Rodrigo Sánchez Valle

I.E.S. Francisco Romero Vargas

2ºCFGS Desarrollo de Aplicaciones Web

Jerez de la Frontera, Cádiz

Tabla de contenido

[Presentación 3](#_Toc71743053)

[Bucle principal del juego 3](#_Toc71743054)

[Dimensiones de la pantalla 4](#_Toc71743055)

[Rectángulos 5](#_Toc71743056)

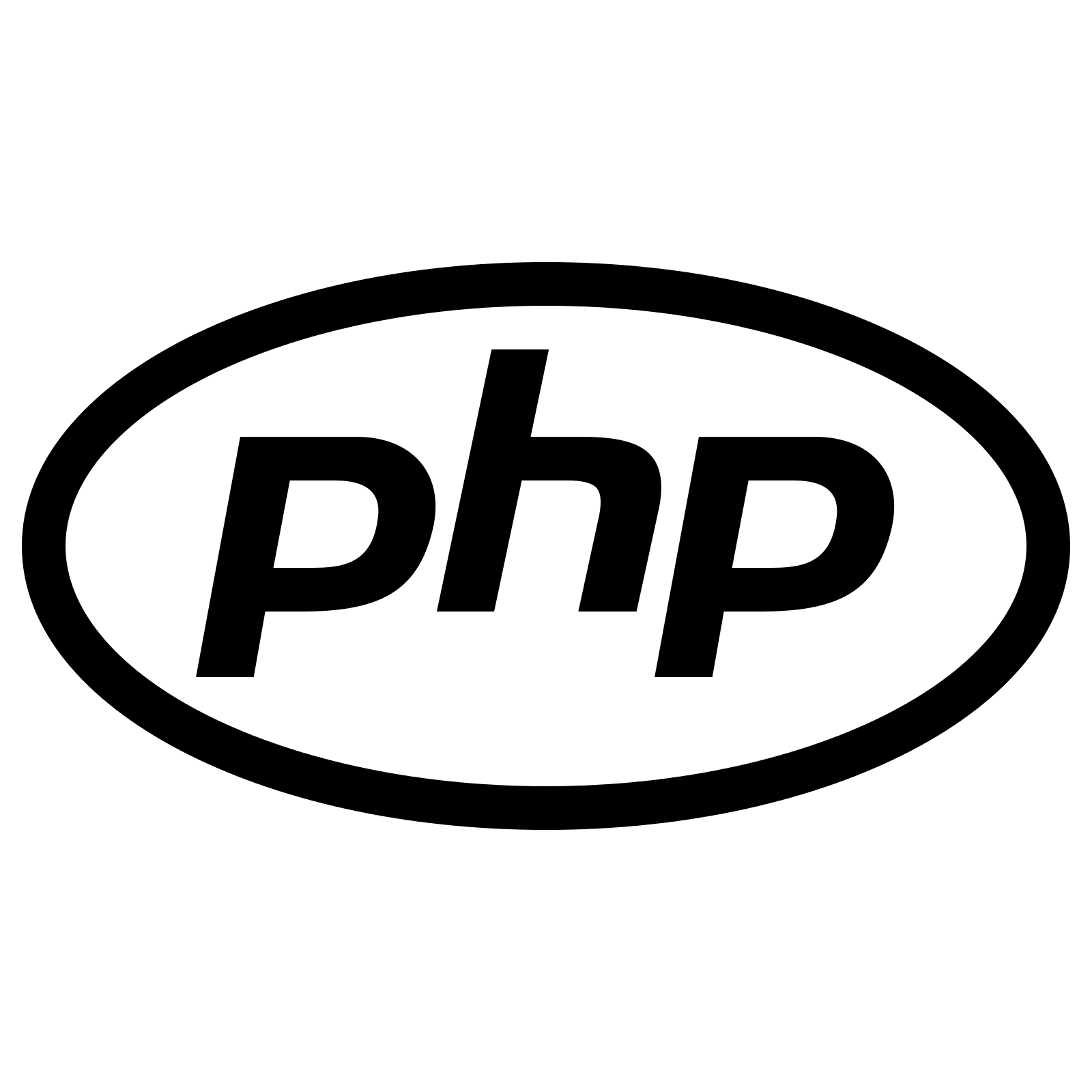
[Detección de teclas 5](#_Toc71743057)

[Hojas de sprites y Tiled 6](#_Toc71743058)

[Tiled 6](#_Toc71743059)

[Hoja de sprites 6](#_Toc71743060)

