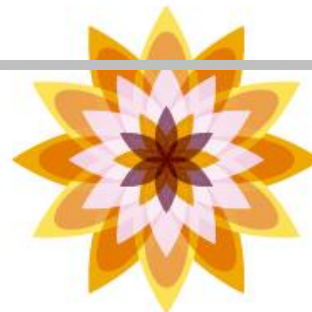
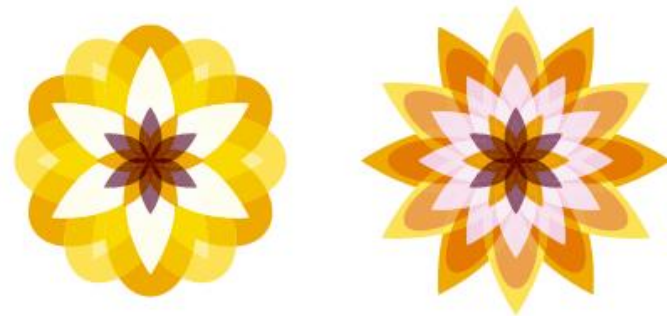


Chapter 12
게임 오버



1. 파이터가 악당에 닿는 세 가지 조건

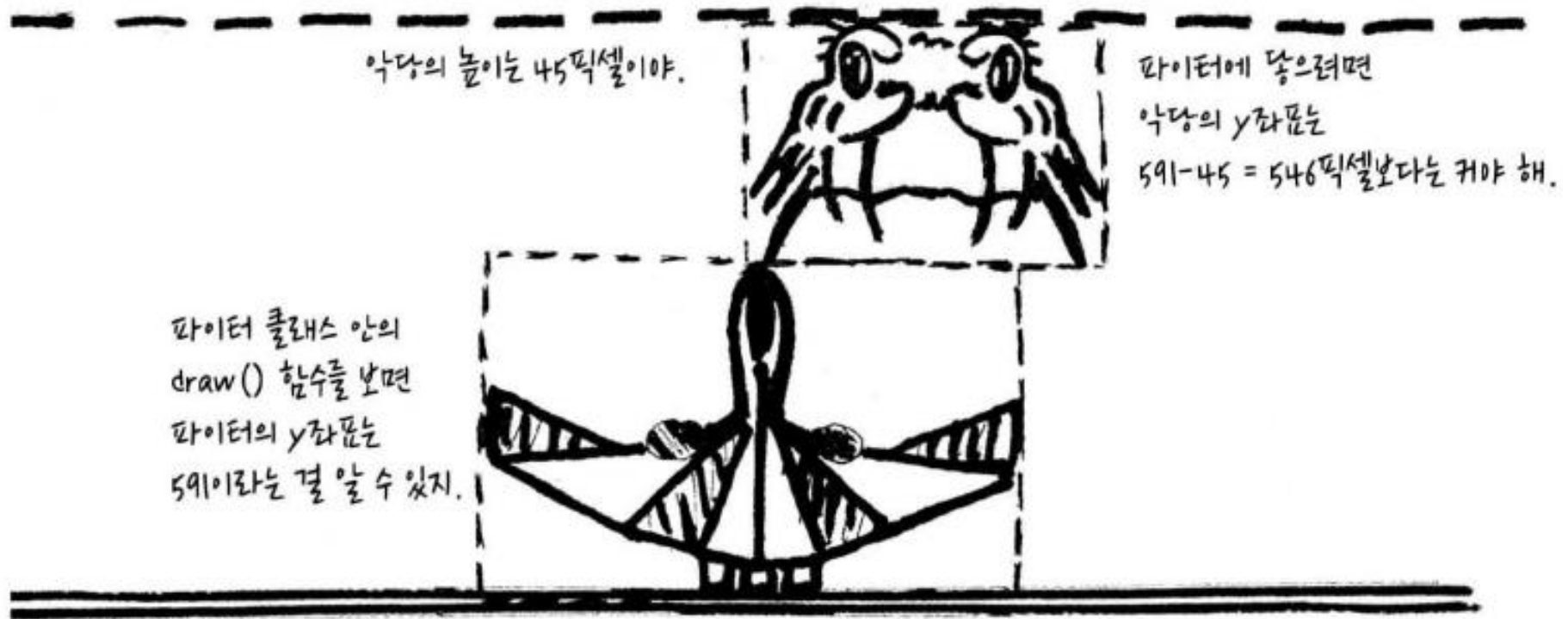
- 아직 게임이 완성되지 않았습니다.
- 악당, 미사일 점수, 파이터까지 만들었지만 게임 오버 상황은 아직 만들지 않았으니까요.
- 악당이 파이터에 닿으면 게임이 끝나도록 만듭시다.
- 지금부터 악당과 파이터가 닿았는지를 감지하는 함수를 만들 것입니다.
- 이 함수는 파이터 클래스 안에 씁니다.

```
class Fighter:
    (중략)
    def hit_by(self, badguy):
        return (
            badguy.y > 546 and
            badguy.x > self.x - 70 and
            badguy.x < self.x + 100
        )
```

