|  |
| --- |
| Software de entretenimiento y Videojuegos Jordán Pascual : pascualjordan@uniovi.es |

|  |  |
| --- | --- |
| 4 | Plataformas - PARTE 3 |

# Niveles y bloque de fin de nivel

Vamos a comenzar añadiendo dos nuevas variables para el control de niveles en **Globales.js** .

|  |
| --- |
| **var *nivelActual*** = 0; **var *nivelMaximo*** = 2;  **var *estados*** = {}; |

Abrimos el **GameLayer.js y** modificamos el método **iniciar()**, queremos cargar el **nivelActual**, ya no cargaremos siempre el “0.txt” .

|  |
| --- |
| **this**.cargarMapa(**"res/"**+***nivelActual***+**".txt"**); |

Vamos a determinar que el símbolo “**C”** determina un elemento de fin de nivel (En este caso va a ser una copa, pero podría ser cualquier cosa, incluso algo invisible).

Si el jugador **colisiona** con “el elemento de fin de nivel” se pasar al siguiente nivel (siempre que no estemos ya en el **nivelMaximo**). Agregar la **C** en el mapa **0.txt**

|  |
| --- |
| ............................................. ............................................. ............................................. ...E......................................... .#####......C.........#...................... ..........####.....................###.#..... ............................................. #.1......#..E..#......#....................## #######..#################################### |

Agregamos un nuevo caso al switch del método **cargarObjetoMapa().** Al procesar una “C” creamos la variable **this.copa .** Este elemento va a ser tan simple que podemos usar una instancia de **Bloque**

Solo necesitamos que la **copa** se **dibuje** y que pueda **colisionar** con el jugador (al ser un modelo puede hacerse)**,** para que detecte **colisiones** el cuerpo debe añadirse como elemento dinámico al **espacio** físico, aunque como realmente se va a mover ni siquiera sería necesario agregarlo.

|  |
| --- |
| cargarObjetoMapa(simbolo, x, y){  **switch**(simbolo) {  **case "C"**:  **this**.**copa** = **new** Bloque(***imagenes***.**copa**, x,y);  **this**.**copa**.**y** = **this**.**copa**.**y** - **this**.**copa**.**alto**/2;  *// modificación para empezar a contar desde el suelo* **this**.**espacio**.agregarCuerpoDinamico(**this**.**copa**);  **break**;  **case "E"**:  **var** enemigo = **new** Enemigo(x,y); |

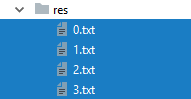
En el método **actualizar()**comprobamos las colisiones entre el **jugador** y la **copa**. Sí hay una colisión aumentamos la variable global **nivelActual** e iniciamos el juego de nuevo.

|  |
| --- |
| actualizar (){  **if** ( **this**.**copa**.colisiona(**this**.**jugador**)){  ***nivelActual***++;  **if** (***nivelActual*** > ***nivelMaximo***){  ***nivelActual*** = 0;  }  **this**.iniciar();  }    *// Jugador se cae* **if** ( **this**.**jugador**.**y** > 480 ){  **this**.iniciar();  } |

En el método **dibujar()** debemos invocar al **dibujar(ScroolX)** de la **copa.** Como es un elemento del mapa se debe especificar el **srcollX**.

|  |
| --- |
| dibujar (){  **this**.calcularScroll();  **this**.**fondo**.dibujar();  **for** (**var** i=0; i < **this**.**bloques**.length; i++){  **this**.**bloques**[i].dibujar(**this**.**scrollX**);  }  **for** (**var** i=0; i < **this**.**disparosJugador**.length; i++) {  **this**.**disparosJugador**[i].dibujar(**this**.**scrollX**);  }  **this**.**copa**.dibujar(**this**.**scrollX**);  **this**.**jugador**.dibujar(**this**.**scrollX**);  **for** (**var** i=0; i < **this**.**enemigos**.length; i++){  **this**.**enemigos**[i].dibujar(**this**.**scrollX**);  } |

Tal y como esta implementado el juego cada **nivel.txt** tiene que tener una C, sino **this.copa** será null y producirá un error. **Agregamos la C a los siguientes niveles.**

****

Probamos el juego, el nivel **0.txt** tiene una **C** para que puedan ser superados. (Aunque también podrían implementarse otras condiciones pasar al siguiente nivel).

# 

# Control por pulsaciones (Servirá para ratón y táctil)

Para agregar control por ratón/táctil debemos hacer dos cosas:

1. Incluir gráficamente botones en la pantalla que le indiquen al usuario como manejar el juego
2. Registrar eventos de ratón/táctil en la aplicación y comprobar si las pulsaciones caen dentro de los botones que hay en la pantalla. Sí las pulsaciones caen en esos botones se generarán **controles** para que sean procesados (hasta el momento generábamos **controles** solo con las teclas)

Accedemos a **Globales.js** y registramos las globales. En **pulsaciones** guardaremos todas las pulsaciones actuales que hay sobre la pantalla. Hay muchas formas distintas de implementar un control por pulsaciones (se pueden considerar muchos de sus eventos específicos**, iniciar pulsación, mover pulsación, levantar pulsación,** etc.)

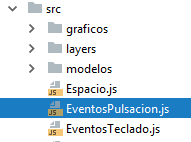
En este caso vamos a intentar emular un sistema de botones tradicional de esta forma no habrá diferencia en la lógica del juego independientemente del sistema de control a utilizar.

* **Inicio:** acabamos de pulsar algo por primera vez (según hacemos click)
* **Mantener:** estamos manteniendo el click en esa posición o moviéndolo

|  |
| --- |
| **var *pulsaciones*** = []; *// actuales registradas* **var *tipoPulsacion*** = {}; *// tipos* ***tipoPulsacion***.**inicio** = 1; ***tipoPulsacion***.**mantener** = 2;  **var *nivelActual*** = 0; **var *nivelMaximo*** = 2; |

|  |  |  |  |
| --- | --- | --- | --- |
| **Nota:** hay juegos que están totalmente concebidos para un control táctil. **Las pulsaciones interactuar directamente con los elementos del juego** (jugador, enemigos, etc.) **NO** con botones simulados en pantalla.   |  |  |  | | --- | --- | --- | |  |  |  | |

Creamos un fichero **src/EventosPulsacion.js.**

****

* Registramos 3 eventos de ratón: **mousedown**, **mousemove** y **mouseup**, cada uno llama a una función
* La función de **mousedown** y **mousemove** registran la información ‘’x’ ‘y’ de la pulsación y la agrega en la lista de **pulsaciones** . Como manejamos un único ratón con una única pulsación posible, todas las pulsaciones tiene la id = 1 (esto cambia cuando usamos pantallas táctiles y eventos touch).
* También registramos el **timeStamp**, se trata de una marca de tiempo de cuando se realizo la pulsación
* La función **mouseup** elimina la pulsación de la lista de **pulsaciones**

Para calcular la X y la X de una pulsación hay que tener en cuenta los posibles márgenes del canvas y el posible factor de escalado (**escaladoMinimo)**.

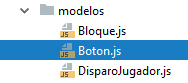
|  |
| --- |
| ***window***.addEventListener(**'mousedown'**, *mousedown*); ***window***.addEventListener(**'mousemove'**, *mousemove*); ***window***.addEventListener(**'mouseup'**, *mouseup*);  **function** *mousedown*( event) {  *agregarPulsacion*(1, ***tipoPulsacion***.**inicio**, event); }  **function** *mousemove* (event){  *agregarPulsacion*(1, ***tipoPulsacion***.***mantener***, event); }  **function** *mouseup*(event) {  *eliminarPulsacion*(1); }  **function** *agregarPulsacion*(id, tipoPulsacion, event ){  **x** = event.pageX - ***canvas***.offsetLeft;  **y** = event.pageY - ***canvas***.offsetTop;   **var** p = {};  p.**x** = **x** /***escaladoMinimo***;  p.**y** = **y** /***escaladoMinimo***;  p.**id** = id; *// Ratón SOLO hay 1* p.**tipo** = tipoPulsacion;  p.**timeStamp** = event.timeStamp;   **var** pulsacionEncontrada = **false**;  **for**(**var** i=0; i < ***pulsaciones***.**length**; i++){  **if** ( ***pulsaciones***[i].id == id){  ***pulsaciones***[i] = p;  pulsacionEncontrada = **true**;  }  }   **if** ( !pulsacionEncontrada ) {  ***pulsaciones***.push(p);  }  }  **function** *eliminarPulsacion*(id){  **for**(**var** i=0; i < ***pulsaciones***.**length**; i++){  **if** ( ***pulsaciones***[i].id == id){  ***pulsaciones***.splice( i, 1);  }  } } |