# 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.

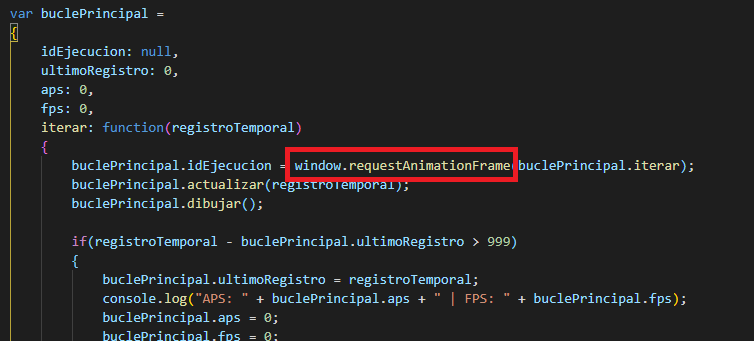
El juego es 2D, cuenta 2 cámaras diferentes. Una cámara cenital para mostrar las cosas desde arriba y nos permite explorar el mundo del juego. Sin embargo, cuando entremos a un nivel, la cámara cambia a una vista lateral para que nos permita explorar el nivel, a este tipo de juegos se le conoce como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

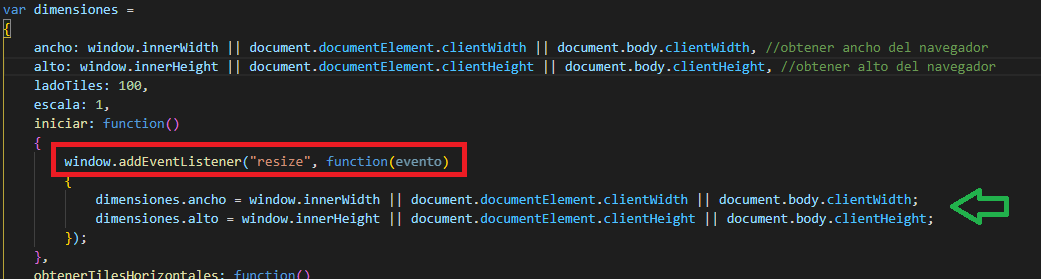
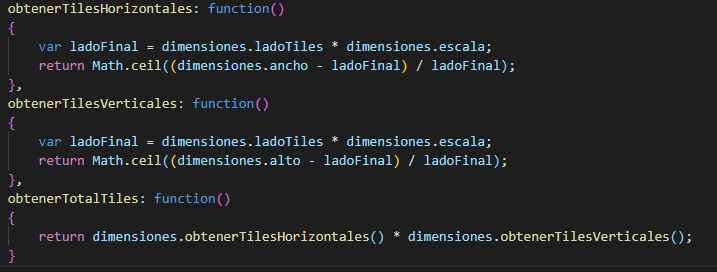
La idea principal es que nuestro navegador se ejecute 60 veces por segundo para que la velocidad de dibujado sea lo suficientemente rápida. Para ello se usa el siguiente método:



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***.*

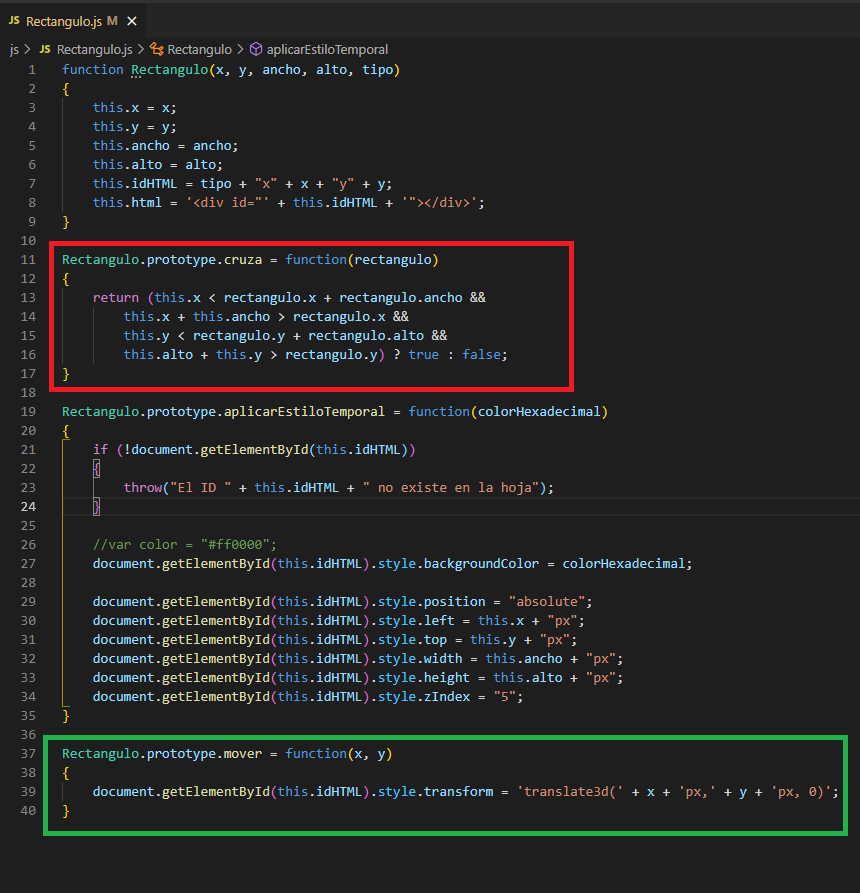
# Dimensiones de la pantalla

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

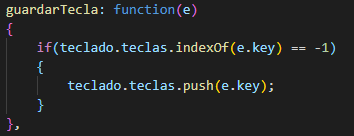
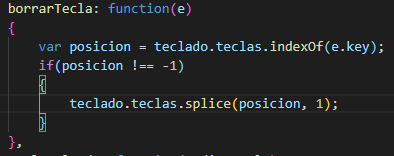
La clase **Rectangulos** tiene una finalidad muy precisa, y es la creación de rectángulos para nuestro juego. Básicamente, nuestro juego funcionará con rectángulos; el personaje, las localizaciones y las hitbox son, en esencia, son rectángulos.

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, arboles 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

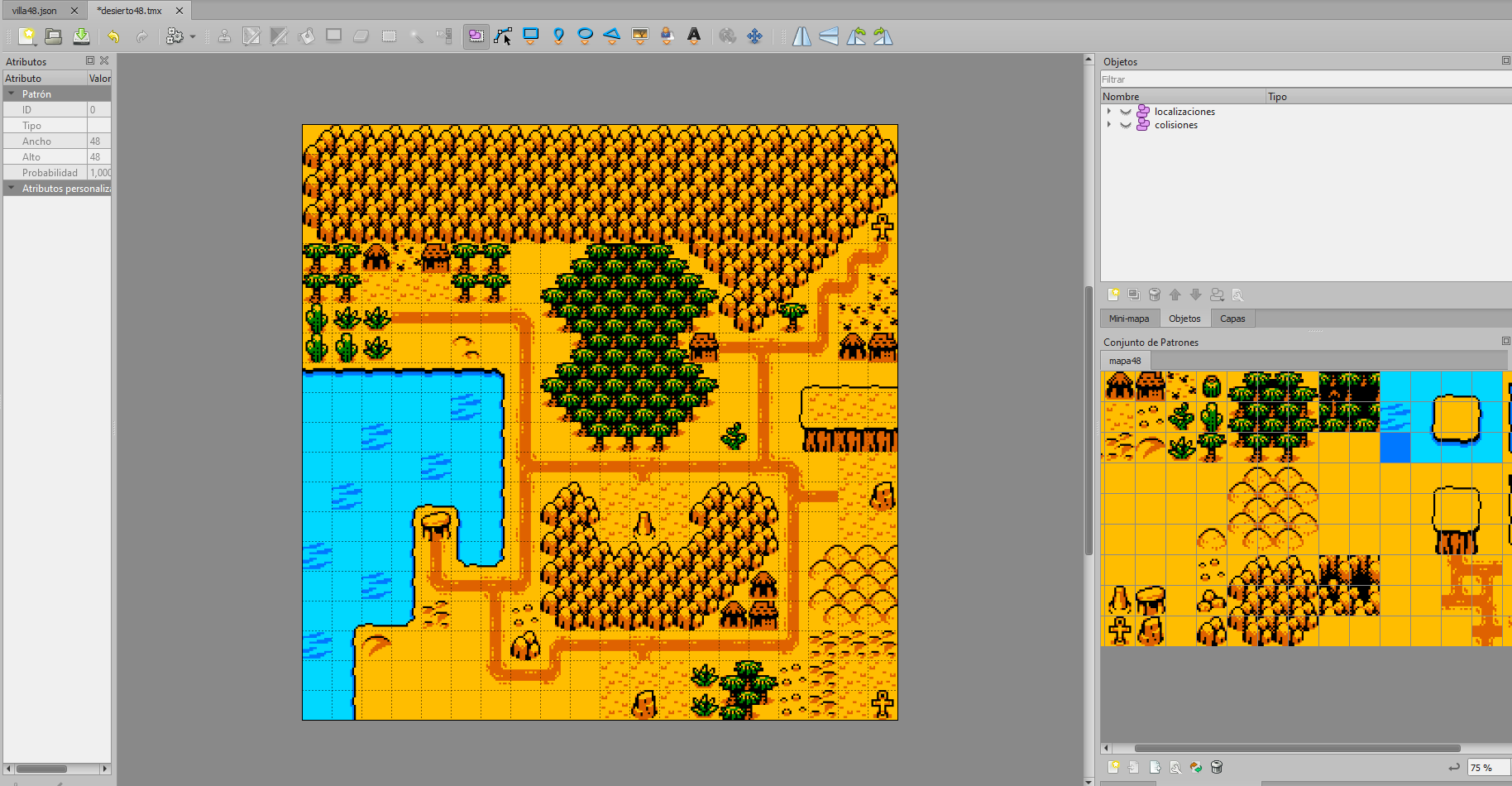
# Detección de teclas

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.

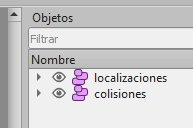
En esta parte del código, lo más interesante es el uso de los eventos **.onkeydown** y **.onkeyup**. El primero se ejecuta cuando el usuario esté presionando la tecla, y el segundo evento se ejecutará cuando este suelte la tecla. Esto lo hacemos para detectar el movimiento del personaje hacia una dirección,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

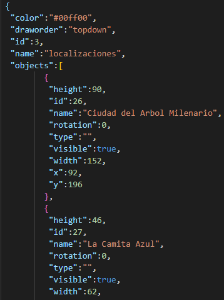
Para el diseño de niveles y el mapa, he decidido usar la aplicación Tiled. Esta herramienta tiene las características idóneas para este tipo de proyectos, ya que el dibujado es muy simple; solo tenemos que bajarnos una plantilla de sprites de internet (<https://opengameart.org/>) y a partir de esta,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

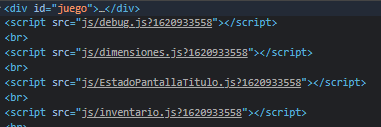
* Colisiones: los cuadrador rojos delimitan las zonas por las que nuestro personaje no podrá caminar, a esto se le denomina “hitbox”. Se les suele aplicar a zonas de agua no navegable, arboles, montañas, delimitando el mapa y cualquier zona a la que no deseemos que el jugador acceda .
* Localizaciones: 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.

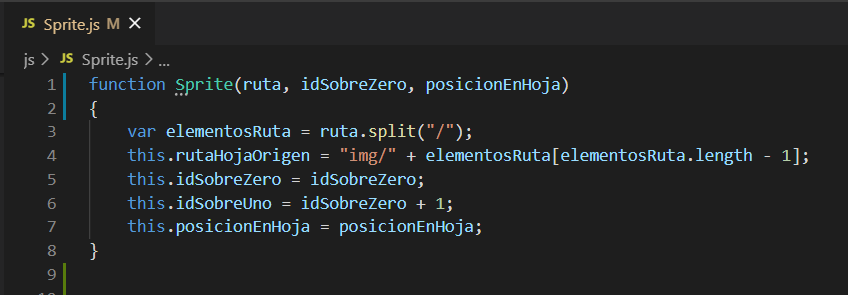
# Cargar archivos sin caché

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.

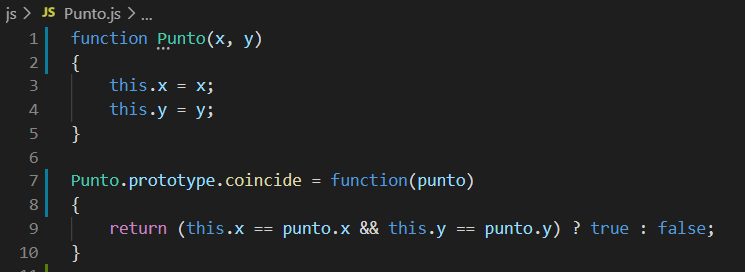
Uno de los motivos por los que me decidí a hacer este fichero es por el tema de los cambios en el código. La primera vez que visitas una web, el navegador lo lee y lo guarda en la memoria, la siguiente vez que entremos, como el navegador ya tiene una versión reciente, y hasta que no pase un cierto tiempo, 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.

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.