Lecture #14. 충돌처리

2D 게임 프로그래밍

이대현 교수

한국공학대학교
TECH UNIVERSITY OF KOREA

학습 내용

- 충돌 검사와 충돌 처리의 개념
- 사각형(바운딩 박스)를 이용한 충돌 검사
- 바운딩 박스를 이용한, 픽셀 단위 정밀도를 가지는 충돌 검사
- 충돌 검사의 실제 적용 방법

충돌 검사(Collision Detection)

- **충돌 검사**
  - 게임 상의 오브젝트 간에 충돌이 발생했는지를 검사하는 것.
  - 모든 게임에서 가장 기본적인 물리 계산.
    - 슈팅, 발차기, 펀치, 때리기, 자동차 충돌
    - 맵 상의 길 이동
  - 기본적으로 시간이 많이 소요되기 때문에, 게임의 오브젝트의 특성에 따라 각종 방법을 통해 최적화해주어야 함.
    - O(N2) 알고리즘
    - nC2 = n(n-1)/1*2=

충돌 처리(Collision Handling)

- 충돌이 확인 된 후, 이후 어떻게 할것인가?
  - 충돌 응답(Collision Response)

- 캐릭터와 아이템의 충돌에 대한 처리는??

- 바닥에 떨어지는 적군 NPC가 바닥과 충돌하면??

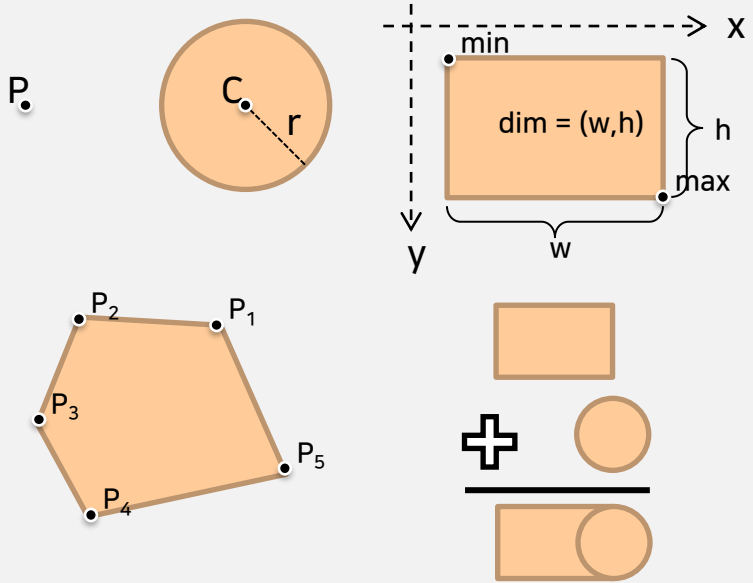
- 사선으로 움직이는 캐릭터가 맵의 벽과 충돌하면?

픽셀 단위의 충돌 검사

- 두 개의 오브젝트들의 모든 점들을 일일이 비교.
- 가장 정확함.
- 각 오브젝트들의 픽셀수를 곱한 만큼의 계산 시간이 소요됨.
  - 캐릭터 픽셀 수 x 공 픽셀 수

