Lecture #16. 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??

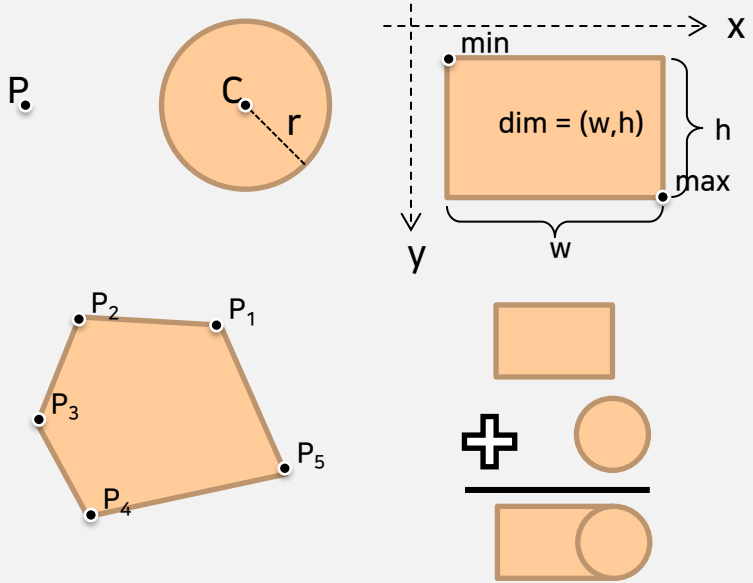
- 사선으로 움직이는 캐릭터가 맵의 벽과 충돌하면?

