

# Create your own Spectrum game Workshop

## using *The MK2 Engine*

### Tutorial 1

[Construction of screens in MK2](#)

[Background](#)

[Map](#)

[Decorations](#)

[Decorations, using scripting](#)

[Decorations without using scripting](#)

### Tutorial 2

[Modifying a hitter](#)

[Modifying the trajectory and number of frames](#)

[Altering the "interval hot hitter"](#)

[Changing the "hot spot"](#)

[Changing the Graphic](#)

### Tutorial 3

[Building a 128K game with several levels \(simple\)](#)

[Configuration](#)

[Building the binary for each level](#)

[Changes to make.bat](#)

[Librarian](#)

[The script](#)

### Tutorial 4

[New enemies module](#)

### Version Differences

#### Version 3.99.2

[Timers](#)

[Scripting](#)

[Checks](#)

[Commands](#)

[Control of push blocks](#)

[Check if we get off the map](#)

[Type of enemy "custom" gift](#)

[Keyboard / joystick configuration for two buttons](#)

[Shooting up and diagonally for side view](#)

[Masked bullets](#)

#### Version 3.99.2 mod

[Animated Tiles](#)

#### Version 3.99.3

[Animated Tiles](#)

[128K Mode](#)

[Type 3 Hotspots](#)

[Pause / Abort](#)

[Message catching objects](#)

#### 3.99.3b

### 3.99.3c

Items Engine

MT Engine MK2 v 0.8

Floating Objects

Carriable boxes

Item containers

Alias in script

MT Engine MK2 v 0.85

Throwing Carriable Boxes

Single-display mode

Counting Enemies

Show level

Enemies Module

Old Style Enemies

Disable Platform

Resurrecting Enemies

MT Engine MK2 v 0.86

MT Engine MK2 v 0.87

The new module of enemies

Modifications to floating objects

Floating object that hurt

Floating objects with Springs

Scripting FO

MT Engine MK2 v 0.88

Scripting stuff without scripting

Additional Tiles Impressions

Simple Item Manager (SIM)

SIM IMPORTANT NOTE

Improvements in JETPAC

MT Engine MK2 v 0.88c

Warning

To do

Decorations

Re-Enter, Redraw, Rehash

Change enemies and backup enemies

Sword

More things

MT Engine MK2 v 0.89 (Nicanor Edition)

Easier Scripting with Alias

Safe Spot

The MK2 Engine is written by the Mojon Twins as an evolutionary upgrade of the MK1 Engine also known as the Churro Maker. The tutorials are written by Nathan to describe the exciting features of the MK2 engine.

The MK2 Engine is a natural progression of the Churro Maker Engine with many of the features used in the Churro Maker Engine. MK2 code is more modularized and 90% of the code has been rewritten for speed and lower memory constraints. There is more support for 128k available with the MK2 engine as well as sound. Most programs written in the Churro Engine can be used with the MK2 Engine with some minor changes. With that in mind, the MK2 Engine is stronger, faster and smaller.

This document is more technical than the Churro Maker Engine, so it is best to read the Churro Maker document before tackling the MK2 engine.

Enjoy the reading about the advances of the MK2 Engine.

# Spectrum games using the MK2 engine

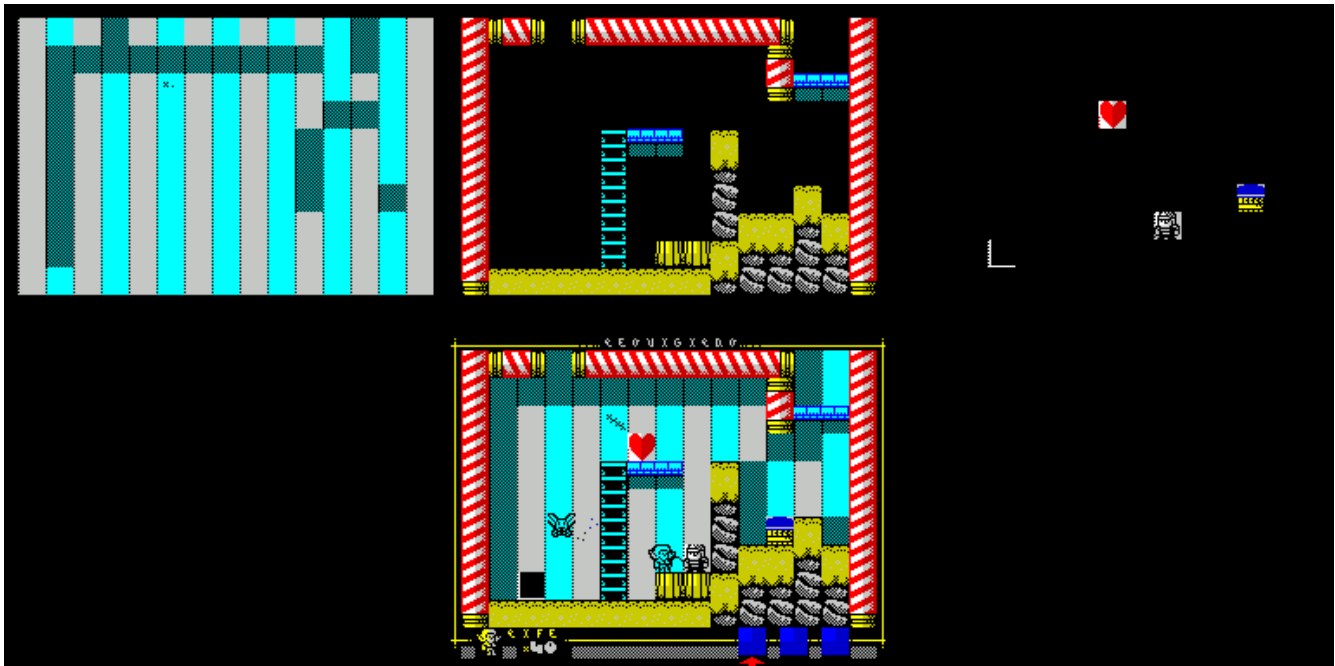
## Tutorial 1

### Construction of screens in MK2

This tutorial applies to 0.88c (or later) version of MK2

MK2 has a construction engine greatly improved screens that allows us to get very good results without using a map "spread" of 48 tiles. To do this, the screen is divided into three logical levels: background, map and decorations. Of course, you can continue to use normal maps of 16 or 48 tiles exactly as you did in La Churrera (well, almost the same: now the map converter will detect if your map is automatically packed or unpacked).

To see exactly how it works take as an example this screen of the third load of Leovigildo:



### Background

To use a background, we will have to leave the empty tile 0 and designing our map considering that the fund will look through the boxes where we put this tile 0. It is as "transparent tile".

The background is programmatically generated by modifying a certain section of the `draw_scr_background` function defined in the `/dev/engine/drawscr.h` file. What is done is to detect if the tile that we extracted from the map is worth 0, in which case it is replaced by a tile of background. Basically, it's about adding code in this chunk:

```
// CUSTOM {  
// Construction of assets.  
    if (0 == gpd) {  
        }  
// } END OF CUSTOM
```

Here we can do practically what we want. Using `gpx` and `gpy`, which are the coordinates of the tile being drawn, we just have to write the correct value in `gpd`. For example, we can use a 15x10 tiles tilemap with a predesigned background, as done in *Sir Ababol 2 48K*, something like this (defined, for example, right at the beginning of `drawscr.h`):

```
// The tilemap of 15x10 tiles:  
  
unsigned char backdrop [] = {  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 21, 0,  
    0, 0, 0, 19, 21, 0, 0, 19, 21, 0, 0, 19, 20, 20, 21,  
    0, 0, 19, 20, 20, 21, 19, 20, 20, 21, 19, 20, 20, 20, 20,  
    0, 19, 20, 20, 20, 20, 21, 20, 20, 19, 20, 20, 20, 20, 20,  
    19, 20, 20, 20, 20, 20, 20, 21, 19, 20, 20, 20, 20, 20, 20,  
};
```

In which case, our code building the fund would be something as simple as:

```
// CUSTOM {  
// Construction of assets.  
    if (0 == gpd) {  
        if (gpy >= 5) {  
            gpd = backdrop [gpx + gpy * 15];  
        }  
    }  
// } END OF CUSTOM
```

We can complicate the topic as much as we want, as long as the result ends in `gpd` (which indicates the tile number that will end up being painted, instead of 0). For example, *Leovigildo III* uses a code that paints the vertical stripes, adds imperfections randomly, and places shadows for the tiles on the map that are obstacles:

```
// CUSTOM {
// Construction of assets.
//   if (0 == gpd) {
//     gpd = (attr (gpx - 1, gpy) > 4 || attr (gpx, gpy - 1) > 4 || attr (gpx - 1, gpy - 1) > 4) ? 22 : 18 + (gpx & 1) + (gpjt ? 0 : 22);
//   }
// } END OF CUSTOM
```

The possibilities are endless.

## Map

The map layer is formed with tiles that are removed from the map you created in Mappy, neither more nor less. If you have added code to replace tile 0 with a background, these tiles will not be drawn.

If we are going to use the new screen layout engine, it is silly to use 48-tile maps, because in these we can do everything directly on the map. With drawing our map in mappy using only the first 16 tiles, the map converter will detect it and create a packed map of 16 tiles.

## Decorations

The decorations are extra tiles that are printed on the two previous layers at the end of the process. This can be done by scripting or using the new `/dev/engine/extraprints.h` module if we are not using scripting in our game (as in the fourth Leovigildo load).

In any case, we will not have to enter the decorations by hand, but the map converter itself will detect them in our `.map` file and export them. For this we just have to pass the parameter "forced". If we use this parameter, when the converter detects a `tile>= 16`, instead of exporting a map in unpacked format, it will ignore those tiles in the exported map (which will therefore be "packed") and add them to A list which it will subsequently export separately:

```
..\utils\map2bin.exe ..\map\mapa.map 3 3 99 map.bin bolts.bin force
```

## Decorations, using scripting

`map2bin map.bin.spt` generate a file (if you have specified that the output is written to `map.bin` to invoke) that must copy the folder `/ script`, along with our main script. The `map.bin.spt` file will include an ENTERING SCREEN xx section for each screen with decorations. Inside, you will have an IF TRUE THEN ... END block with the new DECORATIONS ... END command in which you define a list of tiles that change. In this way, each decoration uses only 2 bytes of the script (compared to the 4 that occupied in the Churrera or in MK2 <0.8, for example), something like this:

```

[...]
ENTERING SCREEN 2
  IF TRUE
  THEN
    DECORATIONS
      7, 1, 16
      8, 1, 18
      7, 3, 22
      8, 3, 23
      7, 4, 25
      8, 4, 24
      1, 7, 27
      1, 8, 26
      7, 8, 19
      8, 8, 20
      9, 8, 20
      10, 8, 20
      11, 8, 21
    END
  END
END
[...]