Agregamos el script **EventosPulsacion.js** al **index.html**

|  |
| --- |
| <**script src="src/Res.js"**></**script**> <**script src="src/EventosTeclado.js"**></**script**> <**script src="src/EventosPulsacion.js"**></**script**> <**script src="src/graficos/Animacion.js"**></**script**> <**script src="Main.js"**></**script**> |

**Representación gráfica de los controles**

Vamos a agregar unos Controles (**botones** y **pad**) al **GameLayer**. Comenzamos creando la clase **modelos/Boton.js**



Básicamente el **Botón** será un modelo/imagen que incluye un método **contienePunto(X,Y)** , procesa la X e Y que recibe y ve si cae dentro del área del botón para ver si lo han pulsado,

|  |
| --- |
| **class** Boton **extends** Modelo {   constructor(rutaImagen, x, y) {  **super**(rutaImagen, x, y)  **this**.**pulsado** = **false**;  }   contienePunto(pX, pY){  **if** ( pY >= **this**.**y** - **this**.**alto**/2 &&  pY <= **this**.**y** + **this**.**alto**/2 &&  pX <= **this**.**x** + **this**.**ancho**/2 &&  pX >= **this**.**x** - **this**.**ancho**/2){  **return true**;  }  **return false**;  }  } |

Agregamos el script **Boton.js** al **index.html**

|  |
| --- |
| <**script src="src/modelos/Modelo.js"**></**script**> <**script src="src/modelos/Boton.js"**></**script**> <**script src="src/modelos/Bloque.js"**></**script**> |

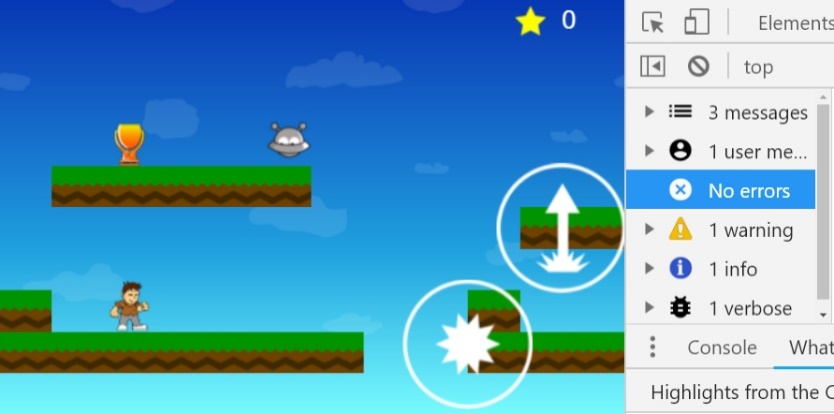
Accedemos al constructor del **GameLayer.js** e inicializamos un botón de salto y uno de disparo.

|  |
| --- |
| iniciar() {  *reproducirMusica*();  **this**.**botonSalto** = **new** Boton(***imagenes***.**boton\_salto**,480\*0.9,320\*0.55);  **this**.**botonDisparo** = **new** Boton(***imagenes***.**boton\_disparo**,480\*0.75,320\*0.83); |

Como todos los elementos del **GameLayer** deben ser dibujados en **dibujar().** (Lo mejor es dibujar los controles táctiles en último lugar para que siempre estén visibles).

|  |
| --- |
| dibujar (){  **this**.calcularScroll();  **this**.**fondo**.dibujar();  **for** (**var** i=0; i < **this**.**bloques**.length; i++){  **this**.**bloques**[i].dibujar(**this**.**scrollX**);  }  **for** (**var** i=0; i < **this**.**disparosJugador**.length; i++) {  **this**.**disparosJugador**[i].dibujar(**this**.**scrollX**);  }  **this**.**copa**.dibujar(**this**.**scrollX**);  **this**.**jugador**.dibujar(**this**.**scrollX**);  **for** (**var** i=0; i < **this**.**enemigos**.length; i++){  **this**.**enemigos**[i].dibujar(**this**.**scrollX**);  }  *// HUD* **this**.**fondoPuntos**.dibujar();  **this**.**puntos**.dibujar();  **this**.**botonDisparo**.dibujar();  **this**.**botonSalto**.dibujar(); } |

Aunque todavía no hemos implementado funcionalidad es buena idea probar el juego y ver que no tiene errores.



# Layers y calcularPulsaciones()

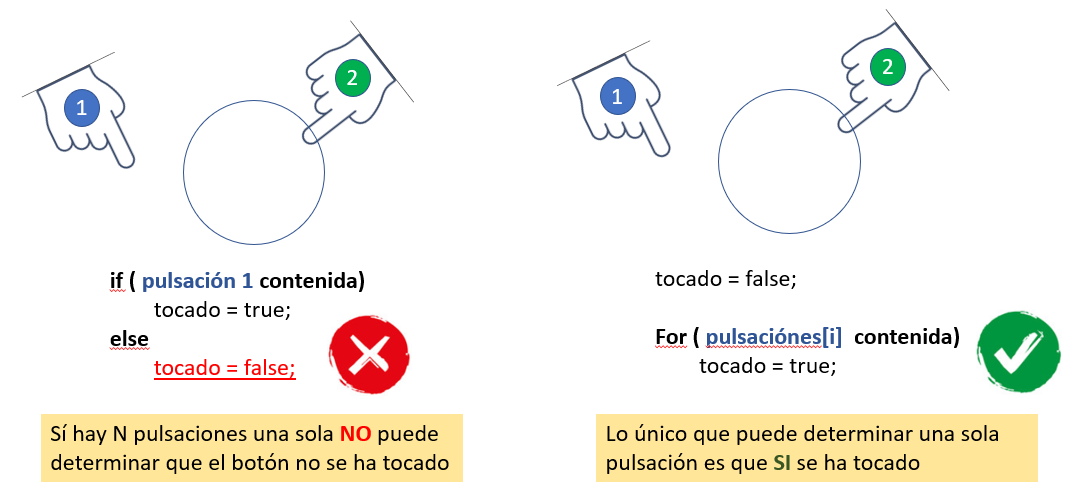
Cuando definimos la clase base **Layer.js** indicamos que las layers podían tener un método **calcularPulsaciones(),** este método estaba pensado para control táctil o mediante ratón.

Sobrescribimos el método **calcularPulsaciones(pulsaciones)** en el **GameLayer.js**

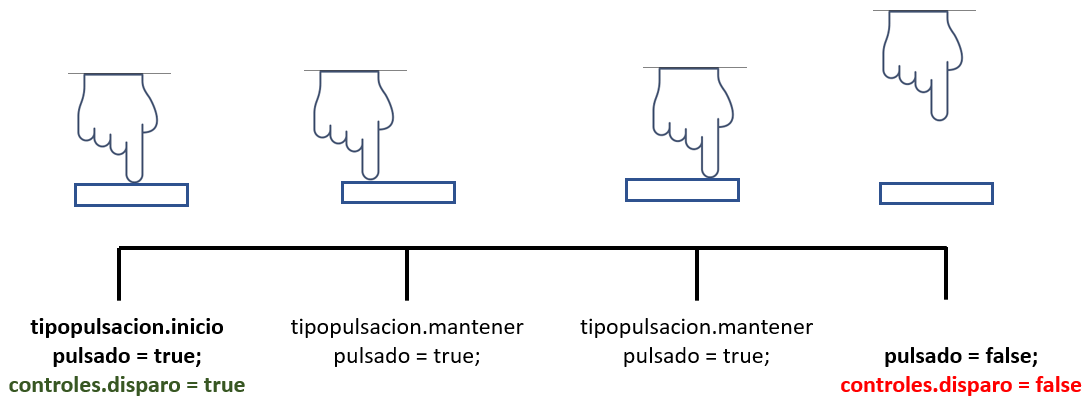
El método tiene que analizar si las pulsaciones que recibe caen en algún control (**botonDisparo** / **botonSalto**) y si es así generar los **controles** correspondientes (los mismos controles que generaban las teclas).

