Argon 충돌처리

충돌범위

충돌체크

1. 충돌검사

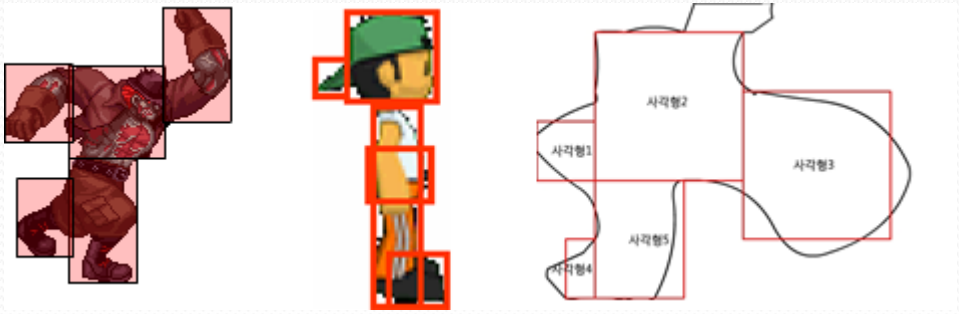
1. 충돌 검사의 기본 개념

- 게임 진행 중에 게임 물체(게임 오브젝트)간의 충돌이 있어났는지를 검사하고 충돌이 일어났을 때 처리하는 프로그램을 말한다.
- 충돌을 검사하기 위해서는 게임 오브젝트를 그대로 사용하지 않고 충돌 감지를 위한 영역(바운딩 박스)를 설정하고 충돌 영역간의 충돌을 감지하게 된다.

- 게임에서 사용되는 이미지(스프라이트 이미지-게임오브젝트)자체는 충돌과 무관하게 사용된다.
- 바운딩 박스 : 이미지에 충돌 효과를 추가하기 위해서는 원 또는 사각형 모양의 영역을 만들어서 적용해 주어야 한다.

※ 충돌체(Collider) : 충돌을 감지하는 영역을 충돌체라고 한다. 충돌체는 충돌 영역이 적용된 오브젝트가 움직일 때 함께 따라서 이동하게 된다.

- 일반적으로 충돌 검사를 위한 영역은 원이나 사각형과 같이 단순하게 설정하는 것이 바람직하다.
- 필요하면 여러 개의 충돌 영역을 조합해서 복잡한 충돌영역을 만들 수도 있지만, 충돌 영역이 복잡하면 충돌 검사에 많은 시간이 소요된다.



2. 기본 충돌 검사 방법

① 점과 원

② 점과 사각형

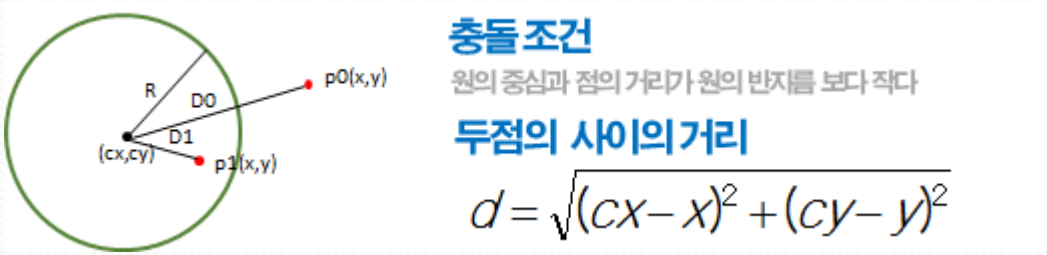
③ 원과 원

④ 사각형과 사각형

⑤ 원과 사각형

2.1 점과 원의 충돌 검사

- 원의 내부에 점이 있을 경우 충돌됨



- 원과 점이 충돌했다는 의미는 점이 원의 영역 안에 포함된다는 의미
- 점이 원안에 있는지를 검사하기 위해서는 원의 원점과 점과의 거리를 비교

원의 중심(원점) 과 충돌 테스트 점과의 거리가 원의 반지름 보다 작으면 원과 점이 충돌 했다고 판단

- 원과 점의 충돌 검사는 독립적인 점이 충돌 했는지 보다, 사각형의 꼭지점등이 원의 내부에 있는지를 판단하는 경우에 많이 사용된다.

```
1.  function hit(x1, y1, x2, y2, dis) {
2.      var distance = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
3.      if (distance < dis) {
4.          return true;
5.      } else {
6.          return false;
7.      }
8.  }

9.  function stateInfo() {
10.     if (hit(cx, cy, x, y, r)) {
11.         ctx.fillText("상태: 충돌", 50, 50);
12.     } else {
13.         ctx.fillText("상태: 안전", 50, 50);
14.     }
15. }
```

