Chapter 19

나를 돌려줘: 다시 우주 침략자 게임

1. 이미지 회전

- 이 장에서는 이미지를 회전하는 방법에 대해 알아보겠습니다.
- 우주 침략자 게임의 파이터 이미지를 회전시켜 봅시다.
- 먼저 File>New File을 눌러 새 파일을 하나 만들고 아래 코드를 씁니다.
- 실행하면 스크린 한가운데 놓인 파이터가 보입니다.

rotated.py

```
import pygame, sys
from pygame.locals import *
pygame.init()
clock = pygame.time.Clock()
screen = pygame.display.set_mode((1000,600))
fighter_image = pygame.image.load("images/fighter.png").convert()
fighter_image.set_colorkey((255,255,255))

class Fighter:
    def __init__(self):
        self.x = 450
        self.y = 270

    def draw(self):
        screen.blit(fighter_image,(self.x,self.y))
```

1. 이미지 회전

- 이 장에서는 이미지를 회전하는 방법에 대해 알아보겠습니다.
- 우주 침략자 게임의 파이터 이미지를 회전시켜 봅시다.
- 먼저 File>New File을 눌러 새 파일을 하나 만들고 아래 코드를 씁니다.
- 실행하면 스크린 한가운데 놓인 파이터가 보입니다.

```
fighter = Fighter()

while 1:
    clock.tick(60)

    for event in pygame.event.get():
        if event.type == QUIT:
            sys.exit()

    screen.fill((0,0,0))
    fighter.draw()

    pygame.display.update()
```

실행

2. turn() 함수

- turn() 함수를 파이터 클래스에 추가합니다.
- __init__() 함수 안에는 방향 변수를 추가합니다.
- 현재 파이터가 가리키는 방향은 위쪽인데, 이 방향을 0이라고 하겠습니다.
- 물론 아래쪽 방향을 0이라고 해도 상관없습니다.
- 악당들에서는 그렇게 했지요.
- 하지만 위쪽을 0으로 정해야 미사일 등의 방향을 정할 때 계산이 쉬워진다는 사실은 기억하세요.

```
class Fighter:
    def __init__(self):
        self.x = 450
        self.y = 270
        self.dir = 0

    def turn(self):
        if pressed_keys[K_a]:
            self.dir += 1
        if pressed_keys[K_z]:
            self.dir -= 1
```

2. turn() 함수

- turn() 함수가 dir에 양수를 더하면 파이터는 시계 반대 방향으로 회전합니다.
- while 루프 안에서 turn() 함수를 불러오는 것도 잊지 마세요.

```
while 1:
(중략)
    fighter.draw()
    fighter.turn()
```

2. turn() 함수

- 이번에는 60분법을 사용하겠습니다
- 파이게임의 rotate() 함수를 사용하려면 60분법으로 표시해야 하니까요
- 아까는 왜 라디안을 쓰라고 강조했냐고요?
- 일반적으로는 라디안을 사용하는 것이 좋으니까요
- 하지만 특정 그래픽 타입에서는 문제가 생길 수도 있습니다
- 파이썬은 라디안을, 그래픽 모듈인 파이게임에서는 60분법을 사용하는 걸 추천해요

3. 파이터 회전

- turn() 함수 안에서 눌린 키들 리스트 (pressed_keys)를 사용하기 때문에 게임 루프 안에서 반드시 이 리스트를 만들어야 합니다.
- 이 코드는 while 루프 안, 끝내기 섹션 바로 밑, screen.fill() 함수 바로 위에 씁니다.

```
while 1:
(중략)
    pressed_keys = pygame.key.get_pressed()
    screen.fill((0,0,0))
```

- self.dir의 값은 바뀌고 있지만, draw() 함수에서 self.dir을 사용하기 전까지 파이터는 돌지 않고 가만히 있을 것입니다.
- 파이터 이미지 표시는 파이터 클래스의 draw() 함수가 합니다.
- 회전된 파이터 이미지 표시는 파이게임 안의 rotate() 함수로 합니다.

