Taller de Desempeño Profesional

Technical Expert

(Phaser)

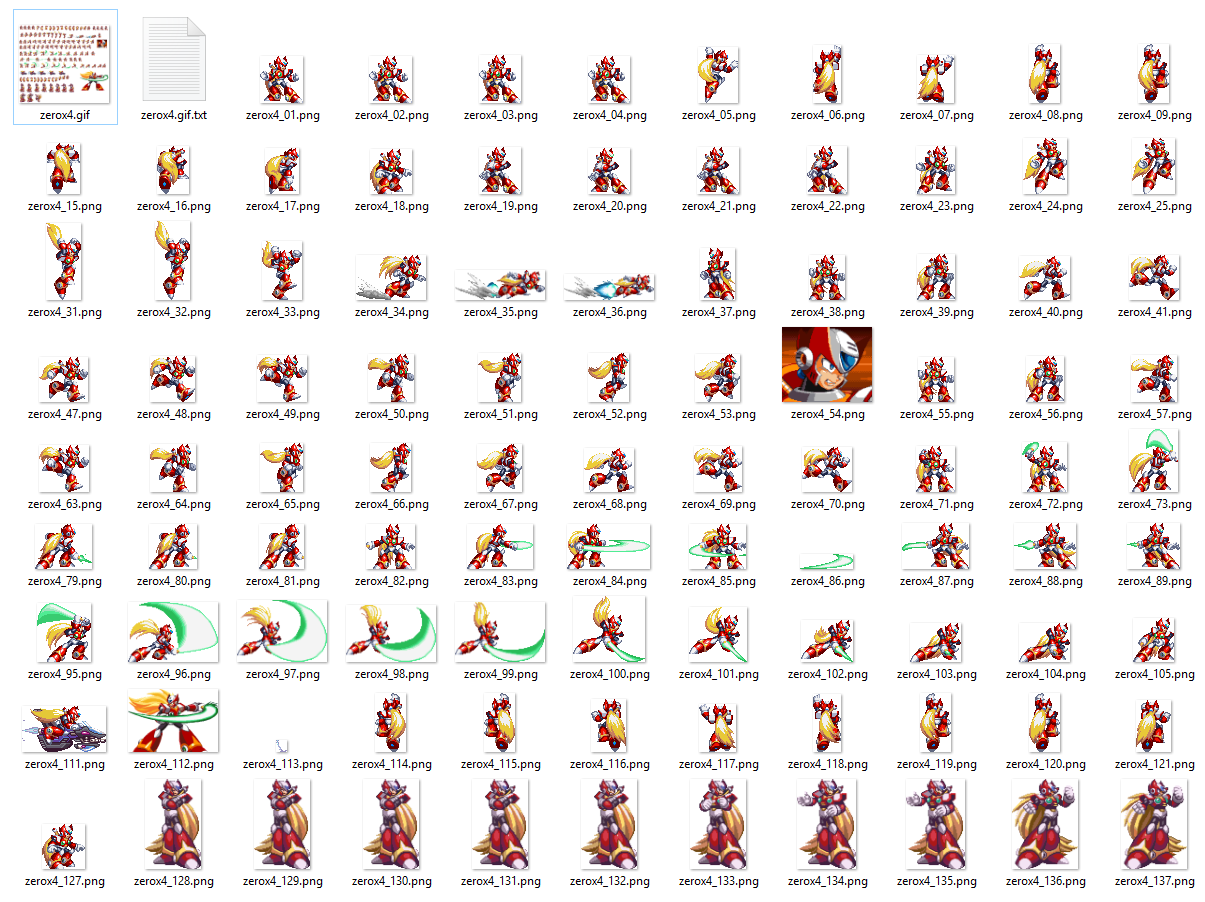
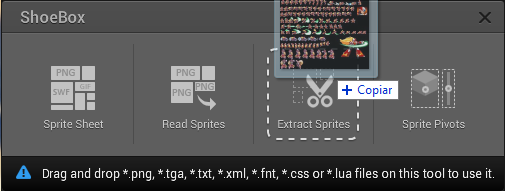


Demo Sprite Atlas & Tilemaps

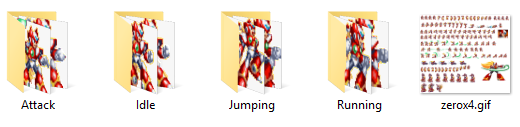
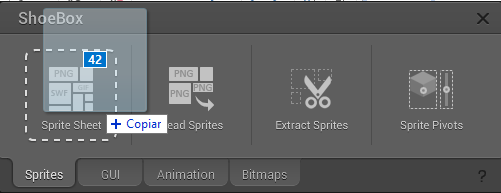
Eduardo Chiang

Para esta demo utilizaremos dos herramientas externas las cuales son soportadas nativamente por Phaser. La primera es ShoeBox, la cual nos permite extraer imágenes individuales de un *spritesheet* como también hacer el proceso inverso.

Primero arrastraremos el *spritesheet* a utilizar a la opción “Extract Sprites” para recortar la plancha original en imágenes individuales.



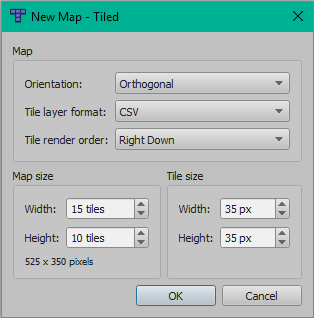
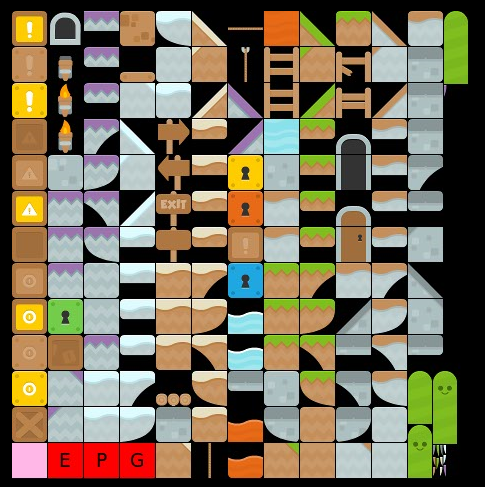
Elegimos las animaciones que queremos utilizar con sus respectivas secuencias de imágenes para reorganizarlas, nombrándolas secuencialmente (Ej: Idle01, Idle02, Idle03, etc). Luego copiamos los contenidos dentro de una sola carpeta y seleccionamos todas las imágenes para arrastrarlas a la opción “Sprite Sheet” de la herramienta.

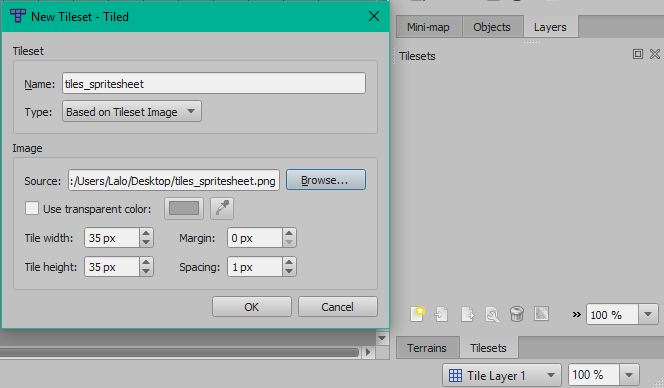
El resultado final es un *spritesheet* que contiene todas las imágenes que utilizaremos como animaciones y un archivo .xml que mapeará cada una de estas imágenes con identificador que será su nombre original.



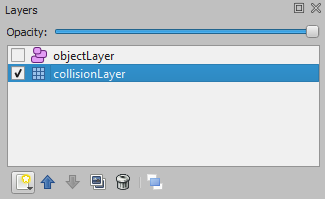
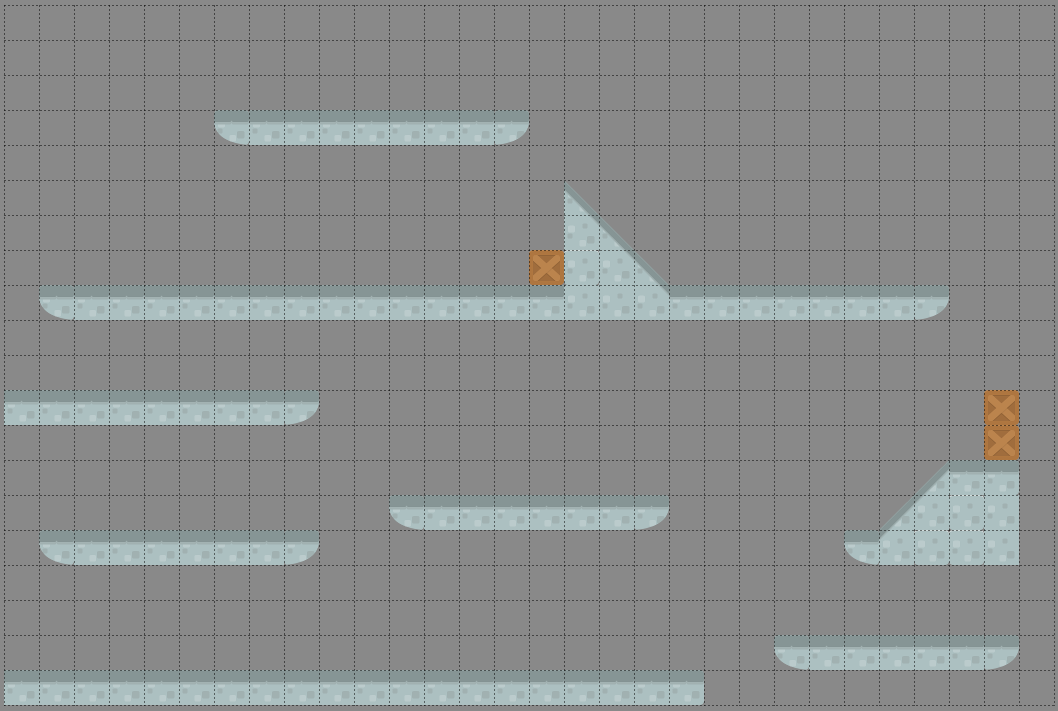
La segunda herramienta que utilizaremos es Tiled, la cual nos permite crear *tilemaps* fácilmente a partir de planchas que contengan *tiles*. Comenzaremos creando un mapa de tamaño 15x10 *tiles* que utilizará *tiles* de tamaño 35x35 pixeles.

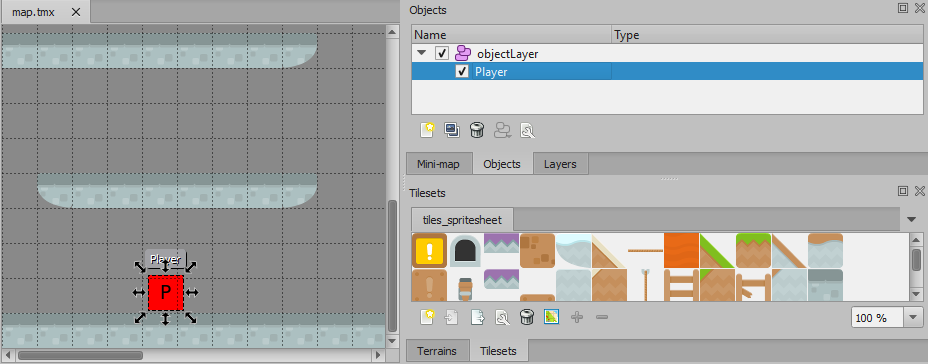
 

Luego asignaremos una plancha de *tiles* cuyo tamaño es de 35x35 pixeles y que tienen un espaciado de 1 pixel entre ellas.



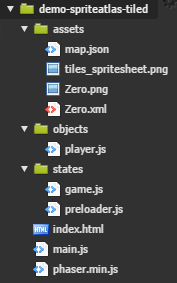
Crearemos dos capas, una de colisión que contendrá todos los objetos sólidos del mapa que queremos crear y una capa de objetos, la cual tendrá elementos individuales identificados por tipos o nombres.





Finalmente, exportamos el mapa a un archivo .json que contiene toda la información de los objetos y sus respectivas posiciones.



El directorio del proyecto tendrá la siguiente estructura:

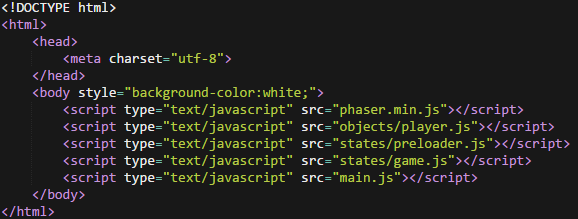
Carpeta de *Assets* Se guardarán todos los archivos multimedia a utilizar en el proyecto.

Carpeta de *Objects* Se guardarán los archivos .js de clases extendidas.

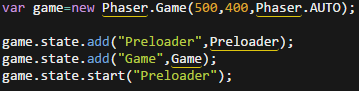
Carpeta de *States* Se guardarán los archivos .js de estados a utilizar en el proyecto.

Raíz En la raíz se encontrará el *index*, el *main* y el archivo que contiene Phaser.

En el *index* sólo se agregan los scripts de javascript como Phaser, *main*, estados y clases extendidas.



En el *main* declararemos la variable *game* que contendrá nuestro objeto Phaser. Será creado con un tamaño de 500x400 y con detección automática para el renderizador, ya sea WebGL o Canvas. Luego se declaran los estados que serán utilizados en la aplicación y especificamos con qué estado iniciar la aplicación.



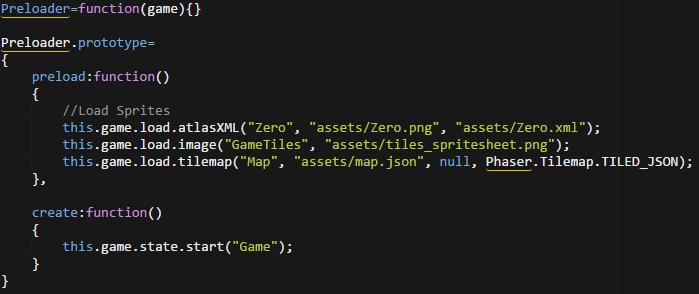
En el estado “Preloader” se declaran los *assets* que se desean precargar.

Cargaremos el *spritesheet* como un atlas de XML, el cual recibe un nombre único para identificarlo, la imagen que contiene las animaciones y el archivo .xml que lo mapea.

Para cargar el *tilemap*, recibirá un nombre único para identificarlo, el archivo .json que contiene toda su información, una URL nula ya que aún no existe en la aplicación y definimos que se utilizará el formato JSON creado por el programa Tiled.

Además, es necesario cargar la imagen que contiene los *tiles*.

Finalmente, redirigimos la aplicación al estado “Game”.



Dentro del estado “Game”, declararemos que se utilizará física del tipo Arcade, la cual sólo utiliza colisiones entre cuadrados, y definiremos una gravedad de -1000.

Luego, cargaremos el *tilemap* y lo adjuntaremos con la imagen que contiene todos los *tiles.* Además, le agregaremos un cuerpo sólido a todos los elementos en la capa de colisión.

Para crear el jugador, obtendremos la posición del objeto “Player” de la capa de objetos.

Finalmente, declaramos que existirá colisión entre el jugador y el mapa creado.



En la clase “Player”, utilizaremos la posición obtenida previamente del *tilemap* y declararemos que tendrá un cuerpo sujeto a la física de la aplicación. Adicionalmente, haremos que la cámara del juego siga el movimiento del jugador ya que previamente se ejecutó la función “resizeWorld()” para ajustar el tamaño de nuestro mundo al del mapa utilizado.

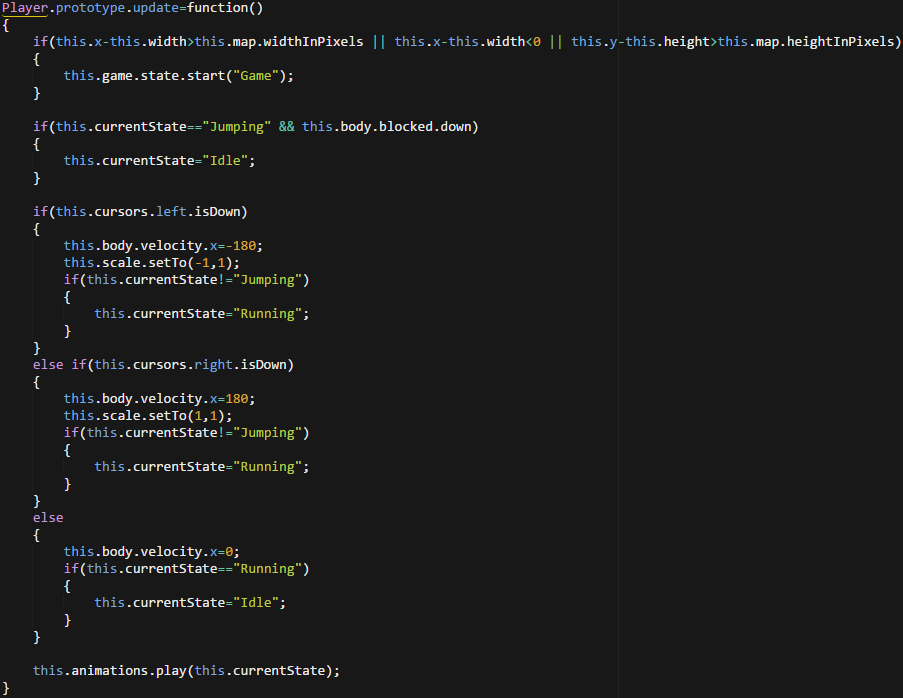
El jugador utilizará las flechas para moverse y la tecla “Z” para atacar.

Para cada animación crearemos un identificador único y entregaremos un arreglo que contenga la secuencia de imágenes que representa dicha animación, además de la velocidad a la que debe ser ejecutada y si es que debe o no continuar reproduciéndose después de completarse.

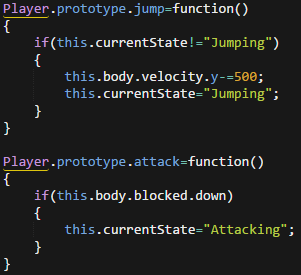


Para manejar las diferentes animaciones que deben reproducirse en diferentes momentos crearemos la variable “currentState”, la cual utilizaremos para realizar validaciones en las acciones del jugador.

Para mover al jugador, le debemos aplicar una fuerza al cuerpo físico que le declaramos previamente, ya sea en el eje X o Y.



Validaremos que el jugador no pueda realizar más de un salto a la vez y que sólo pueda atacar si se encuentra en el piso.



Imágenes de la demo:

