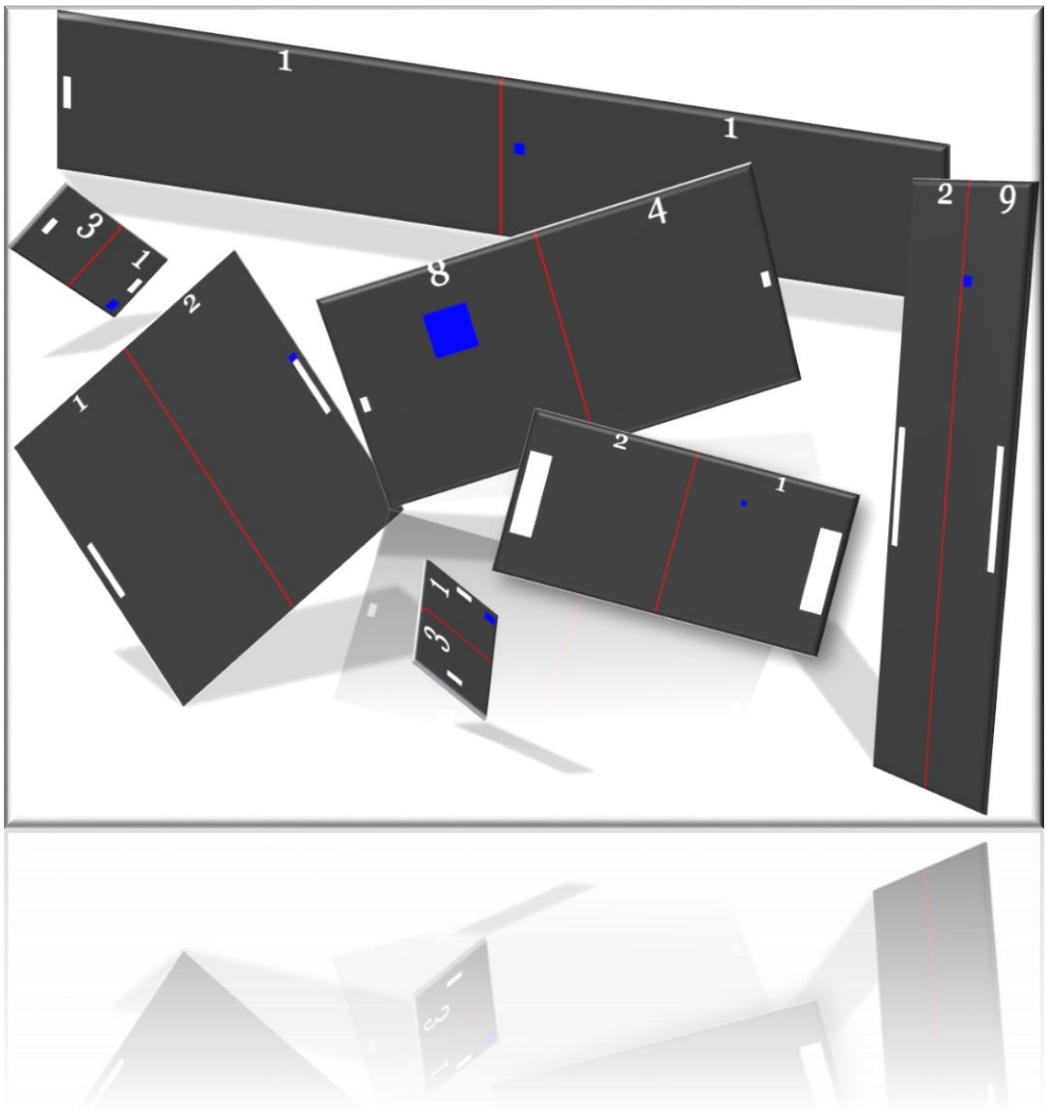


PingPong

CONTENIDO

Introducción 3
Creación de carpeta y ficheros 3
Dibujo del campo 5
Dibujo de la pelota 7
Pequeño 'divertimento' 10
Dibujo de las paletas 12
Movimiento de las paletas 14
Rebote de la pelota con las palas 17
Mostrar marcador 19
Incluir sonido 21
Posibles funcionalidades 'extras' 23
- Aumento de velocidad en los rebotes 23
- Aumento de velocidad en las palas 24
- Pausa/arranque del juego 25
- Habilitar/deshabilitar sonido 27

Resumen de controles 29
FUENTES 29
- pingpong.html 29
- estilos.css 29
- pingpong.js 30

INTRODUCCIÓN

A continuación, vamos a mostrar como fabricar el clásico juego '**pingpong'** tan versionado en multitud de lenguajes y que nos servirá para practicar un poco con **HTML**, con **CSS** y especialmente con **JavaScript**.

La idea es que, al acabar el juego, dispongamos de una página **HTML** sobre la que podamos mostrar un campo, con 2 palas para interceptar el movimiento de la pelota que estará rebotando entre los laterales del campo y las palas.

Asimismo, añadiremos alguna funcionalidad como puede ser incluir un sonido cada vez que la pelota rebote con una pared o pala o un sonido diferente cuando se anote un punto por parte de alguno de los jugadores. También visualizaremos un marcador con los puntos que va consiguiendo cada jugador.

Lo importante sobre todo es, ver cómo podemos parametrizar al máximo todas las variables que nos interesen para que, cambiar de tamaño el campo, las palas, la pelota o, variar la velocidad o indicar las teclas que queremos usar para mover las palas, sea cuestión de cambiar solo un parámetro y ya está.

CREACIÓN DE CARPETA Y FICHEROS

En primer lugar, crearemos una carpeta en la que iremos depositando los diferentes recursos que compondrán nuestro juego. Inicialmente, crearemos una página HTML simplemente con un canvas que represente el campo sobre el cual haremos rebotar la pelota y el resto de funcionalidades descritas. Crearemos también un archivo llamado estilos **.css** donde definir el color del campo. Dicho fichero de estilos, lo referenciaremos en el HTML mediante el tag <**link**> tal y como mostraremos más adelante.

Llamaremos a la página **pingpong.html** y la inicializaremos con el siguiente contenido:

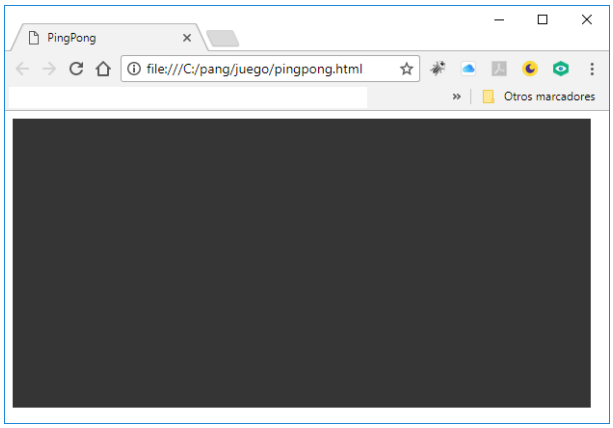
```
<!DOCTYPE HTML>
<html>
<head>
    <title>PingPong</title>
    <link rel="stylesheet" href="estilos.css" />
</head>
<body>
    <div>
        <canvas id="campo" width="600" height="300"></canvas>
    </div>
</body>
</html>
```

Observe como hacemos referencia a nuestro fichero de **estilos.css** y también, como definimos nuestro elemento **canvas** con un identificador (**id**) y una dimensión (**width** y **height**) que podemos variar a nuestro antojo para comprobar como se muestra el campo. Canvas es uno de los nuevos elementos de **HTML5**.

El fichero **estilos.css** tendrá inicialmente el siguiente contenido:

```
#campo {
    background-color: #353535;
}
```

Una vez guardados ambos ficheros, abra la página en un navegador y observe el resultado:



Por el momento, sólo vemos un rectángulo gracias al **canvas**. Le invitamos a que pruebe otros colores y dimensiones para comprobar lo fácil que es cambiar estas características.

A continuación, crearemos un fichero denominado **pingpong.js** que será el que contenga nuestro código fuente **Javascript** y será el encargado de dar la funcionalidad que pretendemos. Este fichero lo crearemos en nuestra carpeta juego y será referenciado en nuestra página HTML mediante la instrucción:

```
<script defer src="pingpong.js"></script>
```

Podemos ver que, **src** indica la ubicación y nombre del fichero **js**. Por otra parte, **defer**, permite que la página pueda ir mostrando partes de la misma mientras se descarga totalmente el archivo **js** pero, la script, no se ejecutará hasta que se haya descargado por completo. **Defer** es una de las nuevas características de **HTML5**.

