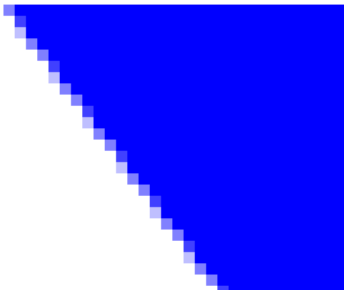
CLASE 2 IG: PONG

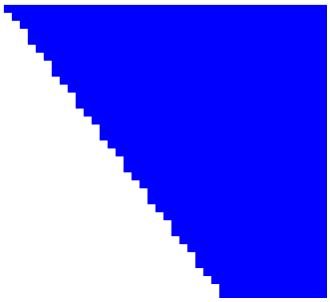
Antes de entrar con la práctica, podemos mostrar en vivo el tema del antialiasing. En WebGL por defecto siempre viene el antialiasing activado. Si queremos evitarlo, hay que indicarlo en el momento de crear el contexto del canvas.

```
gl = canvas.getContext('webgl2', { antialias: false });
```

Si pintamos por ejemplo un triángulo azul sobre fondo blanco, quizás no se note la diferencia, pero si capturamos pantalla y la abrimos con algún visor que no suavice la imagen al hacer zoom, sino que muestre los pixels tal cual (como el Paint de Windows), podremos apreciar la diferencia en los contornos.



con antialiasing



sin antialiasing

Práctica 2

En esta clase intentaremos implementar una versión del clásico Pong.

Partimos de la clase anterior, con una función drawRectangle para crear un rectángulo de color. Aunque por ahora quitaremos las llamadas a dicha función, porque la vamos a hacer diferente, y partimos del canvas vacío (pero animado)

Creamos un objeto bola

La idea es empezar un pong para que lo hagan ellos.

Pintando un único rectángulo que haga de pelota

Vamos a creamos un array inicial con un único rectángulo que haga de pelota.

```
var ball = {
          'x': -0.5, 'y': 0.3,
          'width': 0.1, 'height': 0.1,
          'color':[0,0,1,1]
        };
```

Y reconvertimos nuestra función drawRectangle para que acepte un objeto de este tipo:

```
function drawRectangle (r) {
    var x1 = r.x;
    var x2 = r.x + r.width;
    var y1 = r.y;
    var y2 = r.y + r.height;

    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([
           x1, y1, x2, y1, x1, y2, x1, y2, x2, y1, x2, y2]),
           gl.STATIC_DRAW);
    gl.uniform4fv(colorLocation, r.color);
    gl.drawArrays(gl.TRIANGLES, 0, 6);
}
```

Ahora el código de render quedará así:

```
// draw ball
gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);
drawRectangle(ball);
gl.bindBuffer(gl.ARRAY_BUFFER, null);
```

Si vemos que la pelota no está cuadrada, debemos hacer el canvas cuadrado, puesto que las coordenadas, al estar normalizadas entre -1 y 1, serían más largas en una dirección que en otra. Para ello cambiaríamos los parámetros del canvas en la sección body del html al principio:

```
<canvas width="350" height="350" id="my_Canvas"></canvas>
```

Animando la pelota

Moviendo la pelota

Mover la pelota es tan fácil como modificar sus campos x e y antes de pintar:

```
// animate ball
ball.x += 0.01;
ball.y += -0.02;
```

Estableciendo la velocidad como un vector

En lugar de cambiar las coordenadas a mano, es mejor definir un vector velocidad y añadirlo a la estructura de datos. Con él controlaremos tanto la magnitud como la dirección de movimiento.

```
    var ball = {
         'x': -0.5, 'y': -0.3,
         'width': 0.1, 'height': 0.1,
         'color':[0,0,1,1],
         'speedX':0.01, 'speedY': -0.02
       };
```

El código en el render después de pintar sería:

```
// animate ball
ball.x += ball.speedX;
ball.y += ball.speedY;
```

Calculando los rebotes

Debemos calcular cuando la pelota toca uno de los márgenes. Primero empezamos por el suelo:

```
if (ball.y <=-1) ball.speedY *= -1;
```

Luego continuamos por la derecha:

```
if (ball.x + ball.width >= 1) ball.speedX *= -1;
```

Y luego intenten terminarlo para los 4 bordes.

