Rodrigo Sánchez Valle

I.E.S. Francisco Romero Vargas

2ºCFGS Desarrollo de Aplicaciones Web

Jerez de la Frontera, Cádiz

Tabla de contenido

[Presentación 3](#_Toc72322134)

[Bucle principal del juego 3](#_Toc72322135)

[Dimensiones de la pantalla 4](#_Toc72322136)

[Rectángulos 5](#_Toc72322137)

[Detección de teclas 5](#_Toc72322138)

[Hojas de sprites y Tiled 6](#_Toc72322139)

[Tiled 6](#_Toc72322140)

[Hoja de sprites 6](#_Toc72322141)

[Cargar archivos sin caché 7](#_Toc72322142)

[Sprites 7](#_Toc72322143)

[Tiles 8](#_Toc72322144)

[Paleta de sprites 8](#_Toc72322145)

[Capa de tiles 9](#_Toc72322146)

[Máquina de Estados 10](#_Toc72322147)

[Mapa 10](#_Toc72322148)

[Estado del Mapamundi 11](#_Toc72322149)

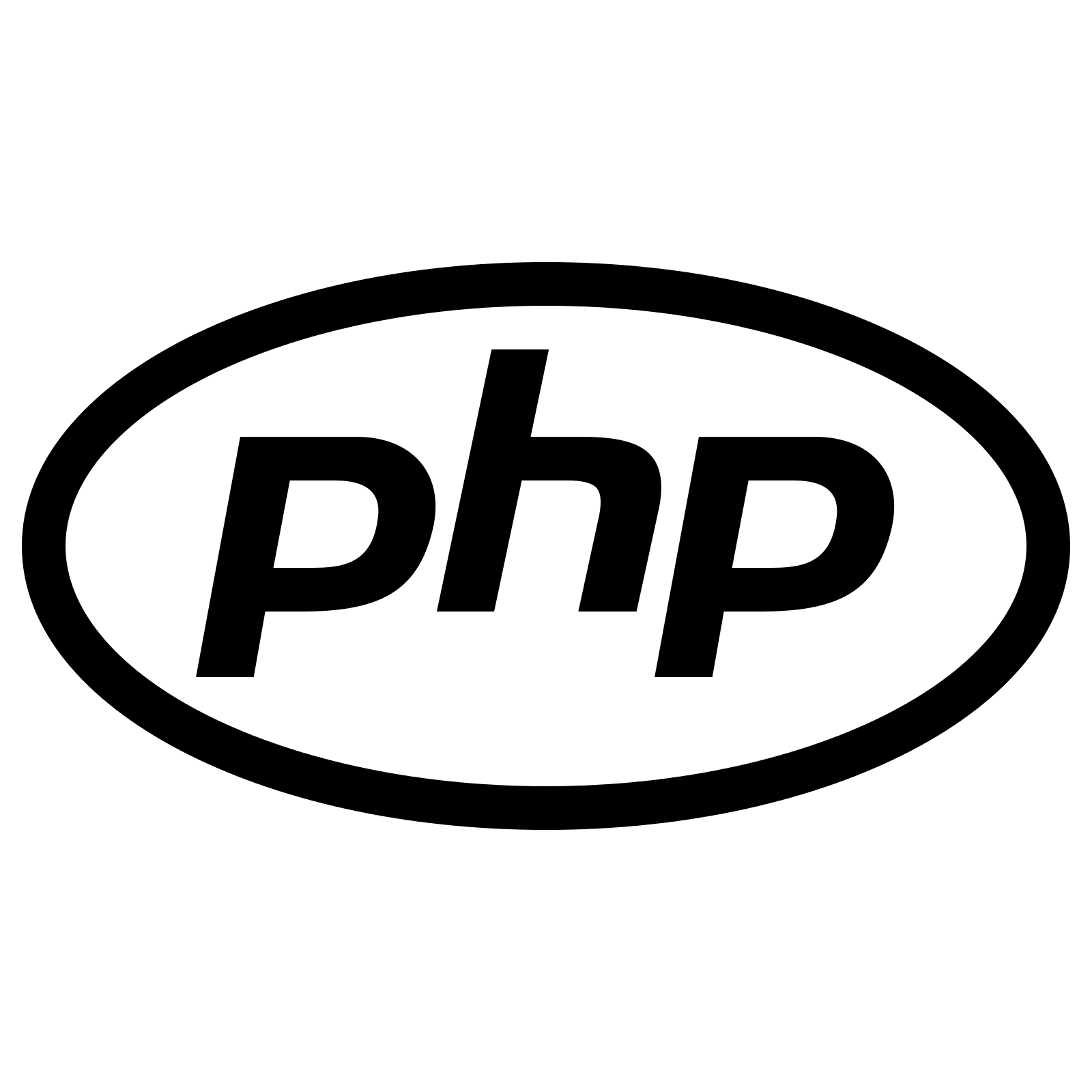
[Inicio 11](#_Toc72322150)

[Iniciadores 11](#_Toc72322151)

[Iniciar Juego 11](#_Toc72322152)

[Encadenar Inicios 12](#_Toc72322153)

# Presentación

Este juego puede ser jugado desde cualquier navegador web y sistema operativo. Para ello usaremos JavaScript desde el cliente, y para el servidor PHP, que está alojado en mi propio ordenador a través de XAMPP, que se encargará de guardar la partida y servir los recursos para el juego.