```

(Each line of Decorations defines X, Y, T for a decoration tile and we can use it anywhere in the script that supports commands – this is great for modifying a torch part of the stage – it's like a sequence of SET\_TILE X, Y) = T).

Once this is done, we will only need to add this line at the beginning of the script (in multi-level game scripts, we will have to add it at the beginning of the script section for the corresponding level):

```
INC_DECORATIONS map.bin.spt
```

When msc3 finds this line, it will include the code generated by map2bin and stored in map.bin.spt in the main script at the beginning of each ENTERING SCREEN block.

### Decorations without using scripting

In the map.bin.spt file, in the end, be C code with definitions of arrays, something like this:

```

// If you use extraprints.h, trim this bit and use it!
const unsigned char ep_scr_00 [] = { 0x21, 16, 0x31, 17, 0x41, 18, 0xC1, 16, 0xD1, 18, 0x23, 22, 0x33, 23, 0x43, 22, 0xA3,
22, 0xB3, 23, 0x24, 25, 0x34, 24, 0xA4, 25, 0xB4, 24, 0x76, 28, 0x47, 29, 0x38, 19, 0x48, 21, 0xA8, 19, 0xB8, 20, 0xC8, 21
};
const unsigned char ep_scr_01 [] = { 0xC3, 18, 0x74, 16, 0x84, 18, 0x25, 27, 0xC5, 27, 0x26, 28, 0x76, 27, 0xC6, 26, 0x77,
29, 0xC7, 28, 0x88, 19, 0x98, 20, 0xA8, 20, 0xB8, 21 };
const unsigned char ep_scr_02 [] = { 0x71, 16, 0x81, 18, 0x73, 22, 0x83, 23, 0x74, 25, 0x84, 24, 0x17, 27, 0x18, 26, 0x78,
19, 0x88, 20, 0x98, 20, 0xA8, 20, 0xB8, 21 };
const unsigned char ep_scr_03 [] = { 0x42, 23, 0x33, 25, 0x43, 22, 0x53, 24, 0x73, 18, 0x44, 24, 0x84, 18, 0xA4, 16, 0x95,
17, 0x18, 19, 0x28, 21, 0x48, 19, 0x58, 20, 0x68, 21 };
const unsigned char *prints [] = { ep_scr_00, ep_scr_01, ep_scr_02, ep_scr_03, 0, 0, 0, 0 };

```

If we want to use decorations with games without scripting, the first thing we have to do is configure this feature in our `/dev/config.h` file:

```
#define ENABLE_EXTRA_PRINTS
```

Once this is done, we will have to trim the code included in `map.bin.spt` and paste it to the beginning of `/dev/engine/extraprints.h`, right where it puts...

```
// [[[ PASTE HERE THE OUTPUT AT THE END OF THE .SPT FILE THAT MAP2BIN PRODUCES ]]]
```

You can see `extraprints.h` in action (and other things that will help you create simple video without scripting) in the fourth charge of Leovigildo.



# Spectrum games using the MK2 engine

## Tutorial 2

### Modifying a hitter

This tutorial applies to 0.88c (or later) version of MK2

MK2 implements two hitters (for the moment, exclusive) that can be activated from /dev/config.h:

```
#define PLAYER_CAN_PUNCH  
#define PLAYER_HAZ_SWORD
```

Activating one of these Two (only one) will be activating the hitter. By default, the first defines a punch that can be given to either side, and the second a punch that can be left, right or up (always in side view, and we will carry something genital when necessary).

Of course, the graphics, travel and behavior of hitters is easily modifiable. We will see here how to modify the sword, the cuffs are simpler and the process is practically the same.

### Modifying the trajectory and number of frames

The main behavior hitters and, in this case, the sword is in the /dev/engine/hitter.h file. Almost at the beginning of it you will see something like this:

```
#ifndef PLAYER_HAZ_SWORD  
unsigned char swoffs_x [] = {8, 10, 12, 14, 15, 15, 14, 13, 10};  
unsigned char swoffs_y [] = {2, 2, 2, 3, 4, 4, 5, 6, 7};  
#endif
```

These two arrays define the position of the sword (which is a sprite of 8x8) with respect to the sprite of the protagonist during all the pictures that lasts its animation. Specifically, if we know how to count, we see that there are nine steps of animation.

When the player shoots, the sword is activated. This activation lasts nine frames. Each frame, the sword is placed in the position indicated by the corresponding index of those two arrays, always taking as origin the position of the player.

When the player looks to the right or to the left:

- The "and" position where the sword is drawn will be  $gpy + swoffs\_y[box]$ , where  $gpy$  is the player's "y" position and frame goes from 0 to 9.
- If the player looks to the right, the "x" position will be  $gpx + swoffs\_x[box]$ , and if he looks to the left it will be  $gpx + 8 - swoffs\_x[box]$ , where  $gpx$  is the "x" position of the player and  $box$  goes From 0 to 8.

When the player throws the sword up, `swoffs_x` and `swoffs_y` are swapped to allow the movement to be vertical:

- The "y" position where the sword is drawn will be `gpy + 6 - swoffs_x [box]`.
- The "x" position where the sword is drawn will be `gpx + swoffs_y [box]`.

