Framework 2D y herramientas

Jocs Electronics

⚠️ Disclaimer

Este framework ha sido desarrollado a lo largo de varios años para dar soporte a las asignaturas que se imparten sobre gráficos en la UPF.

Si bien puede ser usado sin ningún tipo de problema a nivel de licencia, lo cierto es que es muy poco eficiente y no está testeado en todo tipo de arquitecturas.

Por lo que no está recomendado para proyectos profesionales, sino más bien enfocado a un entorno más educativo.

Si tienes interés por hacer un juego profesional, mi consejo es que mires algún otro framework 2D en github.

Game

Hay una clase Game instanciada desde el main que encapsula toda la aplicación y contiene los métodos más relevantes que son:

- ctor: constructor para cargar assets y inicializar variables
- render: para pintar cada frame
- update: para actualizar el estado de las entidades del juego

Podéis crear tantos métodos como queráis en la clase Game pero recomiendo que creeis clases que encapsulan aquellas partes que tenga sentido instanciar varias veces (como personajes, items, etc)

Globales

En Game hay varias variables utiles:

- time te dice cuántos segundos han transcurridos desde que arrancó la app
- window_width y window_height te dice el tamaño de la ventana en pixeles

Otra variable importante es instance que es una variable estática que te permite acceder a la instancia de Game desde cualquier otro fichero de código:

```
Game::instance->time;
```

Image

Mirad todas las funciones de la clase Image, hay algunas muy útiles, y recordad que el framebuffer es de la clase Image:

```
image.getPixel(x,y);
image.setPixel(x,y,color);
image.blendPixel(x,y,color); //blends color with current pixel color using alpha

image.fill(Color(40, 40, 40)); //fills the image with one color
image.flipY(); //flips the image (there is also flipX)

image.drawLine( 0, 0, 100,100, Color::RED );        //draws a line
image.drawImage( font, 0, 0, font.getRect('A'-32,7,9) ); //draws a full image
image.drawImage( sprite, 0, 0, 20,20 ); //draws an scaled image
image.drawImage( sprite, x, y, Rect(0,0,14,18) ); //draws only a part of an image
image.drawText( "hello", 0, 0, font );   //draws text with a bitmapfont (assuming 7x9)

image.resize(w,h);   //resizes image without changing content
image.scale(w,h);    //resizes image and scales the content
image.crop(x,y,w,h); //crops the image to an area of it
image.getArea(x,y,w,h); //returns an image that contains only that area
```

Image

Podéis cargar y guardar imágenes con `loadTGA` y `saveTGA`. Guardar imágenes puede ser útil para haceros herramientas.

Si queréis convertir cualquier imagen a TGA usad el image convertor tool.

Recordad que podéis usar imágenes no solo para mostrar cosas en pantalla pero también para almacenar información sobre qué zonas del mapa son transitables o que *tile* va en cada celda.

Siempre precargad todas las imágenes al arranque de la aplicación para evitar parones durante la ejecución del juego (cargar datos de disco es lento!)

Framebuffer

Tenéis una instancia de Imagen en render que usamos como Framebuffer, donde pintamos lo que queremos que se vea por pantalla.

Al acabar mandamos enviarlo a pantalla usando showFramebuffer.

Aunque podeis trabajar directamente con el framebuffer e ir pintando cada imagen a mano, se recomienda crear clases que os abstraigan un poco de tener que gestionar el framebuffer directamente (por ejemplo la clase Sprite, la clase Camera, clase Map).

```
void Game::render(void)
{
	//do not change framebuffer size
	Image framebuffer(128, 128);

	//add your code here to fill the framebuffer
	//...

	framebuffer.fill( bgcolor );
	framebuffer.drawText( "Hello", 0, 0, font );

	//send image to screen
	showFramebuffer(&framebuffer);
}
```

Input

Si necesitáis el estado del input en algún momento, lo tenéis en la clase Input:

- keystate para saber qué teclas del teclado están pulsadas o lo han sido
- gamepad[n] para el estado de los mandos
- mouse_state para los botones del ratón
- mouse_position para la posición del ratón

También podéis usar los métodos:

- Input::isKeyPressed o Input::wasKeyPressed para saber si han sido pulsadas las teclas
- Input::gamepad[0].isButtonPressed(num) (o wasButtonPressed) para saber si un botón está pulsado.

