**웹 프로그래밍**

#3 벽돌 깨기 게임

|  |
| --- |
| 컴퓨터공학과 |
| 20184102 |
| 신주용 |

앞서 수업시간에 배운 벽돌깨기 게임을 업그레이드 하여

새로운 게임을 제작하여 보았습니다.

분석에 앞서 간략한 설명을 하자면

좌우측 1열씩 블록을 없애고, 해당 부분으로 공이 들어가서 더 잘 깨어질 수 있게 빈틈을 두었습니다.

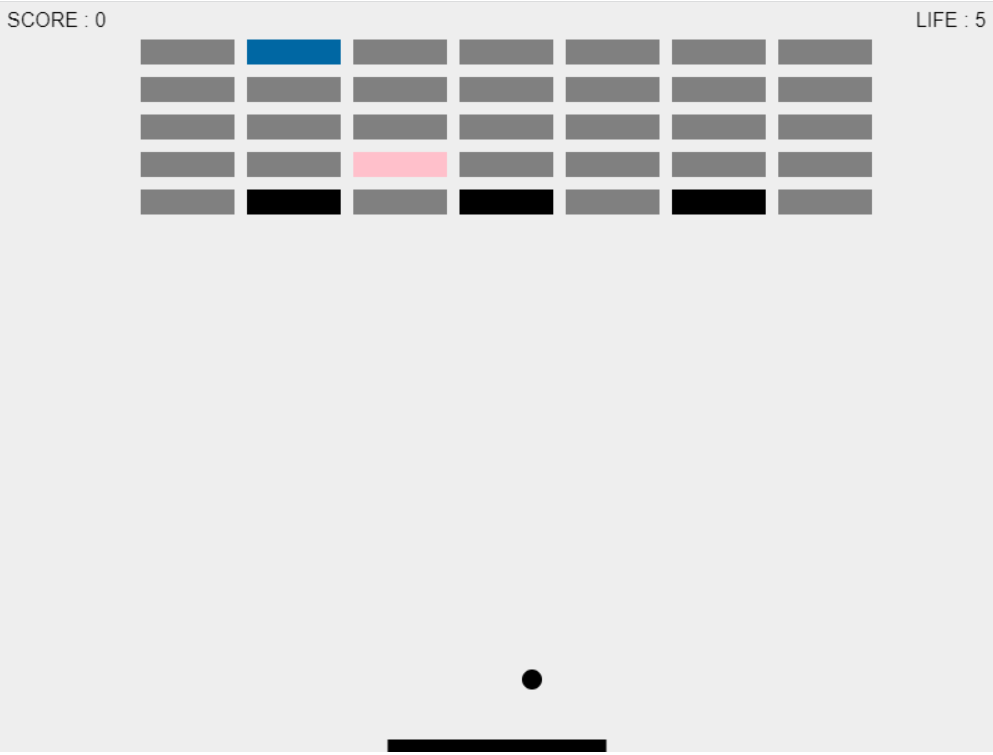
생명을 추가해주는 life블록과 새 공을 추가해주는 블록을 각각 만들어 주어 게임의 단조로움을 조금 해결하였습니다.

맨 아래행의 홀수번째 블록들은 무적블록(깨어지지 않는)으로 설정하여 게임의 변수창출을 도와줍니다.

분석은 새로 추가한 기능들 위주로 하도록 하겠습니다.

작성한 코드와 실행 영상은 파일로 따로 첨부하겠습니다.

**게임 화면**



눈에띄는 큰 특징은 좌우측 블록들이 비어있는 것과

파란, 핑크색, 검은색 블록이 추가된 것입니다.

이 유색 블록들은 각각의 새로운 기능(새로운 공 추가, 라이프 추가, 무적)을 합니다.

공이 시작과 동시에 우측 상단으로 이동하고, 이동하며 만난 블록들(유색 제외)은 3회에 걸쳐 깨어집니다.

한 번 접촉할 때마다 스코어가 1씩 올라가고, 모든 블록이 깨지면 혹은 모든 생명이 없어지면 게임이 종료됩니다.

기본 코드에서는 블록의 상단에 공이 닿으면 공이 느려지다가 멈추거나 공이 블록에 끼여 움직이지 않는 등의 이슈가 있어 해당 부분을 수정하여 다시 작성해주었습니다.

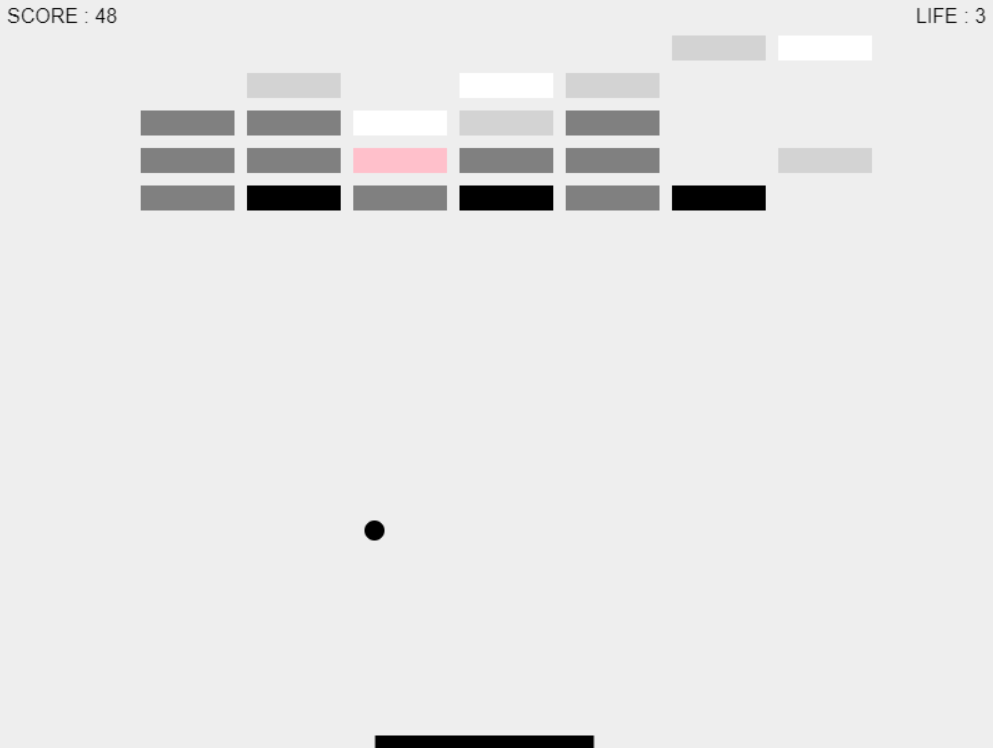
**새로운 공의 생성**



위와 같이 새로운 공은 핑크색으로 색상을 구분해주었습니다.

drawball함수와 기타 공의 움직임에 관여하는 함수들을 동일하게 작성후 선언해주어, 파란색 블록에 공이 닿게되면 반복 호출을 통해 완전히 같은 기능을 합니다.

**블록 레벨(색상변경)**



위의 사진과 같이 블록의 레벨을 3(진한 회색, 회색, 하얀색)으로 구분하여 블록이 한 번에 깨지지 않고, color에 따라 접촉해야할 횟수를 구분해주었습니다.

색상이 바뀌는 부분은 bricks[]에 color라는 새로운 변수를 지정해주어, 접촉시에 color변수가 줄어들고, color에 따라 블록의 색상이 변경되는 방식으로 색상을 변경해주었습니다.

**무적 블록**



위 사진의 검은색 블록은 무적블록으로, 공과 접촉해도 절대 사라지지 않습니다.

기존의 코드와 다르게 모든 블록들의 status를 직접적으로 변경해주지 않고, 앞서 설정한 변수인color가 0이 되면 블록의 status를 0으로 설정해줍니다.

하지만 무적블록들은 color가 음수로 미리 지정되어 있기 때문에 –-를 무한히 반복해도 절대 0이 될 수 없습니다. 따라서 해당 블록들은 절대 깨어지지 않습니다.

앞선 설명에서 큰 기능들에 대한 간략한 분석을 마쳤습니다.