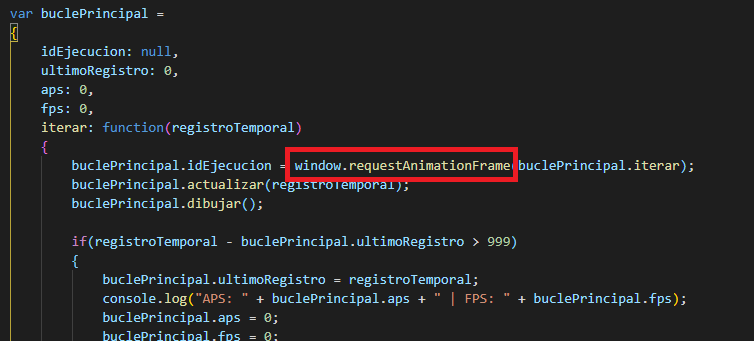
El juego es 2D, cuenta 2 cámaras diferentes. Una cámara cenital para mostrar las cosas desde arriba y nos permite explorar el mundo del juego. Sin embargo, cuando entremos a un nivel, la cámara cambia a una vista lateral para que nos permita explorar el nivel, a este tipo de juegos se le conoce como *Scroll lateral*.

Una de las funciones que tiene es poder guardar la partida en un servidor. De esta forma, el usuario tendrá siempre su partida guardada de forma online y podrá jugar desde cualquier ordenador entrando con su cuenta. A su vez, esto me permite incluir más niveles, objetos o personajes desde el servidor con solo actualizar unas pocas entradas en la base de datos. Así, la próxima vez que el usuario entre a la partida ya podrá ver las novedades sin tener que descargar o actualizar nada.

# Bucle principal del juego

La función de esta parte del código en simple, y es hacer que JavaScript esté siempre abierto, ejecutándose.

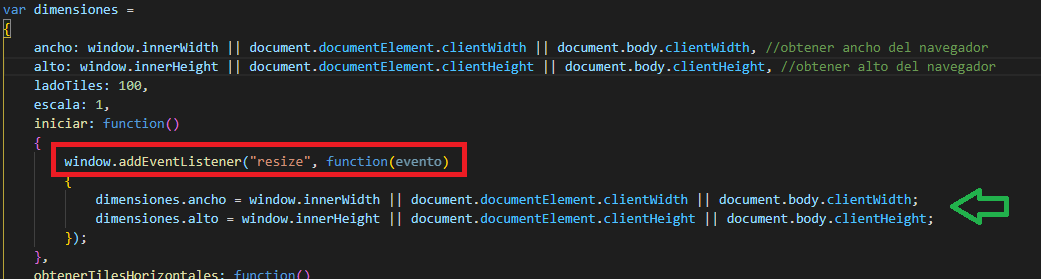
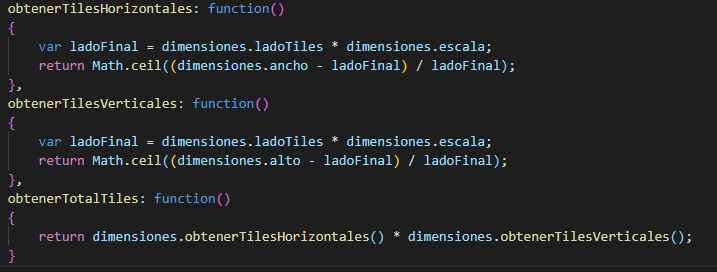
La idea principal es que nuestro navegador se ejecute 60 veces por segundo para que la velocidad de dibujado sea lo suficientemente rápida. Para ello se usa el siguiente método:



El método **window.requestAnimationFrame()** informa al navegador que quieres realizar una animación y solicita que el navegador programe el repintado de la ventana para el próximo ciclo de animación. Haciendo esto se consigue hacer un Callback. Cuando se ejecuta esa parte del código, el valor que devuelve es el tiempo medido en milisegundos desde su ejecución, e inyecta ese valor en **registroTemporal***.*