Gamepad

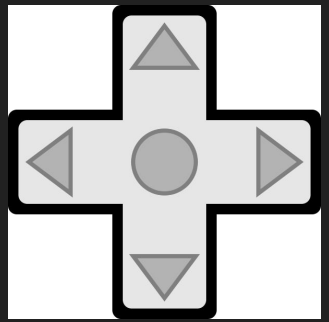
Toda la info del estado de los gamepads la tenéis en Input::gamepad[n] y eso incluye:

- buttons: El estado de qué botones están pulsados
- axis: El estado de los ejes, es decir, los sticks y triggers analogicos (entre -1 a +1)
- hat: El estado del HAT (d-pad)

Además por comodidad también guardo la variable direction que te dice en cuál de las ocho direcciones apunta el stick izquierdo.

Tambien podeis preguntar si un botón fue pulsado usando wasButtonPressed

HAT



Para saber en qué estado está el HAT podemos compararlo con:

- **`PAD_CENTERED`**: si no está en ninguna dirección
- **`PAD_LEFT`**, **`PAD_UP`**, **`PAD_RIGHT`**, **`PAD_DOWN`**
- **`PAD_LEFTUP`**, **`PAD_LEFTDOWN`**, **`PAD_RIGHTUP`**, **`PAD_RIGHTDOWN`**

Pero cada caso se guarda en un bit diferente de la variable, lo que permite fácilmente comparar si está el UP pulsado sin importar si es LEFTUP, UP o RIGHTUP, basta hacer una comparacion AND:

```
if( Input::gamepad[0].hat & PAD_UP )
```

Esto también aplica a la variable direction que referencia al stick izquierdo

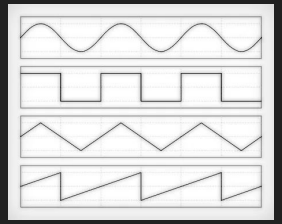
Audio

En el framework hay un pequeño sistema para reproducir sonidos.

Si activamos el sonido llamando a enableSound entonces el sistema llama a la funcion onAudio cada vez que necesita una nueva ventana (buffer) con las muestras de audio para reproducir.

Gestionar la reproducción de audio mediante buffers periódicamente es bastante complicado, así que para simplificar la tarea el framework incluye la clase Synth que gestiona el buffer automáticamente, pero solo permite reproducir sonidos de baja calidad y con ciertas limitaciones.

Synth



La clase Synth contiene 3 osciladores de los que podemos controlar tanto la frecuencia como la amplitud o la forma de onda, para producir sonidos muy básicos.

```
synth.osc1.freq = 440;  //440 Hz is a A note
synth.osc1.amplitude = 0.5;  //0 mute, 1 max volume
synth.osc1.wave = Synth::SQR; //SIN, SQR or SAW
```

Y para controlar el volumen global o la cantidad de ruido blanco que reproducimos:

```
synth.volume = 0.4;
synth.noise_volume = 0.4;
```

Así que si queremos generar sonidos tenemos que modificar las propiedades a lo largo del tiempo, en sincronía con el juego, una tarea muy compleja.

Samples

Otra manera más simple de reproducir sonido es utilizar samples cargados de disco. La clase Synth permite cargar archivos WAV siempre que sean mono, su sampling rate sea 48000 y sean 16bitsPCM o 32bits Float.

```
synth.playSample("data/jump.wav",1,false); //filename, volume, loop
```

Esta función carga (si no se había cargado ya) y reproduce un wav en baja calidad, podemos controlar tanto el volumen como si queremos que se reproduzca en loop.

La función retorna una estructura del tipo SamplePlayback que podemos usar para silenciar un sonido (llamando al método stop de dicha clase).

Herramientas útiles

Editores de imagenes

Es importante dominar al menos un editor de imágenes, no es tan importante cual, sino conocerlo en profundidad, para poder crear rápidamente contenido.

Para hacer pixel art vale cualquier herramienta, pero hay algunas más adaptadas:

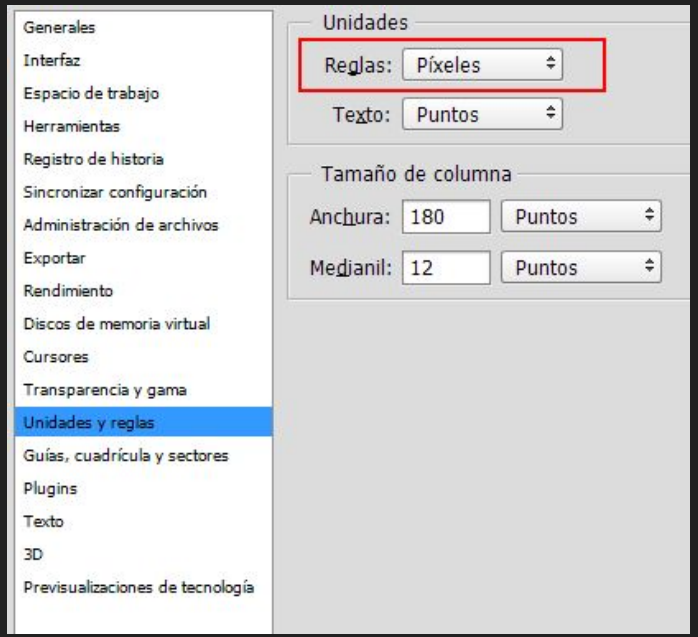
- Photoshop, es una herramienta muy potente pero puede ser demasiado compleja para gente que está empezando.
- ASEprite facil de usar, ideal para animaciones
- Pro Motion util para crear sprites, animaciones, incluso tilemaps
- Rogueditor: para editar tilemaps desde la web

En adelante miraremos como usar photoshop para crear sprites ya que es una herramienta muy completa que mucha gente conoce.

Photoshop para editar Sprites

Photoshop no fue creado para trabajar con pixel art, por eso hay que configurar algunos settings.

Primero, ir a preferencias y en Unidades y reglas, cambiar el valor de Reglas a píxeles, así toda la información que se muestra en los diferentes paneles usarán como unidad de medida los pixeles.

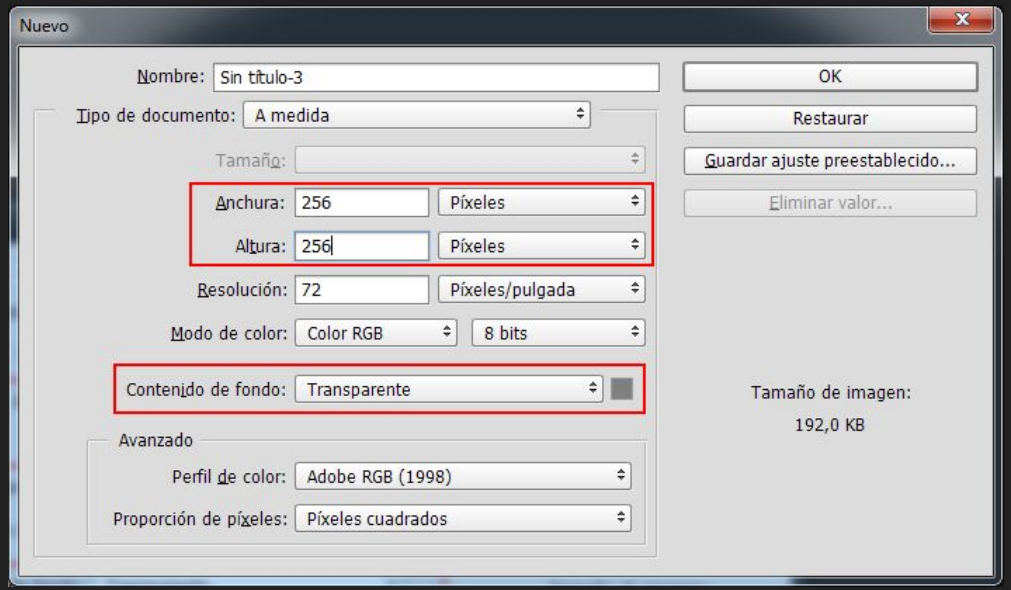


Photoshop: crear imagen

Elige el tamaño de la imagen en pixeles, intenta siempre que sea potencia de 2 (256,512,1024). Hace más fácil luego dividir la imagen en fragmentos regulares.

Cuando crees una imagen nueva que deba tener píxeles transparentes, elige en Contenido de fondo: transparente

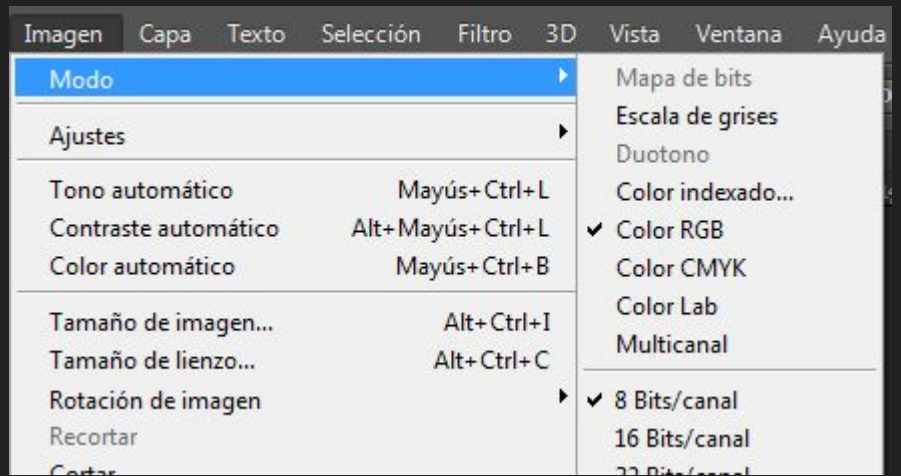
Y siempre usa Modo de color RGB



Photoshop, trabajar en modo RGB

Cuando abras una imagen de internet asegurate de que está en modo RGB y no en modo indexado, de lo contrario no podrás editar correctamente las capas.

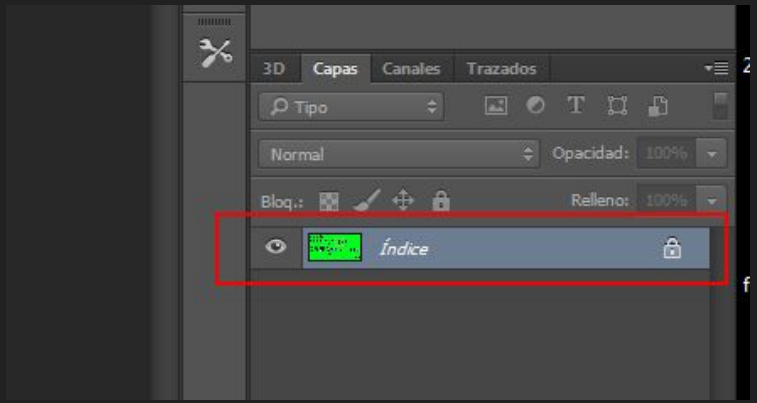
Para ello ve a **`Imagen -> Modo -> Color RGB`**



Photoshop transparencia

Si al borrar de una capa (pulsando la tecla suprimir sobre una selección) vemos que en lugar de aparecer píxeles transparentes se quedan con un color plano, entonces es que nuestra capa no tiene la transparencia activada.

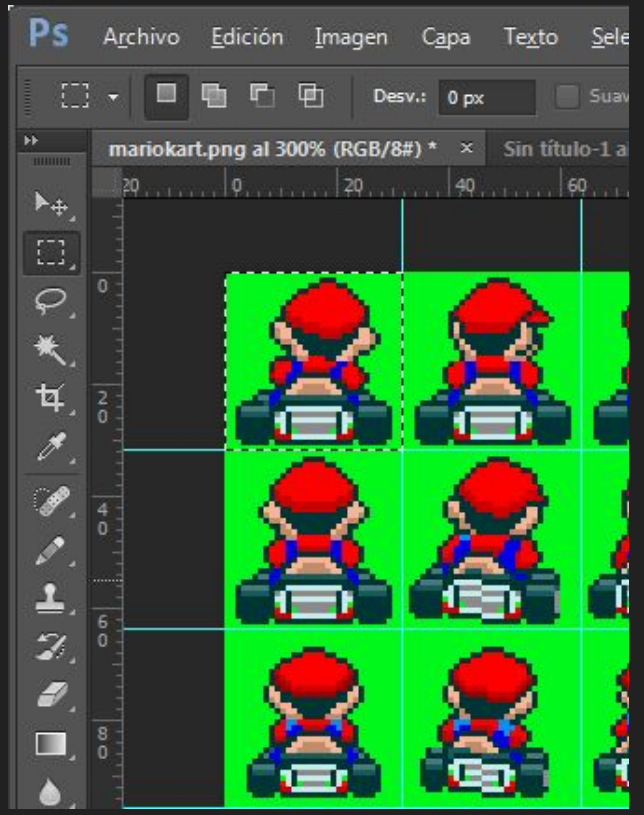
Si te fijas verás que la capa tiene un candado, basta hacer doble click y darle a aceptar para eliminar el candado y que la capa ya permite transparencia. Si no funciona asegurate de estar en modo color RGB.



Photoshop para Spritesheets

Cuando quieras crear sprite sheets, usa las reglas de photoshop para alinear correctamente la rejilla. Ve al menú **`Vista -> Reglas`**

Para añadir reglas perfectamente alineadas, primero crea un recuadro de selección con el tamaño que quieras, y luego una vez correcto, arrastra la regla para crear una guia, y alineala con la selección que has hecho.



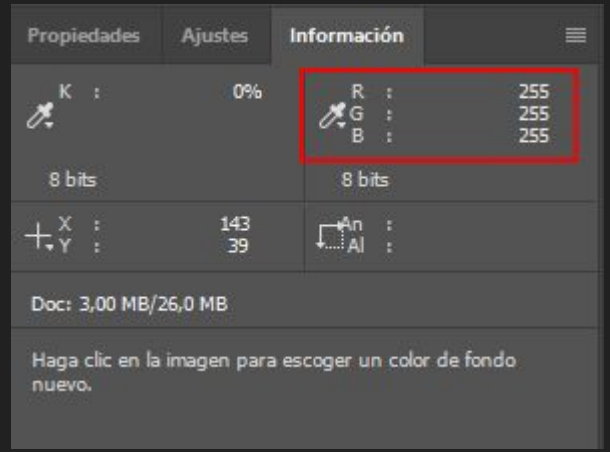
Photoshop para crear mapas

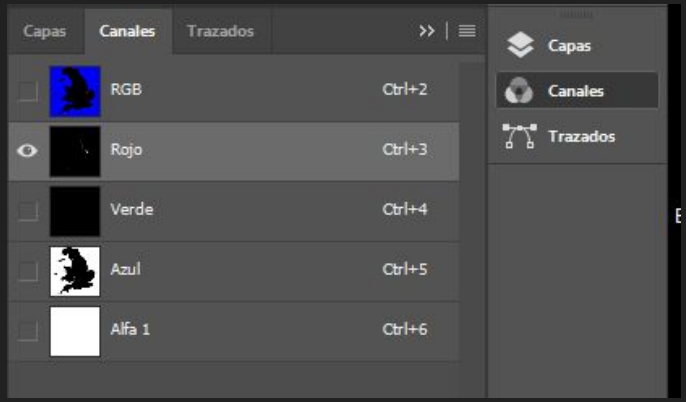
Podeis usar photoshop para crear imágenes que contengan información útil para vuestro juego (qué zona son transitables, que items hay, etc).

Si necesitáis ver el valor RGB de un pixel podeis mirarlo desde el menú de Información. Para activarlo basta ir a **`Ventana->Información`**

Si no sale el color en RGB pulsad sobre la pipeta de la derecha que hay dentro del menú y elegid color RGB.

También puedes mostrar un canal solo de la imagen para visualizar la información concreta de un canal usando el menu de canales de photoshop.





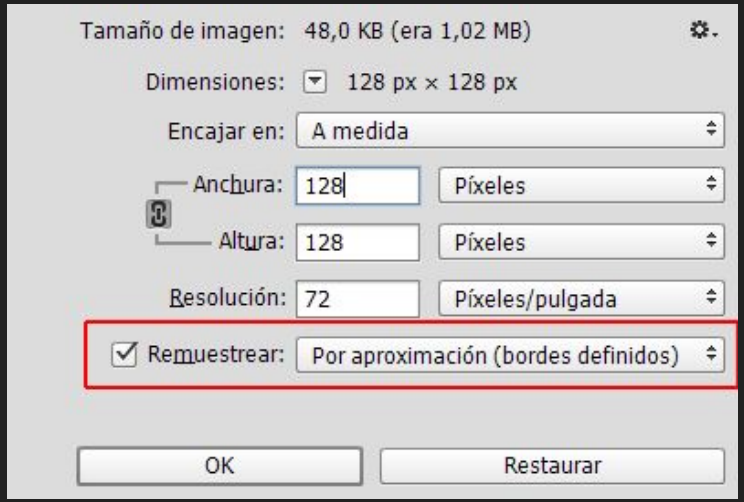
Photoshop: pixel art

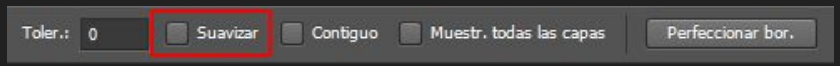
Usar estética pixel-art nos puede facilitar el trabajo ya que simplifica la tarea de crear arte para el juego.

Para ello hemos de asegurarnos de que no hayan píxeles borrosos o semitransparentes.

Cuando escalamos imagenes, Photoshop por defecto aplica un filtrado bilineal, por ello es recomendable siempre desactivar el filtrado.

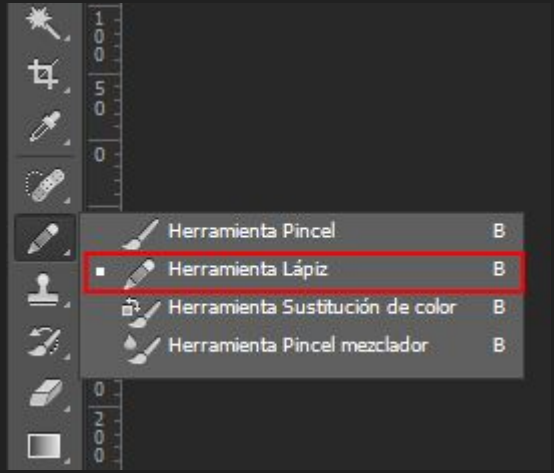
También es importante desactivar el antialiasing de las fuentes, y evitar hacer selecciones suavizadas.





Photoshop: pintando

Cuando vayas a pintar pixel a pixel, usa la herramienta del lápiz, así evitas que haya zonas semitransparentes.



Cuando selecciones áreas de color con la varita mágica, desactiva de la barra superior la opción suavizar para que la selección no tenga antialiasing.

También pon la tolerancia a 0 para que solo seleccione píxeles con el mismo color.

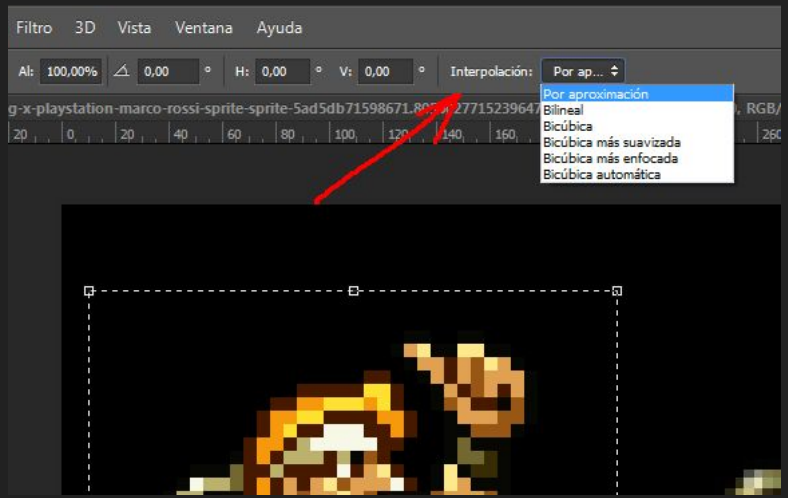
Lo mismo aplica también para el cubo de pintura.

Escalado de imagenes

Cuando trabajamos con resoluciones bajas y pixelart mucho cuidado a la hora de escalar una imagen. Si la escala no es múltiplo de la imagen original (x2, x3, x4 …) los pixeles se veran borrosos.

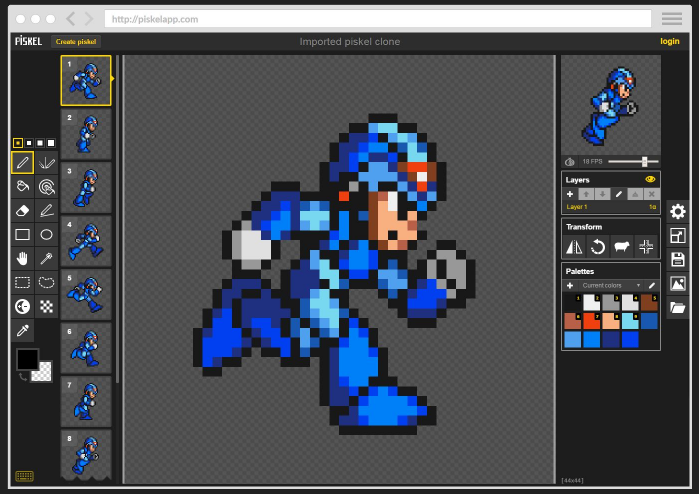
Tambien es importante desactivar la interpolación de photoshop para evitar zonas borrosas. Para ello elige la opción “Aproximación”.

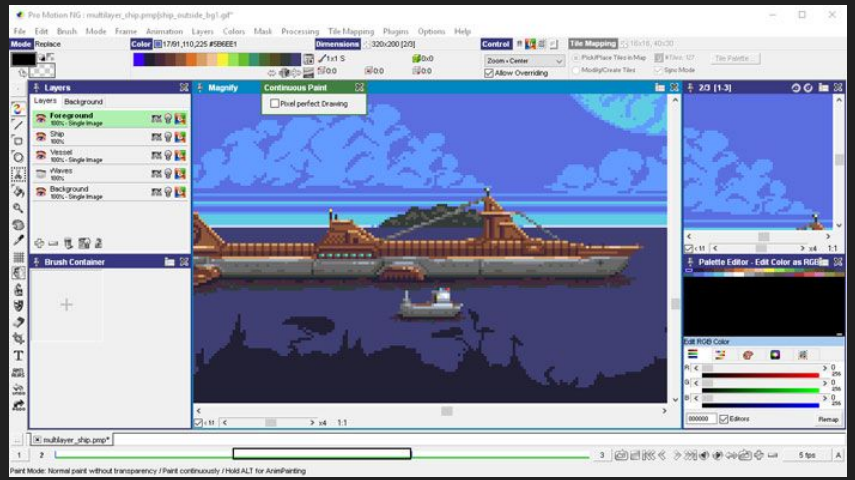
Si usas un programa especifico para pixel art no tendras estos problemas.



Animaciones

De cara a crear animaciones mejor usar herramientas como Piskel, Pro Motion, o ASEprite que os hacen el trabajo más fácil.

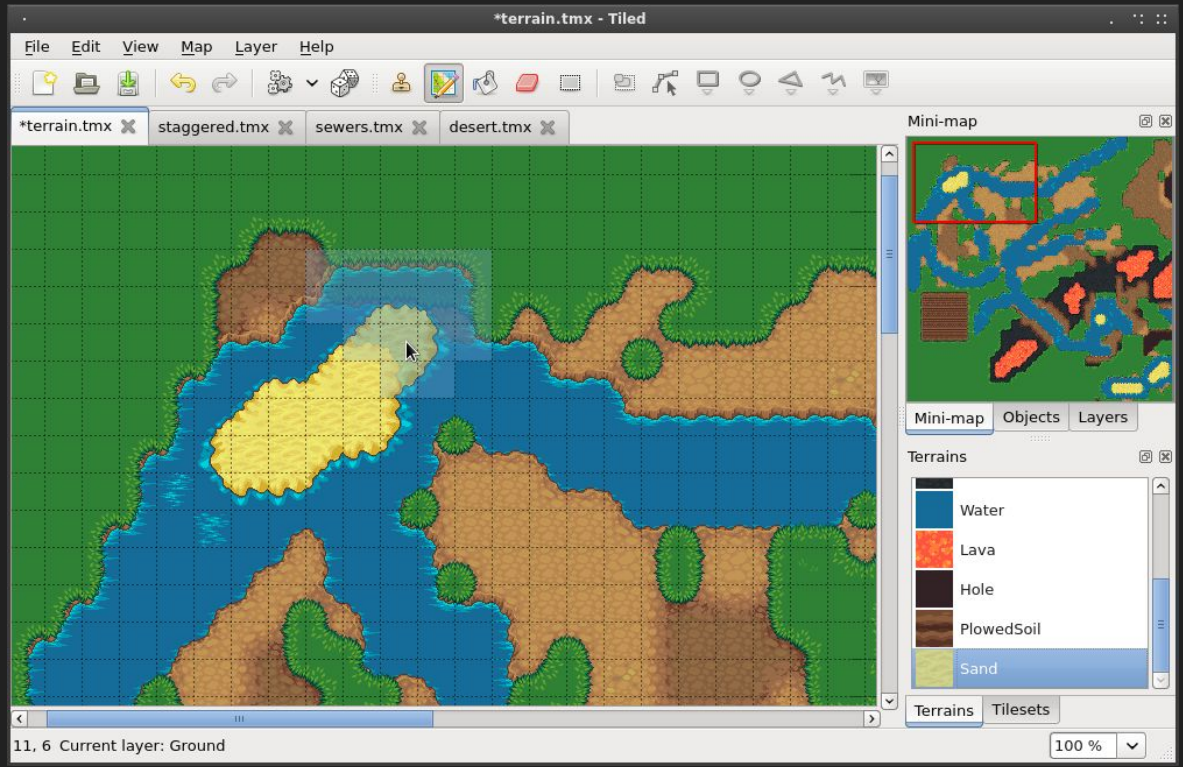




Tilemaps

Para definir los tilemaps hay herramientas como Tiled o Roqueditor

Sin embargo, a veces puede ser útil tener la opción desde dentro del juego de modificar los tiles, de esta manera puedes usarlo tanto para gameplay como para hacer editor de niveles.



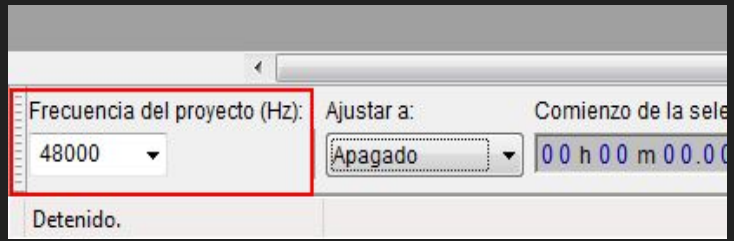
Audacity

Para reproducir audios en Synth hemos de garantizar que cumple el formato adecuado que es mono, sampling rate de 48000 y Float32 o 16PCM

En Audacity podemos convertir los archivos paso a paso:

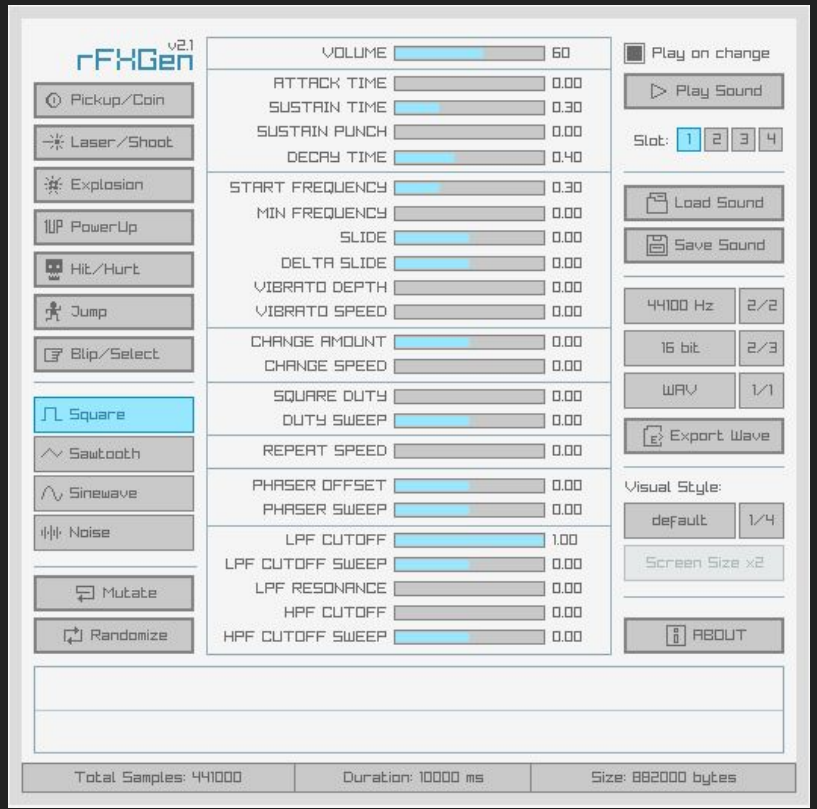
Para pasar a Mono elige la opción **`Pistas -> Pista stereo a mono`**

Para convertir a 48000, primero elige el menu **`Pistas -> Remuestrear: 48000`**
Seguido de cambiar el modo del proyecto en la barra inferior a 48000



Y guardar a disco como WAV especificando 32 Bits flotante o 16 bits con signo

Sonidos 8bits



Links de interes

- Basic 2D platformer tutorial
- Game development tricks list
- Game Programming Patterns
- The Art of Screenshake
- Writing a retro 3D FPS y Raycasting tutorial