3. 파이터 회전

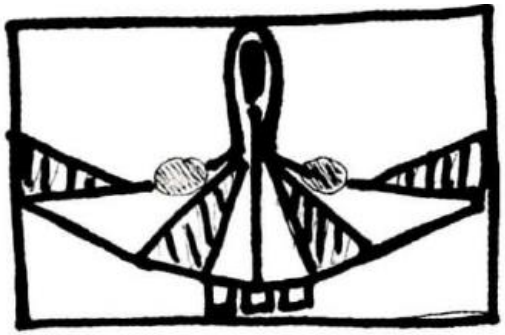

```
class Fighter:
(중략)
    def draw(self):
        rotated = pygame.transform.rotate(fighter_image,self.dir)
        screen.blit(fighter_image rotated,(self.x,self.y))
```

실행

- pygame.transform.rotate()에는 2개의 인자가 필요합니다.
- 첫 번째 인자는 회전 대상입니다. 여기서는 파이터 이미지지요.
- 두 번째 인자는 이미지 회전 각도입니다. 이것은 self.dir로 정합니다.
- 음영 표시된 코드 1번째 줄에서 이 함수의 실행 결과 회전된 이미지를 rotated변수에 저장합니다.
- 그 다음 screen.blit() 함수는 rotated 변수를 (self.x, self.y)에 표시합니다.
- 키보드를 누르면 파이터가 회전하는 거죠.

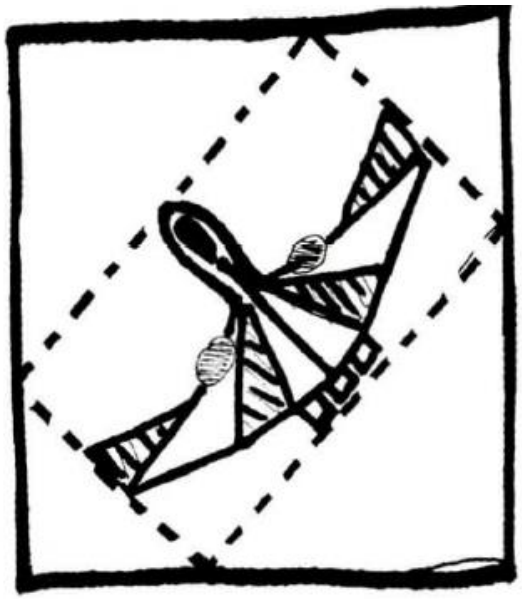
3. 파이터 회전

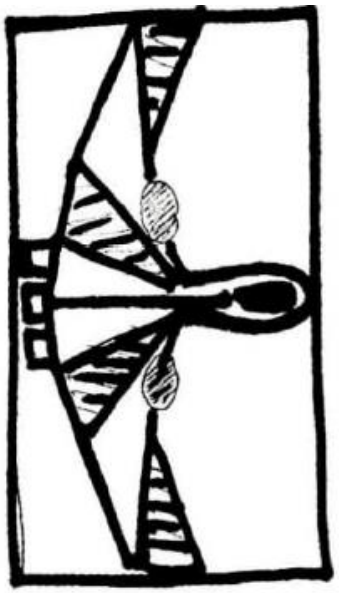
- 그런데 뭔가 이상하다고요?
- 사실 어떤 이미지든 파이게임은 항상 양 옆과 위아래에 선을 그어 가능한 최소 크기의 직사각형을 만든답니다.
- 그리고 이 직사각형의 가장 위 왼쪽에 이미지를 놓는 거예요.
- 파이터가 위로 올라가면 다음과 같은 직사각형이 생깁니다.



3. 파이터 회전

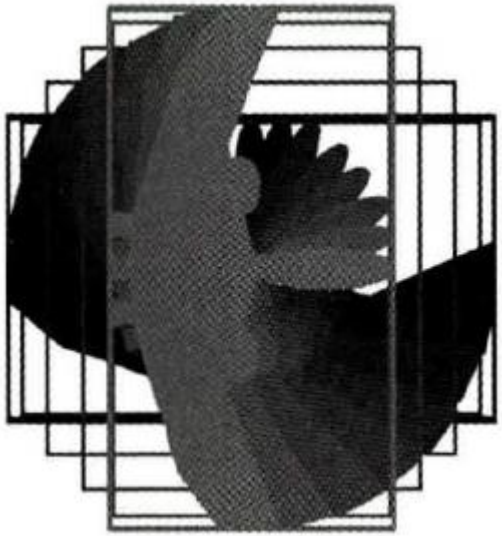
- 파이터가 회전하면, 파이게임이 만드는 직사각형의 모양과 크기가 달라집니다.