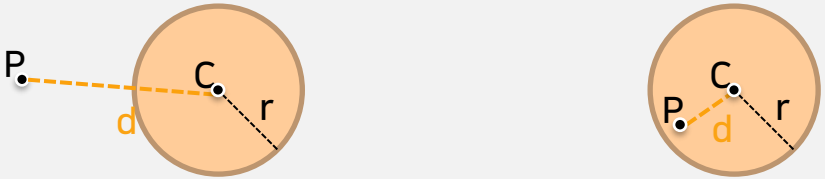
2D 관점에서 충돌 검사의 대상

- 점
- 원
- 사각형
- 볼록 다각형
- 복합 도형



점과 원

$$\|P - C\|^2 \leq r^2$$

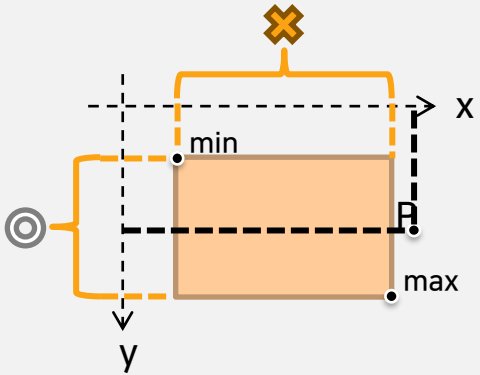


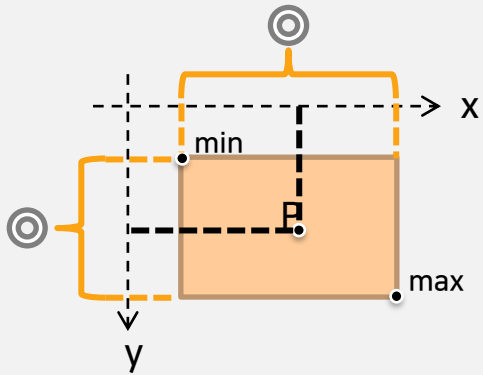
원과 원

$$\|C_2 - C_1\|^2 \leq (r_1 + r_2)^2$$

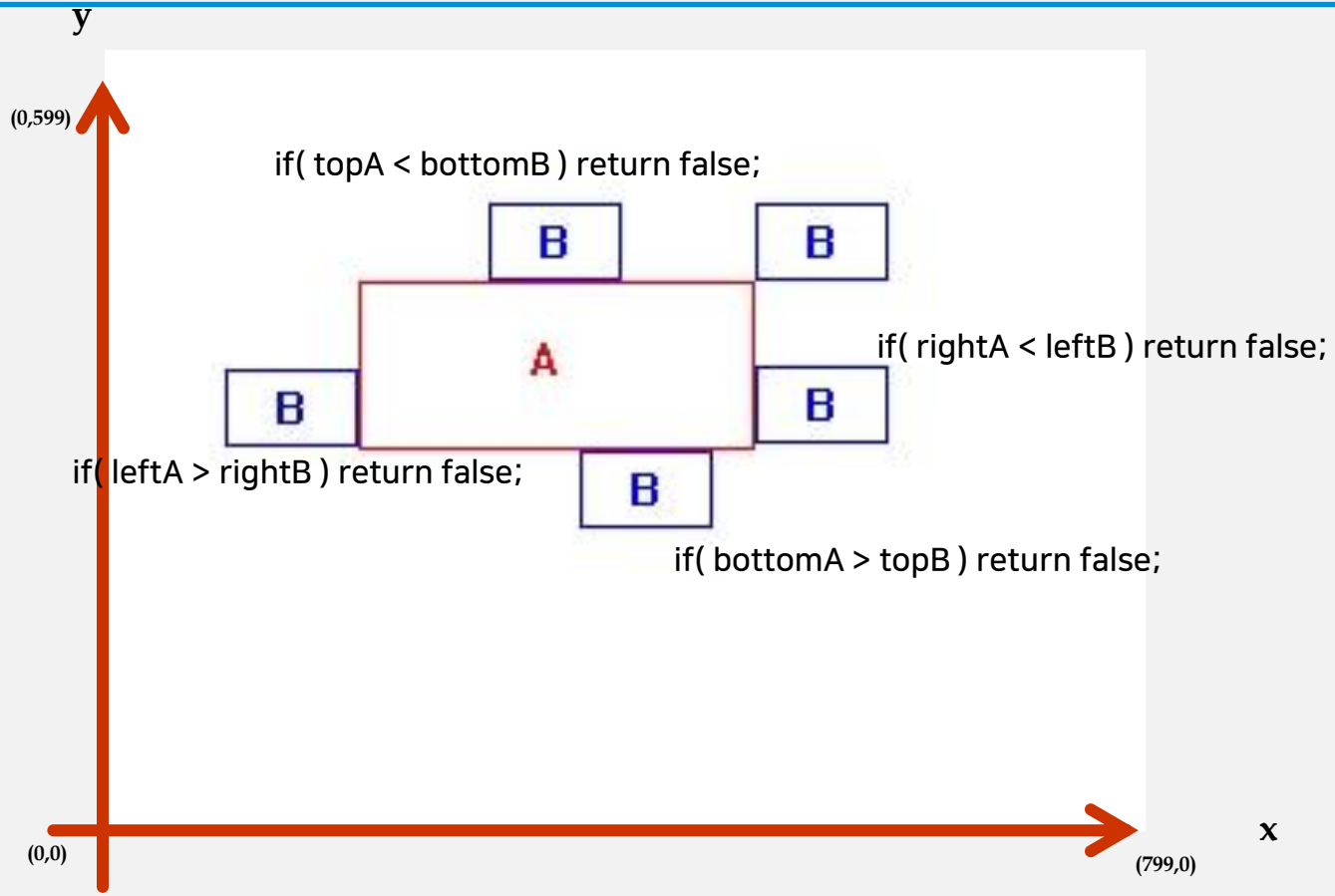
점과 사각형(AAB: Axis Aligned Box)

$$min_x \leq P_x \leq max_x \text{ AND } min_y \leq P_y \leq max_y$$





사각형과 사각형



바운딩 박스(Bounding Box)를 이용한 충돌 검사

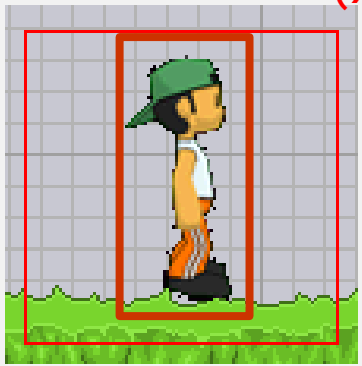
- 오브젝트를 감싸는 사각형(바운딩 박스)의 충돌을 비교.
- 사각형의 두개의 교차 여부만 결정하면 되므로 매우 빠름.
- 오브젝트의 형태가 복잡하면, 충돌 검사 결과가 매우 부정확해짐.

