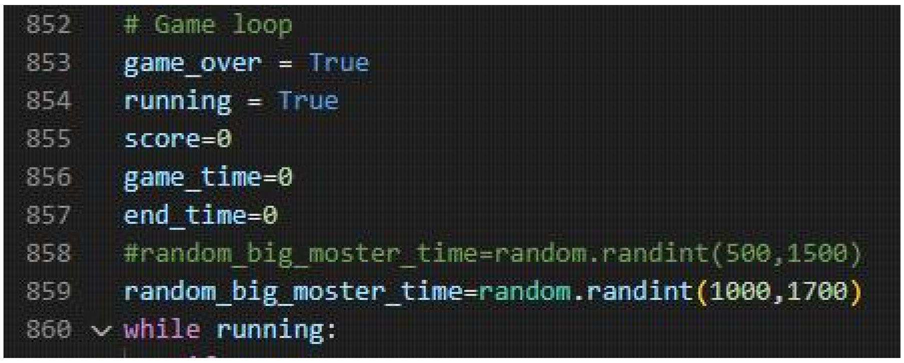
project_4

20171300 지용환

GIT :https://github.com/YONGYONGA/guaZ/tree/master#final_project의project_4 폴더

As this project, I added and changed a lot of elements in the existing shump (shooting game). Since the line of code is about 1200 lines, I will explain the unnecessary part very simply.

Most of the global variables are loaded with sounds and pictures and will be explained when necessary, but there are 3 important variables at the start.



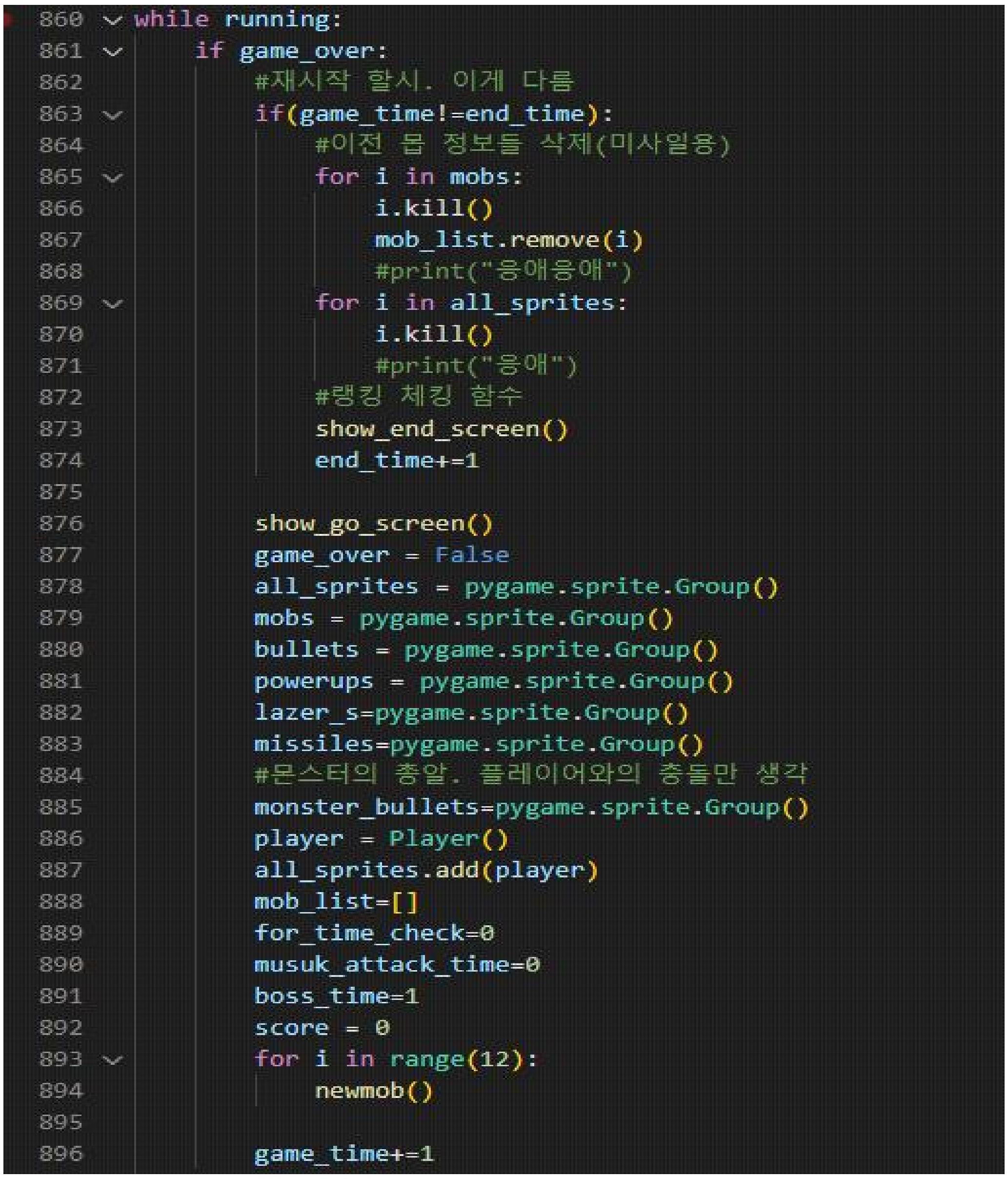
```python
# Game loop
game_over = True
running = True
score=0
game_time=0
end_time=0
#random_big_moster_time=random.randint(500,1500)
random_big_moster_time=random.randint(1000,1700)
while running:
```

game_time = A variable that increases at the start of the game.

end_time= variable that increases at the end of the game.

random_big_monster = Randomly selects a period in which large monsters appear

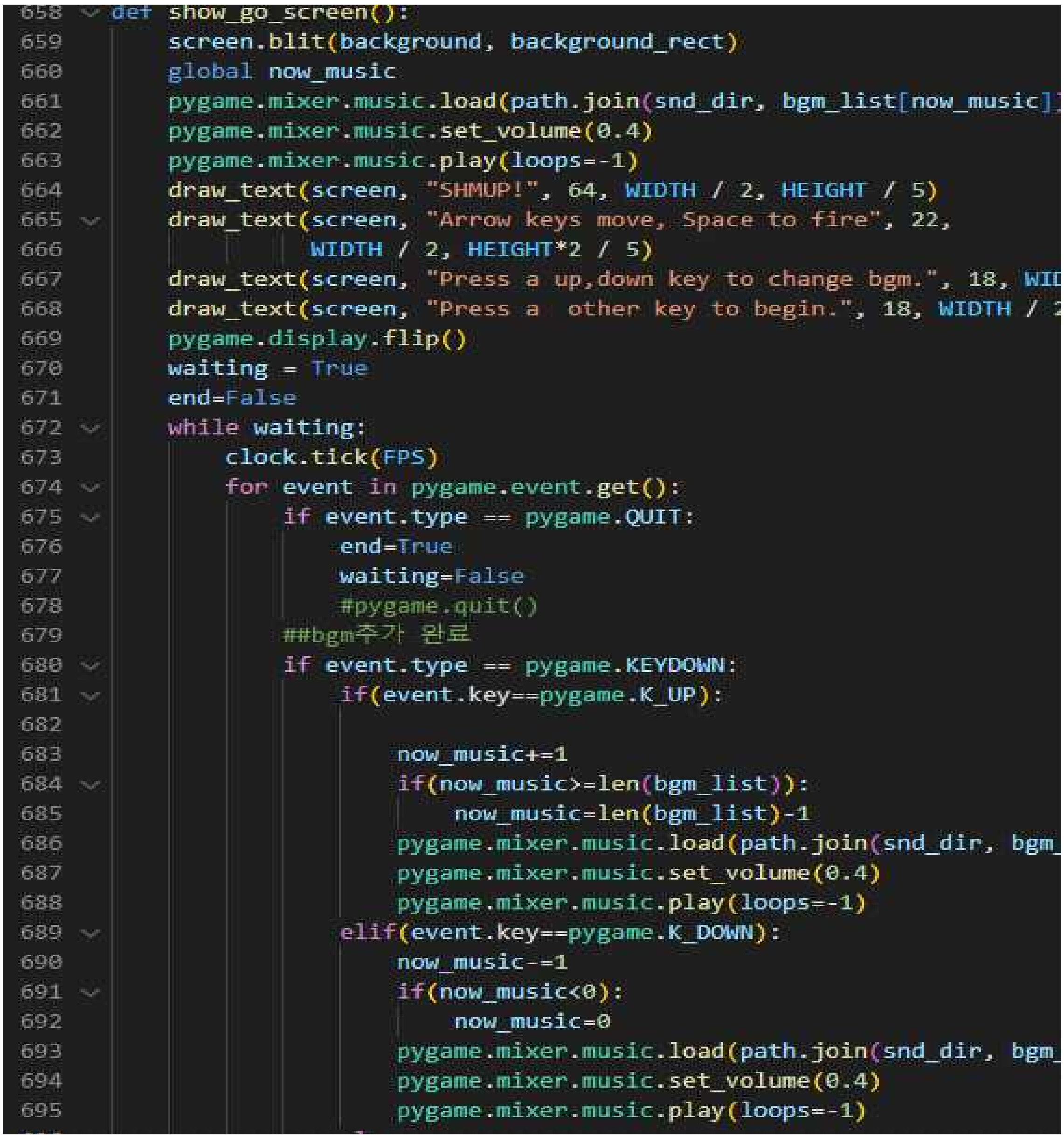
Now let's talk about the game.



```python
while running:
    if game_over:
        #재시작 할시. 이게 다름
        if(game_time!=end_time):
            #이전 몹 정보들 삭제(미사일용)
            for i in mobs:
                i.kill()
                mob_list.remove(i)
                #print("응애응애")
            for i in all_sprites:
                i.kill()
                #print("응애")
            #랭킹 체킹 함수
            show_end_screen()
            end_time+=1

        show_go_screen()
        game_over = False
        all_sprites = pygame.sprite.Group()
        mobs = pygame.sprite.Group()
        bullets = pygame.sprite.Group()
        powerups = pygame.sprite.Group()
        lazer_s=pygame.sprite.Group()
        missiles=pygame.sprite.Group()
        #몬스터의 총알. 플레이어와의 충돌만 생각
        monster_bullets=pygame.sprite.Group()
        player = Player()
        all_sprites.add(player)
        mob_list=[]
        for_time_check=0
        musuk_attack_time=0
        boss_time=1
        score = 0
        for i in range(12):
            newmob()

        game_time+=1
```

game_time and end_time are same since it is initlize =0 at start.
So it passes the if statement and since game_time+=1 mdoes not encounter this if statement until the game is over again.

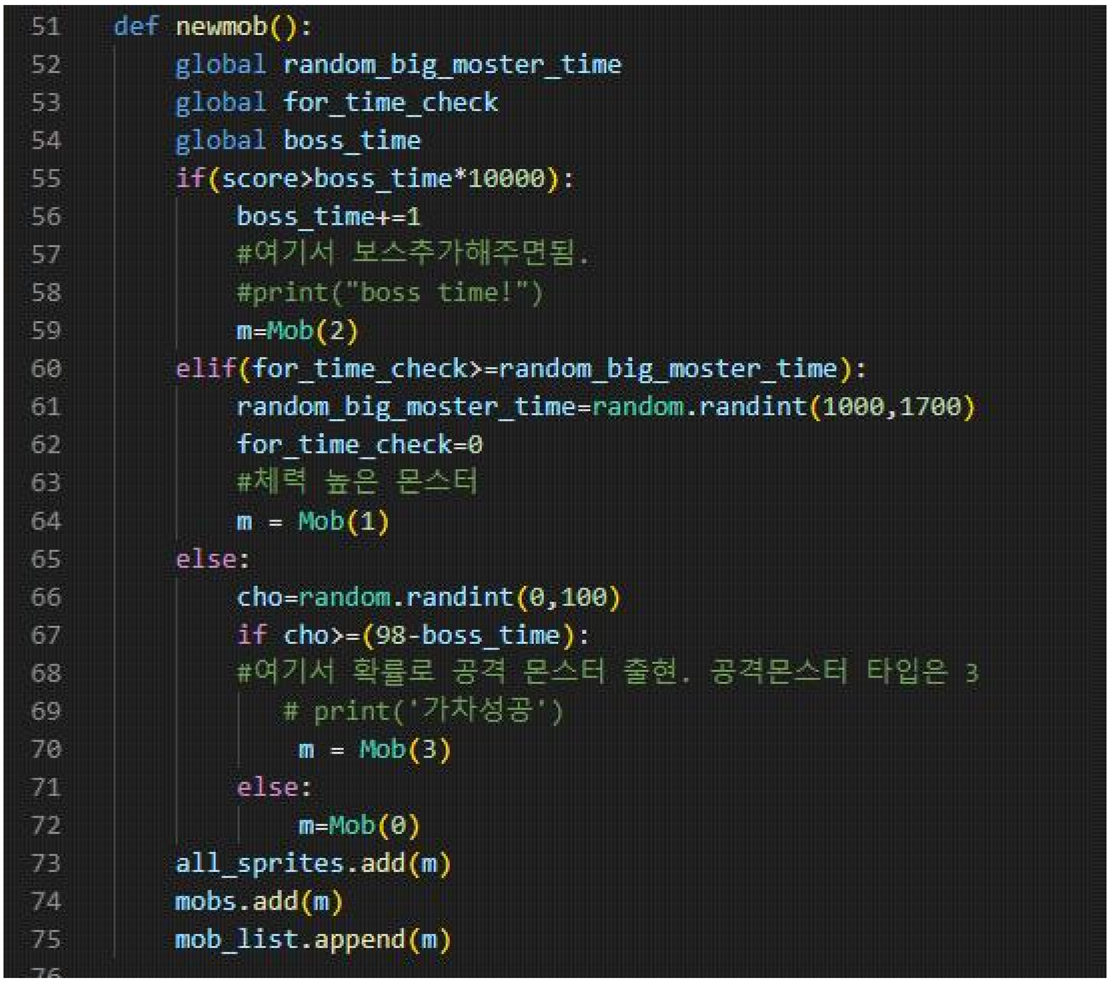
Anyway, first is show_go_screen.



```
def show_go_screen():
    screen.blit(background, background_rect)
    global now_music
    pygame.mixer.music.load(path.join(snd_dir, bgm_list[now_music]
    pygame.mixer.music.set_volume(0.4)
    pygame.mixer.music.play(loops=-1)
    draw_text(screen, "SHMUP!", 64, WIDTH / 2, HEIGHT / 5)
    draw_text(screen, "Arrow keys move, Space to fire", 22,
              WIDTH / 2, HEIGHT*2 / 5)
    draw_text(screen, "Press a up,down key to change bgm.", 18, WID
    draw_text(screen, "Press a  other key to begin.", 18, WIDTH / 2
    pygame.display.flip()
    waiting = True
    end=False
    while waiting:
        clock.tick(FPS)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                end=True
                waiting=False
                #pygame.quit()
            ##bgm추가 완료
            if event.type == pygame.KEYDOWN:
                if(event.key==pygame.K_UP):

                    now_music+=1
                    if(now_music>=len(bgm_list)):
                        now_music=len(bgm_list)-1
                    pygame.mixer.music.load(path.join(snd_dir, bgm_
                    pygame.mixer.music.set_volume(0.4)
                    pygame.mixer.music.play(loops=-1)
                elif(event.key==pygame.K_DOWN):
                    now_music-=1
                    if(now_music<0):
                        now_music=0
                    pygame.mixer.music.load(path.join(snd_dir, bgm_
                    pygame.mixer.music.set_volume(0.4)
                    pygame.mixer.music.play(loops=-1)
```

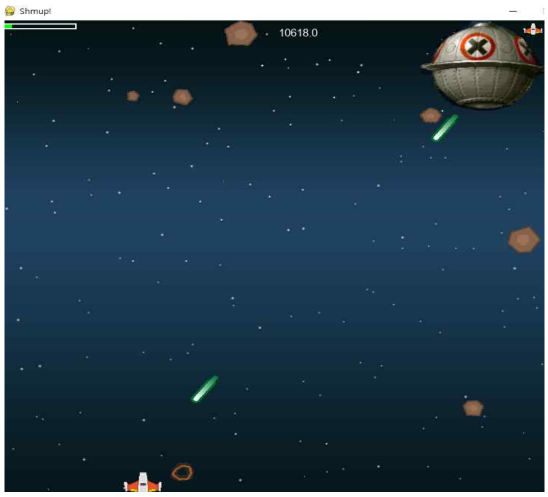
Briefly, it plays the song in the bgm_list and brings up a screen that explains the game. now_music is global variable initial value is 0. We can change this value using up, donw key then can change music in music list.
After this function is finished, all sprite groups are added and most of the variables for time check are initialized.