Leyendo eventos para interactuar

Vamos a pintar el jugador y moverlo con las teclas

Pintando el rectángulo del player

Creamos otra estructura para pintar el player, y en el código de render pintamos los dos: pelota y player

```
var player1 = {
   'x': -0.8, 'y': -0.3,
   'width': 0.1, 'height': 0.4,
   'color':[1,1,1,1]
 };

 // draw ball and player1
 gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);
 drawRectangle(ball);
 drawRectangle(player1);
 gl.bindBuffer(gl.ARRAY_BUFFER, null);
```

Moviendo el player

Para recoger los eventos de teclado debemos decirselo al objeto windows, dándole la función del callback, y luego allí nos llega el código de la tecla:

```
// Invoke `processKey` on the `onkeydown` event
document.onkeydown = onKeyDown;
```

```
// Process key events by updating global orientation values
      function onKeyDown(key) {

      switch (key.keyCode) {
        // up arrow
        case 38: {
          player1.y += 0.05;
          break;
        }
        // down arrow
        case 40: {
          player1.y -= 0.05;
          break;
        }
      }
      }
```

Mejorando el performance del teclado

Ahora mismo el movimiento no es bueno, porque los eventos del teclado de Windows, cuando mantenemos pulsado una tecla, funcionan de la siguiente manera: se dispara un evento inicial primero, y después de unos instantes aparecen un montón de eventos consecutivos. Y no queremos esto.

Lo que vamos a hacer es recoger los eventos KeyDown y KeyUp por separado. De esta forma, en el down activamos un flag que haga mover continuamente al player, y en el up desactivamos el flag y paramos el movimiento.

Comenzaremos capturando los dos eventos, down y up:

```
// Invoke `processKey` on the `onkeydown` event
document.onkeydown = onKeyDown;
document.onkeyup = onKeyUp;
```

Aparte añadimos el campo 'movement' al player1:

```
       var player1 = {
         'x': -0.8, 'y': -0.3,
         'width': 0.1, 'height': 0.4,
         'color':[1,1,1,1],
         'movement': 0
       };
```

Luego creamos los dos callbacks:

```
      // recogemos las teclas
      function onKeyDown(key) {
      switch (key.keyCode) {
        // up arrow
        case 38: {
          player1.movement = 0.02;
```

```
        break;
      }
      // down arrow
      case 40: {
        player1.movement = -0.02;
        break;
      }
    }
    }

    function onKeyUp(key) {
     player1.movement = 0;
     }
```

Y por último animamos el player dentro del render

```
  player1.y += player1.movement;
```

Jugando también en móviles

Para que nos sirva en un móvil tenemos que irnos a los eventos ontouchstart, ontouchmove y ontouchend (lo estoy mirando en este tutorial). Así que lo que hay que hacer es añadir los callbacks:

```
// listeners for mobile events
document.ontouchmove = onTouchMove;
document.ontouchstart = onTouchStart;
document.ontouchend = onTouchEnd;
```

Entonces nos declaramos una variable global llamada prevTouchY, donde almacenar la coordenada Y del pixel que picamos. Y declaramos los tres métodos

```
// eventos en el móvil
function onTouchStart(e) {
    touchobj = e.changedTouches[0]; // reference first touch
    prevTouchY = parseInt(touchobj.clientY);
 }

function onTouchMove(e) {
    touchobj = e.changedTouches[0] // reference first touch
    touchY = parseInt(touchobj.clientY);
    difY = touchY - prevTouchY;
    player1.movement = -difY*0.0005;
    prevTouchY = touchY;
}

function onTouchEnd(e) {
    player1.movement = 0;
}
```

Bloqueando los eventos standard del navegador

La aplicación tal cual es difícil de manejar, debido que el navegador interpreta algunos gestos por su cuenta, como hacer scrolling, zoom, o recargar la página cuando arrastramos hacia abajo. Para evitar este problema, y que los eventos nos lleguen sólo a nuestro código y no al navegador, hay que hacer tres cambios:

1) Para evitar que se pueda desplazar la página html de arriba abajo y aparezcan las barras de scroll, hay que añadir dentro del style del head del html, antes de la declaración de #my_Canvas, la siguiente línea,:

```
body, html { overflow: hidden; }
```

2) Para forzar que el viewport de webgl ocupe todo el ancho de nuestra pantalal, añadir al final del head

```
<meta name="viewport" content="width=device-width, user-scalable=no">
```

3) Para que el área del canvas ocupe casi todo el viewport (95%), modificar la creación del objeto canvas en el body por la siguiente línea

```
<canvas style="width:95%" width = "350" height = "350" id = "my_Canvas"></canvas>
```

Posibles ampliaciones

- Calcular el rebote de la bola con el player
- Calcular el rebote diferente cuando golpeamos con una esquina
- Añadir un segundo player con otras teclas
- Llevar puntuación
- Meter más bolas simultáneas?
- Poder acelerar y frenar la pelota?
- Empezar un Breakout

Cómo incluir en la propia página las instrucciones de cómo manejar

Escribir texto en html es más fácil que en webgl. Simplemente habría que incluir el texto que queremos mostrar al principio del body del html con los tags y :

```
<p>Teclas de control:</p>
<p>    - flecha arriba: sube jugador.</p>
<p>    - flecha abajo: baja jugador. </p>
```

Cómo escribir un marcador sencillo en html

Primero en el body del html escribimos esto:

```
<h3 id="marcador"> Marcador: 0-0</h3>
```

Y luego cuando queramos modificar el texto (por ejemplo, cuando player1 meta un gol) hacemos esto:

```
document.getElementById("marcador").innerHTML = "Marcador 1-0"
```