1. 파이터가 악당에 닿는 세 가지 조건

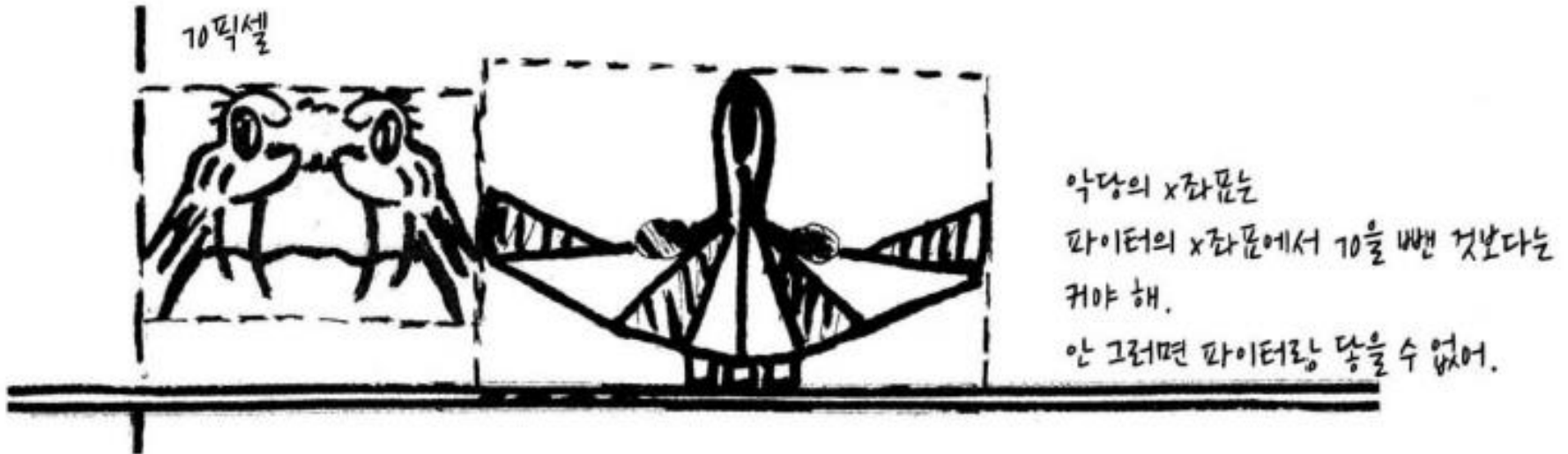
- 위 코드는 딱 두 줄이에요.
- 1번째 줄은 def로 시작하고, 2번째 줄은 return으로 시작합니다.
- 2번째 줄에는 3개의 조건이 들어 있습니다.
- 세 조건이 모두 참일 경우에만 참값을 돌려주라는 뜻입니다.
- 첫 번째 조건은 `badguy.y > 546`입니다.
- 이 말은 악당의 y좌표가 546보다 커야 한다는 말입니다.
- 왜 546일까요?