충돌 검사

boy.py


```
class Boy:


    def get_bb(self):
        return self.x - 50, self.y - 50, self.x + 50, self.y + 50
```

(x+50, y+50)



(x-50, y-50)

ball.py


```
class Ball:
    image = None

    def get_bb(self):
        return self.x - 10, self.y - 10, self.x + 10, self.y + 10
```

grass.py

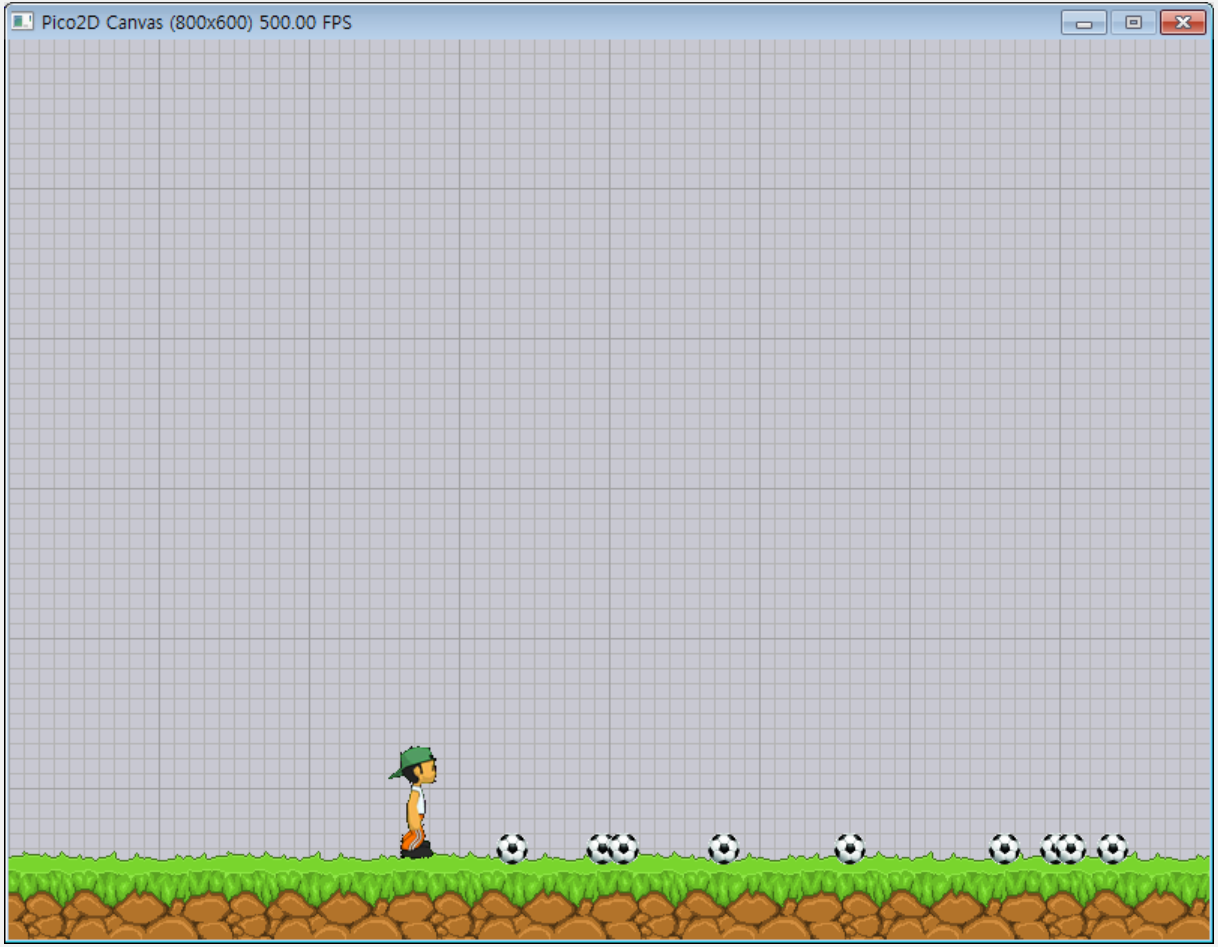
```
class Grass:
    def __init__(self):
        self.image = load_image('grass.png')

    def update(self):
        pass

    def draw(self):
        self.image.draw(400, 30)
        self.image.draw(1200, 30)

    # fill here
    def get_bb(self):
        return 0, 0, 1600-1, 50
```

play_state.py (1)