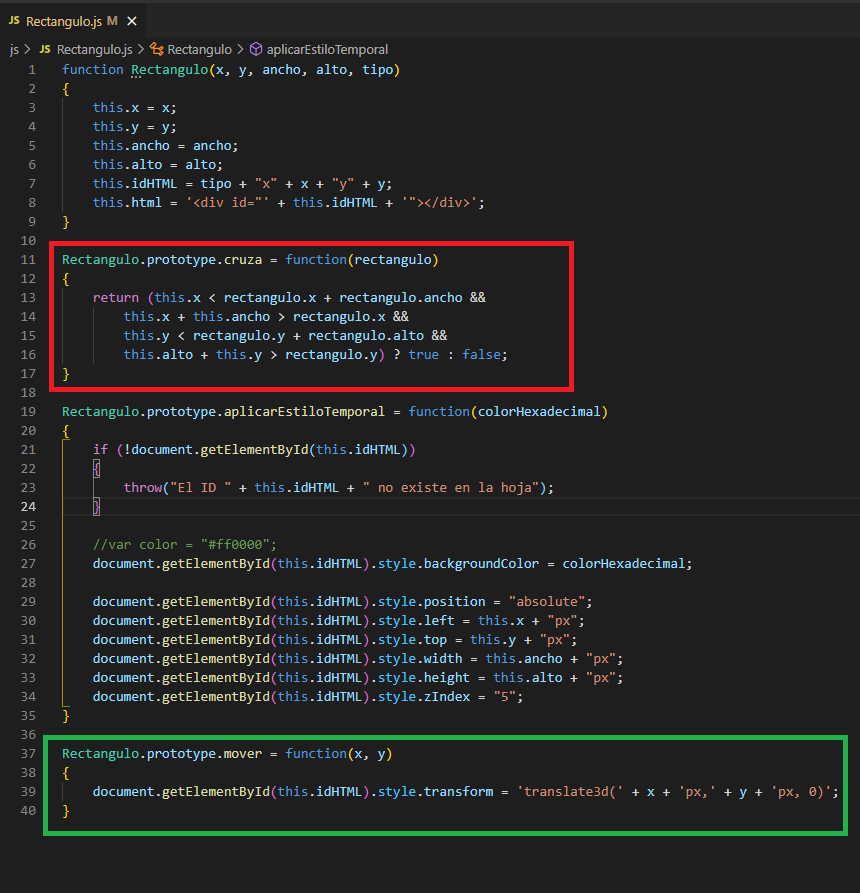
# Dimensiones de la pantalla

Para conseguir que el juego esté bien posicionado en el navegador, hay que obtener el alto y el ancho del mismo, porque no siempre tendrá el mismo tamaño, ya que el usuario puede cambiar el tamaño o minimizar la ventana. Para poder conseguir las dimensiones, se usa la función **addEventListener**

El evento que vamos a “escuchar” es **resize**para poder capturar el tamaño de la ventana cuando esta sufra una modificación y le podamos reasignar un valor a **ancho**y **alto**.

Por último, estas tres funciones tienen una finalidad simple. Obtienen la cantidad de tiles que caben de izquierda a derecha según el tamaño de la pantalla.

# Rectángulos

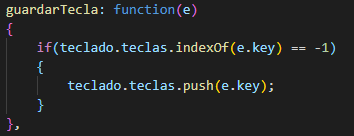
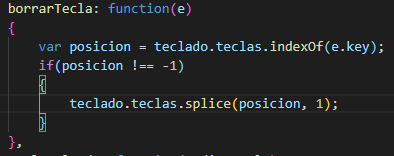
La clase **Rectangulos** tiene una finalidad muy precisa, y es la creación de rectángulos para nuestro juego. Básicamente, nuestro juego funcionará con rectángulos; el personaje, las localizaciones y las hitbox son, en esencia, son rectángulos.

La función **.cruza** devuelve **true** o **false** según si un rectángulo esta “cruzando” a través de otro. Si este devuelve true, significa que las figuras están chocando, si devuelve false, no están en contacto. Esto es fundamental a la hora de saber si el personaje está entrando en un nivel del mapa, o está colisionando con montañas, árboles o cualquier entorno, ya sea en el mapamundi, o en el nivel en el que se encuentre.