1. 파이터가 악당에 닿는 세 가지 조건



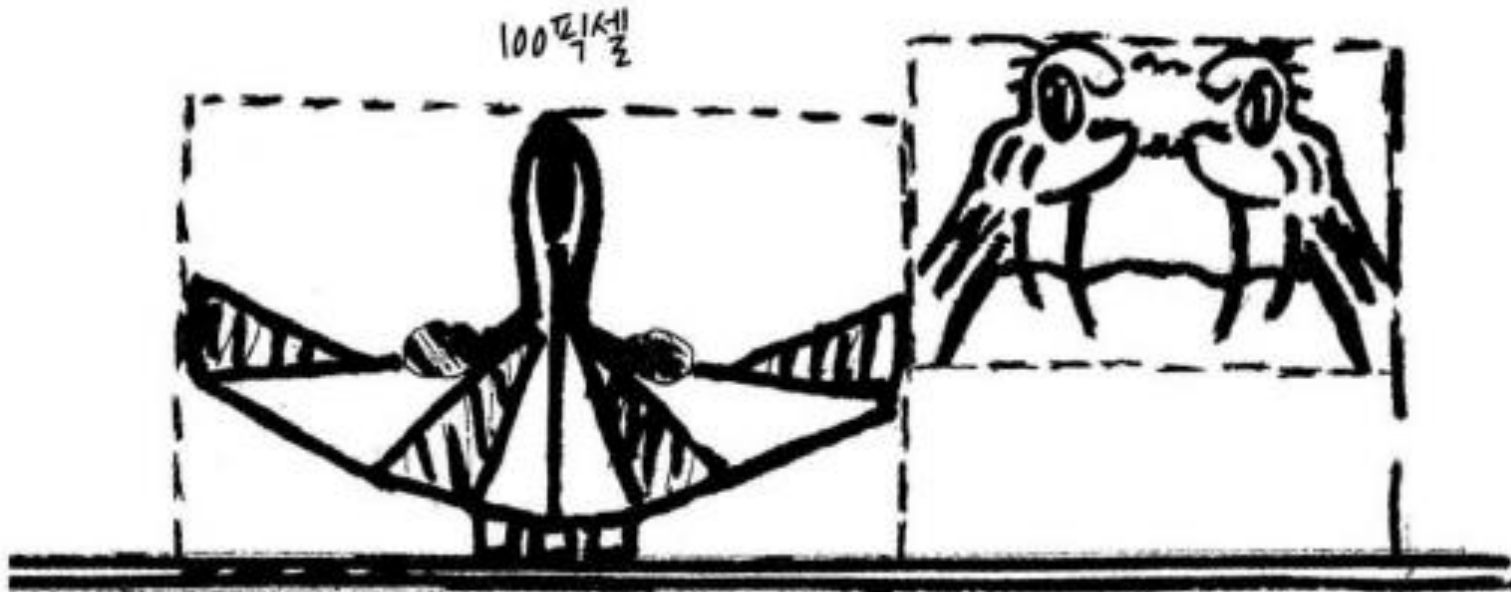
1. 파이터가 악당에 닿는 세 가지 조건

- 두 번째 조건은 $\text{badguy.x} > \text{selfx} - 70$ 입니다.



1. 파이터가 악당에 닿는 세 가지 조건

- 마지막 조건은 $\text{badguy.x} < \text{selfx} + 100$ 입니다.



여기서 악당의 x좌표는
파이터의 x좌표에 100을 더한 것보다 작아야 해.
안 그러면 파이터랑 닿을 수 없어.

1. 파이터가 악당에 닿는 세 가지 조건


- 세 조건이 모두 참이면, 악당은 파이터에 닿습니다.
- 이때 `hit_by()` 함수는 참값을 돌려줍니다.
- 세 조건이 모두 참이지만 악당이 파이터 아래에 있는 경우도 있습니다.
- 악당이 스크린 바닥으로 떨어져 `off_screen()` 함수에 의해 삭제되는 경우 말입니다.
- 만약 `off_screen()` 함수가 없다면, 파이터의 아래쪽과 악당의 위쪽이 겹치는지 확인하는 4번째 조건을 추가해야 했을 것입니다.