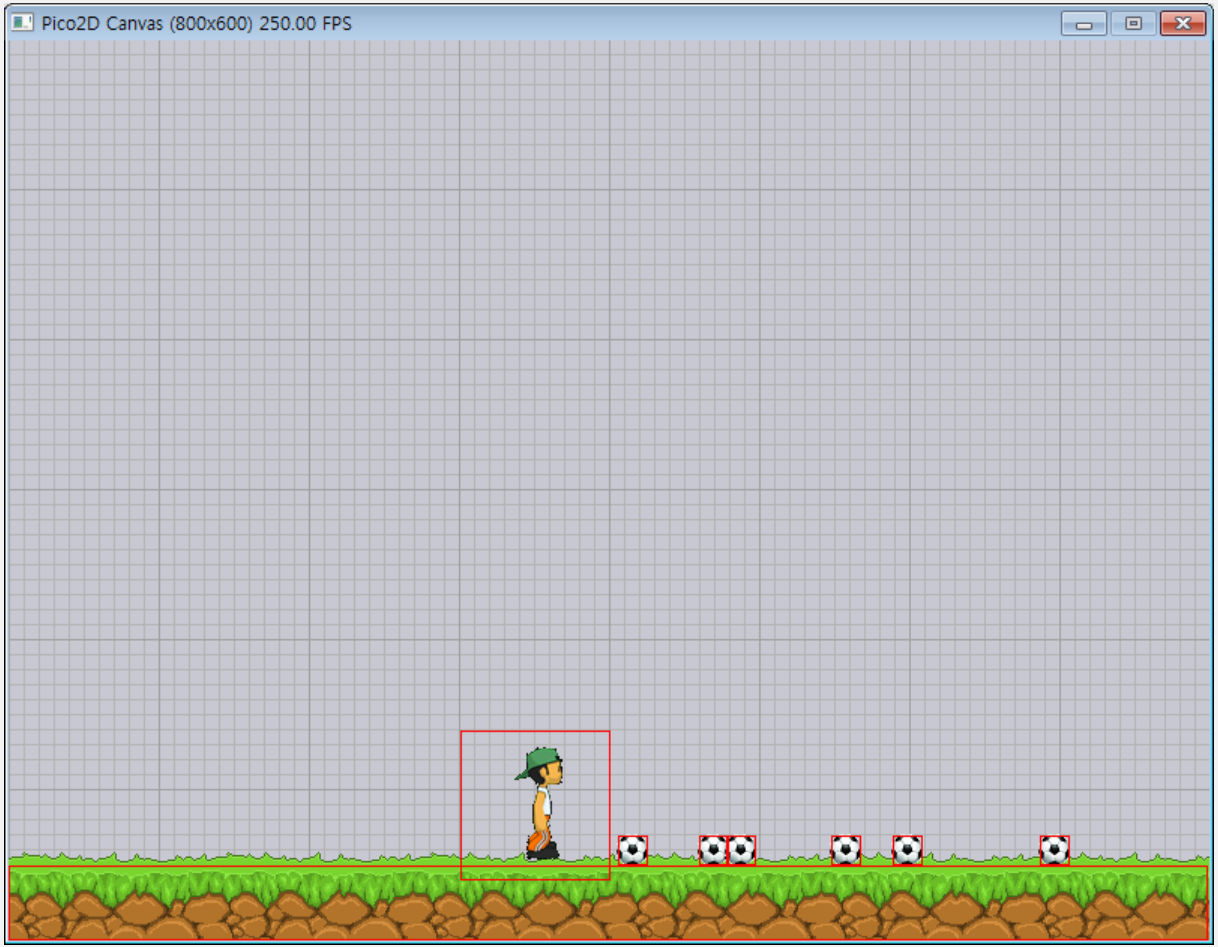

```
def collide(a, b):
    left_a, bottom_a, right_a, top_a = a.get_bb()
    left_b, bottom_b, right_b, top_b = b.get_bb()

    if left_a > right_b: return False
    if right_a < left_b: return False
    if top_a < bottom_b: return False
    if bottom_a > top_b: return False

    return True
```

play_state.py (2)


```
def update():
    for game_object in game_world.all_objects():
        game_object.update()
    for ball in balls:
        if collide(boy, ball):
            print("COLLISION boy:ball")
```