2.2 점과 사각형의 충돌 검사

- 사각형의 내부에 점이 있을 경우 충돌 됨

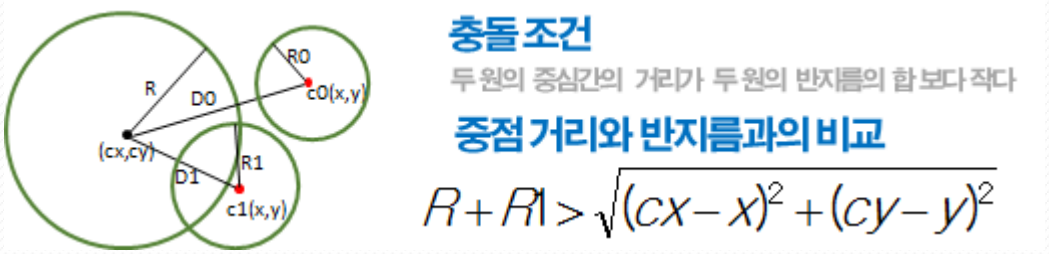
- 어떤 점이 사각형과 충돌했다는 의미는 점이 사각형의 내부에 있다는 의미와 동일
- 점이 사각형의 내부에 있는지를 판단하기 위해서는 점이 사각형 영역을 정의하는 좌표 내부에 있는지 검사

- 점의 x 좌표가 사각형의 왼쪽 좌표 (x0) 보다 크고 오른쪽 좌표(x1) 보다 작고 점의 y 좌표가 사각형의 상단 좌표 (y0) 보다 크고 하단 좌표(y1) 보다 작으면 충돌했다고 판단 할 수 있다.

```
1.  function hit() {
2.      if (x0 < x && x1 > x && y0 < y && y1 > y) {
3.          return true;
4.      } else {
5.          return false;
6.      }
7.  }

8.  function stateInfo() {
9.      var res = hit();
10.     if (res) {
11.         ctx.fillText("상태: 충돌", 50, 50);
12.     } else {
13.         ctx.fillText("상태: 안전", 50, 50);
14.     }
15. }
```

2.3 원과 원의 충돌 검사

- 두 개의 원이 겹쳐지는 경우 충돌 함



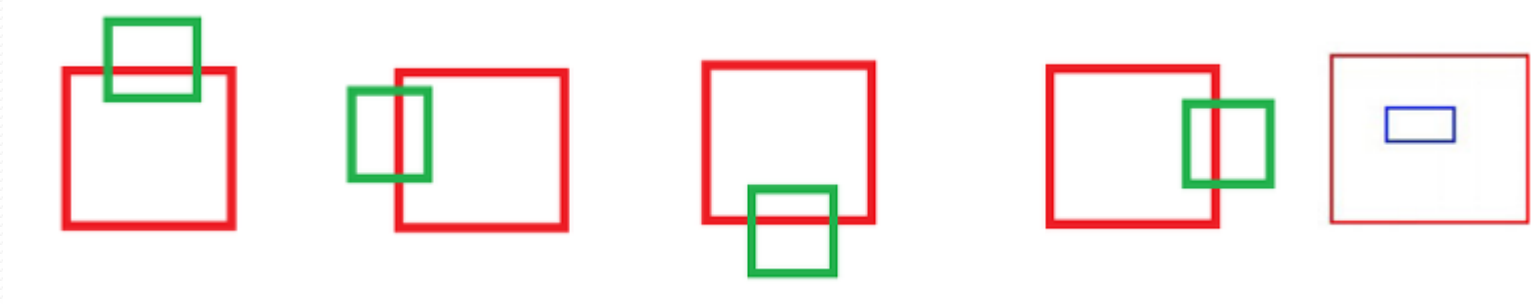
- 원이 충돌했다는 것은 두 개의 원이 겹치는 부분이 생겼다는 것을 의미
- 두 개의 원이 겹쳐지는 경우는 원의 중점간의 거리가 두 원의 반지름의 합보다 작을 경우 발생한다.

- 두 원의 중심간의 거리를 구했을 때 중심간의 거리가 두 원의 반지름의 합 보다 작으므로 두 원이 충돌 했다고 판단 할 수 있다.

```
1.  function hit(x1, y1, x2, y2, dis) {
2.      var distance = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
3.      if (distance < dis) {
4.          return true;
5.      } else {
6.          return false;
7.      }
8.  }

9.  function stateInfo() {
10.     if (hit(cx, cy, x, y, r + r1)) {
11.         ctx.fillText("상태: 충돌", 50, 50);
12.     } else {
13.         ctx.fillText("상태: 안전", 50, 50);
14.     }
15. }
```

2.4 두 사각형의 충돌 검사

- 두 개의 사각형이 겹쳐지는 경우 충돌 함

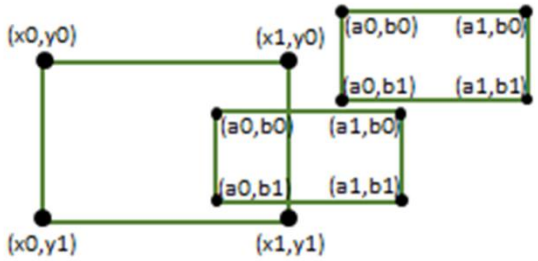


- 사각형의 충돌 검사를 위해서는 좀더 복잡하게 생각해야 할 점들이 있다. 그림과 같이 사각형이 충돌하는 경우는

① 두 사각형의 일부가 겹쳐지는 경우

② 한 쪽 사각형이 다른 쪽 사각형의 내부에 포함되는 경우를 모두 고려해야 한다.