3. 파이터 회전

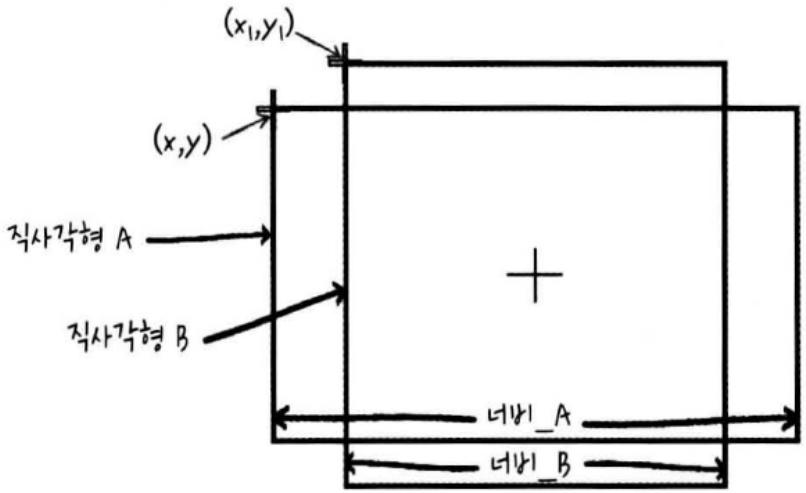
- 따라서 rotate() 함수를 사용해 시계 방향으로 회전시키면 아래 왼쪽처럼 보입니다.
- 우리가 원하는 건 오른쪽인데 말이죠.



3. 파이터 회전

- 오른쪽 같은 직사각형 A, B를 생각해봅시다.
- A의 가장 위 왼쪽의 좌표는 (x, y) 입니다.
- 위 그림처럼 A의 중심과 B의 중심이 겹치도록 B를 A 위에 놓습니다.
- A는 파이터 이미지입니다.
- B는 파이게임이 회전된 이미지 가장자리에 만드는 직사각형입니다.
- x_1은 x에 width_A/2를 더한 후 width_B/2를 뺀 값입니다.
- y좌표는 같은 일을 높이에 대해서 구합니다.

$$y_1 = y + Height\_A/2 - Height\_B/2$$



3. 파이터 회전

- 이 것은 draw() 함수에서 했던 것과 똑같습니다.
- (x, y)좌표를 (self.x, self.y)라고 생각해 보세요.
- 이렇게 하면 파이터가 앞에서 우리가 원했던 오른쪽 그림처럼 회전됩니다.

```
class Fighter:
(중략)
    def draw(self):
        rotated = pygame.transform.rotate(fighter_image,self.dir)
        screen.blit(rotated,(self.x+fighter_image.get_width()/2- rotated.get_width()/2,
self.y+fighter_image.get_height()/2-rotated.get_height()/2))
```

실행

4. 우주 침략자 게임으로 돌아가자

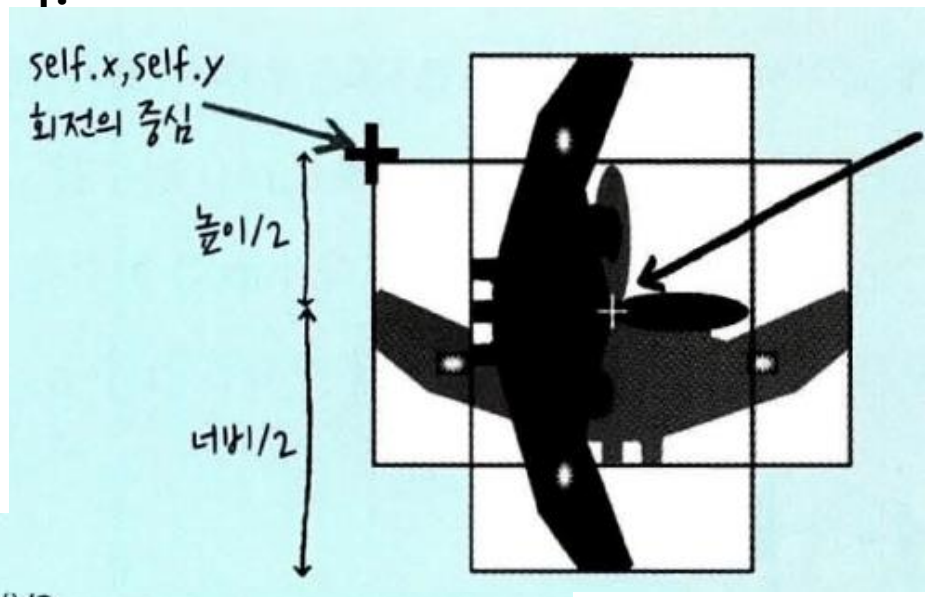
- 이제 이미지 회전 방법을 알겠지요?
- 이렇게 고친 회전 파이터를 앞에서 만든 우주 침략자 게임에 넣어서 업그레이드 해 봅시다.
- 우주 침략자 게임 파일을 열어서 파이터 클래스에 turn() 함수를 추가하고, draw() 함수를 업데이트하고, self.dir을 __init__() 함수에 넣으면 됩니다.
- turn() 함수를 불러오는 것도 잊지 마세요.
- fighter.turn()을 게임 루프 안, fighter.draw() 위에 추가하면 됩니다.