1-1. 줄 나누기

- 화면 밖으로 코드가 빠져나가지 않게 길이를 줄이려면, 괄호 안에서 나눠 써야 합니다.
- 파이썬이 괄호 안의 모든 것이 한 줄의 코드라는 것을 알기 때문입니다.
- 예를 들어 봅시다

```
def draw(self):  
    pygame.draw.line(screen, (255,0,0), (self.x, self.y), (self.x, self.y+8), 1)
```

- 코드가 길어져서 줄이 바뀌는 경우 파이썬은 들여쓰기를 무시하므로 다음과 같이 써도 됩니다.

```
def draw(self):  
    pygame.draw.line(screen, (255,0,0), (self.x, self.y),  
 (self.x, self.y+8), 1)
```


1-1. 줄나누기

- 다른 방법으로는, 연산자operator 앞이나 뒤에 역슬래시 back slash를 쓰기도 합니다.
- 역슬래시 뒤는 띄어쓰기 하지 마세요.

```
y = 3+4+\n    5+6
```

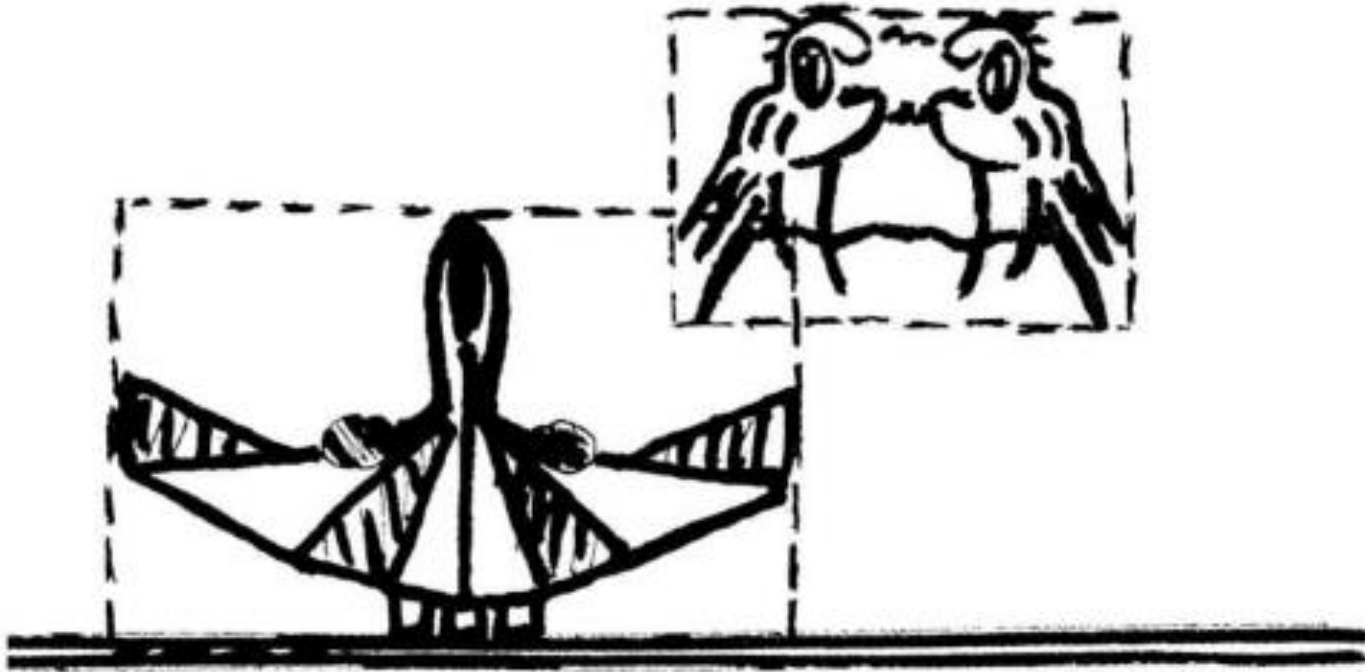
- 파이썬 개발자들은 보통 괄호를 추가하지요.

```
y = (3+4+\n    5+6)
```

- 깔끔해 보이라고 이렇게 쓰기도 합니다
- 이 코드들은 여러분의 텍스트 편집기에서는 한 줄로 보일 수도 있습니다
- 하지만 실제로 코드를 이렇게 쓰기도 합니다. 읽기 쉬우라고요

2. 세 조건이 모두 참이지만, 예외?

