ve puanları artırmak için hızla hareket ederler. Tasarladığımız oyun da tam bu şartlara uyum sağlıyor.

Arcade tarzı bir oyun seçmemizin sebebi ise bu tarz oyunlar, basit ve eğlenceli mekaniklere sahip olmalarıyla bilinir ve birçok oyuncu için nostaljik bir değere sahiptir. Yeni çıkan grafik tasarımların kusursuz ve 3D olduğu oyunların yanında nostaljik ve retro oyun isteyen de bir kitle vardır.

İSİM TERCİHİ

Oyunumuzda isim tercihine oldukça önem verdik. Oyunda yaşadığımız aksaklıklardan yola çıkarak ve akılda kalıcı olmasına özen göstererek oyunumuzun isminin ‘ Void ‘ olmasını kararlaştırdık.

VOID OYUNU İÇERİĞİ VE KODLARI

Raporumuzda kodların yer almasını istedik ve her kod bloğunun ne anlama geldiğini, neden kullanıldığını açıkça paylaştık.

EDIT_LEVEL .PY

Oyunumuz iki kısımdan oluşuyor. Oyunumuzun ilk kısmı edit_level.py kısmından oluşuyor ve bu kısımda main.py kısmını çalıştırdığımızda oynamaya hazır hale gelmesini sağlayacağımız level’ları hazırlıyoruz.

Klavyemizle level seviyelerini arttırıp her level için kendimiz bir oyun tasarlayacağız ve main kısmında oynayabileceğiz.

Kısıtlı zaman sebebiyle tasarladığımız bu oyunu ilerleyen süreçlerde yapay zekanın kendi tasarlayacağı level’larla da gerçekleştirmeyi umuyoruz. Şimdi edit_level.py kısmının kodları ve açıklamalarını inceleyelim;

```
import pygame
import pickle
from os import path
```

Bu kısımda, Pygame ve pickle kütüphaneleri import ediliyor ve bir dosya yolu belirlemek için os modülünden path fonksiyonu kullanılıyor.

1. **pygame**: Pygame, Python programlama dilinde oyun geliştirmek için kullanılan güçlü bir kütüphanedir. Oyun penceresi oluşturmayı, grafikler çizmeyi, fare ve klavye gibi girişleri işlemeyi sağlar. Ayrıca oyun fiziği, ses efektleri ve çok daha fazlası için kullanılabilir.

2. **pickle**: Python'da nesnelerin seri hale getirilmesi ve deseri hale getirilmesi için kullanılan bir modüldür. pickle modülü, verileri bir dosyaya veya ağ üzerinden aktarırken

kullanışlıdır. Bu, özellikle veri yapılarını korumak ve uygulama durumunu saklamak istediğinizde kullanışlıdır.

3. **os.path**: Bu modül, dosya ve dizin işlemleri için kullanılır. path fonksiyonu, belirli bir dosyanın var olup olmadığını kontrol etmek, dosya adını almak, dosya uzantısını kontrol etmek ve benzeri işlemleri gerçekleştirmek için kullanılır. Bu, dosya işlemlerinin yönetilmesini ve işletim sistemine bağımlılığı azaltmayı sağlar.

```
pygame.init()

clock = pygame.time.Clock()
fps = 120 #ekranın her saniyede kaç kez güncellenmesi

#game window
tile_size = 15
cols = 40
margin = 100
screen_width = tile_size * cols  #yaygın oyunlarda
screen_height = (tile_size * cols) + margin  #yaygın oyunlarda

screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption('Level Editor')
```

Bu Python kodu, Pygame kütüphanesini kullanarak bir oyun düzenleyici arayüzü oluşturuyor. Oyunun FPS (Frame Per Second - Saniyedeki Kare Sayısı) değeri 120 olarak ayarlanmış ve ekran boyutu belirlenmiş. Ardından, belirtilen boyutlarda bir oyun penceresi oluşturuluyor ve bu pencerenin başlığı "Level Editor" olarak belirleniyor.

```
#load images
bg_img = pygame.image.load("C:/Masaüstü\OYUNPROJE_RECREA\oyunprojeödevi\
bg_img = pygame.transform.scale(bg_img, size: (screen_width, screen_heigh
dirt_img = pygame.image.load("C:/Masaüstü\OYUNPROJE_RECREA\oyunprojeödev
blob_img = pygame.image.load('img/shooterOpen.png')
lava_img = pygame.image.load('img/hiddenSpikes.png')
coin_img = pygame.image.load('img/coin.png')
exit_img = pygame.image.load('img/goal.png')
save_img = pygame.image.load("C:/Masaüstü\OYUNPROJE_RECREA\oyunprojeödev
load_img = pygame.image.load("C:/Masaüstü\OYUNPROJE_RECREA\oyunprojeödev
```

Bu bölümde, oyun için gerekli olan görselleri yüklüyoruz. İlgili görüntüler dosya yollarından yüklenir ve ekrana uygun boyutlara dönüştürülür. Özetle, oyunun arka planı, zemin taşı, düşman karakter, lav, para, çıkış ve kaydetme/yükleme düğmeleri için görüntüler yüklenir.

```
#define game variables
clicked = False  # mouse click false iken level 1 ' dan başlamak.
level = 1

#define colours
white = (255, 255, 255) # DEĞİŞTİRLEBİLİR.......!
pinky = (246, 155, 209)
blackblue=(0, 0, 135)

font = pygame.font.SysFont( name: 'Futura',  size: 24)
```