블록이 사라지는 부분 등의 간단한 동작 원리와 큰 기능들의 알고리즘은 아래 코드의 주석에서 코드와 함께 한눈에 보기 쉽게 설명하도록 하겠습니다.

**전체 코드**

var canvas = document.getElementById("myCanvas"); // HTML에서 지정해준 게임판(myCanvas)을 JS에서 사용하기 위해 가져옵니다.

var ctx = canvas.getContext("2d"); // 기본 제공되는 그리기 함수를 사용하기 위해 canvas에 getContext를 사용해줍니다.

var ballRadius = 8; // 공의 반지름을 8로 설정해줍니다.

var x = canvas.width/2; // x축 위치를 게임판 너비의 반으로 설정해줍니다.

var y = canvas.height-30; // y축 위치를 높이-30으로 설정해줍니다.

var dx = 4; // dx(공이 움직일 초기 속도)를 4로, y는 -4로 설정해줍니다.

var dy = -4;

var paddleHeight = 10; // paddle의 높이와 너비를 설정해줍니다.

var paddleWidth = 175;

var paddleX = (canvas.width-paddleWidth)/2; // paddleX는 게임판너비 - 패들너비의 반입니다.

var rightPressed = false; // right, leftpressed의 초기값을 False로 설정해줍니다.

var leftPressed = false;

var brickRowCount = 9; // 블럭 배열의 가로세로 크기를 설정해줍니다.

var brickColumnCount = 5;

var brickWidth = 75; // 블럭의 가로세로 길이, 패딩, 탑 레프트 오프셋을 설정해줍니다.

var brickHeight = 20;

var brickPadding = 10;

var brickOffsetTop = 30;

var brickOffsetLeft = 30;

var counter = 0; // 게임이 종료되기 위해 필요한 counter변수를 설정해줍니다. 이는 이후에 게임 종료 판별식에 사용됩니다.

var score = 0; // 스코어를 표기하기 위한 score변수를 선언해줍니다.

var lives = 5; // 생명 초기값을 5로 설정해줍니다.

var newx = canvas.width/2;; // 새로운 공에 사용하기 위한 변수들입니다. 기능과 용도는 앞의 변수와 똑같습니다.

var newy = canvas.height-30;

var drawcou = 0;

var newdx = 4;

var newdy = -4;

var newpaddleX = (canvas.width-paddleWidth)/2;

var bricks = []; // Bricks를 생성해주는 부분입니다.

for(var c=0; c<brickColumnCount; c++) { // 위의 변수파트에서 선언해준 brickColumnCount와

    bricks[c] = [];

    for(var r=0; r<brickRowCount; r++) { // brickRowCount만큼의 2차원 배열을 생성하여 Brick을 생성합니다.(실체는 없음)

        bricks[c][r] = { x: 0, y: 0, status: 1, color: 3};

        if(r == 0 || r == 8) {

            bricks[c][r].status = 0;

        }

    }

}

document.addEventListener("keydown", keyDownHandler, false); // 사용자가 입력한 key와 마우스 입력을 모두 이용하여

document.addEventListener("keyup", keyUpHandler, false); // 패들을 조종하는데 사용합니다.

document.addEventListener("mousemove", mouseMoveHandler, false);

function mouseMoveHandler(e) { // 사용자가 입력한 마우스, 키보드 입력을 을 사용하기 위해 정의한 함수들입니다.

    var relativeX = e.clientX - canvas.offsetLeft;

    if(relativeX > 0 && relativeX < canvas.width) {

        paddleX = relativeX - paddleWidth/2;

    }

}

function keyDownHandler(e) { //

    if(e.keyCode == 39) { // 우측 키를 누르면

        rightPressed = true; // rightPressed가 true로 설정됩니다.