Modifying `swoffs_x` and `swoffs_y` we can modify the course of the sword. As it is, the sword makes a kind of curve.

We can also increase or decrease the number of frames in the animation. We will put a faster sword, we will remove all the even pairs and we will only be left with five pictures of animation. We do this, literally: we load each even value of the arrays. We would stay:

```
#ifndef PLAYER_HAZ_SWORD
unsigned char swoffs_x [] = {8, 12, 15, 14, 10};
unsigned char swoffs_y [] = {2, 2, 4, 5, 7};
#endif
```

Since we have changed the number of tables, we have to modify the code a bit. From line 88 is the code that detects that all the pictures of the sword have passed. Now we have 5 frames instead of 9, so we'll have to leave this like this:

```
#ifndef PLAYER_HAZ_SWORD
if (hitter_frame == 5) {
#endif
```

### Altering the "interval hot hitter"

call "hot hitter interval" to pictures of animation hitter in which is considered that the hitter is hitting. For example, in Ninजार! The fist does not hit while retracting. With the sword, and considering that the paintings originally ranged from 0 to 8, the warm hitter interval was frames 3 to 6, both inclusive:

FRAME:	0	1	2	3	4	5	6	7	8
GOLPEA:	NO	NO	NO	YES	YES	YES	YES	NO	NO

We will have to modify these intervals in two places: when it is detected that we hit a destructible tile (if we are using them), and when it is detected that we hit an enemy.

Let's leave the interval like this, now that we have fewer frames:

FRAME:	0	1	2	3	4
GOLPEA:	NO	YES	YES	YES	NO

In /dev/engine/hitter.h, from line 72, it is where a destructive tile is detected and it is commanded to destroy if we are in the hot hitter range. If you look, the interval is defined if the box > 2 and box < 7. We will change it so that tables 1, 2 and 3 are active, that is, if box > 0 and box < 4. It looks like this:

```
#if defined (BREAKABLE_WALLS) || defined (BREAKABLE_WALLS_SIMPLE)
    if ((attr (gpxx, gpyy) & 16) && (hitter_frame > 0 && hitter_frame < 4))
        break_wall (gpxx, gpyy);
#endif
```

In /dev/engine/enemmods/hitter.h, on line 7, we have a similar range detection, this time to detect if the sword is "hot" when colliding with an enemy. In the same way, we change it to make it look like this:

```
if (hitter_frame > 0 && hitter_frame < 4) {
```

### Changing the "hot spot"

Collision detection hitter with stage or enemies is based on a pixel in the sprite. By default, this pixel is:

- Sword up: (hitter\_x + 4, hitter\_y)
- Sword left: (hitter\_x, hitter\_y + 4)
- Sword right: (hitter\_x + 7, hitter\_y + 4)

To change this we will have to change the assignments to gpxx and gpyy that are in the block of code that governs the positioning of the sword with respect to the player in /dev/engine/hitter.h, that is, here:

```

#ifdef PLAYER_HAZ_SWORD
    if (p_up) {
        hitter_x = gpx + swoffs_y [hitter_frame];
        hitter_y = gpy + 6 - swoffs_x [hitter_frame];
        hitter_next_frame = sprite_sword_u;
    #if defined (BREAKABLE_WALLS) || defined (BREAKABLE_WALLS_SIMPLE)
        gpxx = (hitter_x + 4) >> 4; gpyy = (hitter_y) >> 4;    // <<< AQUI (x, y espada hacia arriba)
    #endif
    } else {
        hitter_y = gpy + swoffs_y [hitter_frame];
    #if defined (BREAKABLE_WALLS) || defined (BREAKABLE_WALLS_SIMPLE)
        gpyy = (hitter_y + 4) >> 4;    // <<< AQUI, (y espada horizontal)
    #endif
    if (p_facing) {
        hitter_x = gpx + swoffs_x [hitter_frame];
        hitter_next_frame = sprite_sword_r;
    #if defined (BREAKABLE_WALLS) || defined (BREAKABLE_WALLS_SIMPLE)
        gpxx = (hitter_x + 7) >> 4;    // <<< AQUI, (x espada hacia la derecha)
    #endif
    } else {
        hitter_x = gpx + 8 - swoffs_x [hitter_frame];
        hitter_next_frame = sprite_sword_l;
    #if defined (BREAKABLE_WALLS) || defined (BREAKABLE_WALLS_SIMPLE)
        gpxx = (hitter_x) >> 4;    // <<< AQUI, (x espada hacia la izquierda)
    #endif
    }
    }
    #if defined (BREAKABLE_WALLS) || defined (BREAKABLE_WALLS_SIMPLE)
        if ((attr (gpxx, gpyy) & 16) && (hitter_frame > 2 && hitter_frame < 7))
            break_wall (gpxx, gpyy);
    #endif
#endif

```

The hot spot is located in a fairly convenient position, I doubt you have to change this unless you draw a very bizarre sword.

## Changing the Graphic

The sword is defined in /dev/extrasprites.ha from the line 259. `_sprite_sword_l` labels, `_sprite_sword_r` and `_sprite_sword_u` contain the frames for the sword to the left, right and up, respectively, using the 4 blocks Of 8×8 pixels with usual masks.