Ahora, nuestra página HTML contendrá lo siguiente:

```
<!DOCTYPE HTML>
<html>
<head>
    <title>PingPong</title>
    <link rel="stylesheet" href="estilos.css" />
    <script defer src="pingpong.js"></script>
```

```
</head>
<body>
    <div>
        <canvas id="campo" width="600" height="300"></canvas>
    </div>
</body>
</html>
```

A continuación, vamos a dibujar una pelota y la haremos rebotar indefinidamente por las paredes de nuestro campo. Simplemente usaremos la siguiente secuencia:

El dibujo del campo consistirá simplemente en dibujar el **canvas** y colocar una línea vertical en el centro del mismo para representar la división del campo. Para ello, obtendremos el objeto asociado al **canvas**:

```
  canvas = document.getElementById("campo");
```

y, mediante su contexto (**context**), usaremos los métodos **fillStyle** para determinar el color del rectángulo a dibujar, **clearRect** para limpiar el campo y **fillRect** para dibujar el rectángulo de separación de color rojo:

```
  context.fillStyle = "red";
  context.clearRect(0, 0, ancho_canvas, alto_canvas);
  context.fillRect(ancho_canvas/2, 0, 2, alto_canvas);
```

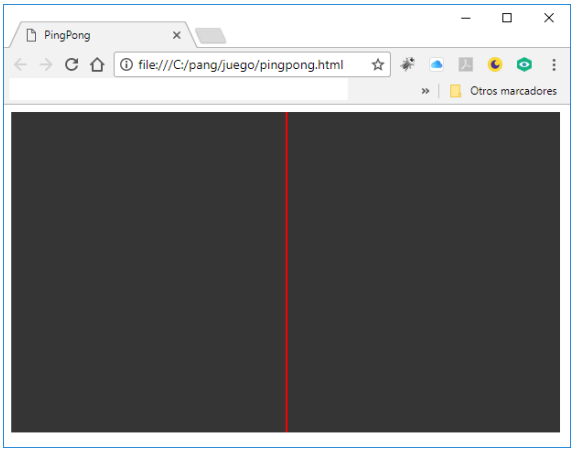
DIBUJO DEL CAMPO

Vamos a empezar a introducir código en nuestro fichero **pingpong.js** creando una función **Principal** que será la que llamemos en primer lugar, donde obtendremos el objeto **canvas** tal y como habíamos comentado, obtendremos las dimensiones del campo (**ancho_canvas** y **alto_canvas**), obtendremos su contexto (**context**) para poder dibujar sobre el campo y dibujaremos el campo llamando a la función **dibuja_campo()** que crearemos para disponer de una primera versión donde se vea precisamente el campo de juego. La script, en este punto poseerá el siguiente contenido:

```
function principal() {
    canvas = document.getElementById("campo");
    ancho_canvas = canvas.width;
```

```
    alto_canvas = canvas.height;
    context = canvas.getContext("2d");
    dibuja_campo();
}
function dibuja_campo() {
    context.fillStyle = "red";
    context.clearRect(0, 0, ancho_canvas, alto_canvas);
    context.fillRect(ancho_canvas / 2, 0, 2, alto_canvas);
}
principal();
```

Si visualizamos nuestra página HTML podemos ver nuestro campo:



Organizando mejor nuestra script, podemos crear una función en la que agruparemos las inicializaciones generales y que llamaremos **inicializa_parametros()** con el siguiente contenido:

```
function inicializa_parametros() {
    // Canvas campo
    canvas = document.getElementById("campo");
    ancho_canvas = canvas.width;
    alto_canvas = canvas.height;
    context = canvas.getContext("2d");
}
```

Ahora, nuestra script **pingpong.js** contiene lo siguiente:

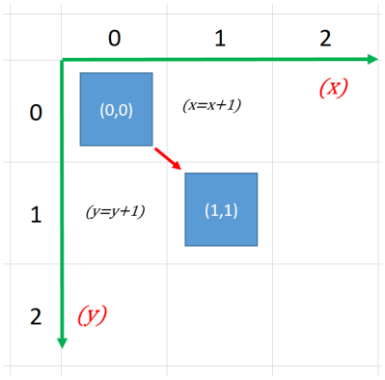
```
function principal() {
    inicializa_parametros();
    dibuja_campo();
}
function inicializa_parametros() {
    // Canvas campo
    canvas = document.getElementById("campo");
```

```
    ancho_canvas = canvas.width;
    alto_canvas = canvas.height;
    context = canvas.getContext("2d");
}
function dibuja_campo() {
    context.fillStyle = "red";
    context.clearRect(0, 0, ancho_canvas, alto_canvas);
    context.fillRect(ancho_canvas / 2, 0, 2, alto_canvas);
}
principal();
```

DIBUJO DE LA PELOTA

Prosigamos con el dibujo de la pelota rebotando por el campo indefinidamente. La idea es realizar un bucle dentro del cual, vayamos dibujando un punto en el campo, cuyas coordenadas varíen en cada una de las iteraciones del bucle. Lo clásico en este juego es que, en cada iteración, las coordenadas **x** e **y** del punto que representan la pelota, se incrementen para hacer avanzar la pelota de una posición a otra. En la secuencia sencilla que inicialmente habíamos descrito para nuestra primera versión del juego, mostrábamos el apartado **'Cálculo de coordenadas pelota'** el cual, se encargaba de determinar cuáles serían las coordenadas **x** e **y** para cada iteración.

Por ejemplo, si la pelota parte de la posición **(0,0)** e incrementamos en una unidad sus coordenadas **x** e **y**, la pelota se desplazará a la posición **(1,1)**:

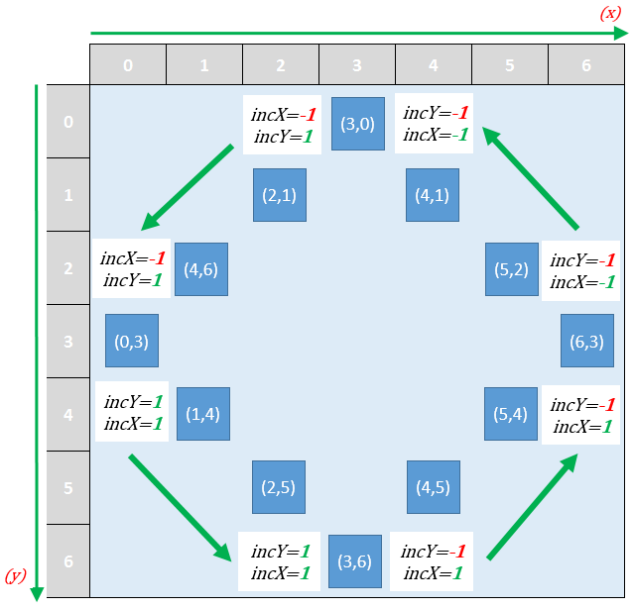


En este supuesto, hemos asumido que, por defecto, la pelota es cuadrada y que posee un alto y ancho de una unidad. También hemos supuesto que el incremento en ambos ejes será también de una unidad. Como podemos imaginar, a mayor incremento, mayor velocidad pero, por el momento empezaremos con una velocidad constante.

Para conseguir que la pelota rebote con los laterales del campo, en cada iteración, tendremos que contrastar la nueva posición de la pelota tras el incremento de sus coordenadas, con los límites del campo. Por tanto, la posición de la pelota sobre el eje **x** no podrá ser inferior a **0** (limite izquierdo del campo) ni podrá ser superior al ancho del campo (limite derecho del campo determinado por nuestra variable **ancho_canvas**). De la misma forma, la posición de la pelota sobre el eje **y**, no podrá ser inferior a **0** (límite superior del campo) ni mayor a la altura del canvas (o sea, **alto_canvas**). Por tanto, cuando se detecte alguna de las situaciones 'limite' tendremos que cambiar el incremento que proceda para que la pelota

cambie de sentido. Si la pelota va a llegar al límite izquierdo o derecho del campo, cambiaremos el signo de su incremento **x**. Si la pelota llega a la parte superior o inferior de la pantalla, cambiaremos su incremento **y**.

Veamos en la siguiente imagen cómo llega una pelota a alguno de los límites y como cambiamos su incremento:



Las flechas indican como llega la pelota a alguno de los limites. Vemos los valores de incremento con los que llegan y también con los que se van tras el rebote en cada uno de los límites del campo.

A continuación, crearemos una función para calcular las coordenadas de la pelota y otra para dibujarla.

Antes de nada, añadiremos algunas inicializaciones para los nuevos elementos descritos (coordenadas, incrementos, dimensiones de la pelota, etc.) en nuestra función de inicialización:

```
function inicializa_parametros() {
    // Canvas campo
    canvas = document.getElementById("campo");
    ancho_canvas = canvas.width;
    alto_canvas = canvas.height;
    context = canvas.getContext("2d");
    // Parametros pelota
    x = ancho_canvas / 2;
    y = alto_canvas / 2;
    ancho_pelota = 14;
    alto_pelota = 14;
    inc_pelota = 1; // Avance de la pelota en cada iteracion
    incX = inc_pelota;
    incY = inc_pelota;
}
```

Decidimos que el punto inicial de la pelota será desde el centro de la pantalla, que el tamaño de la pelota será de **14** pixeles y que el incremento de la pelota será **1**. Todos estos parámetros pueden variarse para comprobar el efecto que se produce sobre el juego, pero de momento, tomemos esta situación como punto de partida.

Las coordenadas de la pelota las calcularemos de la siguiente forma:

```
// Calculo de coordenadas y cambio de incrementos
function calcula_coordenadas_pelota() {
    x += incX;
    y += incY;
    if (x + ancho_pelota > ancho_canvas || x < 0) {
        incX = -incX;
    }
    if (y + alto_pelota > alto_canvas || y < 0) {
        incY = -incY;
    }
}
```

La pelota, la dibujaremos con la siguiente función:

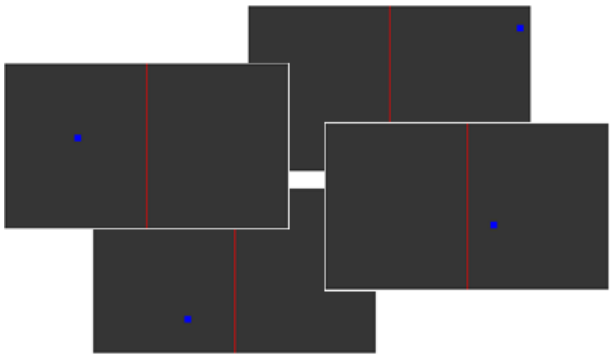
```
// Muestra la pelota en una posicion
function dibuja_pelota(x, y) {
    context.fillStyle = "blue";
    context.fillRect(x, y, ancho_pelota, alto_pelota);
}
```

Ahora, solo nos falta crear una función **'bucle'** que permita reproducir la secuencia que hemos explicado y ponga en marcha todo el mecanismo. Dicha función, tendrá que ser llamada desde la función principal y ambas funciones, tendrán en este punto lo siguiente:

```
function principal() {
    inicializa_parametros();
    bucle();
}
function bucle() {
    dibuja_campo();
    calcula_coordenadas_pelota();
    dibuja_pelota(x, y);
    setTimeout(bucle, 4);
}
```

Observe que usamos la función **setTimout()** para indicar que la función bucle se ha de ejecutar cada **4** milisegundos de forma ininterrumpida.

Abra la página en el navegador y observe cómo el juego va tomando forma y la pelota rebota por todo el campo:

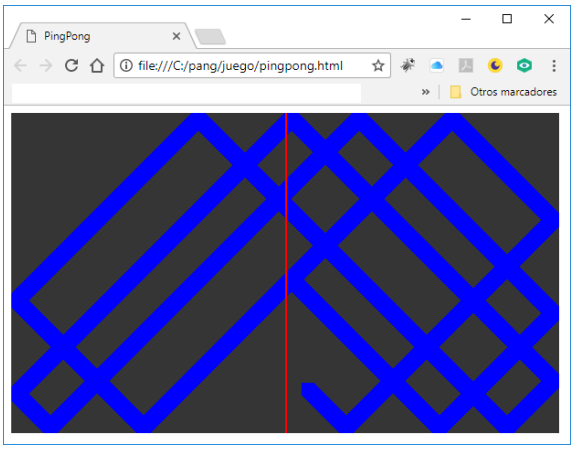


PEQUEÑO 'DIVERTIMENTO'

A modo de diversión, le invitamos a que haga una copia de la carpeta y de sus elementos tal y como los tenemos definidos hasta el momento (p.e. copie carpeta **juegos** a **juegos2**), y en la nueva copia, haga algunas modificaciones para jugar con el resultado conseguido. Una vez haya hecho la copia comente la línea que contiene la sentencia **clearRect** en la función **dibuja_campo()** para comprobar el efecto que se produciría si no limpiáramos el campo cada vez que dibujamos la pelota y dejáramos una estela de su recorrido. Modifique la función para que quede así:

```
function dibuja_campo() {
    context.fillStyle = "red";
    // context.clearRect(0, 0, ancho_canvas, alto_canvas);
    context.fillRect(ancho_canvas / 2, 0, 2, alto_canvas);
}
```

Recargue la página en el navegador y observe el efecto al que nos referimos:



Podríamos seguir jugando con esta derivación para hacer que cada vez que algún incremento cambie de signo y la pelota rebote, se calcule un color aleatorio y que la estela de la pelota fuese cambiando de color. Para ello, bastaría con modificar un poco la función **dibujaPelota()** para que tuviese lo siguiente:

```
function dibuja_pelota(x, y) {
    // context.fillStyle = "blue";
    if (calcula_color) {
        c1 = Math.floor((Math.random() * 255) + 1);
        c2 = Math.floor((Math.random() * 255) + 1);
        c3 = Math.floor((Math.random() * 255) + 1);
        calcula_color = false;
    }
    context.fillStyle = 'rgb(' + c1 + ',' + c2 + ',' + c3 + ')';
    context.fillRect(x, y, ancho_pelota, alto_pelota);
}
```

Ahora, **fillStyle** se definirá mediante **rgb** indicando la composición de cada color que, calculamos aleatoriamente mediante **Math.random()**, buscando un número entre **0** y **255** y quedándonos con la parte entera (**Math.floor()**).

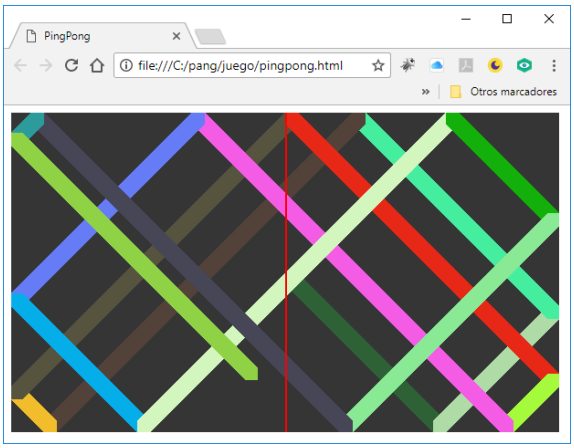
Lógicamente, tendríamos que inicializar las siguientes variables al inicio de nuestro fichero **pongpong.js**:

```
calcula_color = true;
c1 = 0;c2 = 255;c3 = 0;
```

Y por último, tendríamos que modificar nuestra función **calcula_coordenadas_pelota()** para que cada vez que algún incremento cambiara de signo, se activara el switch **calcula_color** y provocara un recalculo de colores aleatorios de forma que la función quedaría así:

```
function calcula_coordenadas_pelota() {
    x += incX;
    y += incY;
    if (x + ancho_pelota > ancho_canvas || x < 0) {
        incX = -incX;
        calcula_color = true;
    }
    if (y + alto_pelota > alto_canvas || y < 0) {
        incY = -incY;
        calcula_color = true;
    }
}
```

Refresque la pantalla y observe el colorido:



DIBUJO DE LAS PALETAS

Seguidamente, continuando con el juego tal y como lo habíamos dejado , vamos a incluir las raquetas de los jugadores para que la pelota pueda rebotar sobre las mismas y realizar el juego que pretendemos.

Para este cometido, hemos de considerar varias acciones. Por una parte, hemos de definir una serie de variables para determinar el tamaño y la posición de las palas. Por otra, hemos de dibujar las palas en si mismas. Además, tendremos que incluir estos 2 nuevos elementos en la lógica de cálculo de coordenadas de la pelota para que cuando tropiece con ellas, la pelota actúe como habíamos explicado antes en relación a los límites laterales del campo.

Empecemos por definir las variables que usaremos para las paletas. Para ello, modificaremos nuestra función **inicializa_parametros()** para que quede de la siguiente manera:

```
function inicializa_parametros() {
    // Canvas campo
    canvas = document.getElementById("campo");
    ancho_canvas = canvas.width;
    alto_canvas = canvas.height;
    context = canvas.getContext("2d");
    // Parametros pelota
    x = ancho_canvas / 2;
    y = alto_canvas / 2;
    ancho_pelota = 14;
    alto_pelota = 14;
    inc_pelota = 1; // Avance de la pelota en cada iteracion
    incX = inc_pelota;
    incY = inc_pelota;

    // Parametros Palas
    separacion = 12; // Separacion desde el extremo del campo
    alto_pala = alto_canvas / 5;
    ancho_pala = 10;
    inc_pala = 2; // Posiciones que avanza cuando se mueve
    // Posiciones de las paletas de jugadores (i)zq y (d)er.
    jiX = separacion;
    jiY = alto_canvas / 2 - alto_pala / 2;
```

```
    jdX = ancho_canvas - separacion - ancho_pala;
    jdY = jiY;
}
```

La **separación**, determina la distancia entre los laterales y las palas. El **alto** de la pala lo definimos como la quinta parte del alto del campo, pero puede tener el valor que deseemos. El ancho de la pala lo fijamos a **10** pixeles. **jiX** y **jiY** representan las coordenadas de la esquina superior izquierda de la pala del jugador izquierdo. **jdX** y **jdY** son las coordenadas del jugador derecho. Ambas paletas se dibujan inicialmente, centradas verticalmente.

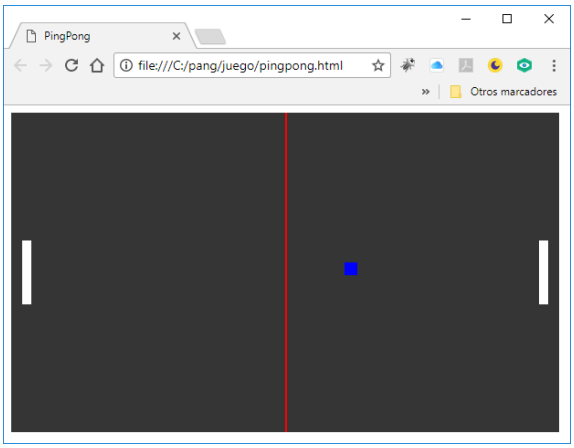
Ahora, crearemos las siguientes 2 funciones para dibujar respectivamente las paletas del jugador izquierdo y del jugador derecho:

```
// Muestra pala del jugador izquierdo
function dibuja_jugadorI(jiX, jiY) {
    context.fillStyle = "white";
    context.fillRect(jiX, jiY, ancho_pala, alto_pala);
}
// Muestra pala del jugador derecho
function dibuja_jugadorD(jdX, jdY) {
    context.fillStyle = "white";
    context.fillRect(jdX, jdY, ancho_pala, alto_pala);
}
```

Por último, incluiremos la llamada a dichas funciones en el bucle de la siguiente manera:

```
function bucle() {
    dibuja_campo();
    calcula_coordenadas_pelota();
    dibuja_pelota(x, y);
    dibuja_jugadorI(jiX, jiY);
    dibuja_jugadorD(jdX, jdY);
    setTimeout(bucle, 4);
}
```

Si refrescamos nuestra página en el navegador, observaremos cómo se muestran las paletas de los jugadores:



MOVIMIENTO DE LAS PALETAS

Para realizar el movimiento de las paletas, asignaremos un par de teclas para cada uno de los jugadores, de forma que indiquen al juego si quieren desplazar su pala hacia arriba o hacia abajo. Por ejemplo, podemos decir que el jugador izquierdo desplazará la pala hacia arriba si pulsa la letra '**A**' y hacia abajo si pulsa la letra '**Z**'. El jugador derecho, moverá su pala pulsando las teclas '**K**' (**arriba**) y '**Z**' (**abajo**).

Para obtener el evento pulsación de teclas, tendremos que añadir unos **listeners** al documento capaces de indicar si una tecla ha sido pulsada o no. La función encargada de capturar estos eventos se llamará **KeyListener()** y contendrá lo siguiente:

```
// KEY LISTENER
function KeyListener() {
    this.pressedKeys = [];
    this.keydown = function (e) {
        this.pressedKeys[e.keyCode] = true;
    };
    this.keyup = function (e) {
        this.pressedKeys[e.keyCode] = false;
    };
    document.addEventListener("keydown", this.keydown.bind(this));
    document.addEventListener("keyup", this.keyup.bind(this));
}
KeyListener.prototype.isPressed = function (key) {
    return this.pressedKeys[key] ? true : false;
};
```

En definitiva, tendremos que cada vez que se pulse una tecla, dispondremos de la función que nos permitirá saber qué se ha pulsado y podremos preguntar por ello para actuar en consecuencia.

Una vez incluida esta función en nuestro código, pasaremos a crear una variable en nuestra función **principal()** tal y como mostramos a continuación:

```
function principal() {
    inicializa_parametros();
    keys = new KeyListener();
    bucle();
}
```

Vamos a definir una serie de variables para poder identificar las teclas pulsadas que nos interesan y lo haremos también para tratar de parametrizar al máximo estos valores de forma que elegir unas teclas diferentes para controlar el movimiento de las paletas sea tan simple como cambiar una tecla por otra.

Así pues, en nuestra función **inicializa_parametros()**, añadiremos las siguientes definiciones a continuación de las que habíamos añadido para los parámetros de las palas:

```
// Caracteres que representan movimientos y controles
CAR_I_ARRIBA_U = "A"; // Mover paleta izquierda arriba mayusculas
```

```
CAR_I_ARRIBA_L = CAR_I_ARRIBA_U.toLowerCase(); // PI arriba minusculas
CAR_I_ABAJO_U = "Z"; // Mover paleta izquierda abajo mayusculas
CAR_I_ABAJO_L = CAR_I_ABAJO_U.toLowerCase(); // PI abajo minusculas

CAR_D_ARRIBA_U = "K"; // Mover paleta derecha arriba mayusculas
CAR_D_ARRIBA_L = CAR_D_ARRIBA_U.toLowerCase(); // PD arriba minusculas
CAR_D_ABAJO_U = "M"; // Mover paleta derecha abajo mayusculas
CAR_D_ABAJO_L = CAR_D_ABAJO_U.toLowerCase(); // PD abajo minusculas

// KeyCode (Movimiento paletas)
KEY_I_ARRIBA_U = CAR_I_ARRIBA_U.charCodeAt(0); // mayusculas
KEY_I_ARRIBA_L = CAR_I_ARRIBA_L.charCodeAt(0); // minusculas
KEY_I_ABAJO_U = CAR_I_ABAJO_U.charCodeAt(0); // Mayusculas
KEY_I_ABAJO_L = CAR_I_ABAJO_L.charCodeAt(0); // minusculas

KEY_D_ARRIBA_U = CAR_D_ARRIBA_U.charCodeAt(0); // Mayusculas
KEY_D_ARRIBA_L = CAR_D_ARRIBA_L.charCodeAt(0); // Minusculas
KEY_D_ABAJO_U = CAR_D_ABAJO_U.charCodeAt(0); // Mayusculas
KEY_D_ABAJO_L = CAR_D_ABAJO_L.charCodeAt(0); // Minusculas

// Inicializacion deteccion de pulsaciones para KeyCode
PI_ARRIBA = false;
PI_ABAJO = false;
PD_ARRIBA = false;
PD_ABAJO = false;
```

Podemos observar varios grupos de variables destinadas a definir lo siguiente:

- Las **CAR_I** o **CAR_D** (p.e. **CAR_I_ARRIBA_U**) permiten definir qué carácter queremos usar para indicar el movimiento de la pala derecha (**D**) o izquierda (**I**) cuando queramos que se desplace hacia **ARRIBA** o hacia **ABAJO**. La terminación **_U** o **_L** indica si la tecla pulsada está en mayúsculas (**UPPERCASE**) o en minúsculas (**LOWERCASE**) para permitir al jugador que pueda jugar independientemente de si el teclado en mayúsculas o minúsculas.
- Las **KEY_I** o **KEY_D** (p.e. **KEY_I_ARRIBA_U**) obtienen el carácter Unicode correspondiente a la letra pulsada.
- Las **PI_** son switches que simplemente indican si la tecla se ha pulsado (**true**) o no (**false**).

A continuación, vamos a definir una nueva función que nos permita evaluar si alguna de las teclas que estamos considerando han sido pulsadas o no. La función se llamará **controlar_pulsacion()** y contendrá lo siguiente:

```
function controlar_pulsacion() {
    if (keys.isPressed(KEY_I_ABAJO_U || KEY_I_ABAJO_L)) { // Abajo Izq
        PI_ABAJO = true;
    } else if (keys.isPressed(KEY_I_ARRIBA_U || KEY_I_ARRIBA_L)) { //
Arriba Izq
        PI_ARRIBA = true;
    }
    if (keys.isPressed(KEY_D_ABAJO_U || KEY_D_ABAJO_L)) { // Abajo Der
        PD_ABAJO = true;
    } else if (keys.isPressed(KEY_D_ARRIBA_U || KEY_D_ARRIBA_L)) { //
Arriba Der
        PD_ARRIBA = true;
    }
```

```
    calcula_coordenadas_pala();
}
```

Como puede observarse, si se ha pulsado alguna de las teclas que hemos definido, se activará el switch correspondiente a **true** y posteriormente, será tenido en cuenta para calcular las coordenadas de la pala en otra nueva función denominada **calcula_coordenadas_pala()** que definimos a continuación:

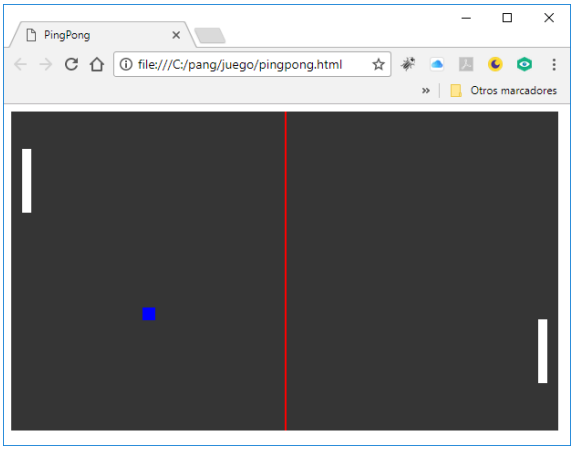
```
function calcula_coordenadas_pala() {
    if (PI_ARRIBA == true) {
        jiY -= inc_pala;
        if (jiY < 0) jiY = 1;
        PI_ARRIBA = false;
    }
    if (PI_ABAJO == true) {
        jiY += inc_pala;
        if (jiY + alto_pala > alto_canvas) jiY = alto_canvas -
alto_pala;
        PI_ABAJO = false;
    }
    if (PD_ARRIBA == true) {
        jdY -= inc_pala;
        if (jdY < 0) jdY = 1;
        PD_ARRIBA = false;
    }
    if (PD_ABAJO == true) {
        jdY += inc_pala;
        if (jdY + alto_pala > alto_canvas) jdY = alto_canvas -
alto_pala;
        PD_ABAJO = false;
    }
}
```

El movimiento hacia arriba de las palas, implica un incremento negativo (**-inc_pala**) mientras que el movimiento hacia abajo, lo implica positivo (**inc_pala**). También puede observarse que se controlan los limites tanto superior como inferior para que las coordenadas **jiY** o **jdY** no puedan ser inferiores a **0** (p.e. **`if (jiY < 0) jiY = 1`**) o superiores al alto del **canvas – alto de la pala** (p.e. **`if (jiY + alto_pala > alto_canvas) jiY = alto_canvas - alto_pala;`**).

Ya sólo nos falta incluir la llamada a la función **controlar_pulsacion()** dentro del bucle. Lo haremos de la siguiente manera:

```
function bucle() {
    dibuja_campo();
    calcula_coordenadas_pelota();
    controlar_pulsacion();
    dibuja_pelota(x, y);
    dibuja_jugadorI(jiX, jiY);
    dibuja_jugadorD(jdX, jdY);
    setTimeout(bucle, 4);
}
```

Si actualizamos nuestra página en el navegador, observaremos cómo podemos mover las palas al pulsar las teclas **A**, **Z** (para jugador **izquierdo**) o **K**, **M** (para el jugador **derecho**). Evidentemente, la pelota pasa por encima de las palas porque aún no hemos incluido la lógica que ha de controlar los rebotes pero nuestro juego, ya tiene una buena parte de la funcionalidad que esperábamos:



REBOTE DE LA PELOTA CON LAS PALAS

A continuación, haremos que cuando las coordenadas de la pelota coincidan con las coordenadas de las palas, el incremento **x** de la pelota cambie de signo generando el efecto '**rebote'**. Sin embargo, cuando la pelota llegue a cualquiera de los laterales y no se cruce con ninguna pala, consideraremos que se ha generado un '**punto'** a favor del jugador contrario y la pelota se perderá por dicho lateral dando origen al inicio de un nuevo '**punto'** a disputar.

Por tanto, creamos la función **inicio_punto()** la cual nos permitirá inicializar la situación inicial de cada uno de los puntos que se vayan a jugar en nuestro juego. La función poseerá el siguiente contenido:

```
// Inicializa valores para el inicio de un punto
function inicio_punto() {
    inicioY = Math.floor((Math.random() * alto_canvas / 2) + 1);
    x = ancho_canvas / 2; // Iniciamos desde el centro del campo
    y = inicioY; // La posicion vertical de la pelota es aleatoria

    // La direccion de la pelota en es aleatoria
    valorX = Math.floor((Math.random() * 100) + 1);
    if (valorX < 50) {
        incX = inc_pelota;
    } else {
        incX = -inc_pelota;
    }
    valorY = Math.floor((Math.random() * 100) + 1);
    if (valorY < 50) {
        incY = inc_pelota;
    } else {
        incY = -inc_pelota;
    }

    // Posiciones de las paletas de jugadores (i)zq y (d)er.
```

```
    jiX = separacion;
    jiY = alto_canvas / 2 - alto_pala / 2;
    jdX = ancho_canvas - separacion - ancho_pala;
    jdY = jiY;
}
```

Vemos que la posición inicial de la pelota, se hallará horizontalmente en la mitad del campo y verticalmente, se calculará una posición aleatoria para darle más imprevisibilidad al juego.

También calcularemos aleatoriamente el sentido **x** e **y** de la pelota para que no sepamos si irá hacia arriba o hacia abajo ni tampoco hacia qué jugador.

Por último, la inicializació0n del punto situará las paletas centradas verticalmente en sus respectivos lados.

Una vez definida la función de inicialización del punto, pasamos a modificar nuestra función **calcula_coordenadas_pelota()** para que tenga en cuenta las paletas o el inicio de punto si la pelota se pierde por alguno de los laterales.

La función **calcula_coordenadas_pelota()** contendrá ahora lo siguiente:

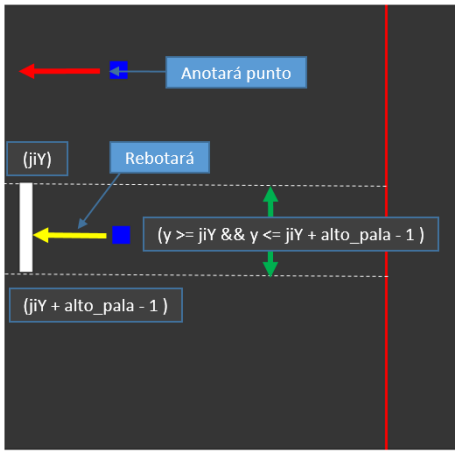
```
function calcula_coordenadas_pelota() {
    x += incX;
    y += incY;
    // Comprobacion de la pelota respecto de la pala izquierda
    if (y >= jiY && y <= jiY + alto_pala - 1) {
        if (x <= jiX + ancho_pala) {
            incX = -incX; // Cambio de direccion horizontal
            x = jiX + ancho_pala;
        }
    } else {
        // Si ha rebasado la posicion de la pala es un punto para el
        // jugador contrario.
        if (x < jiX - separacion) {
            inicio_punto();
        }
    }
    // Comprobacion de la pelota respecto de la pala derecha
    if (y >= jdY && y <= jdY + alto_pala - 1) {
        if (x + ancho_pelota >= jdX) {
            incX = -incX; // Cambio de direccion horizontal
            x = jdX - ancho_pelota;
        }
    } else {
        // Si ha rebasado la posicion de la pala es un punto para el
        // jugador contrario.
        if (x + ancho_pelota > jdX + separacion) {
            inicio_punto();
        }
    }
    // Si la pelota rebota en la parte superior o inferior de la
pantalla
    // cambia de direccion vertical.
    if (y + alto_pelota > alto_canvas || y < 0) {
```

```
        incY = -incY;
    }
}
```

Tal y como habíamos anticipado, la pelota puede encontrarse verticalmente en 2 situaciones respecto del eje vertical (**y**) y por tanto, con 2 consecuencias cuando se halle en la misma posición horizontal (**x**):

- Dentro del rango que ocupa la pala: la pelota rebotará
- Fuera del mismo: la pelota se colará en el lateral y supondrá un punto para el contrario

En la siguiente imagen destacamos uno de los ejemplos resueltos en la función:



En este momento, ya dispondríamos de una versión del juego que nos permitiría mover las palas y hacer que la pelota rebote contras las mismas. Vamos a continuar añadiendo alguna funcionalidad más como por ejemplo, mostrar un marcador con el resultado de la partida.

MOSTRAR MARCADOR

Para organizar un poco mejor nuestro juego, crearemos una función que llamaremos **inicio_partida()** y que contendrá lo siguiente:

```
// Inicializa valores para el inicio de una partida
function inicio_partida() {
    puntosI = 0;
    puntosD = 0;
    inicio_punto();
}
```

En definitiva, podemos observar que contiene un par de variables para acumular los puntos que vaya obteniendo cada jugador y una llamada a la función **inicio_punto()** para empezar un nuevo punto junto con la partida.

Añadiremos la llamada a esta función al final de muestra función **inicializa_parametros()** de forma que quedará de la siguiente manera:

```
function inicializa_parametros() {
```

```
    // Canvas campo
    canvas = document.getElementById("campo");
...
    PD_ABAJO = false;
    inicio_partida();
}
```

Ahora, modificaremos nuestra función **calcula_coordenadas_pelota()** para que cada vez que se inicie un punto, se incremente en uno la variable que representa la puntuación del jugador del lado contrario al lado por donde se ha 'colado' la pelota.

```
function calcula_coordenadas_pelota() {
    x += incX;
    y += incY;
...
    } else {
        // Si ha rebasado la posicion de la pala es un punto para el
        // jugador contrario.
        if (x < jiX - separacion) {
            inicio_punto();
            puntosD += 1;
        }
    }
...
        // Si ha rebasado la posicion de la pala es un punto para el
        // jugador contrario.
        if (x + ancho_pelota > jdX + separacion) {
            inicio_punto();
            puntosI += 1;
        }
    }
```

Vemos que la modificación consiste en incrementar una unidad justo debajo de la llamada a la función **inicio_punto()**. Si la pelota se pierde por el lateral izquierdo, el punto se lo añadimos al jugador derecho y viceversa.

Ahora que ya tenemos inicializada y controlada la puntuación de cada jugador, sólo nos queda mostrarla. Para ello, crearemos la función **dibuja_puntuacion()** con el siguiente contenido:

```
// Muestra puntuacion cada vez que se produce un cambio
function dibuja_puntuacion() {
    context.fillStyle = "white";
    context.font = "48px Georgia";
    context.fillText(puntosI, ancho_canvas / 4, 30);
    context.fillText(puntosD, (ancho_canvas * 3) / 4, 30);
}
```

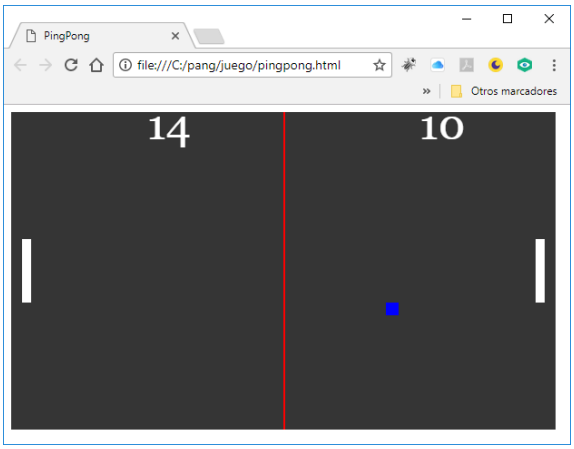
De nuevo, usaremos el objeto **context** y su método **fillText** para situar la puntuación centrando cada marcador en la mitad de cada campo (izquierdo: **1/4** y derecho: **3/4** del ancho total del campo).

Por último, añadiremos la llamada a esta función en nuestra función **bucle()** de la siguiente manera:

```
function bucle() {
    dibuja_campo();
```

```
    calcula_coordenadas_pelota();
    controlar_pulsacion();
    dibuja_pelota(x, y);
    dibuja_jugadorI(jiX, jiY);
    dibuja_jugadorD(jdX, jdY);
    dibuja_puntuacion();
    setTimeout(bucle, 4);
}
```

Veamos cómo se ve en nuestro navegador:



INCLUIR SONIDO

Podemos hacer que cada vez que rebote la pelota o se anote un punto, suene un sonido consiguiendo un mejor acabado del juego.

Para ello, basta con definir un par de objetos de tipo **Audio** a partir de un par de ficheros de audio (p.e. **wav**) y posteriormente, hacerlos sonar con su método **Play()**.

En nuestro caso, usaremos un audio llamado **beep.wav** para los rebotes y otro llamado **punto.wav** para los puntos. Incluiremos también un switch denominado **hay_sonido** para poder quitar y poner fácilmente el sonido.

Así pues, incluiremos en nuestra carpeta juego los archivos **beep.wav** y **punto.wav** que podemos obtener de cualquiera de los muchos recursos existentes en Internet.

Incluimos su definición en la función **inicializa_parametros()** y podemos hacerlo al principio de la misma de la siguiente manera:

```
function inicializa_parametros() {
    // Sonidos
    beep = new Audio("beep.wav");
    punto = new Audio("punto.wav");
    hay_sonido = true;

    // Canvas campo
```

```
    canvas = document.getElementById("campo");
    ...
```

Ahora, modificaremos la función **calcula_coordenadas_pelota()** para hacer sonar los rebotes y los puntos. Dicha función poseerá el siguiente contenido:

```
// Calculo de coordenadas de la pelota
function calcula_coordenadas_pelota() {
    x += incX;
    y += incY;
    // Comprobacion de la pelota respecto de la pala izquierda
    if (y >= jiY && y <= jiY + alto_pala - 1) {
        if (x <= jiX + ancho_pala) {
            incX = -incX; // Cambio de direccion horizontal
            x = jiX + ancho_pala;
            if (hay_sonido) beep.play();
        }
    } else {
        // Si ha rebasado la posicion de la pala es un punto para el
        // jugador contrario.
        if (x < jiX - separacion) {
            inicio_punto();
            puntosD += 1;
            if (hay_sonido) punto.play();
        }
    }
    // Comprobacion de la pelota respecto de la pala derecha
    if (y >= jdY && y <= jdY + alto_pala - 1) {
        if (x + ancho_pelota >= jdX) {
            incX = -incX; // Cambio de direccion horizontal
            x = jdX - ancho_pelota;
            if (hay_sonido) beep.play();
        }
    } else {
        // Si ha rebasado la posicion de la pala es un punto para el
        // jugador contrario.
        if (x + ancho_pelota > jdX + separacion) {
            inicio_punto();
            puntosI += 1;
            if (hay_sonido) punto.play();
        }
    }
    // Si la pelota rebota en la parte superior o inferior de la
pantalla
    // cambia de direccion vertical.
    if (y + alto_pelota > alto_canvas || y < 0) {
        incY = -incY;
        if (hay_sonido) beep.play();
    }
}
```

Evidentemente, podríamos seguir añadiendo funcionalidades y no acabar nunca de mejorarlo, pero en este punto damos por finalizado el juego y ya podemos disfrutar de él.

POSIBLES FUNCIONALIDADES 'EXTRAS'

AUMENTO DE VELOCIDAD EN LOS REBOTES

Si desea seguir añadiendo funcionalidades, podría plantearse aumentar la velocidad de la pelota cada vez que ésta rebota con una paleta. Para ello, definiremos un switch llamado **hay_inc_vel** que indicará si deseamos usar esta prestación o no y lo haremos en la función **inicializa_parametros()** en cualquier parte. Nosotros lo añadimos justo detrás del último parámetro relacionado con la pelota:

```
    inc_pelota = 1; // Avance de la pelota en cada iteracion
    incX = inc_pelota;
    incY = inc_pelota;
    hay_inc_vel = false; // true para aumentar velocidad en cada rebote
```

También modificaremos la función **calcula_coordenadas_pelota()** para que cada vez que el incremento cambie de signo, se aumente el incremento del mismo. Dicha función quedará de la siguiente manera:

```
// Calculo de coordenadas de la pelota
function calcula_coordenadas_pelota() {
    x += incX;
    y += incY;
    // Comprobacion de la pelota respecto de la pala izquierda
    if (y >= jiY && y <= jiY + alto_pala - 1) {
        if (x <= jiX + ancho_pala) {
            incX = -incX; // Cambio de direccion horizontal
            if (hay_inc_vel && Math.abs(incX) < 3) {
                if (incX < 1) {
                    incX -= 1
                } else {
                    incX += 1
                };
                if (incY < 1) {
                    incY -= 1
                } else {
                    incY += 1
                };
            }
            x = jiX + ancho_pala;
            if (hay_sonido) beep.play();
        }
    } else {
        // Si ha rebasado la posicion de la pala es un punto para el
        // jugador contrario.
        if (x < jiX - separacion) {
            inicio_punto();
            puntosD += 1;
```

```
            if (hay_sonido) punto.play();
        }
    }
    // Comprobacion de la pelota respecto de la pala derecha
    if (y >= jdY && y <= jdY + alto_pala - 1) {
        if (x + ancho_pelota >= jdX) {
            incX = -incX; // Cambio de direccion horizontal
            if (hay_inc_vel && Math.abs(incX) < 3) {
                if (incX < 1) {
                    incX -= 1
                } else {
                    incX += 1
                };
                if (incY < 1) {
                    incY -= 1
                } else {
                    incY += 1
                };
            }
            x = jdX - ancho_pelota;
            if (hay_sonido) beep.play();
        }
    } else {
        // Si ha rebasado la posicion de la pala es un punto para el
        // jugador contrario.
        if (x + ancho_pelota > jdX + separacion) {
            inicio_punto();
            puntosI += 1;
            if (hay_sonido) punto.play();
        }
    }
    // Si la pelota rebota en la parte superior o inferior de la
pantalla
    // cambia de direccion vertical.
    if (y + alto_pelota > alto_canvas || y < 0) {
        incY = -incY;
        if (hay_sonido) beep.play();
    }
}
```

Guarde el archivo y refresque su página en el navegador para comprobar como aumenta la velocidad cada vez que la pelota consigue rebotar con la pala. Para evitar que la velocidad sea excesiva, el incremento se ha limitado a **3**.

AUMENTO DE VELOCIDAD EN LAS PALAS

Para aumentar la velocidad de las palas, simplemente ha de modificar la variable **inc_pala** definida en la función **inicializa_parametros()**. Por ejemplo, aumente a 4 este valor y observe como las palas se mueven más deprisa:

```
    inc_pala = 4; // Posiciones que avanza cuando se mueve
```

PAUSA/ARRANQUE DEL JUEGO

Vamos a añadir funcionalidad para que, en un momento dado, pueda pausar el juego para abandonarlo a continuación o simplemente retomarlo.

Para este ejemplo, añadiremos 2 teclas para pausar el juego (**P**) y para arrancarlo (**S**). Al igual que hicimos para el control del movimiento de las palas, definiremos las teclas que queremos usar tanto en mayúsculas como en minúsculas para facilitar su uso al jugador.

En la función **inicializa_parametros()**, justo debajo de la inicialización de las variables de tipo **CAR_** que hicimos para las teclas de movimiento de las palas añadiremos las siguientes definiciones resaltadas en amarillo:

```
    CAR_D_ABAJO_L = CAR_D_ABAJO_U.toLowerCase(); // PD abajo minusculas

    CAR_PAUSA_U = "P"; // Caracter para pausa en mayusculas
    CAR_PAUSA_L = CAR_PAUSA_U.toLowerCase(); // Caracter para pausa en minusculas
    CAR_JUGAR_U = "S"; // Arrancar juego en mayusculas
    CAR_JUGAR_L = CAR_JUGAR_U.toLowerCase(); // Arrancar juego en minusculas

    // KeyCode (Movimiento paletas)
    KEY_I_ARRIBA_U = CAR_I_ARRIBA_U.charCodeAt(0); // mayusculas
```

Un poco más abajo y justo después de las variables de tipo **KEY_**, añadiremos la obtención del carácter de estas teclas modificando lo siguiente:

```
    KEY_D_ABAJO_L = CAR_D_ABAJO_L.charCodeAt(0); // Minusculas

    KEY_PAUSA_U = CAR_PAUSA_U.charCodeAt(0); // Mayusculas
    KEY_PAUSA_L = CAR_PAUSA_L.charCodeAt(0); // Minusculas
    KEY_JUGAR_U = CAR_JUGAR_U.charCodeAt(0); // Mayusculas
    KEY_JUGAR_L = CAR_JUGAR_L.charCodeAt(0); // Minusculas

    // Inicializacion deteccion de pulsaciones para KeyCode
    PI_ARRIBA = false;
```

La última definición, la incluiremos justo antes de finalizar la función antes de la llamada a **inicio_partida()** tal y como mostramos a continuación:

```
    PD_ABAJO = false;

    PAUSA = false;
    inicio_partida();
    }
```

Ahora, modificaremos la función **controlar_pulsacion()** para añadir el control de las nuevas teclas detrás de la llamada a **calcula_coordenadas_pala()** de la siguiente manera:

```
function controlar_pulsacion() {
    if (keys.isPressed(KEY_I_ABAJO_U || KEY_I_ABAJO_L)) { // Abajo Izq
        PI_ABAJO = true;
    } else if (keys.isPressed(KEY_I_ARRIBA_U || KEY_I_ARRIBA_L)) { //
Arriba Izq
        PI_ARRIBA = true;
    }
    if (keys.isPressed(KEY_D_ABAJO_U || KEY_D_ABAJO_L)) { // Abajo Der
        PD_ABAJO = true;
    } else if (keys.isPressed(KEY_D_ARRIBA_U || KEY_D_ARRIBA_L)) { //
Arriba Der
        PD_ARRIBA = true;
    }
    calcula_coordenadas_pala();
    if (keys.isPressed(KEY_PAUSA_U) || keys.isPressed(KEY_PAUSA_L)) {
        PAUSA = true;
        keys.pressedKeys[KEY_PAUSA_U] = false;
        keys.pressedKeys[KEY_PAUSA_L] = false;
    }

    if (keys.isPressed(KEY_JUGAR_U) || keys.isPressed(KEY_JUGAR_L)) {
        JUGAR = true;
        keys.pressedKeys[KEY_JUGAR_U] = false;
        keys.pressedKeys[KEY_JUGAR_L] = false;
    }
    if (PAUSA) {
        PAUSA = false;
        if (confirm("Seguir?") == true) {
            txt = "Si";
        } else {
            txt = "No";
            JUGAR = false;
            inicializa_parametros();
        }
    }
}
```

Podemos observar que si se pulsa la tecla '**P**' (**PAUSA)** se muestra un cuadro de diálogo preguntando si se desea seguir o no. Si se elige seguir, la partida continua em el punto en el que se quedó. Si se desea cancelar, el juego queda parado y el switch **JUGAR** queda a false.

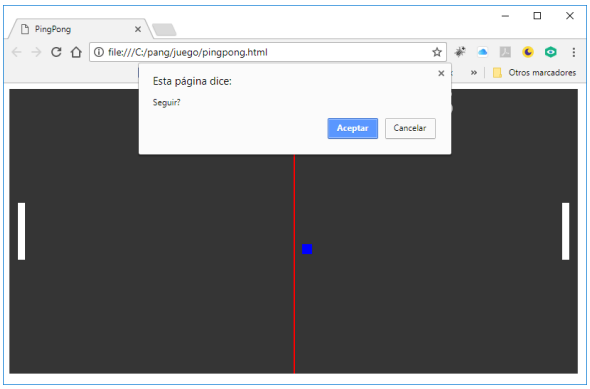
Ahora sólo nos falta condicionar el bucle para que calcule coordenadas y dibuje la pelota si el switch **JUGAR** está activo y añadir precisamente dicho switch a la función principal para que al cargar la página arranque inmediatamente el juego.

Las funciones **principal()** y **bucle()** quedan de la siguiente manera:

```
function principal() {
    inicializa_parametros();
    keys = new KeyListener();
    JUGAR = true;
    bucle();
}

function bucle() {
    dibuja_campo();
    if (JUGAR) {
        calcula_coordenadas_pelota();
        dibuja_pelota(x, y);
    }
    controlar_pulsacion();
    dibuja_jugadorI(jiX, jiY);
    dibuja_jugadorD(jdX, jdY);
    dibuja_puntuacion();
    setTimeout(bucle, 4);
}
```

Probemos nuestro juego refrescando de nuevo la página y pulsando la tecla '**P**' para provocar una pausa:



Si pulsa **Aceptar**, el juego continúa donde se pausó. Si elige **Cancelar**, el juego se detiene y queda a la espera de arrancar una nueva partida pulsando la tecla '**S**' (arranque).

HABILITAR/DESHABILITAR SONIDO

Vamos a habilitar y deshabilitar el sonido del juego mediante la pulsación de una tecla que alternará el estado de esta funcionalidad.

En los ejemplos anteriores, habíamos previsto un switch denominado **hay_sonido** para que fácilmente pudiéramos modificarlo activando o desactivando el sonido según nuestro deseo.

Podemos convenir que, al pulsar la tecla '**O**', se active/desactive el sonido mientras el juego se está llevando a cabo.

Para ello, definiremos un juego de teclas en nuestra función **inicializa_parametros()** de forma similar a como hicimos para la pausar y jugar anteriormente. A continuación, resaltamos en amarillo las sentencias a incluir justo después de las que copiamos a modo de guía. Para el grupo de teclas **CAR_** añadiremos:

```
    CAR_JUGAR_L = CAR_JUGAR_U.toLowerCase(); // Caracter para arrancar juego de
nuevo en minusculas

    CAR_HAY_SONIDO_U = "O"; // Caracter para poner/quitar sonido en mayusculas
    CAR_HAY_SONIDO_L = CAR_HAY_SONIDO_U.toLowerCase(); // Poner/quitar sonido en
minusculas

    // KeyCode (Movimiento paletas)
    KEY_I_ARRIBA_U = CAR_I_ARRIBA_U.charCodeAt(0); // mayusculas
```

Para obtener el carácter a controlar:

```
    KEY_JUGAR_L = CAR_JUGAR_L.charCodeAt(0); // Minusculas

    KEY_HAY_SONIDO_U = CAR_HAY_SONIDO_U.charCodeAt(0); // Mayusculas
    KEY_HAY_SONIDO_L = CAR_HAY_SONIDO_L.charCodeAt(0); // Minusculas

    // Inicializacion deteccion de pulsaciones para KeyCode
    PI_ARRIBA = false;
```

En la función **controlar_pulsacion()**, añadiremos lo siguiente resaltado en amarillo:

```
    keys.pressedKeys[KEY_JUGAR_L] = false;
    }
    if (keys.isPressed(KEY_HAY_SONIDO_U) ||
keys.isPressed(KEY_HAY_SONIDO_L)) {
        hay_sonido = !hay_sonido;
        console.log("hay_sonido " + hay_sonido);
        keys.pressedKeys[KEY_HAY_SONIDO_U] = false;
        keys.pressedKeys[KEY_HAY_SONIDO_L] = false;
    }
    if (PAUSA) {
```

Ya tenemos todo lo que pretendíamos y, por tanto, puede recargar la página para comprobar que cada vez que pulse la tecla '**O**', se activa o desactiva el sonido alternativamente.

RESUMEN DE CONTROLES

Tecla	Acción
A	Arriba paleta izquierda
Z	Abajo paleta izquierda
K	Arriba paleta derecha
M	Abajo paleta derecha
O	Pone/Quita sonido.
P	Pausa el juego
S	Arranca juego tras haberlo parado definitivamente.

FUENTES

PINGPONG.HTML

```
<!DOCTYPE HTML>
<html>

<head>
    <title>PingPong</title>
    <link rel="stylesheet" href="estilos.css" />
    <script defer src="pingpong.js"></script>
</head>

<body>
    <div>
        <canvas id="campo" width="800" height="400"></canvas>
    </div>
</body>

</html>
```

ESTILOS.CSS

```
#campo {
    background-color: #353535;
}
```

PINGPONG.JS

```
function principal() {
    inicializa_parametros();
    keys = new KeyListener();
    JUGAR = true;
    bucle();
}

function bucle() {
    dibuja_campo();
    if (JUGAR) {
        calcula_coordenadas_pelota();
        dibuja_pelota(x, y);
    }
    controlar_pulsacion();
    dibuja_jugadorI(jiX, jiY);
    dibuja_jugadorD(jdX, jdY);
    dibuja_puntuacion();
    setTimeout(bucle, 4);
}

function inicializa_parametros() {
    // Sonidos
    beep = new Audio("beep.wav");
    punto = new Audio("punto.wav");
    hay_sonido = false;

    // Canvas campo
    canvas = document.getElementById("campo");
    ancho_canvas = canvas.width;
    alto_canvas = canvas.height;
    context = canvas.getContext("2d");
    // Parametros pelota
    x = ancho_canvas / 2;
    y = alto_canvas / 2;
    ancho_pelota = 14;
    alto_pelota = 14;
    inc_pelota = 1; // Avance de la pelota en cada iteracion
    incX = inc_pelota;
    incY = inc_pelota;
    hay_inc_vel = false; // true para aumentar velocidad en cada rebote

    // Parametros Palas
    separacion = 12; // Separacion desde el extremo del campo
    alto_pala = alto_canvas / 5;
    ancho_pala = 10;
```

```
    inc_pala = 4; // Posiciones que avanza cuando se mueve
    // Posiciones de las paletas de jugadores (i)zq y (d)er.
    jiX = separacion;
    jiY = alto_canvas / 2 - alto_pala / 2;
    jdX = ancho_canvas - separacion - ancho_pala;
    jdY = jiY;

    // Caracteres que representan movimientos y controles
    CAR_I_ARRIBA_U = "A"; // Mover paleta izquierda arriba mayusculas
    CAR_I_ARRIBA_L = CAR_I_ARRIBA_U.toLowerCase(); // PI arriba
minusculas
    CAR_I_ABAJO_U = "Z"; // Mover paleta izquierda abajo mayusculas
    CAR_I_ABAJO_L = CAR_I_ABAJO_U.toLowerCase(); // PI abajo minusculas

    CAR_D_ARRIBA_U = "K"; // Mover paleta derecha arriba mayusculas
    CAR_D_ARRIBA_L = CAR_D_ARRIBA_U.toLowerCase(); // PD arriba
minusculas
    CAR_D_ABAJO_U = "M"; // Mover paleta derecha abajo mayusculas
    CAR_D_ABAJO_L = CAR_D_ABAJO_U.toLowerCase(); // PD abajo minusculas

    CAR_PAUSA_U = "P"; // Caracter para pausa en mayusculas
    CAR_PAUSA_L = CAR_PAUSA_U.toLowerCase(); // Caracter para pausa en
minusculas
    CAR_JUGAR_U = "S"; // Caracter para arrancar juego de nuevo en
mayusculas
    CAR_JUGAR_L = CAR_JUGAR_U.toLowerCase(); // Caracter para arrancar
juego de nuevo en minusculas

    CAR_HAY_SONIDO_U = "O"; // Caracter para poner/quitar sonido en
mayusculas
    CAR_HAY_SONIDO_L = CAR_HAY_SONIDO_U.toLowerCase(); // Poner/quitar
sonido en minusculas

    // KeyCode (Movimiento paletas)
    KEY_I_ARRIBA_U = CAR_I_ARRIBA_U.charCodeAt(0); // mayusculas
    KEY_I_ARRIBA_L = CAR_I_ARRIBA_L.charCodeAt(0); // minusculas
    KEY_I_ABAJO_U = CAR_I_ABAJO_U.charCodeAt(0); // Mayusculas
    KEY_I_ABAJO_L = CAR_I_ABAJO_L.charCodeAt(0); // minusculas

    KEY_D_ARRIBA_U = CAR_D_ARRIBA_U.charCodeAt(0); // Mayusculas
    KEY_D_ARRIBA_L = CAR_D_ARRIBA_L.charCodeAt(0); // Minusculas
    KEY_D_ABAJO_U = CAR_D_ABAJO_U.charCodeAt(0); // Mayusculas
    KEY_D_ABAJO_L = CAR_D_ABAJO_L.charCodeAt(0); // Minusculas

    KEY_PAUSA_U = CAR_PAUSA_U.charCodeAt(0); // Mayusculas
    KEY_PAUSA_L = CAR_PAUSA_L.charCodeAt(0); // Minusculas
    KEY_JUGAR_U = CAR_JUGAR_U.charCodeAt(0); // Mayusculas
    KEY_JUGAR_L = CAR_JUGAR_L.charCodeAt(0); // Minusculas
```

```
    KEY_HAY_SONIDO_U = CAR_HAY_SONIDO_U.charCodeAt(0); // Mayusculas
    KEY_HAY_SONIDO_L = CAR_HAY_SONIDO_L.charCodeAt(0); // Minusculas

    // Inicializacion deteccion de pulsaciones para KeyCode
    PI_ARRIBA = false;
    PI_ABAJO = false;
    PD_ARRIBA = false;
    PD_ABAJO = false;

    PAUSA = false;
    inicio_partida();
}

// Dibuja el campo
function dibuja_campo() {
    context.fillStyle = "red";
    context.clearRect(0, 0, ancho_canvas, alto_canvas);
    context.fillRect(ancho_canvas / 2, 0, 2, alto_canvas);
}
// Muestra la pelota en una posicion
function dibuja_pelota(x, y) {
    context.fillStyle = "blue";
    context.fillRect(x, y, ancho_pelota, alto_pelota);
}
// Muestra pala del jugador izquierdo
function dibuja_jugadorI(jiX, jiY) {
    context.fillStyle = "white";
    context.fillRect(jiX, jiY, ancho_pala, alto_pala);
}
// Muestra pala del jugador derecho
function dibuja_jugadorD(jdX, jdY) {
    context.fillStyle = "white";
    context.fillRect(jdX, jdY, ancho_pala, alto_pala);
}
// Muestra puntuacion cada vez que se produce un cambio
function dibuja_puntuacion() {
    context.fillStyle = "white";
    context.font = "48px Georgia";
    context.fillText(puntosI, ancho_canvas / 4, 30);
    context.fillText(puntosD, (ancho_canvas * 3) / 4, 30);
}

// Inicializa valores para el inicio de una partida
function inicio_partida() {
    puntosI = 0;
    puntosD = 0;
    inicio_punto();
}
// Inicializa valores para el inicio de un punto
```

```
function inicio_punto() {
    inicioY = Math.floor((Math.random() * alto_canvas / 2) + 1);
    x = ancho_canvas / 2; // Iniciamos desde el centro del campo
    y = inicioY; // La posicion vertical de la pelota es aleatoria

    // La direccion de la pelota en es aleatoria
    valorX = Math.floor((Math.random() * 100) + 1);
    if (valorX < 50) {
        incX = inc_pelota;
    } else {
        incX = -inc_pelota;
    }
    valorY = Math.floor((Math.random() * 100) + 1);
    if (valorY < 50) {
        incY = inc_pelota;
    } else {
        incY = -inc_pelota;
    }

    // Posiciones de las paletas de jugadores (i)zq y (d)er.
    jiX = separacion;
    jiY = alto_canvas / 2 - alto_pala / 2;
    jdX = ancho_canvas - separacion - ancho_pala;
    jdY = jiY;

}
// Calculo de coordenadas de la pelota
function calcula_coordenadas_pelota() {
    x += incX;
    y += incY;
    // Comprobacion de la pelota respecto de la pala izquierda
    if (y >= jiY && y <= jiY + alto_pala - 1) {
        if (x <= jiX + ancho_pala) {
            incX = -incX; // Cambio de direccion horizontal
            if (hay_inc_vel && Math.abs(incX) < 3) {
                if (incX < 1) {
                    incX -= 1
                } else {
                    incX += 1
                }
                if (incY < 1) {
                    incY -= 1
                } else {
                    incY += 1
                }
            }

            x = jiX + ancho_pala;
            if (hay_sonido) beep.play();
```

```
        }
    } else {
        // Si ha rebasado la posicion de la pala es un punto para el
        // jugador contrario.
        if (x < jiX - separacion) {
            inicio_punto();
            puntosD += 1;
            if (hay_sonido) punto.play();
        }
    }
    // Comprobacion de la pelota respecto de la pala derecha
    if (y >= jdY && y <= jdY + alto_pala - 1) {
        if (x + ancho_pelota >= jdX) {
            incX = -incX; // Cambio de direccion horizontal
            if (hay_inc_vel && Math.abs(incX) < 3) {
                if (incX < 1) {
                    incX -= 1
                } else {
                    incX += 1
                }
                if (incY < 1) {
                    incY -= 1
                } else {
                    incY += 1
                }
            }

            x = jdX - ancho_pelota;
            if (hay_sonido) beep.play();
        }
    } else {
        // Si ha rebasado la posicion de la pala es un punto para el
        // jugador contrario.
        if (x + ancho_pelota > jdX + separacion) {
            inicio_punto();
            puntosI += 1;
            if (hay_sonido) punto.play();
        }
    }
    // Si la pelota rebota en la parte superior o inferior de la
pantalla
    // cambia de direccion vertical.
    if (y + alto_pelota > alto_canvas || y < 0) {
        incY = -incY;
        if (hay_sonido) beep.play();
    }
}

function controlar_pulsacion() {
```

```
    if (keys.isPressed(KEY_I_ABAJO_U || KEY_I_ABAJO_L)) { // Abajo Izq
        PI_ABAJO = true;
    } else if (keys.isPressed(KEY_I_ARRIBA_U || KEY_I_ARRIBA_L)) { //
Arriba Izq
        PI_ARRIBA = true;
    }
    if (keys.isPressed(KEY_D_ABAJO_U || KEY_D_ABAJO_L)) { // Abajo Der
        PD_ABAJO = true;
    } else if (keys.isPressed(KEY_D_ARRIBA_U || KEY_D_ARRIBA_L)) { //
Arriba Der
        PD_ARRIBA = true;
    }
    calcula_coordenadas_pala();
    if (keys.isPressed(KEY_PAUSA_U) || keys.isPressed(KEY_PAUSA_L)) {
        PAUSA = true;
        keys.pressedKeys[KEY_PAUSA_U] = false;
        keys.pressedKeys[KEY_PAUSA_L] = false;
    }

    if (keys.isPressed(KEY_JUGAR_U) || keys.isPressed(KEY_JUGAR_L)) {
        JUGAR = true;
        keys.pressedKeys[KEY_JUGAR_U] = false;
        keys.pressedKeys[KEY_JUGAR_L] = false;
    }
    if (keys.isPressed(KEY_HAY_SONIDO_U) ||
keys.isPressed(KEY_HAY_SONIDO_L)) {
        hay_sonido = !hay_sonido;
        console.log("hay_sonido " + hay_sonido);
        keys.pressedKeys[KEY_HAY_SONIDO_U] = false;
        keys.pressedKeys[KEY_HAY_SONIDO_L] = false;
    }
    if (PAUSA) {
        PAUSA = false;
        if (confirm("Seguir?") == true) {
            txt = "Si";
        } else {
            txt = "No";
            JUGAR = false;
            inicializa_parametros();
        }
    }
}

function calcula_coordenadas_pala() {
    if (PI_ARRIBA == true) {
        jiY -= inc_pala;
        if (jiY < 0) jiY = 1;
        PI_ARRIBA = false;
    }
```

```
    if (PI_ABAJO == true) {
        jiY += inc_pala;
        if (jiY + alto_pala > alto_canvas) jiY = alto_canvas -
alto_pala;
        PI_ABAJO = false;
    }
    if (PD_ARRIBA == true) {
        jdY -= inc_pala;
        if (jdY < 0) jdY = 1;
        PD_ARRIBA = false;
    }
    if (PD_ABAJO == true) {
        jdY += inc_pala;
        if (jdY + alto_pala > alto_canvas) jdY = alto_canvas -
alto_pala;
        PD_ABAJO = false;
    }
}

// KEY LISTENER
function KeyListener() {
    this.pressedKeys = [];
    this.keydown = function (e) {
        this.pressedKeys[e.keyCode] = true;
    };
    this.keyup = function (e) {
        this.pressedKeys[e.keyCode] = false;
    };
    document.addEventListener("keydown", this.keydown.bind(this));
    document.addEventListener("keyup", this.keyup.bind(this));
}
KeyListener.prototype.isPressed = function (key) {
    return this.pressedKeys[key] ? true : false;
};
principal();
```