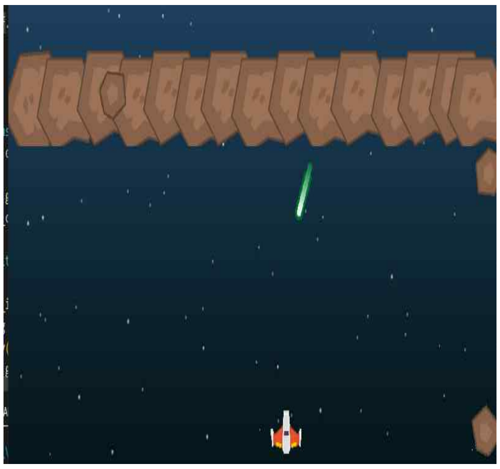
Next, i'll explain newmob()



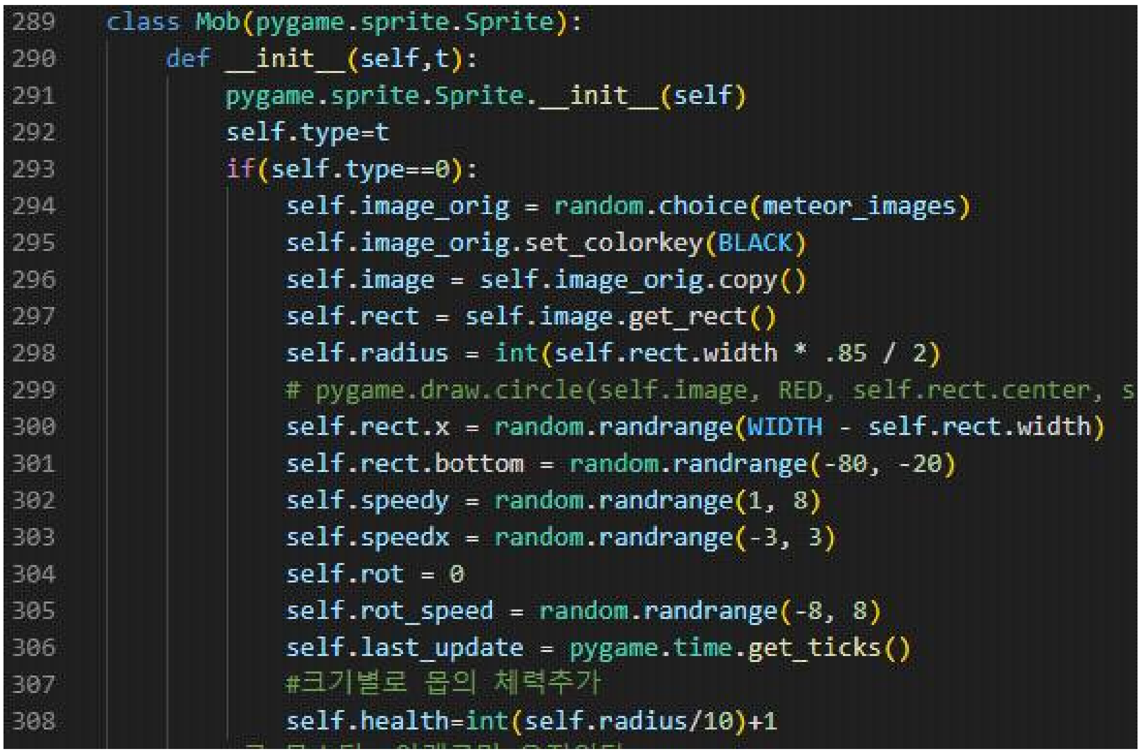
```python
def newmob():
    global random_big_moster_time
    global for_time_check
    global boss_time
    if(score>boss_time*10000):
        boss_time+=1
        #여기서 보스추가해주면됨.
        #print("boss time!")
        m=Mob(2)
    elif(for_time_check>=random_big_moster_time):
        random_big_moster_time=random.randint(1000,1700)
        for_time_check=0
        #체력 높은 몬스터
        m = Mob(1)
    else:
        cho=random.randint(0,100)
        if cho>=(98-boss_time):
        #여기서 확률로 공격 몬스터 출현. 공격몬스터 타입은 3
            # print('가차성공')
            m = Mob(3)
        else:
            m=Mob(0)
    all_sprites.add(m)
    mobs.add(m)
    mob_list.append(m)
```

Using boss_time and some time variable calculate the time the monster will appear. There is one integer in the factor of the Mob class, which is the type of monster, which is a normal monster at 0, a large monster at 1, and a special monster at 3, a boss at 2. After make this class, add this in sprite or list. This list will be used at missile. Note that boss_time increases whenever a boss appears.



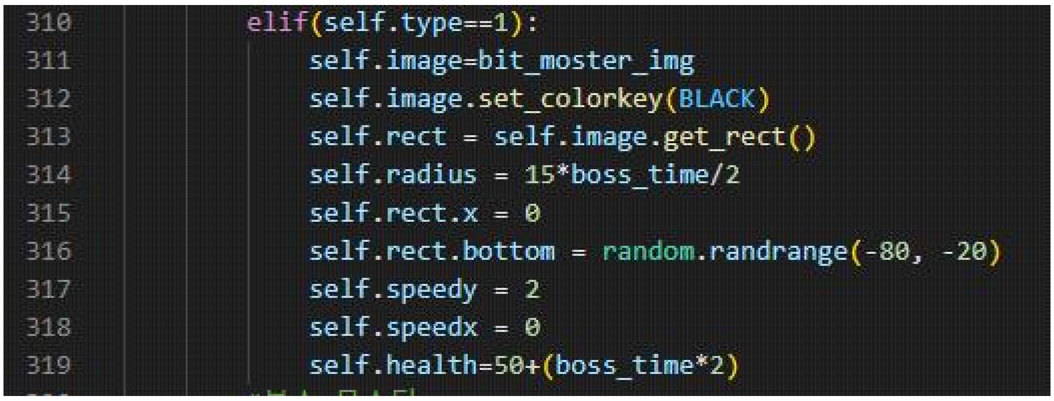


Anyway I'll explain Mob class.



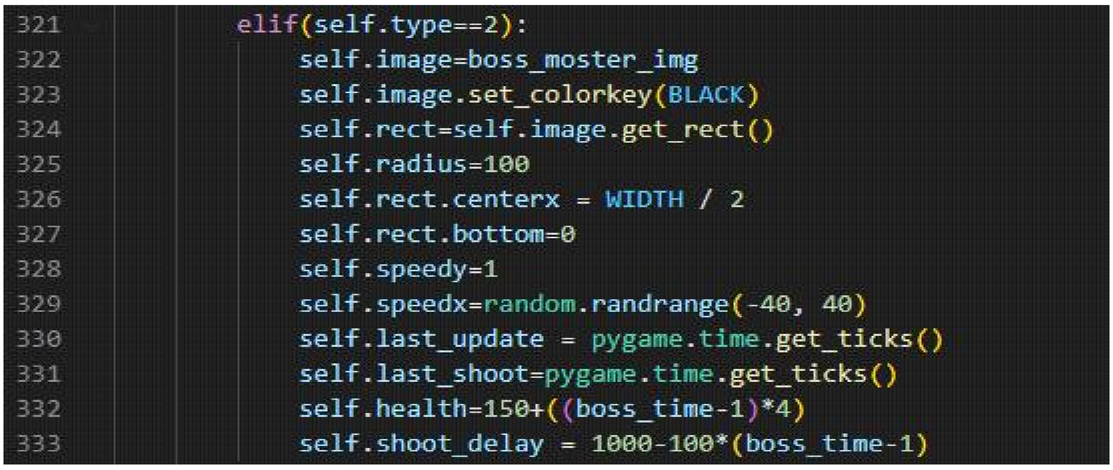
```python
class Mob(pygame.sprite.Sprite):
    def __init__(self,t):
        pygame.sprite.Sprite.__init__(self)
        self.type=t
        if(self.type==0):
            self.image_orig = random.choice(meteor_images)
            self.image_orig.set_colorkey(BLACK)
            self.image = self.image_orig.copy()
            self.rect = self.image.get_rect()
            self.radius = int(self.rect.width * .85 / 2)
            # pygame.draw.circle(self.image, RED, self.rect.center, s
            self.rect.x = random.randrange(WIDTH - self.rect.width)
            self.rect.bottom = random.randrange(-80, -20)
            self.speedy = random.randrange(1, 8)
            self.speedx = random.randrange(-3, 3)
            self.rot = 0
            self.rot_speed = random.randrange(-8, 8)
            self.last_update = pygame.time.get_ticks()
            #크기별로 몹의 체력추가
            self.health=int(self.radius/10)+1
```

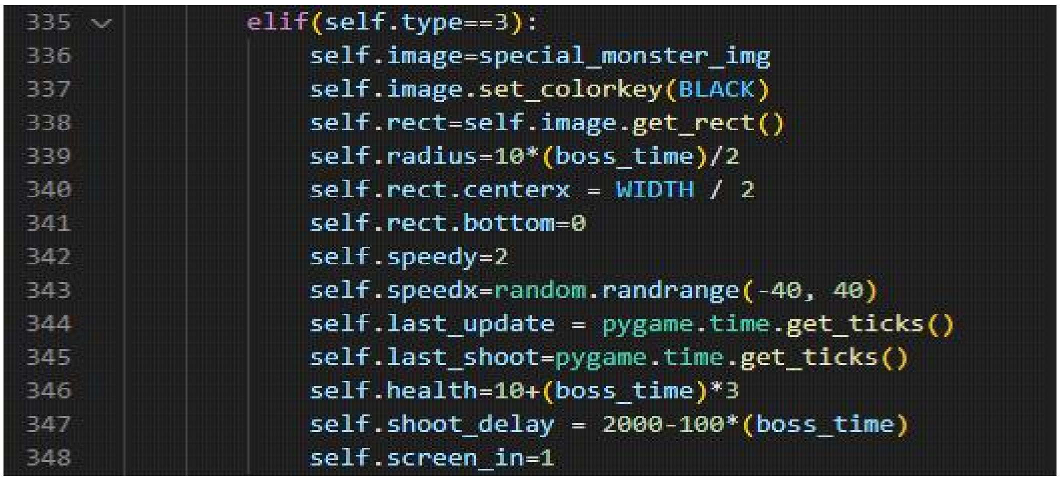
As I explained. type=0 is normal monster. Most of the code is written as is, so detailed explanations will be omitted. Here, health is the physical strength of the monster, and monsters die when their physical strength reaches 0.

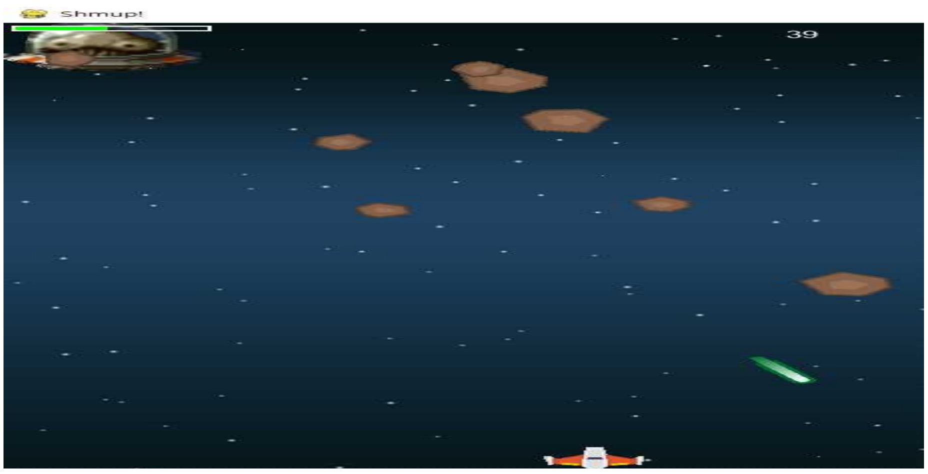


```python
        elif(self.type==1):
            self.image=bit_moster_img
            self.image.set_colorkey(BLACK)
            self.rect = self.image.get_rect()
            self.radius = 15*boss_time/2
            self.rect.x = 0
            self.rect.bottom = random.randrange(-80, -20)
            self.speedy = 2
            self.speedx = 0
            self.health=50+(boss_time*2)
```

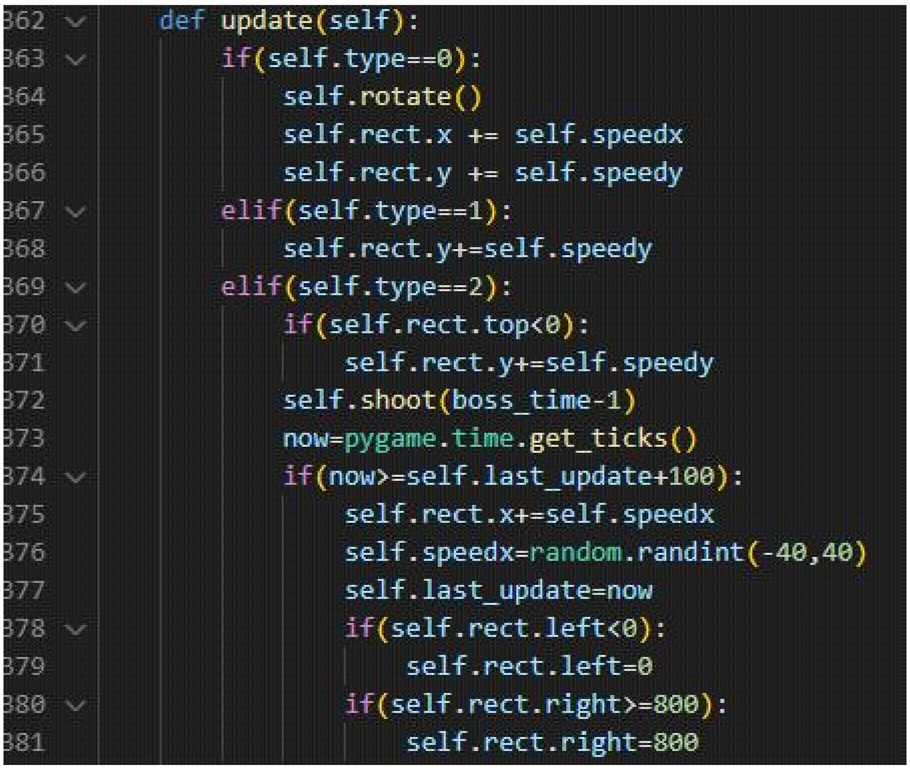
This is big monster. self.radius means the damage taken, not the radius of the monster, and the physical strength is, of course, the physical strength of the monster. This monster is packed horizontally, so it only goes down, so there is no speed on the x-axis, and its stamina increases according to the boss time.


```
        elif(self.type==2):
            self.image=boss_moster_img
            self.image.set_colorkey(BLACK)
            self.rect=self.image.get_rect()
            self.radius=100
            self.rect.centerx = WIDTH / 2
            self.rect.bottom=0
            self.speedy=1
            self.speedx=random.randrange(-40, 40)
            self.last_update = pygame.time.get_ticks()
            self.last_shoot=pygame.time.get_ticks()
            self.health=150+((boss_time-1)*4)
            self.shoot_delay = 1000-100*(boss_time-1)
```
type 2 is boss monster. This monster moves left and right at the top of the screen and fires lasers at the player. So last_shoot and shoot_delay variables exist. Stamina and shoot delay become stronger according to boss time.


```
        elif(self.type==3):
            self.image=special_monster_img
            self.image.set_colorkey(BLACK)
            self.rect=self.image.get_rect()
            self.radius=10*(boss_time)/2
            self.rect.centerx = WIDTH / 2
            self.rect.bottom=0
            self.speedy=2
            self.speedx=random.randrange(-40, 40)
            self.last_update = pygame.time.get_ticks()
            self.last_shoot=pygame.time.get_ticks()
            self.health=10+(boss_time)*3
            self.shoot_delay = 2000-100*(boss_time)
            self.screen_in=1
```
Type 3 is special monster. This monster has low health, but shoots lasers at the player, just like the boss. In addition, it is a monster that is difficult to kill because it moves very quickly from side to side. Of course, this monster also gets stronger according to the boss time.

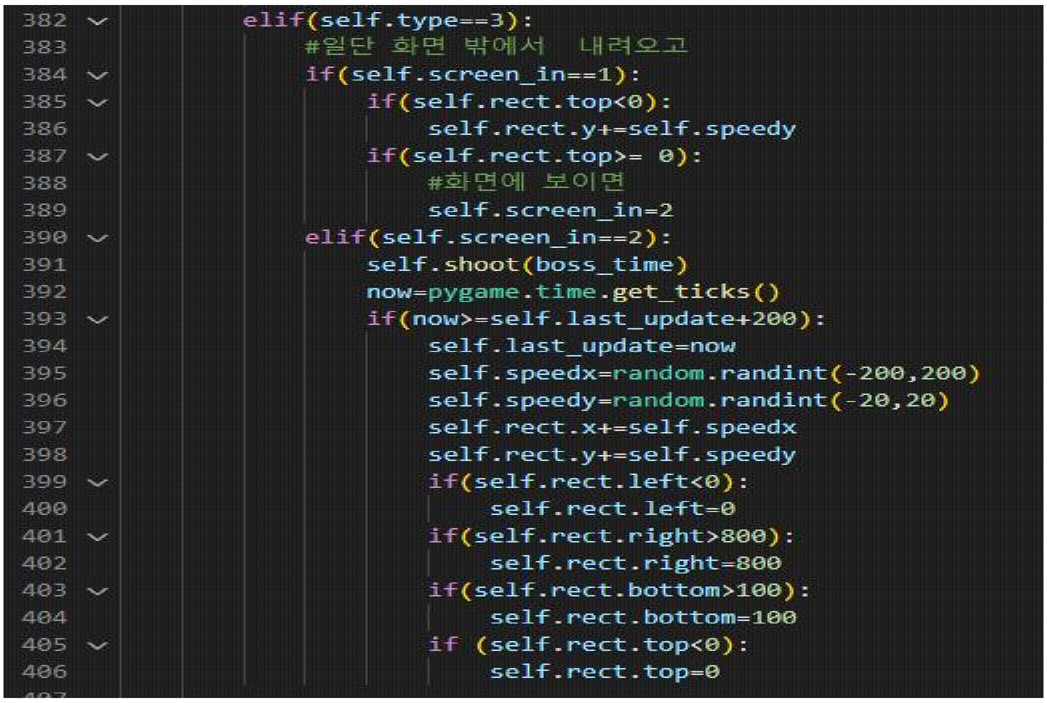


At the update of Mob



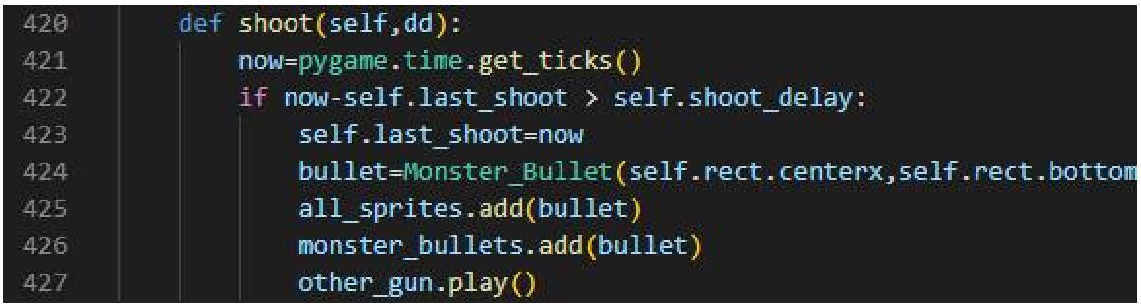
```
    def update(self):
        if(self.type==0):
            self.rotate()
            self.rect.x += self.speedx
            self.rect.y += self.speedy
        elif(self.type==1):
            self.rect.y+=self.speedy
        elif(self.type==2):
            if(self.rect.top<0):
                self.rect.y+=self.speedy
            self.shoot(boss_time-1)
            now=pygame.time.get_ticks()
            if(now>=self.last_update+100):
                self.rect.x+=self.speedx
                self.speedx=random.randint(-40,40)
                self.last_update=now
                if(self.rect.left<0):
                    self.rect.left=0
                if(self.rect.right>=800):
                    self.rect.right=800
```

The update method is different for each type. Normal monsters rotate (description omitted), big monster just come down. But type 2, after coming down to the screen, it randomly moves left and right and shoots at the player. last_update is a variable used to prevent moving too often.
Simialy type 3, a bit complicated, but nothing special.



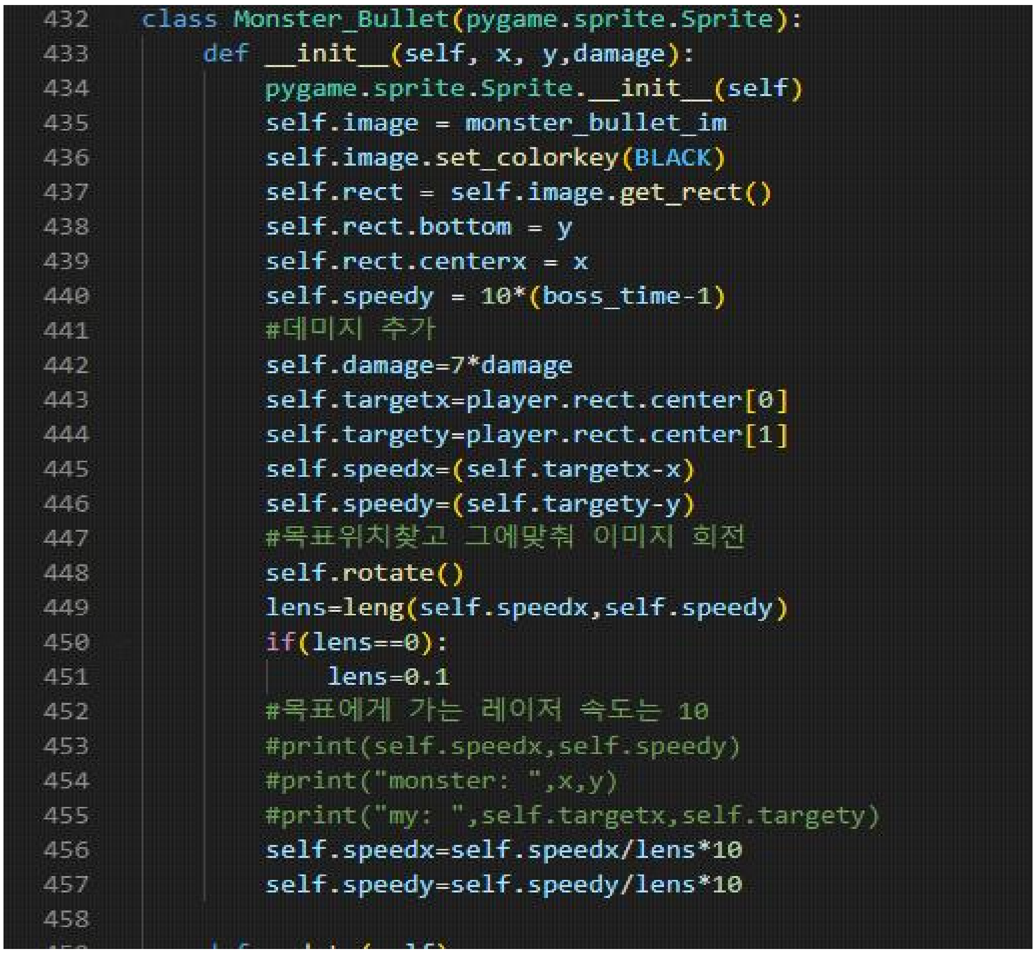
```
        elif(self.type==3):
            #일단 화면 밖에서  내려오고
            if(self.screen_in==1):
                if(self.rect.top<0):
                    self.rect.y+=self.speedy
                if(self.rect.top>= 0):
                    #화면에 보이면
                    self.screen_in=2
            elif(self.screen_in==2):
                self.shoot(boss_time)
                now=pygame.time.get_ticks()
                if(now>=self.last_update+200):
                    self.last_update=now
                    self.speedx=random.randint(-200,200)
                    self.speedy=random.randint(-20,20)
                    self.rect.x+=self.speedx
                    self.rect.y+=self.speedy
                    if(self.rect.left<0):
                        self.rect.left=0
                    if(self.rect.right>800):
                        self.rect.right=800
                    if(self.rect.bottom>100):
                        self.rect.bottom=100
                    if (self.rect.top<0):
                        self.rect.top=0
```

It just moves randomly at regular intervals after coming down to the screen. It also prevented going outside the screen. This type also shoots, and now let's look at shoot.



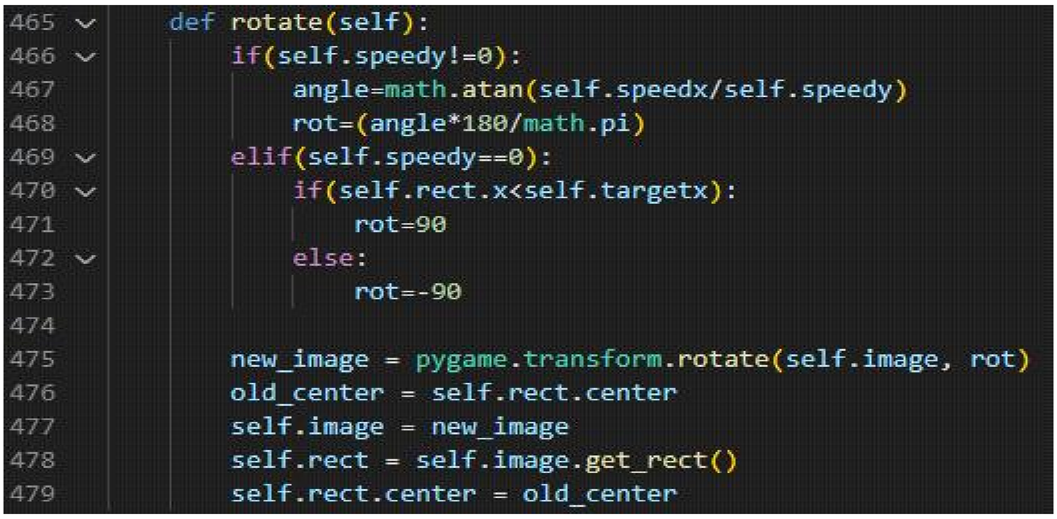
```
    def shoot(self,dd):
        now=pygame.time.get_ticks()
        if now-self.last_shoot > self.shoot_delay:
            self.last_shoot=now
            bullet=Monster_Bullet(self.rect.centerx,self.rect.bottom
            all_sprites.add(bullet)
            monster_bullets.add(bullet)
            other_gun.play()
```

Using last_shoot and shoot_delay, create a monster_bullet class at regular intervals, add it to a sprite, and make a gunshot sound. dd means damage.



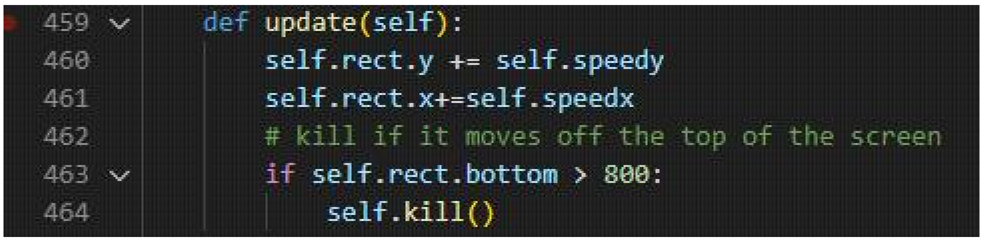
```python
class Monster_Bullet(pygame.sprite.Sprite):
    def __init__(self, x, y,damage):
        pygame.sprite.Sprite.__init__(self)
        self.image = monster_bullet_im
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        self.rect.bottom = y
        self.rect.centerx = x
        self.speedy = 10*(boss_time-1)
        #데미지 추가
        self.damage=7*damage
        self.targetx=player.rect.center[0]
        self.targety=player.rect.center[1]
        self.speedx=(self.targetx-x)
        self.speedy=(self.targety-y)
        #목표위치찾고 그에맞춰 이미지 회전
        self.rotate()
        lens=leng(self.speedx,self.speedy)
        if(lens==0):
            lens=0.1
        #목표에게 가는 레이저 속도는 10
        #print(self.speedx,self.speedy)
        #print("monster: ",x,y)
        #print("my: ",self.targetx,self.targety)
        self.speedx=self.speedx/lens*10
        self.speedy=self.speedy/lens*10

```

The most important thing for a monster bullet is a bullet that goes to the player. So, in this class, the coordinates of the player, targetx and targety, exist. Since the coordinates of the bullet are given in x and y, create a direction vector(speedx, speedy) using this variable and the target variable. And to implement the bullet moving slowly, the size of each vector quantity is set to 1 and multiplied by 10. In other words, the norm value of speedx and speedy at the end is 10. At this time, rotate is used to express the rotation of the bullet in the direction of the player.



```python
    def rotate(self):
        if(self.speedy!=0):
            angle=math.atan(self.speedx/self.speedy)
            rot=(angle*180/math.pi)
        elif(self.speedy==0):
            if(self.rect.x<self.targetx):
                rot=90
            else:
                rot=-90

        new_image = pygame.transform.rotate(self.image, rot)
        old_center = self.rect.center
        self.image = new_image
        self.rect = self.image.get_rect()
        self.rect.center = old_center
```

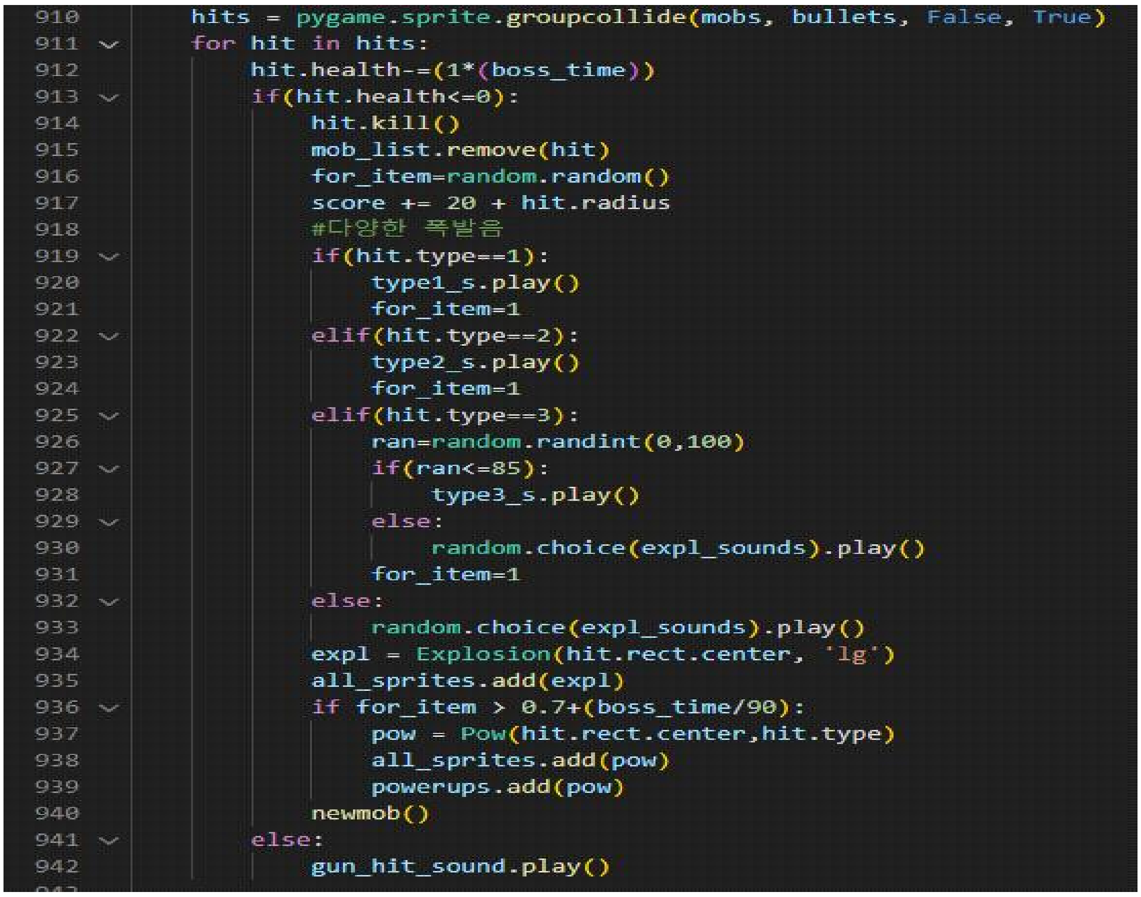
After obtaining the angle of inclination using the atan function, convert it to a radian value. At this time, there may be a division by 0 error, so in special cases, radians are directly assigned. After obtaining the radian value, if self.image is rotated using the transform function, this rotated image is now drawn when draw in the main function.



```python
    def update(self):
        self.rect.y += self.speedy
        self.rect.x+=self.speedx
        # kill if it moves off the top of the screen
        if self.rect.bottom > 800:
            self.kill()
```

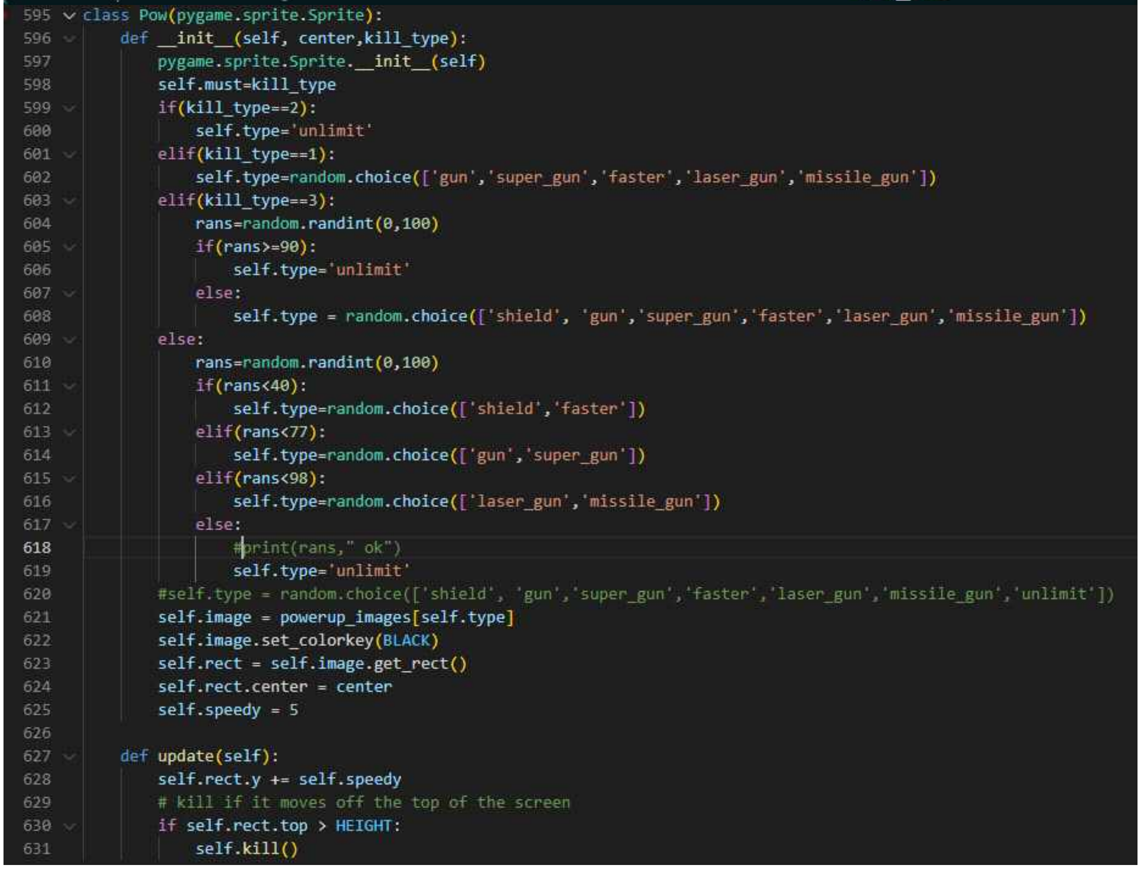
This is update method in Mob class. Just move to target using speedy and speedx. In other words, it hurts if you stay still, but it's easy to avoid. If it goes off screen, it is killed.