Bu bölüm, oyunun temel değişkenlerini ve renklerini tanımlar. Oyunun hangi seviyeden başlayacağını belirten clicked ve level gibi değişkenler tanımlanır. Ayrıca, oyun arayüzünde kullanılacak olan beyaz ve yeşil gibi renkler belirlenir. Son olarak, metinleri

görüntülemek için kullanılacak yazı tipi ve boyutu font değişkeniyle belirlenir. Bu adımlar, oyunun başlangıç ayarlarını ve görsel özelliklerini tanımlar.

```
#create empty tile list
world_data = []  # oyun dünyasını create etmek
for row in range(41):
    r = [0] * 41
    world_data.append(r)

#create boundary
for tile in range(0, 40):
    world_data[39][tile] = 1   # alt sınır
    world_data[0][tile] = 1  # üst sınır
    world_data[tile][0] = 1  # sol satır
    world_data[tile][39] = 1  # sağ satır
```

Bu bölümde, oyun dünyasını temsil etmek için bir liste olan world_data oluşturuluyor. Bu liste, oyun alanındaki farklı karelerin durumunu tutar. Öncelikle, 41 satır ve 41 sütun olacak şekilde bir dizi oluşturulur ve her bir öğe başlangıçta 0 olarak ayarlanır, bu da boş bir kareyi temsil eder.

Ardından, sınırları belirlemek için bir döngü kullanılarak oyun dünyasının sınırları oluşturulur. world_data listesinde alt, üst, sol ve sağ sınırlara değer atayarak bu sınırlar belirlenir. Örneğin, en alt satır (39. satır) tamamen 1 olarak ayarlanırken, diğer satırların sadece kenarlarında birer birim 1 olarak atanır. Bu adımlar, oyun dünyasının çevresini belirleyerek oyuncunun oyun alanını terk etmesini engeller.

```
#function for outputting text onto the screen
def draw_text(text, font, text_col, x, y):   # oyun içi
    img = font.render(text, True, text_col)
    screen.blit(img,  dest: (x, y))
```

Bu kod bloğu, ekrana metin çıktısı vermek için bir işlev tanımlar. draw_text işlevi, belirli bir metni belirtilen bir yazı tipi, renk ve konumda ekrana çizer. Bu, kullanıcıya oyun içinde bilgi veya talimat sağlamak için kullanılır. İşlev, metni ekrana çizmek için font.render() yöntemini kullanır ve bu metni belirtilen konuma ((x, y)) yerleştirir. Bu işlev, oyun içinde metin tabanlı bilgileri görüntülemek için kullanılır.

```
def draw_grid(): #grid çizer
    for c in range(0,40):
        #vertical lines
        pygame.draw.line(screen, blackblue,  start_pos: (c * tile_size, 0),
        #horizontal lines
        pygame.draw.line(screen, blackblue,  start_pos: (0, c * tile_size),

# aşağıdaki kod bloğu ;
#
```

Bu fonksiyon, oyun ekranında bir grid çizmek için kullanılır. draw_grid işlevi, belirli bir ölçekte düzenli aralıklarla dikey ve yatay çizgiler çizer. Her bir döngüde, gridin dikey ve yatay çizgileri çizilir.

Dikey çizgiler, her tile_size aralığında, yatay çizgiler ise ekrana yatay bir çizgi oluşturur. Bu, oyun dünyasının yapısını ve oyuncunun konumunu daha iyi görselleştirmek için kullanılır.

```python
def draw_world():
    for row in range(40):
        for col in range(40):
            if world_data[row][col] > 0:
                if world_data[row][col] == 1:
                    #dirt blocks
                    img = pygame.transform.scale(dirt_img, size: (tile_size, tile_size)) # yüklenen gör
                    screen.blit(img, dest: (col * tile_size, row * tile_size)) # görüntüyü ekranın beli
                if world_data[row][col] == 2:
                    #enemy blocks
                    img = pygame.transform.scale(blob_img, size: (tile_size, tile_size))
                    screen.blit(img, dest: (col * tile_size, row * tile_size))
                if world_data[row][col] == 3:
                    #lava
                    img = pygame.transform.scale(lava_img, size: (tile_size, tile_size // 2))
                    screen.blit(img, dest: (col * tile_size, row * tile_size + (tile_size // 2)))
                if world_data[row][col] == 4:
                    #coin
                    img = pygame.transform.scale(coin_img, size: (tile_size // 2, tile_size // 2))
                    screen.blit(img, dest: (col * tile_size + (tile_size // 4), row * tile_size + (tile
                if world_data[row][col] == 5:
                    #exit
                    img = pygame.transform.scale(exit_img, size: (tile_size,tile_size))
                    screen.blit(img, dest: (col * tile_size, row * tile_size))
```

Bu fonksiyon, oyun dünyasını (world_data listesini) ekrana çizer. Her bir dizi öğesi (kare) için, öğenin değerine bağlı olarak farklı görüntüler çizilir. Örneğin, 1 değeri olan öğeler için zemin taşı (dirt_img), 2 değeri olanlar için düşman karakter (blob_img), 3 değeri olanlar için lav (lava_img), 4 değeri olanlar için para (coin_img), ve 5 değeri olanlar için çıkış (exit_img) görüntüleri çizilir. Her görüntü, belirli bir boyuta dönüştürülür ve ekranda belirtilen konuma yerleştirilir. Bu işlev, oyun dünyasının görsel temsili için kullanılır ve oyun alanındaki farklı öğelerin görünüşünü belirler.

```python
class Button():
    def __init__(self, x, y, image):
        self.image = image
        self.rect = self.image.get_rect()
        self.rect.topleft = (x, y)
        self.clicked = False

    def draw(self):
        action = False

        #get mouse position
        pos = pygame.mouse.get_pos()

        #check mouseover and clicked conditions
        if self.rect.collidepoint(pos):
            if pygame.mouse.get_pressed()[0] == 1 and self.clicked == False:
                action = True
                self.clicked = True

        if pygame.mouse.get_pressed()[0] == 0:
            self.clicked = False

        #draw button
        screen.blit(self.image,  dest: (self.rect.x, self.rect.y))

        return action
```