4. 우주 침략자 게임으로 돌아가자

- 현재 파이터의 y좌표는 591입니다.
- draw() 함수에 직접 591이라고 썼었지요.
- 이제 회전하게 됐으니 y좌표를 591로 정하면 너무 낮습니다.
- 파이터를 회전시키면 날개 끝이 스크린의 바닥에 닿게 됩니다.
- 파이터가 아래쪽에 있으면서도 이미지가 잘린 것처럼 보이지 않게 해야 합니다.
- 원래 파이터에는 self.y라는 변수는 없었습니다.
- 앞에서 말했듯이 그냥 숫자를 draw() 함수 안에 적었으니까요.
- 미사일을 쉽게 업데이트하려면, 미사일의 __init__() 함수 안에 self.y 변수를 만들고 미사일의 draw() 함수 안의 591을 self.y로 바꿉니다.
- self.y는 파이터 이미지가 스크린 바닥에 수평으로 위치할 때 너비의 반에 높이의 반을 더한 값이어야 합니다.
- 너비의 반과 높이의 반의 합은 50+30= 80입니다.
- 스크린의 높이에서 80을 빼면 570이 됩니다.

4. 우주 침략자 게임으로 돌아가자

- 아니면 그냥 아래처럼 __init__() 함수에서 다 정해도 됩니다.
- 이런 식으로 이미지나 스크린의 크기를 바꿀 수 있습니다.
- 값을 리셋할 필요가 없지요.
- 자동으로 파이터의 X좌표를 한가운데, y좌표는 스크린 가장 밑에서부터 80픽셀 만큼 위로 올라온 곳으로 정할 수 있습니다.




```
def __init__(self):
    self.x =screen.get_width()/2-fighter_image.get_width()/2
    self.y =screen.get_height()-(fighter_image.get_width()/2 +fighter_image.get_height()/2)
    self.dir = 0
```

4. 우주 침략자 게임으로 돌아가자

- 파이터는 위아래로 움직이지 않습니다.
- 그러니 회전각에 제한 조건을 추가해 봅시다.
- 'z'를 눌렀을 때, 그리고 self.dir이 -90보다 큰 경우에만 self.dir이 바뀝니다.
- 45, 0, -60 모두 -90보다 큽니다. -120은 90보다 작지요.
- -90도일 때 파이터는 오른쪽을 바라봅니다.
- 'a'를 눌렀을 때도 마찬가지입니다.

```
def turn(self):
    if pressed_keys[K_a] and self.dir < 90:
        self.dir += 1
    if pressed_keys[K_z] and self.dir > -90:
        self.dir -= 1
```

5. 임의의 방향으로 미사일 발사

- 회전하는 파이터는 멋지지만, 중요한 문제가 있습니다.
- 미사일들이 수직으로만 발사된다는 것입니다.
- 이 문제를 고치려면 미사일의 __init__() 함수에 인자를 2개 추가해야 합니다.

```
class Missile:
    def __init__(self,x,y,dir):
        self.x = x
        self.y = 591 y
        self.dir = dir
```

- x는 앞에서 만들어 봤지요.
- x는 미사일이 만들어질 때 파이터의 중심의 x좌표입니다.
- 이제 파이터의 y좌표와 미사일이 발사될 때 파이터가 가리키는 방향을 인자로 줍니다.
- 방향 인자는 dir이라고 했습니다.

