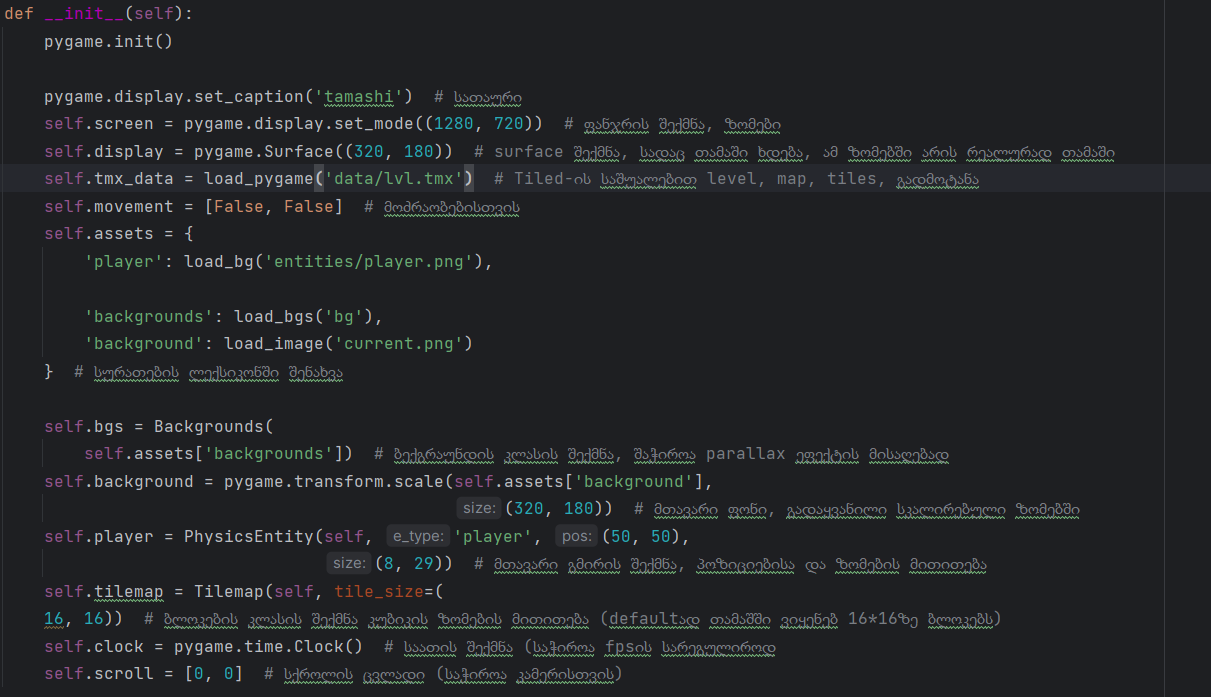
# შუალედური პროექტი

## პლატფორმული სტილის თამაში pygame-ის გამოყენებით

თამაში შექმნილია pygame-ში, main.py ფაილში არის თამაშის ბირთვი, ციკლი.

ინიტიალიზაციისას:



აქ ვქმნი ატრიბუტებს რაც გამოგვადგება თამაშისთვის.

მთავარი ლუპი თამაშის:

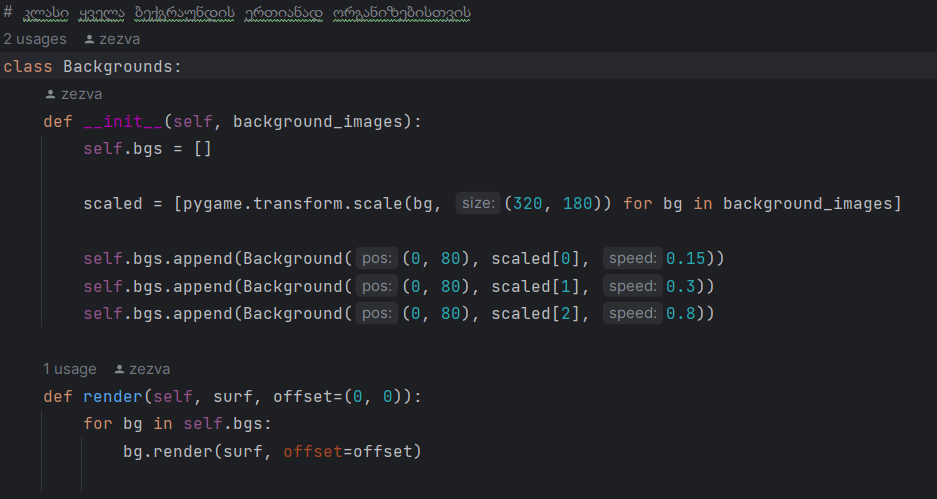
def run(self):  
 while True:  
  