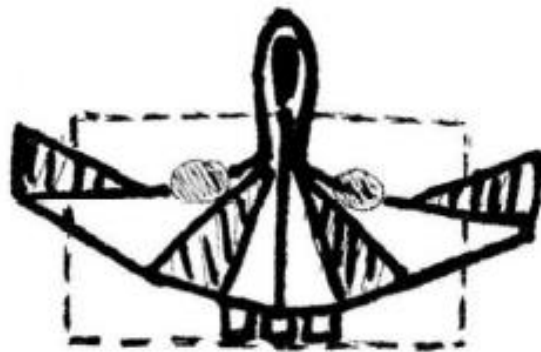
- 세 조건이 모두 참이지만 악당이 파이터와 닿지 않은 경우도 있습니다.



- 위 그림같이 이미지를 둘러싼 직사각형은 서로 겹치지만 실제 이미지는 겹치지 않습니다.
- 파이썬은 이것을 닿았다고 보겠지만 게임 플레이어는 동의하지 않을 거예요.

2. 세 조건이 모두 참이지만, 예외?

- 우리는 간단히 만들기 위해 파이게임의 직사각형 시스템을 사용했습니다.
- 이미지가 실제로 겹치는지 알아내는 것은 이 강의의 범위를 벗어나거든요.
- 간단히 고치려면 파이터의 직사각형을 오른쪽 그림과 같이 줄이면 됩니다.
- 그러면 함수에 쓴 숫자들도 조금 바꾸어야 합니다.



```
class Fighter:
```

```
(중략)
```

```
    def hit_by(self,badguy):
```

```
        return badguy.y > 546 585 and badguy.x > self.x - 70 55 and badguy.x < self.x + 100 85
```

2. 세 조건이 모두 참이지만, 예외?

- 이렇게 고치면 오른쪽 그림 같은 상황도 안 닿은 것으로 간주됩니다.
- 이 정도는 괜찮습니다.
- 어떤 게 좋을지 스스로 결정해 보세요 .
- 또 다른 해결법은 파이터나 악당을 직사각형에 가깝게 그리는 것입니다.



3. 죽음

- 악당이 파이터와 닿았는지 알려 주는 함수를 만들었으니 사용해 봅시다.

```
screen.blit(font.render("Score: "+str(score),True,(255,255,255)),(5,5))
```

```
for badguy in badguys:
    if fighter.hit_by(badguy):
        while 1:
            for event in pygame.event.get():
                if event.type == QUIT:
                    sys.exit()
            pygame.display.update()
```

```
pygame.display.update()
```

3. 죽음

- 음영 표시된 부분의 1번째, 2번째 줄은 악당들 리스트에서 악당이 있는지 찾아서 , 우리가 방금 만든 `fighter.hit_by()` 함수를 불러옵니다.
- 만약 `hit_by()` 함수가 거짓값을 돌려주면, 즉 악당이 파이터에 닿지 않았다면, 프로그램은 리스트에서 다음 악당을 가져와 다시 확인합니다.
- 모든 악당에 대해 거짓이면, 게임 루프는 이 코드를 무시하고 게임을 진전시킵니다.
- 하지만 만약 `hit_by()` 함수가 참값을 돌려주면 `while` 루프 안으로 들어가게 됩니다.
- `while 1:`로 무한 루프를 시작하지요.
- `while` 루프는 게임 루프 안에서 무한 반복됩니다.
- 아무것도 하지 않는 작은 원을 계속 돌지요.
- 그러면 스크린 위에서는 게임이 멈춘 것처럼 보입니다.