5. 임의의 방향으로 미사일 발사

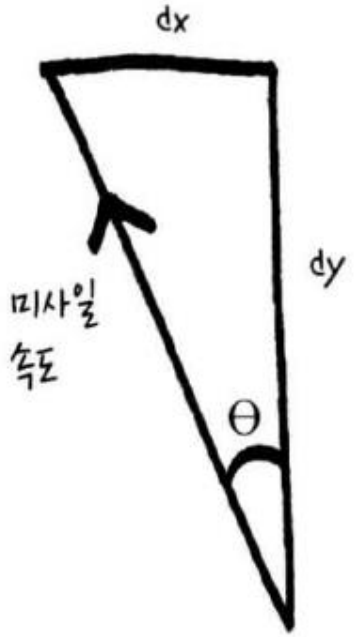
- 미사일은 파이터 클래스의 fire() 함수에서 만들어집니다.

```
class Fighter:
(중략)
    def fire(self):
        global shots
        shots += 1
        missiles.append(Missile(self.x+50 fighter_image.get_width()/2,self.y,self.dir))
```

- 미사일의 x좌표는 파이터의 x좌표에 파이터 이미지의 가로 길이의 반을 더한 값입니다.
- 원래는 그냥 50이라고 썼지요.
- 미사일의 y좌표는 파이터의 y좌표(앞에서 만든 self.y) 입니다.
- 미사일의 dir은 당연히 파이터의 dir와 같습니다.

5. 임의의 방향으로 미사일 발사

- 이제 dir도 있으니 삼각비를 사용해 미사일을 다양한 방향으로 발사시켜 봅시다.
- 삼각비에 의하면 dx = 속도 x sinθ 입니다. 그림에서 θ는 양수입니다.
- 위쪽 방향을 0으로 정했기 때문에, 시계 반대 방향으로 돈 각이 양수입니다.
- 그러나 dx는 왼쪽으로 가므로 음수여야 합니다.
- 따라서 dx = 속도 x sinθ가 됩니다.
- 그림 상으로는 dy는 양수지만 미사일은 스크린 위쪽으로 향하기 때문에, 음수여야 합니다.
- 따라서 dy =- 속도 x cosθ가 됩니다.



5. 임의의 방향으로 미사일 발사


```
class Missile:
    def __init__(self,x,y,dir):
        self.x = x
        self.y = y
        self.dir = dir
        self.dx = -math.sin(math.radians(dir))*5
        self.dy = -math.cos(math.radians(dir))*5
```

- 이제 __init__() 함수는 위와 같은 형태가 됩니다.
- 속도는 5로 고정합니다.
- self.dir은 필요 없습니다.
- dir은 __init__() 함수 밖에서 사용되지 않기 때문입니다.

5. 임의의 방향으로 미사일 발사

- dir을 쓸 때는 60분법을 사용했습니다.
- 앞에서도 말했듯이 파이게임이 60분법을 사용하니까요.
- 하지만 계산은 파이썬이 하고 파이썬 math 모듈에서 불려오는 수학 관련 함수에서는 각을 라디안으로 써야 합니다.
- 따라서 radians() 함수로 도(dir의 값)를 라디안으로 변환해야 합니다.
- 좀 돌아가는 것 같지만 이렇게 하는 것이 좋습니다.
- 프로그래밍을 하다 보면 익숙해질 거예요.
- 아참, 첫 줄에 math 모듈을 가져오는 것도 잊지 마세요.

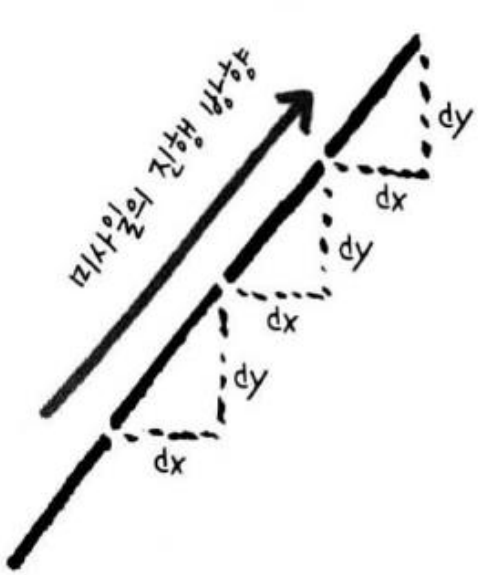
```
import pygame, sys, random, time, math
```