    }

    else if(e.keyCode == 37) {

        leftPressed = true;

    }

}

function keyUpHandler(e) {

    if(e.keyCode == 39) {

        rightPressed = false;

    }

    else if(e.keyCode == 37) {

        leftPressed = false;

    }

}

function collisionDetection() { // 충격 감지(접촉 감지) 함수입니다.

    for(var c=0; c<brickColumnCount; c++) {

        for(var r=0; r<brickRowCount; r++) { // 2차원 배열을 for문을 통해 모두 훑어

            var b = bricks[c][r]; // b변수에 해당 브릭을 지정한다음

            if(b.status == 1) { // b의 status, 존재여부가 1이라면

                if(x > b.x && x < b.x+brickWidth && y > b.y && y < b.y+brickHeight) { // x,y 모두가 해당 브릭의 x,y보다 크고, 해당 브릭의x(y)+브릭크기보다는 작으면서,

                    dy = -dy; // y축을 뒤집어줍니다(이동 방향)

                    if(c == 0&&r == 2) { // 0번째 줄의 2번째(1,3)블럭을 건드리면

                        newdrawBall(); // 새 공을 생성하고

                        drawcou = 1; // drawcou가 1로 설정됩니다. 이는 공을 반복호출하여 그려주기 위한 변수입니다.

                    }

                    if(r == 3 && c == 3) { // 3,3에 있는 브릭을 건드리면

                        lives += 10; // 생명이 10 늘어납니다.

                    }

                    if(dy > 0) { // dy가 0보다 높을때, 양수 방향으로 움직이고 있을 때

                        dy += 1; // 속도를 빠르게 해줍니다.

                        dx += 1 ;

                        if (dy > 10 || dx > 10) { // 하지만 속도가 10 이상이라면

                            dy = 10; // 10으로 고정합니다.

                            dx = 10;

                        }

                    }

                    else if(dy < 0) { // 반대의 경우도 마찬가지로 속도가 빨라지지만, 10이상이라면 고정됩니다.

                        dy -= 1;

                        dx -= 1;

                        if (dy < -10 || dx < -10) {

                            dy = -10;

                            dx = -10;

                        }

                    }

                    b.color--; // 블럭의 색상을 변경하기 위한 변수 b의 컬러를 1만큼 줄여줍니다.

                    if(b.color == 0){ // 만약 b의 컬러가 0이 된다면,

                        b.status = 0; // b의 상태는 0이 되고, 존재하지 않게 됩니다(깨집니다.)

                        counter++; // 이후 카운터를 1만큼 올려줍니다.

                    }

                    if (c == 4 && (r !=1) && (r !=3) && (r !=5) && (r !=7)) { // 무적벽돌에 닿는다면

                        score--; // score가 1만큼 줄어듭니다.(이후 나올 score++에 대응하기 위함)

                    }

                    score++; // 블럭에 한 번 닿을 때마다 score가 1만큼 올라갑니다.(깨어지지 않고 색상만 변경되어도)

                    if(counter == (brickRowCount-2) \* brickColumnCount - 3) { // counter가 7\*5 - 3만큼 올라간다면

                        alert("Victory !"); // 게임에 승리하게 되고

                        document.location.reload(); // 알림을 종료하면, 웹을 다시 불러와 재시작합니다.

                    }

                }

            }

        }

    }

}

// 새로운 공에서도 동일한 기능을 할 수 있도록 선언해준 함수

function newcollisionDetection() { // 앞서 새로 선언해준 변수들을 사용하여 동일한 기능을 하는 collisionDetection함수를 새로 선언해줍니다.

    for(var c=0; c<brickColumnCount; c++) { // 기능이 완전히 같으므로 따로 주석은 달지 않겠습니다.

        for(var r=0; r<brickRowCount; r++) {

            var b = bricks[c][r];

            if(b.status == 1) {

                if(newx > b.x && newx < b.x+brickWidth && newy > b.y && newy < b.y+brickHeight) {

                    newdy = -newdy;

                    if(c == 0 && r == 2) {

                        newdrawBall();

                        ++drawcou;

                    }

                    if(newdy > 0) {

                        newdy += 1;

                        newdx += 1 ;

                        if (newdy > 10 || newdx > 10) {

                            newdy = 10;

                            newdx = 10;

                        }

                    }

                    else if(newdy < 0) {

                        newdy -= 1;

                        newdx -= 1;

                        if (newdy < -10 || newdx < -10) {

                            newdy = -10;

                            newdx = -10;

                        }

                    }

                    b.color--;

                    if(b.color == 0){

                        b.status = 0;

                        counter++;

                    }

                    if (c == 4 && (r !=1) && (r !=3) && (r !=5) && (r !=7)) {

                        score--;

                    }

                    score++;

                    if(counter == (brickRowCount-2) \* brickColumnCount)  {

                        alert("Victory !");

                        document.location.reload();

                    }

                }

            }

        }

    }

}

function drawScore() { // drawScore함수를 선언합니다. Score를 8,20위치에 표시해줍니다.

    ctx.font = "16px Arial";

    ctx.fillStyle = "Black";

    ctx.fillText("SCORE : "+score, 8, 20);

}

function drawLives() {// drawLives함수를 선언합니다. Lives를 게임판-65, 20위치에 표시해줍니다.

    ctx.font = "16px Arial";

    ctx.fillStyle = "Black";

    ctx.fillText("LIFE : "+lives, canvas.width-65, 20);

}

function drawBall() { // drawBall함수를 선언합니다. arc를 사용하여 x,y위치에 BallRadius만큼의 반지름을 가진 공을 그림을 생성합니다.

    ctx.beginPath();

    ctx.arc(x, y, ballRadius, 0, Math.PI\*2);

    ctx.fillStyle = "Black";

    ctx.fill();

    ctx.closePath();

}

function newdrawBall(){ // 위의 drawBall함수와 동일한 기능을 하는 newdrawBall함수를 선언합니다. 새로운 공은 Pink색상인것을 제외하고

    ctx.beginPath(); // 모든 부분이 동일합니다.

    ctx.arc(newx, newy, ballRadius, 0, Math.PI\*2);

    ctx.fillStyle = "Pink";

    ctx.fill();

    ctx.closePath()

}

function drawPaddle() { // drawPaddle함수를 선언합니다. 아래에서 공을 튕겨주는 역할을 할 부분입니다.

    ctx.beginPath();

    ctx.rect(paddleX, canvas.height-paddleHeight, paddleWidth, paddleHeight);

    ctx.fillStyle = "Black";

    ctx.fill();

    ctx.closePath();

}

function drawBricks() { // 브릭을 화면에 표시해줄 drawBricks함수입니다.

    for(var c=0; c<brickColumnCount; c++) {

        for(var r=0; r<brickRowCount; r++) { // c, r크기만큼의 배열을 생성하여

            if(bricks[c][r].status == 1) { // status가 1일때(표시 판별기),

                var brickX = (r\*(brickWidth+brickPadding))+brickOffsetLeft; // 브릭의 가로순서(세로순서), (가로(높이) \* (브릭가로 +패딩)) + 오프셋크기인

                var brickY = (c\*(brickHeight+brickPadding))+brickOffsetTop; // brickX와 brickY의 크기를 선언해주고

                bricks[c][r].x = brickX; // 해당 부분을 bricks의 x좌표, y좌표로 설정해줍니다.

                bricks[c][r].y = brickY;

                ctx.beginPath();

                ctx.rect(brickX, brickY, brickWidth, brickHeight); // 반복적으로 rect를 사용해 앞서 선언한 변수들로 위치를 지정해준 브릭을 생성하고

                if(r == 3 && c == 3) { // 3,3번째 위치에는

                    bricks[c][r].color = 1; // 핑크색 블럭을 생성합니다. 이는 이후에 특별한 기능을 가지게 됩니다.

                    ctx.fillStyle = "Pink";

                }

                else if (c == 0 && r == 2) { // 0,2번째 위치에는 파란색 블럭을 생성합니다. 이것도 특별한 기능을 가지게 됩니다.

                    ctx.fillStyle="#0067a3";

                    bricks[c][r].color = 1;

                }

                else {

                    ctx.fillStyle = "#808080"; // 블럭의 초기 색상은 진한 회색이고, 컬러는 3입니다. 충돌시에 color를 하나씩 줄여

                    if(bricks[c][r].color == 2) { // color의 상태에 따라 색상이 변하도록 만들었습니다.

                        ctx.fillStyle = "#d3d3d3";

                    }

                    else if(bricks[c][r].color == 1) {

                        ctx.fillStyle = "white";

                    }

                }

                if (c == 4 && (r !=1) && (r !=3) && (r !=5) && (r !=7)) { // 예외를 선언해주어 해당 블럭들은 검정 색상에 color를 음수로 설정해주고

                    ctx.fillStyle="Black"; // color가 깎여도 status가 변하지 않도록 해주었습니다.

                    bricks[c][r].color = -1;

                }

                ctx.fill();

                ctx.closePath();

            }

        }

    }

}

function draw() { // 이번 과제에서 가장 중요한 역할을 하는 draw입니다.

    ctx.clearRect(0, 0, canvas.width, canvas.height); // 모든 함수들을 한 번에 호출해줍니다.

    drawBricks();

    drawBall();

    drawPaddle();

    drawScore();

    drawLives();

    collisionDetection();

    if(x + dx > canvas.width-ballRadius || x + dx < ballRadius) { // 공이 x축상에서 오른쪽에 부딪히거나, 왼쪽에 부딪히면

        dx = -dx; // 부호를 반대로 설정해주어 튕겨지게 합니다.

    }

    if(y + dy < ballRadius) { // 위쪽 천장에 부딪히게 되면 y의 부호도 뒤집어집니다.

        dy = -dy;

    }

    else if(y + dy > canvas.height-ballRadius) { // 위쪽 천장이 아닌 아래쪽 천장에 부딪혔을 떄

        if(x >= paddleX && x < paddleX + paddleWidth) { // x가 paddle보다 같거나 크고, 패들X+패들가로가 x보다 크다면

            dy = -dy; // y축을 뒤집어줍니다.

        }

        else { // 그게 아니라면

            lives--; // 생명을 하나 깎습니다.

            if(!lives) { // 생명이 0이라면

                alert("Failed"); // Failed를 출력하고, 게임을 다시 실행합니다.

                document.location.reload();

            }

            else { // 생명이 남아있다면

                x = canvas.width/2; // x축과 y축의 위치를 초기화하고

                y = canvas.height-30;

                dx = 4; // 속도도 다시 설정해줍니다.

                dy = -4;

                paddleX = (canvas.width-paddleWidth)/2; // 패들의 위치도 초기화해줍니다.

            }

        }

    }

    if(rightPressed && paddleX < canvas.width-paddleWidth) { // 키보드 우방향을 눌렀고, 패들이 캔버스 아래 선상에 있을때(벽에 겹치지 않게)

        paddleX += 7;                                        // paddle의 위치를 가로 방향으로 7만큼 이동합니다.(7은 이동속도)

    }

    else if(leftPressed && paddleX > 0) {

        paddleX -= 7;

    }

    x += dx; // x와 y의 위치는 dx, dy만큼 더해줍니다.

    y += dy;

    requestAnimationFrame(draw); // 프레임이 더욱 부드럽게 나오게 하기 위한 함수입니다.

    if (drawcou != 0) { // 앞서 선언해준 drawcou(새로운 공이 나왔다면)

        requestAnimationFrame(newdraw); // newdraw도 함께 지정해줍니다.

    }

}

// 새로운 공의 draw를 위한 함수입니다.

function newdraw() { // 앞서 설명한 draw와 완전히 같은 함수이므로 따로 설명하지 않겠습니다.

    newdrawBall();

    newcollisionDetection();

    if(newx + newdx > canvas.width-ballRadius || newx + newdx < ballRadius) {

        newdx = -newdx;

    }

    if(newy + newdy < ballRadius) {

        newdy = -newdy;

    }

    else if(newy + newdy > canvas.height-ballRadius) {

        if(newx > paddleX && x < paddleX + paddleWidth) {

            newdy = -newdy;

        }

        else {

            lives--;

            drawcou--;

            newx = canvas.width/2;

            newy = canvas.height-30;

            newdx = 4;

            newdy = -4;

            paddleX = (canvas.width-paddleWidth)/2;

        }

    }

    if(rightPressed && newpaddleX < canvas.width-paddleWidth) {

        newpaddleX += 7;

    }

    else if(leftPressed && paddleX > 0) {

        newpaddleX -= 7;

    }

    newx += newdx;

    newy += newdy;

}

draw(); // draw를 호출해주어 모든 기능을 할 수 있게 합니다.

if (drawcou >= 0) { // 동일하게 drawcou가 0 이상이라면, newdraw를 실행합니다.

    newdraw();

}