En **.mover**, gracias a la función **translate3d()** PENDIENTE DE EXPLICAR

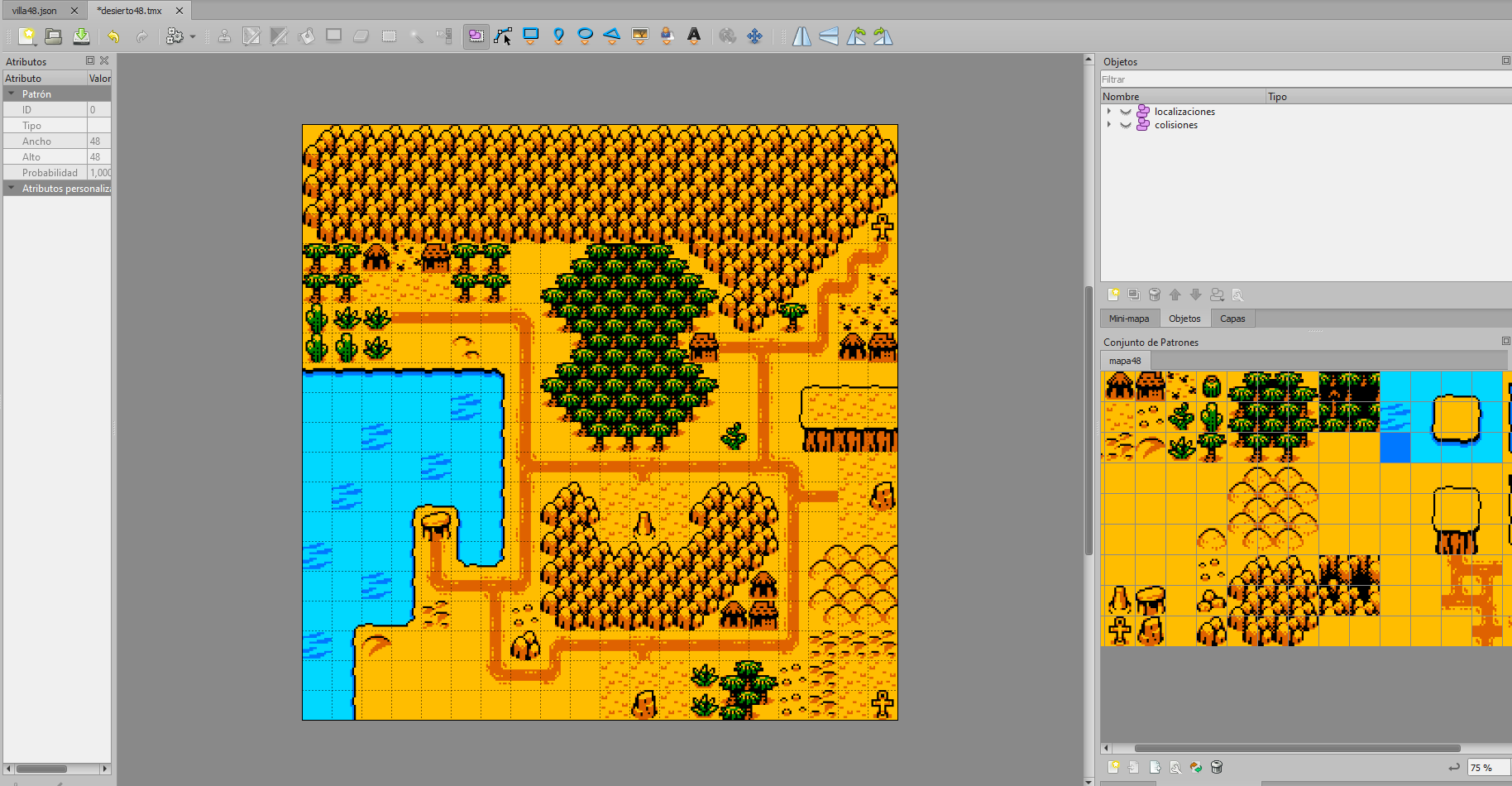
# Detección de teclas

Como es de esperar, esta parte del código nos permite conocer qué tecla del teclado estamos pulsando, para posteriormente asignarle una función, como puede ser mover al personaje de arriba abajo en el minimapa, acceder a un nivel o saltar.

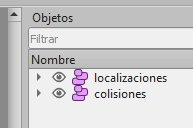
En esta parte del código, lo más interesante es el uso de los eventos **.onkeydown** y **.onkeyup**. El primero se ejecuta cuando el usuario esté presionando la tecla, y el segundo evento se ejecutará cuando este suelte la tecla. Esto lo hacemos para detectar el movimiento del personaje hacia una dirección, y cuando el código detecta que el usuario ha soltado la tecla, hará que nuestro personaje se detenga donde sea.

# Tiled | Flexible level editorHojas de sprites y Tiled

## Tiled

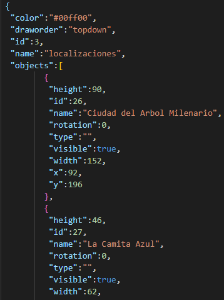
Para el diseño de niveles y el mapa, he decidido usar la aplicación Tiled. Esta herramienta tiene las características idóneas para este tipo de proyectos, ya que el dibujado es muy simple; solo tenemos que bajarnos una plantilla de sprites de internet (<https://opengameart.org/>) y a partir de esta, diseñar tu propio mapa.

## Hoja de sprites

Una herramienta indispensable es la de “seleccionar objetos”. Su función es la de generar un área delimitada por unas coordenadas registradas en un fichero .*json*, un identificador y un color para diferenciarlo todo visualmente.

Aquí podemos observar cómo, al poner visibles las capas de objetos, se nos muestra de forma visual dos tipos de áreas distintas, una verde y una roja. Se puede intuir a simple vista la función de cada tipo de área:

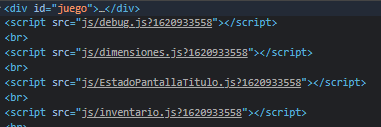
* **Colisiones**: los cuadrador rojos delimitan las zonas por las que nuestro personaje no podrá caminar, a esto se le denomina “hitbox”. Se les suele aplicar a zonas de agua no navegable, arboles, montañas, delimitando el mapa y cualquier zona a la que no deseemos que el jugador acceda .
* **Localizaciones**: los cuadrados verdes marcan las ubicaciones a las que el jugador puede acceder para entrar a un nivel. Estas tienen un nombre, que hace de identificador. Al pulsar una tecla especifica, el usuario accederá al nivel en cuestión.

Para finalizar, debemos exportar el fichero con dos formatos distintos: *json* y *png*. En el *json* podremos encontrar las distintas capas creadas, con las coordenadas en las que se crearon, su tamaño, su nombre y varias características del mismo, las cuales posteriormente usaremos. A demás de todo esto, hay muchos mas datos, como por ejemplos los *tilesets,* que son los datos propios del mapa: nombre, ruta, alto, anto, etc.