Now, all the explanations about the added monsters have been given.
Let's go back to the game's while loop.



```
hits = pygame.sprite.groupcollide(mobs, bullets, False, True)
for hit in hits:
    hit.health-=(1*(boss_time))
    if(hit.health<=0):
        hit.kill()
        mob_list.remove(hit)
        for_item=random.random()
        score += 20 + hit.radius
        #다양한 폭발음
        if(hit.type==1):
            type1_s.play()
            for_item=1
        elif(hit.type==2):
            type2_s.play()
            for_item=1
        elif(hit.type==3):
            ran=random.randint(0,100)
            if(ran<=85):
                type3_s.play()
            else:
                random.choice(expl_sounds).play()
            for_item=1
        else:
            random.choice(expl_sounds).play()
        expl = Explosion(hit.rect.center, 'lg')
        all_sprites.add(expl)
        if for_item > 0.7+(boss_time/90):
            pow = Pow(hit.rect.center,hit.type)
            all_sprites.add(pow)
            powerups.add(pow)
        newmob()
    else:
        gun_hit_sound.play()
```

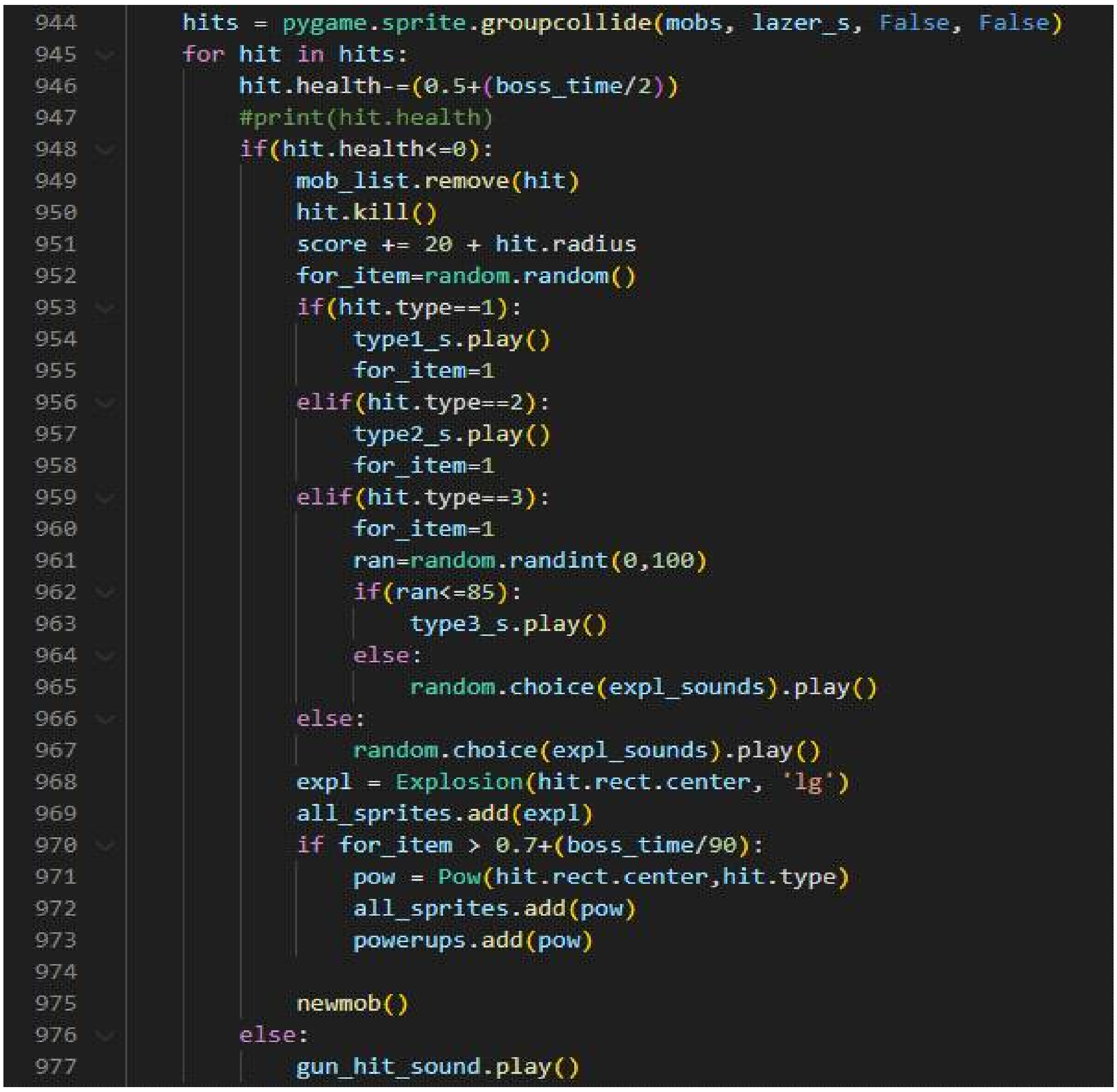
This part is to figure out the collision between normal bullets and monsters. Hit has the class of the monster that collided with the bullet, and we calculate it using this. First, the mob's HP is reduced every time it is hit, and when it runs out of HP, it is removed from kill and mob_list. Depending on the type of dead monster, the sounds that come out are different and the probability of items appearing is also different. (exp1, the explosion animation, is omitted). The for_item variable was used for the probability of the item. If a mob other than a normal monster is caught, this variable is set to 1 and a pow class is always created. If you catch a normal monster, a pow class is created according to the probability. This chance decreases with boss time. The bottom else is when the monster is not yet dead, and simply plays the sound of being shot. Anyway now, I explain about Pow class. (The bullet class is very similar to monster bullets and is already there, so I'll omit it.)



```python
class Pow(pygame.sprite.Sprite):
    def __init__(self, center,kill_type):
        pygame.sprite.Sprite.__init__(self)
        self.must=kill_type
        if(kill_type==2):
            self.type='unlimit'
        elif(kill_type==1):
            self.type=random.choice(['gun','super_gun','faster','laser_gun','missile_gun'])
        elif(kill_type==3):
            rans=random.randint(0,100)
            if(rans>=90):
                self.type='unlimit'
            else:
                self.type = random.choice(['shield', 'gun','super_gun','faster','laser_gun','missile_gun'])
        else:
            rans=random.randint(0,100)
            if(rans<40):
                self.type=random.choice(['shield','faster'])
            elif(rans<77):
                self.type=random.choice(['gun','super_gun'])
            elif(rans<98):
                self.type=random.choice(['laser_gun','missile_gun'])
            else:
                #print(rans," ok")
                self.type='unlimit'
        #self.type = random.choice(['shield', 'gun','super_gun','faster','laser_gun','missile_gun','unlimit'])
        self.image = powerup_images[self.type]
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        self.rect.center = center
        self.speedy = 5

    def update(self):
        self.rect.y += self.speedy
        # kill if it moves off the top of the screen
        if self.rect.top > HEIGHT:
            self.kill()
```

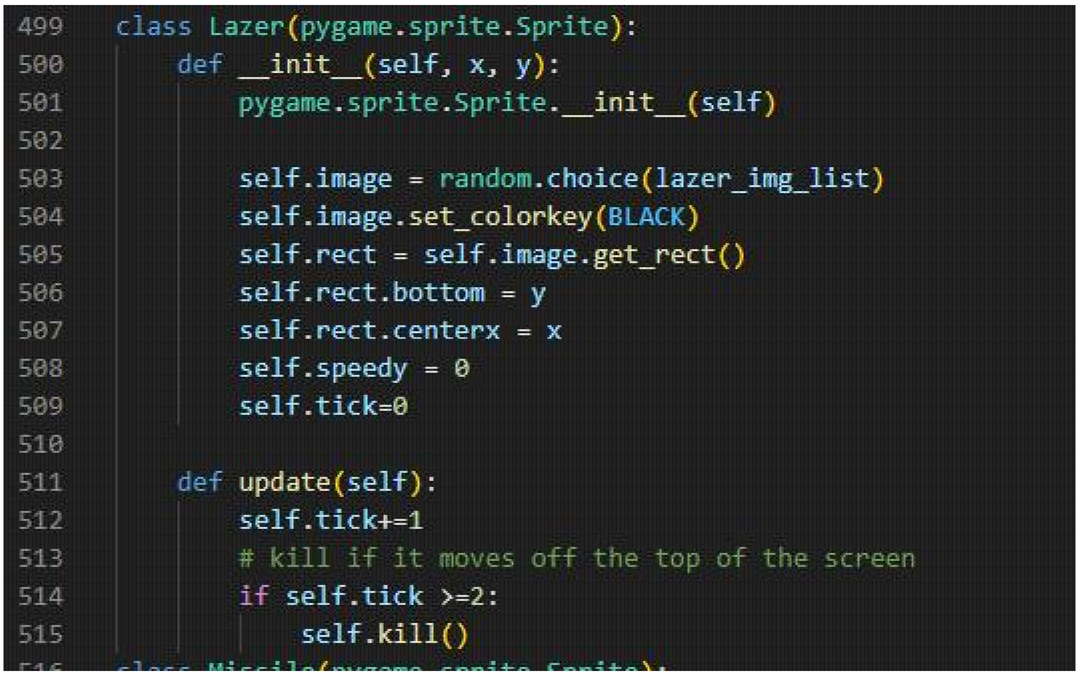
This is a simple, pre-existing class. I simply set the probability of items that appear depending on the dead mob type. For example, killing a boss always gives you an unlimit(invincibility) item. Update is just a method of stepping down and killed when out of the screen.

Now let's go back to the game loop.

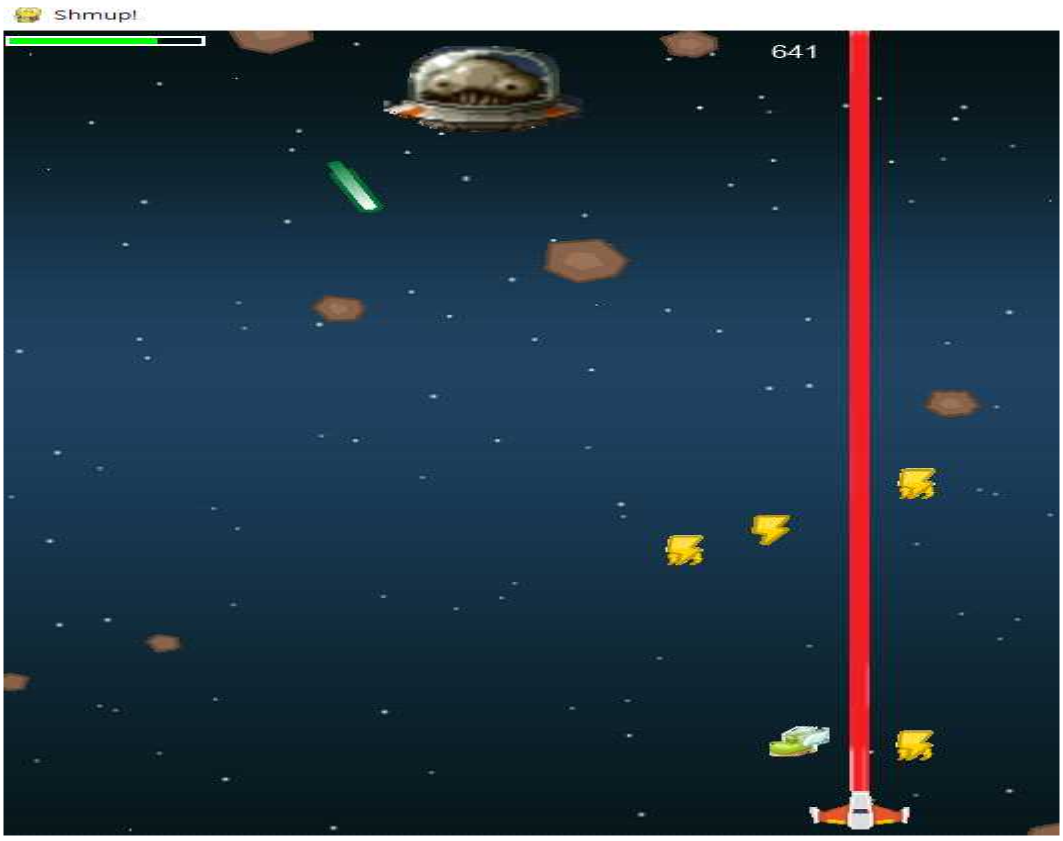


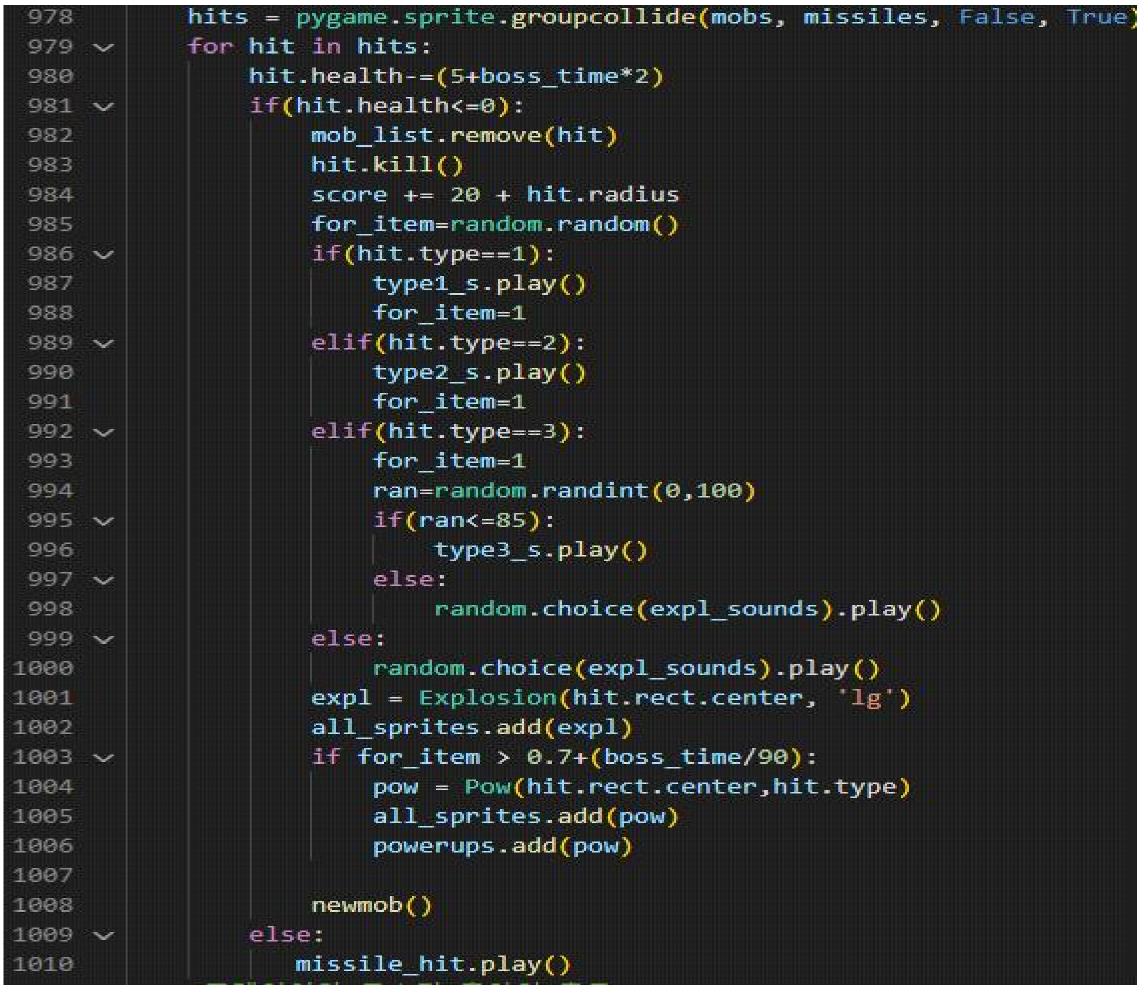
```
944         hits = pygame.sprite.groupcollide(mobs, lazer_s, False, False)
945         for hit in hits:
946             hit.health-=(0.5+(boss_time/2))
947             #print(hit.health)
948             if(hit.health<=0):
949                 mob_list.remove(hit)
950                 hit.kill()
951                 score += 20 + hit.radius
952                 for_item=random.random()
953                 if(hit.type==1):
954                     type1_s.play()
955                     for_item=1
956                 elif(hit.type==2):
957                     type2_s.play()
958                     for_item=1
959                 elif(hit.type==3):
960                     for_item=1
961                     ran=random.randint(0,100)
962                     if(ran<=85):
963                         type3_s.play()
964                     else:
965                         random.choice(expl_sounds).play()
966                 else:
967                     random.choice(expl_sounds).play()
968                 expl = Explosion(hit.rect.center, 'lg')
969                 all_sprites.add(expl)
970                 if for_item > 0.7+(boss_time/90):
971                     pow = Pow(hit.rect.center,hit.type)
972                     all_sprites.add(pow)
973                     powerups.add(pow)
974 
975                 newmob()
976             else:
977                 gun_hit_sound.play()
```

This part, which is wrapped around the collision with the laser gun, is nothing other than the normal bullet collision part and damage. To explain one additional thing, False, False in line 944 means that even if the laser collides with the mob, it is not automatically killed, that is, it continues to float on the screen. In this case, you have to kill it yourself so that no error occurs. Now let's talk about the Lazer class.


```
class Lazer(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)

        self.image = random.choice(lazer_img_list)
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        self.rect.bottom = y
        self.rect.centerx = x
        self.speedy = 0
        self.tick=0

    def update(self):
        self.tick+=1
        # kill if it moves off the top of the screen
        if self.tick >=2:
            self.kill()
```

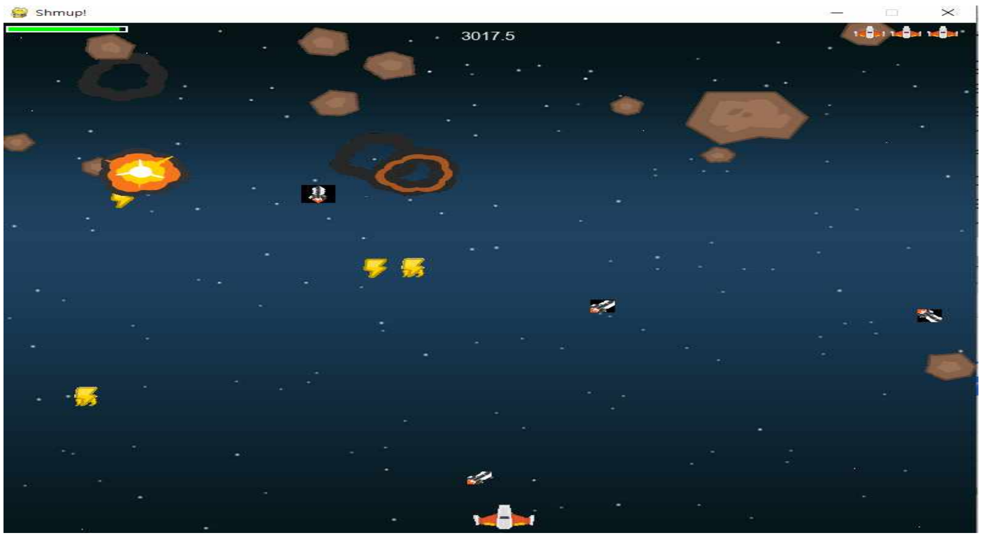
Even if you collide with a mob, it is not automatically killed, so it is automatically killed after 2 ticks. The image in the lazer_img_list is a vertically very long red, yellow, green, and blue picture. If you fire it, it will attack to the end of the screen without moving. So, speedy is 0. In addition, since it is an attack that is automatically deleted after 2 ticks, this attack penetrates all mobs.