Los controles táctiles tienen una pequeña complejidad, NO podemos saber si están pulsados hasta que no analizamos toda la lista de pulsaciones. Para procesarlos:

1. Partimos de la suposición de que nadie los ha pulsado **botonDisparo.pulsado = false,**
2. Recorremos todas las **pulsaciones**, con que una sola este dentro del botón cambiamos el estado **botonDisparo.pulsado = true;**



1. Como queremos asemejarlo a un botón físico debemos comprobar que la pulsación es de **tipoPulsacion.inicio** (Es decir han empezado a tocar el botón) . Las pulsaciones de **inicio**, cambian el control correspondiente en el juego, las que lo mantienen “simplemente” pulsado no hacen nada y cuando se levanta el control vuelve a su estado

****

Una vez analizadas todas las **pulsaciones**, se comprueba que botones **NO** están pulsados.

En muchos juegos que los botones NO estén pulsados implica cambios en los controles, **por ejemplo,** cuando el botón de disparar NO esta pulsado se desactiva el disparar.

Agregamos la implementación de **calcularPulsaciones** al **GameLayer.js**

|  |
| --- |
| calcularPulsaciones(pulsaciones){  *// Suponemos botones no estan pulsados* **this**.**botonDisparo**.**pulsado** = **false**;  **this**.**botonSalto**.**pulsado** = **false**;   **for**(**var** i=0; i < pulsaciones.**length**; i++){  **if** (**this**.**botonDisparo**.contienePunto(pulsaciones[i].**x** , pulsaciones[i].**y**) ){  **this**.**botonDisparo**.**pulsado** = **true**;  **if** ( pulsaciones[i].**tipo** == ***tipoPulsacion***.**inicio**) {  ***controles***.**disparo** = **true**;  }  }    **if** (**this**.**botonSalto**.contienePunto(pulsaciones[i].**x** , pulsaciones[i].**y**) ){  **this**.**botonSalto**.**pulsado** = **true**;  **if** ( pulsaciones[i].**tipo** == ***tipoPulsacion***.**inicio**) {  ***controles***.**moverY** = 1;  }  }    }   *// No pulsado - Boton Disparo* **if** ( !**this**.**botonDisparo**.**pulsado** ){  ***controles***.**disparo** = **false**;  }   *// No pulsado - Boton Salto* **if** ( !**this**.**botonSalto**.**pulsado** ){  ***controles***.**moverY** = 0;  } } |

|  |
| --- |
| **Recordatorio**: La implementación anterior es útil tanto para **botones de pulsación simple** como **botones que se puedan dejar pulsados**.  Si **NO** queremos que los botones se puedan dejar pulsados para repetir una acción lo que hacemos es “reiniciar” el valor del **control** una vez procesado en **procesarControles(),** del **GameLayer**  así tendrán que volver a pulsar el botón.  procesarControles( ){  **if** (***controles***.**continuar**){  ***controles***.**continuar** = **false**;  **this**.**pausa** = **false**;  }  *// disparar* **if** ( ***controles***.**disparo** ){  **var** nuevoDisparo = **this**.**jugador**.disparar();  **if** ( nuevoDisparo != **null** ) {  **this**.**espacio**.agregarCuerpoDinamico(nuevoDisparo);  **this**.**disparosJugador**.push(nuevoDisparo);  }  ***controles***.**disparo** = **false**; *// NO PONER ESTA LINEA*  } |

**Importante**: En cada iteración del **loop(),** fichero **Main.js** habrá que invocar también el método **calcularPulsaciones(pulsaciones)** para que los controles táctiles sean procesados.

|  |
| --- |
| **function** *loop*(){  ***gameLayer***.actualizar();  ***gameLayer***.calcularPulsaciones(***pulsaciones***);  ***gameLayer***.procesarControles();  ***gameLayer***.dibujar(); } |

Abrimos el juego y comprobamos que salta y dispara. (El navegador **NO** puede estar en modo móvil ya que generaría eventos **Touch** en lugar de **Click**)

|  |  |
| --- | --- |
|  |  |

**Problema, registro de eventos muy rápido un mouseDown puede pasar a mouseUp 1ms después, antes de que se ejecute realmente un FP del juego (por lo tanto el mouseDown ni se llega a procesar).**  
Esto hace que la pulsación pase de **tipopulsacion.inicio** a **tipopulsacion.mantener** tan rápido que la lógica del juego ni siquiera pueda procesar la **tipopulsacion.inicio** en algunas iteraciones.

Solo guardamos una pulsación “move”/**tipoPulsacion.mantener** del ratón:

* Sí NO había ninguna pulsación anterior que pisar **pulsaciones.length == 0**
* Sí había una anterior que pisar y han pasado 10ms desde que se agregó. Usamos el atributo **timeStamp**.

|  |
| --- |
| **function** *mousemove* (event){  *~~agregarPulsacion~~*~~(1,~~ ***~~tipoPulsacion~~***~~.~~**~~mantener~~**~~, event);~~  *// Si no hay ninguna, o si han pasado más de 10ms desde la anterior* **if** (***pulsaciones***.**length** == 0 ||  event.timeStamp - ***pulsaciones***[0].timeStamp > 10 ) {    *agregarPulsacion*(1, ***tipoPulsacion***.**mantener**, event);  } } |

**Problema paso de pulsación “inicio” a “mantener”**  
  
Las pantallas táctiles solo generan eventos de mover cuando el dedo cambia de posición, si se mantiene en la misma posición no genera ningún evento. La pulsación de **tipopulsacion.inicio** no pasará por si sola a **tipopulsacion.mantener** si el dedo no se mueve.

Al final de cada iteración del **loop()** habría que llamar a **actualizarPulsaciones();** esta función tiene que indicar que las pulsaciones de **tipoPulsacion.inicio** ya no lo son más una vez han sido procesadas. (podemos incluir la función **actualizarPulsaciones()** en **Main.js** o **EventosPulsacion.js** )

|  |
| --- |
| **function** *loop*(){  ***gameLayer***.actualizar();  ***gameLayer***.calcularPulsaciones(***pulsaciones***);  ***gameLayer***.procesarControles();  ***gameLayer***.dibujar();    actualizarPulsaciones(); }  **function** *actualizarPulsaciones* () {  **for**(**var** i=0; i < ***pulsaciones***.length; i++){  **if** ( ***pulsaciones***[i].**tipo** == ***tipoPulsacion***.**inicio**){  ***pulsaciones***[i].**tipo** = ***tipoPulsacion***.**mantener**;  }  } } |

**Pad para moverse en el eje X.**

De forma muy similar a los anteriores podríamos agregar dos botones uno para la derecha y otro para la izquierda

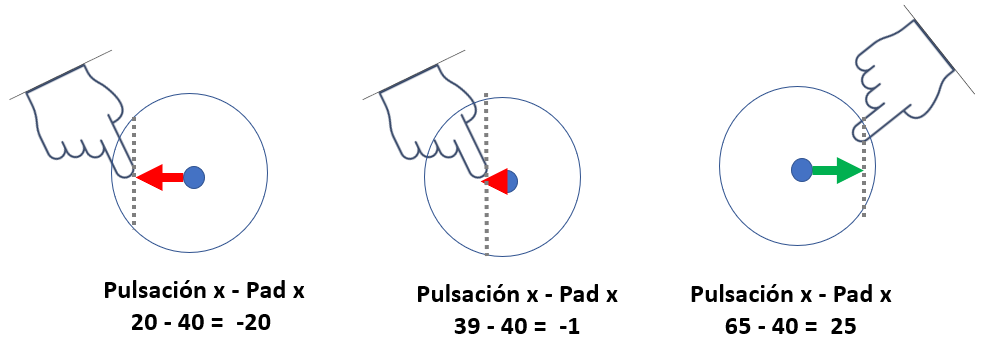
En lugar de dos botones vamos a crear un **Pad** de dirección. Creamos el fichero **modelos/Pad.js**



Básicamente un **Pad** es aún más sencillo que un botón porque registra valores (**controles.moverX**) incluso cuando no lo están pulsando, un joystick no pulsado envía un 0, esto nos evitará tener que analizar las pulsaciones.

En lugar de permitir especificar la imagen en el constructor la ponemos fija: **imágenes.pad** . Además de saber si el **pad** contiene un punto debe poder dar su **orientacionX** y su **orientacionY** (en este juego solo usaremos la **orientacionX**).

Para obtener la **orientación** de la pulsación en un eje solo hay que hacer una resta.



Las orientaciones son muy útiles para crear movimientos con diferentes velocidades, si pulsan el pad en un extremo podrían querer moverse rápido, si lo pulsan cerca del centro despacio.

|  |
| --- |
| **class** Pad **extends** Boton {   constructor(x, y) {  **super**(***imagenes***.**pad**, x, y)  }   obtenerOrientacionX(pX){  **return** pX - **this**.**x**;  }   obtenerOrientacionY(pY){  **return** pY - **this**.**y**;  }  } |

Agregamos en **index.html** el **Pad.js** (a continuación del **Botón**)

|  |
| --- |
| <**script src="src/modelos/Modelo.js"**></**script**> <**script src="src/modelos/Boton.js"**></**script**> <**script src="src/modelos/Pad.js"**></**script**> |

Accedemos al **GameLayer.js y** agregamos la variable **Pad** en el constructor.

|  |
| --- |
| iniciar() {  *reproducirMusica*();  **this**.**botonSalto** = **new** Boton(***imagenes***.**boton\_salto**,480\*0.9,320\*0.55);  **this**.**botonDisparo** = **new** Boton(***imagenes***.**boton\_disparo**,480\*0.75,320\*0.83);  **this**.**pad** = **new** Pad(480\*0.14,320\*0.8); |

También debemos recordar dibujarlo, al final del método **dibujar().**

|  |
| --- |
| **this**.**botonDisparo**.dibujar();  **this**.**botonSalto**.dibujar();  **this**.**pad**.dibujar(); } |

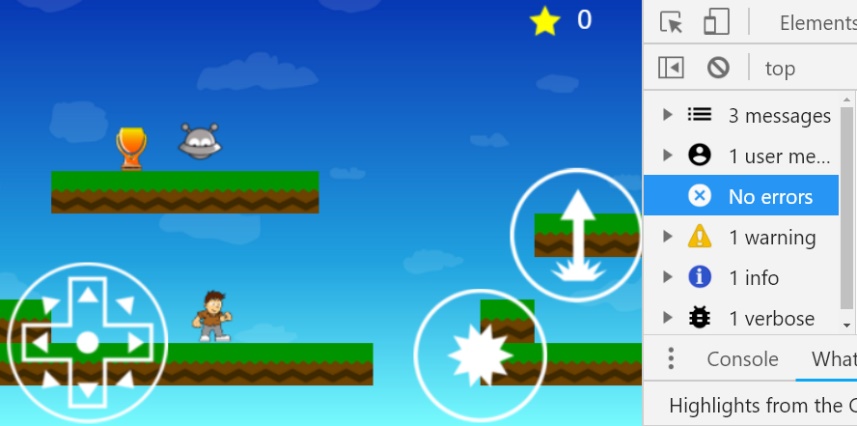
Finalmente modificamos el **calcularPosiciones()**

1. Vemos si la pulsación cae en el Pad, **contienePunto()**
2. Sí cae en el pad obtenemos la **orientación del eje X**
3. En función de la orientación del eje X colocamos un valor u otro en **controles.moverX**

Es mejor que rechacemos los valores muy pequeños de la **orientacionX** del Pad (Ej: entre -20 y 20) por ser muy próximos a 0, esto indica que el usuario tiene el ratón casi en el centro del pad, probablemente no quiera mover el jugador.

|  |
| --- |
| calcularPulsaciones(pulsaciones){  *// Suponemos botones no estan pulsados* **this**.**botonDisparo**.**pulsado** = **false**;  **this**.**botonSalto**.**pulsado** = **false**;  *// suponemos que el pad está sin tocar*  ***controles***.**moverX** = 0;    **for**(**var** i=0; i < pulsaciones.**length**; i++){  **if** (**this**.**pad**.contienePunto(pulsaciones[i].**x** , pulsaciones[i].**y**) ){  **var** orientacionX = **this**.**pad**.obtenerOrientacionX(pulsaciones[i].**x**);  **if** ( orientacionX > 20) { *// de 0 a 20 no contabilizamos* ***controles***.**moverX** = orientacionX;  }  **if** ( orientacionX < -20) { *// de -20 a 0 no contabilizamos* ***controles***.**moverX** = orientacionX;  }  }   **if** (**this**.**botonDisparo**.contienePunto(pulsaciones[i].**x** , pulsaciones[i].**y**) ){ |

Abrimos el juego y comprobamos que se mueve de forma correcta. (No hace falta hacer click, según lo hemos programado la pulsación no tiene que ser **tipopulsacion.inicio** basta con mover el ratón por encima).



**PROBLEMA TELCAS Y CLICKS SIMULTANEOS:** Sí intentamos manejar el juego a la vez con teclas y clicks va a funcionar mal. Esto no quiere decir que un juego no se pueda manejar simultaneamete con teclas y clicks, pero en este caso estamos usando las teclas y los clicks para hacer exactamente lo mismo.

|  |
| --- |
| **Nota:** Hay juegos muy orientados a control táctil, en los que **calcularPulsaciones(p),** comprobaría también si las pulsaciones caen sobre algún elemento del mapa (jugadores, enemigos, etc.), **calcularPulsaciones()** también podría agregar nuevos elementos al juego, Ej  Resultado de imagen de android defense game |

# Soporte múltiples entradas (Teclado, pantalla, GamePad…)

Vamos a implementar un sistema para que se reconozca el tipo de entrada que el usuario quiere utilizar y se habilite únicamente esa entrada.

Podríamos utilizar un menú para ello, pero vamos a optar por un sistema más “transparente”, en el momento que se detecta un evento especifico (pulsación/tecla) se cambia el tipo entrada.

Accedemos a **Globale.js** y declaramos las posibles entradas y la **entrada** actual.

|  |
| --- |
| **var *pulsaciones*** = []; *// actuales registradas* **var *entradas*** = {}; *// tipos* ***entradas***.**pulsaciones** = 1; ***entradas***.**teclado** = 2; ***entradas***.**gamepad** = 3; **var *entrada*** = ***entradas***.**pulsaciones**; |

Accedemos al **Main.js** e indicamos que el **cacularPulsaciones(pulsaciones)** solo se ejecuta cuando **entradas == entradas.pulsaciones**

|  |
| --- |
| **function** *loop*(){  ***gameLayer***.actualizar();  **if** (***entrada*** == ***entradas***.**pulsaciones**) {  ***gameLayer***.calcularPulsaciones(***pulsaciones***);  }  ***gameLayer***.procesarControles();  ***gameLayer***.dibujar();   *actualizarPulsaciones*(); } |

Ahora vamos modificando el valor de **entrada** en función de los eventos detectados.

En **EventosTeclado**.**js** si detectamos un **onKeyDown** la entrada pasará a ser de tipo **entrada.teclado.**

|  |
| --- |
| **function** *onKeyDown*( event) {  ***entrada*** = ***entradas***.**teclado**;  *// agregar la tecla pulsada si no estaba* **var** posicion = ***teclas***.indexOf(event.keyCode); |

En **EventosPulsacion.js** si agregamos una pulsación en **agregarPulsacion()** el tipo de entrada pasará a ser **entrada.pulsaciones.**

|  |
| --- |
| **function** *agregarPulsacion*(id, tipoPulsacion, event ){  ***entrada*** = ***entradas***.**pulsaciones**;   **x** = event.pageX - ***canvas***.offsetLeft;  **y** = event.pageY - ***canvas***.offsetTop; |

# Eventos Táctiles

Realmente no tiene mucho sentido que alguien maneje este juego (aunque quizá otros sí) con un ratón, el modo de control de los botones en pantalla está pensado para un sistema con múltiples pulsaciones simultaneas).

Una vez implementado el control por clicks reconocer eventos táctiles compartirá gran parte de la lógica.

Accedemos a **EventosPulsaciones.js** y añadimos los escuchadores de los eventos Touch.

