

2D 게임 프로그래밍

# 제9강 충돌 처리

이대현  
한국산업기술대학교



# 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

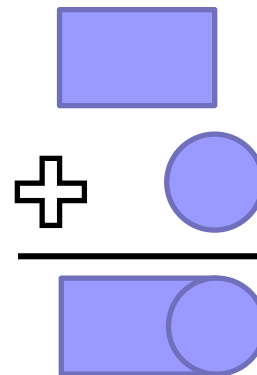
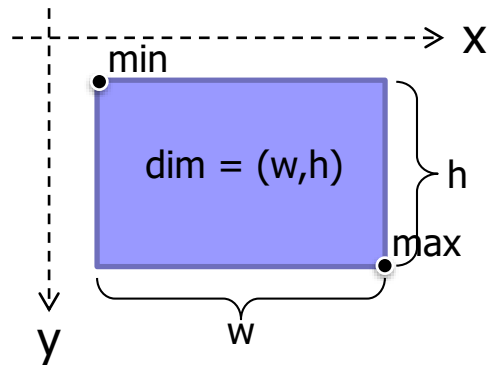
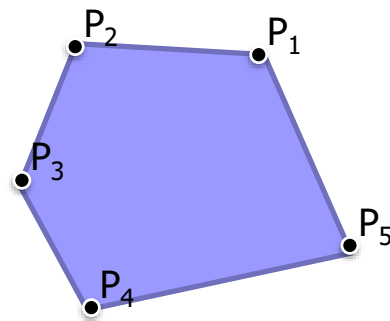
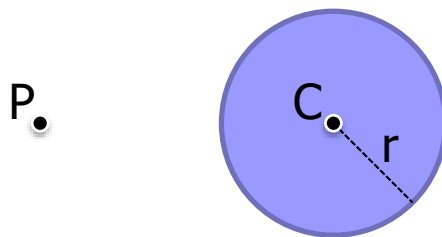
- 게임 상의 오브젝트 간에 충돌이 발생했는지를 검사하는 것.
- 모든 게임에서 가장 기본적인 물리 계산.
  - 슈팅, 발차기, 펀치, 때리기, 자동차 충돌
  - 맵 상의 길 이동
- 기본적으로 시간이 많이 소요되기 때문에, 게임의 오브젝트의 특성에 따라 각종 방법을 통해 최적화해주어야 함.
  - $O(N^2)$  알고리즘

# 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?

# 2D 관점에서 충돌 검사의 대상

- 점
- 원
- 사각형
- 볼록 다각형
- 복합 도형

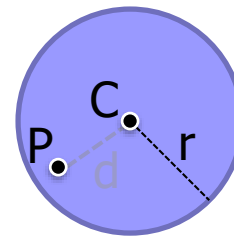
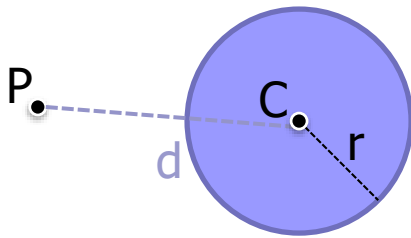


# 픽셀 단위의 충돌 검사

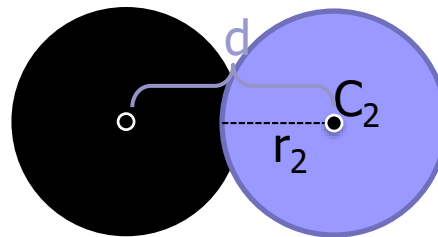
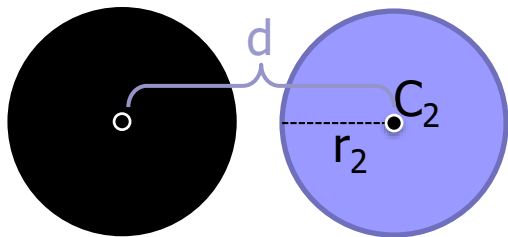


- 두 개의 오브젝트들의 모든 점들을 일일이 비교.
- 가장 정확함.
- 각 오브젝트들의 픽셀수를 곱한 만큼의 계산 시간이 소요됨.
  - 캐릭터 픽셀 수 x 공 픽셀 수

