**INSTITUTO TECNOLÓGICO DE CHIHUAHUA II**

Ingeniería en sistemas computacionales.

Graficacion.

Alonso Salcido.

Documentación para PONG y PONG con uso de phaser js.

14550322 Cano Lozano Adelaido.

13550419 Ortiz Martínez Edith.

07/10/2017

Contenido

[Pong 3](#_Toc495180851)

[1. Crear Canvas y dibujar en él. 3](#_Toc495180852)

[2.Definir y dibujar la pelota. 4](#_Toc495180853)

[3. Rebotar en las paredes. 7](#_Toc495180854)

[4. Controles de paleta y teclado. 8](#_Toc495180855)

[5. Game Over. 11](#_Toc495180856)

[6. Construir el campo de ladrillo. 11](#_Toc495180857)

[7. Detección de colisiones. 13](#_Toc495180858)

[8. Registro de puntos. 16](#_Toc495180859)

[9. Fin del programa. 17](#_Toc495180860)

[Pong with phaser js 18](#_Toc495180861)

[Inicialización del framework. 18](#_Toc495180862)

[Escalada. 19](#_Toc495180863)

[Carga de elementos e impresión en pantalla. 20](#_Toc495180864)

[Mover la pelota. 20](#_Toc495180865)

[Física. 20](#_Toc495180866)

[Rebotar en las paredes. 21](#_Toc495180867)

[Paleta y controles de jugador. 21](#_Toc495180868)

[Game Over. 22](#_Toc495180869)

[Construir los ladrillos. 22](#_Toc495180870)

[Detectar colisiones. 24](#_Toc495180871)

[Puntaje y mostrar ganador. 25](#_Toc495180872)

# **Pong**

Hacer ejercicio y documentar de:

<https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_Breakout_game_pure_JavaScript>

## 1. Crear Canvas y dibujar en él.

Se define el título “PONG” del documento (nuestro juego) el cual se muestra en la barra de título del navegador o en las pestañas de página, mediante el uso de las etiquetas “<title> </title>”.

<title>PONG</title>

Indicamos la información de estilo mediante el uso de las etiquetas “<style> </style>”

<style>  
 \* { padding: 0; margin: 0; }  
 canvas { background: #eee; display: block; margin: 0 auto; }  
 </style>

La propiedad padding de CSS (Cascading Style Sheets/Hojas de estilo en cascada) establece el área de relleno en los cuatro lados de un elemento con un valor inicial de 0.

La propiedad margin de CSS establece el área de margen en los cuatro lados de un elemento con un valor inicial de 0.

La propiedad background de CSS permite ajustar todas las opciones de estilo de fondo disponibles a la vez, incluyendo: color de imagen, origen y tamaño.

La propiedad display de CSS especifica el tipo de renderizado utilizado para un elemento.

Aplica las propiedades padding y margin en general.

\* { }

Aplica las propiedades background, display, margin solo al área del canvas.

canvas { }

Utilizamos las etiquetas <canvas ></canvas> para dibujar gráficos y animaciones.

Este fragmento de código añade un elemento canvas al documento HTML.

<canvas id="myCanvas" width="480" height="320"></canvas>

El atributo global id define un identificador único "myCanvas" que debe ser único en todo el documento. Su propósito es identificar el elemento cuando se enlaza.

width define la anchura del espacio de coordenadas en píxeles CSS. El valor predeterminado es 320.

height define la altura del espacio de coordenadas en píxeles CSS. El valor predeterminado es 480.

Tomamos una referencia al canvas en JavaScript. Agregando el siguiente fragmento de código en el elemento de HTML <script> mediante el uso de las etiquetas <script> </script>.

<script> var canvas = document.getElementById("myCanvas");  
var ctx = canvas.getContext("2d"); </script>

document.getElementById() devuelve una referencia al elemento por su id.

Aquí almacenamos la referencia del elemento <canvas> a la variable canvas.

var canvas = document.getElementById("myCanvas");

Después creamos la variable ctx para almacenar las representaciones en contexto 2D (2 dimensiones).

## 2.Definir y dibujar la pelota.

Para mantener actualizado constantemente el canvas, dibujando en cada frame, necesitamos definir una función que se ejecute una y otra vez, con diferentes valores cada vez para cambiar las posiciones de nuestros elementos del canvas.

function draw() {  
 // drawing code  
}  
setInterval(draw, 10);

Para correr la función una y otra vez usamos la función de JavaScript setInterval(draw, 10); mandando llamar la función draw() cada 10 milisegundos.

Donde los añadimos los siguientes métodos que darán forma a nuestra pelota.

ctx.beginPath();  
ctx.arc(50, 50, ballRadius, 0, Math.PI\*2);  
ctx.fillStyle = "#0095DD";  
ctx.fill();  
ctx.closePath();

Donde:

ctx.beginPath(); inicia la ruta.  
ctx.arc(x, y, 10, 0, Math.PI\*2);Crea la figura de nuestro objeto, donde 50 y 50 establecen la posición inicial del objeto, 10 define el tamaño del radio, 0 establece el valor de incio del ángulo y para definir el tamaño del ángulo final usamos el cálculo Math.PI\*2, haciendo uso de la librería Math.  
ctx.fillStyle = "#0095DD"; Establece el color, el cual será empleado para rellenar los objetos de la ruta.  
ctx.fill(); rellena nuestro objeto.  
ctx.closePath(); indica el cierre de la ruta.

Para hacer que se mueva agregaremos dos variables de posición.

La variable x obtendrá el valor equivalente a la mitad del ancho de nuestro canvas como valor inicial.

La variable y obtendrá el valor de la altura de nuestro canvas y le restara 30 píxeles como valor inicial.

var x = canvas.width/2;  
var y = canvas.height-30;

Ahora actualizaremos los valores de nuestras posiciones ctx.arc(50, 50, 10, 0, Math.PI\*2); por el uso de variables para manejar más fácilmente la posición inicial de nuestra pelota ctx.arc(x, y, 10, 0, Math.PI\*2);

Para agregar el efecto de movimiento de nuestra pelota debemos agregar dos variables más las cuales modifican el valor de la posición de la pelota en cada frame.

var dx = 2;  
var dy = -2;

Para actualizar los valores de x y y debemos actualizar de la siguiente manera

x += dx;  
y += dy; Al añadir estas dos líneas de código a nuestra función estaremos obteniendo el valor actual de x y y , les estaremos agregando el valor actual de las mismas más el valor de dx y dy a sus respectivas variables.

function drawBall() {  
 ctx.beginPath();  
 ctx.arc(x, y, 10, 0, Math.PI\*2);  
 ctx.fillStyle = "#0095DD";  
 ctx.fill();  
 ctx.closePath();  
 x += dx;  
 y += dy;  
}

Para limpiar el canvas antes de cada frame debemos agregar a nuestra función

ctx.clearRect(0, 0, canvas.width, canvas.height); Esta línea evita que la bola deje un rastro de círculos pintados en cada frame.

function draw() {  
 ctx.clearRect(0, 0, canvas.width, canvas.height);  
 ctx.beginPath();  
 ctx.arc(x, y, 10, 0, Math.PI\*2);  
 ctx.fillStyle = "#0095DD";  
 ctx.fill();  
 ctx.closePath();  
 x += dx;  
 y += dy;  
}

Limpiaremos un poco el código para mantenerlo simple, en nuestra función draw añadiremos más comandos por ello lo mantendremos lo más reducido posible.

Moveremos el código de dibujo de la bola a una función separada llamada drawBall() de la siguiente manera: function drawBall() {  
 ctx.beginPath();  
 ctx.arc(x, y, 10, 0, Math.PI\*2);  
 ctx.fillStyle = "#0095DD";  
 ctx.fill();  
 ctx.closePath();  
}

simplificando así nuestra función draw(); y mandando llamar nuestra función encargada de pintar la pelota.

function draw() {  
 ctx.clearRect(0, 0, canvas.width, canvas.height);  
 drawBall();  
 x += dx;  
 y += dy;  
}

## 3. Rebotar en las paredes.

Para detectar la colisión verificamos si la bola está tocando la pared, y si es así, cambiamos la dirección de su movimiento.

Para hacer los cálculos más fáciles, definiremos una variable llamada ballRadius que mantendrá el radio del círculo dibujado y se usará para cálculos. var ballRadius = 10;

Actualizaremos la línea que dibuja la bola dentro de la función drawBall()

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

Hay cuatro paredes para rebotar la pelota, por ahora trabajaremos la parte superior e inferior.  
En cada frame estaremos checando si la pelota está tocando alguno de estos bordes del canvas, para cambiar la dirección de la bola hacia una dirección opuesta.

