[blockBreak_game.html]

2020301052 문희준

```
<!DOCTYPE html>  // HTML 태그는 아니지만, 선언된 페이지의 HTML 버전이 무엇인지를 웹 브라우저에 알려주는 역할을 하는 선언코
<html>           // DOCTYPE 선언을 제외한 모든 다른 HTML 요소를 포함하기 위한 컨테이너, 브라우저에게 해당 문서가 HTML문서임을 알려줌
<head>           // 이 html 파일의 metadata의 집합. 즉 다른 데이터들을 설명하기 위한 태그들과 스타일 정의를 포함
  <style>        // 이 html 파일의 CSS를 명시, 스타일 정보를 정의할 때 사용
    canvas {     // 스크립트를 이용하여 그래픽 콘텐츠를 그릴때 사용
      background: rgb(243, 245, 117); // 배경 색상을 rgb (243, 245, 117) 값으로 지정
    }
  </style>
  <script>  // 스크립트 코드를 요소 내부에 직접 명시, src 속성을 사용하여 외부 스크립트 파일을 참조
    window.onload = function() { // 웹브라우저를 담당하는 window라는 객체가 웹 문서를 불러올 때 실행되는 onload 함수(페이지가 로드된 후 코드실행)
      let canvas = document.getElementById("myCanvas"); // myCanvas라는 id를 가진 요소를 찾아 canvas 변수에 할당.
      let context = canvas.getContext("2d"); // canvas 변수의 2d 그래픽 콘텐츠를 가져와 context 변수에 할당
      let brickRows = 3; // 벽돌의 행 개수를 정할 brickRows 변수에 3 할당
      let brickColumns = 5; // 벽돌의 열 개수를 정할 brickColumns 변수에 5 할당   ] 벽돌 3×5=15개.
      let brickWidth = 75; // 벽돌의 가로 길이를 정할 brickWidth 변수에 75 할당
      let brickHeight = 20; // 벽돌의 세로 길이를 정할 brickHeight 변수에 20 할당   ] 가로 75px, 세로 20px
      let brickPadding = 10; // 벽돌 사이 간의 간격을 정할 brickPadding 변수에 10 할당
      let brickStartX = 30; // 벽돌을 생성할 첫 지점의 X 좌표를 정할 brickStartX 변수에 30 할당
      let brickStartY = 30; // 벽돌을 생성할 첫 지점의 Y 좌표를 정할 brickStartY 변수에 30 할당   ] 벽돌을 30, 30 부터 그림
      let bricks = new Array(brickRows); // brickRows 개수만큼의 행을 가진 2차원 배열을 선언하여 bricks 변수에 할당
      let ball = {  // 공의 상태를 정의할 객체 생성.
        x: canvas.width/2,  // 공의 시작 좌표를 출력할 캔버스 가로 값의 절반으로 설정
        y: canvas.height-30, // 공의 시작 좌표를 캔버스의 맨에서 30에 위치로 설정   ] 공의 시작좌표 : 가로의 중앙, 세로의 맨
        dx: 2,  // 공이 움직일 때의 x좌표가 2px 만큼 움직이게 설정
        dy: -2, // 공이 움직일 때의 y좌표가 2px 만큼 움직이게 설정
        radius: 10,  // 공의 반지름 길이를 10px로 설정
        draw: function() {  // 공을 그리기위한 draw 메소드
          context.beginPath();  // 콘텍스트 변수에 그림을 그리기 위해 beginPath 메소드 사용
          context.arc(this.x, this.y, this.radius, 0, Math.PI * 2); // 콘텍스트 변수에 (x,y) 위치에 중심을 두면서, 반지름 radius를 가지고, 0에서 파이변수 Math.PI x2로 향하는 다지형을 그리기
          context.fillStyle = "#000000"; // 공을 "#000000"(검정)으로
          context.fill(); // 검정으로 공 내부를 채우기
          context.closePath(); // 공 그리기 종료
        }
      };
      let paddle = {  // 공을 받아칠 패들의 상태를 정의할 객체 생성
        height: 10, // 패들의 높이를 10px로 설정
        width: 90, // 패들의 가로 길이를 90px로 설정
        x: 300,  // 패들의 x 시작 좌표를 300으로 설정
        draw: function() {  // 공을 그리기위한 draw 메소드
          context.beginPath();  // 콘텍스트 변수에 그림을 그리기 위해 beginPath 메소드 사용