 self.display.fill((255, 255, 255)) # ყოველი იტერაციის დასაწყისში ე.წ refresh გაკეთება  
  
 self.display.blit(self.background, (0, 0)) # ბექგრაუნდის გამოტანა  
 # პლეიერსა და კამერას შორის კოორდინატების სხვაობის გამოთვლა და  
 # დათვლა თუ რამდენით უნდა გადაიწიოს თამაშის თითოეულმა ელემენტმა კამერის ეფექტის მისაღებად  
 self.scroll[0] += (self.player.rect().centerx - self.display.get\_width() / 2 - self.scroll[0]) / 30  
 self.scroll[1] += (self.player.rect().centery - self.display.get\_height() / 2 - self.scroll[1]) / 30 - 1  
 render\_scroll = (int(self.scroll[0]), int(self.scroll[1])) # საბოლოო ცვლადი  
  
 self.bgs.render(self.display,  
 offset=render\_scroll) # მთავარი ბაქგრაუნდის შემდეგ უფრო ახლოს მყოფი მაგ, მთების, გორაკების გამოტანა  
 self.tilemap.render(self.display, offset=render\_scroll) # ტურის ბლოკების გამოტანა  
  
 self.player.update(self.tilemap,  
 (self.movement[1] - self.movement[0], 0)) # მთავარი გმირის პოზიციების განახლება  
 self.player.render(self.display, offset=render\_scroll) # მთავარი გმირის თამაშში გამოტანა  
  
 # ღილაკების დაჭერის მიხედვით თამაშის ცვლილება  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_LEFT:  
 self.movement[0] = True  
 if event.key == pygame.K\_RIGHT:  
 self.movement[1] = True  
 if event.key == pygame.K\_UP:  
 self.player.velocity[1] = -3  
  
 if event.type == pygame.KEYUP:  
 if event.key == pygame.K\_LEFT:  
 self.movement[0] = False  
 if event.key == pygame.K\_RIGHT:  
 self.movement[1] = False  
 # თამაშის ზედაპირის გამოტანა უფრო მაღალ ეკრანის ზედაპირზე  
 screen\_scaled = pygame.transform.scale(self.display, self.screen.get\_size())  
 self.screen.blit(screen\_scaled, (0, 0))  
 pygame.display.update()  
 # fps  
 self.clock.tick(60)

## ფონი, Background და Backgrounds კლასი

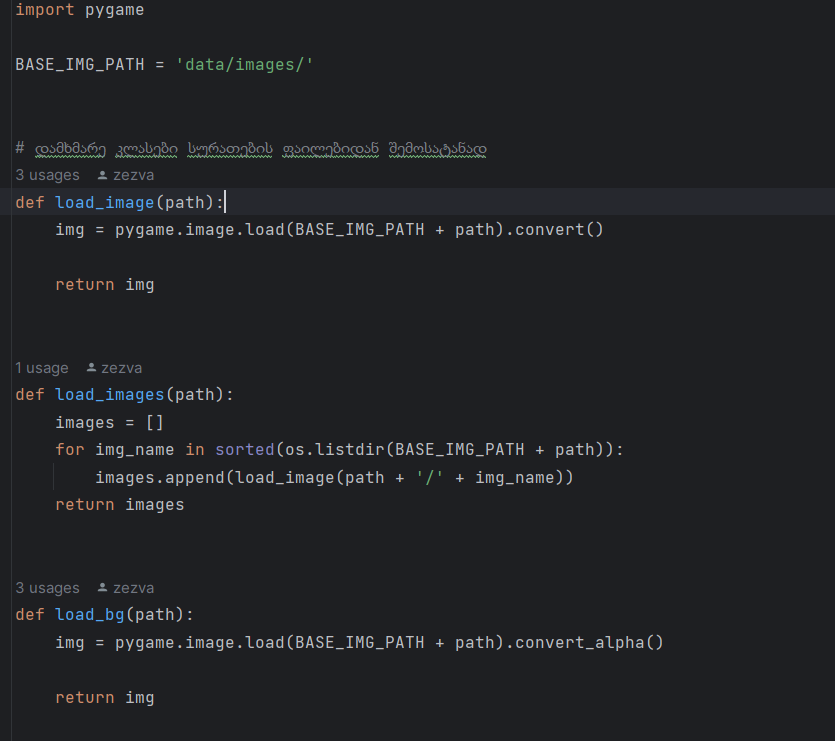
Render() ში ვიყენებ მოდულს უსასრულო ფონის შეგრძნებისთვის თუმცა არ მუშაობს კარგად