- 충돌 검사의 방법은 아래와 같은 조건이면 충돌 했다고 판단하는 방법이다.



충돌 조건

사각형의 변이 다른 사각형의 내부에 위치 한다.

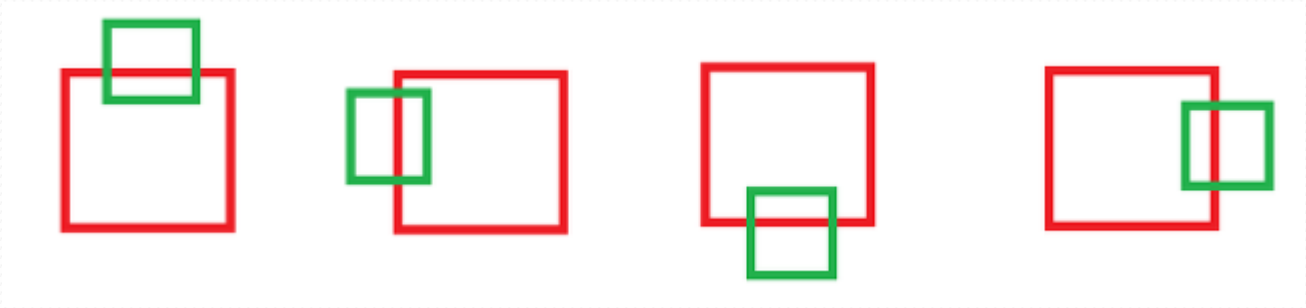
변이 내부에 있는지 검사

(a) 사각형의 왼쪽 변 **x0** 가 다른 사각형의 오른쪽 쪽변 **a1** 보다 작아야 한다.
(b) 사각형의 오른쪽변 **x1** 가 다른 사각형의 왼쪽 변 **a0** 보다 커야 한다.
(c) 사각형의 위쪽변 **y0** 가 다른 사각형의 아래쪽 변 **b1** 보다 작아야 한다.
(d) 사각형의 아래쪽변 **y1** 가 다른 사각형의 아래쪽 변 **b0** 보다 커야 한다.

$(x0 < a1) \&\& (x1 > a0) \&\& (y0 < b1) \&\& (y1 > b0)$

- 1. 첫 번째 사각형의 왼쪽 변이 두 번째 사각형의 오른쪽 변을 넘지 말아야 한다.
- 2. 첫 번째 사각형의 오른쪽 변이 두 번째 사각형의 왼쪽 변을 넘어야 한다.
- 3. 첫 번째 사각형의 위쪽 변이 두 번째 사각형의 아래쪽 변을 넘지 말아야 한다.
- 4. 첫 번째 사각형의 아래쪽 변이 두 번째 사각형의 위쪽 변을 넘어야 한다.

③ ① ④ ②

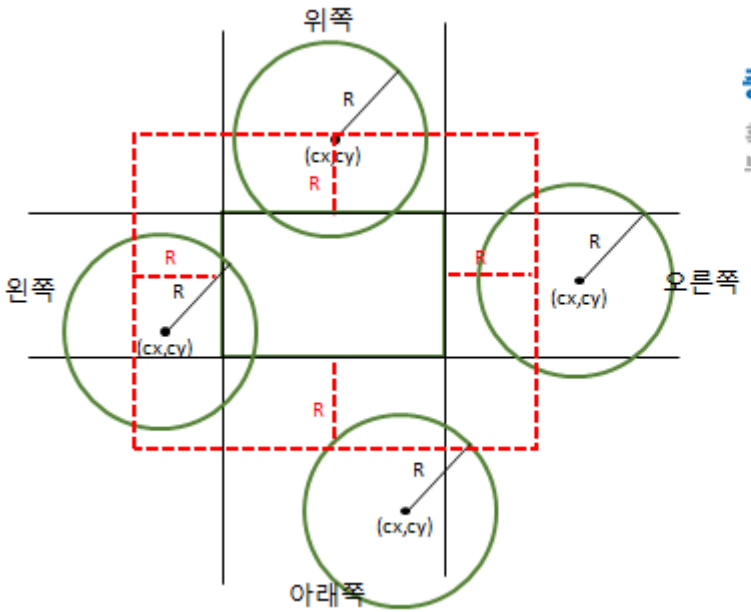


```
1.   function checkHit() {
2.       var nResult = false;
3.
4.       if ((x0 < a1) && (x1 > a0) && (y0 < b1) && (y1 > b0)) {
5.               nResult = true;
6.
7.       }
8.
9.       return nResult;
10.  }
11.
12.  function stateInfo() {
13.      if (checkHit()) {
14.          ctx.fillText("상태: 충돌", 50, 50);
15.      } else {
16.          ctx.fillText("상태: 안전", 50, 50);
17.      }
18.  }
```

2.5 사각형과 원의 충돌

- 원과 사각형이 겹쳐지는 영역이 있는 경우 충돌 함

- 사각형과 원의 충돌은 2 가지 경우로 생각해서 충돌 판정을 한다.
  ① 원이 사각형의 대각선이 아닌 영역(위쪽, 아래쪽, 왼쪽, 오른쪽)에 위치한 경우
  ② 원이 사각형의 대각선 방향에 위치한 경우.