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



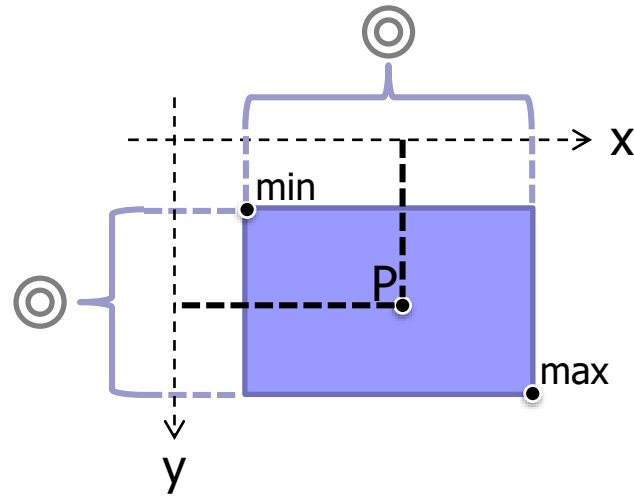
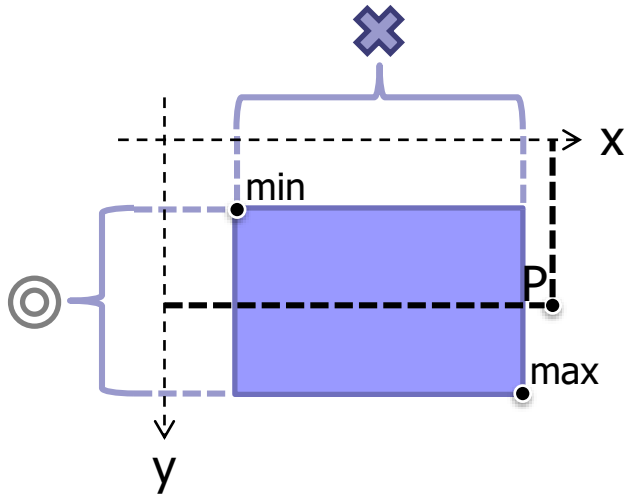
$$\|C_2 - C_1\|^2 \leq (r_1 + r_2)^2$$





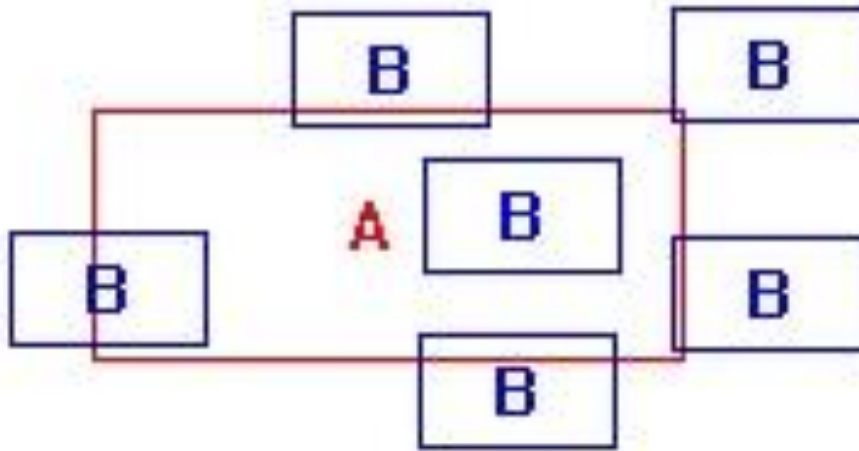
# 점과 사각형(AAB: Axis Aligned Box)