3. 죽음

- 악당에 닿으면 게임이 그냥 멈춰 버리는 거예요.
- 이 while 루프에서 벗어나려면 끝내기 버튼을 누르는 수밖에 없습니다.
- while 루프 안의 코드들이 낮익지 않나요?
- 화면을 새로 고침하고 while 루프를 끝내도록 합니다.
- 뭔가 이상하다고요? 너무 간단한 거 아니냐고요?
- 만약 여러분이 닌텐도에서 일한다면 이렇게 하면 안 되겠지요.
- 하지만 지금 우리는 쓰기 편하고 잘 돌아가기만 하면 됩니다.
- 게임이 너무 쉽다 싶으면, 악당의 속도를 올리거나 악당이 만들어지는 간격을 줄여서 수를 늘려 보세요.
- 이제 게임 오버 스크린과 최종 점수를 추가해 봅시다.

4. 게임 오버

- 게임 오버 이미지 파일은 웹사이트에서 다운받을 수 있습니다.
- 너비는 300픽셀이고 높이는 244픽셀입니다. 직접 만들어도 됩니다.



4. 게임 오버

- 자 그럼, 일단 이미지 를 저장할 변수를 만들어 봅시다.
- 아래 코드를 다른 이미지 변수들과 같이 프로그램 셋업에 씁니다.

```
GAME_OVER = pygame.image.load("images/gameover.png").convert()
```

- 게임 오버 이미지 표시 코드는 파이터와 악당이 스칠 때 실행돼야 합니다.
- 하지만 우리가 방금 만든 while 루프에 빠져 버리기 전이어야겠지요.

4. 게임 오버

```
for badguy in badguys:
    if fighter.hit_by(badguy):
        screen.blit(GAME_OVER, (170, 200))
        while 1:
            for event in pygame.event.get():
                if event.type == QUIT:
                    sys.exit()
            pygame.display.update()
```

- GAME OVER 변수에 저장한 이미지를 (170 , 200)에 표시합니다.
- 그런데 게임 오버 이미지 위에 숫자도 표시해야 합니다.
- 전체 발사 횟수, 맞힌 수, 못 맞힌 수, 정확도, 점수 등을 표시해야 하지요.

4. 게임 오버

```
for badguy in badguys:
    if fighter.hit_by(badguy):
        screen.blit(GAME_OVER, (170, 200))
        while 1:
            for event in pygame.event.get():
                if event.type == QUIT:
                    sys.exit()
            pygame.display.update()
```

- GAME OVER 변수에 저장한 이미지를 (170 , 200)에 표시합니다.
- 그런데 게임 오버 이미지 위에 숫자도 표시해야 합니다.
- 전체 발사 횟수, 맞힌 수, 못 맞힌 수, 정확도, 점수 등을 표시해야 하지요.

5. 발사 횟수

- 이미 점수score 변수는 있으니 발사 횟수total shots, 맞힌 수hits 못 맞힌 수mlsses 등을 추가합시다.
- 정확도accuracy는 다른 변수를 계산해 만들 수 있기 때문에 신경 쓰지 않아도 됩니다.
- 프로그램의 셋업, 점수 변수를 표시한 부분 (score = 이 아래 다음 코드를 추가하세요.

```
shots = 0  
hits = 0  
misses = 0
```

5. 발사 횟수

- 이제 프로그램을 살펴보면서 변수들 업데이트 코드를 추가합시다.
-]점수 변수에 대해서는 이미 만들었지요.
- 먼저 발사 횟수를 봅시다 이 변수는 미사일이 발사될 때마다 1씩 증가해야 합니다.
- 따라서 파이터 클래스에 있는 fire() 함수에 코드를 한 줄 추가해야 합니다.
- global 기억나나요?
- fire() 함수한테 발사횟수shots는 global전역 변수라는 것을 알려 줘야 합니다.

```
class Fighter:
    (중략)
    def fire(self):
        global shots
        shots += 1
        missiles.append(Missile(self.x+50))
```