충돌 조건

확장된 사각형(원의 반지름 R 만큼 확장됨- 빨간색)의 내부에 원의 중심이 위치한다.

확장된 사각형의 내부에 원의 중심이 있는지 검사

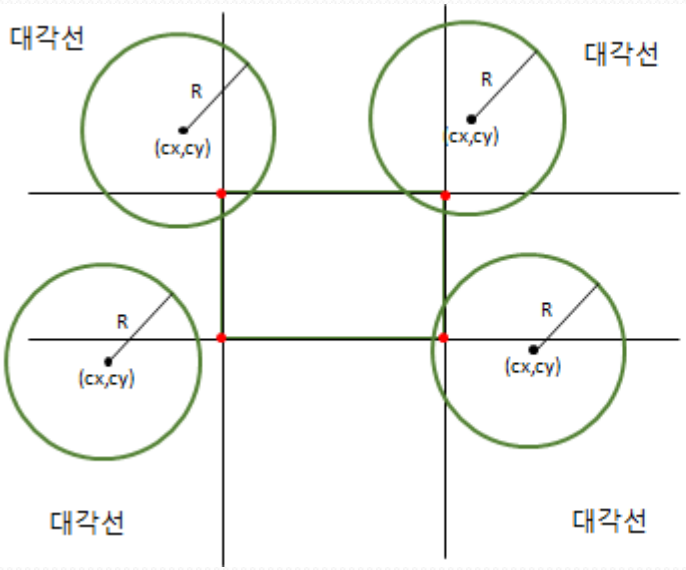
위의 사각형과 점의 충돌을 이용한다.

원이 사각형의 위쪽, 아래쪽, 왼쪽, 오른쪽에 위치하는 경우

1. 사각형을 원의 반지름을 더해서 확장 시킨다.
   : 위 그림의 빨간색 사각형
2. 확장된 사각형 영역에 원의 중점이 포함되는지를 검사한다.
   : 위의 영역에 위치한 원은 원의 반지름 만큼 사각형을 확장했을 때 겹쳐지는 원의 중심은 확장된 사각형에 반드시 포함된다.
3. 원의 반지름 만큼 확장된 사각형의 내부에 원의 중심이 포함된다면 사각형과 원이 서로 충돌하였다.

원이 사각형의 대각선 방향에 위치하는 경우

1. 원의 내부에 사각형의 꼭지점이 포함되는지를 판단하여 충돌 검사를 한다.
   : 위의 방식을 적용하면 충돌 되지 않은 원도 충돌 되었다고 판단하는 경우가 발생함
2. 사각형의 꼭지점이 원의 내부에 포함된다면 원과 사각형은 충돌 하였다.



충돌 조건

사각형의 꼭지점이 원의 내부에 있음

원의 내부에 사각형의 꼭지점이 있는지 검사

위의 원과 점의 충돌을 이용한다.

사각형 원 충돌 정리

1. 원의 반지름 만큼 확장된 사각형의 내부에 원의 중점에 있으면 2번으로 이동 아니면 **충돌 없음 판정**
2. 원의 x좌표가 사각형의 x 범위에 있거나 원의 y좌표가 사각형의 y 범위에 있으면 **충돌확정판정** 아니면 3번으로 이동
3. 원의 중점이 확장된 사각형의 대각선 범위에 있을 경우 해당 사각형 꼭지점이 원의 내부에 있으면 **충돌확정판정** 아니면 충돌 없음 판정

```
function checkHit(rr, cc) {
    var nResult = false, ar, fDistSqr;

    // 큰 장방형 체크
    if ((cc.x > rr.x0 - cc.r) && (cc.x < rr.x1  + cc.r) && (cc.y > rr.y0 - cc.r) && (cc.y < rr.y1 + cc.r)) {
        nResult = true;
        ar = cc.r;
        // 왼쪽 끝 체크
        if (cc.x < rr.x0) {
            if (cc.y < rr.y0) {          // 좌측상단 모서리 체크
                if (distance(rr.x0, rr.y0, cc.x, cc.y) >= ar * ar) {
                    nResult = false;
                }
            } else {
                if (cc.y > rr.y1) {          // 좌측하단 모서리 체크
                    if ((distance(rr.x0, rr.y1, cc.x, cc.y) >= ar * ar)) {
                        nResult = false;
                    }
                }
            }
        } else {
            // 오른쪽 끝 체크
            if (cc.x > rr.x1) {
                if (cc.y < rr.y0) {          // 우측 상단 모서리 체크
                    if (distance(rr.x1, rr.y0, cc.x, cc.y) >= ar * ar) {
                        nResult = false;
                    }
                } else {
                    if (cc.y > rr.y1) {     // 좌측 하단 모서리 체크
                        if (distance(rr.x1, rr.y1, cc.x, cc.y) >= ar * ar) {
                            nResult = false;
                        }
                    }
                }
            }
        }
    }
    return nResult;
}
```

```
function stateInfo() {
  var rr, cc, x0, x1, y0, y1;

  x0 = tb.x;
  x1 = tb.x + tb.w;
  y0 = tb.y;
  y1 = tb.y + tb.h;

  rr = {x0: x0, x1: x1, y0: y0, y1: y1};
  cc = {x: px, y: py, r: pr};

  if (checkHit(rr, cc)) {
    ctx.fillText("상태: 충돌", 50, 50);
  } else {
    ctx.fillText("상태: 안전", 50, 50);
  }
}
```