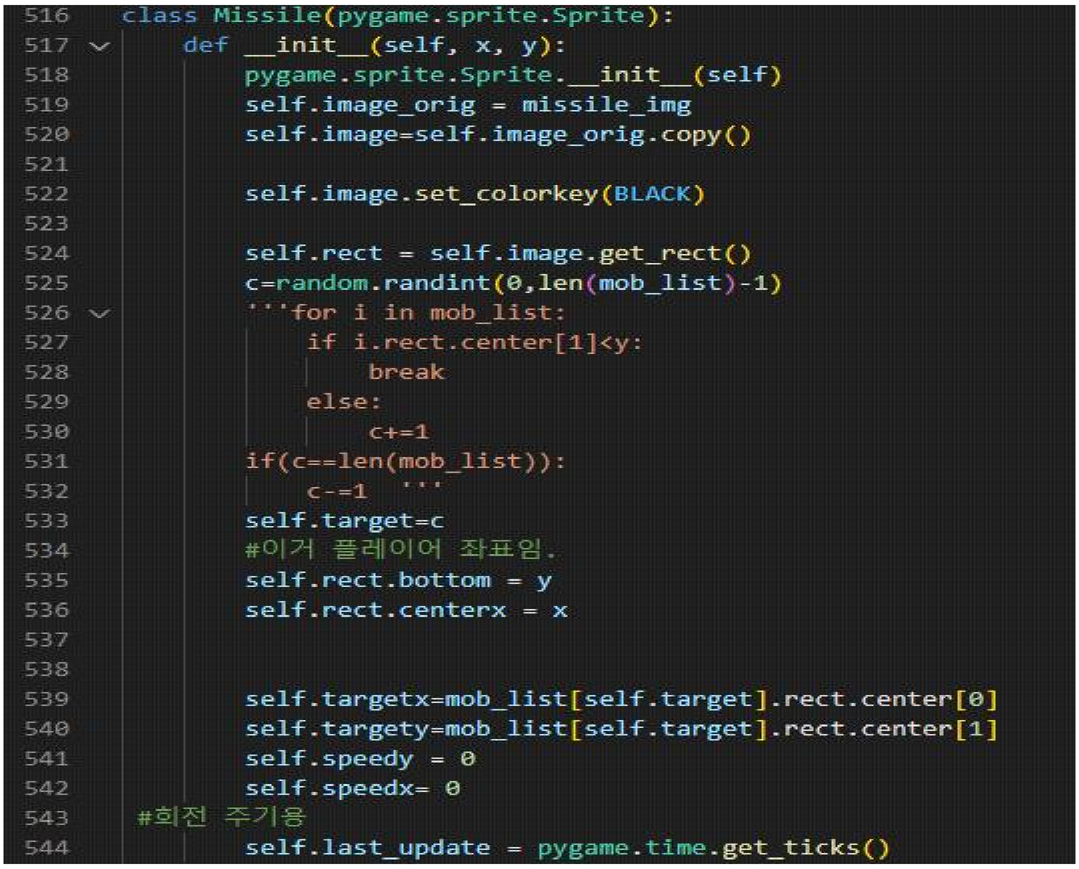




```
hits = pygame.sprite.groupcollide(mobs, missiles, False, True)
for hit in hits:
    hit.health-=(5+boss_time*2)
    if(hit.health<=0):
        mob_list.remove(hit)
        hit.kill()
        score += 20 + hit.radius
        for_item=random.random()
        if(hit.type==1):
            type1_s.play()
            for_item=1
        elif(hit.type==2):
            type2_s.play()
            for_item=1
        elif(hit.type==3):
            for_item=1
            ran=random.randint(0,100)
            if(ran<=85):
                type3_s.play()
            else:
                random.choice(expl_sounds).play()
        else:
            random.choice(expl_sounds).play()
        expl = Explosion(hit.rect.center, 'lg')
        all_sprites.add(expl)
        if for_item > 0.7+(boss_time/90):
            pow = Pow(hit.rect.center,hit.type)
            all_sprites.add(pow)
            powerups.add(pow)

        newmob()
    else:
        missile_hit.play()
```

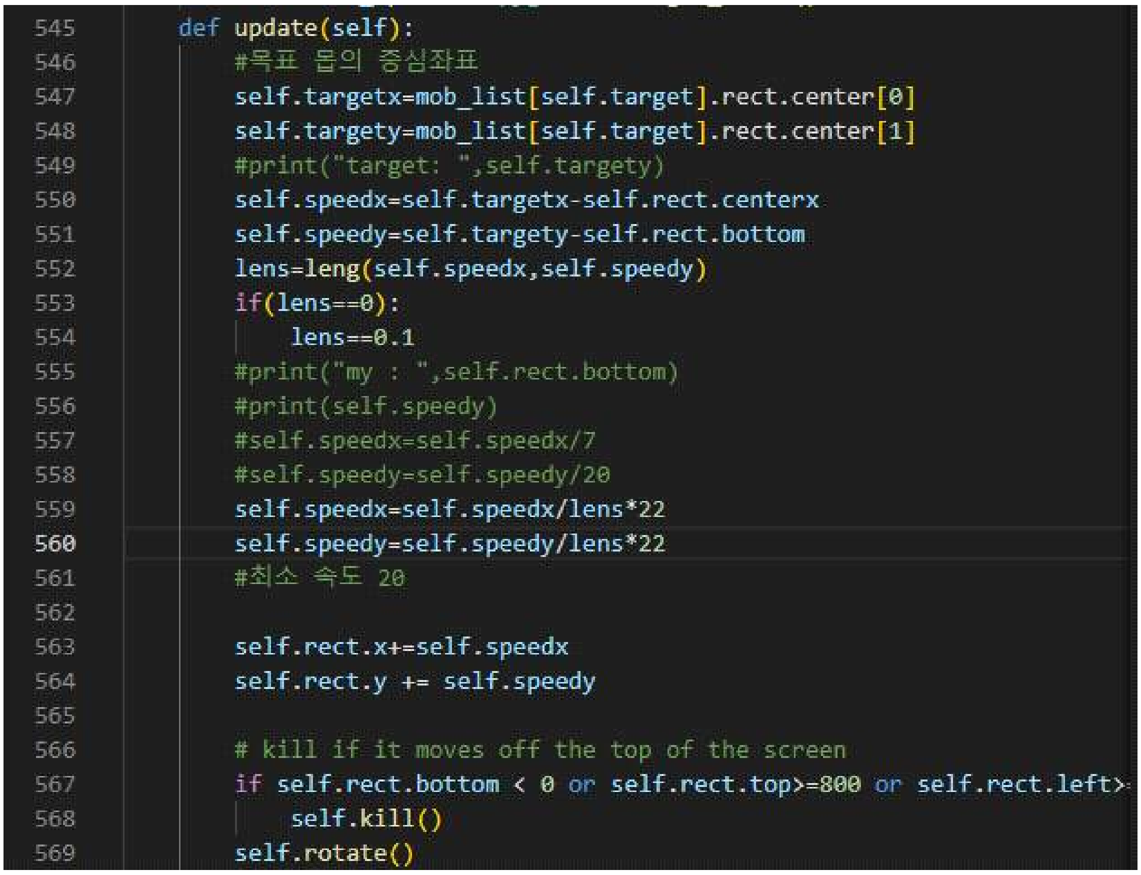
Again in the game loop, this time the missile collides with the mob. The missile is a guided missile that automatically aims and fires at a monster, so has a feature of finding a target and flying with a rotated image to match the target.
Hitting the missile added a special explosion sound.





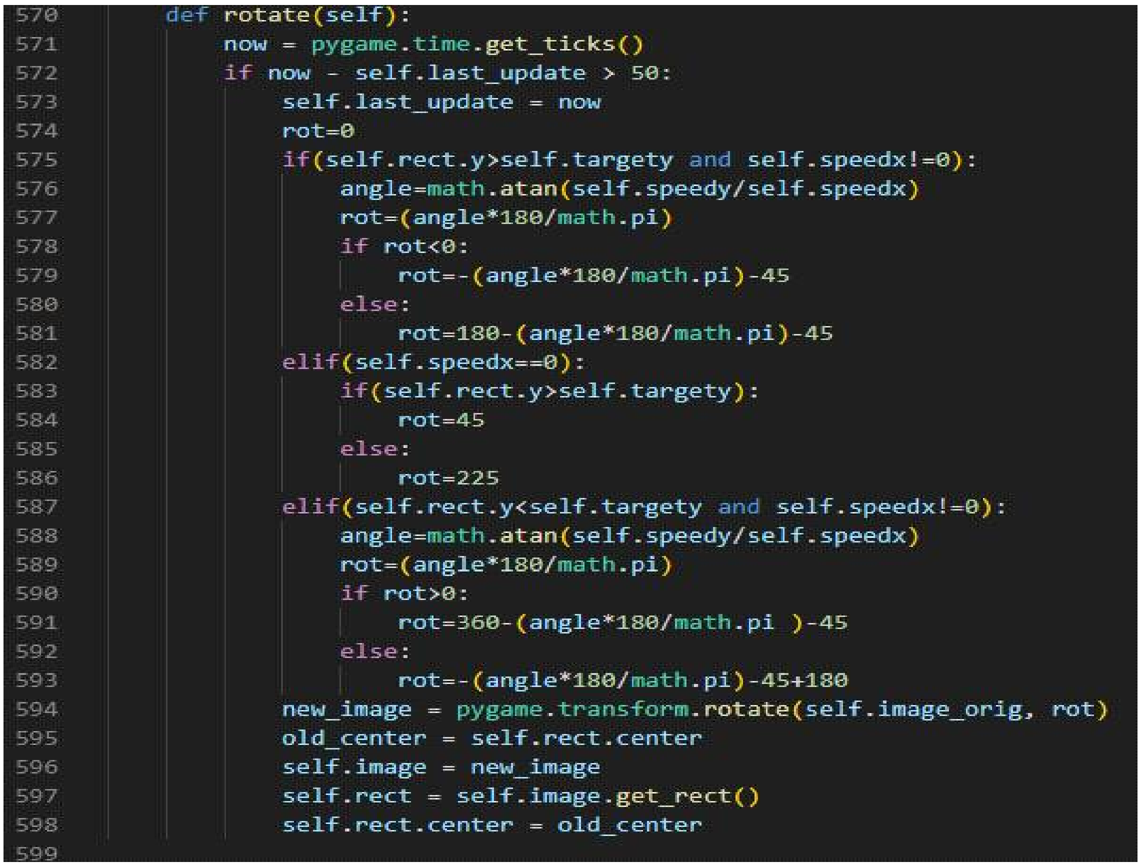
```python
class Missile(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image_orig = missile_img
        self.image=self.image_orig.copy()

        self.image.set_colorkey(BLACK)

        self.rect = self.image.get_rect()
        c=random.randint(0,len(mob_list)-1)
        '''for i in mob_list:
            if i.rect.center[1]<y:
                break
            else:
                c+=1
        if(c==len(mob_list)):
            c-=1  '''
        self.target=c
        #이거 플레이어 좌표임.
        self.rect.bottom = y
        self.rect.centerx = x


        self.targetx=mob_list[self.target].rect.center[0]
        self.targety=mob_list[self.target].rect.center[1]
        self.speedy = 0
        self.speedx= 0
#회전 주기용
        self.last_update = pygame.time.get_ticks()
```

Since there is a target similar to a monster bullet, targetx and targety are randomly selected from the monster list and the monster is set as a target. At this time, a random index is selected from the monster list. The reason is that if a monster dies while the missile is flying, a new monster is added(since dead monster is removed and new monster is appended) at that index, so we aim for that. Also, since the location is continuously tracked, speedx and speedy must be continuously changed in the update.



```python
    def update(self):
        #목표 몹의 중심좌표
        self.targetx=mob_list[self.target].rect.center[0]
        self.targety=mob_list[self.target].rect.center[1]
        #print("target: ",self.targety)
        self.speedx=self.targetx-self.rect.centerx
        self.speedy=self.targety-self.rect.bottom
        lens=leng(self.speedx,self.speedy)
        if(lens==0):
            lens==0.1
        #print("my : ",self.rect.bottom)
        #print(self.speedy)
        #self.speedx=self.speedx/7
        #self.speedy=self.speedy/20
        self.speedx=self.speedx/lens*22
        self.speedy=self.speedy/lens*22
        #최소 속도 20

        self.rect.x+=self.speedx
        self.rect.y += self.speedy

        # kill if it moves off the top of the screen
        if self.rect.bottom < 0 or self.rect.top>=800 or self.rect.left>
            self.kill()
        self.rotate()
```

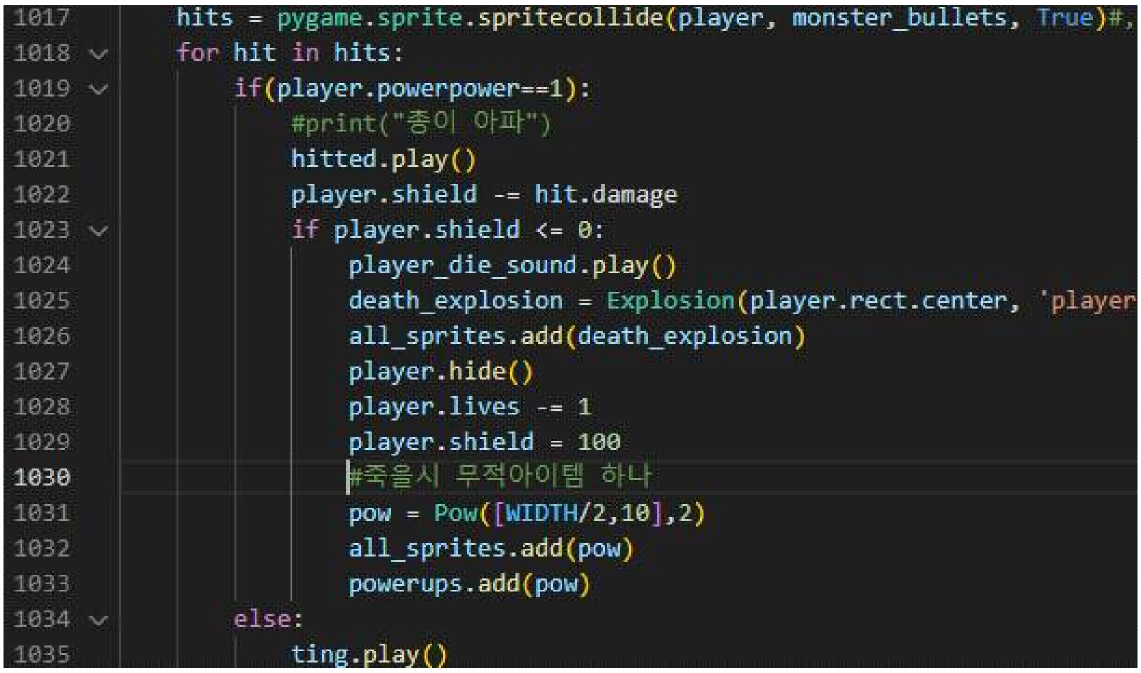
In the update, similar to the monster bullet, the direction vector was continuously created, and the norm size was divided and multiplied by 22 to make it fly slightly slower. Now the perfect guided missile is flying. The remaining part is to rotate the image according to the flying direction. It is in rotate.



```python
    def rotate(self):
        now = pygame.time.get_ticks()
        if now - self.last_update > 50:
            self.last_update = now
            rot=0
            if(self.rect.y>self.targety and self.speedx!=0):
                angle=math.atan(self.speedy/self.speedx)
                rot=(angle*180/math.pi)
                if rot<0:
                    rot=-(angle*180/math.pi)-45
                else:
                    rot=180-(angle*180/math.pi)-45
            elif(self.speedx==0):
                if(self.rect.y>self.targety):
                    rot=45
                else:
                    rot=225
            elif(self.rect.y<self.targety and self.speedx!=0):
                angle=math.atan(self.speedy/self.speedx)
                rot=(angle*180/math.pi)
                if rot>0:
                    rot=360-(angle*180/math.pi )-45
                else:
                    rot=-(angle*180/math.pi)-45+180
            new_image = pygame.transform.rotate(self.image_orig, rot)
            old_center = self.rect.center
            self.image = new_image
            self.rect = self.image.get_rect()
            self.rect.center = old_center
```

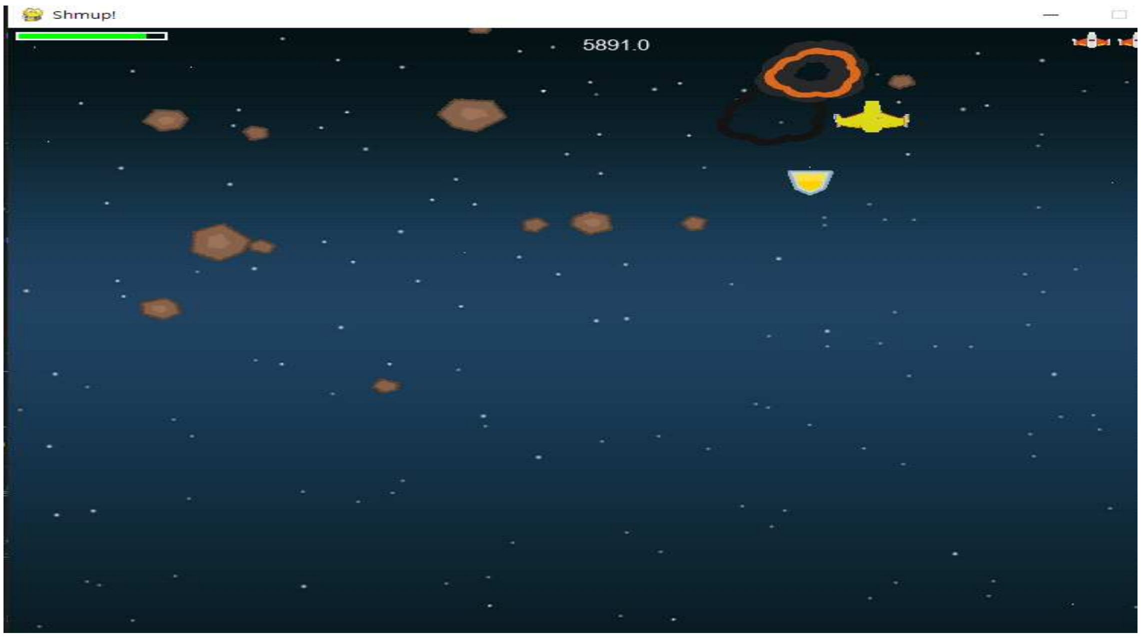
Rotation, of course, uses speedx and speedy, that is, the direction vector. The image has already been rotated by 45 degrees, and unlike monster bullets, it can fly backwards, so the angle of rotation was obtained by dividing the three cases. For example, if the missile is below the mob, you can find the angle for each case, such as subtracting 45 degrees from the tilt angle. Since the image continues to rotate, the original image is left, and if you rotate it according to the angle using this original image and make it self.image, the rotated image is drawn at the time of drawing.

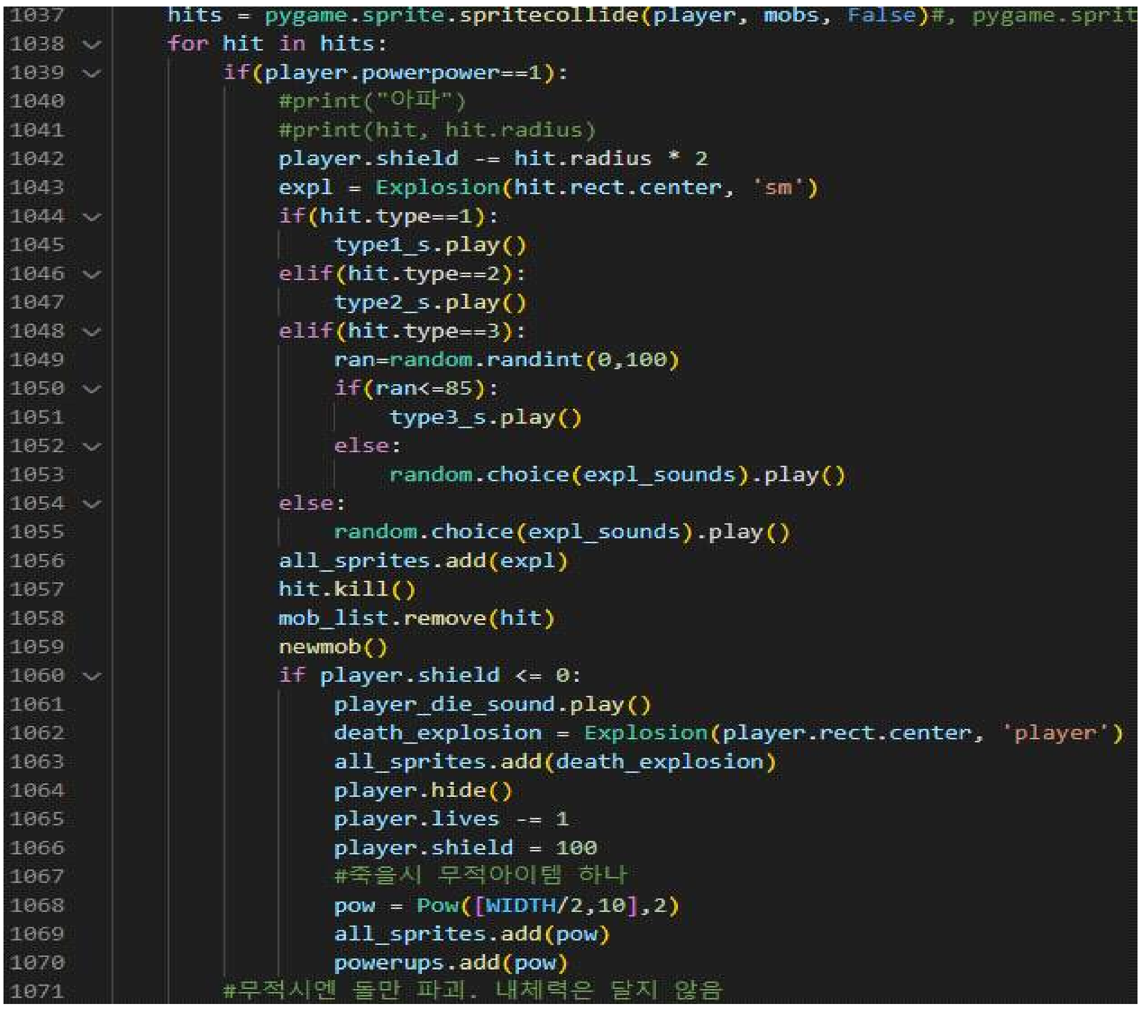
Now back to the while loop



```
    hits = pygame.sprite.spritecollide(player, monster_bullets, True)#,
    for hit in hits:
        if(player.powerpower==1):
            #print("총이 아파")
            hitted.play()
            player.shield -= hit.damage
            if player.shield <= 0:
                player_die_sound.play()
                death_explosion = Explosion(player.rect.center, 'player
                all_sprites.add(death_explosion)
                player.hide()
                player.lives -= 1
                player.shield = 100
                #죽을시 무적아이템 하나
                pow = Pow([WIDTH/2,10],2)
                all_sprites.add(pow)
                powerups.add(pow)
        else:
            ting.play()
```

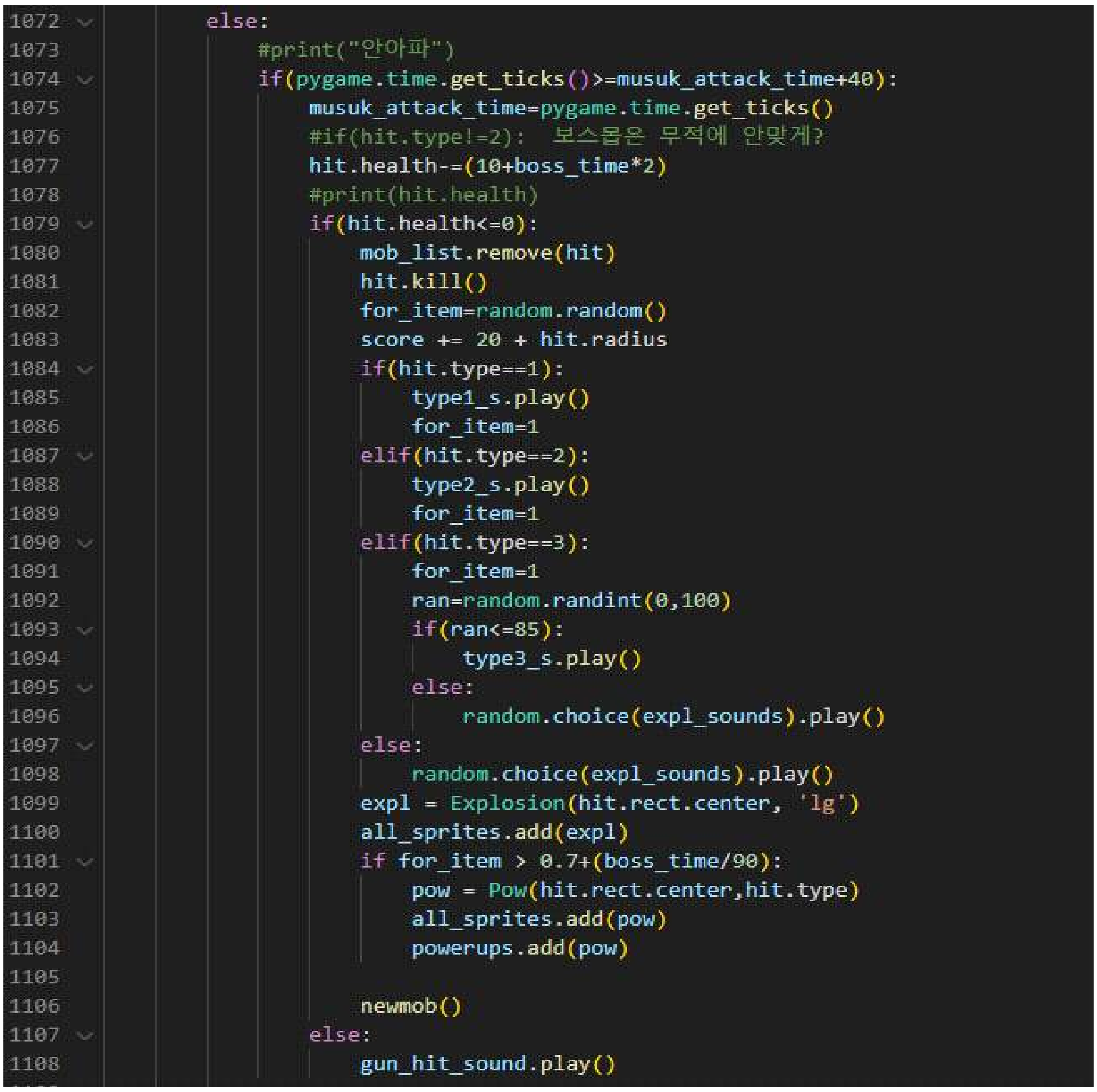
This part detects the collision between the monster's bullet and the player. The powerpower variable is a variable that determines whether the player is currently invincible, powerpower=1 means not invincible. When invincible, the sound of bullets bouncing, if not invincible, the player's shield (stamina) is reduced and if their health becomes 0, the death of the flare, and an invincibility item is dropped. The player class will be explained later. The picture below shows the player invincible. Displays player images of random colors (red, green, blue, yellow).





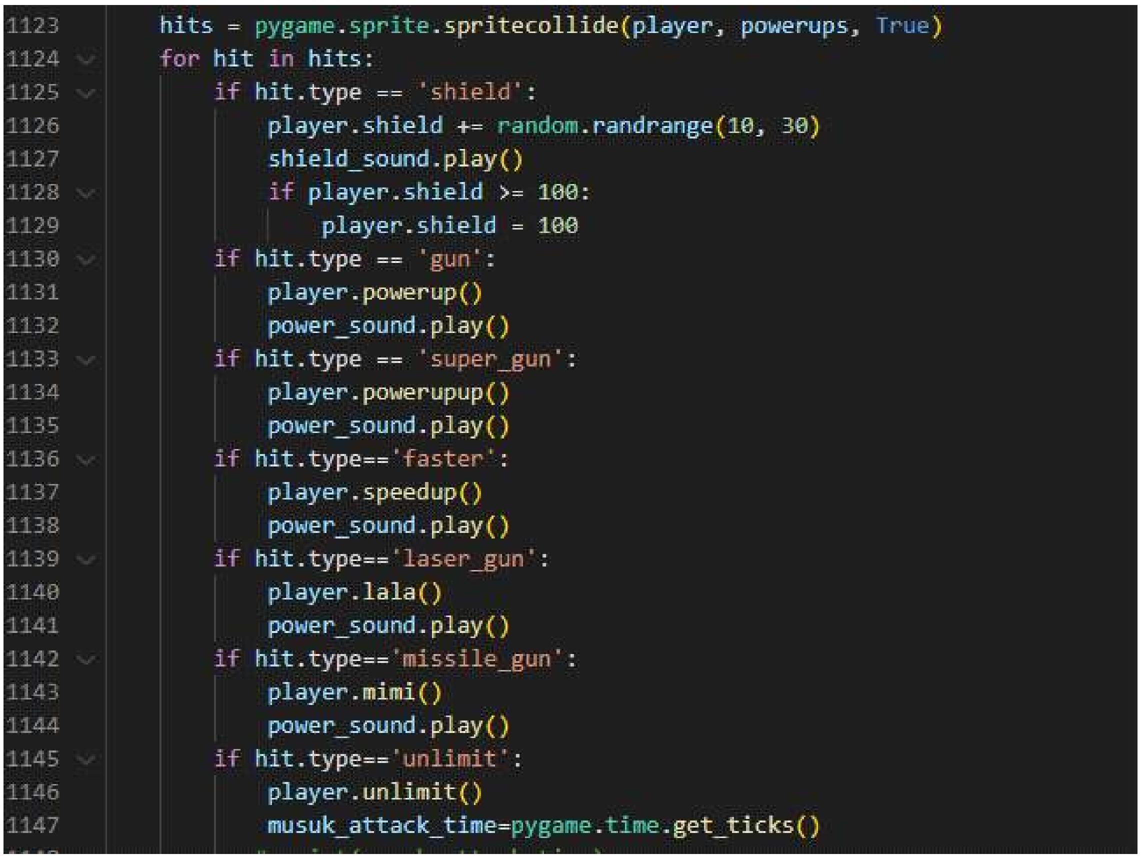
```python
    hits = pygame.sprite.spritecollide(player, mobs, False)#, pygame.sprit
    for hit in hits:
        if(player.powerpower==1):
            #print("아파")
            #print(hit, hit.radius)
            player.shield -= hit.radius * 2
            expl = Explosion(hit.rect.center, 'sm')
            if(hit.type==1):
                type1_s.play()
            elif(hit.type==2):
                type2_s.play()
            elif(hit.type==3):
                ran=random.randint(0,100)
                if(ran<=85):
                    type3_s.play()
                else:
                    random.choice(expl_sounds).play()
            else:
                random.choice(expl_sounds).play()
            all_sprites.add(expl)
            hit.kill()
            mob_list.remove(hit)
            newmob()
            if player.shield <= 0:
                player_die_sound.play()
                death_explosion = Explosion(player.rect.center, 'player')
                all_sprites.add(death_explosion)
                player.hide()
                player.lives -= 1
                player.shield = 100
                #죽을시 무적아이템 하나
                pow = Pow([WIDTH/2,10],2)
                all_sprites.add(pow)
                powerups.add(pow)
        #무적시엔 돌만 파괴. 내체력은 닳지 않음
```

This part is the part that identifies the collision between the player and the monster. If a monster collides while the player is not invincible, the monster will immediately explode and player take damage. The code is long, but I'll skip the details because it's just a mix of all the previous parts.



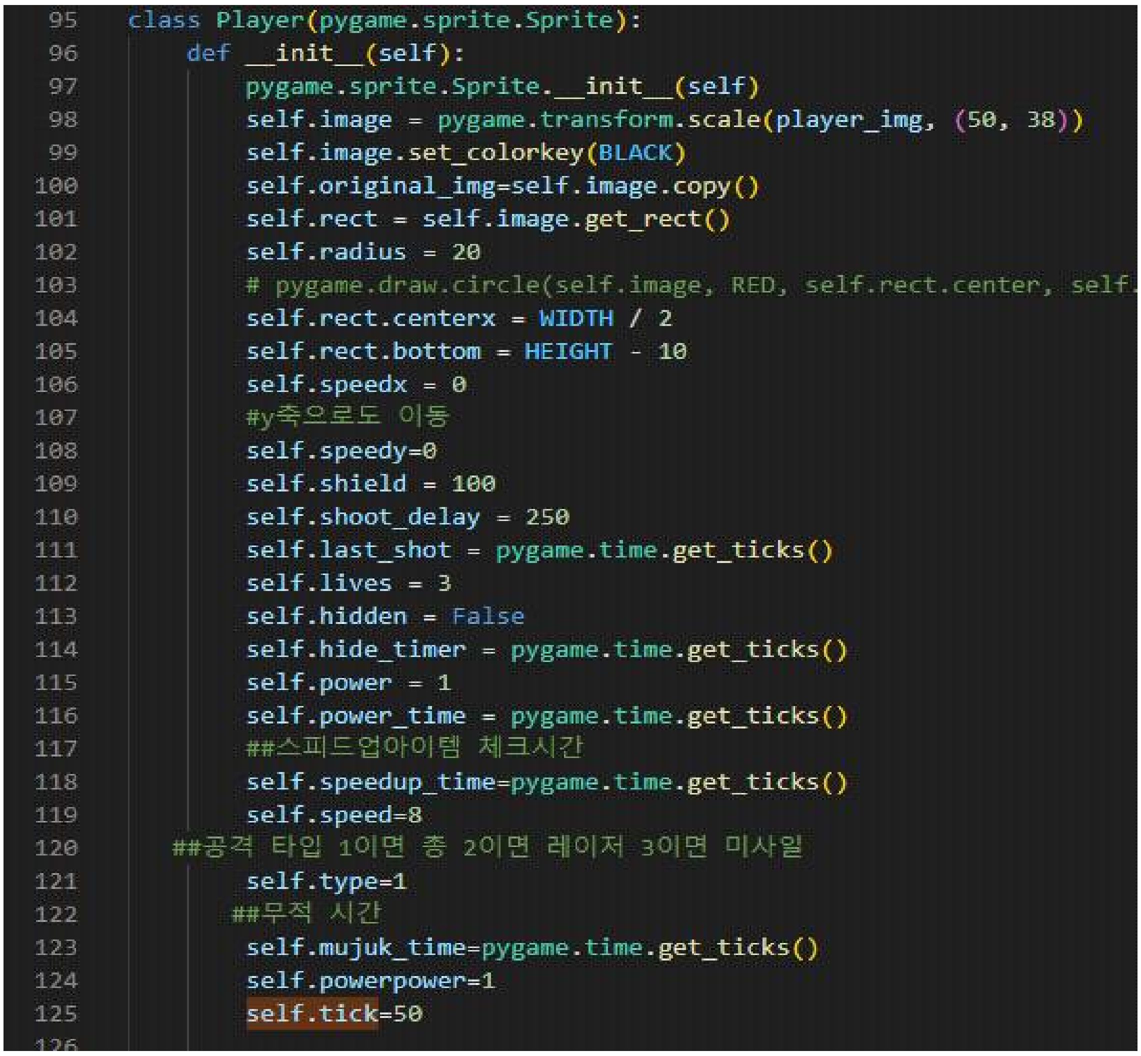
```python
        else:
            #print("안아파")
            if(pygame.time.get_ticks()>=musuk_attack_time+40):
                musuk_attack_time=pygame.time.get_ticks()
                #if(hit.type!=2):  보스몹은 무적에 안맞게?
                hit.health-=(10+boss_time*2)
                #print(hit.health)
                if(hit.health<=0):
                    mob_list.remove(hit)
                    hit.kill()
                    for_item=random.random()
                    score += 20 + hit.radius
                    if(hit.type==1):
                        type1_s.play()
                        for_item=1
                    elif(hit.type==2):
                        type2_s.play()
                        for_item=1
                    elif(hit.type==3):
                        for_item=1
                        ran=random.randint(0,100)
                        if(ran<=85):
                            type3_s.play()
                        else:
                            random.choice(expl_sounds).play()
                    else:
                        random.choice(expl_sounds).play()
                    expl = Explosion(hit.rect.center, 'lg')
                    all_sprites.add(expl)
                    if for_item > 0.7+(boss_time/90):
                        pow = Pow(hit.rect.center,hit.type)
                        all_sprites.add(pow)
                        powerups.add(pow)

                    newmob()
                else:
                    gun_hit_sound.play()
```

If the player is invincible, use musak_attack_time to damage the mob every 40 ticks, and if the mob died, it's treated the same as if it died with previous weapons.