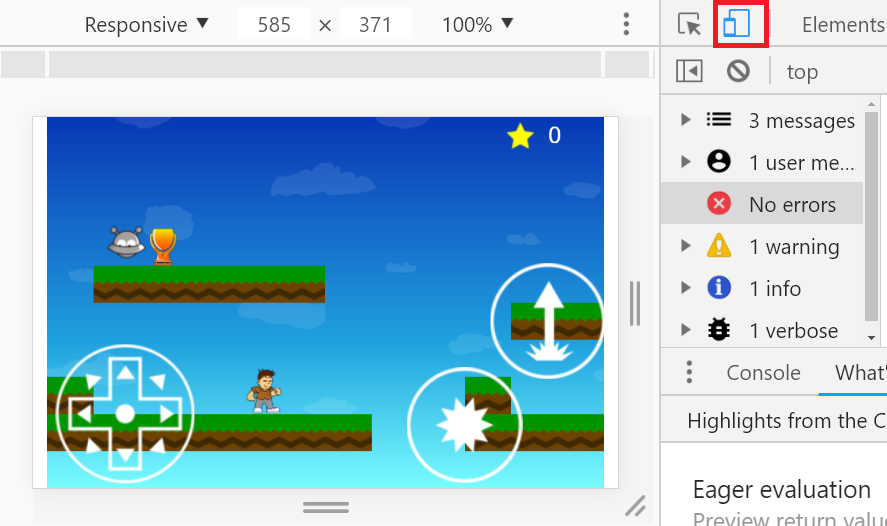
Las diferencias principales de los touch respecto los click son principalmente 2:

1. Hay varios eventos distintos para determinar que una pulsación acabo: **touchend, touchcancel, touchleave** (cada uno tiene sus particularidades, pero para nosotros no son relevantes)
2. Múltiples pulsaciones simultaneas: El evento que guarda los datos de la pulsación no contiene solo una pulsación sino uno array con varias pulsaciones (todos los dedos que hay en la pantalla), este array se obtiene con **evento.changedTouches,** y cada pulsación tiene una id en su atributo **identifier** (cada dedo tiene un identifier)

Cada vez que se detectan touches iteramos la lista y vamos procesando cada uno de los toques con las funciones anteriormente implementada: **agregarPulsacion** y **eliminarPulsacion**

|  |
| --- |
| ***window***.addEventListener(**'touchstart'**, *touchstart*, **false**); ***window***.addEventListener(**'touchmove'**, *touchmove*, **false**);  ***window***.addEventListener(**'touchend'**, *eliminarTouch*, **false**); ***window***.addEventListener(**'touchcancel'**, *eliminarTouch*, **false**); ***window***.addEventListener(**'touchleave'**, *eliminarTouch*, **false**);  **function** *touchstart*( event) {  **var** touches = event.changedTouches;  **for** (**var** i=0; i<touches.length; i++) {  *agregarPulsacion*( touches[i].identifier,  ***tipoPulsacion***.**inicio**, touches[i]);  } }  **function** *touchmove*( event) {  **var** touches = event.changedTouches;  **for** (**var** i=0; i<touches.length; i++) {  *agregarPulsacion*( touches[i].identifier,  ***tipoPulsacion***.**mantener**, touches[i]);  } }  **function** *eliminarTouch* ( event) {  **var** touches = event.changedTouches;  **for** (**var** i=0; i<touches.length; i++) {  *eliminarPulsacion*( touches[i].identifier);  } }  ***window***.addEventListener(**'mousedown'**, *mousedown*); ***window***.addEventListener(**'mousemove'**, *mousemove*); ***window***.addEventListener(**'mouseup'**, *mouseup*); |

Para probar el juego podemos usar la opción de **modo móvil** del Chrome.



La forma completa de probarlo sería subirlo a un servidor web y acceder desde el móvil a la página. Aunque no se requiere en la asignatura hay un documento adicional que explica paso a paso como publicar el juego en un hosting comercial (gratuito).

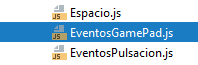
También puede quedar bien dibujar los botones táctiles solo si el juego NO esta pausado y la entrada es **entradas.pulsaciones**.

|  |
| --- |
| dibujar (){  **this**.calcularScroll();   **this**.**fondo**.dibujar();  **for** (**var** i=0; i < **this**.**bloques**.length; i++){  **this**.**bloques**[i].dibujar(**this**.**scrollX**);  }    **for** (**var** i=0; i < **this**.**disparosJugador**.length; i++) {  **this**.**disparosJugador**[i].dibujar(**this**.**scrollX**);  }   **this**.**copa**.dibujar(**this**.**scrollX**);  **this**.**jugador**.dibujar(**this**.**scrollX**);  **for** (**var** i=0; i < **this**.**enemigos**.length; i++){  **this**.**enemigos**[i].dibujar(**this**.**scrollX**);  }   *// HUD* **this**.**fondoPuntos**.dibujar();  **this**.**puntos**.dibujar();  **if** ( !**this**.pausa && **entrada** == ***entradas***.**pulsaciones**) {  **this**.**botonDisparo**.dibujar();  **this**.**botonSalto**.dibujar();  **this**.**pad**.dibujar();  } } |

# \* Gamepad (Para quien tenga un GamePad y lo quiera probar)

Sí disponemos de un GamePad compatible con ordenador podemos utilizarlo también como mecanismo de control (El siguiente ejemplo se ha realizado un mando de XboxOne [enlace](https://es.aliexpress.com/store/product/PC-USB-Wired-Controller-Joystick-Controle-for-Microsoft-Xbox-One-and-PC-Wired-Gamepad-for-Xbox/2174025_32901316344.html?spm=a219c.search0104.3.22.72f7691aqOi6Ld&ws_ab_test=searchweb0_0,searchweb201602_4_10065_10068_10547_10059_10548_10696_100031_10084_10083_10103_451_10618_452_10307_10820_10301_10821_10303_5014715_5014615,searchweb201603_54,ppcSwitch_2&algo_expid=842dbaeb-0ece-436d-9a19-b5fae760fd2c-3&algo_pvid=842dbaeb-0ece-436d-9a19-b5fae760fd2c&transAbTest=ae803_2&priceBeautifyAB=0) estos mandos se integran muy bien con Windows), aunque nos servirían muchos otros.

Se debe crear un nuevo fichero **src/EventosGamePad.js**

****

El evento **gamepadconnected** detecta cuando conectamos un pad, en ese momento sacamos unos mensajes por consola.

El GamePad funciona muy diferente a los mecanismos de entrada vistos anteriormente. Debemos usar **setInterval** para indicar una función que lea los ejes (joysticks) y botones del mando cada pocos milisegundos (nosotros elegimos el tiempo de refresco).

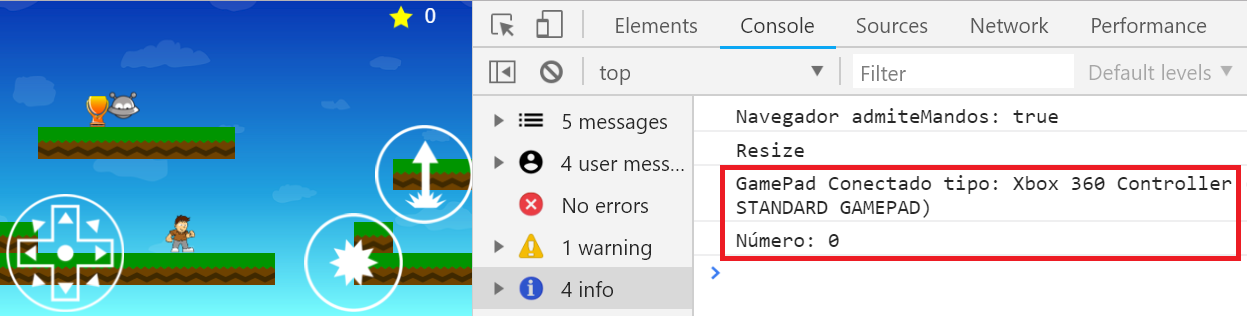
La función **actualizarOrdenes()** , va a obtener el **GamePad 0** (se podrían conectar varios 0, 1, 2, etc.),

1. Tipo de entrada: Mirará si alguno de sus botones está pulsado, si hay uno pulsado el tipo de control **entrada** pasará a **entradas.gamepad** . Si el tipo entrada no fuera **entradas.gamepad** ni siquiera continuamos ejecutando el método, hacemos un **return**.
2. Joysticks: Para obtener la x del joystick izquierdo usamos **axes[0],** la esquina izquierda da valor -1, el centro 0 y la derecha 1.
3. Botones: Los botones se organizan en un array **buttons[]** , con una variable **pressed**. Mientras un botón esta pulsado devuelve **pressed=true**; Nosotros solo vamos a usar 2 botones el 0 y el 2  
     