# Spectrum games using the MK2 engine

## Tutorial 3

### Building a 128K game with several levels (simple)

This tutorial applies to version 0.89 (or later) MK2

The issue of multi-level games are much given to personalization engine. The file included levels128.h and clevels.h handler contains two examples of how:

- The "Goku Evil" mode, in which each level is a compact entity formed map, bolts, behaviors, enemies, hotspots, and tileset Spriteset contained in a single binary.
- The "Ninjar!" Mode, in which each level consists of a series of "assets" stored in the extra RAM that can be combined in any way, besides including a structure of sequence of levels in case we want to repeat a level several Times (for example, Ninjar stores!, which are all on the same real level).

In this brief tutorial we will explain how to mount a 128K multilevel game with simple level handler (Goku Mal).

### Configuration

In config.h have to establish certain policies to achieve what we want. The first are the basic ones, then I put some more than another that comes well (at least for me, you can use something else).

You have to activate these:

```
#define MODE_128K
#define COMPRESSED_LEVELS
```

And I think its is that if you use scripting (because you use scripting!!) activate this one:

```
#define SCRIPTED_GAME_ENDING
```

Notice we DO NOT activate this:

```
//#define EXTENDED_LEVELS
```

This is what he does is put all the ninjar tray! And for now we will leave that quiet. Let's create a simple handler game.

## Building the binary for each level

to build the binary for each level we will use the buildlevel.exe utility that we, like all others in the /utils. If we execute it to pelaco we obtain the parameters that it takes, that are a good billet:

```
D:\Dropbox\nicanor\desarrollo\dev>..\utils\buildlevel.exe
buildlevel v 0.2 [MK2]
usage:

$ buildlevel mapa.map map_w map_h lock font.png work.png spriteset.png enems.ene scr_ini x_ini y_ini max_objs
enems_life behs.txt level.bin

where:
• mapa.map is your map from mappy .map
• map_w, map_h are map dimmensions in screens
• lock is 15 to autodetect lock, 99 otherwise
• font.png is a 256x16 file with 64 chars ascii 32-95
• work.png is a 256x48 file with your 16x16 tiles
• spriteset.png is a 256x32 file with your spriteset
• enems.ene enems/hotspots directly from colocador.exe
• scr_ini, scr_x, scr_y, max_objs, enems_life general level data header
• behs.txt is a tile behaviours file
• level.bin is the output filename.
• decorations.spt if specified, maps are forced to 16 tiles + decorations
```

We are going to use this convention not to mess with filenames (especially if we use many levels), where N is the number of the level:

- MapN.map: The map.
- WorkN.png: The tileset (256x48).
- SpritesN.png: The spriteset (256x32).
- EnemsN.ene: enemies / hotspots.
- BehsN.txt: Behaviors of tiles.
- LevelN.bin: The output file.
- DecorationsN.spt: Decorations to include in the script.

We see several things we need, therefore:

- Map, tileset, spriteset, enemies, as always.
- Behaviors: simply an ordered list of 48 values, like the one we used to put in config.h, something like this:

```
0,24, 8, 8, 8, 8, 8, 8, 1, 1, 8, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

- font.png: It is a 256x16 file with 64 characters of text font (ascii 32-97). It's a bit of a waste of memory, especially if we're going to always use the same source (which is yours), but for now there's no other way to make a bundle by omitting a certain part of the charset. Nor will it involve much wasted space.

Once we have generated our binary and our decorations file (if applicable), we will have to compress the binary and put it in / bin for the librarian and move the decorations file to / script for later inclusion in the main script.

As the buildlevel commands are very long and you write bribes, and as we will have to generate the levels a lot of times during development, we will create a buildlevels.bat at / levels with calls to this command for each level.

```
@echo off
echo BUILDING LEVELS!

echo LEVEL 0
..\utils\buildlevel.exe ..\map\map0.map 4 4 99 ..\gfx\font.png ..\gfx\work0.png ..\gfx\sprites0.png ..\enems\enems0.ene 0
2 7 99 1 behs0.txt level0.bin decorations0.spt
..\utils\apack.exe level0.bin ..\bin\level0c.bin
move decorations0.spt ..\script

... (etc)
```

As you can see, by compressing it generated directly levelN.bin \ bin with the name of levelNc.bin

### Changes to make.bat

First we must remove whatever is build the map, export enemies, graphics, etc. and include a call To our new buildlevels.bat that should look something like this:

```
cd ..\levels
call buildlevels.bat
cd ..\dev
```

We call so that when finished buildlevels.bat follow the execution of make.bat.

### Librarian

Supposedly we would have to have the basics for the librarian. Remember that if you have pregenerated binaries for a specific RAM page (such as texts and script) you should place them in a preloadN.bin file where N is the RAM page where they should go so that librarian takes them into account. To the list of binary files /bin/list.txt we add our phases:

```
title.bin
marco.bin
ending.bin
level0c.bin
...
```

According to this configuration, the phase Nth resource will be in the N + 2-th: the first phase in the resource 3, the second at 4...

## The script

This tutorial deserve if you own, but I will mention here: your Script file will contain scripts for all levels. These scripts are separated into the file with END\_OF\_LEVEL:

```
### level 0

ENTERING GAME
  IF TRUE
  THEN
    ...
  END
  ...
END

...

END_OF_LEVEL

### level 1

ENTERING GAME
  IF TRUE
  THEN
    ...
  END
  ...
END

...