Para el borde superior, si el valor de y es menor que cero, cambia la dirección del movimiento.

if(y + dy < 0) {  
 dy = -dy;  
}

dy = -dy; Si la pelota se movía hacia arriba con una velocidad de 2 píxeles, ahora se moverá con una velocidad de -2 píxeles.  
Para el borde inferior, si el valor de y es mayor que la altura del lienzo.

if(y + dy > canvas.height) {  
 dy = -dy;  
}

Para reducir el código de estas dos instrucciones podemos aplicar las siguientes líneas, una vez que hemos conocido la lógica de los movimientos implicados.

if(y + dy > canvas.height || y + dy < 0) {  
 dy = -dy;  
}

Para asegurar el rebote de izquierda a derecha debemos podemos reciclar el código anterior cambiando la declaración de variable y por la declaración de x. En este caso tomaremos en cuenta el ancho del canvas en vez del alto canvas.width.

if(x + dx > canvas.width || x + dx < 0) {  
 dx = -dx;  
}

Para evitar el efecto de rebote con hundimiento podemos modificar el código tomando en cuenta la circunferencia de la pelota, en vez de el radio de la pelota.

if(x + dx > canvas.width-ballRadius || x + dx < ballRadius) {  
 dx = -dx;  
}  
if(y + dy > canvas.height-ballRadius || y + dy < ballRadius) {  
 dy = -dy;  
}

## 4. Controles de paleta y teclado.

Definiremos una paleta para golpear la bola, necesitamos las siguientes variables.

var paddleHeight = 10;  
var paddleWidth = 75;  
var paddleX = (canvas.width-paddleWidth)/2;

Donde:

paddleHeight es el alto y paddleWidth es el ancho, de la paleta. Para definir la posición de la paleta calculamos la posición de la siguiente manera (canvas.width-paddleWidth)/2 y la almacenamos en la variable paddleX.

Creamos la función que dibujara la paleta en la pantalla.

function drawPaddle() {  
 ctx.beginPath();  
 ctx.rect(paddleX, canvas.height-paddleHeight, paddleWidth, paddleHeight);  
 ctx.fillStyle = "#0095DD";  
 ctx.fill();  
 ctx.closePath();  
}

Para permitir al usuario interactuar con la paleta implementaremos algunos controles de teclado, para ello necesitaremos dos variables para almacenar información sobre si se presiona el botón de control izquierdo o derecho. Dos oyentes del evento keydown y keyup, los cuales ejecutaran el código para manejar el movimiento de la paleta cuando se presionan los botones.

Los botones presionados se definen e inicializan con variables booleanas. El valor predeterminado par ambos es false porque al principio los botones no están presionados.

var rightPressed = false;  
var leftPressed = false;

Para escuchar las pulsaciones de teclas, configuramos dos escuchas de eventos. Cuando una tecla sea presionada se ejecutará la función predeterminada para cada evento hasta que dejen de ser presionadas.

document.addEventListener("keydown", keyDownHandler, false);  
document.addEventListener("keyup", keyUpHandler, false);

Las funciones predeterminadas para cada evento son las siguientes.

function keyDownHandler(e) {  
 if(e.keyCode == 39) {  
 rightPressed = true;  
 }  
 else if(e.keyCode == 37) {  
 leftPressed = true;  
 }  
}  
  
function keyUpHandler(e) {  
 if(e.keyCode == 39) {  
 rightPressed = false;  
 }  
 else if(e.keyCode == 37) {  
 leftPressed = false;  
 }  
}

Cuando se presione una tecla la información se almacena en una variable (rightPressed / leftPressed) obteniendo el valor de true. Cuando se suelta la tecla, se vuelve a ajustar en false.

keyCode mantiene la información sobre la tecla que se presionó.

39 corresponde al cursor derecho y 37 corresponde al cursor izquierdo.

La lógica de desplazamiento de la paleta

if(rightPressed) {  
 paddleX += 7;  
}  
else if(leftPressed) {  
 paddleX -= 7;  
}

Si se pulsa el cursor izquierdo, la pala se moverá 7 píxeles a la izquierda, si se presiona el cursor derecho, la pala se moverá 7 píxeles hacia la derecha.

