Videojuegos retro en Gameboy ... de verdad!

(Parte 2)

Enzo Barbaguelatta D.

MHW

Escuela de Ingeniería Informática

¿Conseguiste pilas?

Recapitulación de ayer

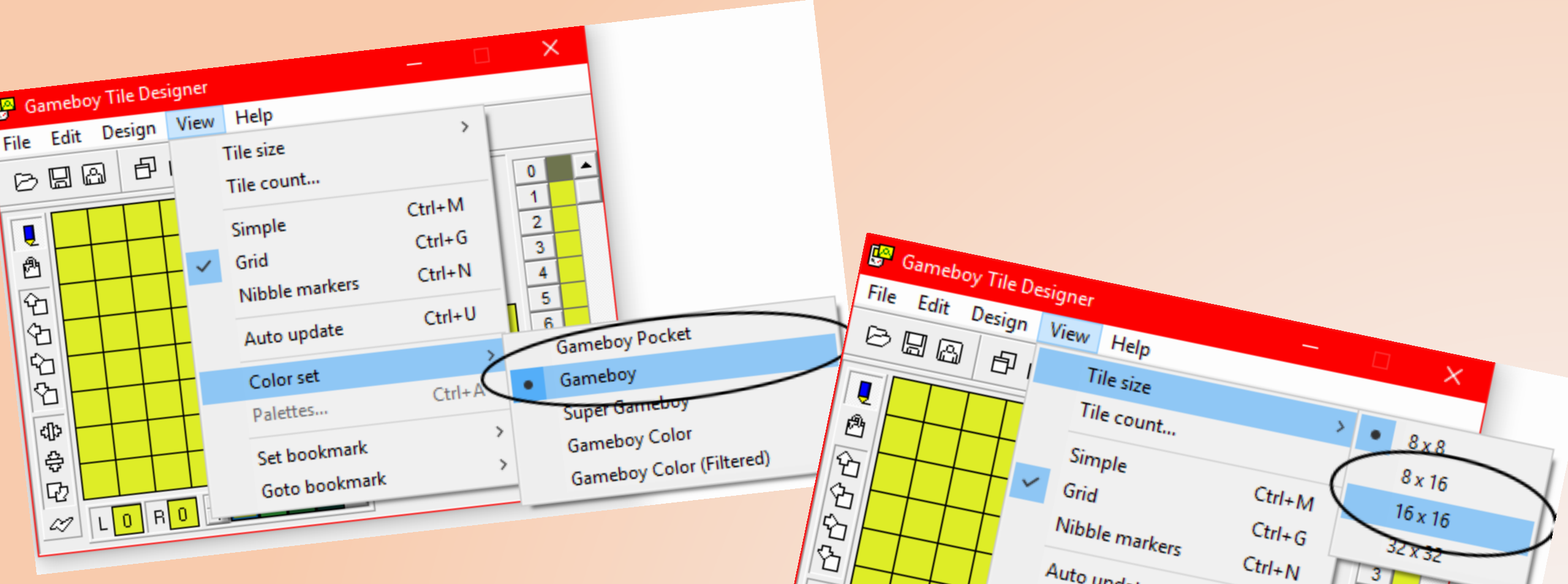
- Una latera y aburrida introducción sobre que es la gameboy, la escena homebrew y haaaaaaaartos videos.
- Bajamos e Instalamos GBDK con ZGB mas un sinfín de herramientas raras.
- Compilamos un hola mundo todo emocionados e inocentes.
- Lloramos porque no nos compiló a la primera
- Conocimos la estructura de un proyecto de ZGB
- Conocimos la magia de las ROM Banks
- Nos quedamos con la ilusión de aprender mas

- La sesión de hoy será mucho mas practica que la de ayer.
- No tanto relleno, mas echar a perder cosas! :D
- Porque … ¿lo que queremos al final es hacer juegos y ser felices, no?
- Preparen sus computadores, sus kits de herramientas locas y sus banderas de Japón (?), haremos hartas cosas hoy!

Hey, si no viniste ayer, puedes echar un vistazo a la PPT del día anterior. Esta sin los videos, pero algo es algo. Las slides son autoexplicables asi que lo seguirás sin mayor ayuda … Mientras, descargate el pack que aparece allí para que te pongas al día. La ppt está en http://elsemieni.net/gb/

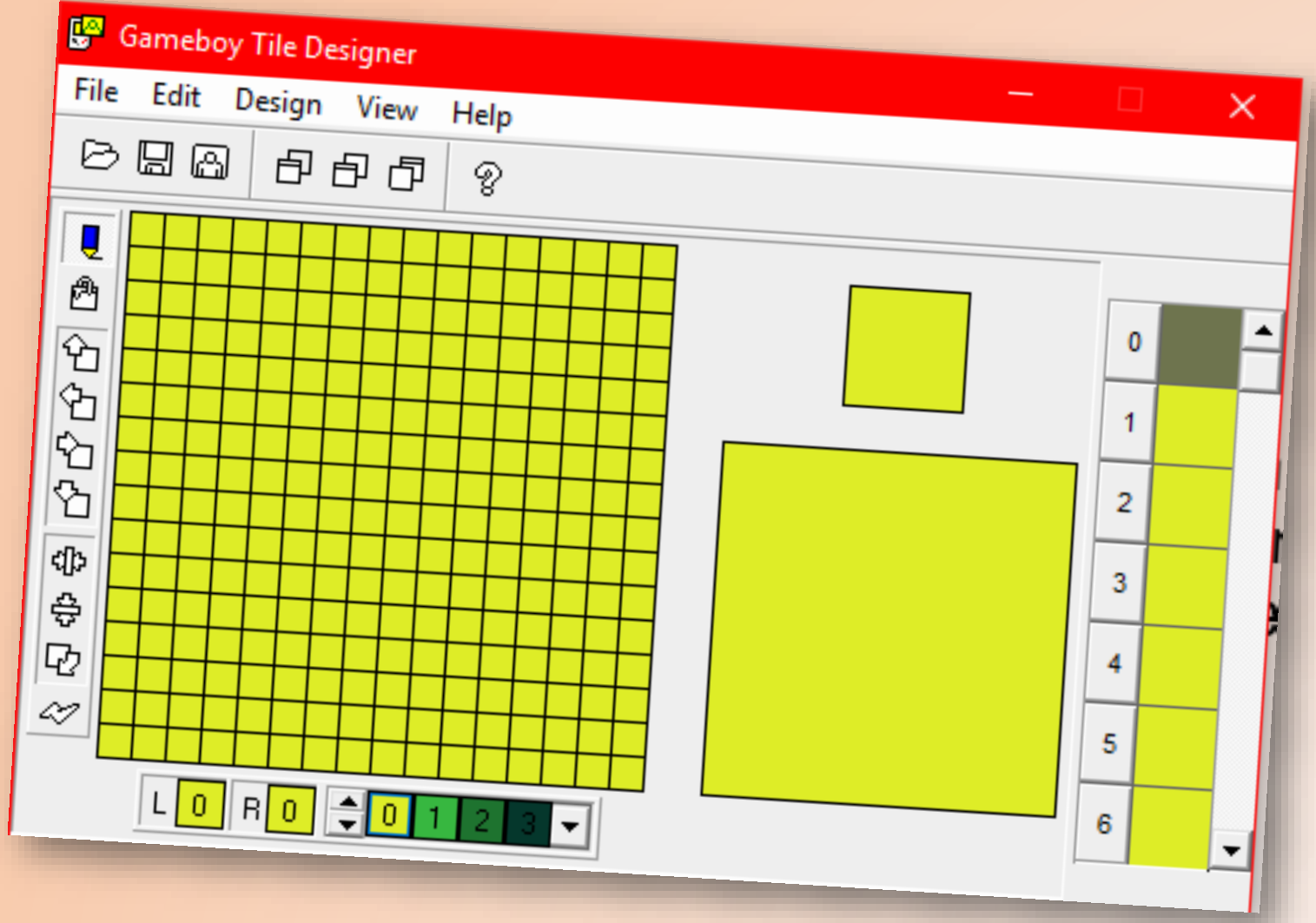
Sprites!

- No, no es una Sprite
- En español chilensis: Los monos en pantalla!
- ZGB soporta Sprites de **8x16** pixeles y **16x16** pixeles.
- Limitaciones raras: **No intentes poner mas de 20 sprites ... Te irá mal.**
- Los sprites se crean con la herramienta **GB Tile Designer** (en la carpeta **/tools/gbtd22**)



Seteando el editor

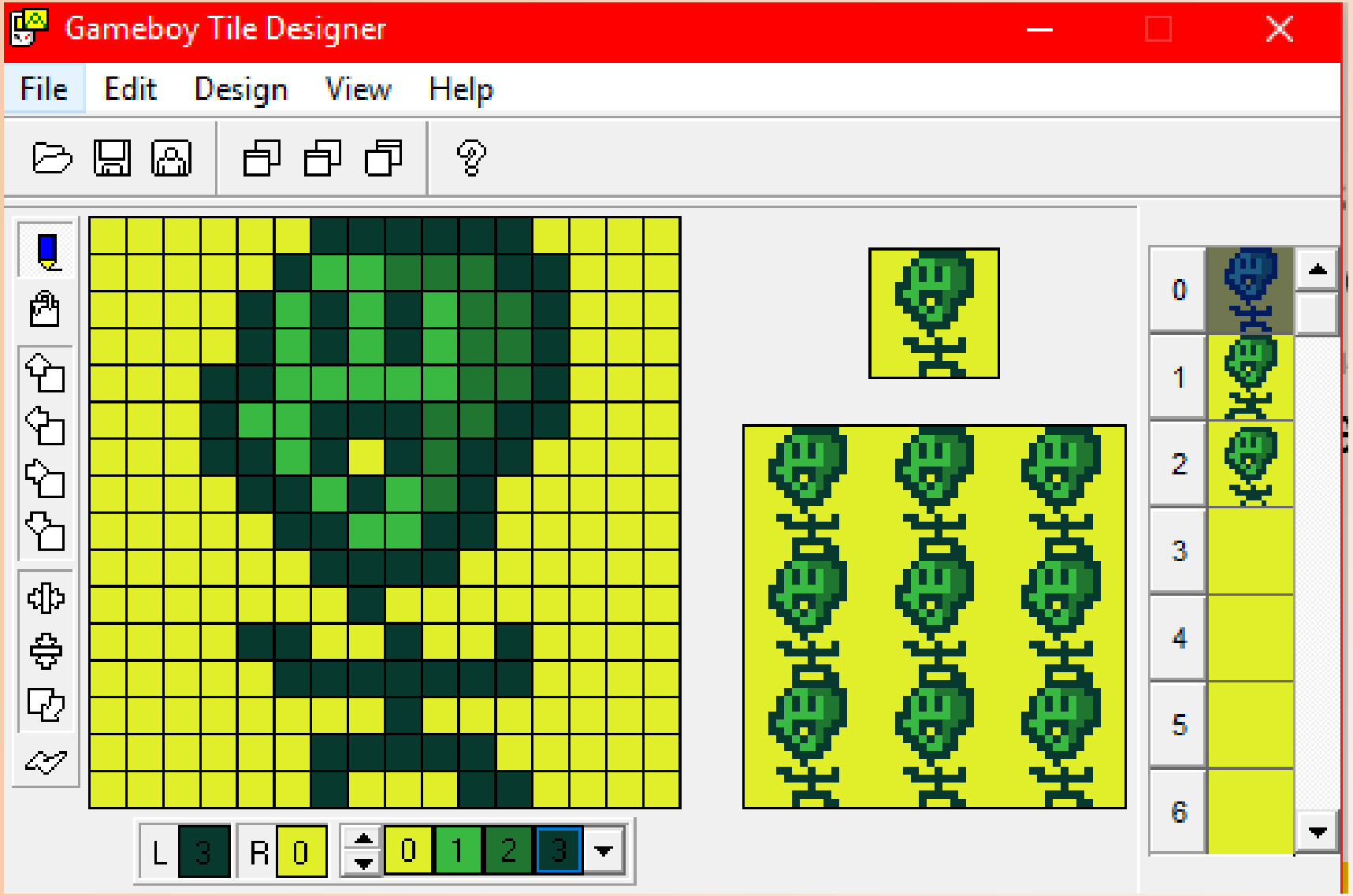
Herramientitas
Para pintar y
correr el dibujo



Puedes pintar hartas texturas en un archivo (por si quieres hacer animaciones). El limite depende del tamaño

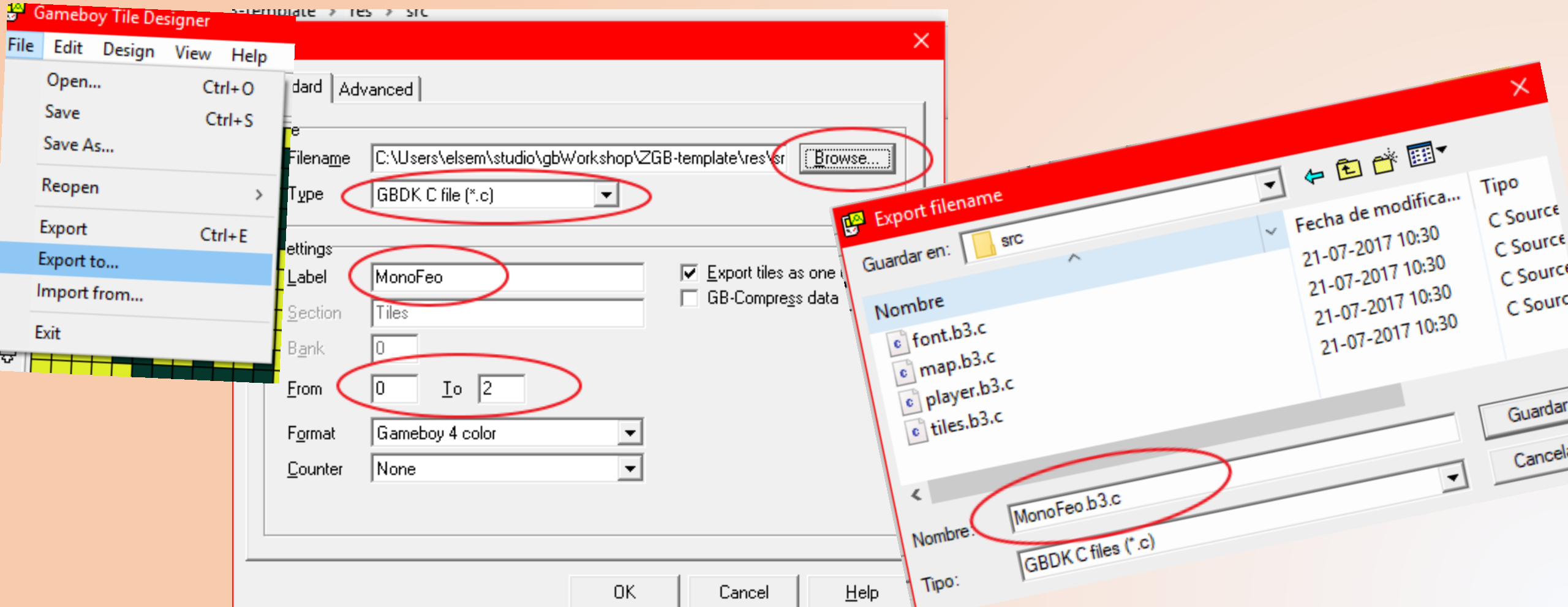
Una fabulosa paleta de 4 colores! xD

NOTA: También puedes copypastear desde tu editor de pixel art preferido… pero te advierto desde ya que el editor ES webiado con los colores. Mi consejo: trata de igualar los colores lo mas exacto posibles para que te pesque los colores.



Hagamos un monito feo con 3 frames de animacion

- Al exportar, ponlo como **archivo GBDK** (es el kit que usamos al fin y al cabo, no?)
- Ponle un **nombre** en Label
- Guárdalo en **/res/src** como **<nombre>.b3.c** (donde **3** es el banco que quieres)
- From-To Pon desde 0 hasta el **numero de texturas que hayas hecho**. En este caso **2** (Recuerda, se empieza desde 0!)



En **/res/src** tendremos **MonoFeo.b3.c** y **MonoFeo.h**

Es nuestro Sprite compilado! :O

```
unsigned char MonoFeo[] =
{
  0x03,0x03,0x07,0x04,0x0F,0x0A,0x0F,0x0A,
  0x1F,0x18,0x1F,0x13,0x1E,0x1A,0x0F,0x0D,
  0x07,0x06,0x03,0x03,0x01,0x01,0x0C,0x0C,
  0x07,0x07,0x00,0x00,0x03,0x03,0x02,0x02,
  0xF0,0xF0,0x18,0xF8,0xC8,0xB8,0xC8,0xB8,
  0xC8,0x38,0x98,0xF8,0xB0,0xF0,0xA0,0x60,
  0xE0,0x60,0xC0,0xC0,0x00,0x00,0x90,0x90,
  0xF0,0xF0,0x80,0x80,0xE0,0xE0,0x30,0x30,
  0x03,0x03,0x07,0x04,0x0F,0x0A,0x0F,0x0A,
  0x1F,0x18,0x1F,0x13,0x1E,0x1A,0x0F,0x0D,
  0x07,0x06,0x03,0x03,0x01,0x01,0x0C,0x0C,
  0x07,0x07,0x01,0x01,0x0F,0x0F,0x18,0x18,
  0xF0,0xF0,0x18,0xF8,0xC8,0xB8,0xC8,0xB8,
  0xC8,0x38,0x98,0xF8,0xB0,0xF0,0xA0,0x60,
  0xE0,0x60,0xC0,0xC0,0x00,0x00,0x90,0x90,
  0xF0,0xF0,0x80,0x80,0xC0,0xC0,0x60,0x60,
  0x00,0x00,0x03,0x03,0x07,0x04,0x0F,0x0A,
  0x0F,0x0A,0x1F,0x18,0x1F,0x13,0x1E,0x1A,
  0x0F,0x0D,0x07,0x06,0x03,0x03,0x01,0x01,
  0x0C,0x0C,0x07,0x07,0x01,0x01,0x02,0x02,
  0x00,0x00,0xF0,0xF0,0x18,0xF8,0xC8,0xB8,
  0xC8,0xB8,0xC8,0x38,0x98,0xF8,0xB0,0xF0,
  0xA0,0x60,0xE0,0x60,0xC0,0xC0,0x00,0x00,
  0x90,0x90,0xF0,0xF0,0xC0,0xC0,0x20,0x20
};
```

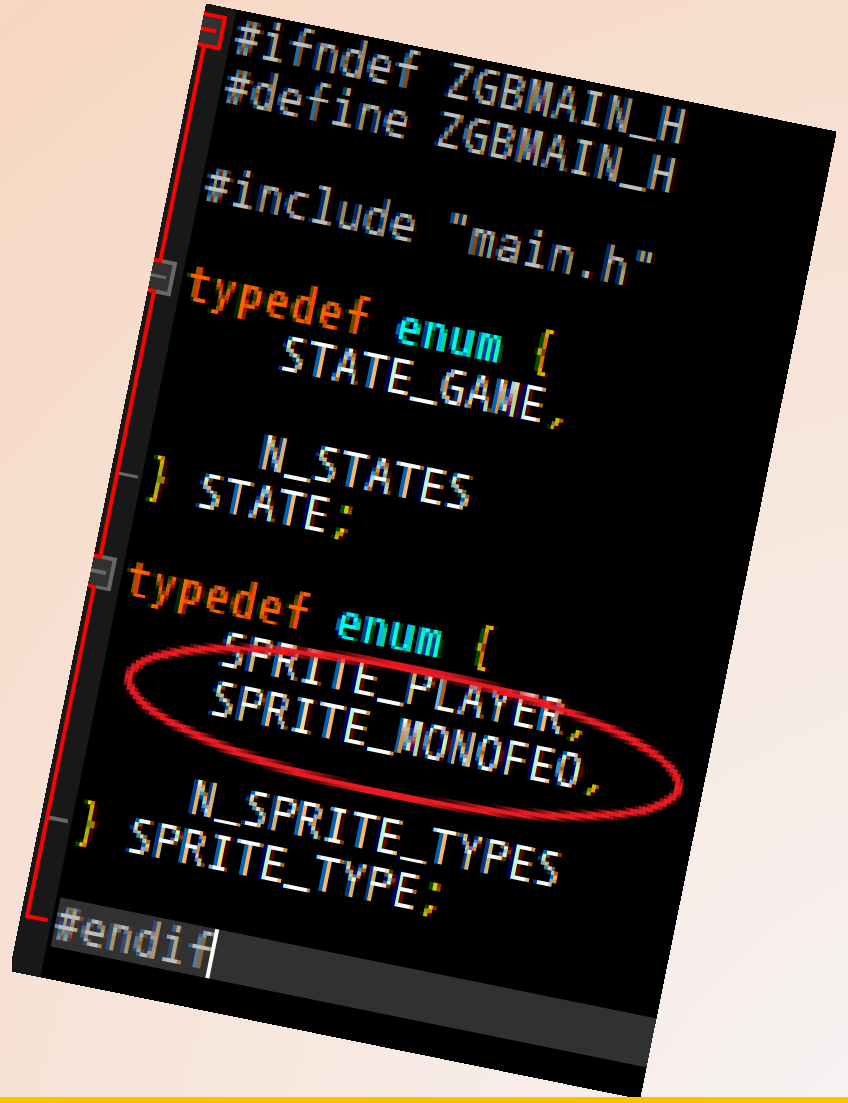
```
/* Bank of tiles. */
#define MonoFeoBank 0
/* Start of tile array. */
extern unsigned char MonoFeo[];

/* End of MONOFEO.H */
```

Pero... no nos sirve así suelto. Hay que integrarlo al juego!

- En /include/ crear **SpriteMonoFeo.h**

```
#ifndef SPRITE_MONOFEO_H
#define SPRITE_MONOFEO_H

#include "main.h"

DECLARE_SPRITE(SPRITE_MONOFEO);

#endif
```

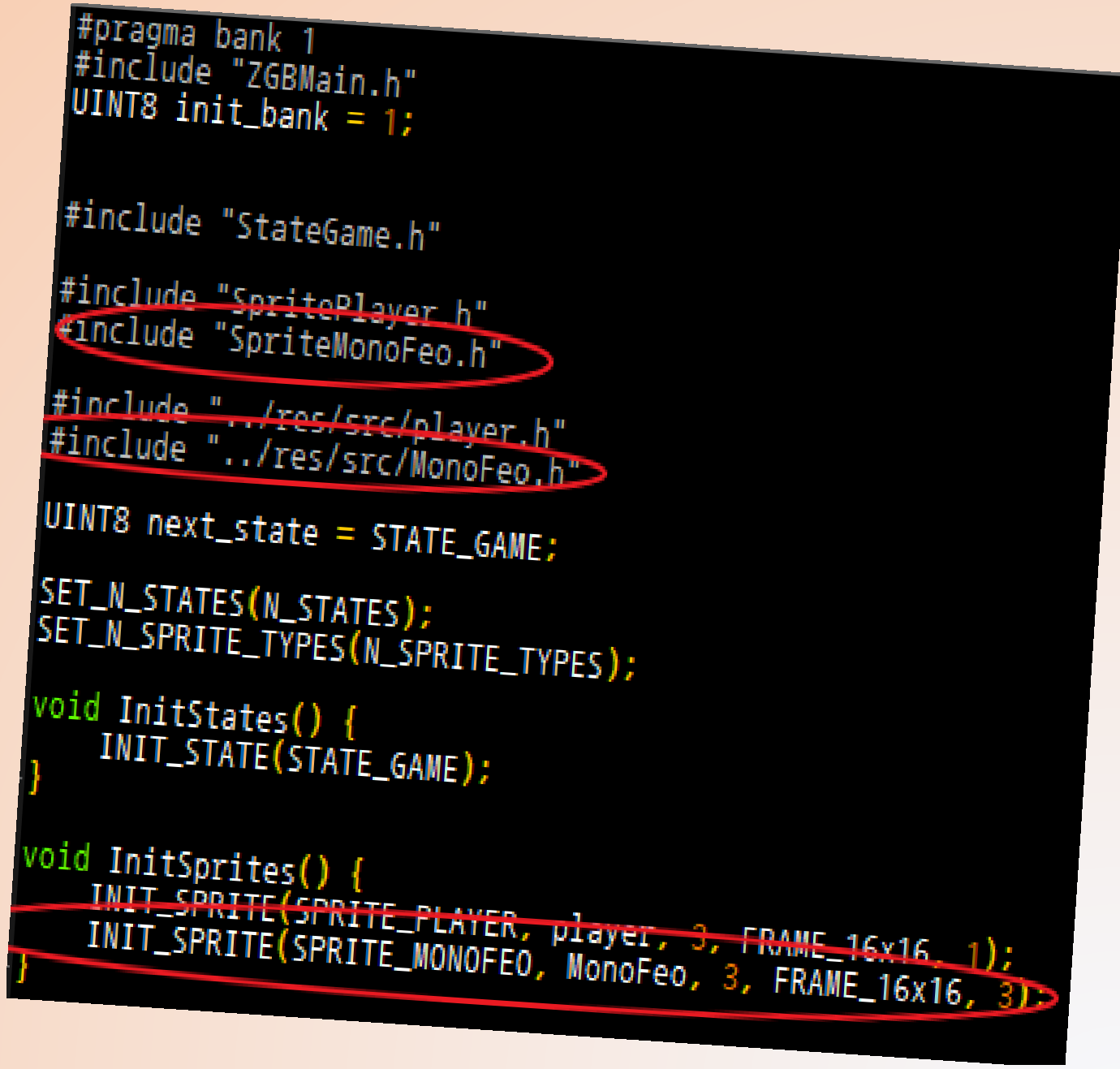
- Agrega nuestro **SPRITE_MONOFEO** a **/include/ZGBMain.h**



NOTA: Si te da mucha lata escribir eso, puedes basarte en los que ya existen en el template
Asegúrate si de cambiar todas las referencias, porque si no compila, 130% que es por eso.

Vayamos a /src/ y agregar lo siguiente a **ZGBMain_Init.c**

- El include que creamos recién, así como también el include de la textura compilada.
- En **InitSprites** (donde la gameboy procesa nuestros sprites), colar nuestro mono feo.
- Los parámetros son: **SPRITE_MONOFEO**, El nombre de la textura, el banco donde lo pusimos (**el 3** en este caso), el tamaño, y cuantos frames pusimos (**3** también).

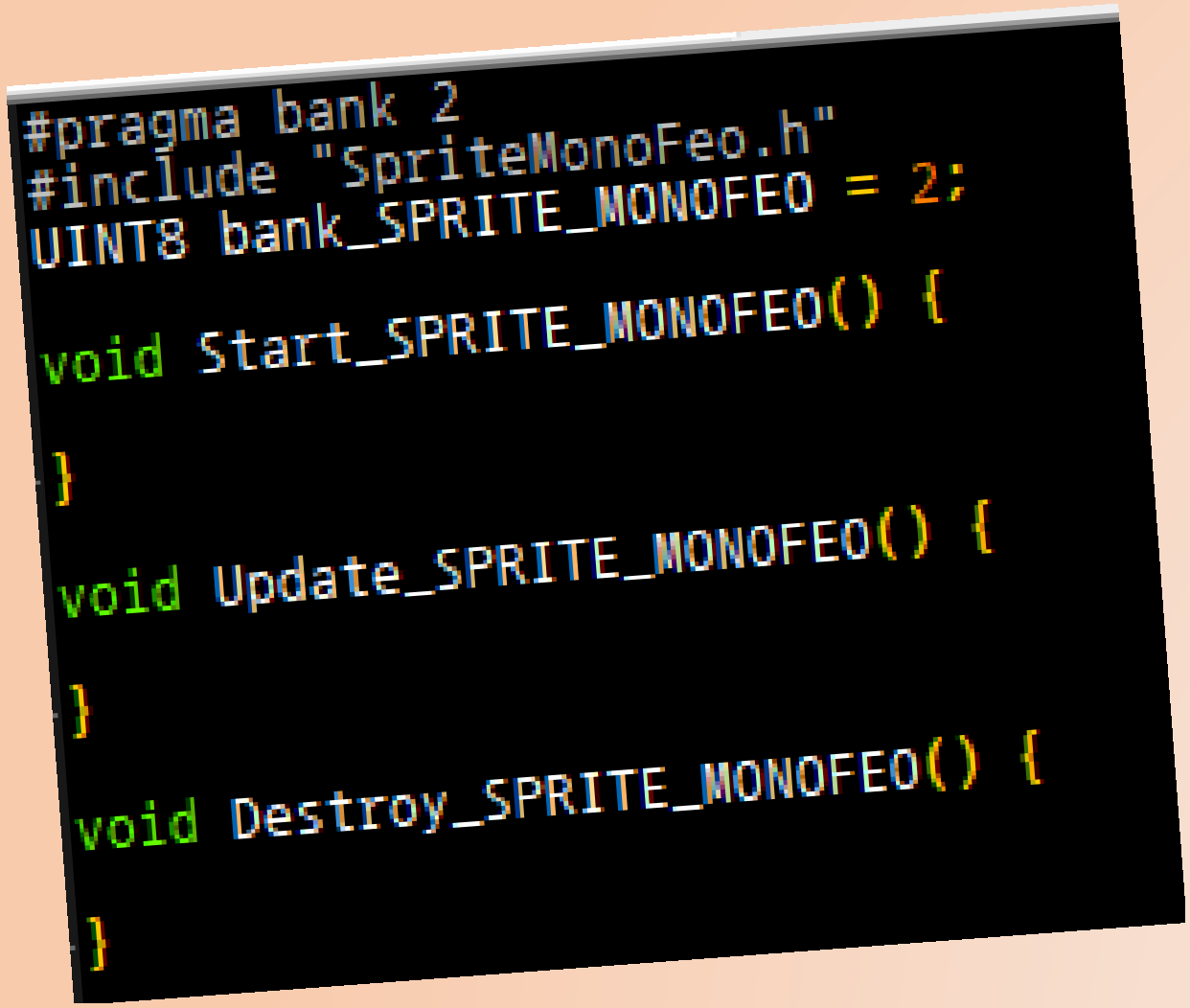


- Aquí mismo, en **/include/** creamos **SpriteMonoFeo.c**
- Lo rellenamos con esto, no te olvides de setear bien los nombres y bancos.

```
#pragma bank 2
#include "SpriteMonoFeo.h"
UINT8 bank_SPRITE_MONOFEO = 2;

void Start_SPRITE_MONOFEO() {

}

void Update_SPRITE_MONOFEO() {

}

void Destroy_SPRITE_MONOFEO() {

}
```

- **Waaaaaaaait! Banco 2? No era que estábamos en el banco 3?**

- **Si! Pero en el banco 3 guardábamos la textura. Aquí guardamos el código, y el código lo pondremos en el banco 2, ya? ☺**



```
#pragma bank 2
#include "SpriteMonoFeo.h"
UINT8 bank_SPRITE_MONOFEO = 2;

void Start_SPRITE_MONOFEO() {

}

void Update_SPRITE_MONOFEO() {

}

void Destroy_SPRITE_MONOFEO() {

}
```

Este código se explica bastante intutivamente:

- La función **create** se llama cuando **se crea** el MonoFeo en el juego.
- La función **update** se llama **una vez cada frame** (o sea, siempre). Aquí es donde va toda la gracia!
- La función **destroy** se llama cuando… **se destruye** la wea xD

No hemos terminado aun xD

- Falta lo mas importante: Integrarlo al escenario po!
- Vamos a **StateGame.c** (nuestro escenario)
- Mira donde marqué. **Comenta la línea donde sale scroll_target** (quitaremos así el Sprite de gameboy de la pantalla).
- **Agrega el código que puse ahí.** Al MonoFeo y al Player (es la gameboy fea esa).
- **El 10, 10 son las coordenadas** donde estará en el escenario.
- Que hace scroll_target? **La cámara del juego siempre enfocará al Sprite que tu le indiques**... en este caso al MonoFeo.

```
#pragma bank 2
#include "StateGame.h"
UINT8 bank_STATE_GAME = 2;

#include "..\res\src\tiles.h"
#include "..\res\src\map.h"

#include "ZGBMain.h"
#include "Scroll.h"
#include "SpriteManager.h"

extern UINT8 n_sprite_types;
void Start_STATE_GAME() {
    UINT8 i;

    SPRITES_8x16;
    for(i = 0; i != n_sprite_types; ++ i) {
        SpriteManagerLoad(i);
    }
    SHOW_SPRITES;

    //scroll_target = SpriteManagerAdd(SPRITE_PLAYER, 50, 50);
    scroll_target = SpriteManagerAdd(SPRITE_MONOFEO, 10, 10);
    SpriteManagerAdd(SPRITE_PLAYER, 50, 50);

    InitScrollTiles(0, 2, tiles, 3);
    InitScroll(mapWidth, mapHeight, map, 0, 0, 3);
    SHOW_BKG;
}

void Update_STATE_GAME() {
}
```

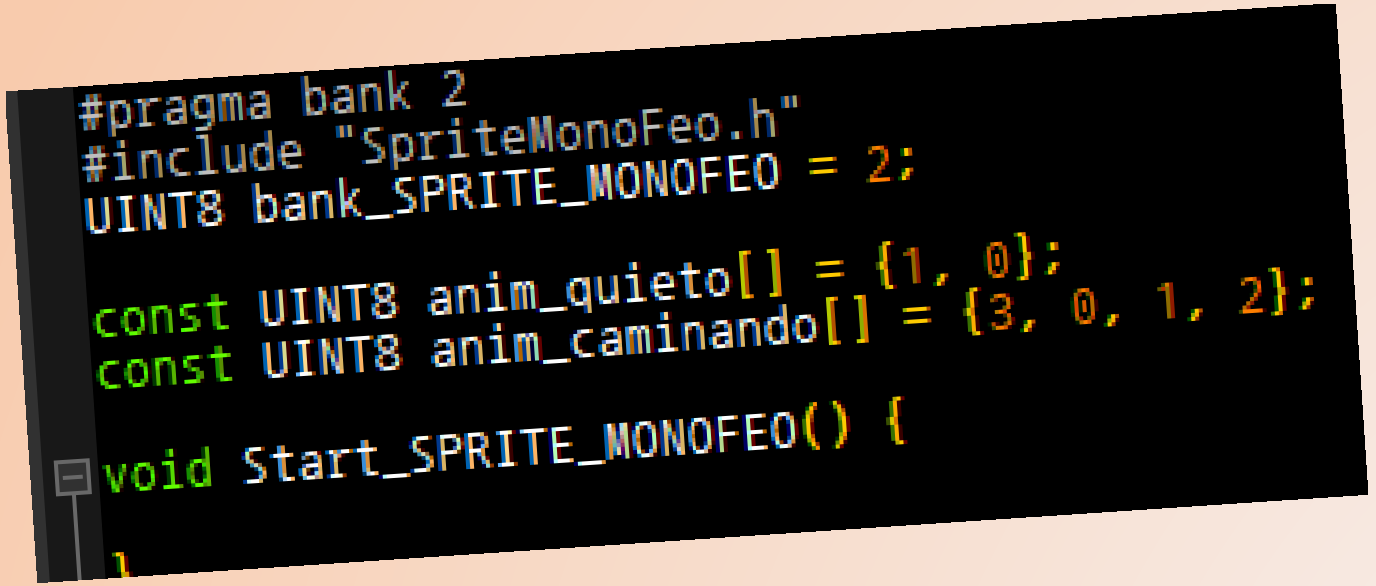
¿Compilemos?

Felicidades, un mono mas feo que la cresta! xD

Pero no hace niuna wea

Hagamos que el MonoFeo sea mas entrete

- Animémoslo? Volvamos a **SpriteMonoFeo.c** Agrégale esto



```
#pragma bank 2
#include "SpriteMonoFeo.h"
UINT8 bank_SPRITE_MONOFEO = 2;

const UINT8 anim_quieto[] = {1, 0};
const UINT8 anim_caminando[] = {3, 0, 1, 2};

void Start_SPRITE_MONOFEO() {

}
```

- Agregamos 2 animaciones: **anim_quieto** y **anim_caminando**.
- El primer numero de arreglo es la cantidad de números del arreglo (recuerda que en C necesitas saber el largo, jeje), y los siguientes números, el código de frame de la animación. (Recuerdas cuales eran?)

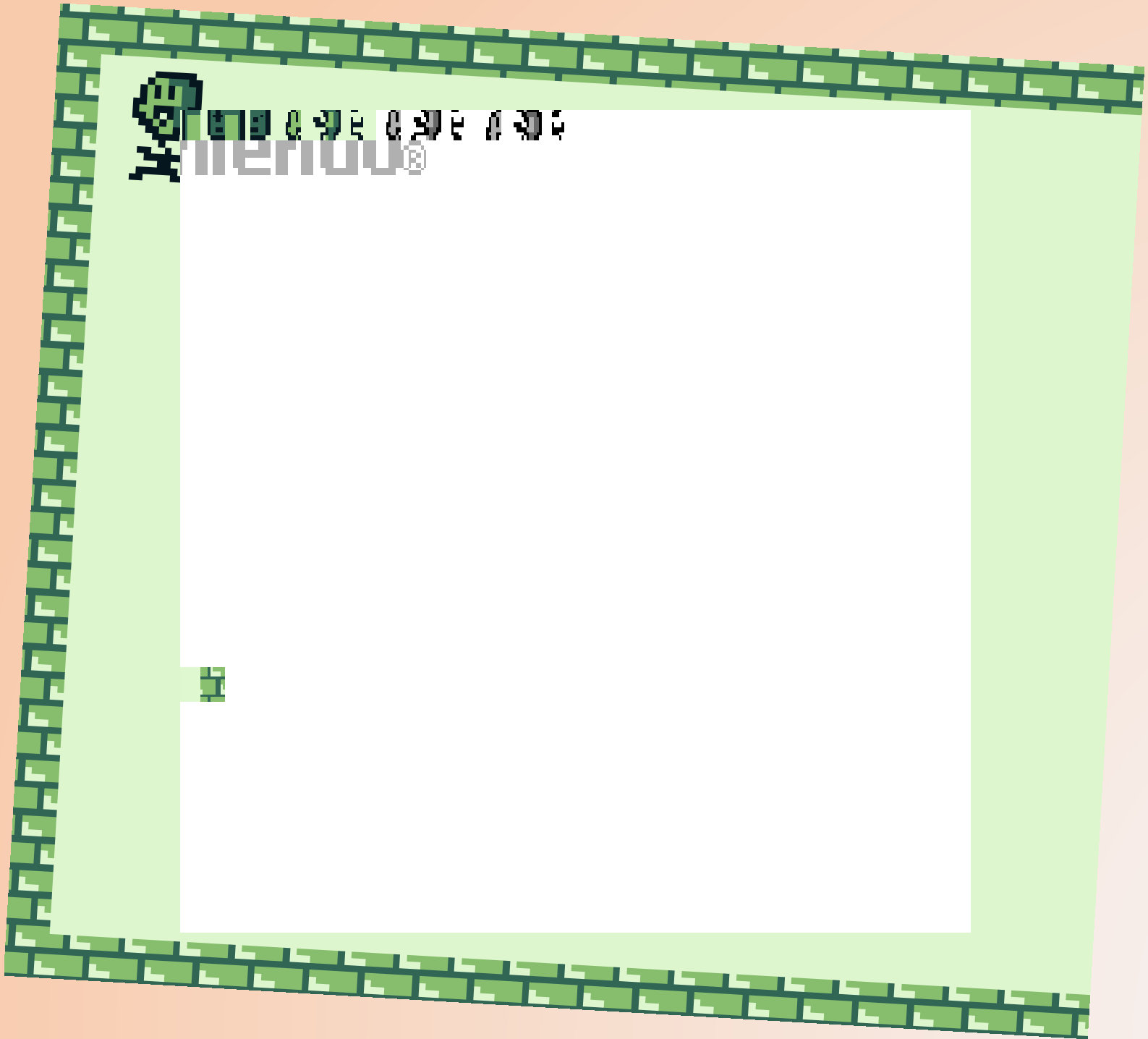
- Ahora haz que use esa animación desde un principio.

```
void Start_SPRITE_MONOFEO() {
    SetSpriteAnim(THIS, anim_caminando, 15);
}
```

- **Anim_caminando** es la animación, y 15 la **cantidad que se demorará** cada frame.
- Y noooo olvides incluir **SpriteManager.h** para que reconozca la función que acabamos de incluir.

```
#include "SpriteManager.h"
```

¿Compilemos otra vez?



Hagamos que Player... no se, se mueva

- Vamos a **SpritePlayer.c** esta vez.
- Inyectemosle **SpriteManager.h** también, pero además **Keys.h**

```
#include "SpriteManager.h"
#include "Keys.h"
```

- En la función Update preguntemos si se están presionado las teclas… y si es así, modificar la posición del Sprite, modificando sus coordenadas x e y.

```
void Update_SPRITE_PLAYER() {
    if(KEY_PRESSED(J_UP)) {
        THIS->y --;
    }
    if(KEY_PRESSED(J_DOWN)) {
        THIS->y ++;
    }
    if(KEY_PRESSED(J_LEFT)) {
        THIS->x --;
    }
    if(KEY_PRESSED(J_RIGHT)) {
        THIS->x ++;
    }
}
```

NOTA: **J_A, J_B, J_SELECT, J_START** también los puedes usar para lo que se te antoje!

Un método alternativo

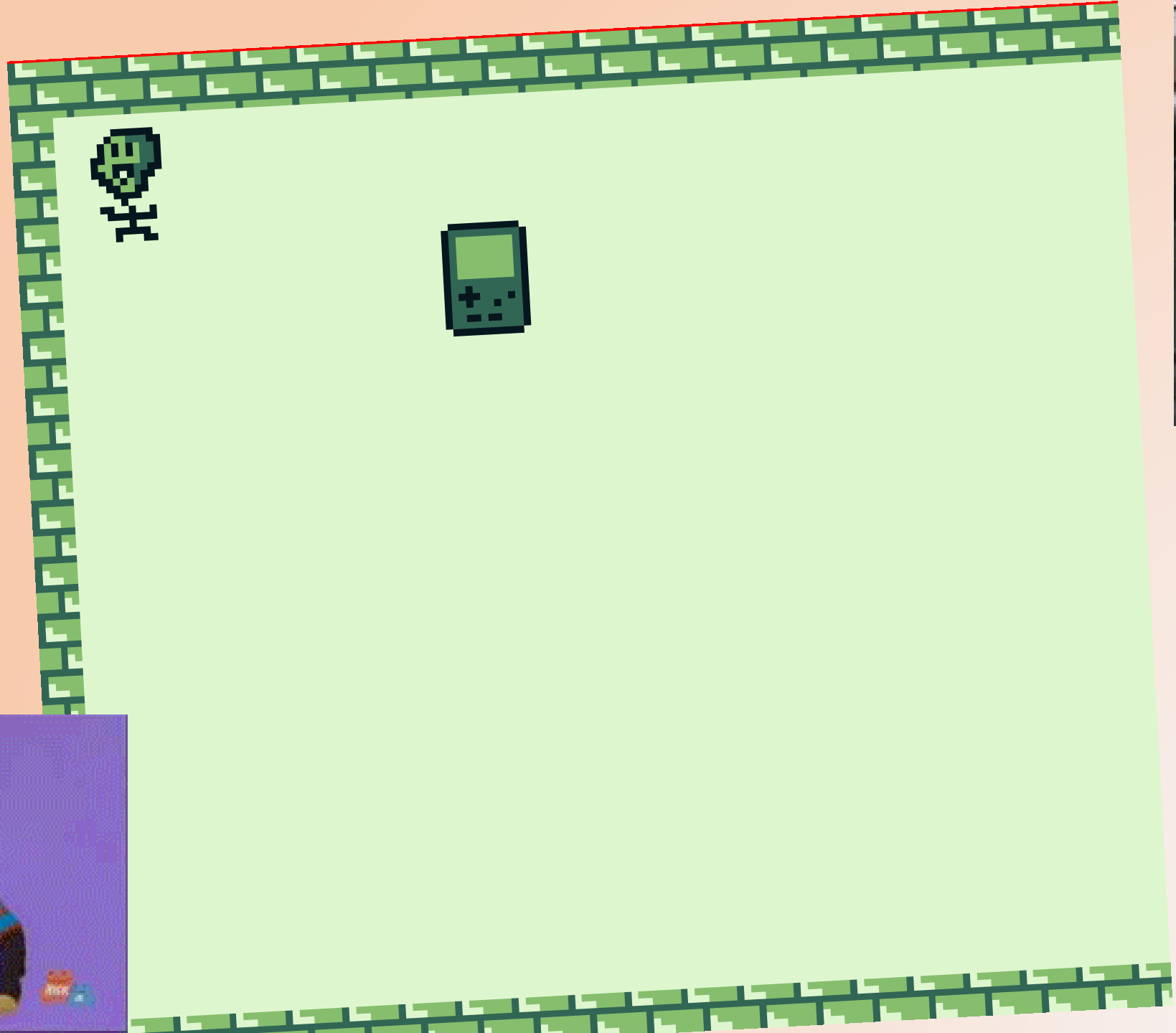
- Lo anterior funciona… pero puede hacer que el movimiento sea mas lento si es que hay lag. (Si, ni la gameboy se salva xD)
- **Esta es otra forma de hacer lo mismo, pero a prueba de lag y colisiones.**

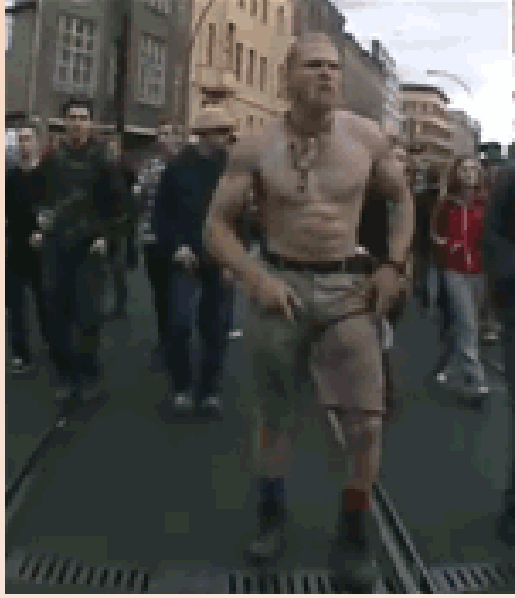
```
void Update_SPRITE_PLAYER() {

    if(KEY_PRESSED(J_UP)) {
        TranslateSprite(THIS, 0, -1 << delta_time);
    }
    if(KEY_PRESSED(J_DOWN)) {
        TranslateSprite(THIS, 0, 1 << delta_time);
    }
    if(KEY_PRESSED(J_LEFT)) {
        TranslateSprite(THIS, -1 << delta_time, 0);
    }
    if(KEY_PRESSED(J_RIGHT)) {
        TranslateSprite(THIS, 1 << delta_time, 0);
    }
}
```

NOTA 1: delta_time ya viene definido magicamente por default! ;)

NOTA 2: TranslateSprite retorna el ID del tile donde quedará parado. Mas de eso luego ☺

¿Volvamos a compilar?





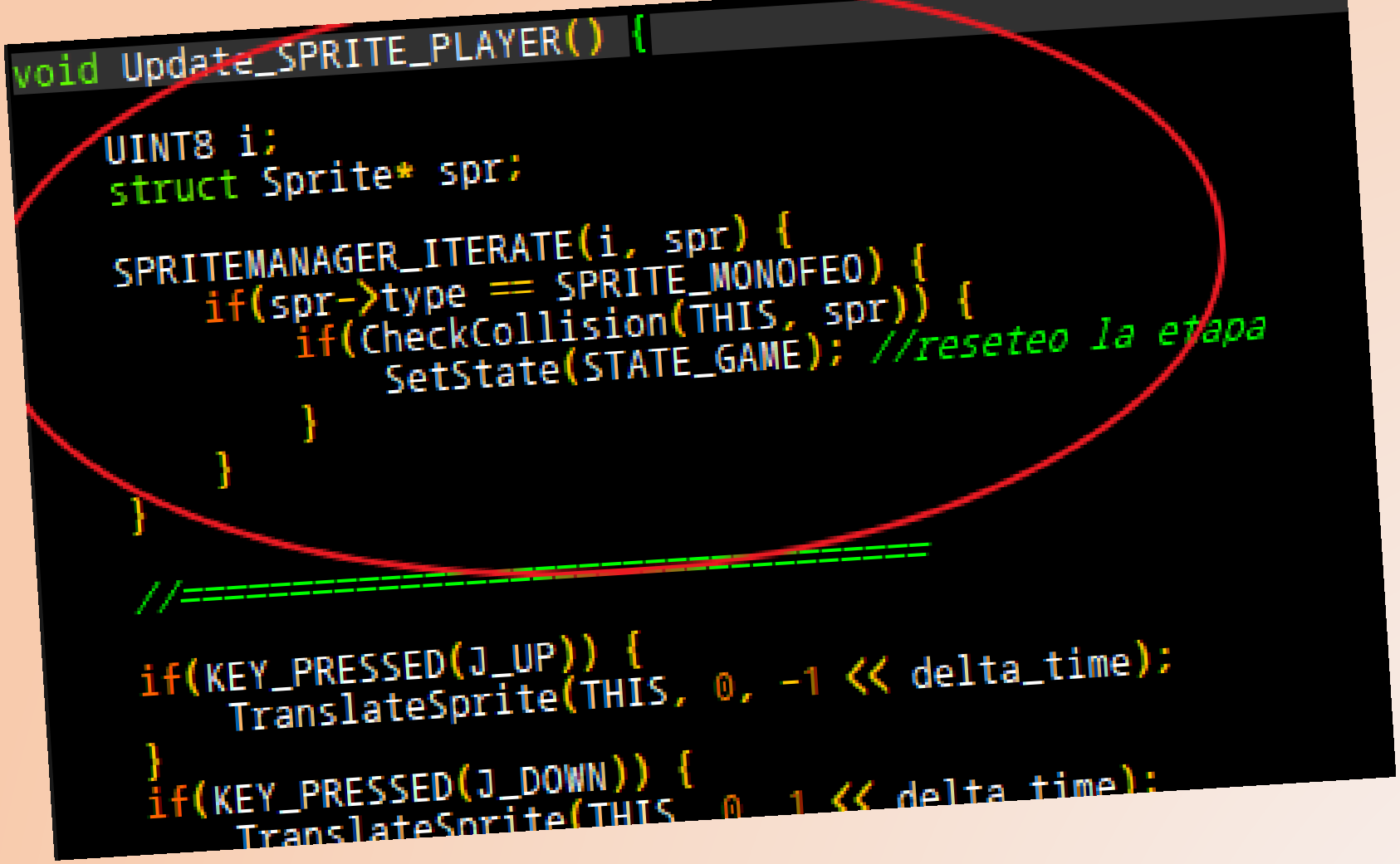
Hagamos que el Player muera con el MonoFeo

- Eso se hace con colisiones!
- Volvamos a **SpritePlayer.c**
- Incluyamos ahora estas cosas para **tener colisiones**, y para **reconocer a MonoFeo** dentro de Player.

```
#include "SpriteMonoFeo.h"
#include "ZGBMain.h"
```

- Agreguemos esto en el update del Player, antes de los movimientos.

```
void Update_SPRITE_PLAYER() {

    UINT8 i;
    struct Sprite* spr;

    SPRITEMANAGER_ITERATE(i, spr) {
        if(spr->type == SPRITE_MONOFEO) {
            if(CheckCollision(THIS, spr)) {
                SetState(STATE_GAME); //reseteo la etapa
            }
        }
    }

    //=================================

    if(KEY_PRESSED(J_UP)) {
        TranslateSprite(THIS, 0, -1 << delta_time);
    }
    if(KEY_PRESSED(J_DOWN)) {
        TranslateSprite(THIS, 0, 1 << delta_time);
```



Mira todos los sprites en el escenario, pero toma solo los de tipo **MonoFeo**. Pregunta si esta colisionando, y si es así, **resetea el escenario.**

¿Musica disco?

- Puedes autodestruir sprites cuando quieras, llamando esta función

```
SpriteManagerRemove(THIS);
```

- No solo puedes crear sprites al crear escenarios, sino donde quieras (dentro de una función de un Sprite… por ejemplo, al disparar).

- Puedes crear todos los **Player y MonoFeos que quieras** en tu pantalla. (Recuerda la limitación de los 20 sprites si).

- ¿Te diste cuenta que la cámara NO sigue la gameboy (Player)? **Eso es porque en un principio cambiamos la cámara al MonoFeo.** Vuelvela a Player y prueba.

Ideas

Mas ideas

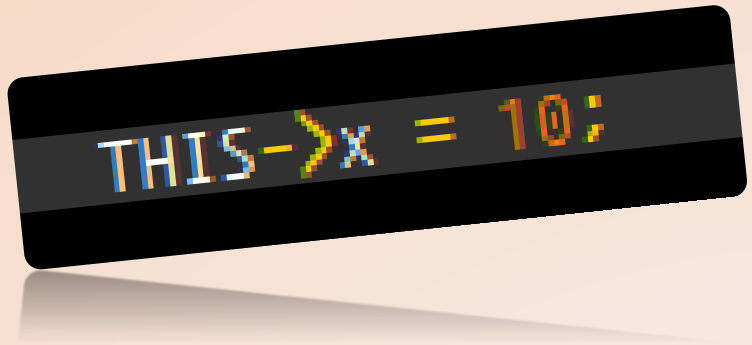
- En cada Sprite en pantalla puedes guardar 8 bytes de datos (lo que quieras… salud, daños, objetivos, estados, etc) si lo deseas! Tendrás que acordarte si un poco de structs si quieres hacer eso.

- He aquí un ejemplo con **MonoFeo**.

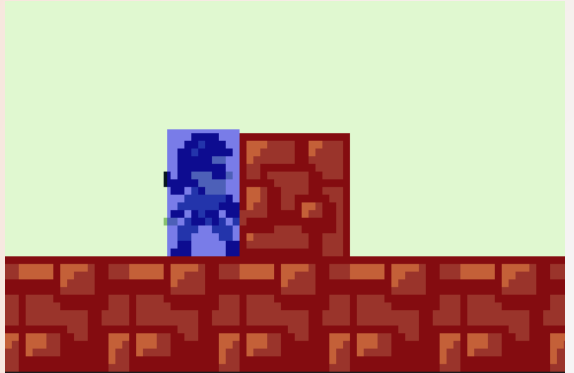
```
#pragma bank 2
#include "SpriteMonoFeo.h"

#include "SpriteManager.h"

UINT8 bank_SPRITE_MONOFEO = 2;

//struct con mis datos
struct MonoFeoInfo {
    INT8 salud;
};

const UINT8 anim_quieto[] = {1, 0};
const UINT8 anim_caminando[] = {3, 0, 1, 2};

void Start_SPRITE_MONOFEO() {

    //acceder a mis datos
    struct MonoFeoInfo* data = (struct MonoFeoInfo*)THIS->custom_data;
    data->salud = 1;

    SetSpriteAnim(THIS, anim_caminando, 15);


}

void Update_SPRITE_MONOFEO() {

    //acceder a mis datos
    struct MonoFeoInfo* data = (struct MonoFeoInfo*)THIS->custom_data;
    data->salud ++; //no sirve de nada, pero es para el ejemplo

}

void Destroy_SPRITE_MONOFEO() {

}
```

Que tiene THIS en los Sprites ?

Tiene hartas cosas:



- **x :** Coordenada X en el escenario
- **y:** Coordenada Y en el escenario
- **flags:** Si lo seteas como OAM_VERTICAL_FLAG lo inviertes. Vuelve como antes seteandolo en 0.
- **lim_x:** Que tan lejos puede ir tu sprite de la pantalla antes que se autodestruya.
- **lim_y:** Que tan lejos puede ir tu sprite de la pantalla antes que se autodestruya.
- **coll_x:** Ajusta la caja de colisión de tu Sprite.
- **coll_y:** Ajusta la caja de colisión de tu Sprite.
- **coll_w:** Ajusta la caja de colisión de tu Sprite.
- **coll_h:** Ajusta la caja de colisión de tu Sprite.



Pero estamos en medio de la nada

Hagamos escenarios!

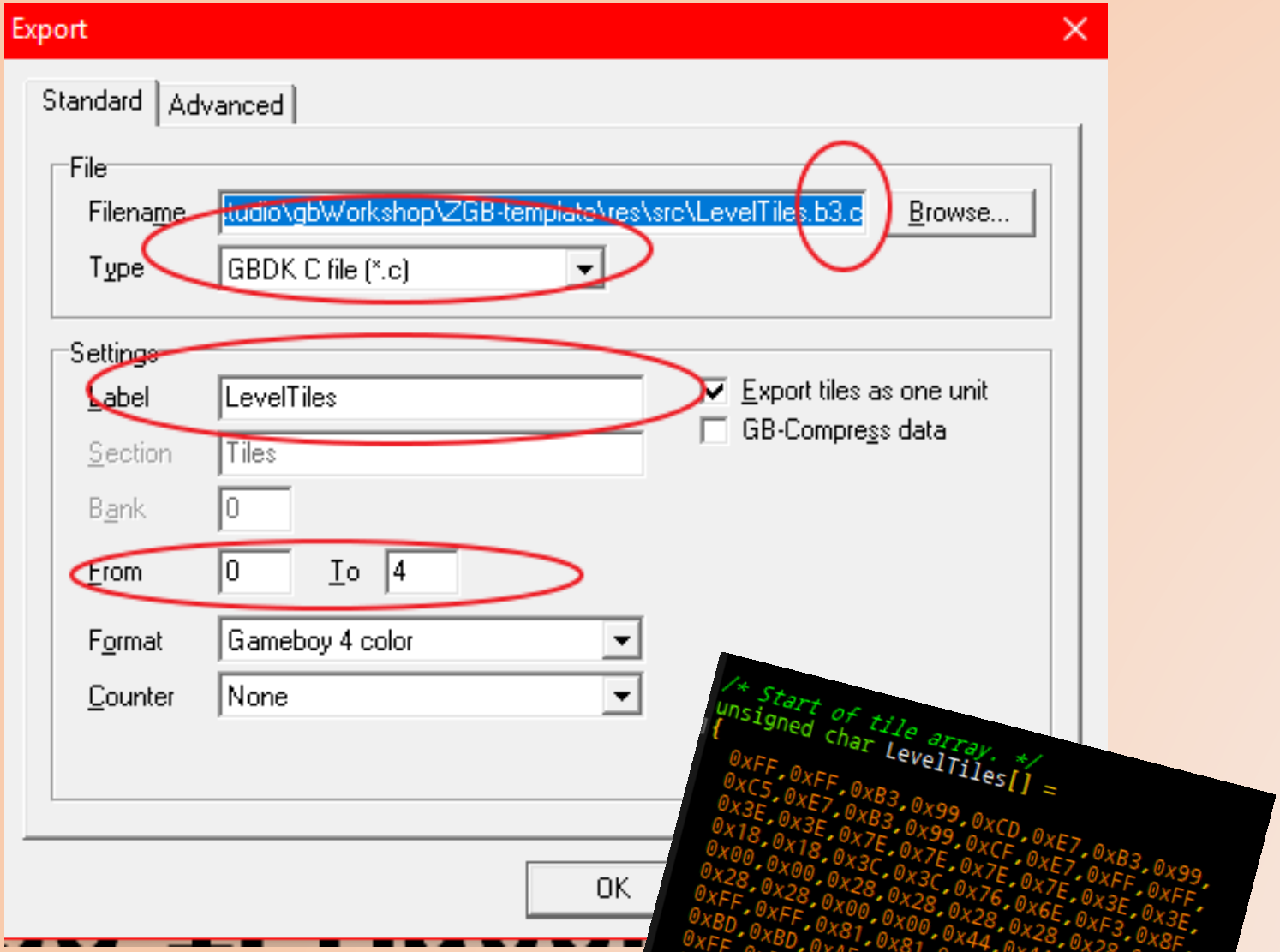
- Similar a muchos engines, el juego se divide en **Estados/Escenas**.
- Español chilensis: Un estado/escena es un escenario, o una pantalla de juego.
- En ZGB se llaman **States** (sorry Unity lovers)
- Los escenarios se hacen con **Game Boy Map Builder (GBMB)**, y las texturas del escenario (tiles), al igual que los sprites, con el **Game Boy Tile Designer (GBTD)**

State Level

State Title

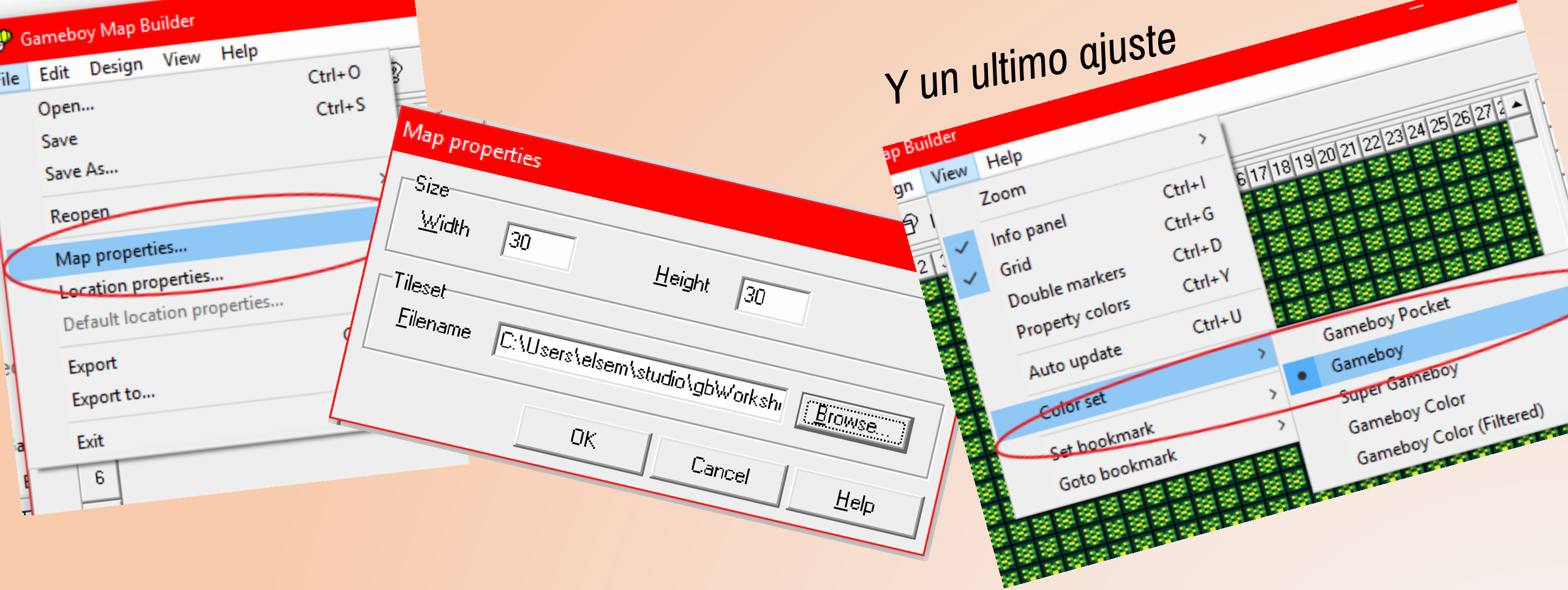
Paso 1: Hacer los tiles

Exactamente igual como los sprites, pero de 8x8



- También es igual al exportar… Recuerda lo que había que setear.
- Solo procura escoger un banco adecuado, y algún nombre que te haga acordar que son Tiles (y no Srpites).

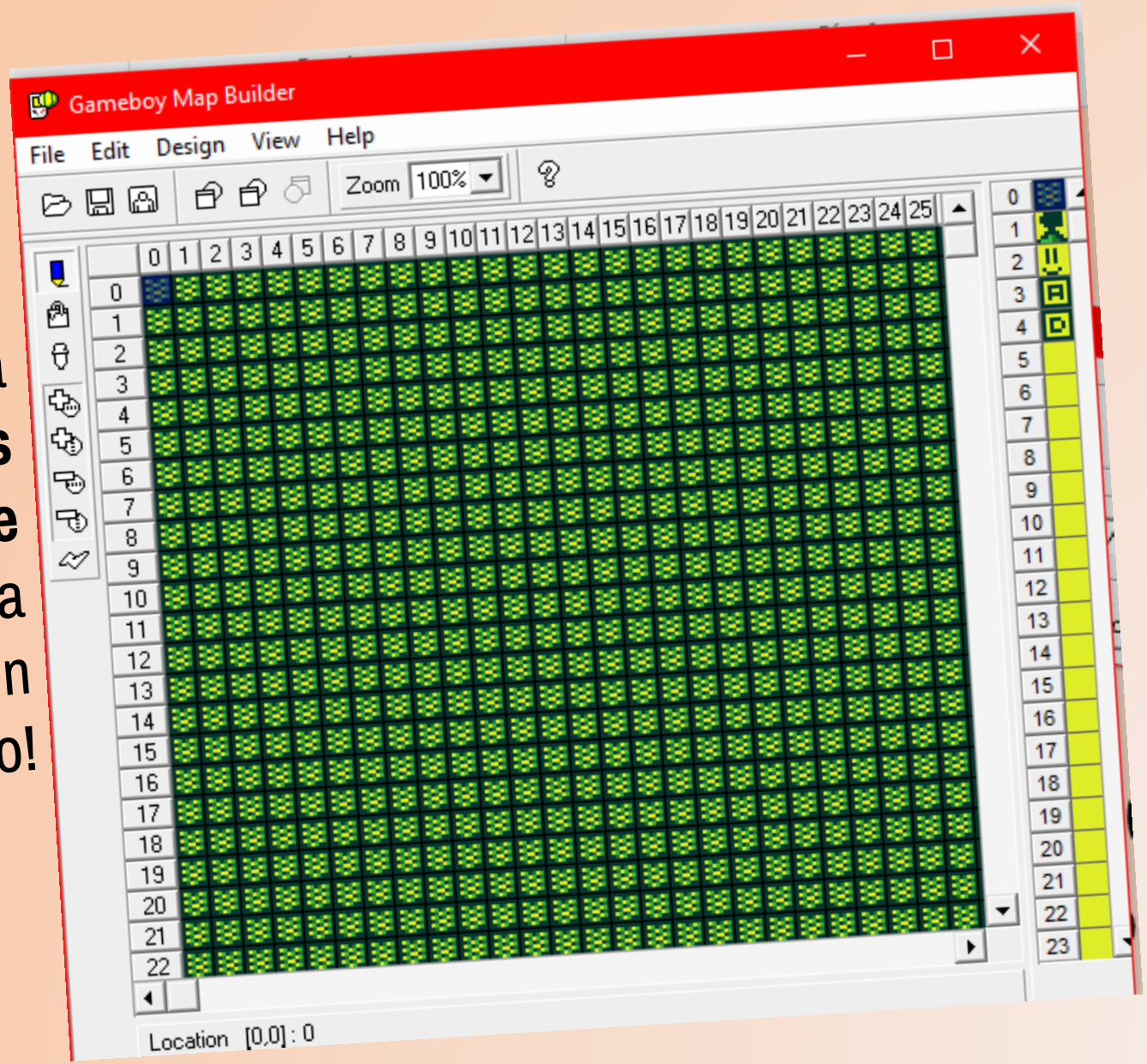
- **NOTA: Recuerda dejar un tile en blanco, porque no existen tiles vacíos para dejar espacios en blanco en el escenario.**



Paso 2: Setear el mapa (GBMB).

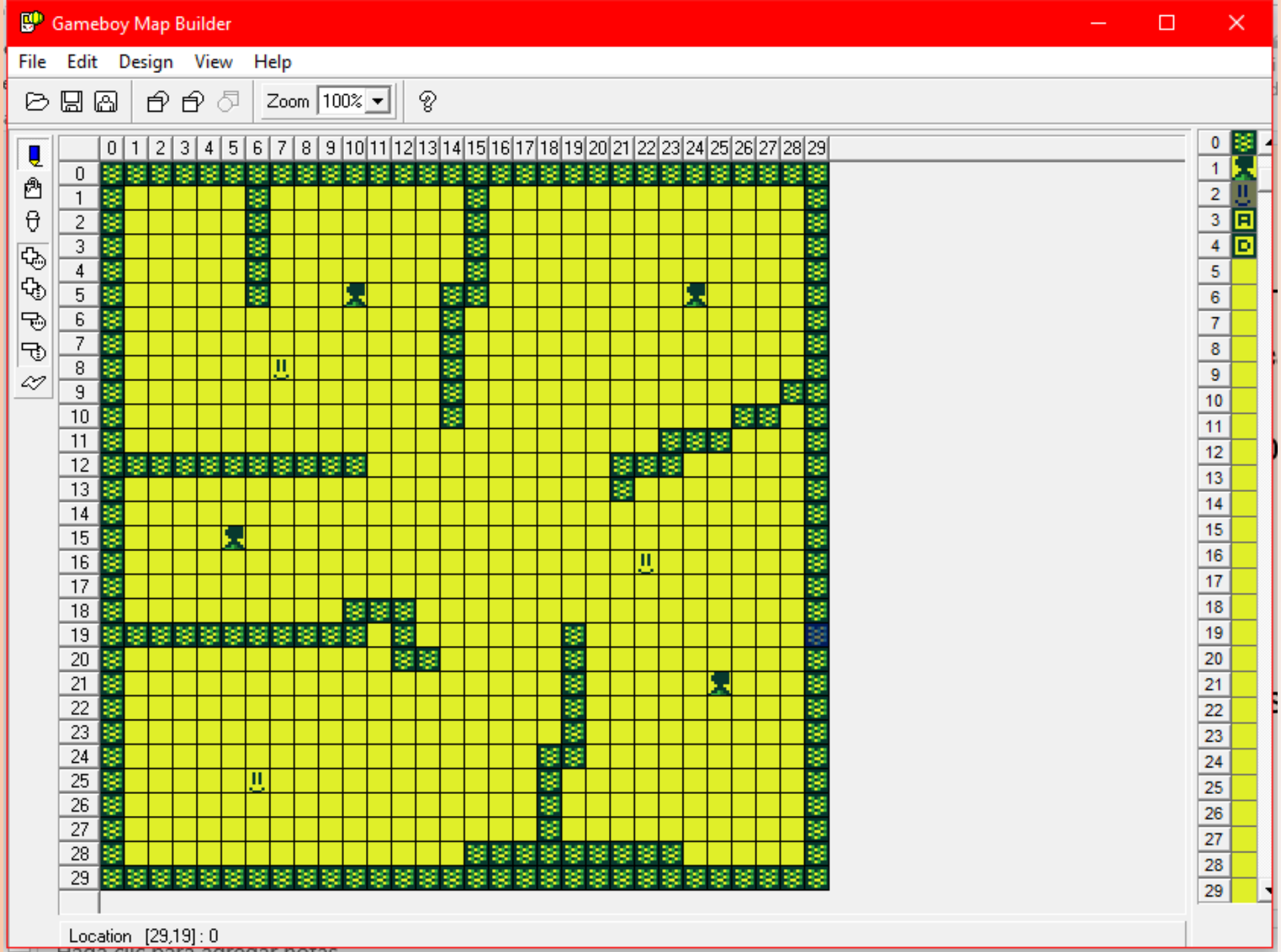
- Acá setea el tamaño del mapa, y tus tiles **SIN COMPILAR**.
- Psst, 20x18 es el tamaño de la pantalla, para referencia.

¡El escenario está **lleno de nuestros bloques! (Tile 0)**... Por eso la nota de incluir un tile en blanco!

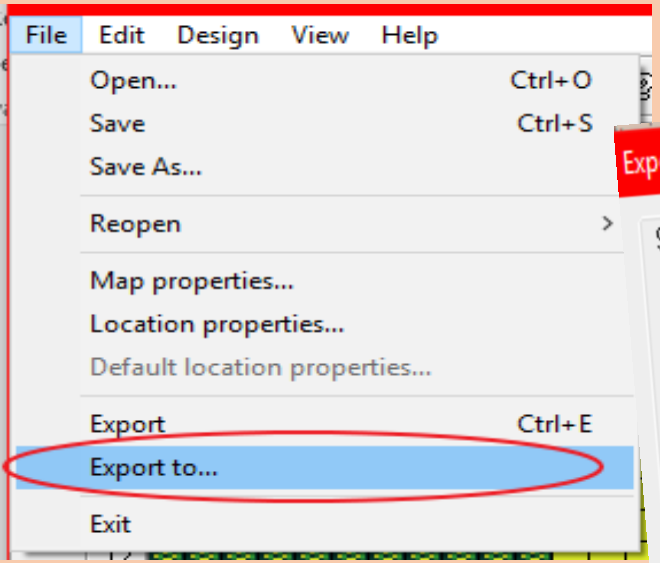


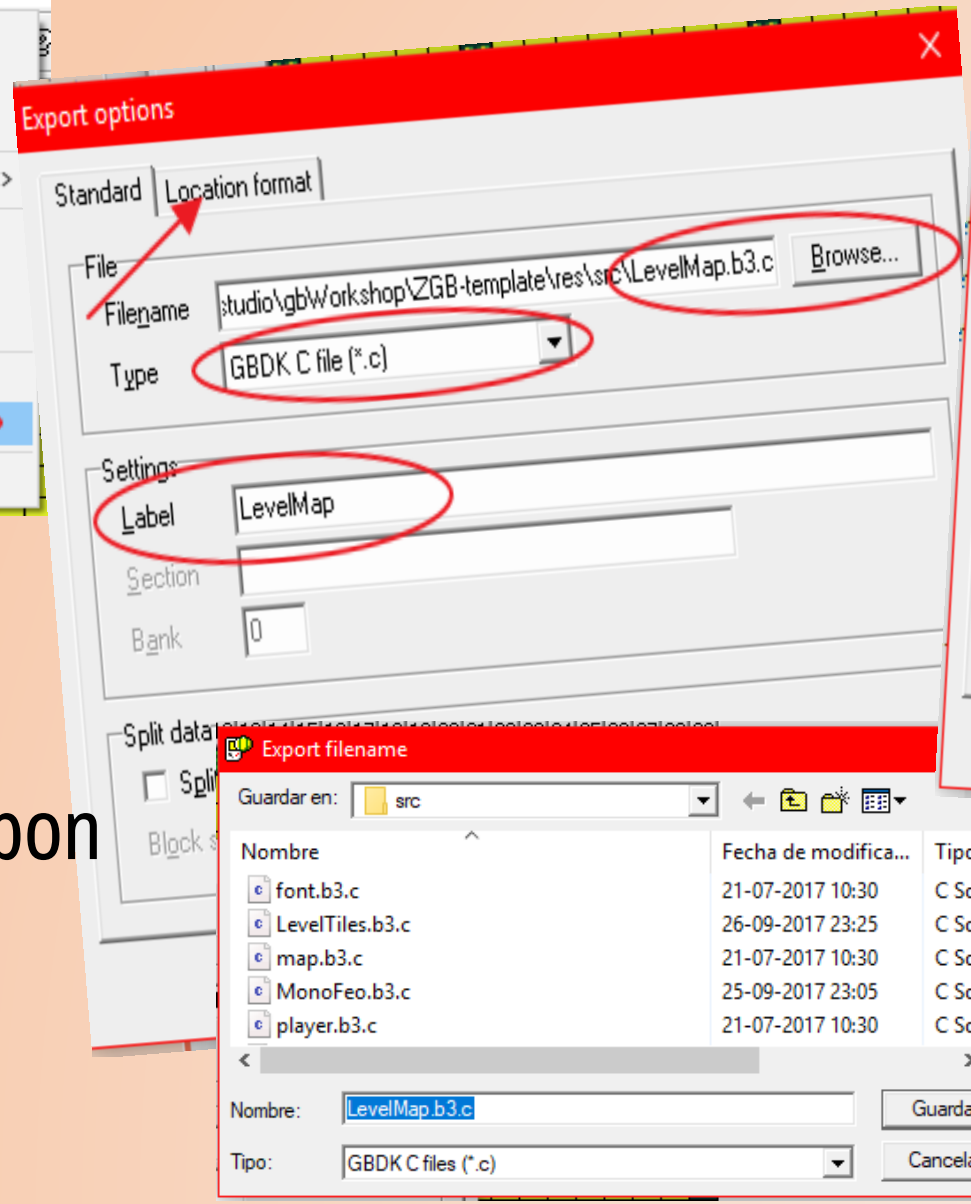
Tomas un tile (que acabas de hacer) de la paleta aquí y dibujas tu mapa ☺

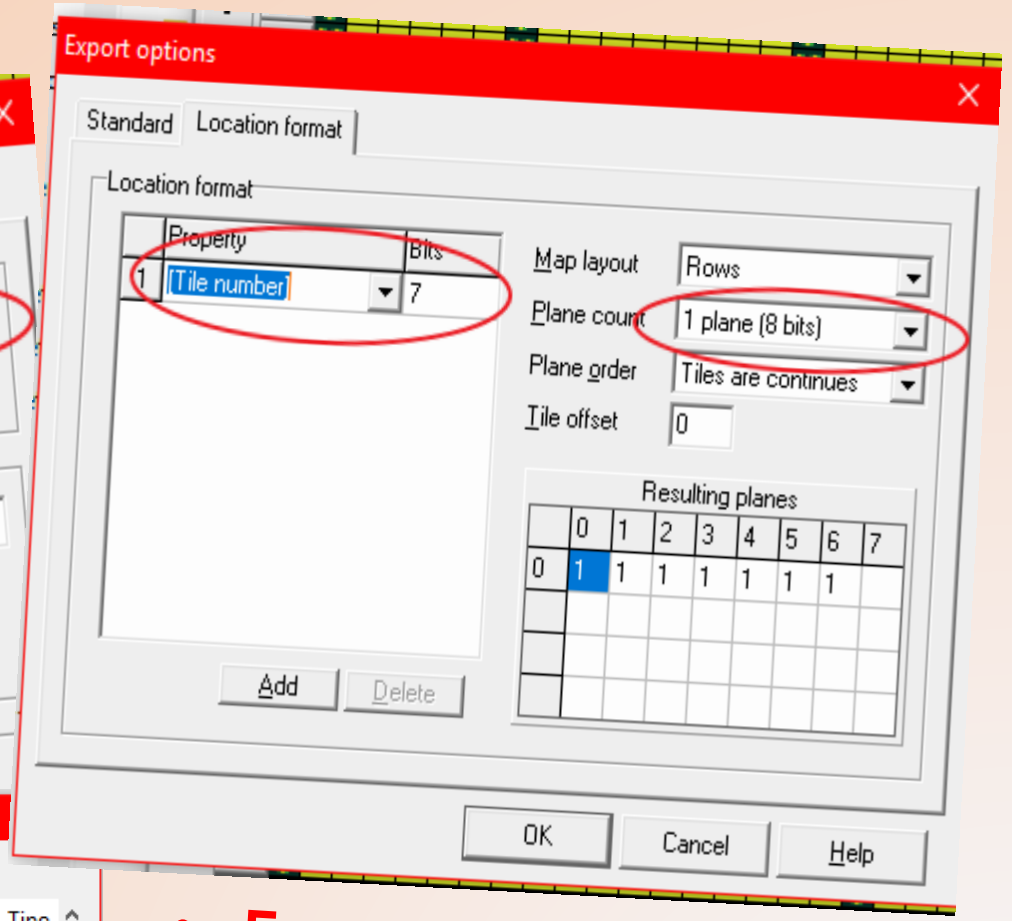
Click izquierdo seleccionas, click derecho pintas.


Gameboy Map Builder
File
Edit
Design
View
Help
Zoom
100%
Location [29,19] : 0

Al igual que los sprites/tiles, los mapas se compilan (exportan).







Lo mismo aquí:
Expórtalo
a formato **GBDK**, pon el **nombre y en el archivo el banco** donde lo pondrás.

- **En property pon [Tile number]. Asegurate que quede en 7.**
- **Plane count: 1 plane (8 bits)**

```c
#define LevelMapWidth 30
#define LevelMapHeight 30
#define LevelMapBank 0

unsigned char LevelMap[] =
{
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x05,0x05,0x05,0x05,0x05,0x00,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x05,0x00,0x05,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x00,
   0x00,0x05,0x05,0x05,0x05,0x05,0x00,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x05,0x00,0x05,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x00,
   0x00,0x05,0x05,0x05,0x05,0x05,0x00,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x05,0x00,0x05,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x00,
   0x00,0x05,0x05,0x05,0x05,0x05,0x00,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x05,0x00,0x05,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x00,
   0x00,0x05,0x05,0x05,0x05,0x05,0x00,0x05,0x05,0x05,
   0x01,0x05,0x05,0x05,0x00,0x00,0x05,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x01,0x05,0x05,0x05,0x05,0x00,
   0x00,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x00,0x05,0x05,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x00,
   0x00,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x00,0x05,0x05,0x05,0x05,0x05,
   0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x05,0x00,
```

En /res/src tendremos entonces nuestro mapa **LevelMap.b3.c** y **LevelMap.h**

Y su TileSet (**LevelTiles.b3.c** y **LevelTiles.h**)

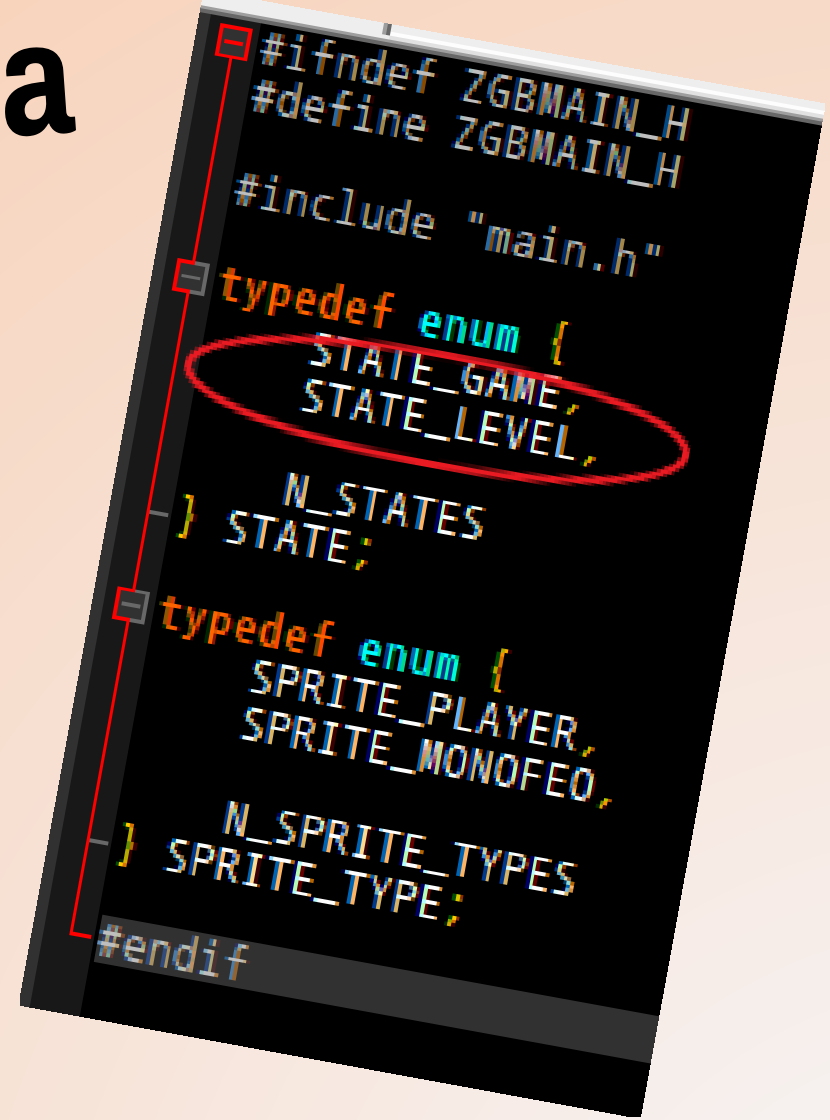
```c
#define LevelMapWidth 30
#define LevelMapHeight 30
#define LevelMapBank 0

extern unsigned char LevelMap[];

/* End of LEVELMAP_H */
```

NOTA: Suponiendo que nuestro escenario lo llamaremos Level

La parte fome ahora

- En /include/ crear **StateLevel.h**

```
#ifndef STATE_LEVEL_H
#define STATE_LEVEL_H

#include "main.h"

DECLARE_STATE(STATE_LEVEL);

#endif
```



- Y modificar **ZGBMain.h** para añadir nuestro State

NOTA: Si te da mucha lata escribir eso, puedes basarte en los que ya existen en el template. Asegúrate si de reemplazar todos los valores que es razón #1 si no compila xD

- En /src/ modificar **ZGBMain_Init.c** para cargar nuestro estado nuevo.

```
#pragma bank 1
#include "ZGBMain.h"
UINT8 init_bank = 1;

#include "StateGame.h"
#include "StateLevel.h"

#include "SpritePlayer.h"
#include "SpriteMonoFeo.h"

#include "../res/src/player.h"
#include "../res/src/MonoFeo.h"

UINT8 next_state = STATE_LEVEL; //estado inicial!

SET_N_STATES(N_STATES);
SET_N_SPRITE_TYPES(N_SPRITE_TYPES);

void InitStates() {
    INIT_STATE(STATE_GAME);
    INIT_STATE(STATE_LEVEL);
}

void InitSprites() {
    INIT_SPRITE(SPRITE_PLAYER, player, 3, FRAME_16x16, 1);
    INIT_SPRITE(SPRITE_MONOFEO, MonoFeo, 3, FRAME_16x16, 3);
}
```

NOTA: next_state es el estado con que el juego comenzará. Puedes hacer que vaya a otros estados al principio si quieres.

- Yyyy… creamos nuestro **StateLevel.c** aquí mismo.
- Es de hecho, casi un copypaste del StateGame de antes, solo cambia Game por Level xD

- … Y claro, nuestro mapa con los tiles. LevelMapWidth y LevelMapHeight los crea el editor de mapas!

- InitScrollTiles, sus parámetros son 0, Cantidad de tiles, tus tiles, banco donde están los tiles.

- InitScroll, sus parámetros son, Largo, alto, el mapa, las colisiones (de ahi), 0, el banco donde esta el mapa.

```
#pragma bank 2
#include "StateLevel.h"
UINT8 bank_STATE_LEVEL = 2;

#include "..\res\src\LevelTiles.h"
#include "..\res\src\LevelMap.h"

#include "ZGBMain.h"
#include "Scroll.h"
#include "SpriteManager.h"

extern UINT8 n_sprite_types;
void Start_STATE_LEVEL() {
    UINT8 i;

    // ===============================================
    //eso carga sprites... dejemoslo asi nomas
    SPRITES_8x16;
    for(i = 0; i != n_sprite_types; ++ i) {
        SpriteManagerLoad(i);
    }
    SHOW_SPRITES;
    // ===============================================

    //los mismos sprites del otro escenario nomas

    scroll_target = SpriteManagerAdd(SPRITE_MONOFEO, 10, 10);
    SpriteManagerAdd(SPRITE_PLAYER, 50, 50);

    // ===============================================

    //cargar tiles
    InitScrollTiles(0, 2, LevelTiles, 3);

    //cargar mapa
    InitScroll(LevelMapWidth, LevelMapHeight, LevelMap, 0, 0, 3);
    SHOW_BKG;
}

void Update_STATE_LEVEL() {

}
```

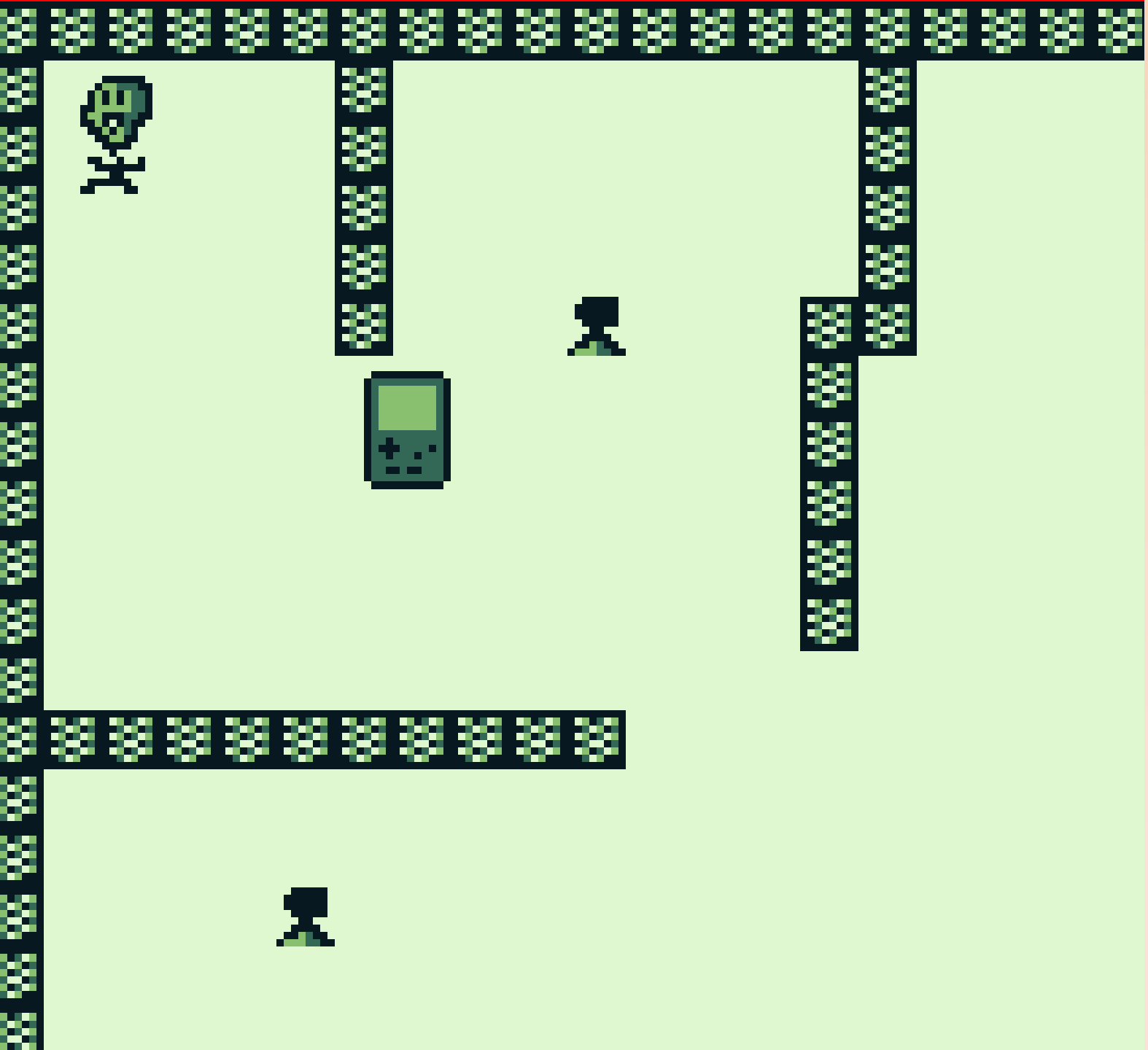
Al igual que los sprites, los states tienen los siguientes estados:

- Start: Cuando carga el escenario
- Update: Se llama una vez cada frame.

El update puede ser útil si tienes un estado sin sprites, y quieres hacer algo allí (presionar un botón para ir a la escena siguente...)

```
#pragma bank 2
#include "StateLevel.h"
UINT8 bank_STATE_LEVEL = 2;

#include "..\res\src\LevelTiles.h"
#include "..\res\src\LevelMap.h"

#include "ZGBMain.h"
#include "Scroll.h"
#include "SpriteManager.h"

extern UINT8 n_sprite_types;
void Start_STATE_LEVEL() {
    UINT8 i;

    // ===============================================
    //eso carga sprites... dejemoslo asi nomas
    SPRITES_8x16;
    for(i = 0; i != n_sprite_types; ++ i) {
        SpriteManagerLoad(i);
    }
    SHOW_SPRITES;
    // ===============================================

    //los mismos sprites del otro escenario nomas

    scroll_target = SpriteManagerAdd(SPRITE_MONOFEO, 10, 10);
    SpriteManagerAdd(SPRITE_PLAYER, 50, 50);

    // ===============================================

    //cargar tiles
    InitScrollTiles(0, 2, LevelTiles, 3);

    //cargar mapa
    InitScroll(LevelMapWidth, LevelMapHeight, LevelMap, 0, 0, 3);
    SHOW_BKG;
}

void Update_STATE_LEVEL() {

}
```

¿Compilemos?



- Por favor, la música se esta poniendo cada vez peor
- Wow, un escenario nuevo!
- Pero **no hay colisiones...**
- Si toco al MonoFeo y muero, **vuelvo al escenario que teníamos antes**.
- Eso es porque al colisionar, lo mandábamos explícitamente al **STATE_GAME** (el que venia con el template).

```
if(CheckCollision(THIS, spr)) {
    SetState(STATE_GAME); //reseteo la
}
```

- Para cambiarlo, ¿es obvio, no?

Hora de añadir colisiones

- Cero ciencia esto. Ve a nuestro StateLevel.c
- Modifica la parte donde cargamos el mapa y los tiles. **Agreguemosle que tiles colisionaran**.
- Asegúrate que el arreglo ese **siempre termine en 0** (por ende, el 0 nunca es colisionable).


```
// =======================================

UINT8 collision_tiles[] = {1, 3, 4, 0};

//cargar tiles
InitScrollTiles(0, 2, LevelTiles, 3);

//cargar mapa
InitScroll(LevelMapWidth, LevelMapHeight, LevelMap, collision_tiles, 0, 3);
SHOW_BKG;
```

Cambié un poco el escenario porque me mande una pifea porque se me olvidó lo anterior xDD

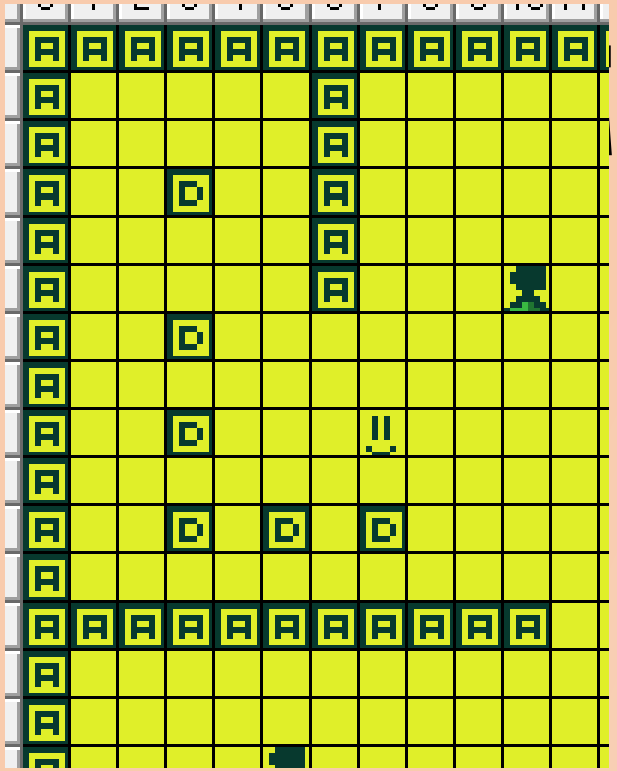
NOTA 1: Recuerda definir la variable al principio para evitarte el manso error de compilación xD

NOTA 2: ¿Recuerdas la nota donde dije que **TranslateSprite retornaba el ID de tile donde estas**? Ahora todo cobra sentido, por si lo quisieras usar para algún chequeo mas avanzado ☺

Pero aaaaantes de compilar

```
SpriteManagerAdd(SPRITE_MONOFEO, 10, 10);
```

¿No hay una forma menos latera de agregar sprites (En este caso, monos feos) que no sea pixel por pixel?



Sip, hay un par de formas!

- Empieza dibujando tiles “especiales” en el mapa, los cuales vendrían siendo marcadores espaciales. En mi caso serán los D

- **Método 1**: Buscar manualmente. (Para 1 sprite).

```
//3 es banco, 4 es tile a buscar
ScrollFindTile(LevelMapWidth, LevelMap, 3, 98, 0, 0, LevelMapWidth, LevelMapHeight, &resultadoX, &resultadoY);
```

- Donde 3 es el banco del mapa, y 98 el **“supuesto” ID del tile a buscar**. El resultado se almacena en **resultadoX** y **resultadoY** .
- Ambos resultados los **multiplicas por 8** (tamaño del tile), y tadaaa! Tienes las coordenadas donde debes colocar. (var << 3)

- **Metodo 2, el facil**: Buscar automáticamente.
- Para eso nos vamos a **/src/ZGB_Main.c** … y cambiamos el código por otro mas entendible.

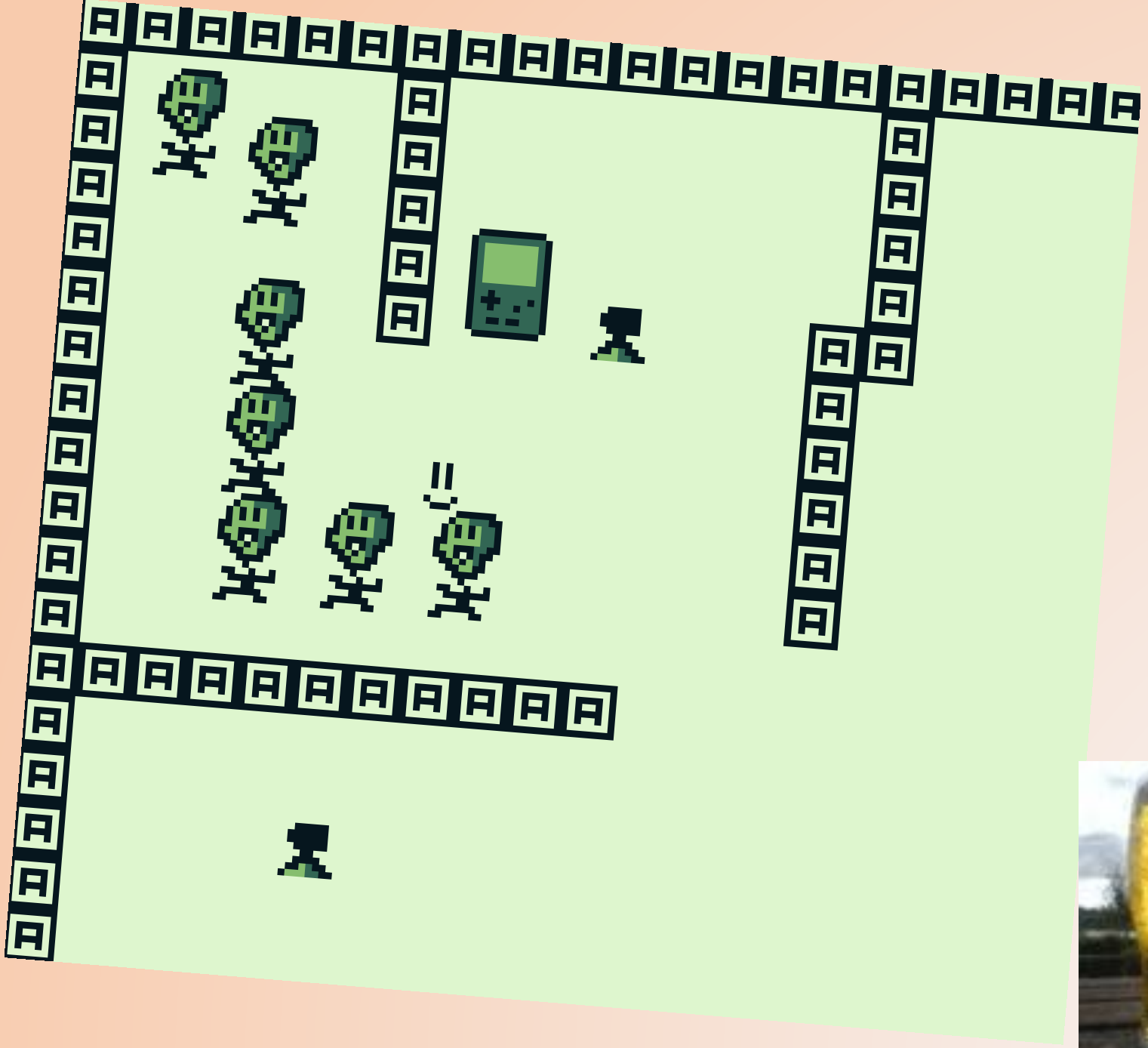
```
#include "ZGBMain.h"

UINT8 GetTileReplacement(UINT8* tile_ptr, UINT8* tile) {


    if(current_state == STATE_LEVEL) {
        *tile = 5; //dejar tile vacio
        switch(*tile_ptr) {

            case 4: return SPRITE_MONOFEO;
            case 30: return 5u;
            case 31: return 5u;
        }
        //aplicar el tile si no entro a nada
        *tile = *tile_ptr;

    }

    return 255u;
}
```

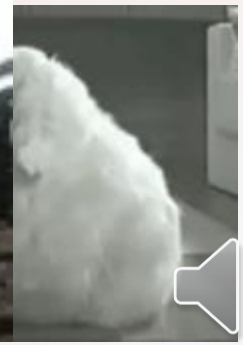
En palabras simples:

- Mira el switch: **Se buscan los tiles 4, 30 y 31… SOLO en el mapa STATE_LEVEL**.
- Si encuentra el 4, **spawnea allí un MonoFeo**.
- Si encuentra un 30 o un 31 (solo para el ejemplo), no spawneara nada, **pero “borrara” el tile (pondrá el 5 en su lugar).**
- De lo contrario, no pasa nada.

NOTA: La desventaja? Siempre que pases allí se spawnearan!

¿Compilemos ahora si?

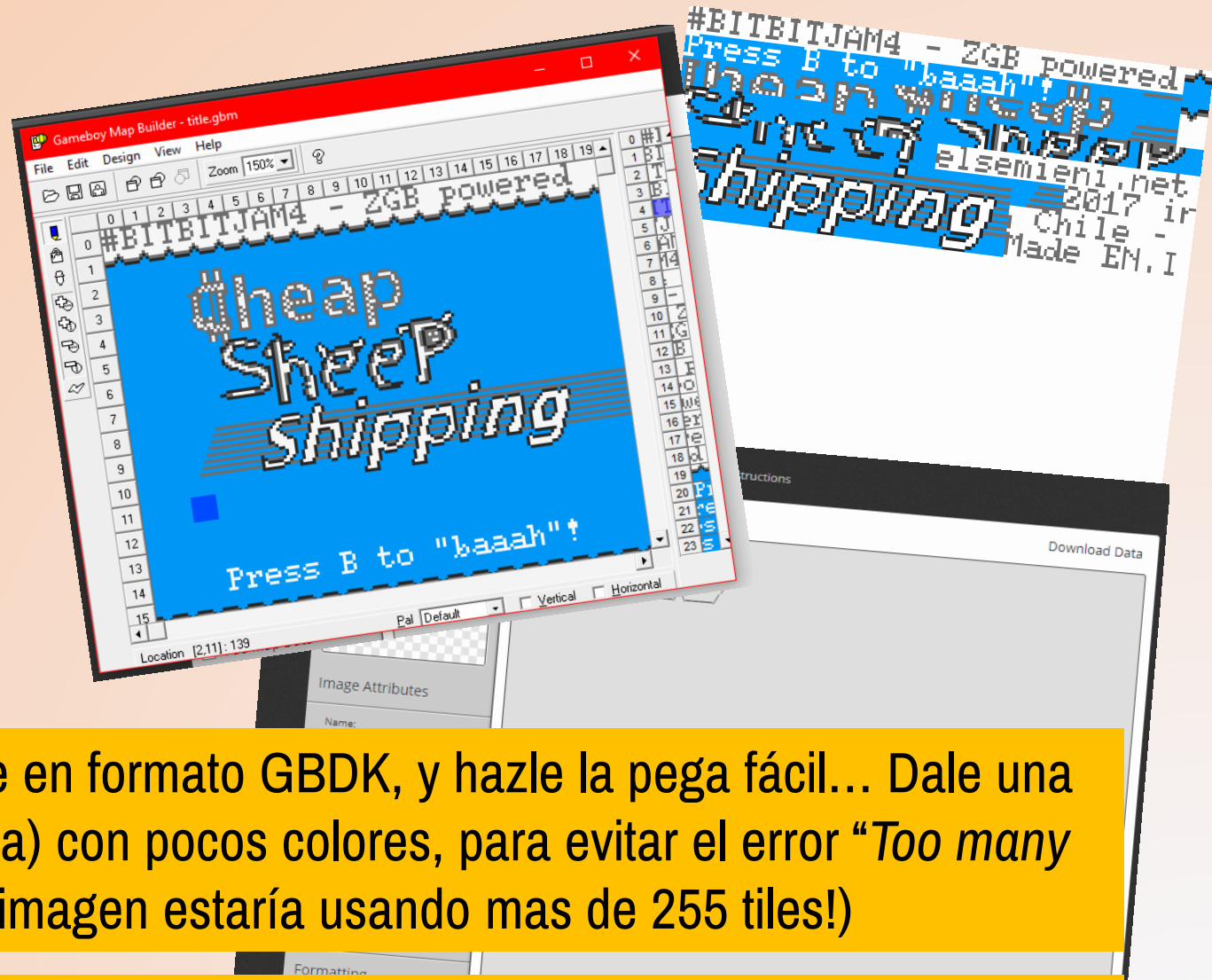




Para mostrar imágenes en vez de escenarios

- Puedes recortar una imagen en pedacitos en el Tile Designer, y reconstruirlo en el Map Builder.

- O usas esta pagina que hace la pega por ti: http://www.chrisantonellis.com/gameboy/gbtdg/



NOTA 1: Recuerda setearlo para que genere en formato GBDK, y hazle la pega fácil... Dale una imagen sencilla de 160x144 (tamaño pantalla) con pocos colores, para evitar el error "*Too many unique tiles for one tilemap.* " (La imagen estaría usando mas de 255 tiles!)

NOTA 2: Al exportar, adapta el recurso para que cumpla el formato de nombres del ZGB, o sea para que las variables tengan los nombres, y los bancos adecuados!

Un estado para presionar un botón e ir a otro

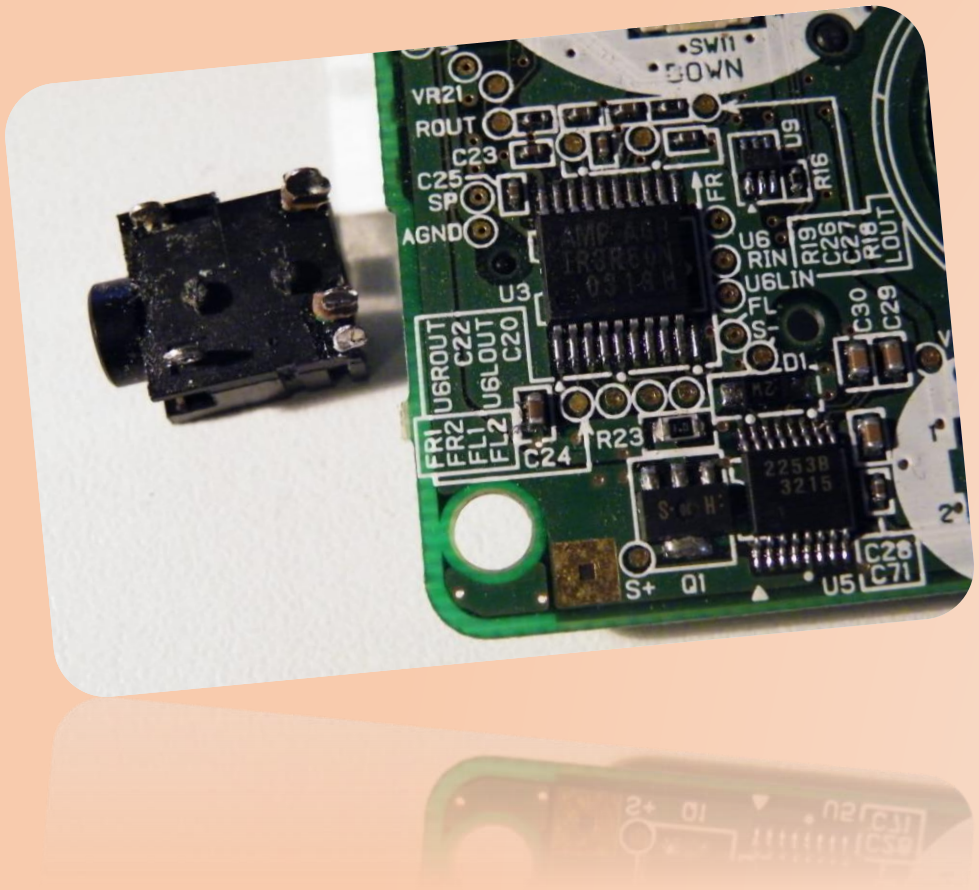
```
void Start_STATE_OPENING() {

    HIDE_WIN;

    SetPalette(BG_PALETTE, 0, 8, bgOpening_palette, bank_STATE_OPENING);

    PlayMusic(jingle_mod_Data, 4, 0);

    InitScrollTilesColor(0, 128, OpeningTiles, 4);
    InitScroll(OpeningMapWidth, OpeningMapHeight, OpeningMap, 0, 0, 4);
    SHOW_BKG;

}

void Update_STATE_OPENING() {

    //para que al pasar de nivel se alcance a leer la historia
    if (KEY_TICKED(J_B)){
        SetState(STATE_TITLE);
    }
}
```

- Solo cargar el escenario y esperar que se presione un botón en el Update

NOTA: Recuerda incluir Keys.h

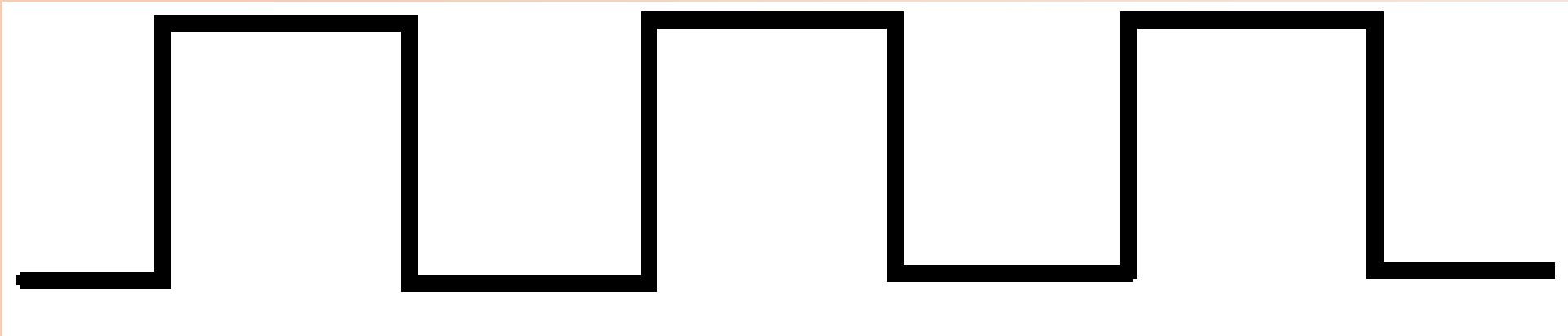

tiful
lite,
Para la wea, po

¡Metamos sonido!



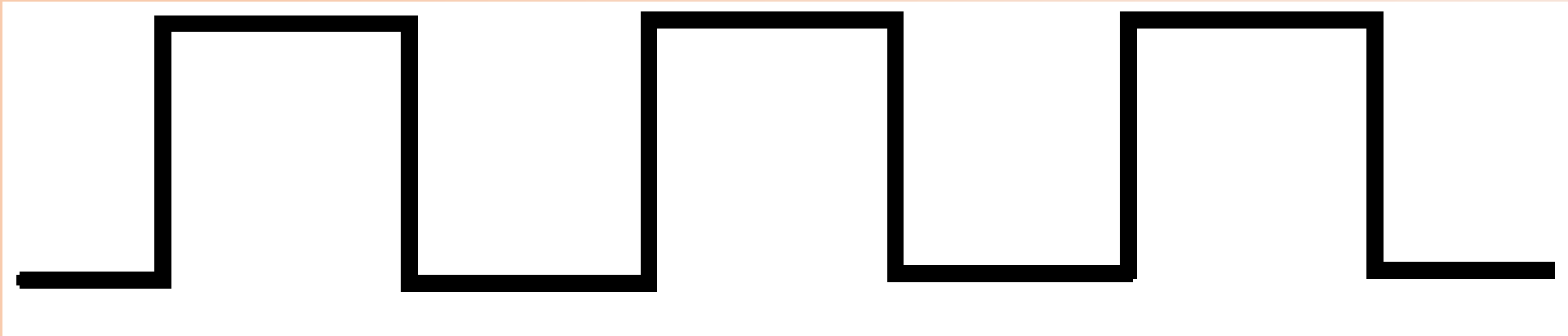
- **NOPE!** En los 90 las consolas no tenían tarjeta de sonido que reproduzca MP3.
- Ni tampoco WAV, MIDI, Señales de humo, Spotify, etc etc ¿Entonces que?
- GameBoy en su CPU **solo tiene 4 canales** que generan tonos y ruiditos, los cuales mezclandolos, se podía obtener una melodía 8-bit en estéreo! (Porque es un proce de 8 bit)
- ¡Las melodías y sonidos hay que armarlos a medida!

- Ahora bien, tampoco puedes poner cualquier cosa en estos 4 canales. Cada uno está hecho para una funcionalidad especifica.



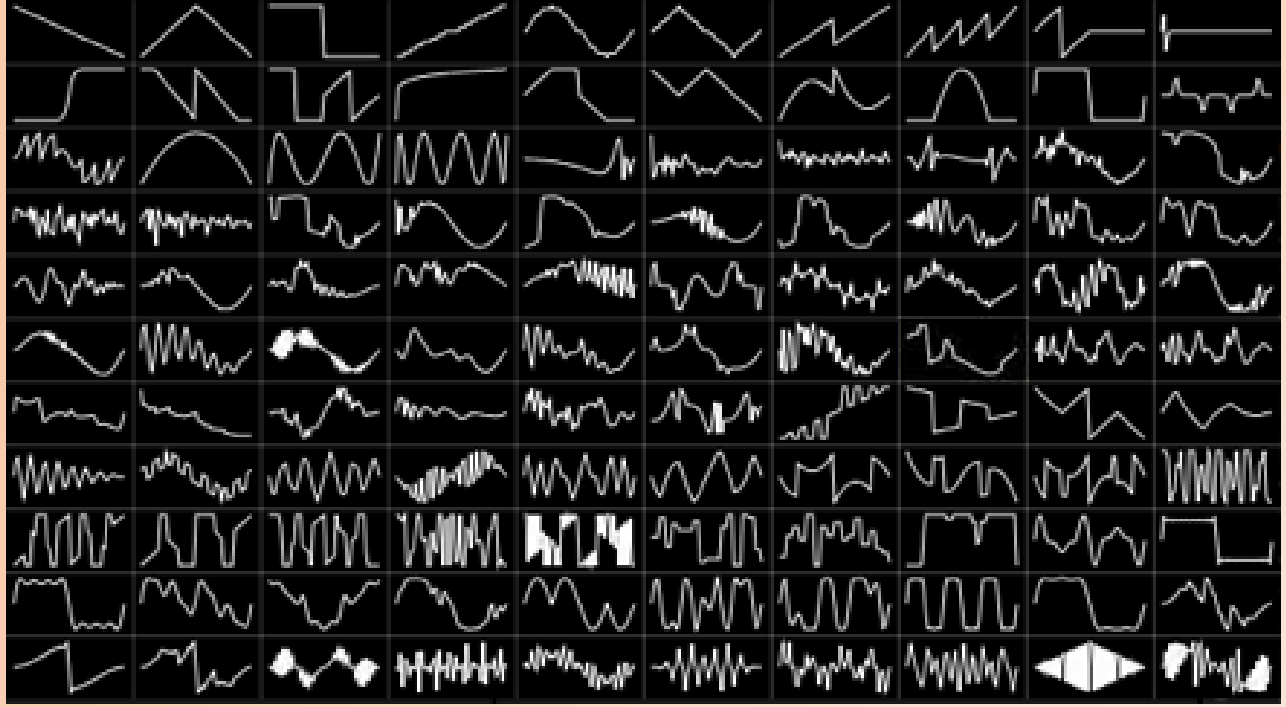
Canal 1: Una onda rectangular (square wave)

Uno la puede retocar cosillas como la amplitud, la duración, frecuencia, blablablá para ajustarla para que suene como tu quieras



Canal 2: Otra onda rectangular (square wave)

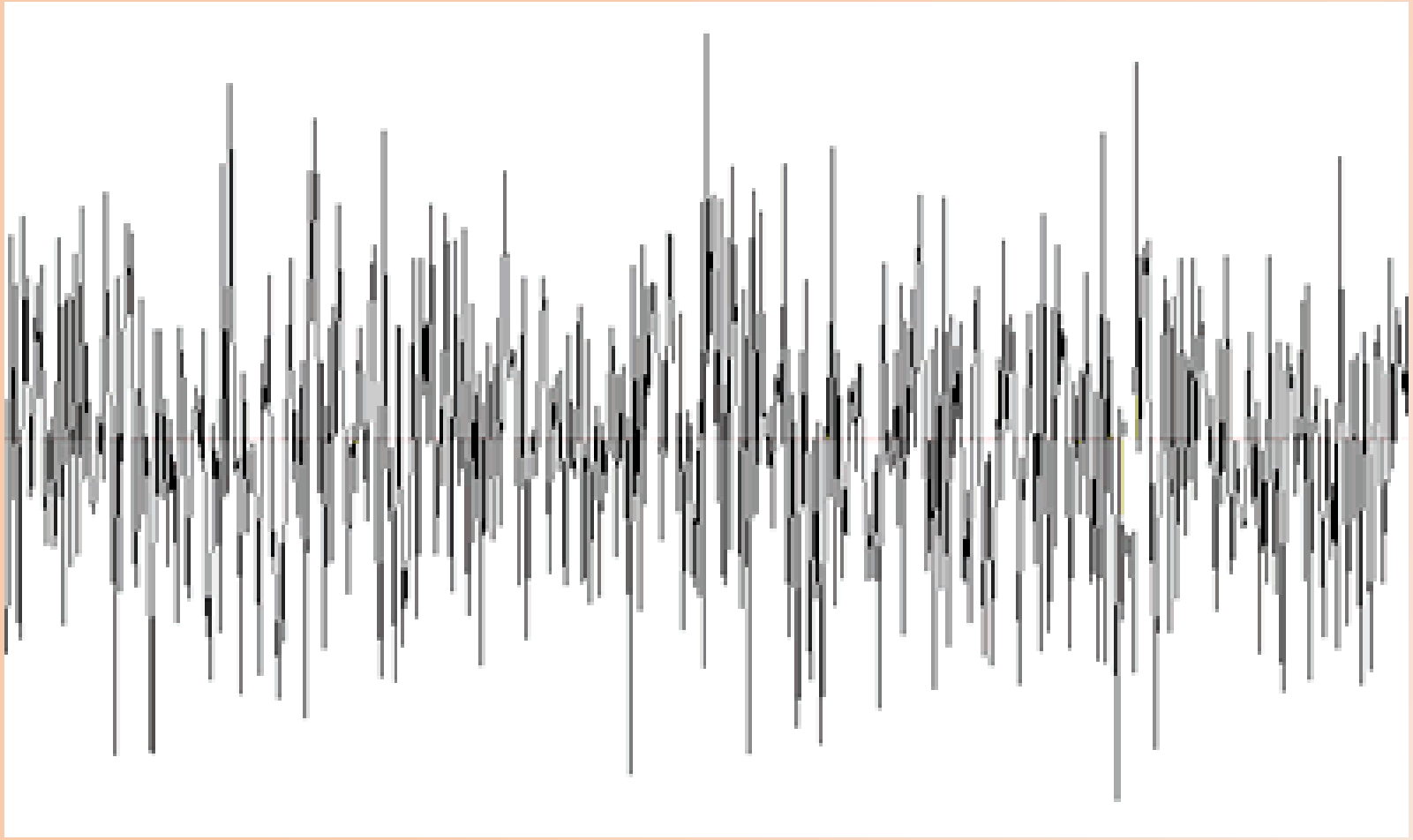
Porque no hay primera sin segunda, si ya tenias suficiente con un square wave, aquí tienes otro para hacer efectos interesantes con los 2.



Canal 3: Una tabla de ondas programables (wavetable)

Si tienes creatividad, crea tus propios instrumentos… de milésimas de segundos de duración xDDD

NOTA: Si haces miles de pedacitos cortos para el canal 3, puedes tener un audio de pocos segundos :D
Pero es demasiado engorroso, y la Gameboy ni tiene la capacidad de, así que olvídalo.



Canal 4: Generador de ruido (noise generator)

Si, el mismo sonidito cuando no teni señal en tu tele de tubo y te quedai con las ganas de ver a Luchito Jara en Mucho Gusto

Ahora, ¿que pasa cuando unimos los 4 canales?

Berklee College of Music

Be it known that having completed the requirements prescribed by the faculty of the Berklee College of Music

John L. Carlini

has been awarded the Professional Diploma with a Major in

Arranging-Composition

In witness whereof, we have affixed herewith the seal of this institution and our duly authorized signatures,

Given at Boston, Massachusetts, this month of August, 1973.

PRESIDENT

ADMINISTRATOR

DEAN

Obtienes instantáneamente un título profesional de compositor musical para videojuegos y vives feliz por el resto de tus días hasta que Kim lance la bomba H.

Una mala noticia

- **No te darán un título por unir 4 canales de sonido.**
- Para hacer sonar cualquier cosa en la GameBoy, en cualquiera de los 4 canales, deberás setear unos registros (variables mas feas) a mano!
- El valor de los registros depende del sonido que quieres sacar, en formatos bizarros.

```
NR10_REG = 0x1E;
NR11_REG = 0x10;
NR12_REG = 0xF3;
NR13_REG = 0x00;
NR14_REG = 0x87;
```

```
NR52_REG = 0x80;
NR51_REG = 0x11;
NR50_REG = 0x77;
```

- **Al menos en ZGB hay una función para eso**

```
#include<Sound.h>

...

PlayFx(CHANNEL_4, 4, 0x0c, 0x41, 0x30, 0xc0);
```

El primer y el segundo parámetro es el canal, el resto son los valores de los registros…

Otra mala noticia

¿Que valores colocar en esos malditos registros?

La documentación de eso es un mundo de fantasía y entretenimiento ilimitado!

FF10 - NR10 - Channel 1 Sweep register (R/W)

```
Bit 6-4 - Sweep Time
Bit 3   - Sweep Increase/Decrease
          0: Addition    (frequency increases)
          1: Subtraction (frequency decreases)
Bit 2-0 - Number of sweep shift (n: 0-7)
```

Sweep Time:

```
000: sweep off - no freq change
001: 7.8 ms  (1/128Hz)
010: 15.6 ms (2/128Hz)
011: 23.4 ms (3/128Hz)
100: 31.3 ms (4/128Hz)
101: 39.1 ms (5/128Hz)
110: 46.9 ms (6/128Hz)
111: 54.7 ms (7/128Hz)
```

The change of frequency (NR13,NR14) at each shift is calculated by the following formula

```
X(t) = X(t-1) +/- X(t-1)/2^n
```

FF11 - NR11 - Channel 1 Sound length/Wave pattern duty (R/W)

```
Bit 7-6 - Wave Pattern Duty (Read/Write)
Bit 5-0 - Sound length data (Write Only) (t1: 0-63)
```

Wave Duty:

```
00: 12.5% ( _-------_-------_------- )
01: 25%   ( __------__------__------ )
10: 50%   ( ____----____----____---- ) (normal)
11: 75%   ( ______--______--______-- )
```

Sound Length = (64-t1)*(1/256) seconds

The Length value is used only if Bit 6 in NR14 is set.

FF1A - NR30 - Channel 3 Sound on/off (R/W)

```
Bit 7 - Sound Channel 3 Off  (0=Stop, 1=Playback)  (Read/Write)
```

FF1B - NR31 - Channel 3 Sound Length

```
Bit 7-0 - Sound length (t1: 0 - 255)
```

Sound Length = (256-t1)*(1/256) seconds

This value is used only if Bit 6 in NR34 is set.

FF1C - NR32 - Channel 3 Select output level (R/W)

```
Bit 6-5 - Select output level (Read/Write)
```

Possible Output levels are:

```
0: Mute (No sound)
1: 100% Volume (Produce Wave Pattern RAM Data as it is)
2:  50% Volume (Produce Wave Pattern RAM data shifted once to the right)
Volume (Produce Wave Pattern RAM data shifted twice to the right)
```

33 - Channel 3 Frequency's lower data (W)

ts of an 11 bit frequency (x).

R34 - Channel 3 Frequency's higher data (R/W)

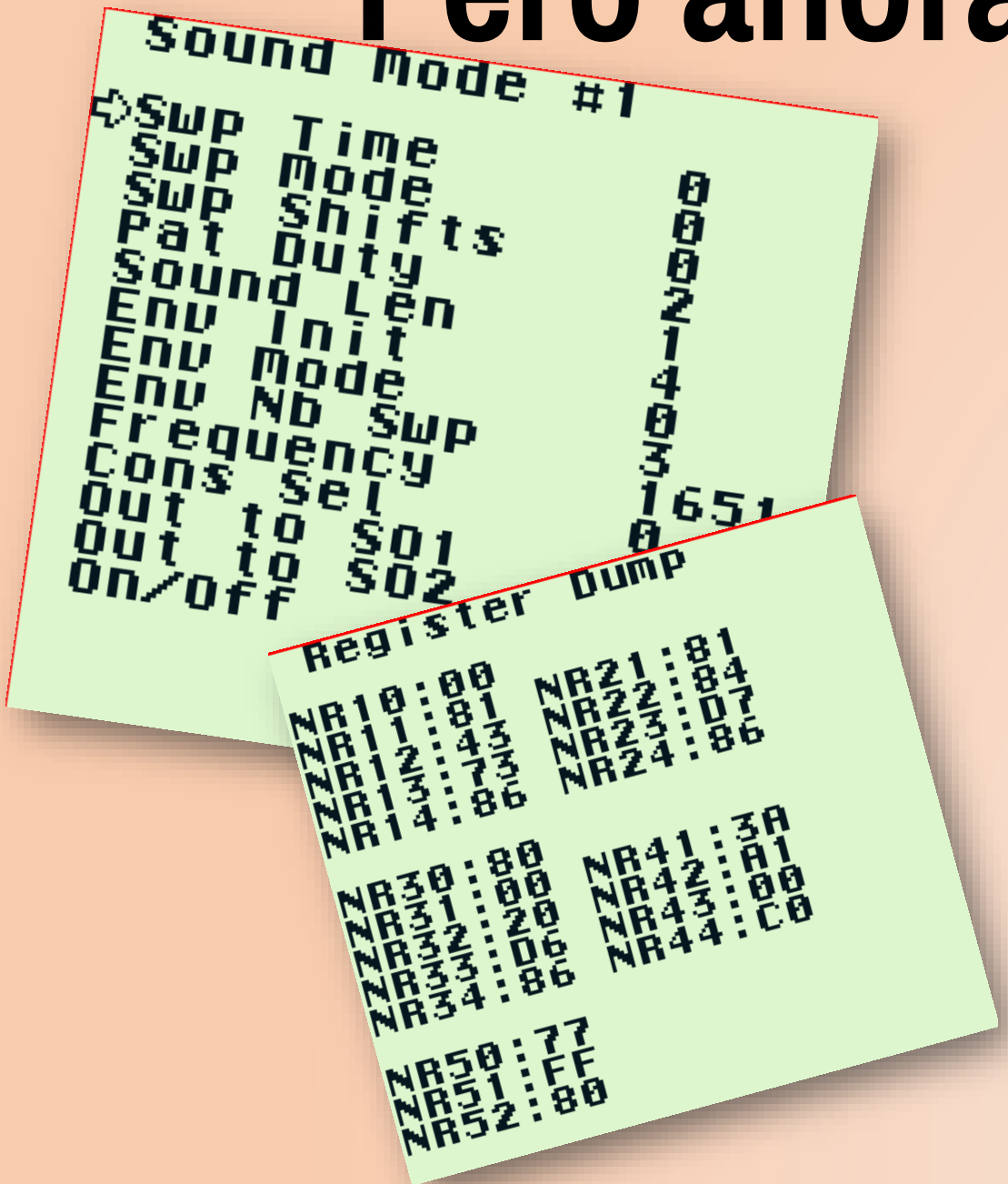
```
- Initial (1=Restart Sound)     (Write Only)
- Counter/consecutive selection (Read/Write)
  (1=Stop output when length in NR31 expires)
0 - Frequency's higher 3 bits (x) (Write Only)
```

cy = 4194304/(64*(2048-x)) Hz = 65536/(2048-x) Hz

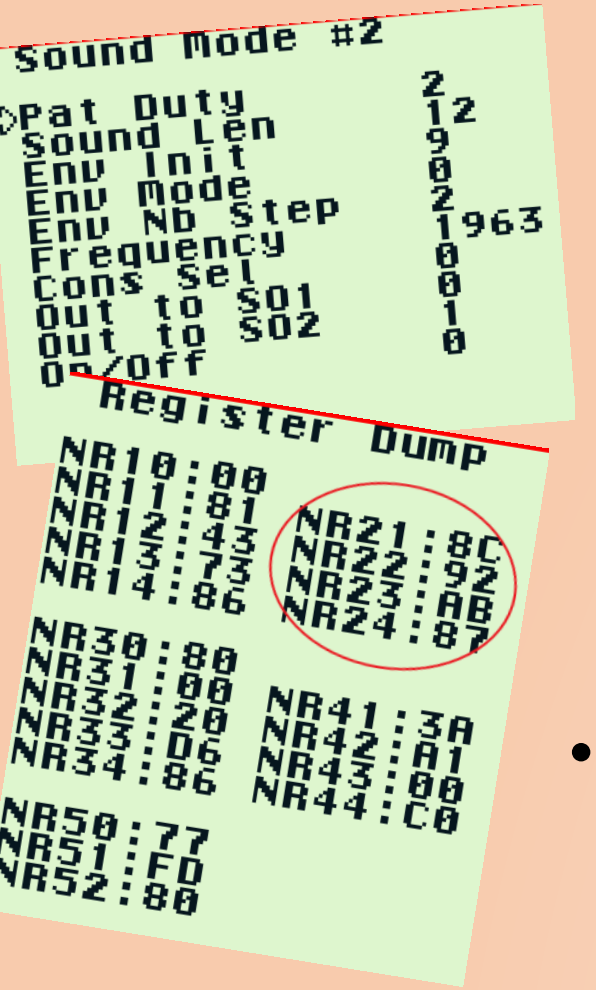
F3F - Wave Pattern RAM

s - Waveform storage for arbitrary sound data

Pero ahora, una buena noticia



- Entra a **/tools/gbt** y corre la ROM **sound.gb**
- Este pseudojuego/herramienta es un generador de sonidos para Gameboy! :D
- Elijes que canal (sound mode) quieres, ajustas los parámetros que se te de la gana, *y la herramienta genera los valores que debes ingresar para tener el resultado*!
- Con **SELECT** elijes un canal, **con las flechas** elijes y ajustas los valores (con A presionado lo haces mas rápido y con B vas a los limites). Para ver si suena bien o no, pruébalo con **START** y para finalmente generar los valores, presiona **SELECT + A.**



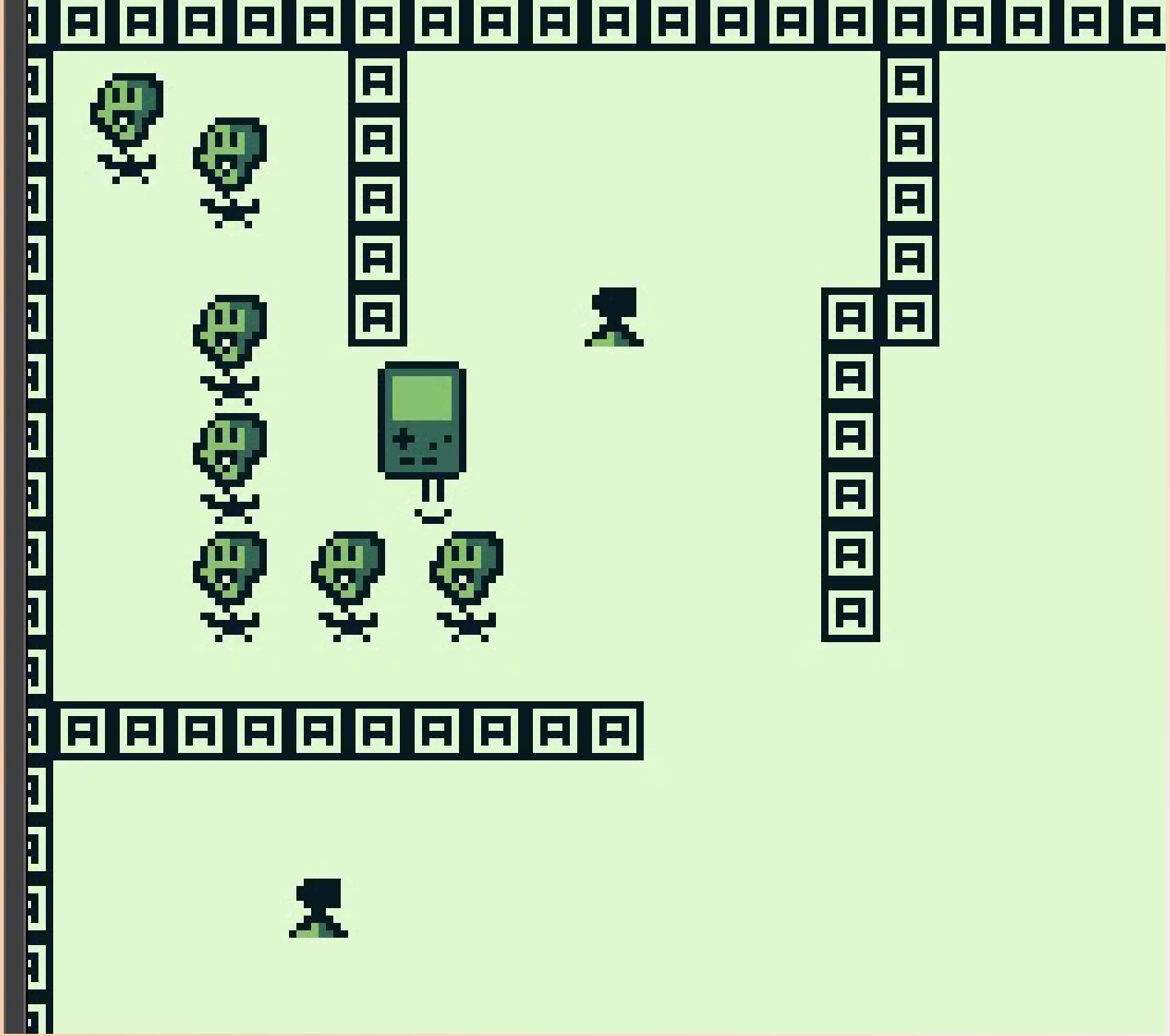
- Ahora solo te queda meter el sonido en el juego… ¿donde? ¡Donde quieras!

```
SPRITEMANAGER_ITERATE(i, spr) {
    if(spr->type == SPRITE_MONOFEO) {
        if(CheckCollision(THIS, spr)) {

            PlayFx(CHANNEL_2, 2, 0x8C, 0x92, 0xAB, 0x87);

            SetState(STATE_GAME); //reseteo la etapa
        }
    }
}
```

- ¡Pssst, no olvides de activar el sonido en la Gameboy antes (por ejemplo, al crear tu estado), y de incluir **Sound.h** en los archivos donde hagas algo con sonido!

```
//pssst, asi se activa el audio de la gameboy
NR52_REG = 0x80;    //activar sonido ...
NR51_REG = 0xFF;    // ...con todos los canales...
NR50_REG = 0x77;    // ... CON TODO EL POWER!
```

```
#include "Sound.h"
```



Ahora que ya sabes como hacer sonidos en la Gameboy

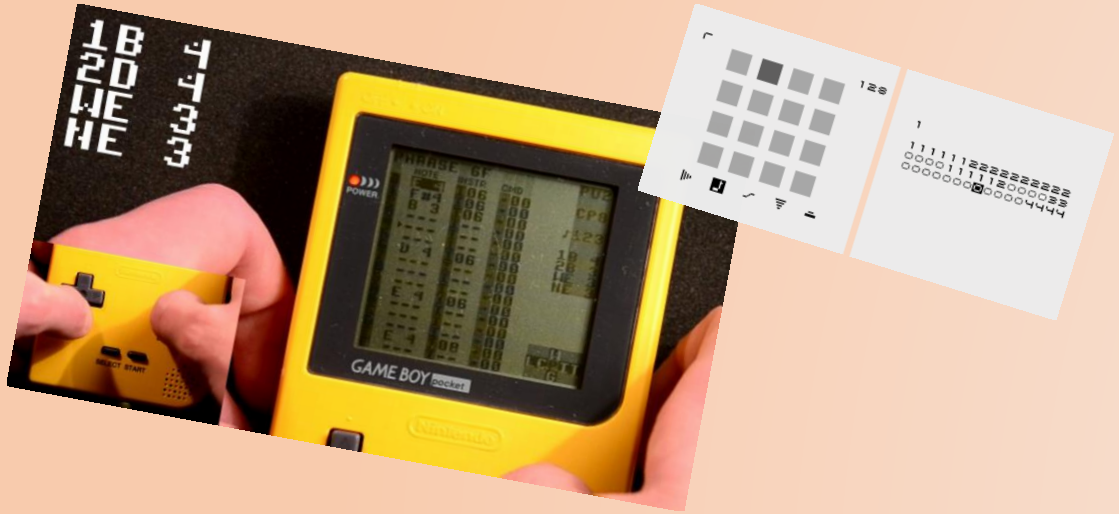
Hazte la Overtura 1812 de Tchaikovsky

(con los cañones, claro)

OH WAIT

¿En serio las melodías se hacen así?

- **En 1990 si**
- Peeero obviamente no las hacían a mano. Tampoco nosotros… ni siquiera en los días de hoy.
- Ellos usaban herramientas para componer. Esas herramientas se llaman **Trackers**
- Los trackers son como los archivos MIDI, pero 0xFFFF veces mas old school (y por ende cool)
- Ahora hay muchos Trackers para componer música para Gameboy!
- Y no, **Famitracker no es una opción** (es para NES, no Gameboy)

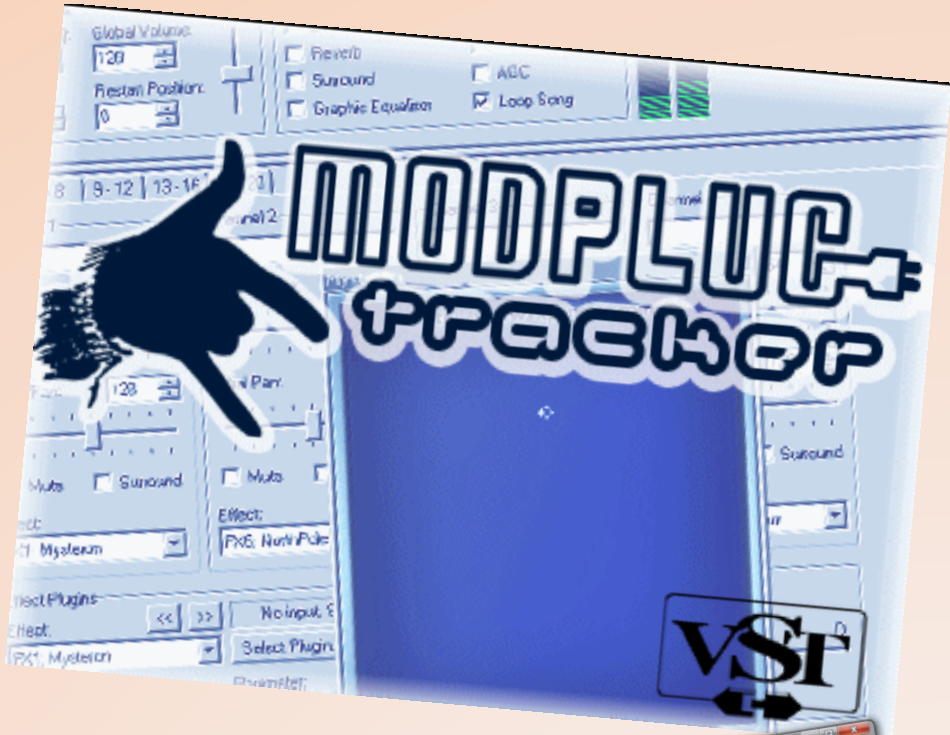


LSDj y **Nanoloop** son muy populares, pero son mas para hacer DJ sets que para juegos

Carillon no está mal, pero para esta ocasión no nos conviene ☺

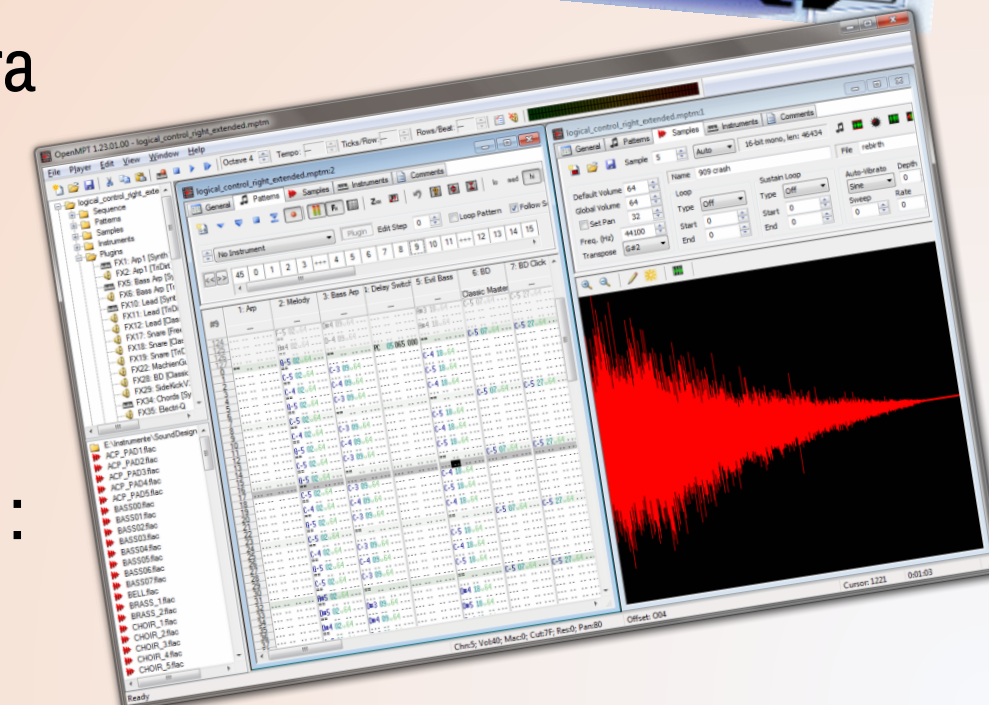
Usaremos GBT-Player

- **Es entre comillas “fácil” ™, entretenida, y funciona fantástico con ZGB (porque es del mismo autor jajaja).**



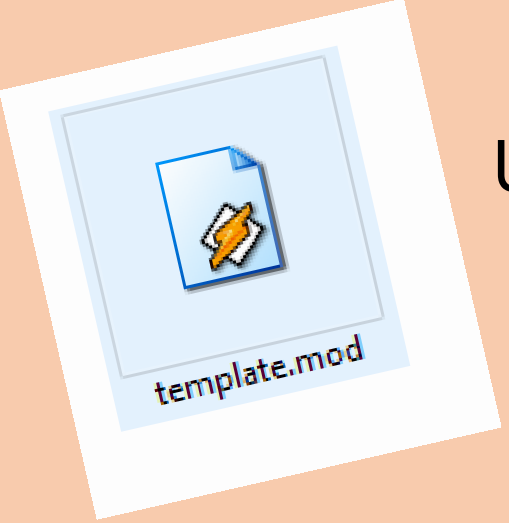
2 contras:

- Tenemos que usar los instrumentos que GBT-Player nos ofrezca. No podemos crear los propios (pero para ser sinceros, basta y sobra para ponerse creativos).
- No tiene un editor dedicado… se basa en archivos .mod (formato musical de la muuuy vieja escuela).



Asi que usaremos un editor musical para formatos .mod :

OpenMPT! (está en **/tools/openmpt**)



Usaremos el template que está en /tools/gbt

No se si nos dará el tiempo para enseñarles a componer en OpenMPT, pero les repasaré lo básico por si se entusiasman.

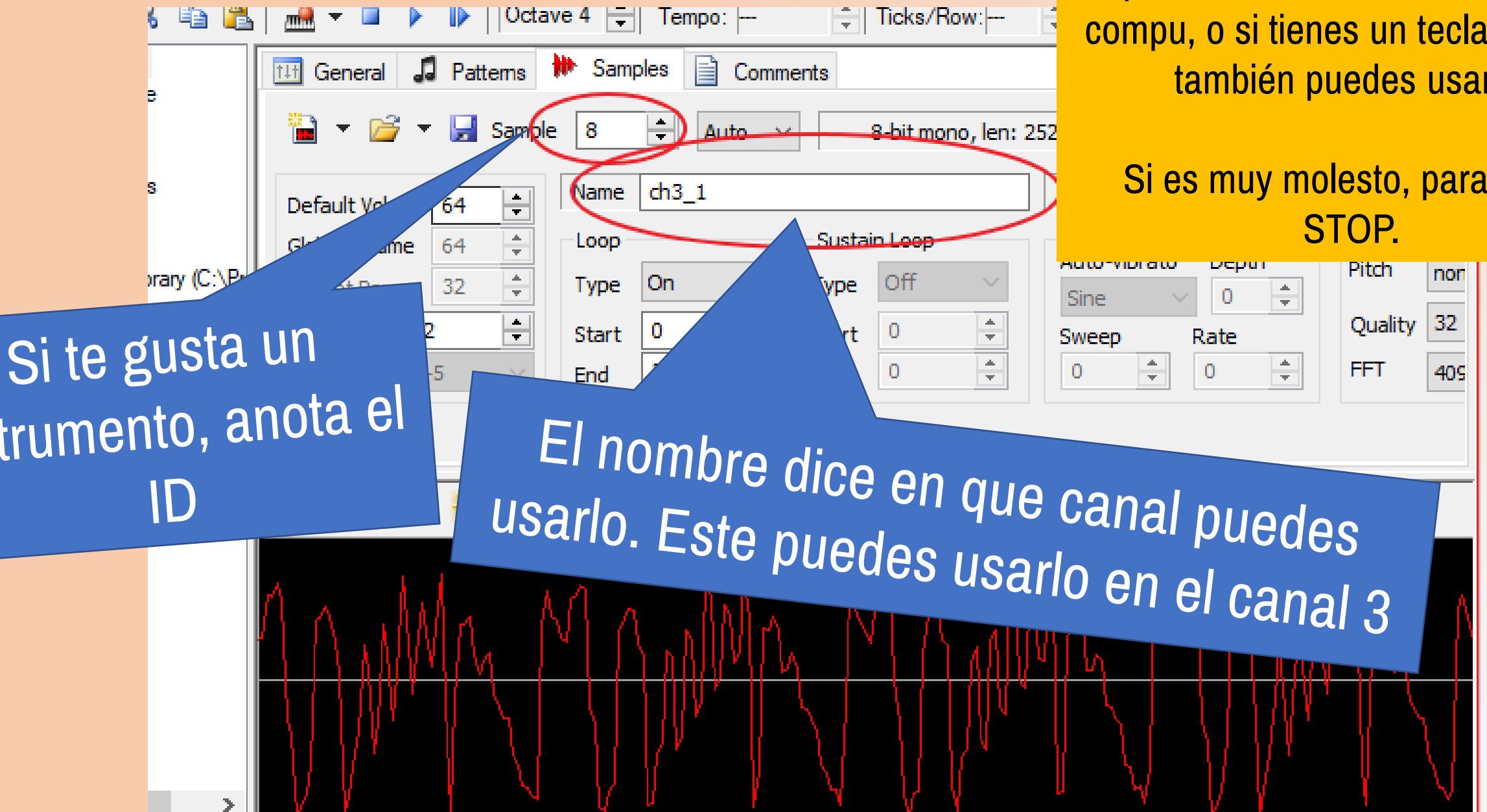
Componer música

Ver instrumentos

Ver instrumentos

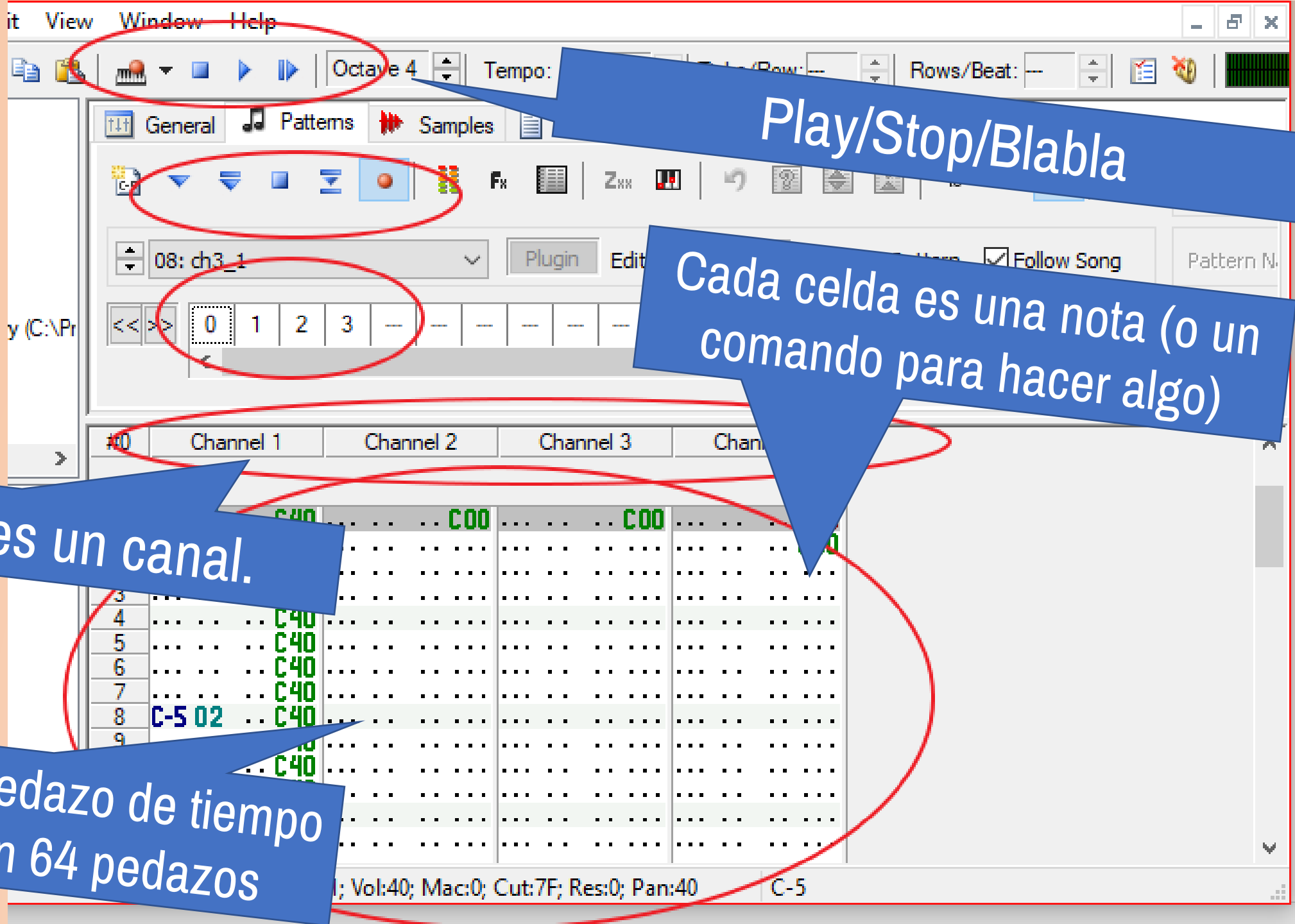
Para probar un instrumento, puedes empezar a tocar con el teclado de tu compu, o si tienes un teclado MIDI, también puedes usarlo.

Si es muy molesto, paralo con STOP.

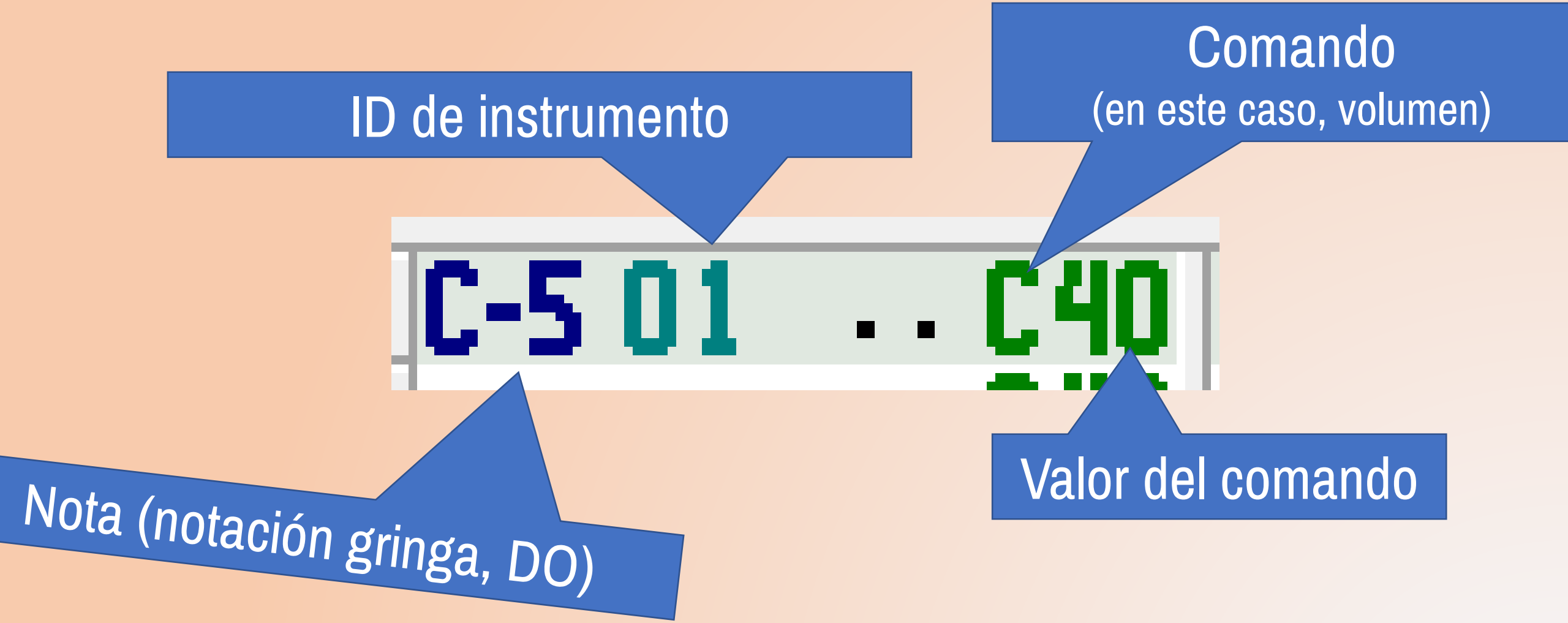


Si te gusta un instrumento, anota el ID

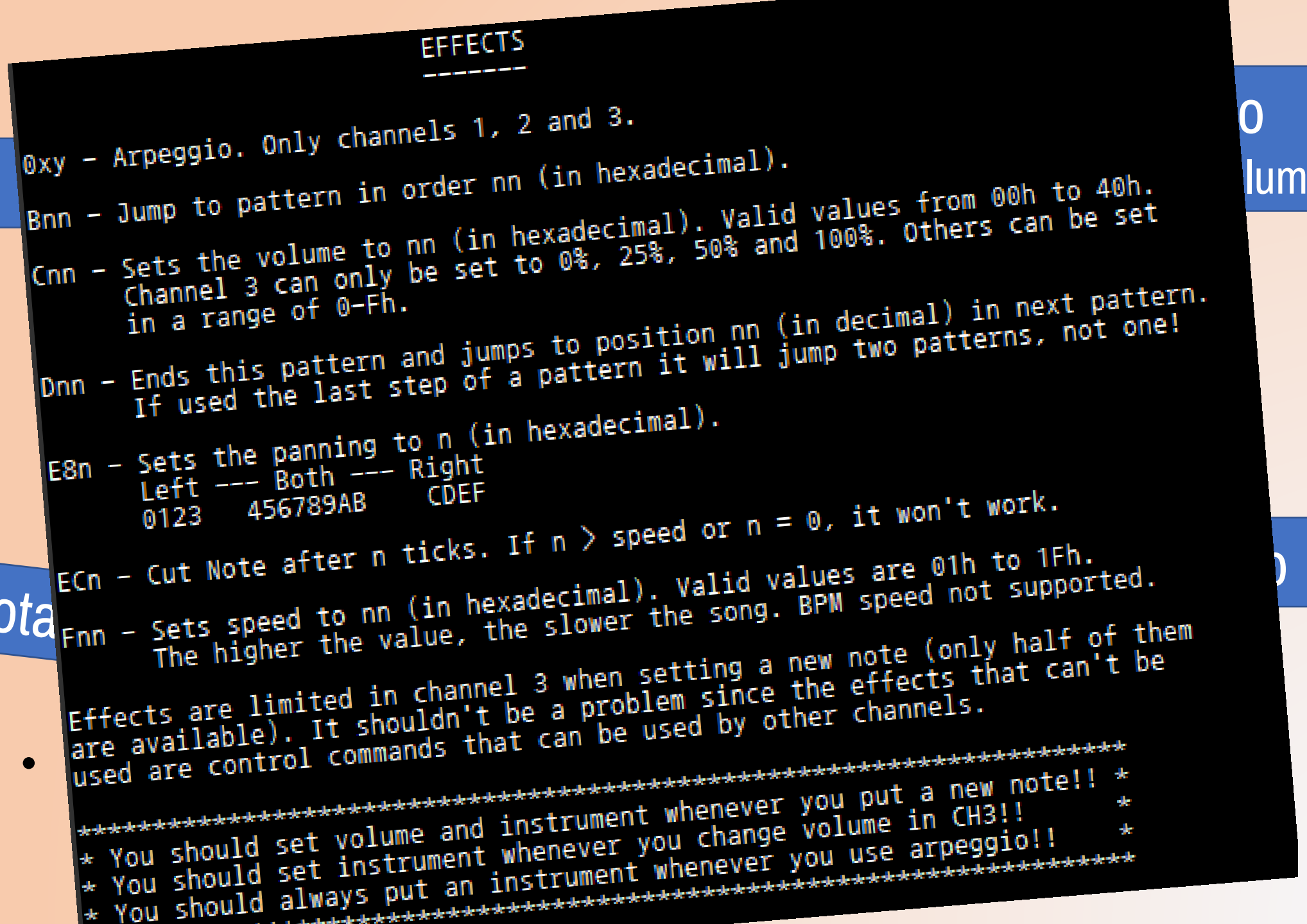
El nombre dice en que canal puedes usarlo. Este puedes usarlo en el canal 3


View
Window
Help
Octave 4
Tempo:
Rows/Beat:
General
Patterns
Samples
Play/Stop/Blabla
08: ch3_1
Plugin
Edit
Follow Song
Pattern N
Cada celda es una nota (o un comando para hacer algo)
Channel 1
Channel 2
Channel 3
Cada columna es un canal.
C-5 02
Cada fila es un pedazo de tiempo
Un compas son 64 pedazos
Vol:40; Mac:0; Cut:7F; Res:0; Pan:40
C-5

Componer



- Una lista de comandos que puedes utilizar para Gameboy la puedes encontrar en **/tools/gbt/mod_instructions.txt**



```
                    EFFECTS
                    -------

0xy - Arpeggio. Only channels 1, 2 and 3.

Bnn - Jump to pattern in order nn (in hexadecimal).

Cnn - Sets the volume to nn (in hexadecimal). Valid values from 00h to 40h.
      Channel 3 can only be set to 0%, 25%, 50% and 100%. Others can be set
      in a range of 0-Fh.

Dnn - Ends this pattern and jumps to position nn (in decimal) in next pattern.
      If used the last step of a pattern it will jump two patterns, not one!

E8n - Sets the panning to n (in hexadecimal).
      Left --- Both --- Right
      0123   456789AB    CDEF

ECn - Cut Note after n ticks. If n > speed or n = 0, it won't work.

Fnn - Sets speed to nn (in hexadecimal). Valid values are 01h to 1Fh.
      The higher the value, the slower the song. BPM speed not supported.

Effects are limited in channel 3 when setting a new note (only half of them
are available). It shouldn't be a problem since the effects that can't be
used are control commands that can be used by other channels.

**************************************************************************
* You should set volume and instrument whenever you put a new note!!     *
* You should set instrument whenever you change volume in CH3!!         *
* You should always put an instrument whenever you use arpeggio!!       *
**************************************************************************
```

Nota (nota

- lumen)


PAREN EL SEMESTRE
ME QUIERO BAJAR
memegenerator.es

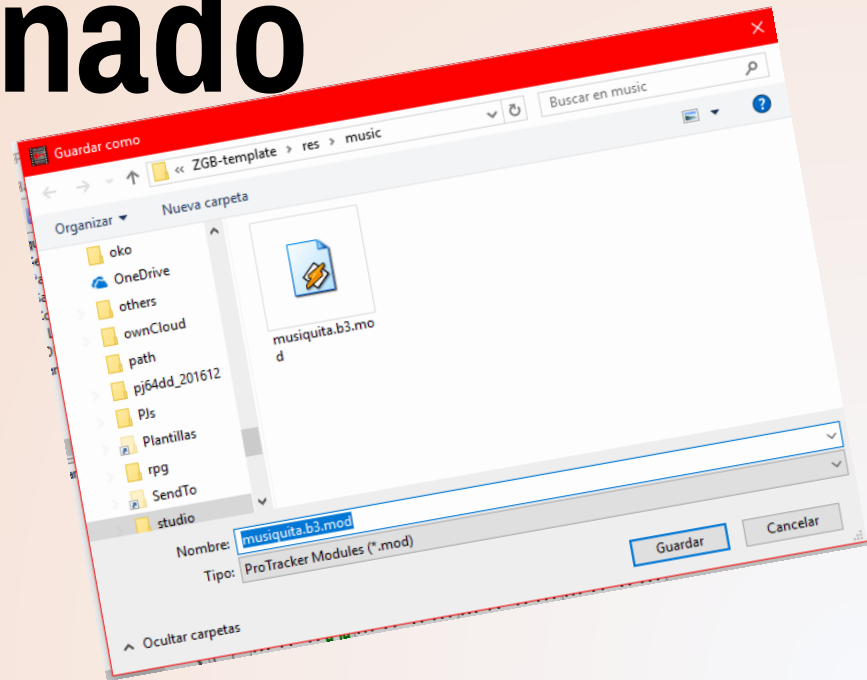
Como sea

- **Hacer melodías en OpenMPT** da para otro workshop entero… y no somos músicos tampoco…
- **Si aun así te interesa, puedes encontrar aquí un buen tutorial de OpenMPT:**

https://wiki.openmpt.org/Tutorial:_Getting_Started

Una vez hayas terminado

Guarda la melodía en tu proyecto, en la carpeta **/res/music** como **nombre.b3.mod**
(donde **b3**, al igual que las texturas, **es el banco donde lo pondrás**).
Estas se compilan a la hora de compilar la ROM ☺



¡Ahora hay que integrarla al proyecto!

- ¿Pongamosla en el State Level? Vamos a **src/State_Level.c**
- Afuera de toda función, arriba, créate esta variable global externa (nuestra melodía):

```
#include "SpriteManager.h"

#include "Sound.h"

extern UINT8 n_sprite_types;

extern UINT8* musiquita_mod_Data[];

void Start_STATE_LEVEL() {
    UINT8 i;
    UINT8 collision_tiles[] = {1, 3, 4, 0};
```

Ahem… si, asegúrate de poner **[nombreArchivo]_mod_Data**

- Démole **PLAY** a la sabrosura chico!

```
    //pssst, asi se activa el audio de la gameboy
    NR52_REG = 0x80;    //activar sonido ...
    NR51_REG = 0xFF;    // ...con todos los canales...
    NR50_REG = 0x77;    // ... CON TODO EL POWER!

    PlayMusic(musiquita_mod_Data, 3, 1);
}
```

Donde el primer parámetro es la musiquita, después el banco y si es en loop o no (o termina).

Extrañaré el beautiful life...


"That's all Folks!"

¿Y ahora que?

- ¡Es tu turno! **¡A hacer juegos se ha dicho!**
- Ahora ya tienes las herramientas básicas para crear juegos de GameBoy!
- ¿Crees que son pocas? Un juego no lo hace el engine, sino la imaginación.
- Ojo que esto es solo un tutorial básico para hacer cosas en Gameboy. Quedaron muchas cosas en el tintero, pero se hicieron esfuerzos descomunales para calzar todo esto en una hora y media xD

Ideas para desarrollar

- **GameJams!** Es una oportunidad perfecta para relucir tecnologías raras como ésta.
- **BitBitJam:** GameJam online solo para consolas viejas: Todos los años altura Julio http://bitbitjam.com
- Publicar tus juegos online y enseñarlos a la comunidad: https://gamejolt.com https://itch.io
- En un mundo lleno de Unity, algo hecho a la medida para una consola como Gameboy siempre es novedad!
- Ayudas a crecer la escena homebrew chilena :D

¿Aprender mas?

- http://gbdev.gg8.se Blog y Wiki de desarrollo de Gameboy
- https://github.com/Zal0/ZGB Repositorio y documentación oficial de ZGB
- http://gbdk.sourceforge.net Documentación oficial de GBDK
- https://github.com/avivace/awesome-gbdev Recopilacion de xuxerias para desarrollar en Gameboy y ser feliz
- https://www.youtube.com/watch?v=RZUDEaLa5Nw Gameboy autopsy. Video interesante que explica todo de la gameboy
- https://www.youtube.com/watch?v=HyzD8pNlpwI The ultimate gameboy talk!
- http://www.littlesounddj.com/lsd/ LSDj, por si quieres ser DJ con GameBoy!
- https://www.google.cl/?gws_rd=ssl Google es tu amigo!
- http://procatinator.com Hay que relajarse digo yo de vez en cuando, no?

¿Dudas antes que me escape con los millones de dólares?

Las PPT las puedes encontrar aquí en
http://elsemieni.net/gameboy/

Videojuegos retro en Gameboy ... de verdad!

(Parte 2)

Enzo Barbaguelatta D.

MHW

Escuela de Ingeniería Informática