END_OF_LEVEL

...
```

Now you have to tell the engine where to find the script for each level. Msc3.exe, in the file msc-config.h that generates, creates a few constants that will suit us very well:

```
#define SCRIPT_0 0x0000
#define SCRIPT_1 0x0179
#define SCRIPT_2 0x01CD
...
```

In addition, this make.bat line:

```
..\utils\sizeof.exe ..\bin\texts.bin 49152 "#define SCRIPT_INIT" >> msc-config.h
```



Add another SCRIPT\_INIT constant that tells us where scripts start in memory. So, we will have the first script start at SCRIPT\_INIT + SCRIPT\_0, and so on. Well, these are the values that we have to put in the array of levels that appears at the end of levels128.h:

```
LEVEL levels [ ] = {  
    {3, 3, SCRIPT_INIT + SCRIPT_0},  
    {4, 4, SCRIPT_INIT + SCRIPT_1},  
    ...  
};
```

- The first number is the number of resource librarian that contains the bundle of the level that is. As you can see, we start at 3.
- The second number is the number of music within the OGT to sound of fondow.
- Finally, we have the expression that calculates where the concrete script to start for each phase begins.

With this and a sponge cake, the theme should work half what. And if not, as it is complicated the first time (and the second), there is always the forum to go slowly and step by step.

# Spectrum games using the MK2 engine

## Tutorial 4

### New enemies module

This tutorial applies to version 0.89 (or later) MK2  
In previous versions, the arrows are something "broken"

In Leovigildo completely rewrote the module enemies to make it better. Now you can choose which doll and what behavior each enemy has in multiple combinations (almost always), and define whether an enemy fires or not.

To place the "new" enemies you have to use the MK2 Positioner that is in enems. Go ahead with this, the old setter is still there. For the next version I think to smoke all the support to the traditional enemies, so their thing is to get accustomed.

When you give an enemy in MK2 Putter comes out a dialog with three things to fill: SPR, TYPE and SHOOTs.

#### 1. Linear:

SPR: Number of sprite of our spriteset. 0, 1, 2 or 3.

TYPE: 1

SHOOTs: 0 or empty: not fire. 1: yes it shoots.

The bug will go from the source box to the destination box as usual. The speed gets in by selecting the destination box, as always. Must be 1, 2, 4.

#### 2. Fantys:

SPR: Number of sprite. 0, 1, 2 or 3. Yes, you can now choose.

TYPE: 2

SHOOTs: the same.

It does not matter where we put the destination box and what speed we say. It is ignored.

#### 3. Drops:

SPR: It is ignored. Leave it empty.

TYPE: 9

SHOOTs: is ignored.

The drop goes from the source box to the destination box. You have to put it vertically. Speed also affects.

#### 4. Arrows:

SPR: It is ignored. Leave it empty.

TYPE 10

SHOOTs: If you put 1 the arrow will "trap" (only activated if the player touches any of the boxes where the path is defined). If you put 0, it will always leave.

The arrow goes from the source box to the destination box. You have to put it horizontally. It also affects speed.

The MK2 is very green. Edit an enemy already put is a pain, because the parameters SPR, TYPE and SHOOTs appear combined. I'll get better. It also does not take long to remove the bug and create it again. If you know binary and maybe you can understand those numbers yep some code dev / engine / enems.h

Oh, and you can not forget to remove the `#define USE_OLD_ENEMS` config.h, my god.

## Version Differences

### Version 3.99.2

Come on, the churreras are going out like hotcakes. We're breaking it, and we can think of new things every day. We'll get them in as we can think of games that take them.

These are the new things that are in this version of the churrera:

#### Timers

Added to the churrera a timer that we can use automatically or from the script. The timer takes an initial value, counts toward Down, can be recharged, can be set every how many frames is decremented or decide what to do when it runs out.

```
#define TIMER_ENABLE
```

TIMER\_ENABLE includes the code required to operate the timer. This code will need some other directives that specify the form of function:

```
#define TIMER_INITIAL 99  
#define TIMER_REFILL 25  
#define TIMER_LAPSE 32
```

TIMER\_INITIAL specifies the initial value of the timer. The time, which are set with the setter as type 5 hotspots, will recharge the value specified in TIMER\_REFILL. The maximum value of the timer, both for the as recharging, is 99. To control the amount of time elapses between each decrement of the timer, we specify in TIMER\_LAPSE the number of frames that must pass.

```
#define TIMER_START
```

If TIMER\_START is set, the timer will be active from the beginning.

We also have some directives that define what will happen when the timing to zero. It is necessary to uncomment those that apply:

```
#define  
TIMER_SCRIPT_0
```

Defining this, when it reaches zero the timer will execute a section script special, ON\_TIMER\_OFF. It is ideal for carrying all the control of the timer by scripting, as it happens in Cadàverion.

```
// #define TIMER_GAMEOVER_0
```

Defining this, the game will end when the timer reaches zero.

```
// # define TIMER_KILL_0  
// # define TIMER_WARP_TO 0  
// # define TIMER_WARP_TO_X 1  
// # define TIMER_WARP_TO_Y 1
```

If `TIMER_KILL_0` is set, a life will be subtracted when the timer reaches zero. If, in addition, `TIMER_WARP_TO` is defined, it will also be changed to the screen the player appears in the coordinates `TIMER_WARP_TO_X` and `TIMER_WARP_TO_Y`.