Para evitar que la paleta desaparezca del borde del lienzo si mantenemos la tecla durante demasiado tiempo, moveremos la paleta sólo dentro de los límites de la lona.

if(rightPressed && paddleX < canvas.width-paddleWidth) {  
 paddleX += 7;  
}  
else if(leftPressed && paddleX > 0) {  
 paddleX -= 7;  
}

La posición en paddleX se moverá entre 0 y el ancho del canvas.

En la función draw() mandamos llamar la función drawPaddle().

## 5. Game Over.

Golpear la pared inferior terminará el juego. Editaremos el segundo bloque si es un bloque de if else que activará nuestro estado de “superposición” cuando la pelota choque con el borde inferior del lienzo. Mostrando un mensaje de alerta y reiniciando el juego recargando la página.

if(y + dy < ballRadius) {  
 dy = -dy;  
} else if(y + dy > canvas.height-ballRadius) {  
 alert("GAME OVER");  
 document.location.reload();  
}

Para que nuestra pelota rebote con la paleta necesitamos detectar la colisión de la pelota con nuestra paleta.

if(y + dy < ballRadius) {  
 dy = -dy;  
} else if(y + dy > canvas.height-ballRadius) {  
 if(x > paddleX && x < paddleX + paddleWidth) {  
 dy = -dy;  
 }  
 else {  
 alert("GAME OVER");  
 document.location.reload();  
 }  
}

## 6. Construir el campo de ladrillo.

Para configurar las variables de ladrillo, definimos el número de filas y columnas de ladrillos, su ancho y alto, el relleno entre los ladrillos para que no se toquen entre sí y un desplazameinto superior e izquierdo para que no empiecen a ser dibujados desde el borde del canvas.

var brickRowCount = 3;  
var brickColumnCount = 5;  
var brickWidth = 75;  
var brickHeight = 20;  
var brickPadding = 10;  
var brickOffsetTop = 30;  
var brickOffsetLeft = 30;

Nuestros ladrillos estarán almacenados en un arreglo bidimensional. Usaremos “c” para referirnos a las columnas del ladrillo, y “r” para las filas. Estas contendrán un objeto que contendrá la posición x y y para pintar cada ladrillo en la pantalla.

var bricks = [];  
for(c=0; c<brickColumnCount; c++) {  
 bricks[c] = [];  
 for(r=0; r<brickRowCount; r++) {  
 bricks[c][r] = { x: 0, y: 0 };  
 }  
}

Necesitamos una función que pinte los ladrillos en la pantalla por lo que deberá recorrer la matriz.

function drawBricks() {  
 for(c=0; c<brickColumnCount; c++) {  
 for(r=0; r<brickRowCount; r++) {  
 bricks[c][r].x = 0;  
 bricks[c][r].y = 0;  
 ctx.beginPath();  
 ctx.rect(0, 0, brickWidth, brickHeight);  
 ctx.fillStyle = "#0095DD";  
 ctx.fill();  
 ctx.closePath();  
 }  
 }  
}

Hasta este punto estamos almacenando y pintando los ladrillos en pantalla, el problema es que los almacenamos y pintamos en la misma posición por lo tanto es necesario agregar dos variables más para establecer la posición de cada ladrillo para cada iteración.

var brickX = (c\*(brickWidth+brickPadding))+brickOffsetLeft;  
var brickY = (r\*(brickHeight+brickPadding))+brickOffsetTop;

Ahora cada ladrillo se puede colocar en su lugar correcto con relleno entre cada ladrillo, dibujado en un desplazamiento de la izquierda y del borde superior.

function drawBricks() {  
 for(c=0; c<brickColumnCount; c++) {  
 for(r=0; r<brickRowCount; r++) {  
 var brickX = (c\*(brickWidth+brickPadding))+brickOffsetLeft;  
 var brickY = (r\*(brickHeight+brickPadding))+brickOffsetTop;  
 bricks[c][r].x = brickX;  
 bricks[c][r].y = brickY;  
 ctx.beginPath();  
 ctx.rect(brickX, brickY, brickWidth, brickHeight);  
 ctx.fillStyle = "#0095DD";  
 ctx.fill();  
 ctx.closePath();  
 }  
 }  
}

Ahora solo debemos llamar la función drawBricks(); desde draw();.

## 7. Detección de colisiones.

Para detectar las colisiones debemos crear una función que recorrerá todos los ladrillos y comparará la posición de cada ladrillo con las coordenadas de la pelota a medida que se dibuja cada cuadro.

Para una mejor legibilidad vamos a definir una variable b, para almacenar el objeto ladrillo en cada bucle que detecte la colisión.

var b = bricks[c][r];

Creamos una función que detecta las colisiones que la pelota hará con los ladrillos, si la pelota está dentro de las coordenadas de algún ladrillo, la dirección de la pelota cambiará. Para esto se deben cumplir 4 condiciones.

* La posición de la pelota en x es mayor que la posición x del ladrillo.
* La posición x de la pelota es menor que la posición x del ladrillo más su ancho.
* La posición de la pelota en y es mayor que la posición y del ladrillo.
* La posición y de la pelota es menor que la posición y del ladrillo más su altura.

function collisionDetection() {  
 for(c=0; c<brickColumnCount; c++) {  
 for(r=0; r<brickRowCount; r++) {  
 var b = bricks[c][r];  
 if(x > b.x && x < b.x+brickWidth && y > b.y && y < b.y+brickHeight) {  
 dy = -dy;  
 }  
 }  
 }  
}

Para que los ladrillos desaparezcan después de ser golpeados por la pelota, debemos añadir un parámetro adicional para indicar si queremos pintar cada ladrillo en la pantalla o no. Esto se logra añadiendo una propiedad de status en nuestro objeto correspondiente al ladrillo.

var bricks = [];  
for(c=0; c<brickColumnCount; c++) {  
 bricks[c] = [];  
 for(r=0; r<brickRowCount; r++) {  
 bricks[c][r] = { x: 0, y: 0, status: 1 };  
 }  
}

Ahora se debe comprobar el valor de la propiedad status de cada ladrillo en la función drawBricks() antes de dibujarlo mediante la condicional if(bricks[c][r].status == 1)

function drawBricks() {  
 for(c=0; c<brickColumnCount; c++) {  
 for(r=0; r<brickRowCount; r++) {  
 if(bricks[c][r].status == 1) {  
 var brickX = (c\*(brickWidth+brickPadding))+brickOffsetLeft;  
 var brickY = (r\*(brickHeight+brickPadding))+brickOffsetTop;  
 bricks[c][r].x = brickX;  
 bricks[c][r].y = brickY;  
 ctx.beginPath();  
 ctx.rect(brickX, brickY, brickWidth, brickHeight);  
 ctx.fillStyle = "#0095DD";  
 ctx.fill();  
 ctx.closePath();  
 }  
 }  
 }  
}

Si ocurre una colisión, estableceremos el estado del ladrillo dado para 0 que no se pinte en la pantalla.

function collisionDetection() {  
 for(c=0; c<brickColumnCount; c++) {  
 for(r=0; r<brickRowCount; r++) {  
 var b = bricks[c][r];  
 if(b.status == 1) {  
 if(x > b.x && x < b.x+brickWidth && y > b.y && y < b.y+brickHeight) {  
 dy = -dy;  
 b.status = 0;  
 }  
 }  
 }  
 }  
}

Ahora mandamos llamar la función collisionDetection() desde nuestra función draw().

## 8. Registro de puntos.

Necesitamos una variable para registrar la puntuación. var score = 0;

También necesitamos una función drawScore() para crear y actualizar la visualización de la puntuación.

function drawScore() {  
 ctx.font = "16px Arial";  
 ctx.fillStyle = "#0095DD";  
 ctx.fillText("Score: "+score, 8, 20);  
}

Donde:

ctx.font = "16px Arial"; Establece el tamaño y tipo de fuente del texto.  
 ctx.fillStyle = "#0095DD"; Establece el color de la fuente.  
 ctx.fillText("Score: "+score, 8, 20);Establece el texto que se colocará en el canvas y la posición donde se colocará.