5. 임의의 방향으로 미사일 발사

- 이제 미사일 클래스 안에 있는 move() 함수를 업데이트해야 합니다.
- 전에는 self.y를 -5씩 바꿨었지요.
- 이제 다양한 방향으로 움직이므로, self.x와 self.y 모두 업데이트해야 합니다.

```
class Missile:
(중략)
    def move(self):
        self.y -= 5
        self.x += self.dx
        self.y += self.dy
```

6. 버그 수정 1: 회전되어 나가는 미사일



- 코드를 다 쓰고, 미사일을 발사시켜 보세요.
- 두 가지 문제가 보일 것입니다.
  - 첫째, 다양한 방향으로 미사일이 발사되지만 여전히 이미지는 수직입니다.
  - 둘째, 정확히 파이터의 코 부분에서 발사되지 않습니다.
- 파이터를 회전시키면 더 뚜렷이 확인할 수 있습니다.
- 미사일의 길이를 미사일이 움직이는 거리와 같도록 바꾸면, 그림에서 본 것처럼 움직이게 됩니다.
- 미사일의 가장 앞쪽 끝의 좌표는 항상 (x, y)입니다.
- 매 게임 루프마다 미사일의 가장 앞쪽 끝은 x방향으로는 dx만큼, y방향으로는 dy만큼 움직입니다.
- 따라서 뒤쪽 끝의 좌표는 (x-dx, y-dy)가 됩니다.
- 이 좌표들은 미사일을 그릴 때 사용한 line() 함수의 4번째 인자입니다.

6. 버그 수정 1: 회전되어 나가는 미사일

- 따라서, 미사일의 draw() 함수를 아래와 같이 바꿉니다.
  - 미사일 이미지를 사용했지만 다시 직선으로 그린 후 고쳐봅시다.

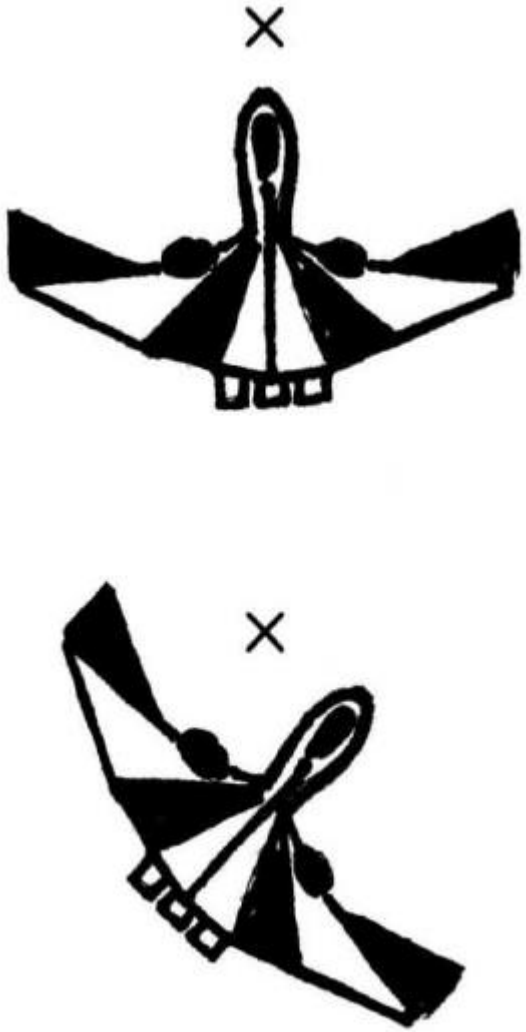
```
class Missile:
(중략)
    def draw(self):
        screen.blit(fighter_image,(self.x,591))
        pygame.draw.line(screen,(255,0,0),(self.x,self.y),(self.x-self.dx,self.y-self
.dy),1)
```

실행

- 만약 미사일의 길이를 길게 하거나 짧게 줄이고 싶으면 두 번째 좌표 인자에 있는 dx와 dy에 숫자를 곱하세요.
- 예를 들면 (self.x - 2*self.dx, self.y - 2*self.dy) 이렇게요.