6. 맞힌 수

- 맞힌 수 hits는 미사일이 악당을 맞혔을 때 사용됩니다.
- 따라서 미사일과 악당 사이의 충돌을 감지하는 루프 안에 씁니다.

```
i = 0
while i < len(badguys):
    j = 0
    while j < len(missiles):
        if badguys[i].touching(missiles[j]):
            badguys[i].score()
            hits += 1
            del badguys[i]
            del missiles[j]
            i -= 1
            break
        j += 1
    i += 1
```

6. 맞힌 수

- `badguys [i].score()` 바로 밑에 `hits += 1` 이라고 썼습니다.
- 두 코드는 같은 일을 서로 다른 방식으로 합니다.
- 맞힌 수는 1씩 증가하지만, 점수는 100씩 증가합니다.
- 맞힌 수도 점수처럼 함수를 만들어도 괜찮습니다.
- 아니면 점수를 맞힌 수처럼 만들어도 됩니다.
- 마지막에 점수를 100으로 나누어 맞힌 수를 구할 수도 있습니다.
- 세 방법 모두 가능합니다.
- 앞에서도 말했지만 작동만 한다면 원하는 대로 해도 됩니다.

6. 맞힌 수

- 그런데 점수를 함수로 만든 데는 이유가 있습니다.
- 나중에 다른 종류의 악당을 추가할 수도 있기 때문입니다.
- 악당 타입 2, 타입 3처럼 말이죠.
- 악당 타입 2를 죽이면 점수가 다르게 올라가도록 할 수도 있습니다.
- 점수와 맞힌 수를 분리하면, 나중에 점수를 이리저리 바꿔 볼 자유가 생기는 셈입니다.
- 하나의 클래스로 비슷하지만 완전히 똑같지는 않은 객체를 만드는 방법은 나중에 다룰 것입니다.

7. 못 맞힌 수

- 이제 못 맞힌 수 `misses`를 봅시다.
- 미사일 삭제 코드 바로 아래 추가합니다.

```
while i < len(missiles):  
    missiles[i].move()  
    missiles[i].draw()  
    if missiles[i].off_screen():  
        del missiles[i]  
        misses += 1  
        i -= 1  
    i += 1
```

7. 못 맞힌 수

- 발사된 미사일이 악탕을 맞추지 못하고 스크린 위로 빠져 나갈 때마다, 못 맞힌 수에 1을 더합니다.
- 맞힌 수에서처럼 새로 함수를 만들 이유도 없고, 코드가 추가되는 함수도 없습니다.
- `off_screen()` 함수는 미사일이 발사될 때만 실행되는 것은 아닙니다.
- 모든 게임 루프에서 모든 미사일에 대해 1번씩 실행됩니다.
- 따라서 `off_screen()` 함수가 참값을 돌려줄 때만 못 맞힌 수에 1을 더하도록 했습니다.

8. 숫자 표시

- 이제 이 변수들에 저장된 숫자를 게임 오버 이미지 위에 표시합니다.
- 아래 코드를 `screen.blit(GAME_OVER)` 코드 밑, 게임을 열려 버리는 `while` 루프 위에 씁니다.
- 가장 아래 있는 `for` 루프안입니다.

```
screen.blit(GAME_OVER,(170,200))
```

```
screen.blit(font.render(str(shots),True,(255,255,255)),(266,320))
screen.blit(font.render(str(score),True,(255,255,255)),(266,348))
screen.blit(font.render(str(hits),True,(255,255,255)),(400,320))
screen.blit(font.render(str(misses),True,(255,255,255)),(400,337))
screen.blit(font.render(str(100*hits/shots)+"%",True,(255,255,255)),(400,357))
```

```
while 1:
```

8. 숫자 표시

- 위 코드로 각각의 숫자가 스크린 어디에 표시되는지 알 수 있습니다.
- 점수 표시할 때 본 적 있는 코드지요?
- 게임 오버 이미지의 단어들과 어울리도록 숫자들을 정확한 좌표에 놓아야 합니다.
- 네 줄은 각각 발사 횟수 , 점수 , 맞힌 수 , 못 맞힌 수를 표시합니다.
- 이 값들은 변수로부터 바로 가져옵니다.
- 마지막 줄은 정확도를 나타냅니다.
- 정확도 변수는 따로 만들지 않았지만,%로 나타낸 정확도는 맞힌 수를 발사횟수로 나눈 다음 100을 곱하여 구합니다
- 마지막 줄은 2개의 문자열을 합치는 코드입니다.
- 두 번째 문자열은 % 표시입니다.