Para otorgar una puntuación cada vez que se golpea un ladrillo, incrementamos el valor de la variable de puntuación score++; .Esto se logra modificando la función collisionDetection()

function collisionDetection() {  
 for(c=0; c<brickColumnCount; c++) {  
 for(r=0; r<brickRowCount; r++) {  
 var b = bricks[c][r];  
 if(b.status == 1) {  
 if(x > b.x && x < b.x+brickWidth && y > b.y && y < b.y+brickHeight) {  
 dy = -dy;  
 b.status = 0;  
 score++;  
 }  
 }  
 }  
 }  
}

Ahora solo llamamos la función drawScore() desde la función draw().

## 9. Fin del programa.

Para mostrar un mensaje ganador cuando todos los ladrillos han sido destruidos debemos agregar una condición en nuestra función collisionDetection()

function collisionDetection() {  
 for(c=0; c<brickColumnCount; c++) {  
 for(r=0; r<brickRowCount; r++) {  
 var b = bricks[c][r];  
 if(b.status == 1) {  
 if(x > b.x && x < b.x+brickWidth && y > b.y && y < b.y+brickHeight) {  
 dy = -dy;  
 b.status = 0;  
 score++;  
 if(score == brickRowCount\*brickColumnCount) {  
 alert("YOU WIN, CONGRATULATIONS!");  
 document.location.reload();  
 }  
 }  
 }  
 }  
 }  
}

Si la pelota toca el borde inferior del canvas habrá terminado el juego mostrando un mensaje de alerta y reiniciando el juego al hacer cualquier otro evento.

# **Pong with phaser js**

Hacer ejercicio y documentar de:

<https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser>

## Inicialización del framework.

Necesitamos descargar el código fuente de phaser para aplicarlo en nuestro documento HTML.

1. Ingresamos en la página: <http://phaser.io/download/stable> .
2. Elegimos la opción min.js, para mantener el código fuente lo más reducido posible.
3. Guardamos el código Phaser dentro de un directorio /js directamente en la misma ubicación donde tengamos nuestro archivo indexP.html.

<!DOCTYPE html>  
<html>  
<head>  
 <meta charset="utf-8" />  
 <title>Gamedev Phaser Workshop - lesson 01: Initialize the framework</title>  
 <style>\* { padding: 0; margin: 0; }</style>  
 <script src="js/phaser.min.js"></script>  
</head>  
<body>  
<script>  
 var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {  
 preload: preload, create: create, update: update  
 });  
 function preload() {}  
 function create() {}  
 function update() {}  
</script>  
</body>  
</html>

Donde:

Establecemos la ruta de referencia de nuestro framework con nuestro documento HTML El elemento <canvas> es generado automáticamente por el framework..  
 <script src="js/phaser.min.js"></script>  
Creamos un nuevo objeto Phaser.Game y lo asignamos a la variable game.

Los parámetros son:

* Ancho 480 y alto 320 del elemento <canvas>.
* El método de representación AUTO dejando que Phaser decida cuál usar entre CANVAS y WEBGL.
* El id del elemento <canvas> se deja en null para que Phaser cree su propio id.
* Se establecen tres funciones de Phaser que cargan e inician el juego, además una función actualiza el bucle del juego en cada fotograma.
  + preload. Se encarga de precargar los activos.
  + create. Se ejecuta una vez cuando todo está cargado y listo.
  + update. Se ejecuta en cada trama.

<script>  
 var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {  
 preload: preload, create: create, update: update  
 });  
 function preload() {}  
 function create() {}  
 function update() {}  
</script>

## Escalada.

Para que nuestro canvas se dimensiona automáticamente independientemente del tamaño de pantalla.

El objeto de escalamiento de Phaser nos permite utilizar algunos métodos prácticos y propiedades.

function preload() {  
 game.scale.scaleMode = Phaser.ScaleManager.SHOW\_ALL;  
 game.scale.pageAlignHorizontally = true;  
 game.scale.pageAlignVertically = true;  
}

scaleMode tiene algunas opciones disponibles para escalar el canvas:

con SHOW\_ALL. escalamos el canvas, manteniendo intacta la relación de aspecto, por lo que las imágenes no serán sesgadas. En algunos casos habrá franjas negras en la pantalla.