실습 디버그를 위한 충돌박스 그리기



boy.py

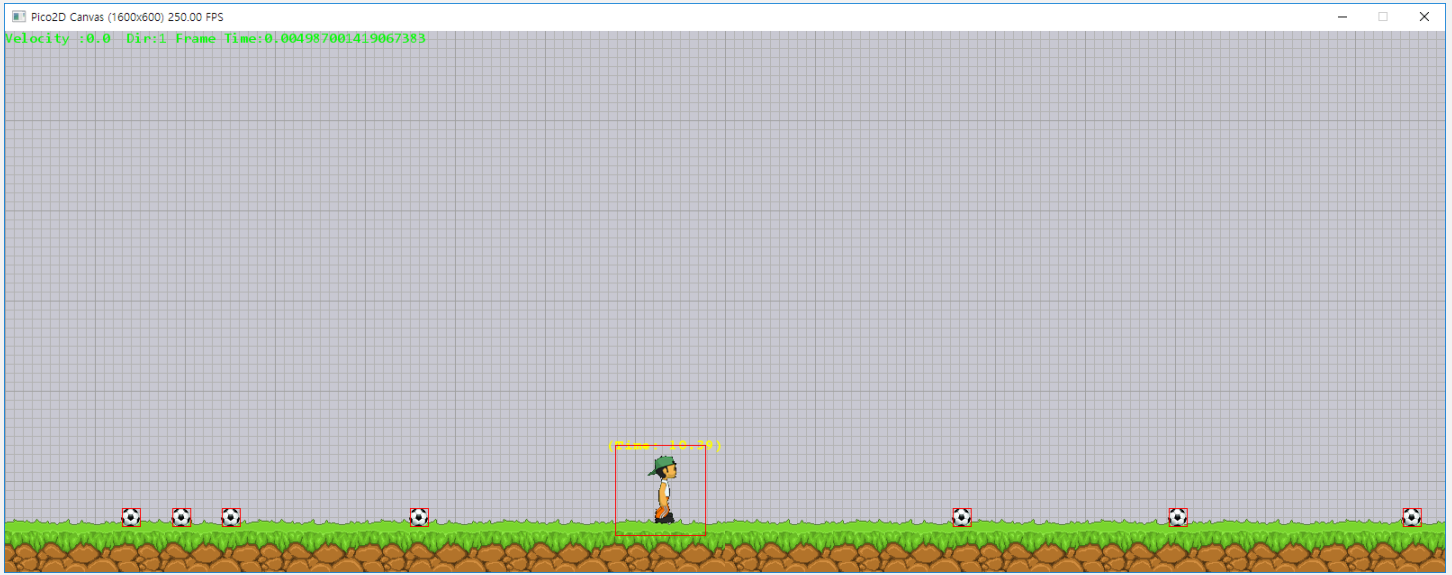
```
class Boy:
```


```
def draw(self):
    self.cur_state.draw(self)
    self.font.draw(self.x - 60, self.y + 50,
                   '(Time: %3.2f)' % get_time(), (255, 255, 0))
    draw_rectangle(*self.get_bb())
```

ball.py

```
class Ball:
```


```
def draw(self):
    self.image.draw(self.x, self.y)
    draw_rectangle(*self.get_bb())
```



충돌 처리

충돌 처리

- 충돌 이후에 어떻게 할 것인가?
- 미리 정책을 정해야 함.
- 캐릭터가 공을 만났다… 그래서? 그 다음은?
  - 공을 없앤다..

main_state.py


```
def update():
    for game_object in game_world.all_objects():
        game_object.update()
    for ball in balls.copy():
        if collide(boy, ball):
            balls.remove(ball)
            game_world.remove_object(ball)
```

실습 움직이는 물체와 충돌 처리

ball.py – 새로운 BigBall 클래스 추가


```
class BigBall(Ball):
    MIN_FALL_SPEED = 50  # 50 pps = 1.5 meter per sec
    MAX_FALL_SPEED = 200 # 200 pps = 6 meter per sec
    image = None

    def __init__(self):
        if BigBall.image == None:
            BigBall.image = load_image('ball41x41.png')
        self.x, self.y = random.randint(0, 1600-1), 500
        self.fall_speed = random.randint(BigBall.MIN_FALL_SPEED,
                                         BigBall.MAX_FALL_SPEED)

    def get_bb(self):
        return self.x - 20, self.y - 20, self.x + 20, self.y + 20
```

main_state.py


```
from boy import Boy
from grass import Grass
from ball import Ball, BigBall
```

main_state.py


```
def enter():
    global boy
    boy = Boy()
    game_world.add_object(boy, 1)

    global grass
    grass = Grass()
    game_world.add_object(grass, 0)

    global balls
    balls = [Ball() for i in range(10)] + [BigBall() for i in range(10)]
    game_world.add_objects(balls, 1)
```

grass.py


```
class Grass:

    def draw(self):
        self.image.draw(400, 30)
        self.image.draw(1200, 30)
        draw_rectangle(*self.get_bb())

    def get_bb(self):
        return 0, 0, 1600-1, 50
```

play_state.py


```
def update():
    for game_object in game_world.all_objects():
        game_object.update()
    for ball in balls.copy():
        if collide(boy, ball):
            balls.remove(ball)
            game_world.remove_object(ball)
    for ball in balls:
        if collide(grass, ball):
            ball.stop()
```

바닥과 충돌이 되면 멈춘다!