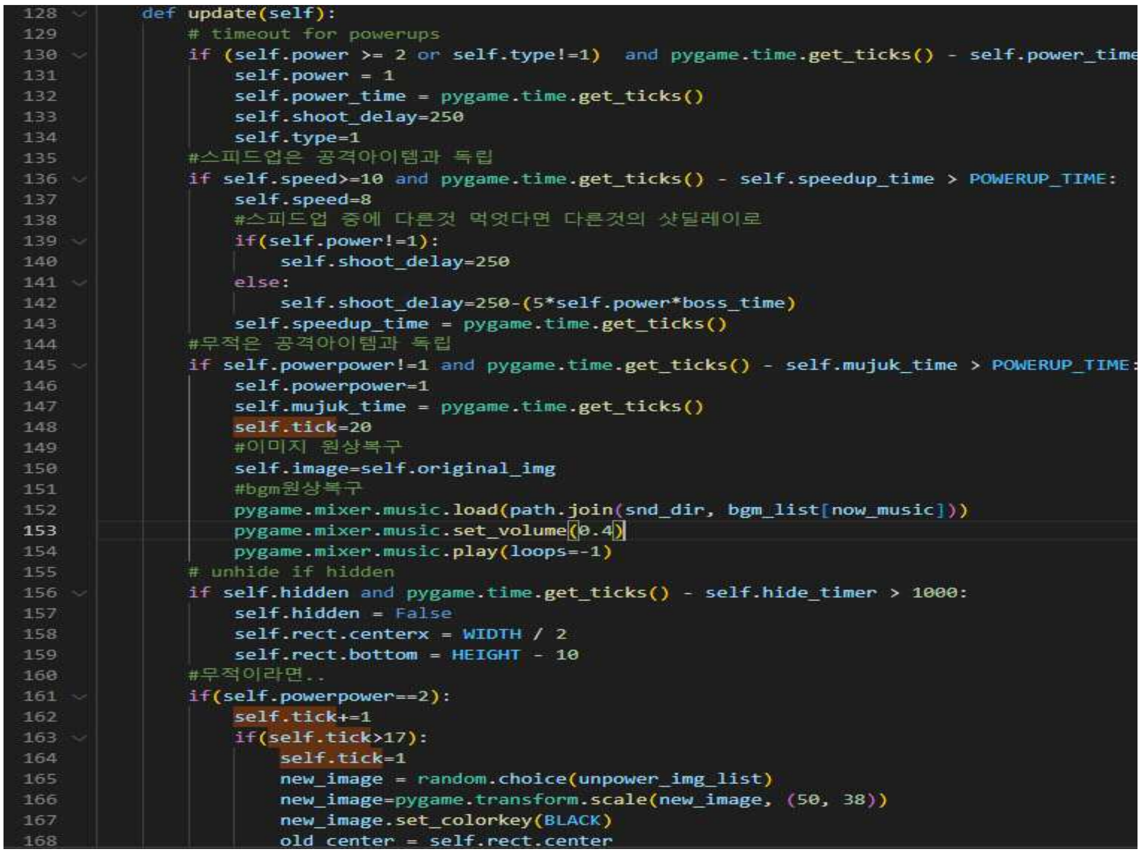
```
    hits = pygame.sprite.spritecollide(player, powerups, True)
    for hit in hits:
        if hit.type == 'shield':
            player.shield += random.randrange(10, 30)
            shield_sound.play()
            if player.shield >= 100:
                player.shield = 100
        if hit.type == 'gun':
            player.powerup()
            power_sound.play()
        if hit.type == 'super_gun':
            player.powerupup()
            power_sound.play()
        if hit.type=='faster':
            player.speedup()
            power_sound.play()
        if hit.type=='laser_gun':
            player.lala()
            power_sound.play()
        if hit.type=='missile_gun':
            player.mimi()
            power_sound.play()
        if hit.type=='unlimit':
            player.unlimit()
            musuk_attack_time=pygame.time.get_ticks()
```

This is the final part of the collision detection using sprites, the item and player collision detection part. Whenever each of the 7 items is eaten, the player executes each function and creates a situation suitable for the item. At this time, the shield is just a simple item that restores the player's physical strength. Before explaining the remaining six cases, let's explain the player class.

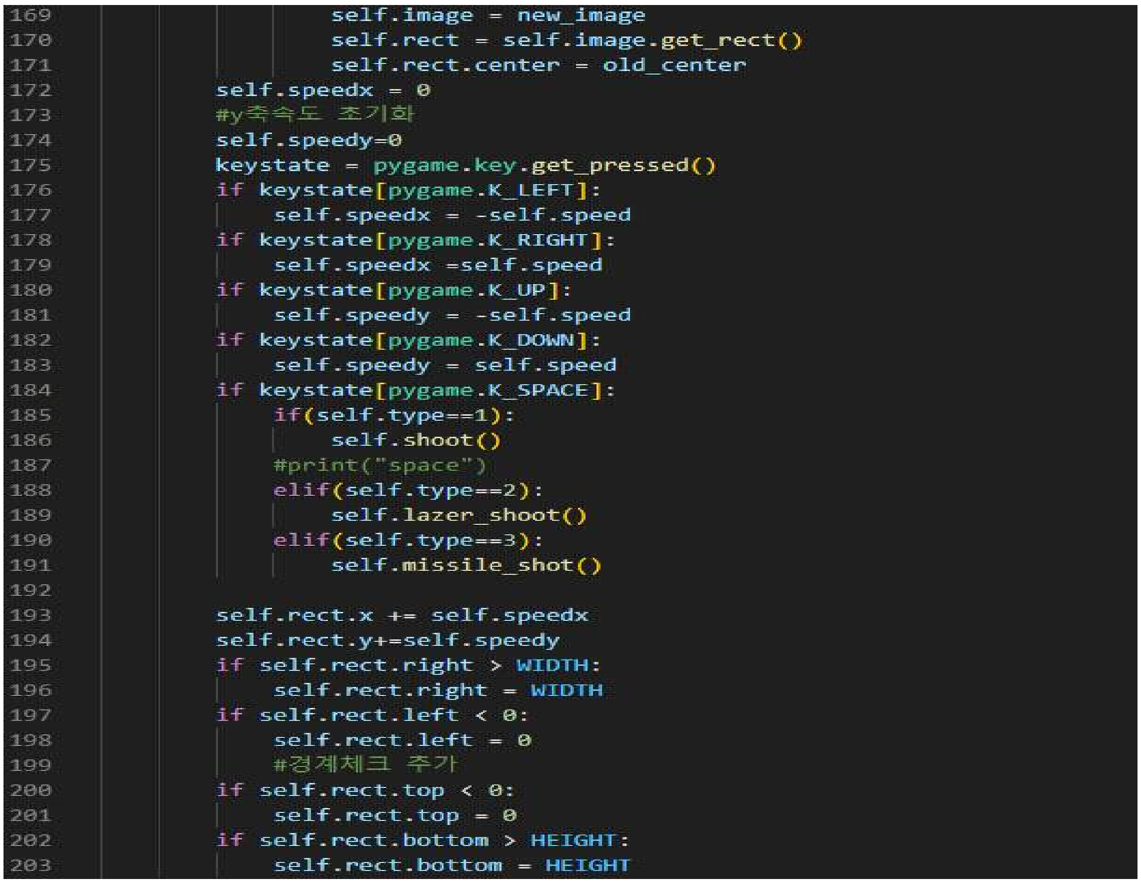


```
 95  class Player(pygame.sprite.Sprite):
 96      def __init__(self):
 97          pygame.sprite.Sprite.__init__(self)
 98          self.image = pygame.transform.scale(player_img, (50, 38))
 99          self.image.set_colorkey(BLACK)
100          self.original_img=self.image.copy()
101          self.rect = self.image.get_rect()
102          self.radius = 20
103          # pygame.draw.circle(self.image, RED, self.rect.center, self.
104          self.rect.centerx = WIDTH / 2
105          self.rect.bottom = HEIGHT - 10
106          self.speedx = 0
107          #y축으로도 이동
108          self.speedy=0
109          self.shield = 100
110          self.shoot_delay = 250
111          self.last_shot = pygame.time.get_ticks()
112          self.lives = 3
113          self.hidden = False
114          self.hide_timer = pygame.time.get_ticks()
115          self.power = 1
116          self.power_time = pygame.time.get_ticks()
117          ##스피드업아이템 체크시간
118          self.speedup_time=pygame.time.get_ticks()
119          self.speed=8
120      ##공격 타입 1이면 총 2이면 레이저 3이면 미사일
121          self.type=1
122         ##무적 시간
123          self.mujuk_time=pygame.time.get_ticks()
124          self.powerpower=1
125          self.tick=50
126
```

This is the player's constructor. The meaning of the variable name up to line 114 is the same, and the explanation will be omitted because it is the code that existed before. Power is a variable that determines how many bullets are fired at once when shooting a gun. It is basically one shot(1), and increases to 3 when eating an item. power_time is a variable to check the duration of an attack item(laser, missile, 3bullet ...). Simialy, speedup_time and mujuk time are variable to check duation of each item(speedup, invincibility). speed means player speed, and it can increase. type means attack type. If 1, it fires bullet, if 2, fires laser, if3, it fires missile. As mentioned, powerpower is a variable that checks invincibility, and tick is a variable that checks the period when the player's image changes colorfully when invincible.



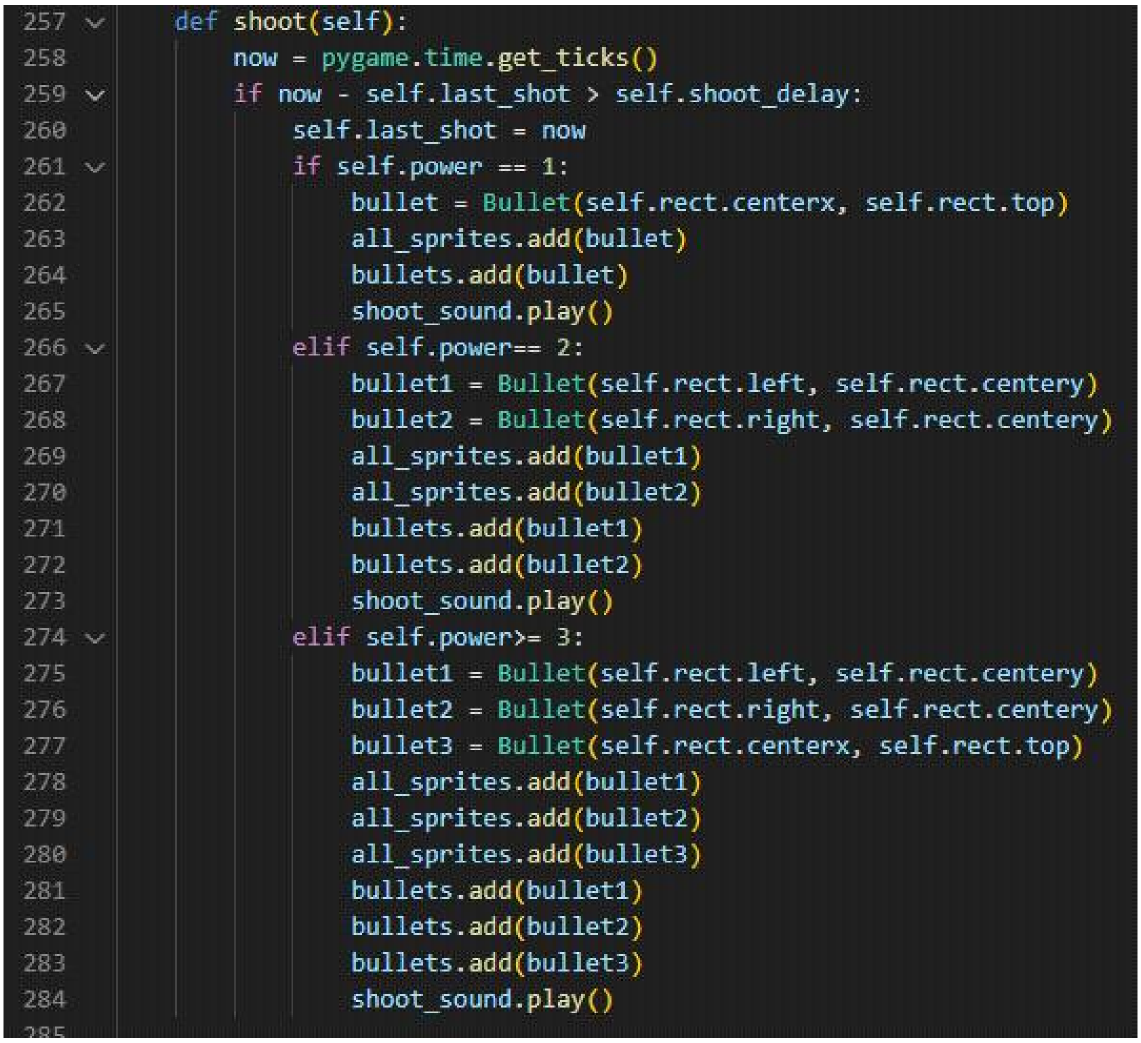
```python
    def update(self):
        # timeout for powerups
        if (self.power >= 2 or self.type!=1)  and pygame.time.get_ticks() - self.power_time
            self.power = 1
            self.power_time = pygame.time.get_ticks()
            self.shoot_delay=250
            self.type=1
        #스피드업은 공격아이템과 독립
        if self.speed>=10 and pygame.time.get_ticks() - self.speedup_time > POWERUP_TIME:
            self.speed=8
            #스피드업 중에 다른것 먹엇다면 다른것의 샷딜레이로
            if(self.power!=1):
                self.shoot_delay=250
            else:
                self.shoot_delay=250-(5*self.power*boss_time)
            self.speedup_time = pygame.time.get_ticks()
        #무적은 공격아이템과 독립
        if self.powerpower!=1 and pygame.time.get_ticks() - self.mujuk_time > POWERUP_TIME:
            self.powerpower=1
            self.mujuk_time = pygame.time.get_ticks()
            self.tick=20
            #이미지 원상복구
            self.image=self.original_img
            #bgm원상복구
            pygame.mixer.music.load(path.join(snd_dir, bgm_list[now_music]))
            pygame.mixer.music.set_volume(0.4)
            pygame.mixer.music.play(loops=-1)
        # unhide if hidden
        if self.hidden and pygame.time.get_ticks() - self.hide_timer > 1000:
            self.hidden = False
            self.rect.centerx = WIDTH / 2
            self.rect.bottom = HEIGHT - 10
        #무적이라면..
        if(self.powerpower==2):
            self.tick+=1
            if(self.tick>17):
                self.tick=1
                new_image = random.choice(unpower_img_list)
                new_image=pygame.transform.scale(new_image, (50, 38))
                new_image.set_colorkey(BLACK)
                old_center = self.rect.center
```



```python
                self.image = new_image
                self.rect = self.image.get_rect()
                self.rect.center = old_center
        self.speedx = 0
        #y축속도 초기화
        self.speedy=0
        keystate = pygame.key.get_pressed()
        if keystate[pygame.K_LEFT]:
            self.speedx = -self.speed
        if keystate[pygame.K_RIGHT]:
            self.speedx =self.speed
        if keystate[pygame.K_UP]:
            self.speedy = -self.speed
        if keystate[pygame.K_DOWN]:
            self.speedy = self.speed
        if keystate[pygame.K_SPACE]:
            if(self.type==1):
                self.shoot()
            #print("space")
            elif(self.type==2):
                self.lazer_shoot()
            elif(self.type==3):
                self.missile_shot()

        self.rect.x += self.speedx
        self.rect.y+=self.speedy
        if self.rect.right > WIDTH:
            self.rect.right = WIDTH
        if self.rect.left < 0:
            self.rect.left = 0
            #경계체크 추가
        if self.rect.top < 0:
            self.rect.top = 0
        if self.rect.bottom > HEIGHT:
            self.rect.bottom = HEIGHT
```

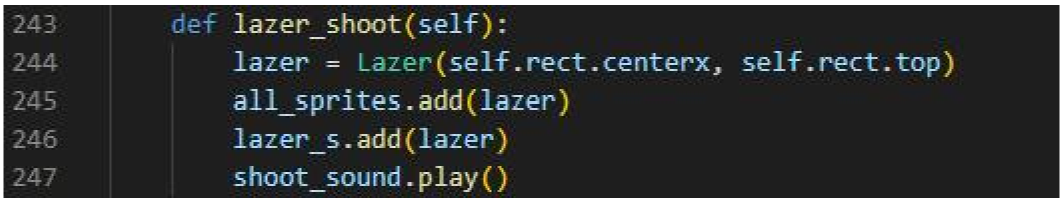
All parts of the upper part are updates, which control everything such as the player's movement and behavior. Up to line 136, when an attack item is eaten, it is returned to its original state after a certain period of time. Up to line 144 is the part that restores the increased shoot_delay and movement speed when a speed-up item is eaten. Up to line 155, it restores the status of the invincible item. It restores the changed player's image, and if you eat the invincible item, the bgm changes. This changed bgm is also restored to its original state. In addition, when invincible (powerpower==2), the player's image is randomly picked from the unpower_img_list and changed every 17 ticks, also exists in lines 161-171. The rest is for your keyboard input. If you press the direction keys, you can freely move up, down, left, right, up and down on the screen, and if you press the space, you can fire a gun, laser, or missile using the method, depending on the type.
Now let's take a look at the shoot functions according to the type.

If type is 1.



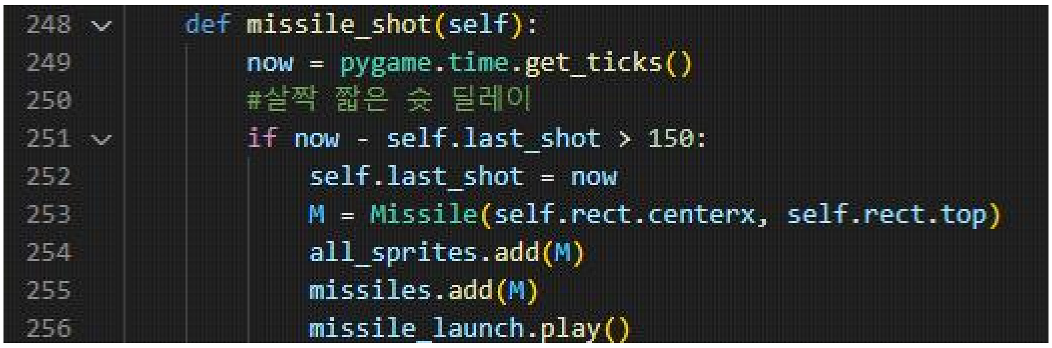
```
    def shoot(self):
        now = pygame.time.get_ticks()
        if now - self.last_shot > self.shoot_delay:
            self.last_shot = now
            if self.power == 1:
                bullet = Bullet(self.rect.centerx, self.rect.top)
                all_sprites.add(bullet)
                bullets.add(bullet)
                shoot_sound.play()
            elif self.power== 2:
                bullet1 = Bullet(self.rect.left, self.rect.centery)
                bullet2 = Bullet(self.rect.right, self.rect.centery)
                all_sprites.add(bullet1)
                all_sprites.add(bullet2)
                bullets.add(bullet1)
                bullets.add(bullet2)
                shoot_sound.play()
            elif self.power>= 3:
                bullet1 = Bullet(self.rect.left, self.rect.centery)
                bullet2 = Bullet(self.rect.right, self.rect.centery)
                bullet3 = Bullet(self.rect.centerx, self.rect.top)
                all_sprites.add(bullet1)
                all_sprites.add(bullet2)
                all_sprites.add(bullet3)
                bullets.add(bullet1)
                bullets.add(bullet2)
                bullets.add(bullet3)
                shoot_sound.play()
```

Fires from one to three shots depending on power. I just added the part where power == 3, and this part made it possible to fire 3 bullets while making bullets in 3 places on the left, center and right at the same time.