Para mantener centrado nuestro canvas independientemente del tamaño de la pantalla. utilizamos las líneas.

game.scale.pageAlignHorizontally = true;  
 game.scale.pageAlignVertically = true;

Para darle color al fondo de nuestro canvas agregamos la línea

game.stage.backgroundColor = '#eee';

## Carga de elementos e impresión en pantalla.

Crearemos una variable JavaScript para representar una pelota. var ball;

Para cargar la pelota usaremos el objeto game creado por Phaser ejecutando el método load.image() dentro de la función preload()

function preload() {  
 // ...  
 game.load.image('ball', 'img/ball.png');  
}

Dónde ball es el nombre de nuestro objeto, y el segundo parámetro de nuestro método es la ruta relativa al elemento gráfico.

Para mostrarlo en la pantalla usaremos otro método de Phaser llamado add.sprite(). dentro de nuestra función create()

function create() {  
 ball = game.add.sprite(50, 50, 'ball');  
}

## Mover la pelota.

Para actualizar la posición de la pelota en pantalla añadimos las siguientes líneas de código en nuestra función update()

ball.x += 1;  
 ball.y += 1;

Las cuales obtienen el valor actual de las posiciones tanto en x como en y de nuestro objeto ball y le suman el valor actual más 1 píxel.

## Física.

Vamos a inicializar el motor de la física de la arcada en nuestro juego. utilizamos para ello el método physics.startSystem() al principio de la función create().

game.physics.startSystem(Phaser.Physics.ARCADE);

Para habilitar nuestra bola para el sistema de física. agregamos la siguiente línea en nuestra función create()

game.physics.enable(ball, Phaser.Physics.ARCADE);

Para mover nuestra pelota en la pantalla, podemos establecer el método velocity. Reemplazando nuestro antiguo modelo de animación de la pelota.

ball.body.velocity.set(150, 150);

## Rebotar en las paredes.

Utilizaremos la propiedad collideWorldsBound. Para indicar al canvas que queremos establecer sus bordes como límites del espacio de juego.

ball.body.collideWorldBounds = true;

Para habilitar el rebote de la bola necesitamos establecer la propiedad bounce, para que la pelota rebote al momento de tocar un borde de nuestro canvas.

ball.body.bounce.set(1);

## Paleta y controles de jugador.

Debemos añadir una variable para representar la paleta, cargar la imagen correspondiente y después configurar lo necesario.

Se crea la variable paddle var paddle;

En la función preload, cargamos la imagen de nuestra paleta load.image().

function preload() {  
 // ...  
 game.load.image('ball', 'img/ball.png');  
 game.load.image('paddle', 'img/paddle.png');  
}

Para renderizar la paleta, iniciamos nuestra paleta añadiendo la llamada add.sprite() dentro de la función create().

paddle = game.add.sprite(game.world.width\*0.5, game.world.height-5, 'paddle');

para asegurarnos de que la paleta este centrada siempre, debemos añadir paddle.anchor.set(0.5,1); justo debajo de la línea anterior.

Para hacer colisionar nuestra paleta con la pelota tenemos que habilitar la física para l a paleta.

game.physics.enable(paddle, Phaser.Physics.ARCADE);

Para habilitar la detección de colisiones entre la paleta y la pelota agregamos el método collide() dentro de la función update().

function update() {  
 game.physics.arcade.collide(ball, paddle);  
}

Los parámetros de este método son la pelota y la paleta. Habilitamos immovable para que la paleta no se pueda mover de manera extraña después de la colisión.

paddle.body.immovable = true;

Para controlar la paleta debemos establecer la posición de la paleta en donde está la posición de entrada.

function update() {  
 game.physics.arcade.collide(ball, paddle);  
 paddle.x = game.input.x;  
}

Ahora en cada frame, la posición en x de la paleta se ajustara a la posición en x de la entrada. Para fijar la posición por defecto en el centro de la pantalla usamos la siguiente línea.

paddle.x = game.input.x || game.world.width\*0.5;

Para colocar la pelota en la posición por defecto sobre nuestra paleta debemos añadir.

ball = game.add.sprite(game.world.width\*0.5, game.world.height-25, 'ball');  
ball.anchor.set(0.5);

Nuestra pelota empezará sobre nuestra paleta pero irá hacia abajo para ello solo debemos modificar la línea ball.body.velocity.set(150, -150); para que se dirija hacia arriba.

## Game Over.

Para que nuestro juego termine si la pelota toca el borde inferior, desactivaremos la colisión de la pelota con este borde.

game.physics.arcade.checkCollision.down = false;

Para que la pelota compruebe los límites del canvas y ejecute la función enlazada al evento onOutOfBounds usamos las siguientes líneas de código, justo debajo de la línea anterior.

ball.checkWorldBounds = true;  
ball.events.onOutOfBounds.add(function(){  
 alert('Game over!');  
 location.reload();  
}, this);

## Construir los ladrillos.

Definiremos las variables necesarias para nuestros ladrillos. bricks crea un grupo de ladrillos, newBrick un objeto añadido a cada grupo y brickInfo almacenará todos los datos del ladrillo.

var bricks;  
var newBrick;  
var brickInfo;

Cargamos la imagen del ladrillo con load.image()

function preload() {  
 // ...  
 game.load.image('brick', 'img/brick.png');  
}

Para dibujar los ladrillos creamos la función initBricks y la mandamos llamar dentro de la función create().

function create(){  
 // ...  
 initBricks();  
}

function initBricks() {  
 brickInfo = {  
 width: 50,  
 height: 20,  
 count: {  
 row: 7,  
 col: 3  
 },  
 offset: {  
 top: 50,  
 left: 60  
 },  
 padding: 10  
 };  
}

El objeto brickInfo contiene toda la información del ladrillo, ancho y altura, el número de filas y columnas, el desplazamiento superior e izquierdo y el relleno entre cada fila y columna.

Para crear los ladrillos en pantalla agregamos un nuevo grupo vació para contener los ladrillos. bricks = game.add.group();

Para recorrer las filas y columnas de nuestra matriz e ir añadiendo los ladrillos utilizaremos el siguiente ciclo.

for(c=0; c<brickInfo.count.col; c++) {  
 for(r=0; r<brickInfo.count.row; r++) {  
 // create new brick and add it to the group  
 }  
}

Para dibujar cada ladrillo dentro de la matriz necesitamos añadir la siguiente estructura.

for(c=0; c<brickInfo.count.col; c++) {  
 for(r=0; r<brickInfo.count.row; r++) {  
 var brickX = (r\*(brickInfo.width+brickInfo.padding))+brickInfo.offset.left;  
 var brickY = (c\*(brickInfo.height+brickInfo.padding))+brickInfo.offset.top;  
 newBrick = game.add.sprite(brickX, brickY, 'brick');  
 game.physics.enable(newBrick, Phaser.Physics.ARCADE);  
 newBrick.body.immovable = true;  
 newBrick.anchor.set(0.5);  
 bricks.add(newBrick);  
 }  
}

## Detectar colisiones.

Comprobamos dentro de la función update() la detección de colisiones entre bolas y ladrillos.

function update() {  
 game.physics.arcade.collide(ball, paddle);  
 game.physics.arcade.collide(ball, bricks, ballHitBrick);  
 paddle.x = game.input.x || game.world.width\*0.5;  
}

La posición de la pelota se calcula contra las posiciones de todos los ladrillos del grupo.

Dentro de la función ballHitBrick eliminamos el ladrillo en cuestión de la pantalla simplemente ejecutando el método kill().

function ballHitBrick(ball, brick) {  
 brick.kill();  
}

## Puntaje y mostrar ganador.

Añadiremos dos variables para guardar los puntos del jugador y una para mostrar en pantalla el puntaje.

var scoreText;  
var score = 0;

Para agregar la puntuación en pantalla.

scoreText = game.add.text(5, 5, 'Points: 0', { font: '18px Arial', fill: '#0095DD' });

Para actualizar los puntos que genere el jugador incrementaremos la variable score cada vez que se rompa un ladrillo y usaremos el método setText para mostrar el nuevo marcador en pantalla. Todo esto dentro de la función ballHitBrick().

function ballHitBrick(ball, brick) {  
 brick.kill();  
 score += 10;  
 scoreText.setText('Points: '+score);  
}

Para mostrar un mensaje ganador al usuario debemos agregar unas cuantas condiciones.

var count\_alive = 0;  
 for (i = 0; i < bricks.children.length; i++) {  
 if (bricks.children[i].alive == true) {  
 count\_alive++;  
 }  
 }  
 if (count\_alive == 0) {  
 alert('Ganaste o\_O, felicitaciones!');  
 location.reload();  
 }