Bu Button sınıfı, bir düğme oluşturmak ve etkileşimli hale getirmek için kullanılır. Başlatıldığında, düğmenin konumunu (x, y) ve görüntüsünü (image) alır, bir dikdörtgen (rect) oluşturur ve fare tıklamasını izlemek için clicked bayrağını başlatır. draw metodu, düğmenin ekrana çizilmesini sağlar ve fare tıklama durumunu kontrol eder. Fare düğmenin üzerindeyken ve tıklanmışsa action değişkenini True olarak ayarlar. Tıklama bırakıldığında clicked bayrağını sıfırlar. Bu işlev, düğmenin tıklanıp tıklanmadığını kontrol eder ve gerektiğinde bir eylem gerçekleştirilmesini sağlar.

```
#create load and save buttons
save_button = Button(screen_width // 2 - 150, screen_height - 80, save_img)
load_button = Button(screen_width // 2 + 50, screen_height - 80, load_img)
```

Bu kod, oyun ekranında kaydetme ve yükleme düğmeleri oluşturur. save_button ve load_button nesneleri, Button sınıfı kullanılarak tanımlanır. save_button ekranın alt kısmında ortalanmış ve biraz sola kaydırılmış olarak konumlandırılırken, load_button aynı hizada ancak biraz sağa kaydırılmış olarak yerleştirilir. Her iki düğme de ilgili görüntüler (save_img ve load_img) ile birlikte tanımlanır. Bu düğmeler, oyun verilerini kaydetmek ve yüklemek için kullanıcı etkileşimine olanak sağlar.

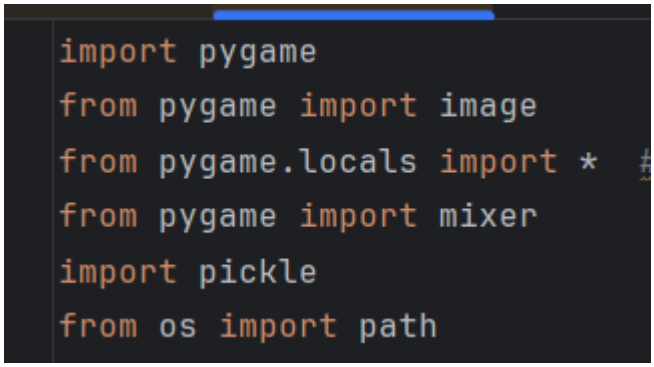
```
#main game loop
run = True
while run:

    clock.tick(fps)

    #draw background
    screen.fill(pinky)
    screen.blit(bg_img, dest: (0, 0))

    #load and save level
    if save_button.draw():
        #save level data
        pickle_out = open(f'level{level}_data', 'wb')
        pickle.dump(world_data, pickle_out)
        pickle_out.close()
    if load_button.draw():
        #load in level data
        if path.exists(f'level{level}_data'):
            pickle_in = open(f'level{level}_data', 'rb')
            world_data = pickle.load(pickle_in)


    #show the grid and draw the level tiles
    draw_grid()
    draw_world()
```

```
#text showing current level
draw_text( text: f'Level: {level}', font, white, tile_size, screen_height - 60)
draw_text( text: 'Press UP or DOWN to change level', font, white, tile_size, screen_height - 30)

#event handler
for event in pygame.event.get():
    #quit game
    if event.type == pygame.QUIT:
        run = False
    #mouseclicks to change tiles
    if event.type == pygame.MOUSEBUTTONDOWN and clicked == False:
        clicked = True
        pos = pygame.mouse.get_pos()
        x = pos[0] // tile_size
        y = pos[1] // tile_size
        #check that the coordinates are within the tile area
```

```
            #check that the coordinates are within the tile area
            if x < 40 and y < 40:
                #update tile value
                if pygame.mouse.get_pressed()[0] == 1:
                    world_data[y][x] += 1
                    if world_data[y][x] > 5:
                        world_data[y][x] = 0
                elif pygame.mouse.get_pressed()[2] == 1:
                    world_data[y][x] -= 1
                    if world_data[y][x] < 0:
                        world_data[y][x] = 5
        if event.type == pygame.MOUSEBUTTONUP:
            clicked = False
        #up and down key presses to change level number
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                level += 1
            elif event.key == pygame.K_DOWN and level > 1:
                level -= 1

    #update game display window
    pygame.display.update()

pygame.quit()
```

Bu ana oyun döngüsü, oyunun sürekli olarak çalışmasını sağlar. Döngü her iterasyonda ekranı günceller, arka planı çizer ve kullanıcı girişlerini işler. Kaydetme ve yükleme düğmeleriyle oyun dünyasını kaydeder veya yükler. Fare tıklamalarıyla kare değerlerini değiştirir ve kullanıcıya mevcut seviyeyi ve seviye değiştirme talimatlarını gösterir. Olay işleyici, oyunu kapatma, fare tıklamaları ve klavye ile seviye değiştirme işlemlerini yönetir. Döngü her seferinde ekranı günceller ve kullanıcı etkileşimlerini izler, bu sayede oyun sürekli olarak yenilenir ve kullanıcı girdilerine tepki verir.

MAİN



Bu kod, Pygame kütüphanesini ve oyun geliştirme için gereken çeşitli modülleri içe aktarır. pygame, temel Pygame işlevlerini sağlar, image resimlerle çalışmayı mümkün kılar, locals Pygame sabitlerini içerir, mixer sesleri yönetir, pickle veri serileştirme ve serileştirmeyi geri almayı sağlar, ve os.path dosya ve dizin işlemleri için kullanılır. Bu modüller, oyun geliştirme sürecinde farklı görevleri gerçekleştirmek için kullanılır.

Bu kod, Pygame ve ses mikseri modüllerini başlatmak için gerekli ayarları yapar. pygame.mixer.pre_init() ses örnekleme hızını, bit derinliğini, kanal sayısını ve tampon boyutunu ayarlar. Ardından mixer.init() ses mikserini başlatır ve pygame.init() Pygame kütüphanesinin tüm modüllerini başlatır. Bu işlemler, oyunda seslerin düzgün ve verimli bir şekilde çalışmasını sağlar.

```
clock = pygame.time.Clock()
fps = 120
```

Bu kod, Pygame'de bir saat nesnesi oluşturur ve oyunun saniyede kaç kare (FPS - Frames Per Second) hızında çalışacağını belirler. pygame.time.Clock() ile bir saat nesnesi oluşturulur ve fps = 120 ile oyunun 120 FPS hızında çalışması ayarlanır. Bu, oyun döngüsünün her saniyede 120 kez yenilenmesini sağlar, bu da oyunun akıcı ve tutarlı bir hızda çalışmasını temin eder.

```
screen_width = 600
screen_height = 600

screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption('VOID')
```

Bu kod, Pygame kütüphanesini kullanarak bir oyun penceresi oluşturur. Pencerenin genişliği ve yüksekliği, sırasıyla 600 piksel olarak belirlenir. pygame.display.set_mode() fonksiyonuyla pencerenin boyutları belirlenir ve pygame.display.set_caption() fonksiyonuyla pencerenin başlığı "TOMB" olarak ayarlanır. Bu adımlar, oyunun grafik arayüzünü başlatmak ve kullanıcıya oyunun adını göstermek için gereklidir.

```
font = pygame.font.SysFont( name: 'Bauhaus 93',  size: 70)
font_score = pygame.font.SysFont( name: 'Bauhaus 93',  size: 30)

#define game variables
tile_size = 15
game_over = 0
main_menu = True  #oyun henüz ana menüde
level = 1  #kaçıncı seviye , ilk seviye henüz
max_levels = 7 #toplam 7 seviye var.
score = 0

#define colours
white = (255, 255, 255)
blue = (0, 0, 255)
```

Bu kod, oyunun kullanacağı yazı tiplerini ve diğer bazı değişkenleri tanımlar. font ve font_score değişkenleri, oyun içinde kullanılacak yazı tiplerini belirler. tile_size, oyun taşlarının boyutunu belirlerken, game_over ve main_menu değişkenleri oyunun durumunu izler. level ve max_levels değişkenleri, mevcut seviye ve oyunun toplam seviye sayısını belirtir. score değişkeni, oyuncunun puanını tutar. white ve blue gibi renkler, oyun arayüzünde kullanılacak renkleri belirler. Bu değişkenler, oyunun genel yapılandırması ve görsel özelliklerini belirlemek için kullanılır.

```
#load images
bg_img = pygame.image.load("C:/Masaüstü\OYUNPROJE_RECREA\oyunpro
bg_img = pygame.transform.scale(bg_img,  size: (600, 600))
restart_img = pygame.image.load("C:/Masaüstü\OYUNPROJE_RECREA\oyu
start_img = pygame.image.load('img/button.png')
start_img = pygame.transform.scale(start_img,  size: (80, 80))
exit_img = pygame.image.load('img/button.png')
exit_img = pygame.transform.scale(exit_img,  size: (80, 80))
```

```
#load sounds
pygame.mixer.music.load("C:/Masaüstü\OYUNPROJE_RECREA\oyun
pygame.mixer.music.play(-1, start: 0.0, fade_ms: 5000)
coin_fx = pygame.mixer.Sound("C:/Masaüstü\OYUNPROJE_RECREA
coin_fx.set_volume(0.5)
game_over_fx = pygame.mixer.Sound('img/game_over.wav')
game_over_fx.set_volume(0.5)
```

Bu bölüm, oyun için gerekli olan görsel ve ses dosyalarını yükler. bg_img, oyun arka planı için bir resim yükler ve boyutunu ayarlar. restart_img, yeniden başlatma düğmesi için bir resim yükler. start_img ve exit_img, başlatma ve çıkış düğmeleri için resimler yükler ve boyutlarını ayarlar. pygame.mixer.music.load() ile oyun müziği yüklenir ve pygame.mixer.music.play() ile oynatılır. coin_fx ve game_over_fx gibi ses efektleri de yüklenir ve ses seviyeleri ayarlanır. Bu adımlar, oyunun görsel ve işitsel unsurlarını yükleyerek oyuncuya daha zengin bir deneyim sunar.

```
def draw_text(text, font, text_col, x, y):
    img = font.render(text, True, text_col)
    screen.blit(img, dest: (x, y))
```

Bu fonksiyon, ekrana metin çıktısı vermek için kullanılır. draw_text işlevi, belirli bir metni belirtilen bir yazı tipi, renk ve konumda ekrana çizer. Metni bir resme dönüştürmek için font.render() yöntemi kullanılır ve bu metin belirtilen konuma ((x, y)) yerleştirilir. Bu işlev, oyun içinde bilgi veya talimat sağlamak için kullanılır ve metin tabanlı bilgilerin görüntülenmesini sağlar.

```
def reset_level(level):
    player.reset( x: 80,  y: 525)
    blob_group.empty()
    coin_group.empty()
    lava_group.empty()
    exit_group.empty()

    #load in level data and create world
    if path.exists(f'level{level}_data'): #o dosya var mı yok mu diye kon
        # level derecesine göre dosya adı da değişir.
        pickle_in = open(f'level{level}_data', 'rb') #rb = read binary
        #dosyayı okur.
        world_data = pickle.load(pickle_in)#verileri yükleyip worlddata
    world = World(world_data)#world nesnesi oluşturur.
    #create dummy coin for showing the score
    score_coin = Coin(tile_size // 2, tile_size // 2) #başlangıç koordina
    coin_group.add(score_coin) #score_coin nesnesini coin_group adlı grub
    return world #oluşturulan world nesnesini geri döndürür.
```

Bu fonksiyon, bir seviyenin sıfırlanması için kullanılır. Belirli bir seviyenin verilerini sıfırlar ve yeni bir oyun dünyası oluşturur. İlk olarak, oyuncunun konumunu sıfırlar ve tüm düşman, para, lav ve çıkış gruplarını temizler. Daha sonra, belirtilen seviyenin veri dosyasını yükler ve bu verilere dayanarak yeni bir oyun dünyası oluşturur. Son olarak, puanı göstermek için bir sahte para ekler ve oluşturulan dünyayı döndürür. Bu işlev, oyuncunun seviyeyi yeniden başlatmasını ve yeni bir oyun dünyası oluşturmasını sağlar.

```python
class Button():
    def __init__(self, x, y, image):
        self.image = image #button resmi
        self.rect = self.image.get_rect() #düğmenin konumunu ve boyutu
        #koordinatlar
        self.rect.x = x
        self.rect.y = y
        self.clicked = False #düğme tıklanmadı

    def draw(self):
        action = False

        #get mouse position
        pos = pygame.mouse.get_pos()

        #check mouseover and clicked conditions
        if self.rect.collidepoint(pos): #çarpışıyolar mı
            if pygame.mouse.get_pressed()[0] == 1 and self.clicked == False:
                action = True
                self.clicked = True

        if pygame.mouse.get_pressed()[0] == 0:
            self.clicked = False

        #draw button
        screen.blit(self.image, self.rect)
        return action
```

Bu Button sınıfı, fareyle etkileşime girebilen bir düğme oluşturur. Başlatıcı yöntemi (__init__), düğmenin konumunu (x, y) ve görüntüsünü (image) alır. draw yöntemi, fare etkileşimlerini izler ve düğmenin görüntüsünü ekrana çizer. Fare düğmenin üzerine geldiğinde ve tıklandığında bir eylem gerçekleşir. Bu sınıf, kullanıcı arabirimi için düğmeler oluşturmak ve bunlarla etkileşime girmek için kullanılır.

```python
class Player():
    def __init__(self, x, y):
        self.reset(x, y) #başlangıç durumu

    def update(self, game_over):
        dx = 0
        dy = 0
        if game_over == 0:
            #get keypresses
            key = pygame.key.get_pressed() #basılan tuşlar
            if key[pygame.K_UP] : #yukarı ok (5 piksel)
                self.vel_y = -5
                self.vel_x = 0
            if key[pygame.K_DOWN] :
                self.vel_y = 5
                self.vel_x = 0
            if key[pygame.K_RIGHT] :
                self.vel_x = 5
                self.vel_y = 0
            if key[pygame.K_LEFT]:
                self.vel_x = -5
                self.vel_y = 0

            #hareketi güncelledi.
            dy += self.vel_y
            dx += self.vel_x
```

```
                                                                    ⚠12 ⚠205 ✔162
    #check for collision
    for tile in world.tile_list:
        #check for collision in x direction
        #oyuncunun boyutları
        #tile (karo) çarpıştı mı ?
        if tile[1].colliderect(self.rect.x + dx, self.rect.y, 15, 15):
            dx = 0
        #check for collision in y direction
        if tile[1].colliderect(self.rect.x, self.rect.y + dy, 15, 15):
            dy=0

    #check for collision with enemies
    if pygame.sprite.spritecollide(self, blob_group, dokill: False):
        game_over = -1 #oyun bitti
        game_over_fx.play() #oyun bitiş sesi

    #check for collision with lava
    if pygame.sprite.spritecollide(self, lava_group, dokill: False):
        game_over = -1
        game_over_fx.play()

    #check for collision with exit
    if pygame.sprite.spritecollide(self, exit_group, dokill: False):
        game_over = 1 #oyun kazanıldı

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

```
    elif game_over == -1: #oyun kaybedildi.
        self.image = self.dead_image #ölü resmi
        #game over yazısı
        draw_text( text: 'GAME OVER!', font, blue, (screen_width // 2) - 200, screen_height // 2)
        #görsel bi efekt sadece her oyunda var hemen hemen
        if self.rect.y > 200:
            self.rect.y -= 5

    #draw player onto screen
    screen.blit(self.image, self.rect)
    return game_over

def reset(self, x, y):
    self.images = []
    img = pygame.image.load("C:/Masaüstü\OYUNPROJE_RECREA\oyunprojeödevi\img/boy.png")
    img = pygame.transform.scale(img,  size: (15, 15))
    self.images.append(img)
    self.dead_image = pygame.image.load('img/ghost.png')
    self.image = self.images[0]
    self.rect = self.image.get_rect()
    self.rect.x = x
    self.rect.y = y
    self.vel_y = 0
    self.vel_x = 0
```

Bu Player sınıfı, oyun içindeki oyuncu karakterinin davranışlarını ve özelliklerini tanımlar. Başlatıcı yöntemi (__init__), oyuncunun başlangıç konumunu alır ve reset yöntemini çağırarak oyuncuyu başlangıç konumuna yerleştirir. update yöntemi, oyuncunun oyun dünyasında nasıl hareket edeceğini ve çevresiyle nasıl etkileşime gireceğini belirler. Oyuncu klavye girişlerine yanıt verir, çarpışmaları kontrol eder ve çeşitli nesnelerle etkileşime girer. reset yöntemi, oyuncuyu belirli bir konuma ve başlangıç durumuna sıfırlar. Bu sınıf, oyuncu karakterinin davranışlarını ve görünümünü kontrol etmek için kullanılır.

```
class World():
    def __init__(self, data):
        self.tile_list = []

        #load images
        brick = pygame.image.load("C:/Masaüstü\OYUNPROJE_RECREA\oyunprojeödevi\img\duvarforgame.jpg")
#nesneleri yönetmek
#boyutları
        row_count = 0
        for row in data:
            col_count = 0
            for tile in row:
                if tile == 1:
                    img = pygame.transform.scale(brick, size: (tile_size, tile_size))
                    img_rect = img.get_rect()
                    img_rect.x = col_count * tile_size
                    img_rect.y = row_count * tile_size
                    tile = (img, img_rect)
                    self.tile_list.append(tile)
                if tile == 2:
                    blob = Enemy(col_count * tile_size, row_count * tile_size)
                    blob_group.add(blob)
                if tile == 3:
                    lava = Lava(col_count * tile_size, row_count * tile_size + (tile_size // 2))
                    lava_group.add(lava)
```

```
                if tile == 4:
                    coin = Coin(col_count * tile_size + (tile_size // 2), row_count * tile_size + (tile_
                    coin_group.add(coin)
                if tile == 5:
                    exit = Exit(col_count * tile_size, row_count * tile_size)
                    exit_group.add(exit)
                col_count += 1
            row_count += 1

    def draw(self):
        for tile in self.tile_list:
            screen.blit(tile[0], tile[1])
```

Bu World sınıfı, oyun dünyasının yapısını oluşturur ve görsel temsili için gerekli işlemleri gerçekleştirir. Başlatıcı yöntemi (__init__), bir veri listesi alır ve bu veri listesindeki her bir öğeyi oyun dünyasındaki farklı nesnelere dönüştürür. Örneğin, 1 değeri olan öğeler için tuğla bloğu oluşturulur, 2 değeri olanlar için düşman karakterler eklenir, 3 değeri olanlar için lav alanları eklenir, 4 değeri olanlar için para nesneleri eklenir ve 5 değeri olanlar için çıkış noktaları eklenir. draw yöntemi, dünya üzerindeki her bir nesneyi ekrana çizer. Bu sınıf, oyun dünyasının oluşturulması ve görsel temsili için kullanılır.

```python
class Enemy(pygame.sprite.Sprite):
#düşman sağa sola hareket eder
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load('img/shooterOpen.png')
        self.image = pygame.transform.scale(self.image, size: (tile_size, tile_size))
        self.rect = self.image.get_rect()#düşmanın boyutu
        self.rect.x = x
        self.rect.y = y
        self.move_direction = 1
        self.move_counter = 0

    def update(self):
        self.rect.x += self.move_direction
        self.move_counter += 1
        if abs(self.move_counter) > 50:#sağa sola gider.(zaman birimi)
            self.move_direction *= -1
            self.move_counter *= -1
```

Bu Enemy sınıfı, düşman karakterlerin davranışlarını ve özelliklerini tanımlar. Başlatıcı yöntemi (__init__), düşmanın başlangıç konumunu alır ve pygame.sprite.Sprite sınıfının başlatıcı yöntemini çağırarak sprite'ın temel özelliklerini ayarlar. Düşmanın görüntüsü ve boyutu belirlenir ve bir dikdörtgen sınırlayıcı (rect) oluşturulur. Düşman, bir yönde hareket eder ve bir hareket sayacı kullanılarak her belirli bir mesafede yön değiştirir. update yöntemi, düşmanın konumunu günceller ve hareket sayacını izler. Hareket sayacı belirli bir değeri aştığında, düşman yönünü tersine çevirir ve hareket sayacını sıfırlar. Bu sınıf, düşman karakterlerin hareketini ve davranışlarını kontrol etmek için kullanılır.

```python
class Lava(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        img = pygame.image.load('img/hiddenSpikes.png')
        self.image = pygame.transform.scale(img, size: (tile_size, tile_size // 2))
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
```

Bu Lava sınıfı, lav alanlarının davranışlarını ve özelliklerini tanımlar. Başlatıcı yöntemi (__init__), lavın başlangıç konumunu alır ve pygame.sprite.Sprite sınıfının başlatıcı yöntemini çağırarak sprite'ın temel özelliklerini ayarlar. Lavın görüntüsü belirlenir ve boyutu ayarlanır, ardından bir dikdörtgen sınırlayıcı (rect) oluşturulur. Bu sınıf, lav alanlarının görünümünü ve konumunu belirlemek için kullanılır.

```
class Coin(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        img = pygame.image.load('img/coin.png')
        self.image = pygame.transform.scale(img, size: (tile_size // 2, tile_size // 2))
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
```

Bu Coin sınıfı, oyuncunun toplayabileceği para nesnelerinin davranışlarını ve özelliklerini tanımlar. Başlatıcı yöntemi (__init__), paranın başlangıç konumunu alır ve pygame.sprite.Sprite sınıfının başlatıcı yöntemini çağırarak sprite'ın temel özelliklerini ayarlar. Para nesnesinin görüntüsü belirlenir ve boyutu ayarlanır, ardından bir dikdörtgen sınırlayıcı (rect) oluşturulur ve bu dikdörtgen nesnenin merkezine yerleştirilir. Bu sınıf, para nesnelerinin görünümünü, konumunu ve oyuncuyla etkileşimini belirlemek için kullanılır.

```
class Exit(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        img = pygame.image.load('img/goal.png')
        self.image = pygame.transform.scale(img, size: (tile_size, tile_size))
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
```

Bu Exit sınıfı, seviyenin sonuna ulaşmak için oyuncunun hedef alacağı çıkış noktalarının davranışlarını ve özelliklerini tanımlar. Başlatıcı yöntemi (__init__), çıkış noktasının

başlangıç konumunu alır ve pygame.sprite.Sprite sınıfının başlatıcı yöntemini çağırarak sprite'ın temel özelliklerini ayarlar.

Çıkış noktasının görüntüsü belirlenir ve boyutu ayarlanır, ardından bir dikdörtgen sınırlayıcı (rect) oluşturulur. Bu sınıf, çıkış noktalarının görünümünü, konumunu ve oyuncuyla etkileşimini belirlemek için kullanılır.

```
player = Player( x: 80, y: 525) #koordinat

blob_group = pygame.sprite.Group() #ex : düşman eklemek istenirse bu gruba eklenebilir.
lava_group = pygame.sprite.Group()
coin_group = pygame.sprite.Group()
exit_group = pygame.sprite.Group()
```

Bu kod parçacığı, oyuncu karakterini ve çeşitli grupları oluşturur. Player sınıfından bir örnek oluşturularak player değişkenine atanır. Ardından, düşmanları temsil etmek için bir blob_group, lav alanlarını temsil etmek için bir lava_group, para nesnelerini temsil etmek için bir coin_group ve çıkış noktalarını temsil etmek için bir exit_group oluşturulur. Bu gruplar, oyun dünyasındaki çeşitli nesneleri yönetmek ve bunlarla etkileşime girmek için kullanılır.

```
#create dummy coin for showing the score
#score gösteriyor
score_coin = Coin(tile_size // 2, tile_size // 2) #oyunun köşesi
coin_group.add(score_coin)
```

Bu kod parçası, skoru göstermek için sahte bir para nesnesi oluşturur ve bu nesneyi coin_group adlı gruba ekler. Para nesnesinin konumu, tile_size'ın yarısı olarak belirlenir, bu sayede para nesnesi oyun ekranının sol üst köşesinde görünür. Bu sahte para nesnesi, oyuncunun topladığı gerçek paraların sayısını temsil etmek için kullanılır.

```python
if path.exists(f'level{level}_data'):
    pickle_in = open(f'level{level}_data', 'rb')
    world_data = pickle.load(pickle_in)

world = World(world_data)
```

Bu kod parçası, belirli bir seviyenin veri dosyasını yükler ve bu verilere dayanarak bir oyun dünyası oluşturur. Öncelikle, belirli bir seviyenin veri dosyasının var olup olmadığını kontrol eder. Eğer dosya mevcutsa, pickle_in adında bir dosya nesnesi oluşturulur ve bu dosyadan veriler okunur. Okunan veriler, bir oyun dünyası oluşturmak için kullanılır. Oyun dünyası, World sınıfının bir örneği olarak world değişkenine atanır. Bu işlem, belirli bir seviyenin verilerini yükleyerek ve oyun dünyasını oluşturarak oyunun belirli bir seviyesini başlatır.

```python
restart_button = Button(screen_width // 2 - 50, screen_height // 2 + 100, restart_img)
start_button = Button(screen_width // 2 - 110, screen_height // 2, start_img)
exit_button = Button(screen_width // 2 + 70, screen_height // 2, exit_img)
```

Bu kod parçası, ekran üzerindeki düğmeleri oluşturur. Button sınıfından üç düğme oluşturulur: yeniden başlatma (restart_button), başlatma (start_button) ve çıkış (exit_button) düğmeleri. Her düğmenin konumu, ekranın ortasına yerleştirilmiştir, fakat yatayda farklılaşmalar yaparak düğmeler arasında bir mesafe bırakılmıştır. Bu düğmeler, kullanıcı arayüzü üzerinde kullanıcının belirli eylemleri gerçekleştirmesini sağlar, örneğin oyunu yeniden başlatmak, oyunu başlatmak veya oyunu kapatmak gibi.

```python
run = True
while run:

    clock.tick(fps)

    screen.blit(bg_img, dest: (0, 0)) #background resmi

#false durumunda exit ve start
    if main_menu == True:
        if exit_button.draw():
            run = False
        if start_button.draw():
            main_menu = False

#eğer ana menüde değilsek oyun world çizer.
    else:
        world.draw()
#her şeyi günceller.
        if game_over == 0:#oyun devam mı değil mi ?
            blob_group.update() #blob günceller.
            # update score
            #check if a coin has been collected
            if pygame.sprite.spritecollide(player, coin_group, dokill: True):#sprite ile coin çakı
                score += 1 #artar
                coin_fx.play() #ses efekti
            draw_text('X ' + str(score), font_score, white, tile_size - 10, y: 10)
```

```
        #ekrana çizer
        #gruptaki spritelar oyun ekranına aktarılır
        blob_group.draw(screen)
        lava_group.draw(screen)
        coin_group.draw(screen)
        exit_group.draw(screen)

        #oyuncunun pozisyonunu ve diğer özelliklerini günceller
        game_over = player.update(game_over)

        #if player has died
        if game_over == -1:
            if restart_button.draw():
                world_data = []
                world = reset_level(level)
                game_over = 0
                score = 0

        #if player has completed the level
        if game_over == 1:
            #reset game and go to next level
            level += 1
            if level <= max_levels:#yeni level
                #reset level
                world_data = []
                world = reset_level(level)
                game_over = 0
```

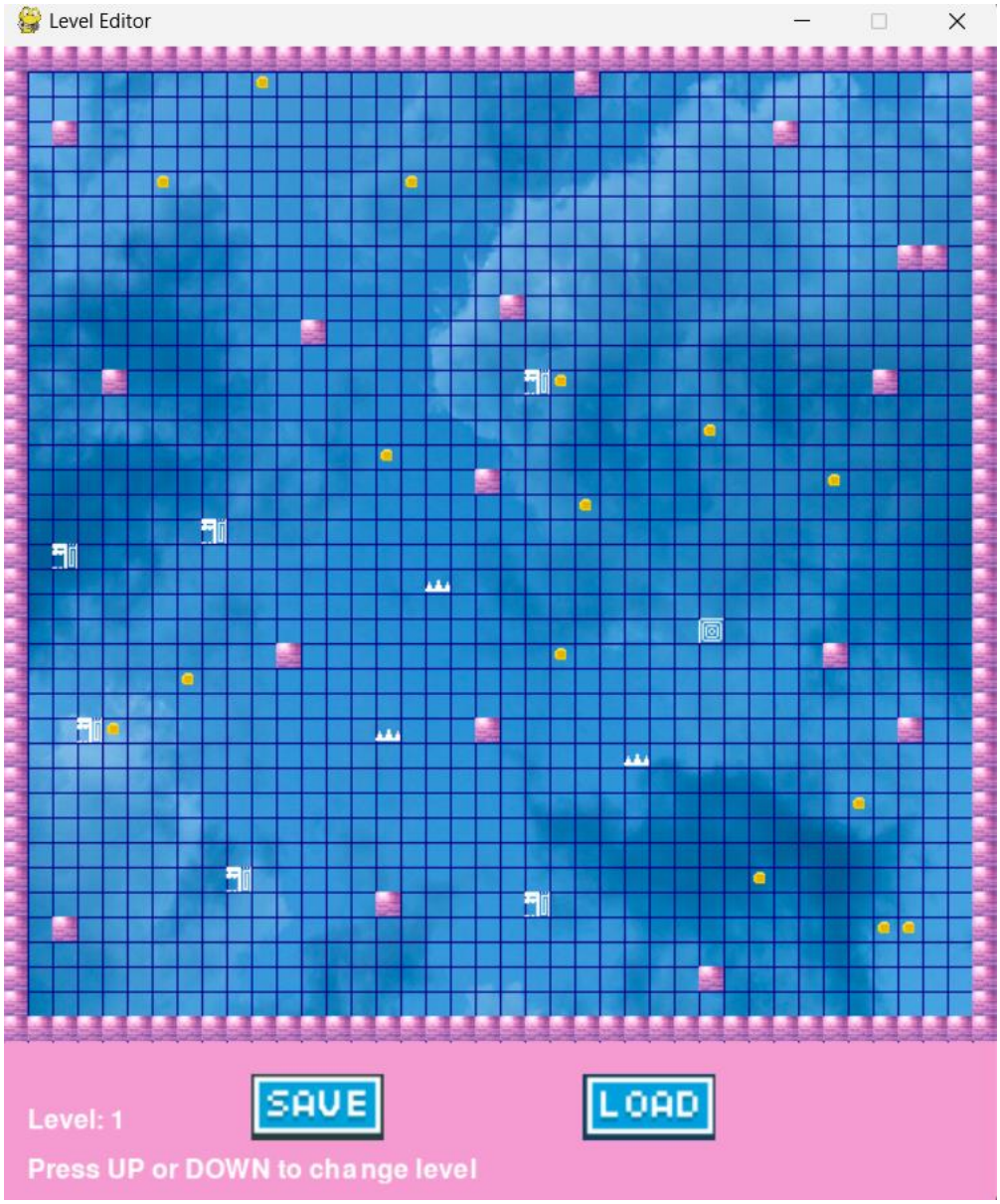
```
                game_over = 0
            else:
                draw_text( text: 'YOU WIN!', font, blue, (screen_width // 2) - 140, screen_height // 2)
                if restart_button.draw():#yeni oyun başlar çünkü tamamladı
                    level = 1
                    #reset level
                    world_data = []
                    world = reset_level(level)
                    game_over = 0
                    score = 0

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

    pygame.display.update()

pygame.quit()
```

Bu kod parçası, ana oyun döngüsünü oluşturur. run değişkeni True olduğu sürece döngü çalışır. Her döngüde, ekranın belirli bir FPS (frame per second) değerine ayarlanmasını sağlayan clock.tick(fps) işlemi gerçekleştirilir. Ana menü gösterilirken,

çıkış ve başlatma düğmelerinin durumu kontrol edilir ve kullanıcının bir eylem yapması beklenir.

Oyun başladığında, oyun dünyası (world) çizilir ve oyuncunun etkileşime girdiği düşmanlar, lav alanları, para nesneleri ve çıkış noktaları gösterilir. Oyun sırasında, oyuncunun durumu ve skoru güncellenir. Oyuncu öldüğünde veya bir seviyeyi tamamladığında, buna göre oyun durumu yeniden başlatılır veya bir sonraki seviyeye geçilir. Kullanıcının ekranı kapatması durumunda, run değişkeni False olur ve oyun döngüsü sona erer. Bu kod parçası, oyunun ana oynanışını kontrol eder ve kullanıcı etkileşimlerine yanıt verir.

Oyunun edit_level.py çıktısı;



Main.py çıktısı;

KAYNAKÇA

https://github.com/Ng0cBoM/TombOfTheMask---Pygame

Free Coin Sound Effects Download - Pixabay

Free Video Oyunları Music MP3 Download - Pixabay

https://unsplash.com/photos/low-angle-view-of-blue-clouds-qD9xzm7yK9U

Colors RGB and RGBA (w3schools.com)

https://www.canva.com/design/DAGGEDHAhtc/z8AIQcuzRidy9sZ7De9hGQ/edit?ui=eyJFIjp7IkE_IjoiSCJ9fQ

VOID - Logo (canva.com)