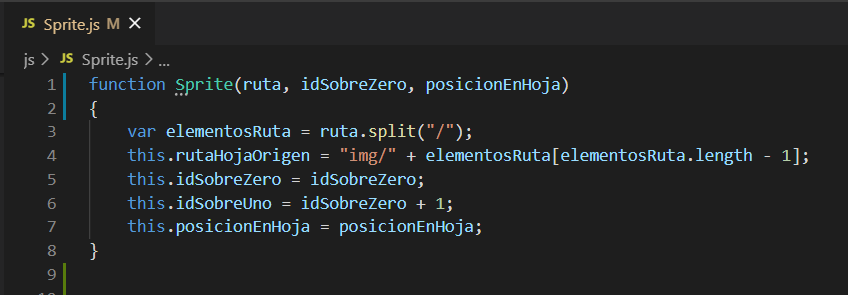
# Cargar archivos sin caché

La función de este fichero tiene una finalidad muy definida, y es cargar los archivos *.js* necesarios para el juego sin necesidad de recurrir a la caché. La única “complicación”, por llamarlo de alguna manera, que he tenido con este fichero, ha sido el orden en el que se ordenan todos los *.js*, ya que hay algunos que dependen de otros, lo cual hace que se deban ejecutar antes que otros.

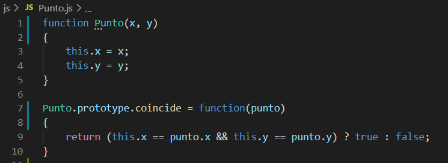
Uno de los motivos por los que me decidí a hacer este fichero es por el tema de los cambios en el código. La primera vez que visitas una web, el navegador lo lee y lo guarda en la memoria, la siguiente vez que entremos, como el navegador ya tiene una versión reciente, y hasta que no pase un cierto tiempo, este no considera necesario descargar de nuevo los archivos. Eso, para mí como desarrollador, es problemático, porque al hacer algún cambio en los ficheros y querer ver el cambio, al estar guardado el estado del navegador en caché, tendría que recargar pulsando Ctrl+F5, pero por profesionalidad y por comodidad, decidí hacerlo de esta manera.

La forma en la que lo conseguí es sencilla. Solo tenia que añadir al nombre del fichero, un numero distinto cada vez que se recargue la pagina, para que el navegador considere que es necesario cargar en memoria todos los ficheros. Usé la función **getTimestamp()**paraañadir una numeración distinta gracias a **DateTime();**

# Sprites

Esta clase se encargará de administrar todas las capas de sprites del juego, o mejor dicho, en la hoja de sprites.

En primer lugar, necesitamos reconstruir la ruta, porque en el *.json* de nuestro mapa, la ruta aparece de esta manera tan “peculiar”. Separamos la cadena cada vez que se encuentra el carácter “/” y cogemos solo el último elemento para obtener el nombre del archivo.

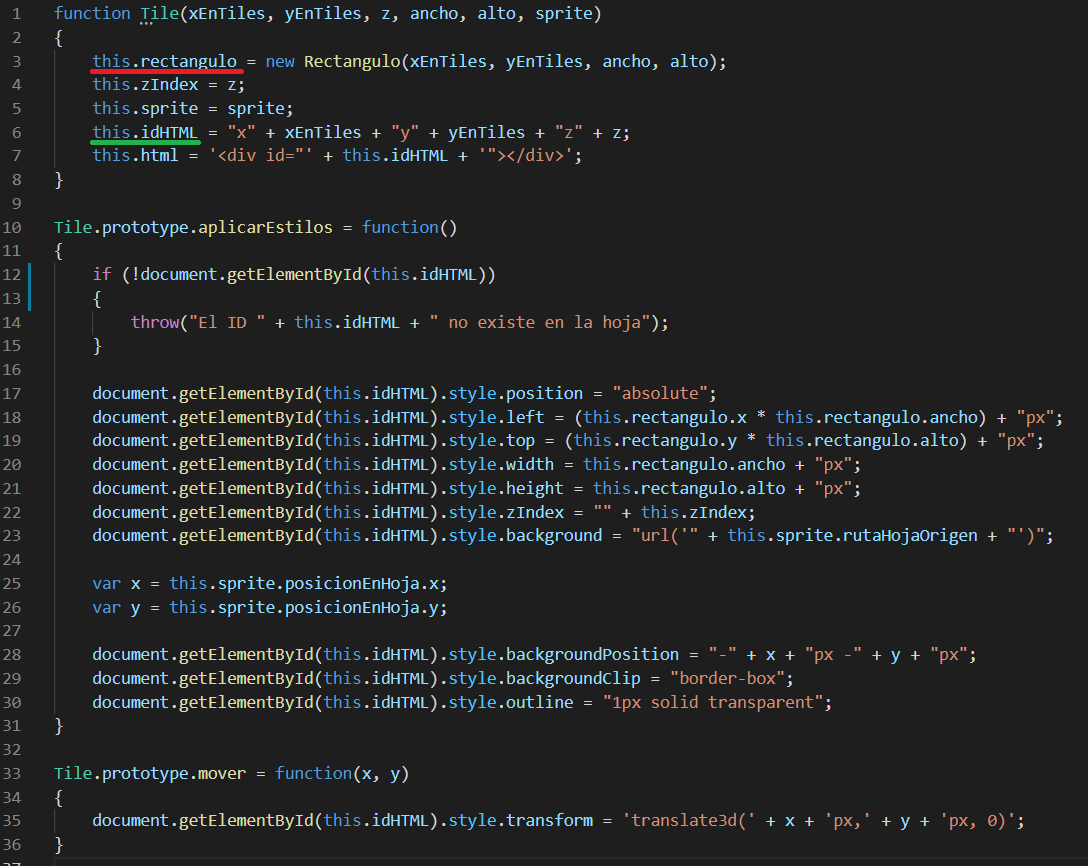
EL RESTO DEL FICHERO SERÁ EXPLICADO MAS ADELANTE (IDSOBREZERO IDSOBREUNO)