ball.py

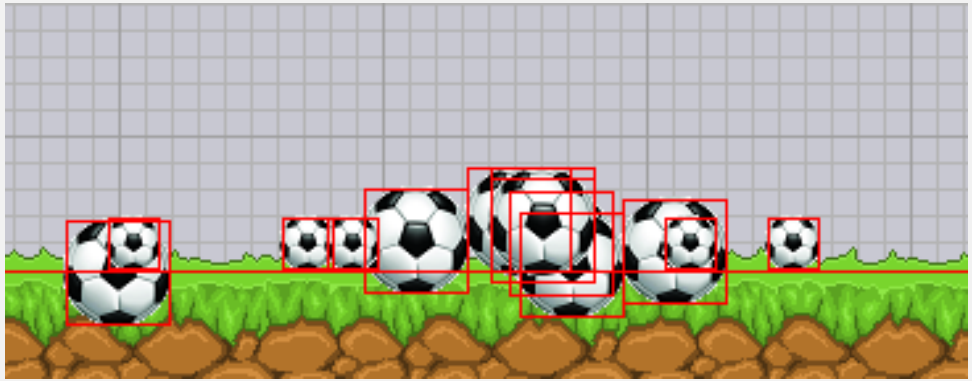

```
class Ball:

    def stop(self):
        self.fall_speed = 0
```

Frame Time 이 길어지면 문제가 발생한다!

```
def update():
    for game_object in game_world.all_objects():
        game_object.update()
    for ball in balls.copy():
        if collide(boy, ball):
            balls.remove(ball)
            game_world.remove_object(ball)
    for ball in balls:
        if collide(grass, ball):
            ball.stop()

    delay(0.9)
```

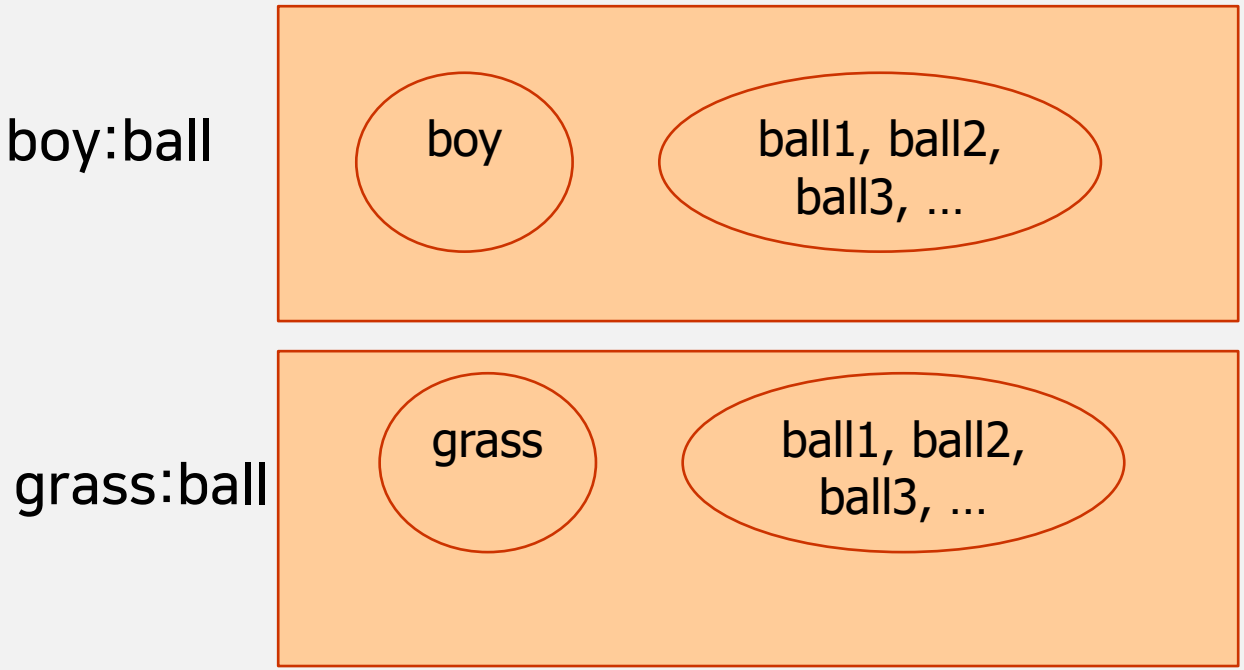
강제로 로직을 느리게 해보자!

망했다! 왜??



좀 더 객체 지향적인 방법은??