   Para poder diferenciar mejor se trata de una pulsación “simple” o” mantenida” vamos a crear un array **botonesPulsados.** Usando el array **botonesPulsados** podemos saber si el botón ya estaba pulsado (por lo tanto lo están “manteniendo”) y eso no se materializaba en el cambio del valor de un control (solo estábamos cambiando los **controles** en la pulsación inicial del botón y cuando dejábamos de pulsarlo).

|  |
| --- |
| ***console***.log(**"Navegador admiteMandos: "**+*admiteMandos*());  *// Los que hay pulsados actualmente* **var *botonesPulsados*** = [];  **function** *admiteMandos*() {  **return "getGamepads" in *navigator***; }  ***window***.addEventListener(**"gamepadconnected"**, **function**(e) {  ***console***.log(**"GamePad Conectado tipo: "**+e.gamepad.id);  *// Se podrían conectar varios, multijugador* ***console***.log(**"Número: "**+e.gamepad.index);   *// Leer botones del mando* **if** ( ***navigator***.getGamepads()[0] != **null** ){  *setInterval*(*actualizarOrdenes*,1000/30);  }  });  **function** *actualizarOrdenes*(){  *// Obtener gamePad en cada iteración* **var** gP1 = ***navigator***.getGamepads()[0];   **for**(**var** i=0; i < gP1.buttons.length; i++){  **if** ( gP1.buttons[i].pressed ) {  ***entrada*** = ***entradas***.**gamepad**;  }  }  **if** ( **entrada** != ***entradas***.**gamepad**){  **return**;  }  *// Pad de la izquierda gP1.axes[0] - X , gP1.axes[1] - Y  // Los joystics van valores entre -1(izquierda) y 1(derecha)  // El centro del joystick es 0  // Float con 2 decimales* ***controles***.**moverX** = *parseFloat*(gP1.axes[0]).toFixed( 2 );   **if** ( gP1.buttons[0].pressed){ *// 1 es el botón B* **if** ( ***botonesPulsados***[0] == **false** ) {  ***botonesPulsados***[0] = **true**;  ***controles***.**moverY** = 1;  }  } **else** {  ***botonesPulsados***[0] = **false**;  ***controles***.**moverY** = 0;  }   **if** ( gP1.buttons[2].pressed){ *// 2 es el botón A* **if** ( ***botonesPulsados***[2] == **false** ) {  ***botonesPulsados***[2] = **true  *controles***.**disparo** = **true**;  }  } **else** {  ***botonesPulsados***[2] = **false  *controles***.**disparo** = **false**;  } } |

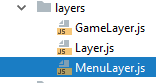
Se agrega el **EventosGamePad.js** al **index.html**

|  |
| --- |
| <**script src="src/EventosTeclado.js"**></**script**> <**script src="src/EventosPulsacion.js"**></**script**> <**script src="src/EventosGamePad.js"**></**script**> |

Abrimos el juego y conectamos un mando compatible (El ejemplo está configurado para un mando de XboxOne, aunque el driver no sé porque dice Xbox 360)  
  


# Layers – pantalla de menú

Creamos el fichero **layers/MenuLayer.js** en el vamos a definir una capa/pantalla de Menú.



|  |
| --- |
| **class** MenuLayer **extends** Layer {   constructor() {  **super**();  **this**.iniciar();  }   iniciar() {  **this**.**fondo** =  **new** Fondo(***imagenes***.**menu\_fondo**,480\*0.5,320\*0.5);  **this**.**boton** =  **new** Boton(***imagenes***.**boton\_jugar**,480\*0.5,320\*0.7);  }   dibujar (){  **this**.**fondo**.dibujar();  **this**.**boton**.dibujar();  }  } |

Agregamos el **MenuLayer.js** al index.html justo después del **Layer.js**

|  |
| --- |
| <**script src="src/layers/Layer.js"**></**script**> <**script src="src/layers/MenuLayer.js"**></**script**> <**script src="src/layers/GameLayer.js"**></**script**> |

Vamos a **Main.js,** ahora que tenemos varias capas/pantallas **, layer** va a ser un puntero a la capa que queremos que se muestre y actualice.

|  |
| --- |
| *// Canvas y contexto* **var *canvas*** = ***document***.getElementById(**"canvas"**); **var *contexto*** = ***canvas***.getContext(**"2d"**); **var *escaladoMinimo*** = 1;  *// Capas* **var *layer***;  **var *gameLayer***; **var *menuLayer***; |

Modificamos la funcion **iniciarJuego()** para instanciar el **menuLayer** y asignarselo tambien a **layer**, va a ser la capa visible cuando comience el juego.

|  |
| --- |
| *// Inicio capas y bucle del juego* **function** *iniciarJuego*() {  *~~gameLayer = new GameLayer();~~* ***menuLayer*** = **new** MenuLayer();  ***layer*** = ***menuLayer***;  *setInterval*(*loop*, 1000 / 30); } |

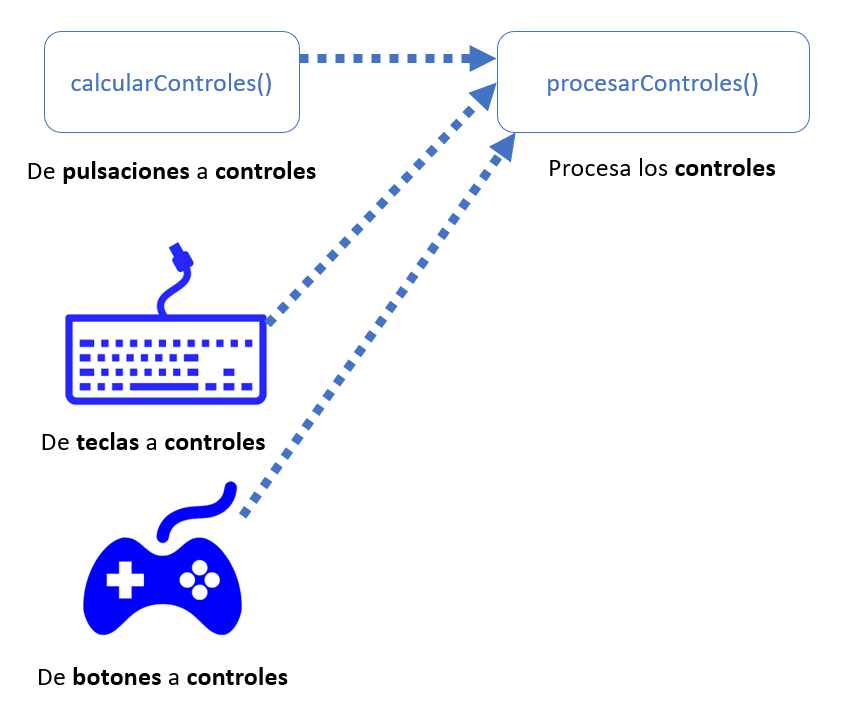
|  |
| --- |
| **Cuando crear los objetos layers:** No estaría muy mal crear todos los objetos layer en la función **iniciarJuego(),** aunque tambien se puede esperar a que se vayan a utilizar (bajo demanda) |

El **loop()** NO debe actualizar el **gameLayer**, sino la capa a la que apunte la variable **layer**.

|  |
| --- |
| **function** *loop*(){  ***layer***.actualizar();  **if** ( ***teclas***.**length** == 0) {  ***layer***.calcularPulsaciones(***pulsaciones***);  }  ***layer***.procesarControles();  ***layer***.dibujar();   *actualizarPulsaciones*(); } |

Finalmente volvemos a **MenuLayer.js** e implementamos la función **calcularPulsaciones(),** si resulta que han pulsado el botón especificamos un **control.continuar=true.**

La función **procesarControles()** que debe contener la logica que va a si **control.continuar=true**. No procesar la logica en **calcularControles()**