Punto se refiere a un punto en una rejilla de coordenadas. La función de esta clase es indicar si dos puntos coinciden o no. El uso del objeto **Punto.prototype.coincide** es muy importante, ya que esto hace que esta función ocupe menos espacio en memoria. Todas las instancias que vayamos a crear de *Punto* y objetos no van a tener su propia copia de este método, solo habrá una para todas, y como este método va a hacer lo mismo en todos los casos, nos va a ahorrar un montón de memoria.

# Tiles

Los tiles son cada uno de estos cuadraditos que nosotros hemos generado anteriormente, y cada uno de ellos son un cuadradito del mapa que contiene su propio Sprite.

this.rectangulo va a ser la posición del Tile, el rectángulo que lo va a representar.

this.idHTML es la id del elemento HTML que representará el tile en el navegador

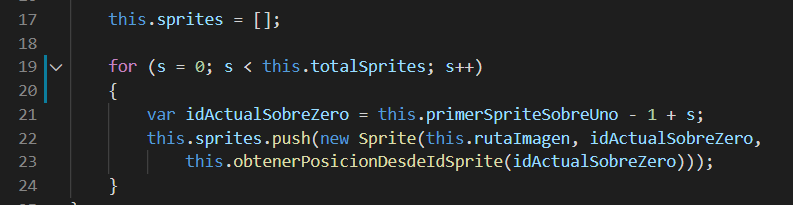
El resto del código simplemente le da un estilo a los Tiles en píxeles.

# Paleta de sprites

En su medida, nuestra paleta de sprites se puede parecer a la paleta de colores de un pintor, que es donde tiene todos los colores para su dibujo. En nuestro caso, la paleta de sprites va a contener todos los sprites necesarios para nuestro mapa.

Los **datosSprites** que recibe nuestra función vienen del archivo **.*json*** del mapa creado para el juego.

La siguiente parte del código se encarga de la parte más interesante de la función, que es guardar los sprites:

Hacemos un bucle del total de sprites, que contiene el ancho de la imagen medido en sprites multiplicado por el alto de la imagen medido en sprites.

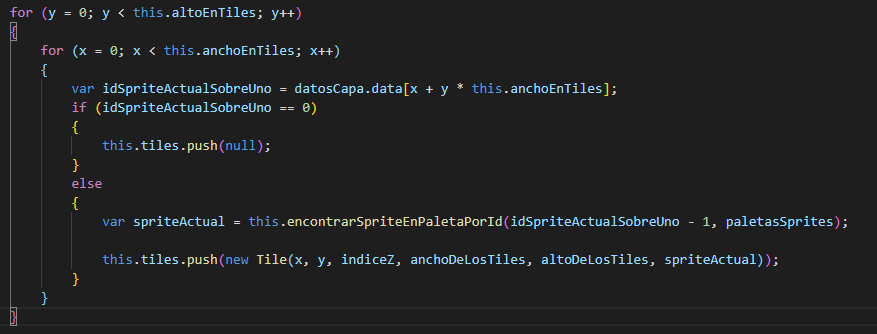
Se crea la variable **idActualSobreZero** para hacer que el id del Sprite, en vez de ser 1, sea 0, ya que Tiled tiene el “inconveniente” de que el primer id de la lista de sprites es 1.

Continuamos rellenando el array de sprites con la función **.push()** , creamos un nuevo Sprite y le pasamos los argumentos necesarios para crearlo; la ruta, el id sobre cero y la posición en la hoja.