          context.rect(this.x, canvas.height - this.height, this.width, this.height); // 콘텍스트 변수에 (x, canvas의 높이 - 높이) 위치에 가로 width, 세로 height 크기의 직사각형 그리기
          context.fillStyle = "blue"; // 직사각형을 파란색으로
```

```
        context.fill();   // 파랑으로 직사각형 내부를 채우기
        context.closePath(); // 패스 그리기 종료
    }
}

class Brick {  // 벽돌의 상태를 정의할 클래스 객체 선언
    constructor(x, y, status) { // 생성자를 통하여 x, y, status의 값을 받아옴
        this.x = x;          // 벽돌의 x값 속성 설정
        this.y = y;          // 벽돌의 y값 속성 설정
        this.status = status; // 벽돌의 유무 속성 설정
    }
    draw() {      // 벽돌을 그리기위한 draw 메소드
        if (this.status == 1) {   // 만약 벽돌이 존재해야하는 위치인 경우
            context.beginPath(); // 컨텍스트 변수에 그림을 그리기 위해 beginPath 메소드 사용
            context.rect(this.x, this.y, brickWidth, brickHeight); // 콘텍스트 변수에 (x, y) 위치부터
            context.fillStyle = "brown"; // 직사각형을 갈색으로                  가로: brickWidth, 세로: brickHeight의
            context.fill(); // 갈색으로 직사각형 내부를 채우기                  직사각형 그리기
        }  context.closePath(); // 벽돌 그리기 종료
    }
    check() {     // 공과 벽돌이 겹쳤는지 확인하는 check 메소드
        if (this.status == 1) {  // 만약 벽돌이 존재하는 위치인 경우
            if (ball.x > this.x && ball.x < this.x + brickWidth  // 공의 (x,y)좌표가 벽돌의 (x,y)좌표보다
                && ball.y > this.y && ball.y < this.y + brickHeight) { 크고, (x+벽돌의 가로길이, y+벽돌의
                                                                    세로길이)
                ball.dy = -ball.dy; // 공의 Y방향을 음수로 바꿈 (공을 반대방향으로)   보다 작으면 (벽돌의 경계와 부딪히면)
                this.status = 0; // 벽돌이 부서졌으므로 벽돌이 없는 상태(0)으로 초기화
            }
        }
    }
}
function init() {  // 게임을 시작하기 전 벽돌을 초기화하는 init 메소드
    document.addEventListener("mousemove", mouseHandler, false); //addEventListener 방법을 사용하여 mouseHandler
                                                                  이벤트 핸들러를 등록
    for (let r = 0; r < brickRows; r++) { // 벽돌 생성의 행 개수만큼 반복
        bricks[r] = new Array(brickColumns); // bricks 배열 변수에 벽돌의 열 만큼 배열 생성.
        for (let c = 0; c < brickColumns; c++) { // 벽돌 생성의 열 개수만큼 반복
            let brickX = (c * (brickWidth + brickPadding)) + brickStartX; // 열 번호 x (벽돌의 가로 + 벽돌 사이 거리)
            let brickY = (r * (brickHeight + brickPadding)) + brickStartY; // 행 번호 x (벽돌의 세로 + 벽돌 사이 거리)
                                                                  + 벽돌 왼쪽의 첫 지점을 벽돌의 X위치로 설정
                                                                  + 벽돌 상단의 첫 지점을 벽돌의 Y위치로 설정
            bricks[r][c] = new Brick(brickX, brickY, 1);
        } // bricks 이차원 배열에 새로운 벽돌의 (x,Y) 좌표 저장.
    }
}
```

```
function mouseHandler(e){ // 마우스를 사용할 때 마우스 이벤트를 처리할 mouseHandler 메소드
    let x = e.clientX - canvas.offsetLeft; // e.clientX (사용자가 마우스를 움직일 때의 X 좌표)에서
                                           캔버스의 x값 좌표의 여백값을 뺀 값을 x에 할당
    if (x > 0 && x < canvas.width) // x값이 0보다 크고, 캔버스의 가로 길이보다 작은 경우 (마우스가 캔버스 안에 있을경우)
        paddle.x = x - paddle.width / 2; // 패들의 x좌표에 x값-패들의 길이/2을 할당
}                                        (사용자의 마우스를 움직일때 마우스 커서가 패들의 중앙에오게 설정)
function gameLoop() { //게임 실행에 필요한 모든 작업을 모아놓은 gameLoop 메소드.
    context.clearRect(0, 0, canvas.width, canvas.height); // 콘텍스트 변수에 (0,0)부터
                                                          끝: 캔버스의 가로길이, 세로: 캔버스의 세로길이 의
    ball.draw(); // 공 그리기 메소드 호출                   지워진 직사각형 영역을 그리기
    paddle.draw(); // 패들 그리기 메소드 호출
    for(let r = 0; r < brickRows; r++) { //
        for(let c = 0; c< brickColumns; c++){ //  ] 벽돌의 행,열별 만큼 이중 포문을 이용하여 반복
            bricks[r][c].check(); // bricks 이차원 배열에 check 메소드 호출 (공이 벽돌과 부딪히는지 확인하기 위해서)
            bricks[r][c].draw(); // bricks 이차원 배열에 draw 메소드 호출 (벽돌의 위치를 그리기 위해)
        }
    }
    if (ball.x + ball.dx > canvas.width - ball.radius // (공의 x좌표 + 공의 x좌표변화량)가 (캔버스 길이 - 공의 반지름길이)보다
        || ball.x + ball.dx < ball.radius)            크거나, (공의 x좌표 + 공의 x좌표 변화량)가 공의 반지름길이보다
        ball.dx = -ball.dx; // 공의 x좌표 변화량을 음수로 바꿈 (공의 좌우방향을 반대로)  작을경우 → 공이 캔버스의 가로길이보다 넘어갈 경우
    if(ball.y + ball.dy < ball.radius) //(공의 y좌표 + 공의 y좌표 변화량)이 공의 반지름보다 작은 경우
        ball.dy = -ball.dy; // 공의 y좌표 변화량을 음수로 바꿈 (공의 상하방향을 반대로)
    else if (ball.y + ball.dy > canvas.height - ball.radius) { //(공의 y좌표 + 공의 y좌표 변화량)이(캔버스의 높이 - 공의반지름)
        if (ball.x > paddle.x && ball.x < paddle.x + paddle.width) //공의 x좌표가 패들의 x좌표보다 크거나.   보다 클경우 → 공이 캔버스 밖으로 넘어간 경우
            ball.dy = -ball.dy; // 공의 y좌표 변화량을 음수로 바꿈   공의 x좌표가(패들의 x좌표+패들의 길이)보다
        else                                                        작을 경우 → 공이 패들에 튀길때 경우
            clearInterval(interval); //공의 위치를 계속 변화 시킴.
    }
    ball.x += ball.dx; // 공의 x좌표를 공의 x좌표 변화량만큼 증가, 저장
    ball.y += ball.dy; // 공의 y좌표를 공의 y좌표변화량만큼 증가, 저장
}
init(); // 게임 시작 전 벽돌 생성을 초기화하는 메소드 실행
setInterval(gameLoop, 10); // gameLoop 메소드를 10ms 마다 반복하여 실행
}
</script>
</head>
<body> // 이 html 파일의 텍스트, 하이퍼링크, 이미지, 리스트 등과 같은 콘텐츠를 포함하는 영역을 표시하는 태그
    <canvas id="myCanvas" width="600" height="400"></canvas> //myCanvas라는 id를 가지는 캔버스의 가로 길이를 600,
</body>                                                      세로길이를 400으로 설정
</html>
```