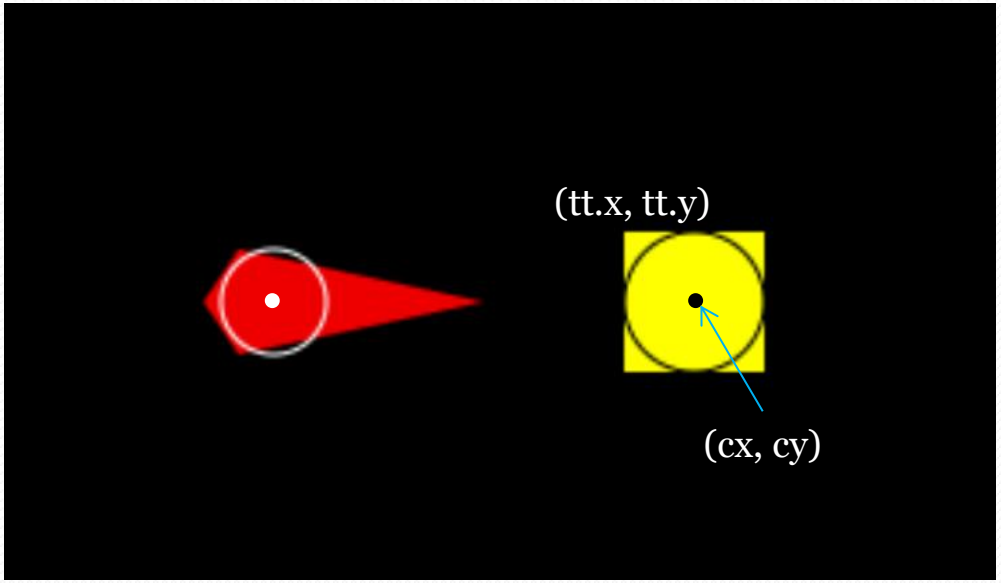
2. 5주 실습 기본 파일

oo_비행선.js

```
1.  'use strict';
2.  var vcanvas, ctx;
3.  var r_right, r_left, r_up, r_down;
4.  var stype = 0;
5.  var sx = 200, sy = 200, vel = 1;

6.  var tt = {x: 500, y: 180, wh: 40, c: "yellow"};

7.  function drawTarget() {
8.      ctx.fillStyle = tt.c;
9.      ctx.fillRect(tt.x, tt.y, tt.wh, tt.wh);
10. }
```

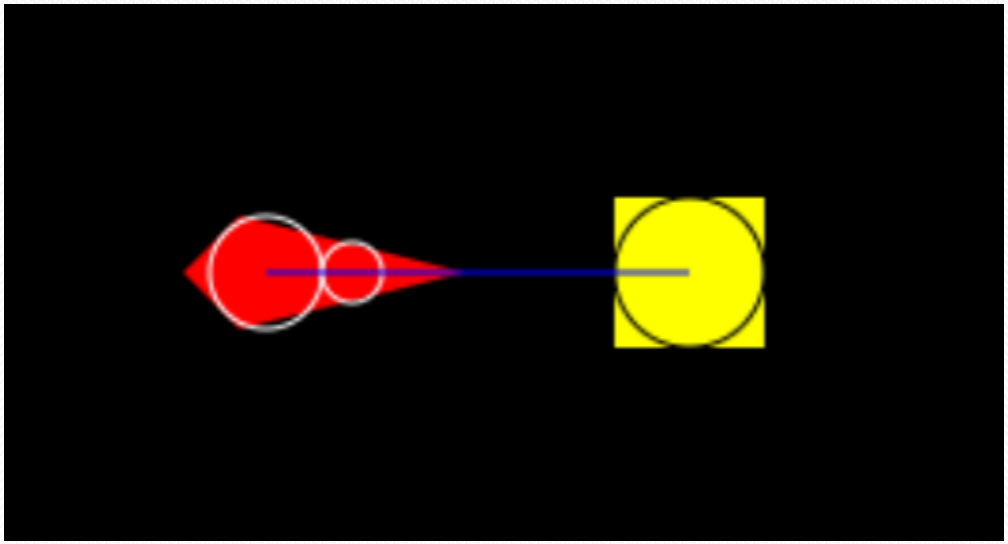
3. 충돌범위

3.1 충돌 범위 : 원 설정 (1)



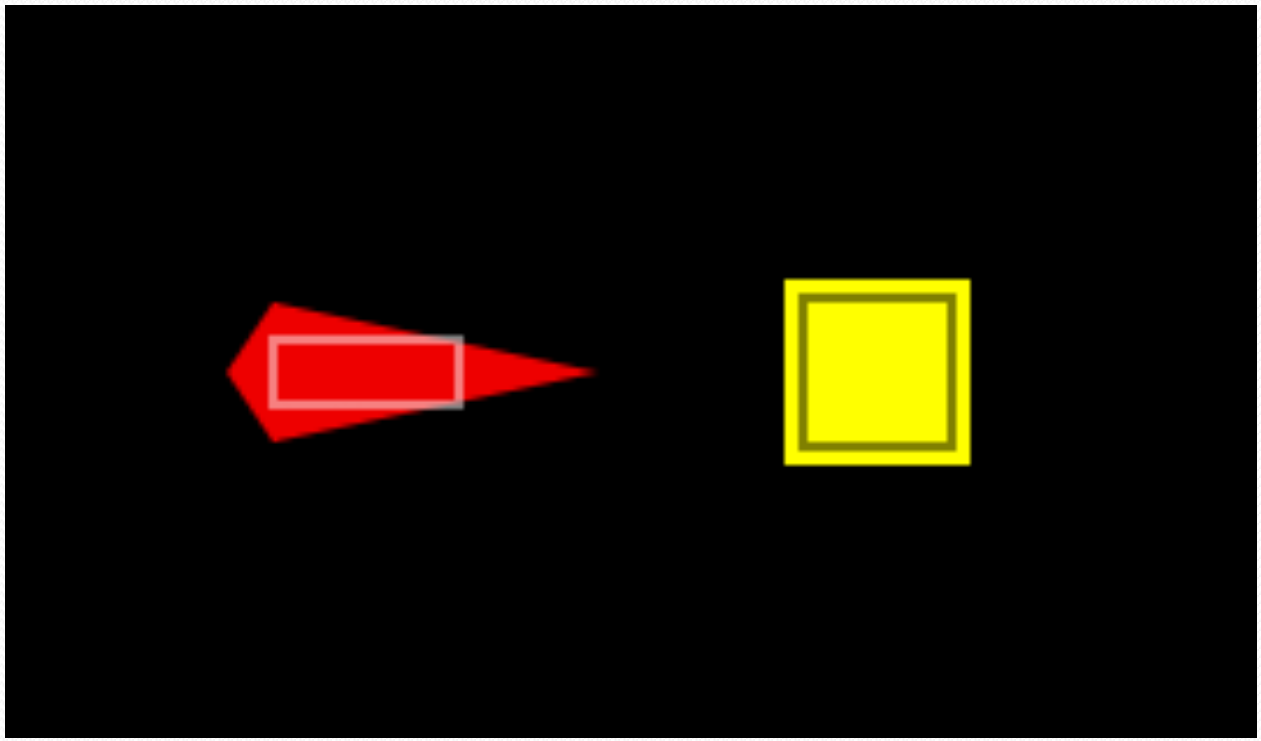
```
1.  function boundary() {
2.      var cx, cy, cr;
3.
4.      cx = tt.x + tt.wh / 2;
5.      cy = tt.y + tt.wh / 2;
6.      cr = tt.wh / 2;
7.
8.      ctx.strokeStyle = "black";
9.      ctx.beginPath();
10.     ctx.arc(cx, cy, cr, 0, 2 * Math.PI);
11.     ctx.stroke();

12.     ctx.strokeStyle = "white";
13.     ctx.beginPath();
14.     ctx.arc(sx + 7, sy, 15, 0, 2 * Math.PI);
15.     ctx.stroke();
16. }
```

3.2 충돌 범위 : 원 설정 (2)



```
1.   function boundary() {
2.       var cx, cy, cr, sx2, sr1 = 15, sr2 = 8;
3.
4.       ctx.strokeStyle = "white";
5.       ctx.beginPath();
6.       ctx.arc(sx + 7, sy, sr1, 0, 2 * Math.PI);
7.       ctx.stroke();
8.
9.       sx2 = sx + 7 + sr1 + sr2;
10.      ctx.beginPath();
11.      ctx.arc(sx2, sy, sr2, 0, 2 * Math.PI);
12.      ctx.stroke();
13.
14.      cx = tt.x + tt.wh / 2;
15.      cy = tt.y + tt.wh / 2;
16.      cr = tt.wh / 2;
17.
18.      ctx.strokeStyle = "black";
19.      ctx.beginPath();
20.      ctx.arc(cx, cy, cr, 0, 2 * Math.PI);
21.      ctx.stroke();
22.  }
```

3.3 충돌 범위 : 박스 설정



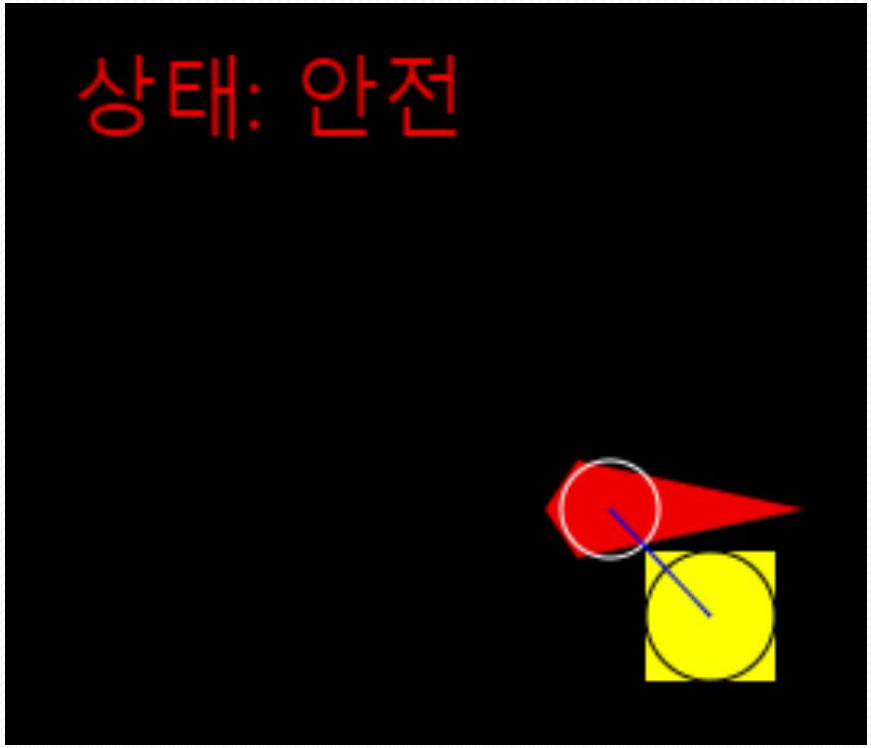
```
var tt = {x: 500, y: 180, wh: 40, c: "yellow"};
```

---------- ◇ ----------

```
1.  function boundaryRect() {
2.      ctx.strokeStyle = "white";
3.      ctx.strokeRect(sx - 7, sy - 7, 40, 14);
4.
5.      ctx.strokeStyle = "black";
6.      ctx.strokeRect(tt.x + 5, tt.y + 5, 30, 30);
7.  }
```

4. 충돌체크

4.1 충돌 체크 : 원 사용 (1)

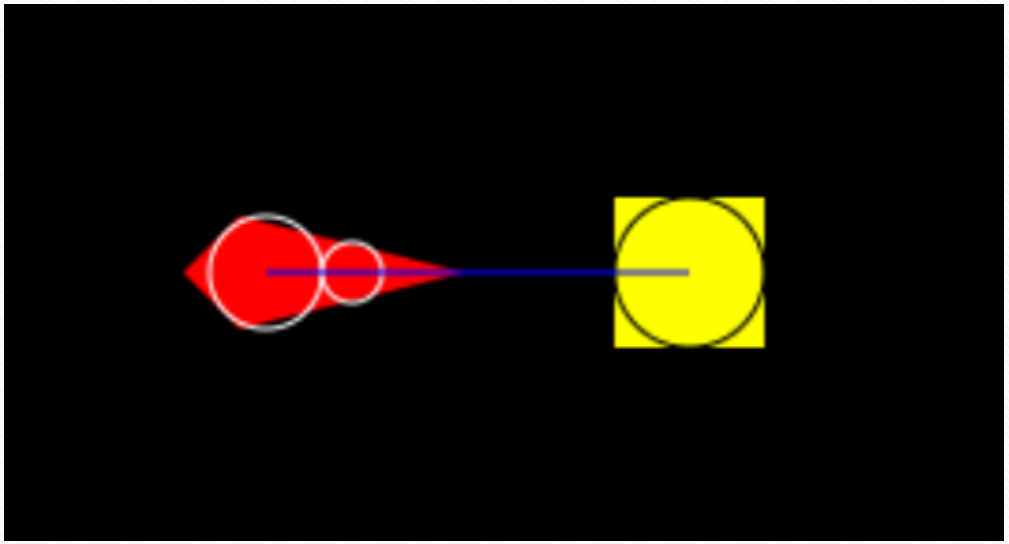


```
1.function hitShipCircle (x1, y1, x2, y2, dis) {
2.     var distance = Math.sqrt((x1-x2) * (x1-x2) + (y1-y2) * (y1-y2));
3.     if (distance < dis) {
4.         return true;
5.     } else {
6.         return false;
7.     }
8.}
```

[함수 콜 방법]

```
hitShipCircle(sx + 7, sy, tt.x + tt.wh / 2, tt.y + tt.wh / 2, sr + tr)
```

4.2 충돌 체크 : 원 사용 (2)

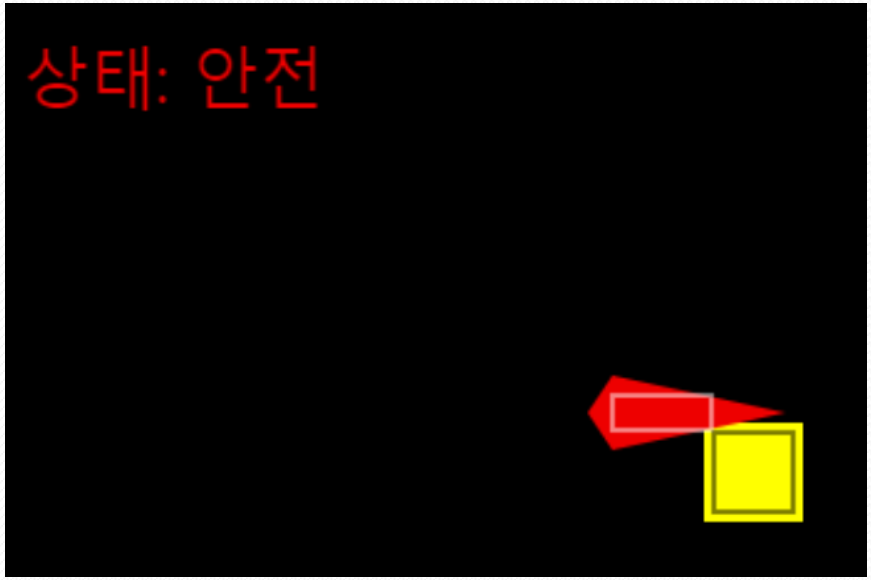


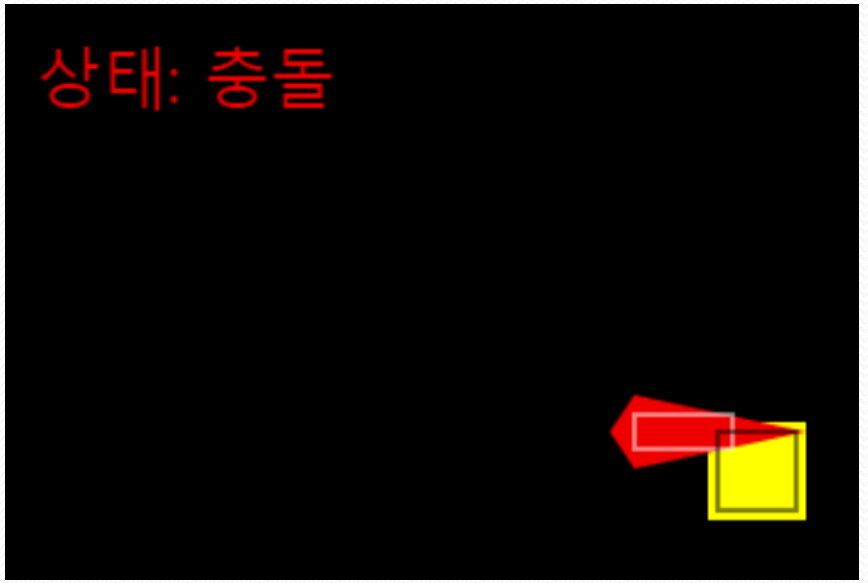
```
1. function hitship_circle(x1, y1, x2, y2, sr, tr) {
2.     var x1s, sr2 = 8, distance1, distance2;
3.
4.     x1s = x1 + sr + sr2;
5.     distance1 = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
6.     distance2 = Math.sqrt((x1s- x2) * (x1s - x2) + (y1 - y2) * (y1 - y2));
7.
8.     if (distance1 < (sr + tr) || distance2 < (sr2 + tr)) {
9.         return true;
10.    } else {
11.        return false;
12.    }
13.}
```

[함수 콜 방법]

```
hitship_circle(sx + 7, sy, tt.x + tt.wh / 2, tt.y + tt.wh / 2, sr, tr)
```

충돌 체크 : 박스 사용





```
1.  function hitship_rect() {
2.      var s, t;
3.
4.      s = {x1: sx - 10, y1: sy - 7, x2: sx + 30, y2: sy + 7};
5.      t = {x1: tt.x, y1: tt.y, x2: tt.x + tt.wh, y2: tt.y + tt.wh};
6.      return (s.x2 > t.x1 && s.x1 < t.x2) && (s.y2 > t.y1 && s.y1 < t.y2);
7.  }
8.
9.  function stateInfo() {
10.     var sr = 15, tr = tt.wh / 2;
11.     ctx.font = "20pt arial bold";
12.     if (hitship_rect()) {
13.         ctx.fillText("상태: 충돌", 50, 50);
14.     } else {
15.         ctx.fillText("상태: 안전", 50, 50);
16.     }
17. }
```

인클래스 실습

충돌 적용하여 Argon 완성 (Ship 충돌 & Rocket 충돌)