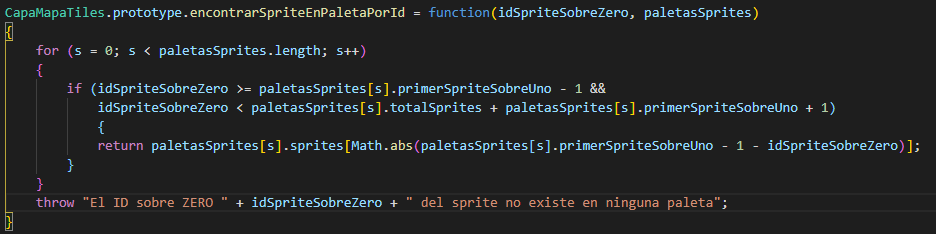
EXPLICAR FUNCIÓN . MOVER

# Capa de tiles

La capa de tiles consiste en, valga la redundancia, las diferentes capas que va a tener nuestro juego web. Una primera capa sería el fondo, o sea, el mapa, por encima puede estar los elementos que no se pueden atravesar, como muros o montañas, las localizaciones donde el personaje puede acceder, y como última capa, el propio personaje. En este proyecto, se desarrolló con una única capa, para simplificar el trabajo, pero no se descarta que en un futuro se modifique el código para hacerlo funcionar en varias capas.



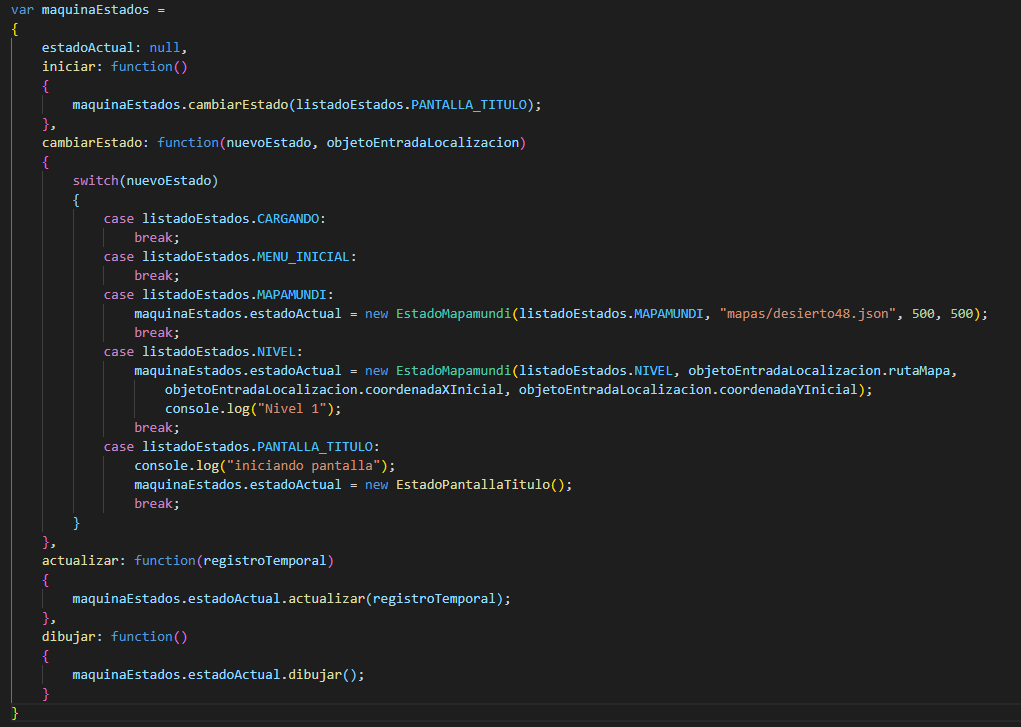
Recorremos los 20 tiles verticales de arriba hacia abajo con el bucle externo, y con el interno, los 20 tiles horizontales de izquierda a la derecha. Lo que leeremos aquí es este array de nuestro mapa .*json*, una coordenada,para saber que Sprite tenemos que seleccionar.

Como estamos trabajando con una única paleta de sprites, se hace un bucle de la paleta de sprites para leer cada una de ellas. Accedemos a la paleta actual y obtenemos el valor del primer Sprite para saber cuál es la paleta de sprites con la que estamos trabajando. En programación, eso se saca de la siguiente forma:

# Máquina de Estados

Es el mecanismo que se va a encargar de alternar los estados del juego. Algunos de sus estados son: visualizando el mapa del mundo, dentro del nivel, mientras el juego carga, el menú del juego…

Crearemos un fichero para crear una constante para listar el estado de nuestro juego.

Después, crearemos el fichero importante, el de máquina de estados. Este controlará en qué estado está el juego actualmente y qué es lo que se está ejecutando.

Nada mas iniciarlo, cambiaremos su estado y elegimos el primer estado el que queremos que aparezca, en este caso **PANTALLA\_TITULO.**

Al **listadoEstados.MAPAMUNDI,** en resumen,la añadimos esa línea para darle una identificación, a demás de los atributos necesarios para crearlo. Lo mismo sucede con el estado **listadoEstados.NIVEL** y el estado **listadoEstados.PANTALLA\_TITULO**

Tendremos una función llamada **cambiarEstado,** que evaluará mediante un switch el estado del juego.