8. 숫자 표시

- 앞쪽의 코드를 실행시키면 정확도가 16자리 소수점으로 표시되는 것을 볼 수 있습니다.
- 이건 너무 많지요. 그러니 소수 첫째자리까지만 표시되도록 고쳐봅시다.

```
screen.blit(font.render(str(100*hits/shots)+"%", True, (255,255,255)), (400,357))  
screen.blit(font.render("{:.1f}%".format(100*hits/shots), True, (255,255,255)), (400,357))
```

- 정확히 설명하기는 어렵지만 간단히 말하자면, % 스타일 포맷 코드를 { } 사이에 넣어서 format() 함수 안에 있는 정보를 어떻게 표시할지 알려 주는 거랍니다.
- 즉, format() 함수에게 주어진 인자를 {} 안에 주어진 지시대로 표시하는 거지요.

8. 숫자 표시

- 다른 기호도 추가할 수 있어요.
- 이 경우에는 % 표시를 추가했지요.
- {}에 %를 붙인 것은 문자열이 됩니다.
- 문자열이니까 11 " 사이에 넣었지요.{} 안에 있는 .1은 format() 함수한테 소수 첫째자리까지만 표시하라는 뜻이에요 .
- .2라고 쓰면 소수 둘째자리까지 표시하겠지요.
- f는 실수float라는 뜻입니다.
- 실수는 소수점이 있는 수이지요. (정확한 수학적 정의는 조금 더 복잡합니다.)
- f를 써서 파이썬한테 정수가 아니라 실수라고 알려 주는 거예요.
- 파이썬 3.x.x 버전에서는 항상 실수값으로 계산해 주니까 반드시 필요한 건 아니지만요.
- 그래도 나중에 중요해질 거니까 항상 쓰는 습관을 들이세요.

9. 한 발도 못 쏘고 죽었을 때

- 어떤 수를 0으로 나눌 수 없습니다.
- 따라서 만약 발사 횟수가 0일 때 악당과 닿으면 에러가 발생합니다.
- 파이썬은 비명을 지르며 무너질 것입니다.
- 이 문제를 해결하기 위해, 맨 끝에 있는, 정확도를 계산하는 `screen.blit()` 코드를 다음 코드로 바꿉니다.

```
screen.blit(font.render("{:.1f}%".format(100*hits/shots), True, (255, 255, 255)), (400, 357))  
if shots == 0:  
    screen.blit(font.render("--", True, (255, 255, 255)), (400, 357))  
else:  
    screen.blit(font.render("{:.1f}%".format(100*hits/shots), True, (255, 255, 255)), (400, 357))
```

실행

9. 한 발도 못 쏘고 죽었을 때

- 어떤 수를 0으로 나눌 수 없습니다.
- 따라서 만약 발사 횟수가 0일 때 악당과 닿으면 에러가 발생합니다.
- 파이썬은 비명을 지르며 무너질 것입니다.
- 이 문제를 해결하기 위해, 맨 끝에 있는, 정확도를 계산하는 `screen.blit()` 코드를 다음 코드로 바꿉니다.

```
screen.blit(font.render("{:.1f}%".format(100*hits/shots), True, (255, 255, 255)), (400, 357))  
if shots == 0:  
    screen.blit(font.render("--", True, (255, 255, 255)), (400, 357))  
else:  
    screen.blit(font.render("{:.1f}%".format(100*hits/shots), True, (255, 255, 255)), (400, 357))
```

실행

- 이제 발사 횟수가 0이 되더라도 그 성가신 식은 더 이상 나오지 않을 것입니다.
대신 파이썬은 "--"를 출력하겠지요.
- 이제 우주 침략자 게임 만들기가 끝났습니다. 게임을 플레이해 보세요.



Thank You