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

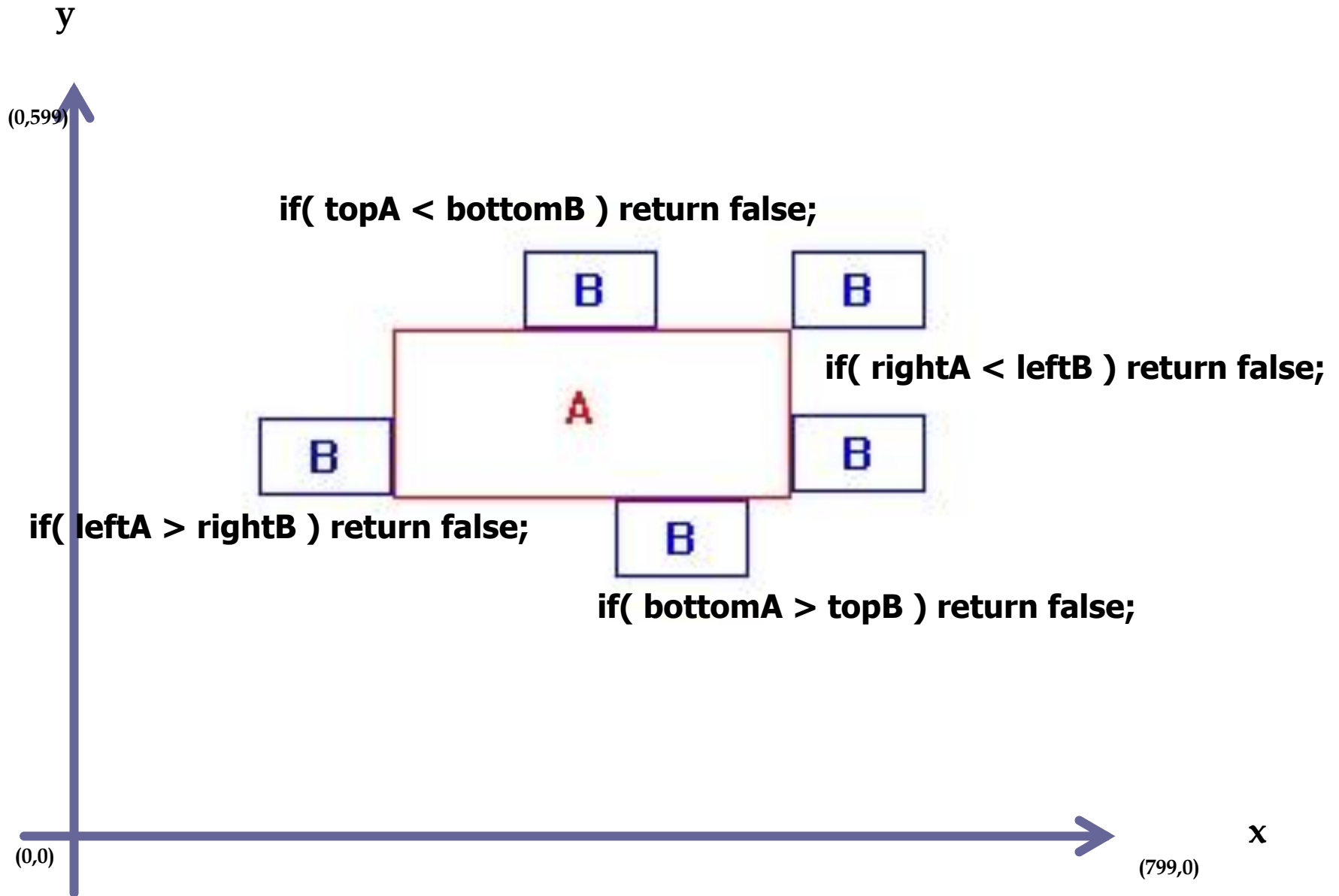


# 사각형과 사각형

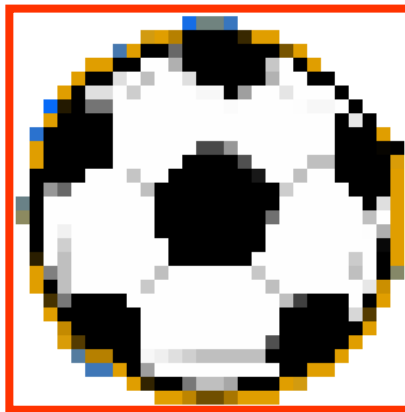
충돌이 있는 경우, B의 옆면들 중 적어도 하나는 A의 안쪽에 있다.



# 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.

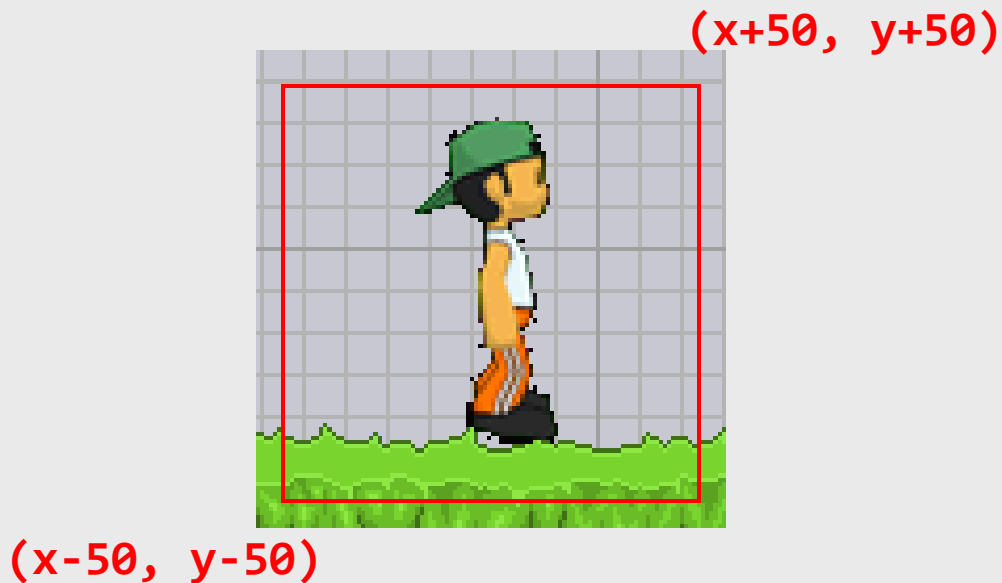


# 충돌 검사



```
class Boy:
```

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





```
class Ball:
```

```
    image = None;
```

```
    def __init__(self):
```

```
        self.x, self.y = random.randint(200, 790), 60
```

```
        if Ball.image == None:
```

```
            Ball.image = load_image('ball21x21.png')
```

```
    def update(self, frame_time):
```

```
        pass
```

```
    def draw(self):
```

```
        self.image.draw(self.x, self.y)
```

```
    def get_bb(self):
```

```
        return self.x - 10, self.y - 10, self.x + 10, self.y + 10
```

# collision.py (1)



```
def create_world():
    global boy, grass, balls
    boy = Boy()
    balls = [Ball() for i in range(10)]
    grass = Grass()

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()

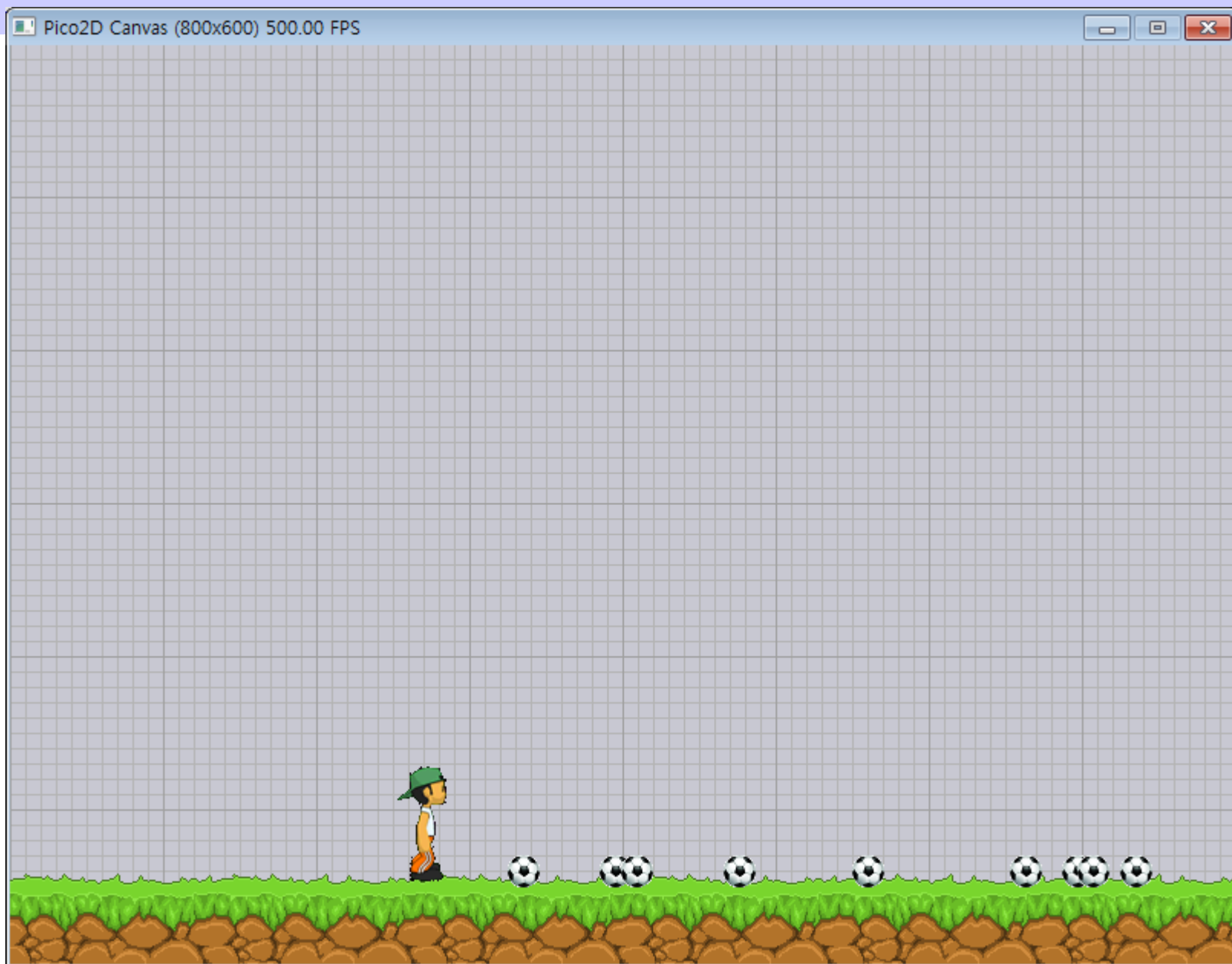
    if left_a > right_b: return False
    if right_a < left_b: return False
    if top_a < bottom_b: return False
    if bottom_a > top_b: return False

    return True
```





```
def update(frame_time):  
    boy.update(frame_time)  
    for ball in balls:  
        ball.update(frame_time)  
  
    for ball in balls:  
        if collide(boy, ball):  
            print("collision")
```





# 충돌 처리

# 충돌 처리

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

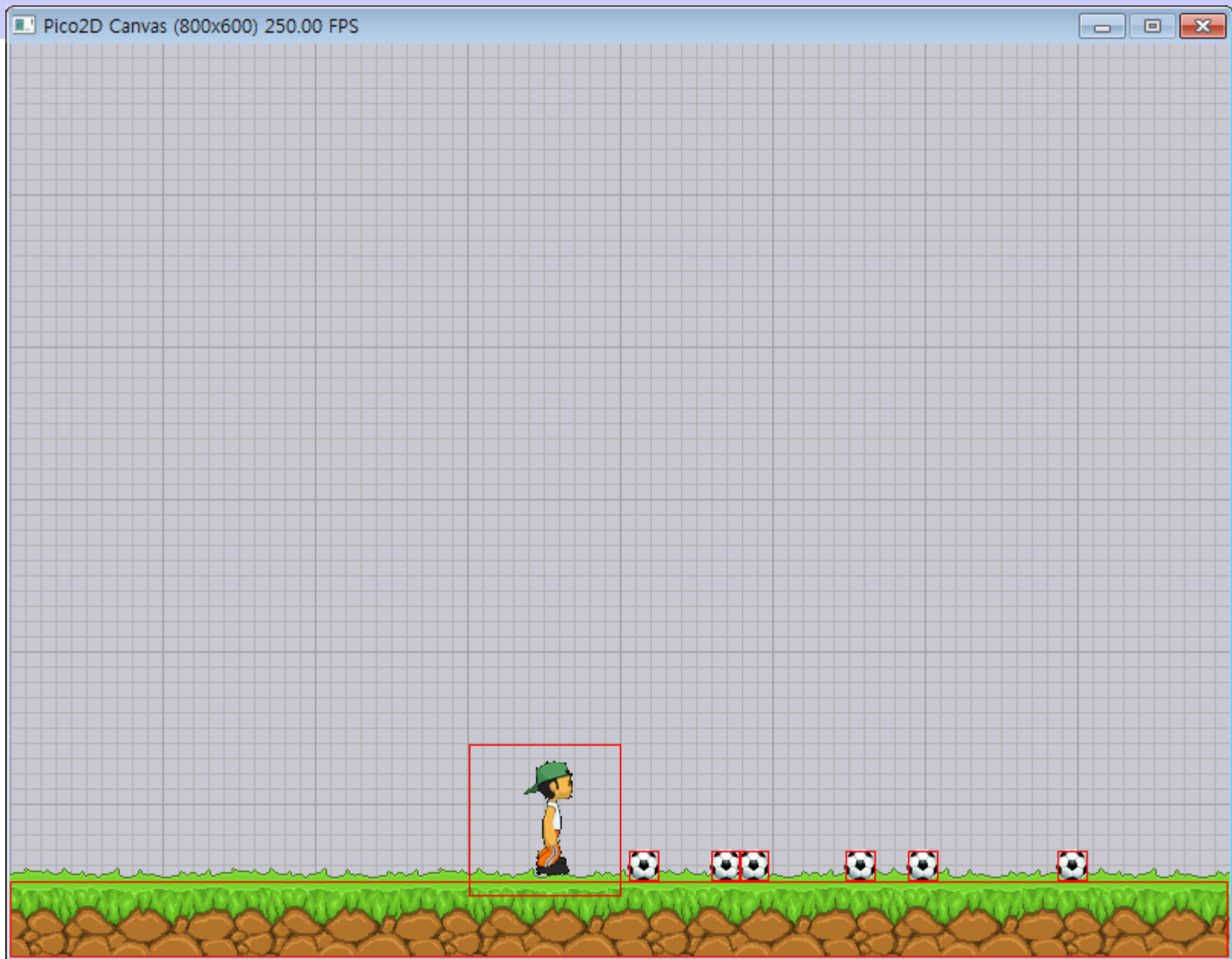




```
def update(frame_time):  
    boy.update(frame_time)  
    for ball in balls:  
        ball.update(frame_time)  
  
    for ball in balls:  
        if collide(boy, ball):  
            balls.remove(ball)
```



# 디버그를 위한 충돌 박스 그리기





```
class Boy:
```

```
    def draw_bb(self):  
        draw_rectangle(*self.get_bb())
```



```
class Ball:
```

```
    def draw_bb(self):  
        draw_rectangle(*self.get_bb())
```





```
def draw(frame_time):  
    clear_canvas()  
    grass.draw()  
    boy.draw()  
    for ball in balls:  
        ball.draw()
```

```
    grass.draw_bb()  
    boy.draw_bb()  
    for ball in balls:  
        ball.draw_bb()
```

```
update_canvas()
```



# 움직이는 물체와 충돌 처리



```
def create_world():  
    global boy, grass, balls, big_balls  
    boy = Boy()  
    big_balls = [BigBall() for i in range(10)]  
    balls = [Ball() for i in range(10)]  
    balls = big_balls + balls  
    grass = Grass()
```

# ball.py



```
class BigBall(Ball):
    image = None
    def __init__(self):
        self.x, self.y = random.randint(100, 700), 500
        self.fall_speed = random.randint(50,120)
        if BigBall.image == None:
            BigBall.image = load_image('ball41x41.png')

    def update(self, frame_time):
        self.y -= frame_time * self.fall_speed

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





```
def update(frame_time):  
    boy.update(frame_time)  
    for ball in balls:  
        ball.update(frame_time)  
  
    for ball in balls:  
        if collide(boy, ball):  
            balls.remove(ball)  
  
    for ball in big_balls:  
        if collide(grass, ball):  
            ball.stop()
```

바닥과 충돌이 되면 멈춘다!



```
class BigBall(Ball):
    image = None
    def __init__(self):
        self.x, self.y = random.randint(100, 700), 500
        self.fall_speed = random.randint(50,120)
        if BigBall.image == None:
            BigBall.image = load_image('ball41x41.png')

    def update(self, frame_time):
        self.y -= frame_time * self.fall_speed

    def stop(self):
        self.fall_speed = 0

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





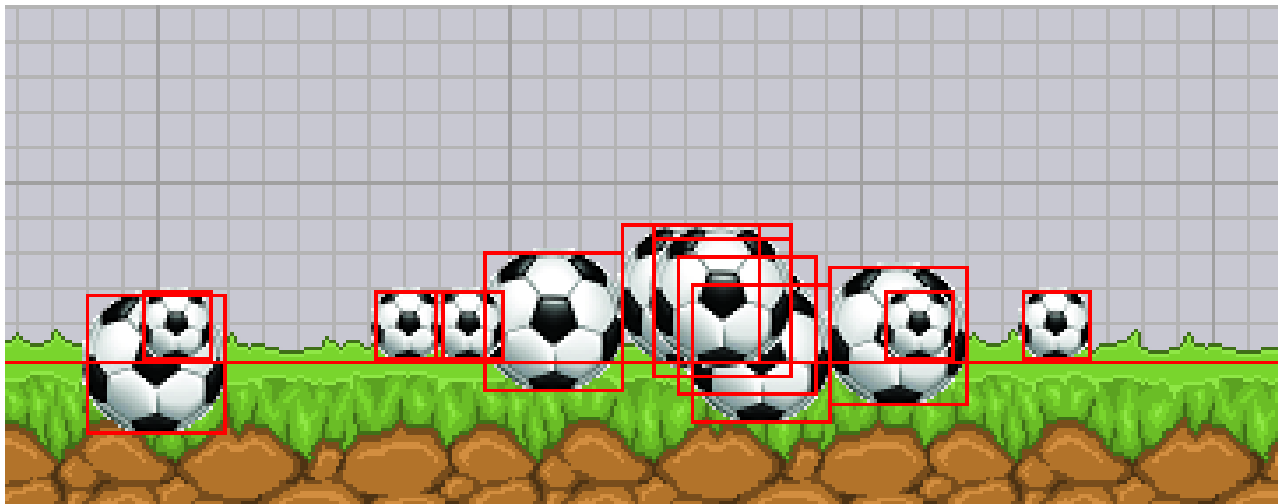
# Frame Time 이 길어지면 문제가 발생한다!

```
def update(frame_time):  
    boy.update(frame_time)  
    for ball in balls:  
        ball.update(frame_time)  
  
    for ball in balls:  
        if collide(boy, ball):  
            balls.remove(ball)  
  
    for ball in big_balls:  
        if collide(grass, ball):  
            ball.stop()
```

delay(0.2)

강제로 로직을 느리게 해보자!

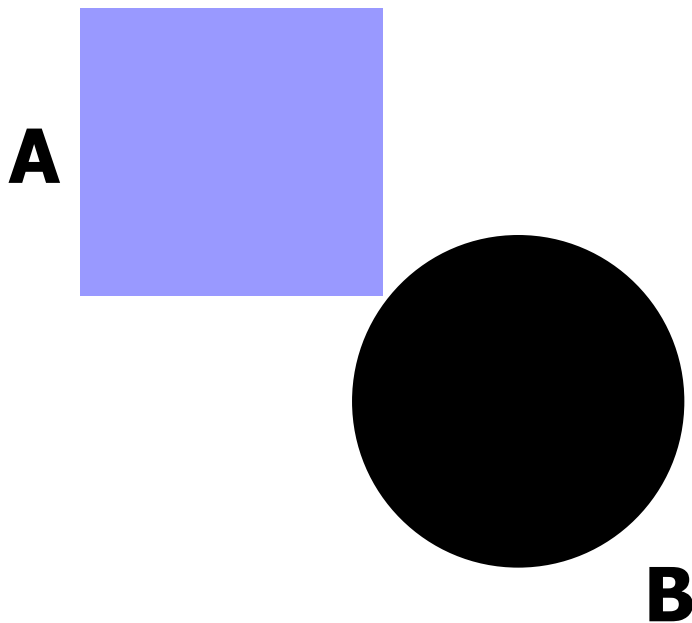
망했잖아! 왜??



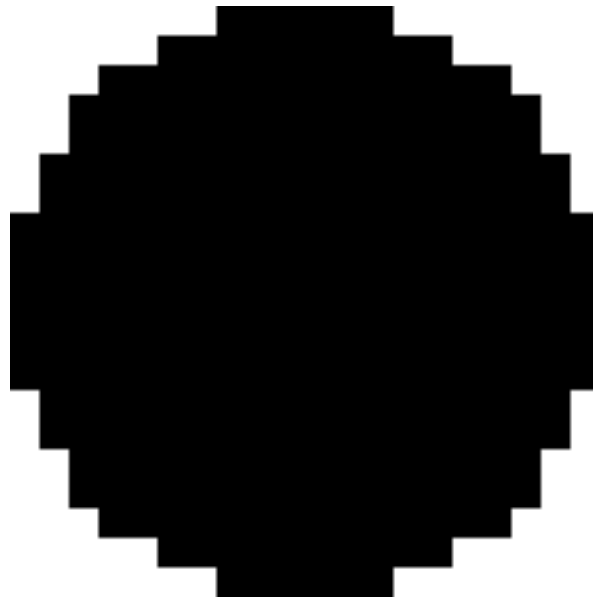
# 바운딩 박스를 이용한, 픽셀 단위 정밀도를 가지는 충돌 검사

- 예) 사각형 A와 원 B의 충돌 검사

- 픽셀 단위로 일일이 비교하면, A의 픽셀수 x B의 픽셀수 만큼의 비교가 필요.

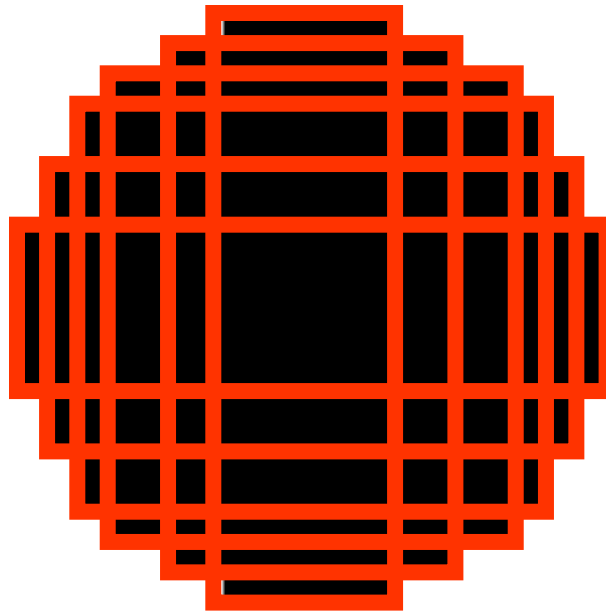


원을 확대하면



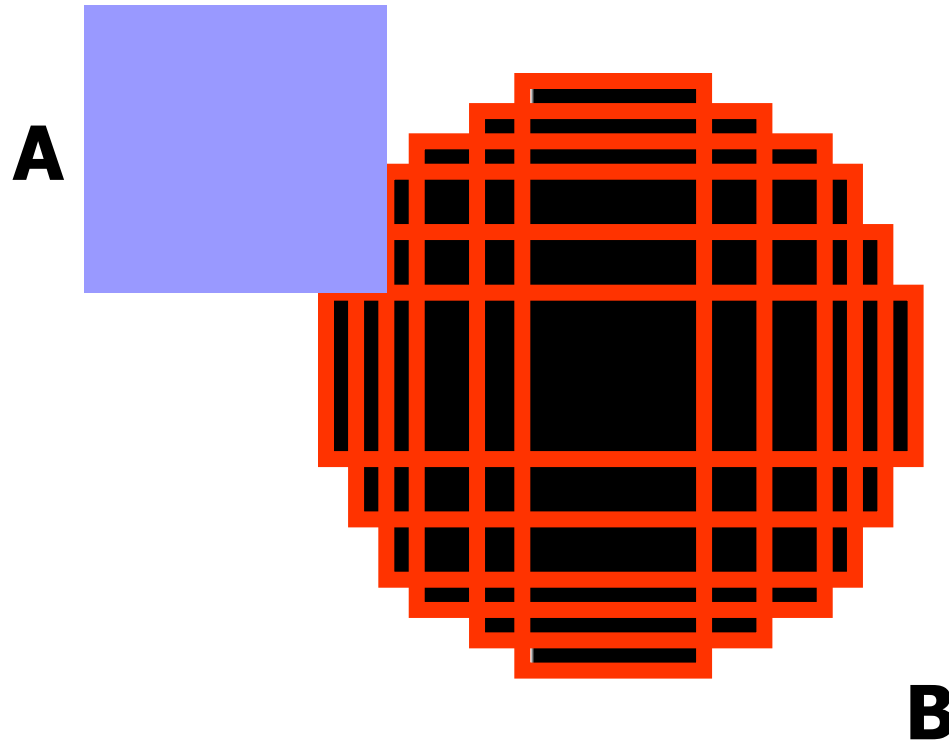
**B**

원 이미지를 6개의 사각형으로 나타낼 수 있다!!



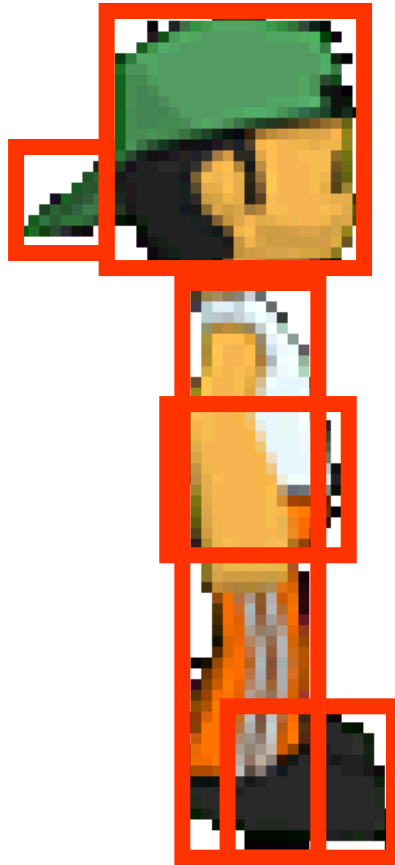
**B**

사각형 A와 B를 구성하는 여섯개의 사각형을 비교하면 된다.



# 충돌 검사의 실제 적용 방법 (1)

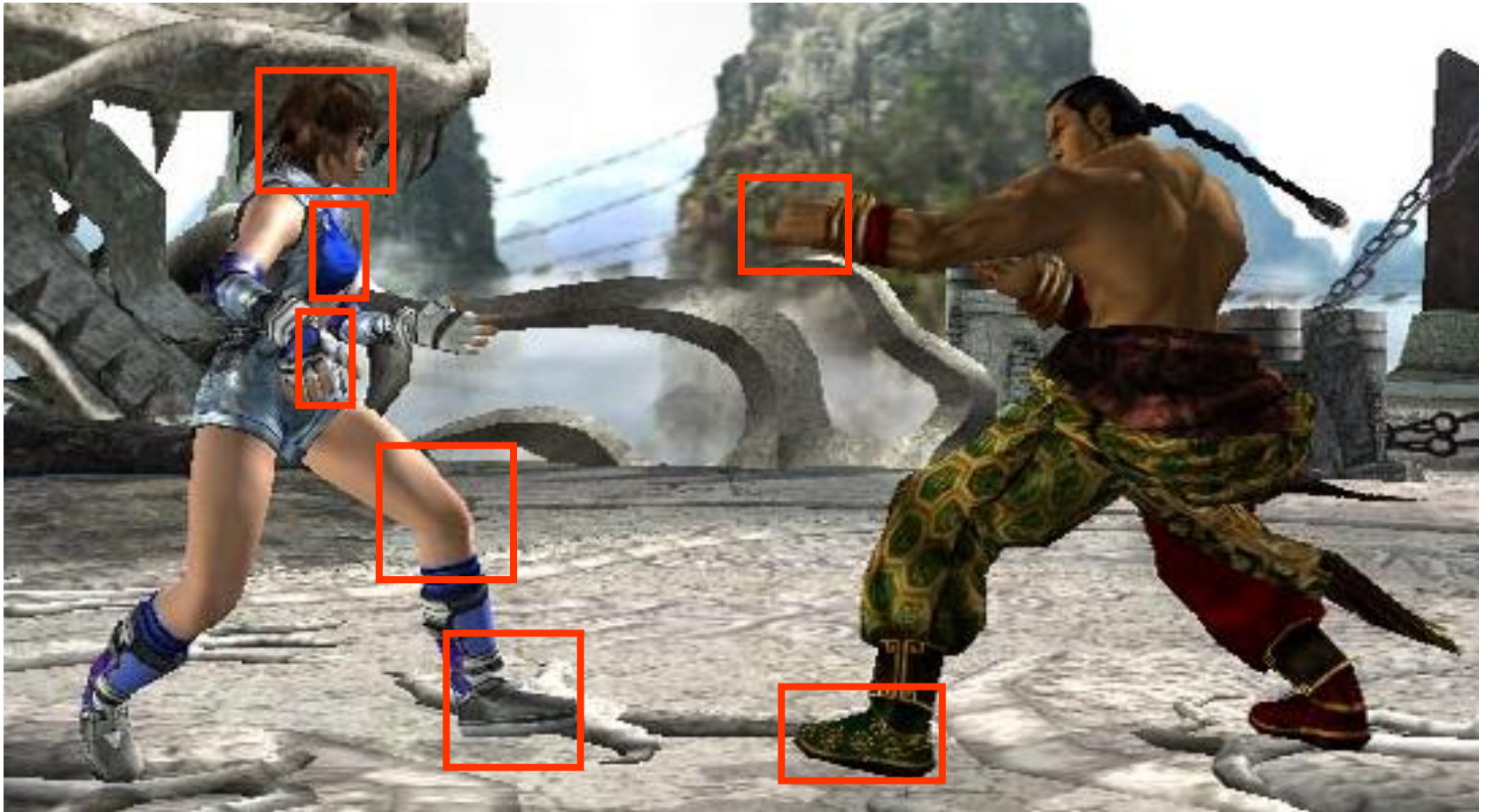
- 정확도를 높이면 한편, 속도 측면에서도 효율적으로 하기 위해, 오브젝트를 적절한 개수의 바운딩 박스로 나눈다.
- 잘게 나누면 나눌수록, 정확도는 높아진다.





## 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