REVISAR

# Mapa

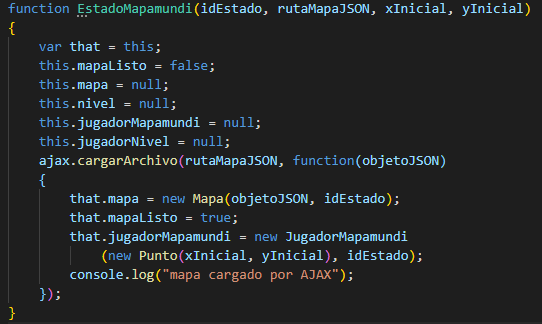
El archivo del mapa es el fichero mas grande y contiene funciones algo mas complejas. Gracias a este, podremos controlar tanto los mapas cenitales como los laterales.

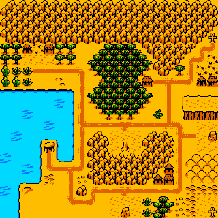
Para poder entender bien el mecanismo de esta parte del proyecto, vayamos enumerando cada parte del mismo

PENDIENTE INICIARCAPAS

|  |  |
| --- | --- |
| this.posicion = new Punto(0,0); | Será 0,0 porque gracias a CSS, podemos mover el mapa entero de un lado a otro, y así conseguiremos que no tenga mucho impacto en el rendimiento, |
| this.posicionActualizada = new Punto(0,0); | Sirve para distinguir si en cada actualización el mapa debe moverse o no, ya que, si no debe moverse, es inútil estar llamando al método que lo va a mover o dibujar, porque eso desperdicia recursos. |
| this.anchoMedidoEnTiles = parseInt(objetoJSON.width);  this.altoMedidoEnTiles = parseInt(objetoJSON.height);  this.anchoDeLosTiles = parseInt(objetoJSON.tilewidth);  this.altoDeLosTiles = parseInt(objetoJSON.tileheight); | Obtenemos el string que viene del fichero json y cogemos el atributo, y como viene en formato de texto, hay que transformarlo a numero |
| this.iniciarCapas(objetoJSON.layers); | Seleccionamos las capas que necesitamos del fichero json |
|  |  |
|  |  |
|  |  |
|  |  |

# Estado del Mapamundi

El propósito de esta clase es controlar la ejecución del juego mientras se está en el mapamundi, es decir, mientras el jugador explora el mapa del mundo para acceder a los niveles.

De las partes más interesantes del código de este fichero es, **ajax.cargarArchivo()** de forma asíncrona. Le tendremos que pasar una ruta y una función anónima donde admitiremos un objeto *.json*, y dentro de esto tenemos que coger los datos y dárselos al mapa. Dentro, simplemente tenemos que coger los datos y dárselos al mapa. El problema viene cuando dentro de la función anónima, al ser un bloque de código totalmente independiente al resto, no puedo acceder, por ejemplo, al .*mapa*. Así que decidí hacer un “arreglo”, que consiste en crear una **var that = this;**. Gracias a esto, desde dentro de la función anónima podremos acceder al **EstadoMapamundi()**. Ahora si podremos hacer referencia al objeto **EstadoMapamundi()**, a su atributo **mapa**, por ejemplo.

Gracias a esto último que se ha desarrollado, ya se vería el mapamundi de nuestro juego web.

PENDIENTE INICIAR CAPAS INICIAR ELEMENTOSMAPA ACTUALIZAR Y DIBUJAR

# Inicio

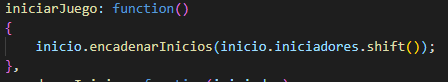
Crearemos un array de funciones.

## Iniciadores

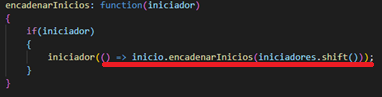
En **iniciadores** indicaremos que queremos que se inicie y en qué orden, y a demás estamos haciendo que se ejecuten una detrás de la otra, es decir, hasta que no se ejecute la primera y haya acabado, no comenzará la segunda, y así sucesivamente.

## Iniciar Juego

Lo interesante de esta función es que haremos un Callback. Callback significa que podremos pasar una función como argumento a otra función, de esta forma la primera función ejecutará su código y cuando termine, se ejecutará detrás la segunda función.

En **iniciarJuego** llamaremos a **encadenarInicios**, le pasamos un iniciador como argumento, así que le pasamos **inicio**, le pasamos nuestro arraya **iniciadores** y usamos la función de JavaScript **.shift()**. Shift devuelve el primer elemento y luego lo borra del array, cada vez que lo ejecutemos irá borrando uno a uno los elementos del array hasta que este esté vacío.

## Encadenar Inicios

Como es lógico, una vez ejecutada y finalizada la función **iniciarJuego()**, eventualmente no devolveremos nada porque los elementos de **iniciadores** habrán sido borrados, y esto puede causar una excepción. Lo primero que tenemos que comprobar si el iniciadorexiste.

Necesitamos seguir usando la mecánica del Callback para hacer que una función llame a la siguiente.

Básicamente, este **primer bloque** está pasando una función como Callback, que se ejecutará cuando el **iniciador** termine, así que, por ejemplo, cuando **maquinaEstados.iniciar()** termine, se ejecutará el Callback que hay en el argumento. Esta sintaxis nos permite colocar una función anónima “()” y la flecha “=>” indica que **iniciador** devuelve como argumente lo que se ejecuta a la derecha de esta flecha. En resumen, es una función que se llama así misma.

CONTROLES