- 충돌 처리해야할 대상들을 그룹화해서 처리



game_world.py

```
def add_collision_pairs(a, b, group):

    if group not in collision_group:
        print('Add new group ', group)
        collision_group[group] = [ [], [] ] # list of list : list pair

    if a:
        if type(b) is list:
            collision_group[group][1] += b
        else:
            collision_group[group][1].append(b)

    if b:
        if type(a) is list:
            collision_group[group][0] += a
        else:
            collision_group[group][0].append(a)
```

```
def all_collision_pairs():
    for group, pairs in collision_group.items():
        for a in pairs[0]:
            for b in pairs[1]:
                yield a, b, group
```

```
def remove_object(o):
    for layer in objects:
        try:
            layer.remove(o)
            remove_collision_object(o)
            del o
            return
        except:
            pass
    raise ValueError('Trying destroy non existing object')
```

```
def remove_collision_object(o):
    for pairs in collision_group.values():
        if o in pairs[0]:
            pairs[0].remove(o)
        if o in pairs[1]:
            pairs[1].remove(o)
```

충돌 그룹 추가

```
def enter():
    global boy, grass
    boy = Boy()
    grass = Grass()
    game_world.add_object(grass, 0)

    global balls
    balls = [Ball() for i in range(10)]
    game_world.add_objects(balls, 1)
    game_world.add_object(boy, 1)


    game_world.add_collision_pairs(boy, balls, 'boy:ball')
```

충돌 감지에 따른 충돌 처리

```
def update():
    for game_object in game_world.all_objects():
        game_object.update()

    for a, b, group in game_world.all_collision_pairs():
        if collide(a, b):
            print('COLLISION ', group)
            a.handle_collision(b, group)
            b.handle_collision(a, group)
```

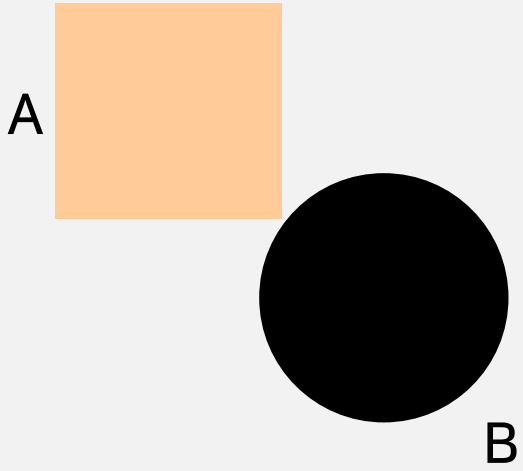
객체별 충돌 처리

```
class Boy:



    def handle_collision(self, other, group):
        pass
```

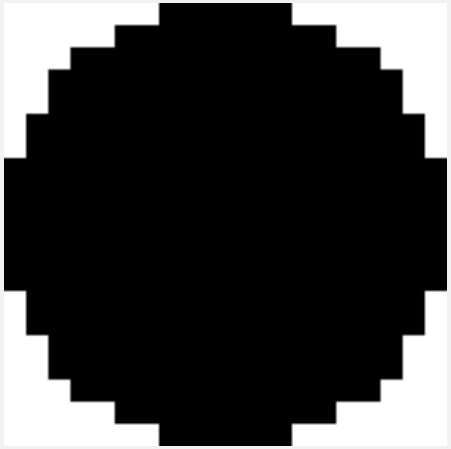
```
class Ball:



    def handle_collision(self, other, group):
        if group == 'boy:ball':
            game_world.remove_object(self)
```

바운딩 박스를 이용한, 픽셀 단위 정밀도를 가지는 충돌 검사

- 예) 사각형 A와 원 B의 충돌 검사
  - 픽셀 단위로 일일이 비교하면, A의 픽셀수 x B의 픽셀수 만큼의 비교가 필요.

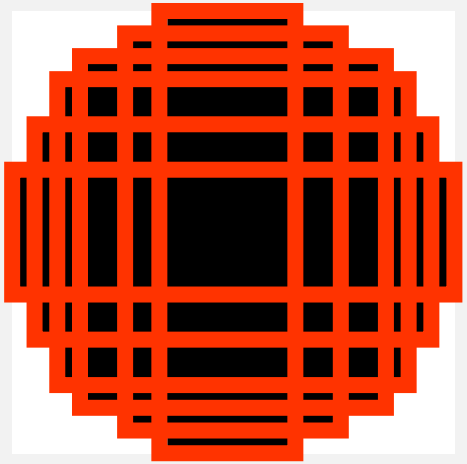


원을 확대하면



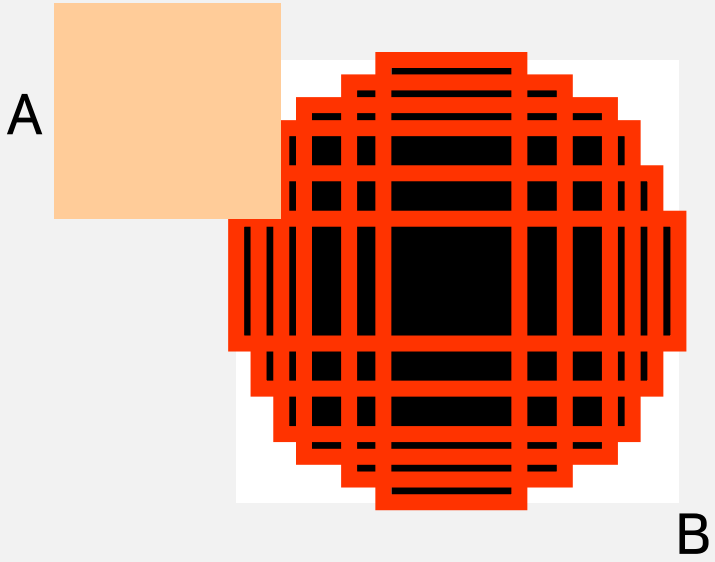
B

원 이미지를 6개의 사각형으로 나타낼 수 있다!!



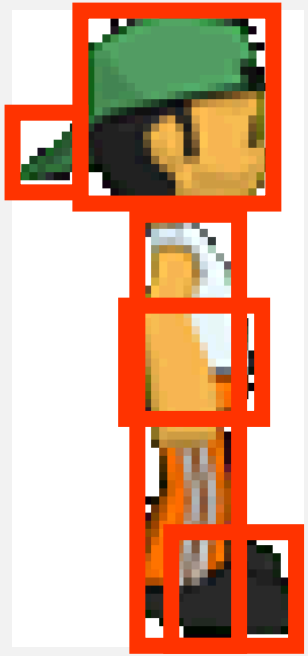
B

사각형 A와 B를 구성하는 여섯개의 사각형을 비교하면 된다.



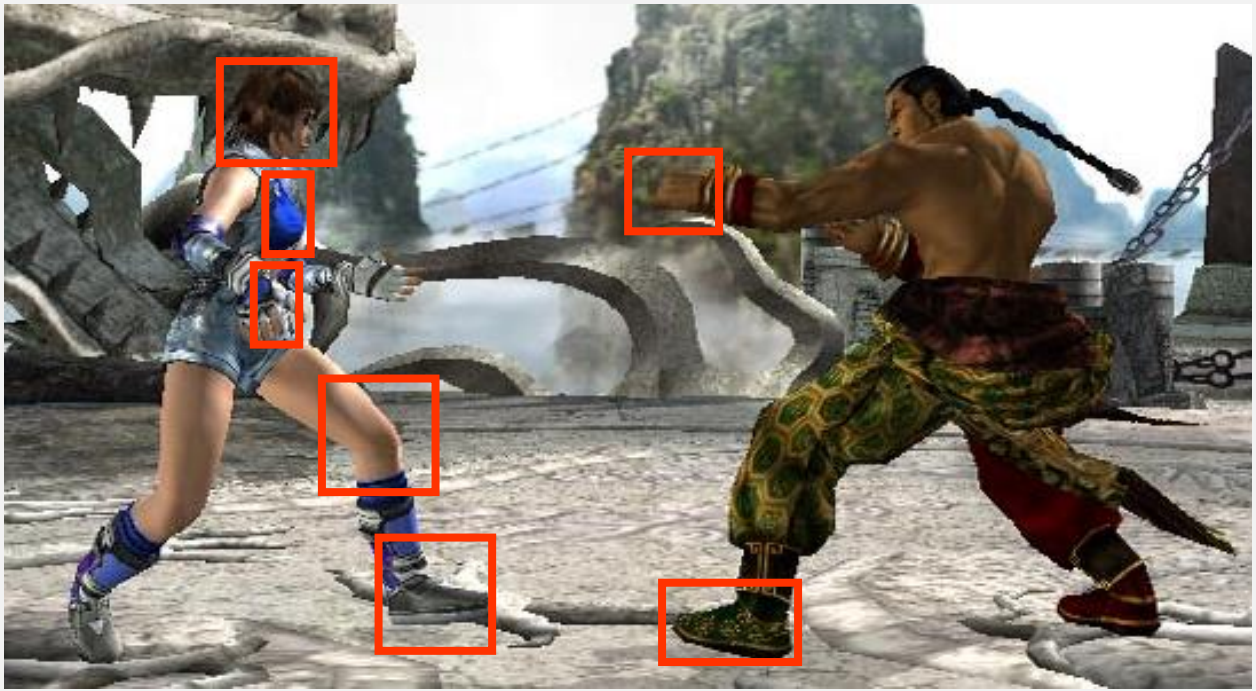
충돌 검사의 실제 적용 방법 (1)

- 정확도를 높이면 한편, 속도 측면에서도 효율적으로 하기 위해, 오브젝트를 적절한 개수의 바운딩 박스로 나눈다.
- 잘게 나누면 나눌수록, 정확도는 높아진다.



충돌 검사의 실제 적용 방법 (2)

- 게임의 특성에 따라 필요한 부분만 바운딩 박스를 적용한다.
  - 격투 대전 게임에서 가격에 사용되는 손 또는 발 부분, 가격이 가해지는 머리, 복부, 배 부분만을 바운딩 박스로 적용.



충돌 처리의 활용

- 트리거(Trigger)
  - 특정 위치에 캐릭터가 들어갈 경우, 이벤트를 발생