7. 버그 수정 2: 코에서 미사일 발사

- 이제 파이터가 회전할 때 미사일이 파이터의 코에서 나오지 않는 문제를 해결합시다.
- 회전 전에는 파이터의 코에서 미사일이 나왔습니다.
- 파이터가 회전하면 파이터의 코도 움직이지만, 미사일은 여전히 예전 위치에서 발사됩니다.
- 파이터의 코의 새로운 좌표 계산을 위해 다시 삼각비를 사용하는 것도 가능하지만, 그보다 파이터의 회전 중심에서 미사일이 발사되도록 하는 것이 훨씬 쉽습니다.
- 파이터의 회전 중심의 x좌표는 파이터의 x좌표에 파이터의 너비의 반을 더한 값입니다.
- 이 값은 이미 미사일에게 알려 주었지요. __init__() 함수 안의 x입니다.



7. 버그 수정 2: 코에서 미사일 발사

- 회전 중심의 y좌표는 파이터의 y좌표보다 파이터의 높이의 반절만큼 아래인 위치입니다.
- 전에는 미사일에게 파이터의 y좌표를 알려 줬습니다.
- 이제는 미사일에게 파이터의 y좌표에 파이터의 높이의 반을 더한 값을 알려 줘야 합니다.
- x좌표를 계산하는 것과 비슷하지요.

```
class Fighter:
(중략)
    def fire(self):
        global shots
        shots += 1
        missiles.append(Missile(self.x+fighter_image.get_width()/2,self.y+fighter_image.
    get_height()/2,self.dir))
```

실행

8. 버그 수정 3: 미사일은 발사될 때 그려

- 회전하지 않은 파이터가 미사일을 발사할 때는 제대로 보이지만, 회전할 때는 파이터의 코 쪽에서 비뚤어진 미사일이 나옵니다.
- 회전 중심에서 발사되는 미사일이 코 쪽으로 올 때 쯤이면 코는 왼쪽이나 오른쪽으로 회전해버렸기 때문입니다.
- 이런 일을 막으려면 미사일을 플레이어의 코앞으로 점프시키고 그려야 합니다.
- 발사 위치는 여전히 중심이지만, 그려지기 시작하는 것은 코를 지난 다음일 테니까요.
- 오른쪽 그림 에서의 점선은 앞에서 설명했던 dx와 dy의 경로의 확대 버전입니다

8. 버그 수정 3: 미사일은 발사될 때 그려

- 미사일의 시작점을 코앞으로 정하기 위해서는 우리는 그냥 미사일의 시작점에 dx와 dy의 확대 버전을 더하기만 하면 됩니다.
  - 음수지만 더한다고 표현합니다.

```
class Missile:
    def __init__(self,x,y,dir):
        self.x = x - math.sin(math.radians(dir))*60
        self.y = y - math.cos(math.radians(dir))*60
        self.dx = -math.sin(math.radians(dir))*5
        self.dy = -math.cos(math.radians(dir))*5
```

- 회전 중심을 찾고 코에 도착하기 위해 x와 y의 값을 얼마나 바꿔야 하는지 계산하기 위해 삼각비를 사용했습니다.
- 코의 위치를 정하기 위해 삼각비를 쓰지 않겠다고 했지만 쓴 거나 다름없네요.

8. 버그 수정 3: 미사일은 발사될 때 그려

- 위 코드의 순서를 바꿔서 만들지도 않은 변수를 참조하지 않게 할 수도 있습니다

```
class Missile:
(중략)
    def draw(self):
        pygame.draw.line(screen,(255,0,0),(self.x,self.y),(self.x-self.dx,self.y-self.
dy),1)
        rotated = pygame.transform.rotate(missile_image,self.dir)
        screen.blit(rotated,(self.x-4,self.y))
```

- 쉽지요? 파이터만큼 미사일을 회전시키기 위해 rotate() 함수를 사용했습니다.

8. 버그 수정 3: 미사일은 발사될 때 그려

- dir을 __init__() 함수의 밖에서 사용하기 때문에 self.dir = dir 코드를 __init__() 함수에 추가해야 합니다.

```
class Missile:
    def __init__(self,x,y, dir):
        self.x = x-math.sin(math.radians(dir))*60
        self.y = y-math.cos(math.radians(dir))*60
        self.dx = -math.sin(math.radians(dir))*5
        self.dy = -math.cos(math.radians(dir))*5
        self.dir = dir
```

- 미사일 이미지를 쓰려면 다운받고, 파이썬으로 가져오고, 배경도 투명하게 해야 겠지요.
- 셋업에 있는 아래 코드가 하는 일이 그거랍니다.

```
missile_image = pygame.image.load("images/missile.png").convert()
missile_image.set_colorkey((255,255,255))
```

Thank You