Sí **control.continuar=true**. se creará el **GameLayer** y se asignará a la variable **layer** (esto va a hacer que se visualice esa pantalla),

|  |
| --- |
| iniciar() {  **this**.**fondo** =  **new** Fondo(***imagenes***.**menu\_fondo**,480\*0.5,320\*0.5);  **this**.**boton** =  **new** Boton(***imagenes***.**boton\_jugar**,480\*0.5,320\*0.7); }  calcularPulsaciones(pulsaciones){  **this**.**boton**.**pulsado** = **false**;   **for**(**var** i=0; i < pulsaciones.**length**; i++){  **if** (**this**.**boton**.contienePunto(pulsaciones[i].**x** , pulsaciones[i].**y**) ){  **this**.**boton**.**pulsado** = **true**;  **if** ( pulsaciones[i].**tipo** == ***tipoPulsacion***.**inicio**) {  ***controles***.**continuar** = **true**;  }  }  }   *// No pulsado - Botón Disparo* **if** ( !**this**.**boton**.**pulsado** ){  ***controles***.**continuar** = **false**;  } }  procesarControles( ) {  *// siguiente pantalla* **if** (***controles***.**continuar**) {  ***gameLayer*** = **new** GameLayer();  ***layer*** = **gameLayer**;  ***controles***.**continuar** = **false**;  } } |



|  |
| --- |
| **Táctil, cada pantalla puede tener botones diferentes :** El control por pulsaciones **EventosPulsacion** tiene la particularidad de que debe ser interpretado de forma diferente en cada Layer (ya que cada Layer tiene distintos botones), por eso tenemos un método **calcularPulsaciones** en cada Layer.  **Teclas**: sí quisiéramos utilizar teclas también en el **MenuLayer** , seguramente habría que “activar” el **controles.continuar** con alguna tecla concreta. |

# Accedemos a EventosTeclado.js en onKeyDown declaramos que la barra espaciadora activa el control continuar.

|  |
| --- |
| **function** *onKeyDown*( event) {  ***entrada*** = ***entradas***.**teclado**;  *// agregar la tecla pulsada si no estaba* **var** posicion = ***teclas***.indexOf(event.keyCode);  **if** ( posicion == -1 ) {  ***teclas***.push(event.keyCode);  **switch** ( event.keyCode ){  **case** 32:  ***controles***.**disparo** = **true**;  ***controles***.**continuar** = **true**;  **break**; |

En **onKeyUp()** debemos acordarnos también de desactivar el **controles.continuar**.

|  |
| --- |
| **function** *onKeyUp*( event) {  *// sacar la tecla pulsada* **var** posicion = ***teclas***.indexOf(event.keyCode);  ***teclas***.splice( posicion, 1);  ***console***.log(**"Tecla levantada"**);   **switch** ( event.keyCode ){  **case** 32:  ***controles***.**disparo** = **false**;  ***controles***.**continuar** = **false**;  **break**; |

Sí hemos implementado entrada por **GamePad** habría que hacer que alguno de los botones pusiera **controles.continuar = true** al pulsarlo y **controles.continuar = false** al dejar de pulsarlo.

Sí hemos usado un mando también tenemos que agregar el **controles.continuar** a **EventosGamePad.js.**

|  |
| --- |
| **function** *actualizarOrdenes*(){  *// Obtener gamePad en cada iteración* **var** gP1 = ***navigator***.getGamepads()[0];   **for**(**var** i=0; i < gP1.buttons.length; i++){  **if** ( gP1.buttons[i].pressed ) {  ***entrada*** = ***entradas***.**gamepad**;  }  }  **if** ( **entrada** != ***entradas***.**gamepad**){  **return**;  }  *// Pad de la izquierda gP1.axes[0] - X , gP1.axes[1] - Y  // Los joystics van valores entre -1(izquierda) y 1(derecha)  // El centro del joystick es 0  // Float con 2 decimales* ***controles***.**moverX** = *parseFloat*(gP1.axes[0]).toFixed( 2 );   **if** ( gP1.buttons[1].pressed){ *// 0 es el botón X* **if** ( ***botonesPulsados***[0] == **false** ) {  ***botonesPulsados***[0] = **true**;  ***controles***.**moverY** = 1;  }  } **else** {  ***botonesPulsados***[0] = **false**;  ***controles***.**moverY** = 0;  }   **if** ( gP1.buttons[2].pressed){ *// 2 es el botón A* **if** ( ***botonesPulsados***[2] == **false** ) {  ***botonesPulsados***[2] = **true  *controles***.disparo = **true**;  controles.continuar = **true**;  }  } **else** {  botonesPulsados[2] = **false** controles.disparo = **false**;  controles.continuar = **false**;  } } |

# \* Pausa y mensajes en el juego

En muchos casos puede ser interesante mostrar mensajes o instrucciones a los usuarios durante la partida, estos mensajes se pueden enviar con un **Boton** o un **Texto (o incluso ambos)**

En este ejemplo vamos a optar por un **Boton** (al pulsar el botón el mensaje se quita, aunque también podrían ir por tiempo, con una cuenta atrás).

Accedemos al **GameLayer** y en su **constructor** agregamos dos nuevas variables **mensaje** y **pausa**.

En este caso es **mejor definirlas en el constructor**, sino siempre que iniciemos el juego lo hará pausado (esto muchas posibilidades de configurar el funcionamiento)

|  |
| --- |
| constructor() {  **super**();  **this**.**mensaje** = **new** Boton(***imagenes***.**mensaje\_como\_jugar**, 480/2, 320/2);  **this**.**pausa** = **true**;  **this**.iniciar(); } |

El **this.mensaje** apunta al mensaje actual pero puede cambiarse en cualquier momento, instanciando otro botón con otra imagen.

En el método **dibujar()** indicamos que el mensaje se dibuje en último lugar y solo si el juego está en **pausa.**

|  |
| --- |
| dibujar (){  **this**.calcularScroll();  **this**.**fondo**.dibujar();  **for** (**var** i=0; i < **this**.**bloques**.length; i++){  **this**.**bloques**[i].dibujar(**this**.**scrollX**);  }  **for** (**var** i=0; i < **this**.**disparosJugador**.length; i++) {  **this**.**disparosJugador**[i].dibujar(**this**.**scrollX**);  }  **this**.**copa**.dibujar(**this**.**scrollX**);  **this**.**jugador**.dibujar(**this**.**scrollX**);  **for** (**var** i=0; i < **this**.**enemigos**.length; i++){  **this**.**enemigos**[i].dibujar(**this**.**scrollX**);  }  **this**.**fondoPuntos**.dibujar();  **this**.**puntos**.dibujar();  **if** ( !**this**.**pausa** && **entrada** == ***entradas***.**pulsaciones**) {  **this**.**botonDisparo**.dibujar();  **this**.**botonSalto**.dibujar();  **this**.**pad**.dibujar();  }  **if** ( **this**.**pausa** ) {  **this**.**mensaje**.dibujar();  } } |

Falta implementar que cuando estamos en **pausa** el juego se detenga, basta con no ejecutar el método **actualizar()**.

|  |
| --- |
| actualizar (){  **if** (**this**.**pausa**){  **return**;  }    **if** ( **this**.**copa**.colisiona(**this**.**jugador**)){ |

Hemos declarado que el juego se inicia con **this.pausa=true** , por lo que comienza pausado y con un mensaje de explicación.

**Para quitar la pausa – tactcil**

De forma táctil pulsando en la pantalla. En **calcularPulsaciones()** establecemos que con que si nos llega una pulsación de **tipoPulsacion.inicio** en cualquier parte de la pantalla activamos el **controles.continuar = true;**

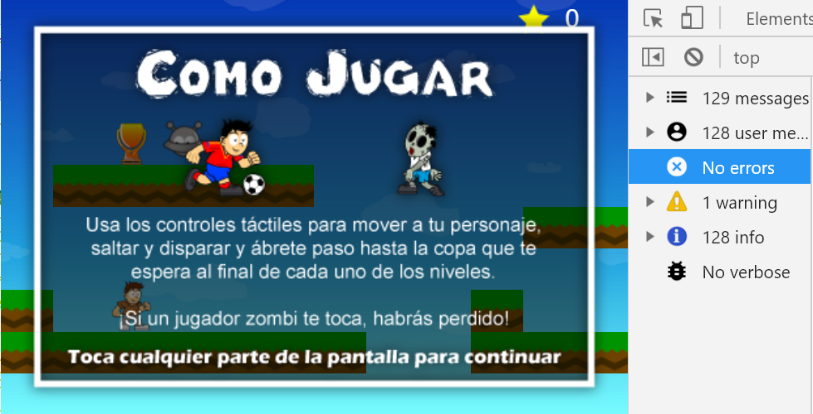
|  |
| --- |
| calcularPulsaciones(pulsaciones){  *// Suponemos botones no estan pulsados* **this**.**botonDisparo**.**pulsado** = **false**;  **this**.**botonSalto**.**pulsado** = **false**;  *// suponemos que el pad esta en el centro* ***controles***.**moverX** = 0;  *// Suponemos a false* ***controles***.**continuar** = **false**;   **for**(**var** i=0; i < pulsaciones.**length**; i++){  *// MUY SIMPLE SIN BOTON cualquier click en pantalla lo activa* **if**(pulsaciones[i].**tipo** == ***tipoPulsacion***.**inicio**){  ***controles***.**continuar** = **true**;  }   **if** (**this**.**pad**.contienePunto(pulsaciones[i].**x** , pulsaciones[i].**y**) ){ |

En el **procesarControles(),** sí **controles.continuar** **== true,** quitamos la **pausa** .

|  |
| --- |
| procesarControles( ){  **if** (***controles***.**continuar**){  ***controles***.**continuar** = **false**;  **this**.**pausa** = **false**;  }  *// disparar* |

|  |
| --- |
| **Recordar, teclado y Gamepad**. Estamos “reusando” el **controles.continuar** el cual se ponía a true también en los otros métodos de entrada: teclado (barra espaciadora), mando ( botón 0). |

Comprobamos que el mensaje aparece y desaparece.



Para cambiar el mensaje solo hay que redefinir el Botón y pausar el juego, por ejemplo en el método **actualizar()** cuando se detecta la colisión con la copa, se puede cambiar el mensaje por el de “has ganado”.

|  |
| --- |
| actualizar (){  **if** (**this**.**pausa**){  **return**;  }   **if** ( **this**.**copa**.colisiona(**this**.**jugador**)){  ***nivelActual***++;  **if** (***nivelActual*** > ***nivelMaximo***){  ***nivelActual*** = 0;  }  **this**.**pausa** = **true**;  **this**.**mensaje** =  **new** Boton(***imagenes***.**mensaje\_ganar**, 480/2, 320/2);  **this**.iniciar();  } |

# \*Enemigo – No caer, detectar fuera.

En esta parte vamos a ampliar el **Espacio** de físicas para que los elementos puedan “saber si tienen algún pie/lado en el aire”. Se puede utilizar para implementar una lógica que evite que se caigan.

Los **enemigos** se mueven siempre hacia un lado hasta que “chocan” contra un **bloque**, se mueven incluso si se van a caer. Para que los elementos sean capaces de detectar caídas se debe contemplar esto en el **Espacio** de físicas.

Accedemos a **Espacio.js** y en la función **actualizar()** declaramos 2 variables nuevas (**fueraPorDerecha** y **fueraPorIzquierda**) las cuales suponemos a “**true**” antes de hacer cualquier comprobación esta fuera.

|  |
| --- |
| actualizar(){  **for**( **var** i=0; i < **this**.**dinamicos**.length; i++){  *// aplicar gravedad ( dinamicos)* **this**.**dinamicos**[i].**vy** = **this**.**dinamicos**[i].**vy** + **this**.**gravedad**;  *// maxima velocidad de caida por gravedad* **if** (**this**.**dinamicos**[i].**vy** > 20) {  **this**.**dinamicos**[i].**vy** = 20;  }   *// reiniciar choques* **this**.**dinamicos**[i].**choqueAbajo** = **false**;  **this**.**dinamicos**[i].**fueraPorDerecha** = **true**;  **this**.**dinamicos**[i].**fueraPorIzquierda** = **true**;    *//derecha* **this**.moverDerecha(i);  **this**.moverIzquierda(i); |

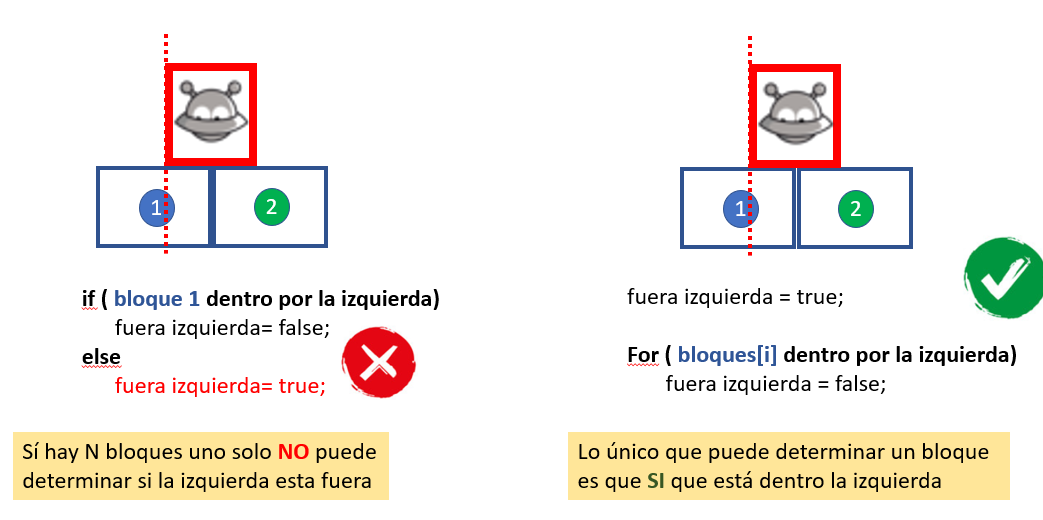
Estas dos variables van a determinar si la esquina derecha o la esquina izquierda del elemento no tienen “contacto” con ningún bloque. Es posible que el elemento no se haya caído, pero alguna de sus lados no tiene ya contacto con “el suelo”.

Esta comprobación es bastante sencilla de realizar, basta con acceder al método **moverAbajo()** y una vez comprobamos que hay choque con un bloque. Si su lado derecho e izquierdo estan dentro del bloque significa que no está fuera.

En el momento que su parte derecha / izquierda este dentro de un bloque ya está dentro por ese lado.



|  |
| --- |
| *// Choca!!* **if** ( (abajoDinamico + **this**.**dinamicos**[i].**vy**) >= arribaEstatico &&  (arribaDinamico + **this**.**dinamicos**[i].**vy**) < abajoEstatico  && izquierdaDinamico < derechaEstatico  && derechaDinamico > izquierdaEstatico ){   **if** (movimientoPosible >= arribaEstatico - abajoDinamico ){  *// Más restrictivo que el anterior!* movimientoPosible = arribaEstatico - abajoDinamico ;  **this**.**dinamicos**[i].**choqueAbajo** = **true**;    **if** (derechaDinamico <= derechaEstatico) {  **this**.**dinamicos**[i].**fueraPorDerecha** = **false**;  }   **if** (izquierdaDinamico >= izquierdaEstatico) {  **this**.**dinamicos**[i].**fueraPorIzquierda** = **false**;  }  } } |



Ahora el **Enemigo.js** puede usar estas 2 variables en su lógica (método **actualizar()** )

|  |
| --- |
| actualizar (){  *// Actualizar animación* **this**.**animacion**.actualizar();   **switch** (**this**.**estado**){  **case *estados***.**moviendo**:  **this**.**animacion** = **this**.**aMover**;  **break**;  **case *estados***.**muriendo**:  **this**.**animacion** = **this**.**aMorir**;  **break**;  }   **if** ( **this**.**estado** == ***estados***.**muriendo**) {  **this**.**vx** = 0;  } **else** {   **if** ( **this**.**vx** == 0){  **this**.**vxInteligencia** = **this**.**vxInteligencia** \* -1;  **this**.**vx** = **this**.**vxInteligencia**;  }   **if** (**this**.**fueraPorDerecha** ){  *// mover hacia la izquierda vx tiene que ser negativa* **if** ( **this**.**vxInteligencia** > 0){  **this**.**vxInteligencia** = **this**.**vxInteligencia** \* -1;  }  **this**.**vx** = **this**.**vxInteligencia**;  }  **if** (**this**.**fueraPorIzquierda** ){  *// mover hacia la derecha vx tiene que ser positiva* **if** ( **this**.**vxInteligencia** < 0){  **this**.**vxInteligencia** = **this**.**vxInteligencia** \* -1;  }  **this**.**vx** = **this**.**vxInteligencia**;  }  } } |

No invertir siempre la velocidad directamente, asegurarse del valor que tenía antes, sino hay riesgo de que “bloqueo entre rebotes”.