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

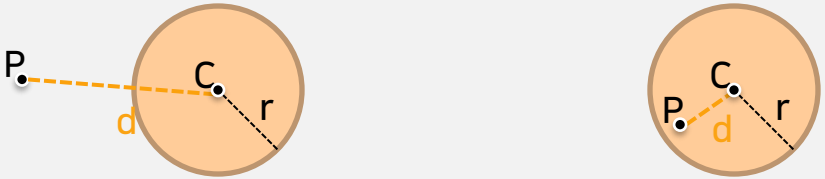
2D 관점에서 충돌 검사의 대상

- 점
- 원
- 사각형
- 볼록 다각형
- 복합 도형



점과 원

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

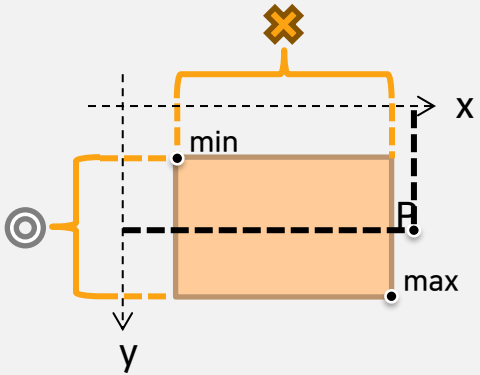


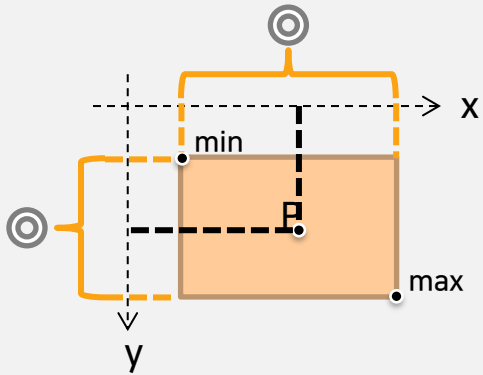
원과 원

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

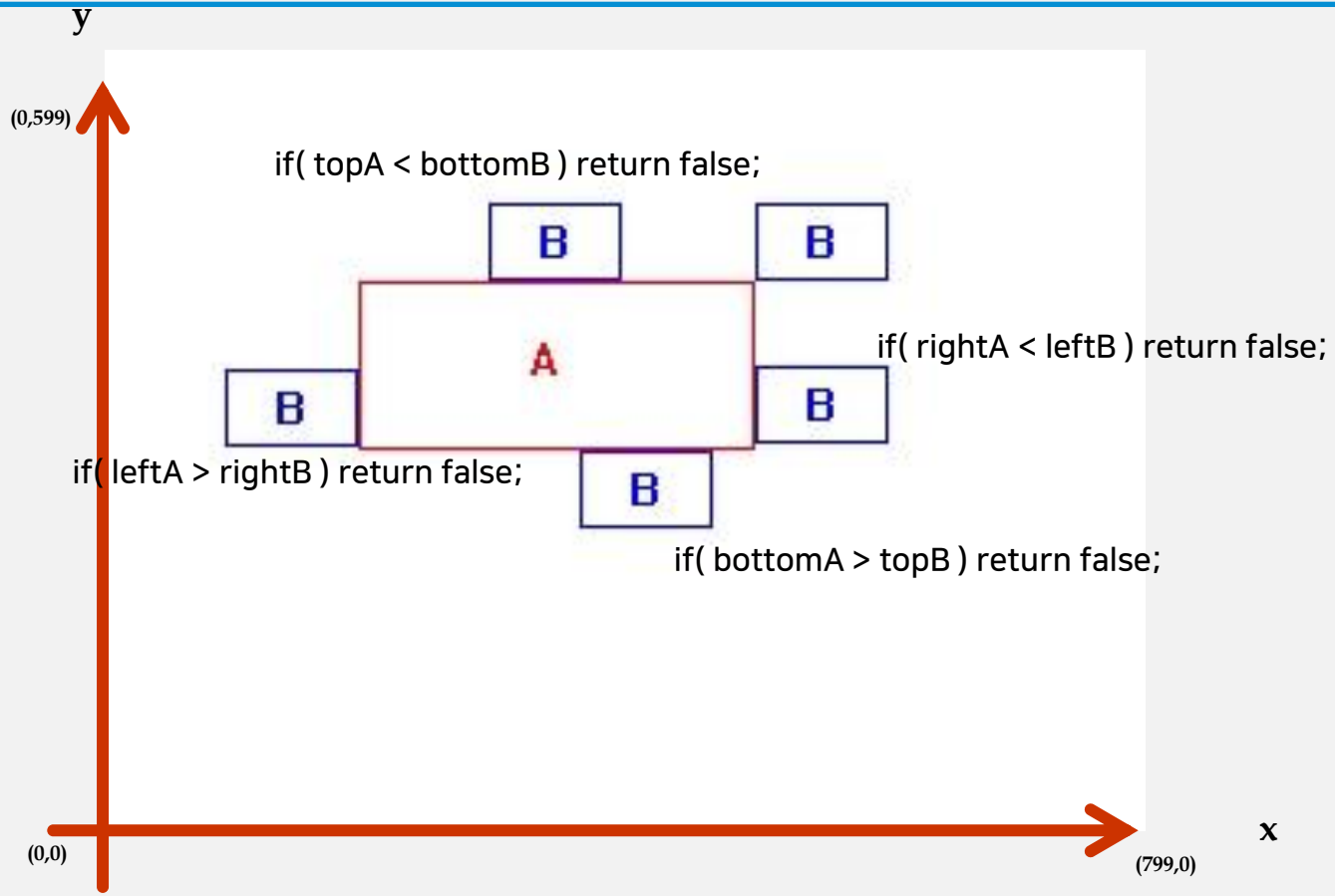
점과 사각형(AAB: Axis Aligned Box)

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





사각형과 사각형



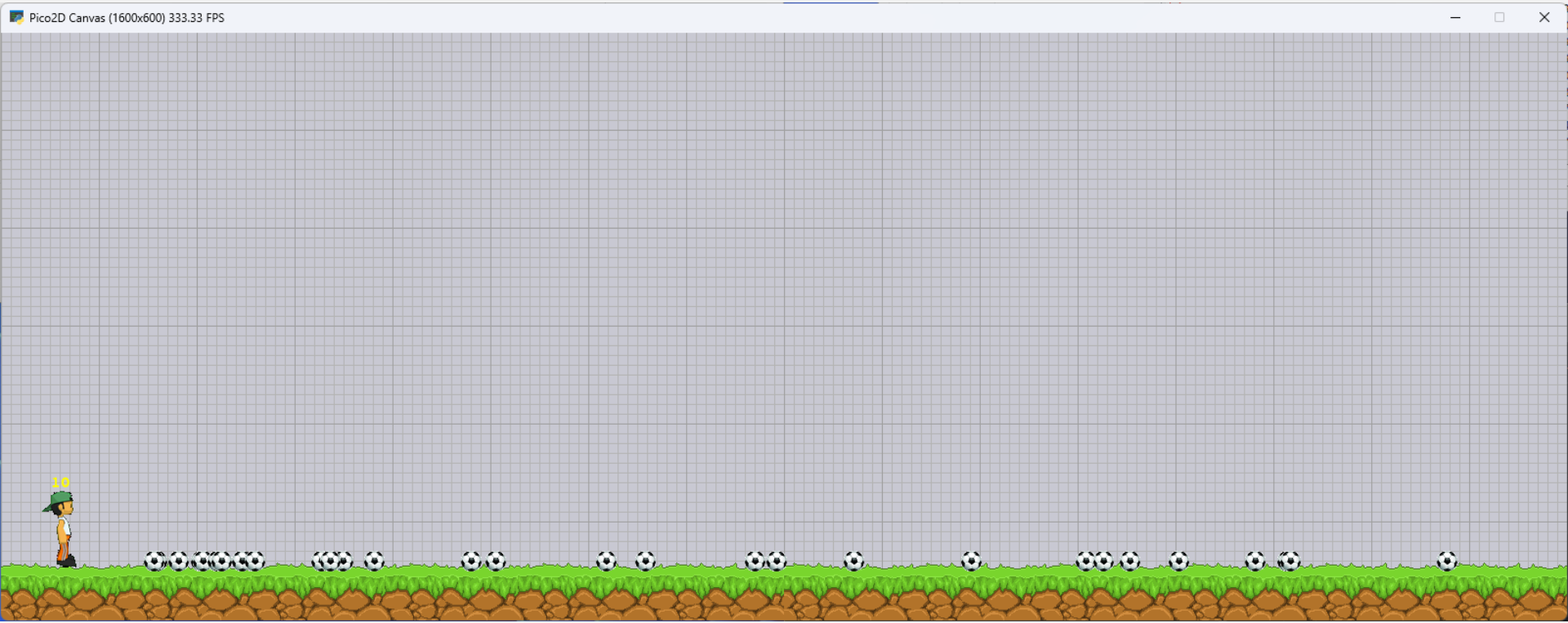
바운딩 박스(Bounding Box)를 이용한 충돌 검사

- 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

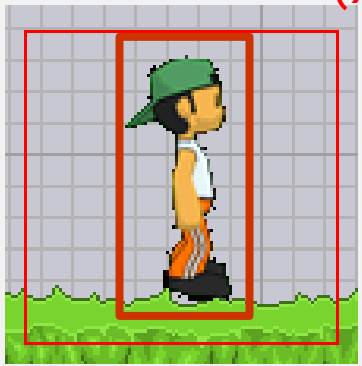
바닥에 공 배치 – play_mode.py


```
def init():

    # 중략

    global balls
    balls = [Ball(random.randint(100, 1600-100), 60, 0) for _ in range(30)]
    game_world.add_objects(balls, 1)
```



boy.py


```
class Boy:

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



ball.py


```
class Ball:

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

grass.py


```
class Grass:

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

game_world.py


```
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_mode.py


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

실습

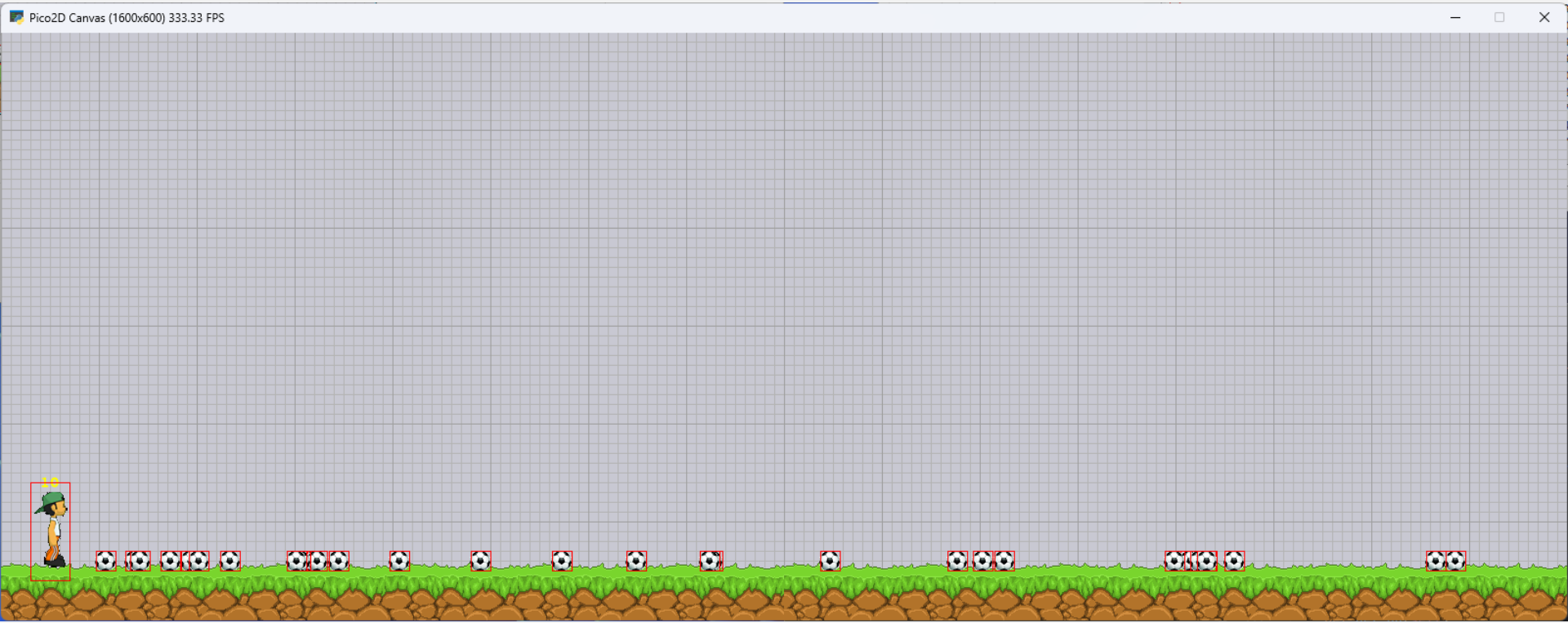
디버그를 위한 충돌박스 그리기

boy.py - class Boy


```
def draw(self):
    self.state_machine.draw()
    self.font.draw(self.x-10, self.y + 50, f'{self.ball_count:02d}', (255, 255, 0))
    draw_rectangle(*self.get_bb())
```

ball.py


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



충돌 처리

충돌 처리

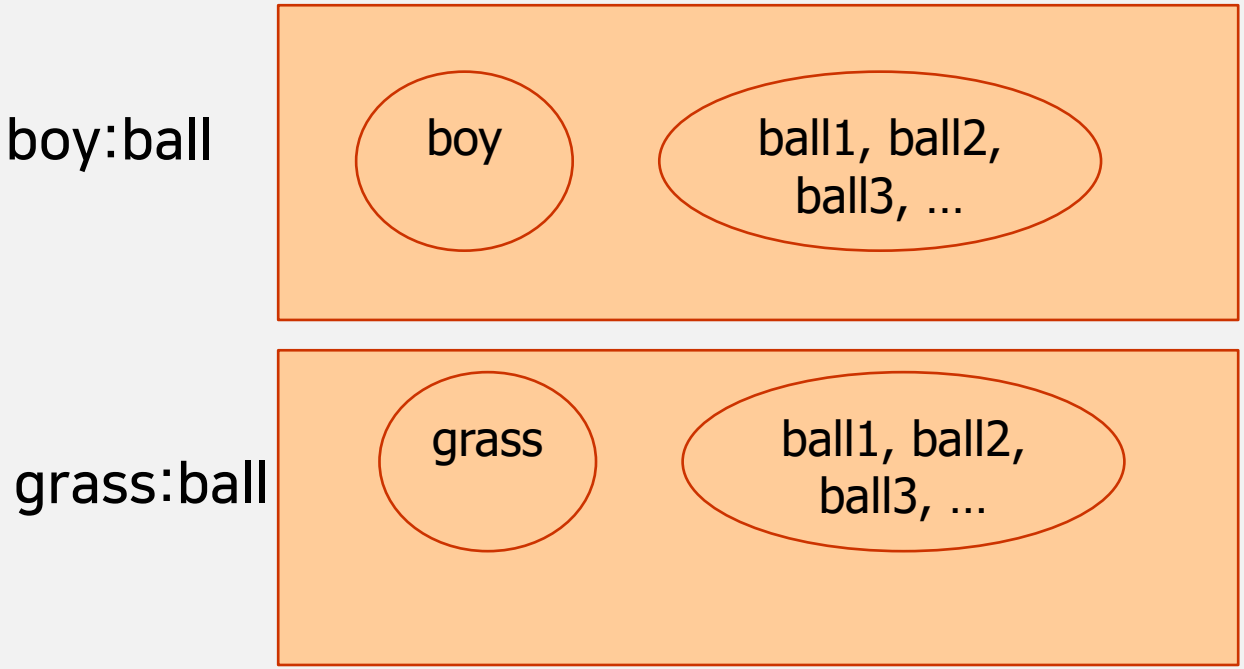
- 충돌 이후에 어떻게 할 것인가?
- 미리 정책을 정해야 함.
- 캐릭터가 공을 만났다… 그래서? 그 다음은?
  - 공을 없앤다..

play_mode.py


```
def update():
    game_world.update()

    for ball in balls.copy():
        if game_world.collide(boy, ball):
            print('COLLISION boy:ball’)
            boy.ball_count += 1
            game_world.remove_object(ball)
            balls.remove(ball)
```

좀 더 객체 지향적인 방법은??

- 충돌 처리가 필요로 하는 두개의 객체 A, B 를 먼저 일일이 등록하고, 나중에 등록된 페어 전체에 대해서 충돌 검사를 수행한 후, 충돌이 있으면 A, B 에게 충돌 처리를 할 수 있도록 지시.
- 충돌 처리해야할 대상들을 그룹화해서 처리



충돌 그룹 구현

game_world.py – 충돌 페어 등록


```
collision_pairs = {}

def add_collision_pair(group, a, b):
    if group not in collision_pairs:
        print(f'Added new group {group}')
        collision_pairs[group] = [ [], [] ]
    if a:
        collision_pairs[group][0].append(a)
    if b:
        collision_pairs[group][1].append(b)
```

game_world.py (2) – 충돌 객체의 제거


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

def remove_object(o):
    for layer in objects:
        if o in layer:
            layer.remove(o)
            remove_collision_object(o)
            del o
            return
    raise ValueError('Cannot delete non existing object')
```

충돌 페어 추가 및 충돌 처리 업데이트


```
def init():
    # 중략
    game_world.add_collision_pair('boy:ball', boy, None)
    for ball in balls:
        game_world.add_collision_pair('boy:ball', None, ball)


def update():
    game_world.update()
    game_world.handle_collisions()
```

game_world.py - 충돌 감지에 따른 충돌 처리


```
def handle_collisions():
    for group, pairs in collision_pairs.items():
        for a in pairs[0]:
            for b in pairs[1]:
                if collide(a, b):
                    a.handle_collision(group, b)
                    b.handle_collision(group, a)
```

객체별 충돌 처리 – 소년 boy.py


```
def handle_collision(self, group, other):
    if group == 'boy:ball':
        self.ball_count += 1
```

객체별 충돌 처리 – 볼 ball.py

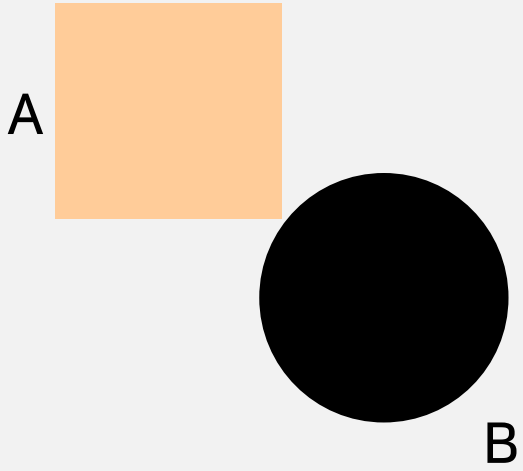

```
def handle_collision(self, group, other):
    if group == 'boy:ball':
        game_world.remove_object(self)
```

충돌 검사 및 처리 기본 절차

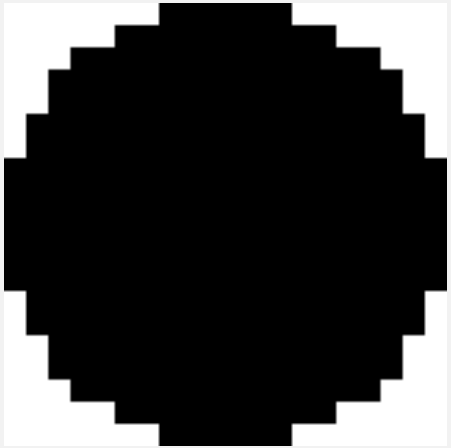
- 충돌 처리가 필요한 객체에 대해서 충돌 영역 정의
- 디버그를 위해서 충돌 영역을 시각화할수 있도록 설정
- 공간에 이미 존재하는 여러 객체 들 중 충돌 처리가 필요한 두개의 객체 A, B 를 골라서 등록
- 만약 게임 실행 중에 생성된 객체에 대한 충돌 처리가 필요하면, 그때마다 실시간으로 A, B 를 등록. 만약 한쪽이 이미 등록된 상태라면, None 으로 처리.
- 등록된 모든 충돌 페어에 대해서 매 프레임마다 충돌 검사를 실시.
- 충돌이 발생한 경우, A,B 각각에 대해서 충돌 처리하도록 지시

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.

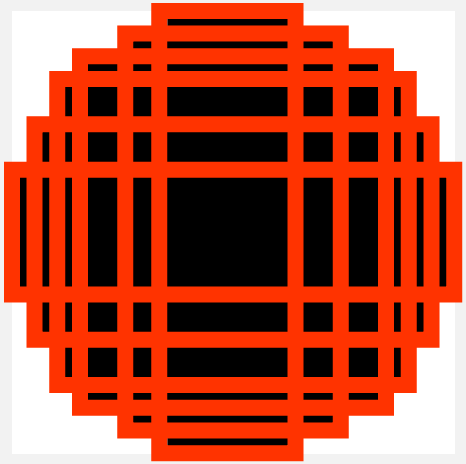


원을 확대하면



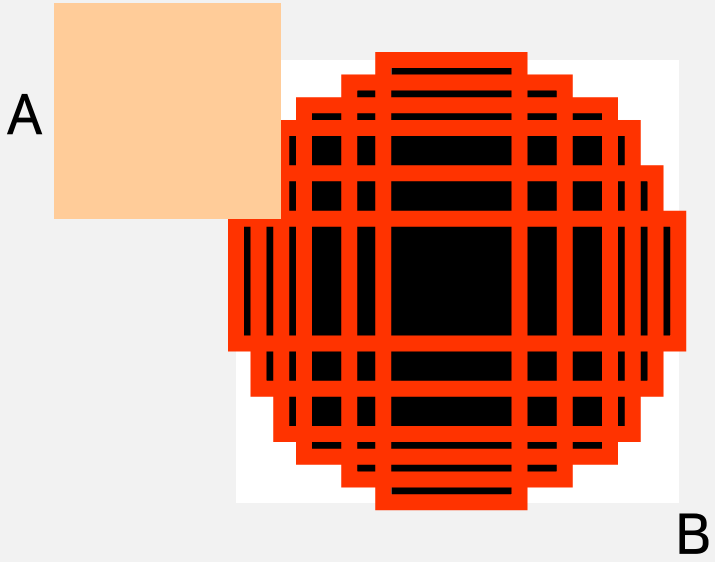
B

원 이미지를 6개의 사각형으로 나타낼 수 있다!!



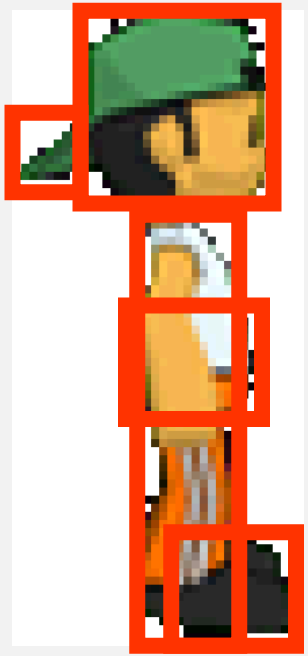
B

사각형 A와 B를 구성하는 여섯개의 사각형을 비교하면 된다.



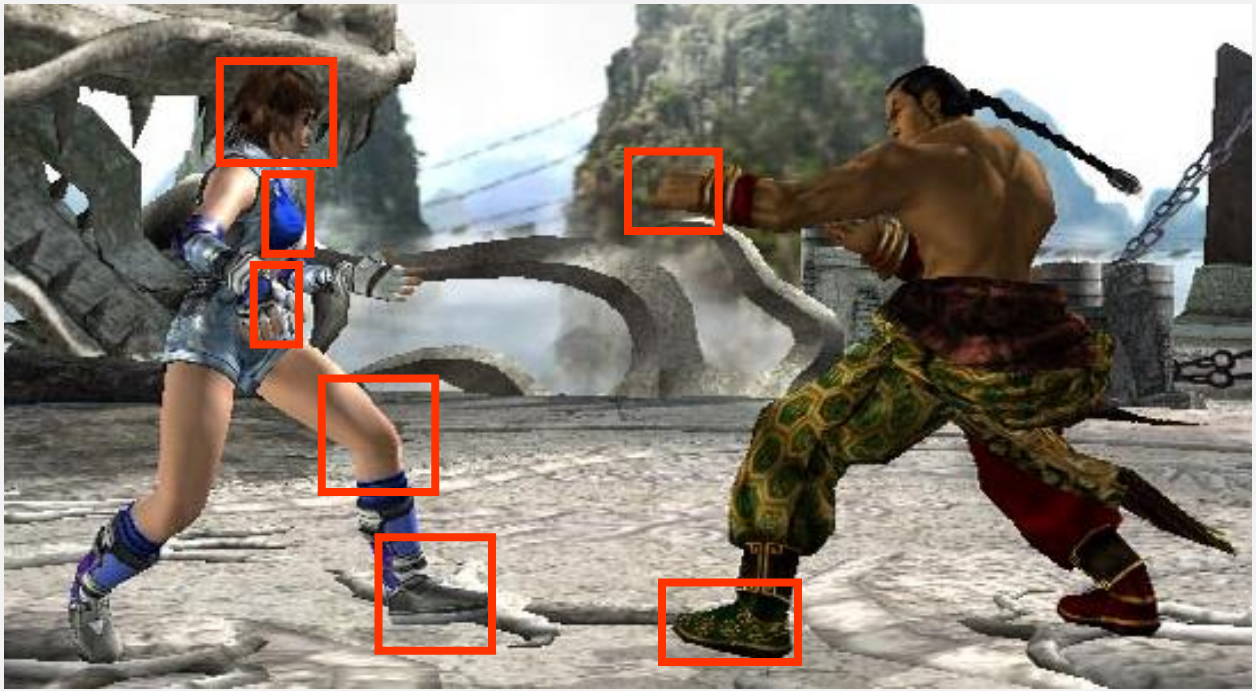
충돌 검사의 실제 적용 방법 (1)

- 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