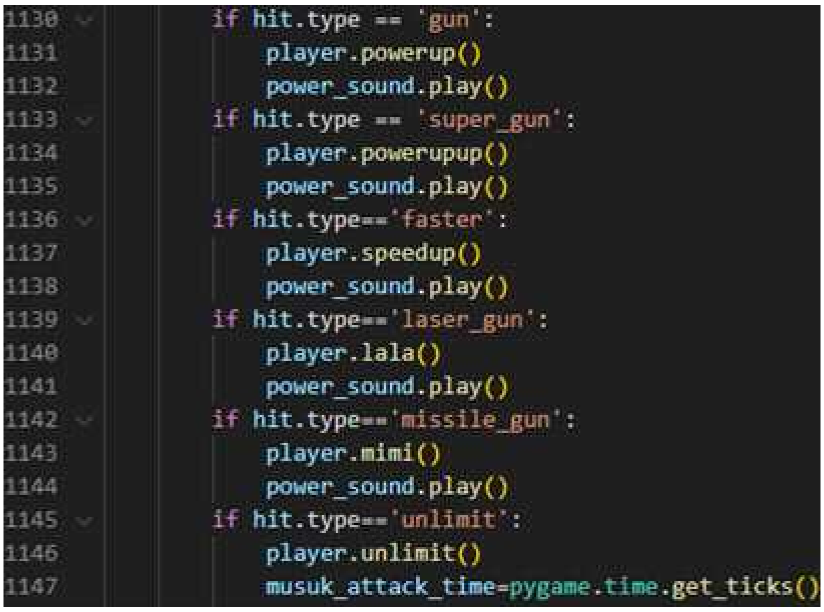
```python
    def lazer_shoot(self):
        lazer = Lazer(self.rect.centerx, self.rect.top)
        all_sprites.add(lazer)
        lazer_s.add(lazer)
        shoot_sound.play()
```

In case of type2, lazer_shoot just creates a lazer class whenever you press space. As explained earlier, a laser only lives for 2 ticks, so it will disappear immediately if you release your hand from space, and if you keep holding down space, it will fire continuously, i.e. it will look like a real laser.



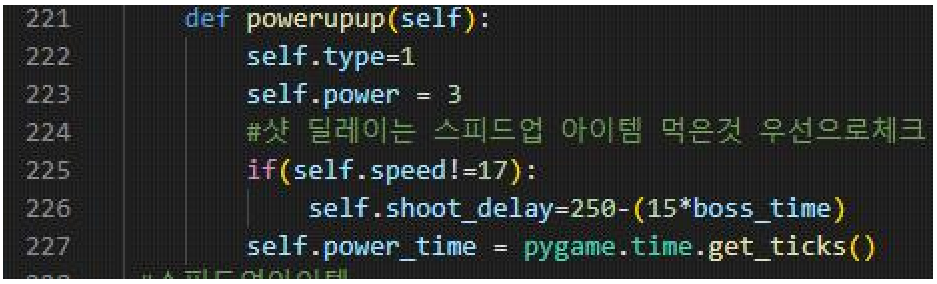
```python
    def missile_shot(self):
        now = pygame.time.get_ticks()
        #살짝 짧은 슛 딜레이
        if now - self.last_shot > 150:
            self.last_shot = now
            M = Missile(self.rect.centerx, self.rect.top)
            all_sprites.add(M)
            missiles.add(M)
            missile_launch.play()
```

In missile_shot of type 3, every 150 ticks (a normal gun has a delay of 250, so the missile fires faster), it creates a missile class and fires it.
Now, I will explain the function of the item that I skipped the explanation earlier.



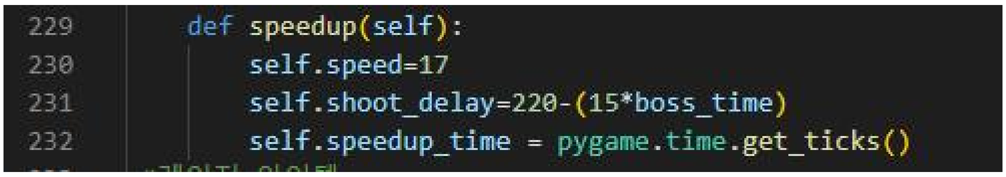
```python
            if hit.type == 'gun':
                player.powerup()
                power_sound.play()
            if hit.type == 'super_gun':
                player.powerupup()
                power_sound.play()
            if hit.type=='faster':
                player.speedup()
                power_sound.play()
            if hit.type=='laser_gun':
                player.lala()
                power_sound.play()
            if hit.type=='missile_gun':
                player.mimi()
                power_sound.play()
            if hit.type=='unlimit':
                player.unlimit()
                musuk_attack_time=pygame.time.get_ticks()
```

First is the ‘gun’. This is an existing item that fires two shots, that is, increases the player's power by 1.
The super gun is an item that fires three items, and it executes the powerupup function.



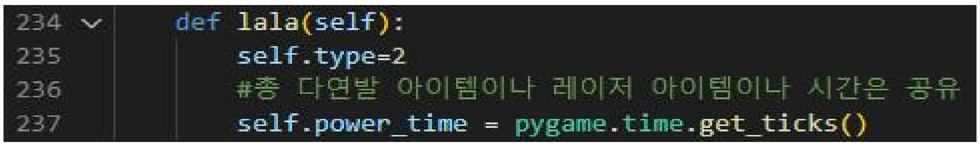
```python
    def powerupup(self):
        self.type=1
        self.power = 3
        #샷 딜레이는 스피드업 아이템 먹은것 우선으로체크
        if(self.speed!=17):
            self.shoot_delay=250-(15*boss_time)
        self.power_time = pygame.time.get_ticks()
```

It's simple, since it's a gun, type=1, and 3 shots, so power=3. When I made it, it seemed like the temp was too old, so I reduced the shot_delay. It reduces delay 15*boss time. The reason why self.speed!=17 is checked at this time is that the speedup item also reduces shot_delay, and since the speedup item reduces it to a greater extent, it is to maintain the delay if it is in a speedup state.
faster executes the speedup function.



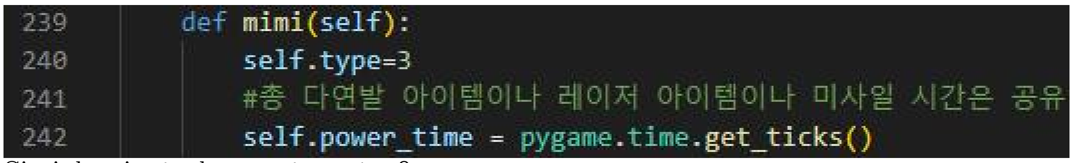
```python
    def speedup(self):
        self.speed=17
        self.shoot_delay=220-(15*boss_time)
        self.speedup_time = pygame.time.get_ticks()
```

Quite simply, it increases the player's speed, and also greatly reduces shot_delay.
laser_gun execute lala function.



```python
    def lala(self):
        self.type=2
        #총 다연발 아이템이나 레이저 아이템이나 시간은 공유
        self.power_time = pygame.time.get_ticks()
```

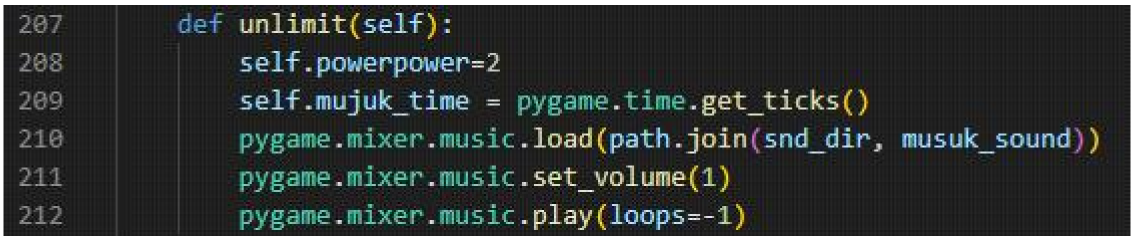
If you just set the type to 2, it will just fire the laser when you press space in update(). At this time, since the attack motion must change every time whenever attack item is eaten, power_time is shared and newly updated.
If you eat missile_gun,



```python
    def mimi(self):
        self.type=3
        #총 다연발 아이템이나 레이저 아이템이나 미사일 시간은 공유
        self.power_time = pygame.time.get_ticks()
```

Simialry just change type to 3

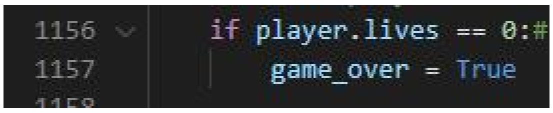
Finally, if you eat unlimit, it execute, unlimit function.



```
    def unlimit(self):
        self.powerpower=2
        self.mujuk_time = pygame.time.get_ticks()
        pygame.mixer.music.load(path.join(snd_dir, musuk_sound))
        pygame.mixer.music.set_volume(1)
        pygame.mixer.music.play(loops=-1)
```

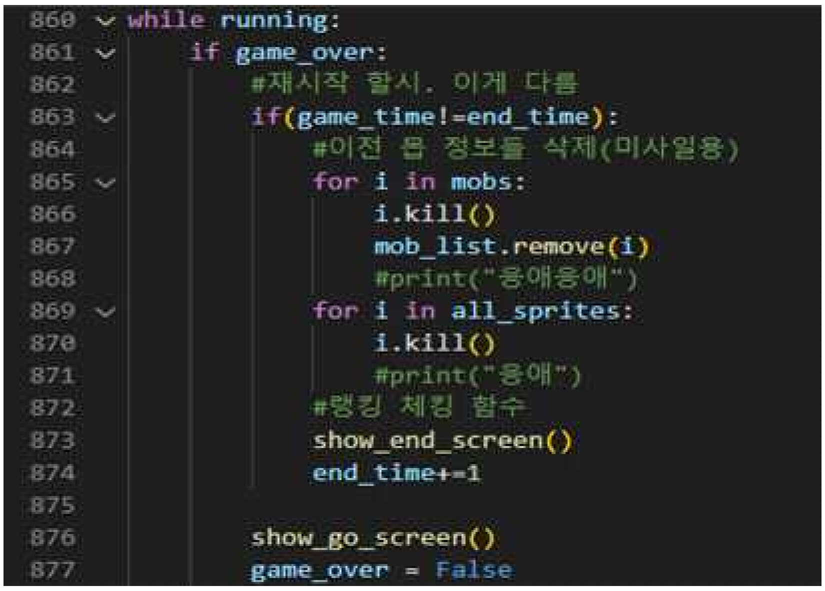
Of course, the invincible variable, POWERPOWER, is set to 2, and the bgm is specially changed when invincible is eaten. This bgm returns to its original state using bgm_list and now_music when the invincibility duration ends in update.

Now, I have finished all the parts used in the game (I will omit the small parts of writing letters, showing health bars, and flipping).



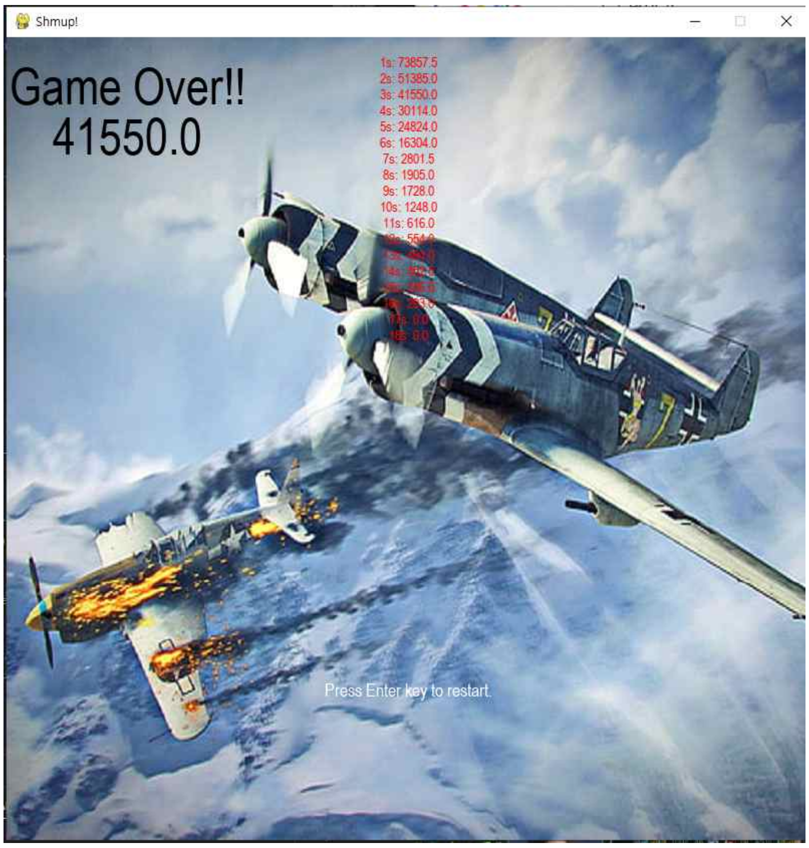
```
    if player.lives == 0:#
        game_over = True
```

When your lives run out, game_over becomes true, and you return to the first if statement.
I'll show you the first part of the game loop again.

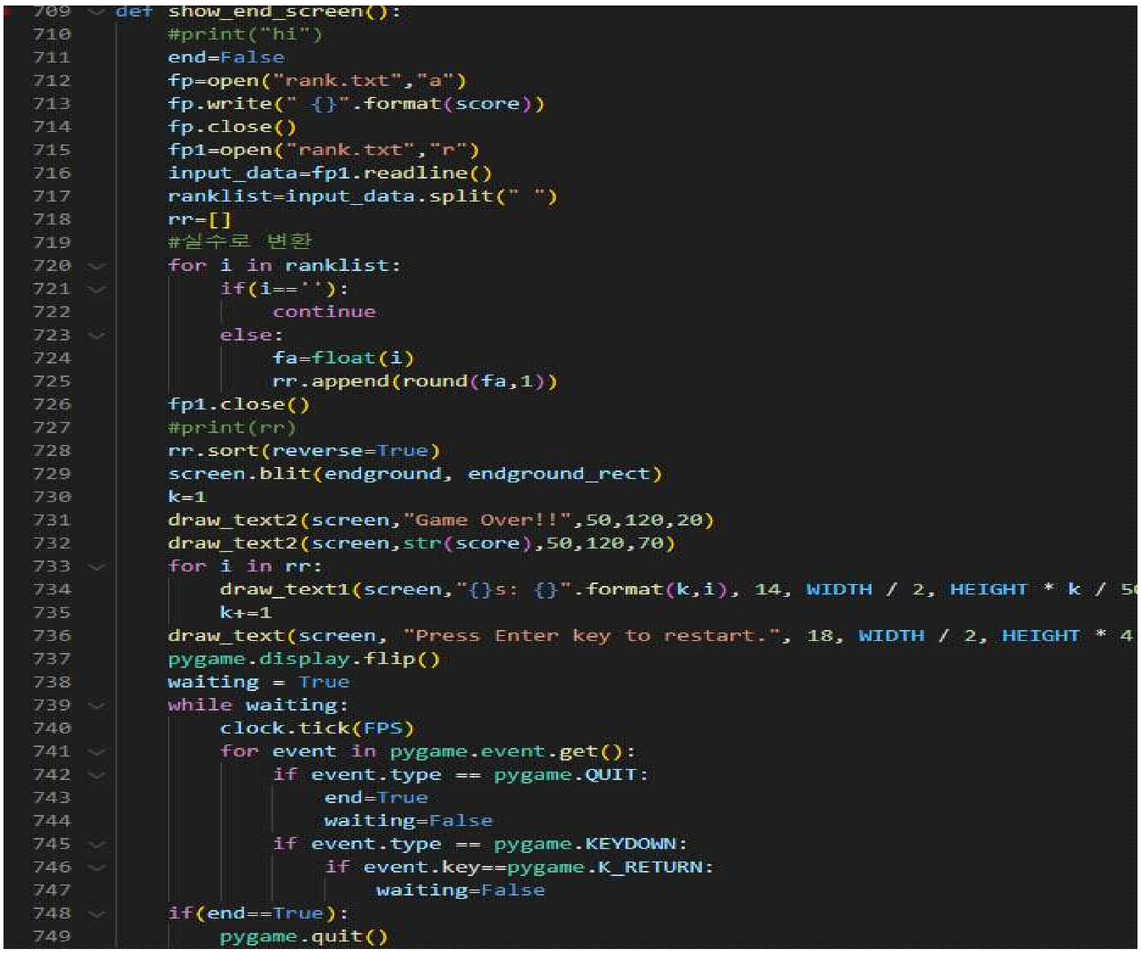


```
while running:
    if game_over:
        #재시작 할시. 이게 다름
        if(game_time!=end_time):
            #이전 몹 정보들 삭제(미사일용)
            for i in mobs:
                i.kill()
                mob_list.remove(i)
                #print("응애응애")
            for i in all_sprites:
                i.kill()
                #print("응애")
            #랭킹 체킹 함수
            show_end_screen()
            end_time+=1

        show_go_screen()
        game_over = False
```

Note that at this tiem, game_time is 1, but end_time is 0. I,e it goes to if statement. At this time, delete everything remaining in the sprite and mob_list, and now execute the show_end_screen function.



```
Game Over!!
41550.0

1s: 73857.5
2s: 51385.0
3s: 41550.0
4s: 30114.0
5s: 24824.0
6s: 16304.0
7s: 2801.5
8s: 1905.0
9s: 1728.0
10s: 1248.0
11s: 616.0
12s: 554.0
13s: [illegible]
14s: [illegible]
15s: [illegible]
16s: [illegible]
17s: 0.0
18s: 0.0

Press Enter key to restart.
```



```python
def show_end_screen():
    #print("hi")
    end=False
    fp=open("rank.txt","a")
    fp.write(" {}".format(score))
    fp.close()
    fp1=open("rank.txt","r")
    input_data=fp1.readline()
    ranklist=input_data.split(" ")
    rr=[]
    #실수로 변환
    for i in ranklist:
        if(i==''):
            continue
        else:
            fa=float(i)
            rr.append(round(fa,1))
    fp1.close()
    #print(rr)
    rr.sort(reverse=True)
    screen.blit(endground, endground_rect)
    k=1
    draw_text2(screen,"Game Over!!",50,120,20)
    draw_text2(screen,str(score),50,120,70)
    for i in rr:
        draw_text1(screen,"{}s: {}".format(k,i), 14, WIDTH / 2, HEIGHT * k / 5
        k+=1
    draw_text(screen, "Press Enter key to restart.", 18, WIDTH / 2, HEIGHT * 4
    pygame.display.flip()
    waiting = True
    while waiting:
        clock.tick(FPS)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                end=True
                waiting=False
            if event.type == pygame.KEYDOWN:
                if event.key==pygame.K_RETURN:
                    waiting=False
    if(end==True):
        pygame.quit()
```

This function shows the rankings on the screen in order from 1st. Open rank.txt, write the current score, and save (fp.close()). After that, open the file again, read the information in the rank file as it is, split it into ranklist, and save it. Here, since the str type is stored in the ranklist, it is rounded to the list called rr and stored as a float type. Finally, if you simply sort in descending order with rr.sort and output it as it is, the rankings up to now are displayed in descending order. This function cannot escape from the while loop until the game is turned off or Enter is pressed, and when Enter is pressed, it escapes, and now the show_go_screen function described at the beginning is executed and the game is run again.