ფონის გამოტანა და ბექგრაუნდ კლასების შექმნა სიჩქარეების, პოზიციების მინიჭება



## სურათების გადმოტანა ფაილებიდან

ამისთვის მაქვს დამხმარე ფუნქციები 

## ტური

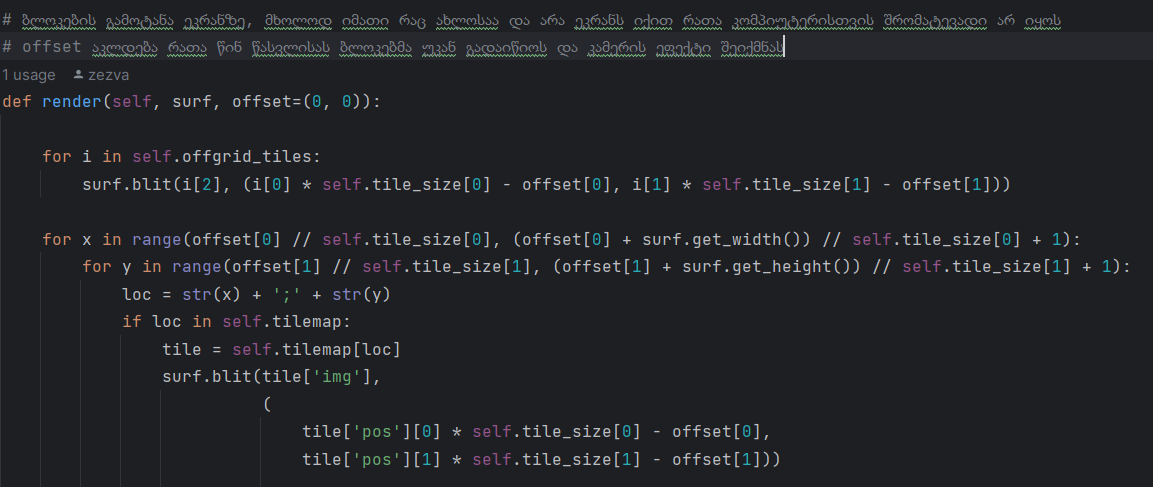
ტურის შექმნა ხდება tiled აპლიკაციის საშუალებით რათა თითოეული ბლოკი ხელით არ შეიქმნას, ამას დიდი დრო უნდა. Tiled-ს აქვს ბიბლიოთეკა

from pytmx.util\_pygame import load\_pygame

რომელიც საშუალებას მაძლევს თვითონ აპლიკაციაში შექმნილი ტური გადმოვიტანო თამაშში თავისი სურათებით, კოორდინატებით

იმ ბლოკების მართკუთხედებად გარდაქმნა და სიაში ჩამატება რასაც შეიძლება დავეჯახოთ, ხდება def physics\_rects\_around(self, pos): ფუნქციის საშუალებით

თვითონ ეკრანზე გამოტანა კი render ფუნქციით:



## მოთამაშის კლასი

ამ კლასში( PhysicsEntity) არის მოთამაშის ატრიბუტები, ზომა, პოზიცია, განახლებისა და ეკრანზე გამოტანის ფუნქციები.

განახლების ფუნქცია მოიცავს მოთამაშის პოზიციების შეცვლას, შეუძლია თუ არა მას მოძრაობა. მაგალითად კოდის მონაკვეთი შეუძლია თუ არა მოთამაშეს x კოორდინატზე გადაადგილება:

