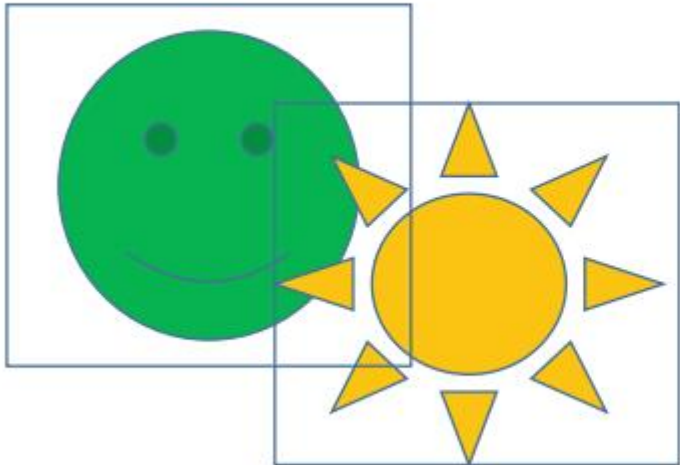


2차원에서의 충돌체크

2022년 1학기 윈도우 프로그래밍

충돌 체크 검사

- 물체간의 충돌이 일어났는지 검사하고 충돌이 일어났을 때 처리하는 프로그램
 - 충돌 영역은 대개 원 또는 사각형으로 설정한다

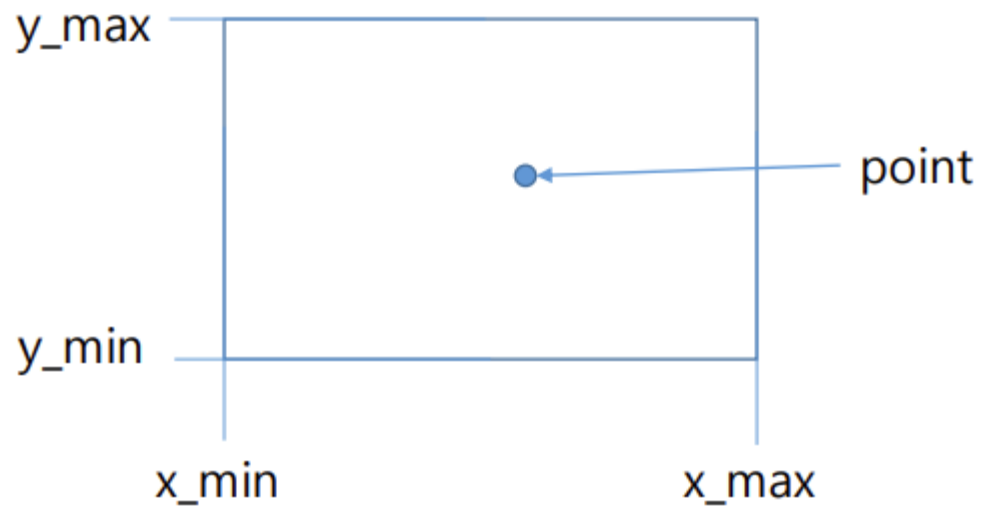


점 대 점 충돌 체크

- 점을 기준으로 비교
 - 두 점이 같은 경우
 - `if ((point1.x == point2.x) && (point1.y == point2.y)) → 충돌`
 - 두 점의 거리가 일정 영역 안에 있는 경우
 - `if (distance (point1, point2) < threshold_value) → 충돌`
이때, distance : 두 점 간의 거리
threshold_value: 정해진 일정 영역

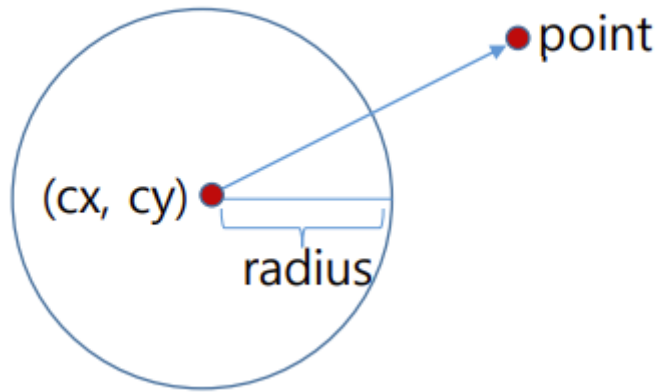
점 대 사각형 충돌 체크

- 점이 사각형의 내부에 있는 경우
 - if ((x_max > point.x) && (point.x > x_min)
&& (y_max > point.y) && (point.y > y_min)) → 충돌



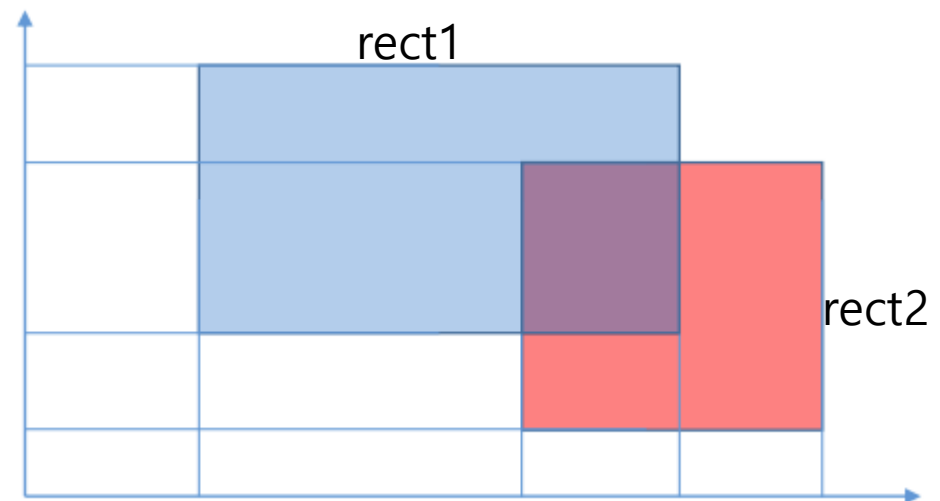
점 대 원 충돌체크

- 점이 원 내부에 있는 경우
 - 원의 중심과 점과의 거리가 원의 반지름보다 작은 경우 → 충돌
 - 원의 중심과 점과의 거리: $\sqrt{(\text{point.x} - \text{cx})^2 + (\text{point.y} - \text{cy})^2}$
- $\sqrt{(\text{point.x} - \text{cx})^2 + (\text{point.y} - \text{cy})^2} < \text{radius} \rightarrow \text{충돌}$



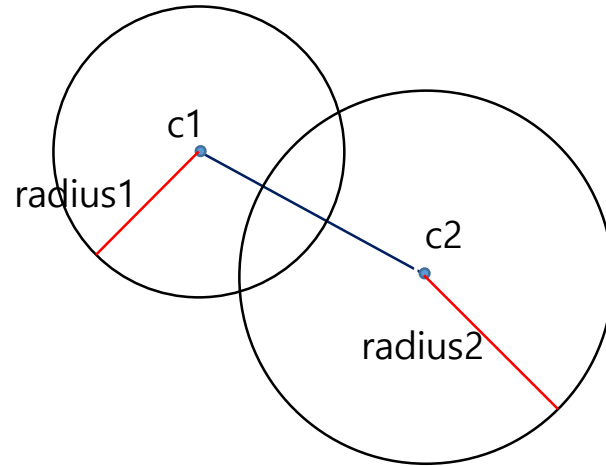
사각형 대 사각형 충돌체크

- 두 사각형의 가장자리를 비교한다.
 - rect1의 left가 rect2의 right보다 작아야 한다.
 - rect1의 top이 rect2의 bottom보다 작아야 한다.
 - rect1의 right가 rect2의 left보다 커야 한다.
 - rect1의 bottom이 rect2의 top보다 커야 한다.
 - if ((rect1.left < rect2.right) && (rect1.top < rect2.bottom) && (rect1.right > rect2.left) && (rect1.bottom > rect2.top)) → 충돌
- 이 때, 첫 번째 사각형: rect1,
두 번째 사각형: rect2



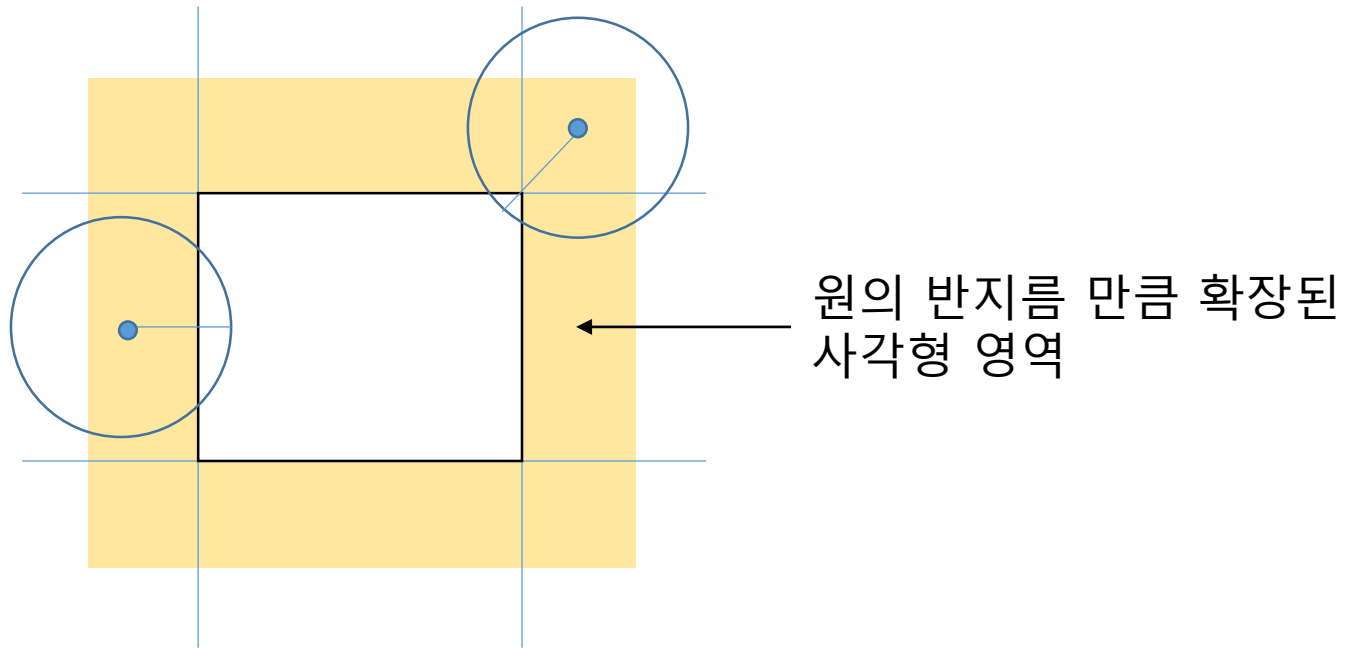
원 대 원 충돌체크

- 두 원의 중심 사이의 거리를 비교한다.
 - 두 원의 중심간의 거리가 반지름의 합과 비교한다.
 - $\sqrt{(c1.x - c2.x)^2 + (c1.y - c2.y)^2} < (\text{radius1} + \text{radius2}) \rightarrow \text{충돌}$



사각형 대 원 충돌체크

- 사각형에 대하여 영역에 따라 검사
 - 사각형을 원의 반지름만큼 확장
 - 원의 중심점이 그 사각형 안에 있다면 → 충돌
 - 원이 사각형 대각선 쪽에 있을 때, 사각형의 꼭지점이 원안에 있다면 → 충돌



대표적 충돌체크 알고리즘

- AABB (Axis-Aligned Bounding Box) 알고리즘
 - 축으로 정렬된 경계 상자 알고리즘
 - 각 축에 대하여 겹쳐지는지 검사하여 충돌체크
 - 간단하게 검사가 가능
 - 물체가 회전하게 되는 경우에는 충돌 경계 상자의 범위가 커지므로 정확한 충돌체크가 어렵다.
- OBB (Oriented Bounding Box) 알고리즘
 - 방향성이 있는 경계 상자 알고리즘
 - 물체의 방향에 따라 경계 상자의 방향을 바꾼다.
 - 충돌을 검사하기 위하여 많은 연산이 필요



충돌 체크 적용

- 게임에서 객체 간의 충돌 체크는 꼭 필요한 요소
 - 게임의 특성, 객체의 모양 / 객체의 개수 등에 따라 적절한 방법의 알고리즘을 선택하여 적용
 - 부분 충돌 체크:
 - 객체의 일부분 적용: 객체의 머리, 손, 발 등 나눠서 충돌 체크를 적용할 수도 있음
 - 공간의 일부분 적용: 검사해야 할 객체들이 많은 경우에는 공간을 분할하여 특정 공간만 충돌 체크를 적용할 수 도 있

유용한 함수

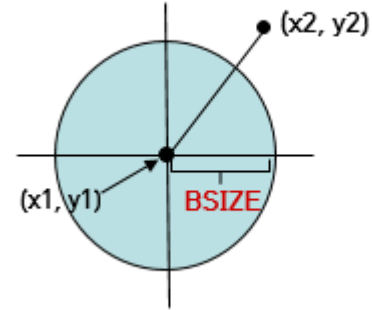
- BOOL **IntersectRect** (LPRECT lprcDest, CONST RECT *lprcSrc1, CONST RECT lprcSrc2);
 - 두 RECT (lprcSrc1, lprcSrc2)가 교차되었는지 검사한다.
 - lprcDest: 교차된 RECT 부분의 좌표값이 저장된다.
- BOOL **PtInRect** (CONST RECT *lprc, POINT pt);
 - 특정 좌표 pt가 lprc 영역 안에 있는지 검사한다.

유용한 코드

- 원과 점의 충돌

```
//--- (x1, y1)과 (x2, y2)간의 길이  
float LengthPts (int x1, int y1, int x2, int y2)  
{  
    return (sqrt((x2-x1)*(x2-x1) +(y2-y1)*(y2-y1)));  
}
```

```
//--- (x1, y1)과 (x2, y2)의 길이가 반지름보다 짧으면 true, 아니면 false  
bool InCircle (int x1, int y1, int x2, int y2)  
{  
    if (LengthPts (x1, y1, x2, y2) < BSIZE)    //--- BSIZE: 반지름  
        return true;  
    else  
        return false;  
}
```



유용한 코드

- 원과 원 충돌

//--- 두 원의 원점 사이의 거리를 구한 후 반지름과 비교

```
bool isCollision (Circle c1, Circle c2)
```

```
{
```

```
    float length = sqrt(((c2.x - c1.x) * (c2.x - c1.x)) + ((c2.y - c1.y) * (c2.y - c1.y)));
```

```
    if ( length <= (c2.radius + c1.radius) )
```

```
        return true;
```

```
    else
```

```
        return false;
```

```
}
```