```
// # define TIMER_AUTO_RESET
```

If this option is set, the timer will return to maximum after reaching zero automatically. If you are going to perform the control by scripting, better leave it commented

```
#define SHOW_TIMER_OVER
```

If this is defined, in the case that we have defined either `TIMER_SCRIPT_0` or well `TIMER_KILL_0`, a "TIME'S UP!" Poster will be displayed. When the timer reaches zero.

### Scripting:

As we have said, the timer can be administered from the script. Is interesting that, if we decided to do this, let's activate `TIMER_SCRIPT_0` so that when the timer reaches zero, the `ON_TIMER_OFF` section of our script and that control is total.

In addition, these checks and commands are defined:

### Checks:

```
IF TIMER >= x  
IF TIMER <= x
```

Which will be fulfilled if the value of the timer is greater than or equal to or less or equal than the specified value, respectively.

### Commands:

```
SET_TIMER a, b
```

It is used to set the `TIMER_INITIAL` and `TIMER_LAPSE` values from the script.

```
TIMER_START
```

It is used to start the timer.

```
TIMER_STOP
```

It is used to stop the timer.

## Control of push blocks

We have improved the engine so that more can be done with the tile 14 of type 10 (pushable tile) that simply push it or stop the trajectory of the enemies. Now we can tell the engine to launch the PRESS\_FIRE section of the current screen just after pushing a pushable block. Besides, the number of the tile that is "stepped" and the final coordinates are stored in three Flags that we can configure, to be able to use them from the script to do checks.

This is the system that is used in the script of Cadàveriön to control that put the statues on the pedestals, to give an example.

Recall what we had so far:

```
#define PLAYER_PUSH_BOXES  
#define FIRE_TO_PUSH
```

The first one is necessary to activate the pushable tiles. The second obliges the player to press FIRE to push and therefore is not mandatory. Let's see now the new directives:

```
#define ENABLE_PUSHED_SCRIPTING  
#define MOVED_TILE_FLAG 1  
#define MOVED_X_FLAG 2  
#define MOVED_Y_FLAG 3
```

Enabling ENABLE\_PUSHED\_SCRIPTING, the tile to be pressed and its coordinates will store in the flags specified by the MOVED\_TILE\_FLAG, MOVED\_X\_FLAG and MOVED\_Y\_FLAG. In the code shown, the tile is will store in flag 1, and its coordinates in flags 2 and 3.

```
#define PUSHING_ACTION
```

If we define this, in addition, the PRESS\_FIRE AT ANY and PRESS\_FIRE of the current screen.

We recommend to study the Cadàveriön script, which, besides being a good example of the use of the timer and the pushbutton control, results be a rather complex script that employs a lot of advanced techniques.

## Check if we get off the map

It is advisable to put limits on your map so that the player can not Exit, but if your map is narrow you may want to take advantage of the screen. In that case, you can activate:

```
#define PLAYER_CHECK_MAP_BOUNDARIES
```

It will add checks and will not let the player leave the map. eye! If you can avoid using it, the better: you will save space.

## Type of enemy "custom" gift

Until now we had left the type 6 enemies without code, but we have thought that it is not difficult for us to put one, for example. It behaves like the bats of Cheril the Goddess. To use them, place them in the of enemies as type 6 and uses these directives:

```
#define ENABLE_CUSTOM_TYPE_6  
#define TYPE_6_FIXED_SPRITE 2  
#define SIGHT_DISTANCE 96
```

The first one activates them, the second defines which sprite to use (minus 1, if you want the sprite of enemy 3, put a 2. Sorry for the slut, but saving bytes). The third one says how many pixels you see from far away. If he sees you, he follows you. If not, return to your site (where you've put it with the setter).

This implementation, in addition, uses two directives of the enemies of type 5 to operate:

```
#define FANTY_MAX_V 256  
#define FANTY_A 12
```

Define there the acceleration and the maximum speed of your type 6. If you go to also use type 5 and you want other values, be a man and modify the engine.

## Keyboard / joystick configuration for two buttons

There are side view games that are best played with two buttons. If you activate this directive:

```
#define USE_TWO_BUTTONS
```

The keyboard will be the following, instead of the usual one:

- A = Left
- D = Right
- W = Up
- S = Down
- N = jump
- M = shot

If joystick is selected, FIRE and M fire, and N skips.

## Shooting up and diagonally for side view

Now you can let the player shoot up or diagonally. To do this define this:

```
#define CAN_FIRE_UP
```

This configuration works best with USE\_TWO\_BUTTONS, since this separates "Top" of the jump button.

If you do not hit "up", the character will fire to where he is looking. Yes Press "up" while shooting, the character will shoot up. Yes, In addition, you are pressing an address, the character will shoot on the diagonal indicated.

### **Masked bullets**

For speed, the bullets do not wear masks. This works fine if the background on which they move is dark (few active INK pixels). But nevertheless, there are situations where this does not happen and looks bad. In that case, we can activate masks for bullets:

```
#define MASKED_BULLETS
```

### **Version 3.99.2mod**

This was a special version with a thing that Radastan asked us, the...

### **Animated Tiles**

Everything is based on tilanim.h. This file is included if config.h is defined in ENABLE\_TILANIMS directive. In addition, the value of this directive is what defines The number of smaller tile that is considered animated.

In tilanim.h there are, in addition to the definition of data, two functions:

Void add\_tilanim (unsigned char x, unsigned char y, unsigned char t) is called from the function that paints the current screen if it detects that the tile that you are going to paint is >= ENABLE\_TILANIMS. Add an animated tile to the list tiles.

Void do\_tilanim (void) is called from the main loop. Basically select a random animated tile among all the stored ones, change the Frame (from 0 to 1, from 1 to 0) and draws it.

To use it, you just have to define the ENABLE\_TILANIMS directive in config.h with the smaller animated tile. For example, if your last four tiles (8 in total) are animated, put the value 40. Then, on the map, you have to put the smaller tile of the pair, that is, tile 40 for 40-41, the 42 to 42-43... If you do not do that, funny things will happen. The code is (It has to be) minimal, do not check anything, so take care.

By the way, this has not been proven. If you put it in your game and peta, damages one touch.



## Version 3.99.3

### Animated Tiles

If you define:

```
#define ENABLE_TILANIMS 32 // If defined, animated tiles are enabled.  
// the value specifies first animated tile pair.
```

In config.h, tiles >= that specified index are considered animated. In the tileset, they come in pairs. If, for example, "46" is defined, then the only pair of tiles 46 and 47 will be animated. The engine will detect them and each frame will cause one of the tiles 46 to change state.

There can be up to 64 animated tiles on the same screen. If you put more, it will not work.

### 128K Mode

You have to do a lot of manual work with this. I'm sorry, but it's like that. First you will have to create a make.bat that will build everything you need. For that you can rely on the file spare / make128.bat and adapt it to your project.

The 128K mode is the same as the 48K but use WYZ Player and also supports several levels. You can not have longer levels, but you can have several levels.

To use it, you need to activate three things in config.h:

```
#define MODE_128K // Experimental!  
#define COMPRESSED_LEVELS // use levels.h instead of map.h and enems.h (!)  
#define MAX_LEVELS 4 // # of compressed levels
```

In MAX\_LEVELS you have to specify the number of levels you are going to use.

In churromain.c you have to change the position of the pile and place it below of the main binary:

```
#pragma output STACKPTR = 24299
```

Then you have to modify levels128.h, which is where the level structure is defined and is included in 128K mode. There you will see an array levels, with information about the Levels. In principle, very little information is included:

```
// Level struct  
LEVEL levels [MAX_LEVELS] = {  
3,2,  
(4.3),  
{5.4},  
{6.5}  
};
```

The first value is the resource number (see below) that contains the level. The second value is the song number in WYZ PLAYER that should ring while it is played at level.

To prepare a level, you have to use the new buildlevel.exe utility in / utils. This utility takes the following parameters:

```
$ Buildlevel map.map map_w map_h lock font.png work.png spriteset.png  
Extrasprites.bin enems.ene scr_ini x_ini y_ini max_objs enems_life behs.txt level.bin
```

- Map.map Is mappy mappy
- Map\_w, map\_h Are the dimensions of the map on screens.
- Lock 15 for autodetect locks, 99 if no locks
- Font.png is a 256x16 file with 64 ascii characters 32-95
- Work.png is a 256x48 file with the tileset
- Spriteset.png is a 256x32 file with the spriteset
- Extrasprites.bin you find it in / levels
- Enems.ene the file with the enemies / hotspots of colocador.exe
- Scr\_ini, scr\_x, scr\_y, max\_objs, enems\_life level values
- Behs.txt a file with tile types, separated by commas
- Level.bin is the output file name.

When we have all levels built, we have to compress them with apack:

```
$ /utils/apack.exe level1.bin level1c.bin  
...
```

When we have all levels compressed, we will have to create the images binary files that will be loaded into the extra RAM pages. For that we use the utility Librarian that is in the folder / bin. In fact, it is a good idea to work in Folder / bin for this.

The librarian utility uses a list list with the compressed binaries that should be getting into the binary images that will go on the extra pages of RAM. The first thing we will have to put in is the title.bin, marco.bin and ending.bin, in that order. If you do not have .bin you should use a Length 0, but you must specify it. Then we will add our levels. By example:

```
Title.bin  
Marco.bin  
Ending.bin  
Level1c.bin  
Level2c.bin  
Level3c.bin  
Level4c.bin
```

There we added four compressed levels.

When you run librarian, you will be filling in 16K images destined to go in the extra RAM. First it will create ram3.bin, then ram4.bin and finally ram6.bin, according to I need more space.

It will also generate the file librarian.h, which we will have to copy to / dev. Here we can see the resource number associated with each binary:

```

RESOURCE resources [] = {
    {3, 49152}, // 0: title.bin
    {3, 50680}, // 1: marco.bin
    {3, 50680}, // 2: ending.bin
    {3, 52449}, // 3: level1c.bin
    {3, 55469}, // 4: level2c.bin
    {3, 58148}, // 5: level3c.bin
    {3, 60842} // 6: level4c.bin
};

```

These resource numbers are the ones we will have to specify in the array levels mentioned above. In particular, resources 3, 4, 5 and 6 are those containing the four levels.

With all this done and prepared, we will have to mount the tape. For this there are which create a suitable loader.bas (you can see an example in /spare/loader.bas) and build a .tap with each block of RAM (again, the example in /spare/make.bat builds the tape with binaries in RAM3 and RAM4).

You will also need RAM1.BIN to build RAM1.TAP, containing the player of WYZ with songs. For this you have to modify /mus/WYZproPlay47aZX.ASM in / mus to include your songs. You have an example in / spare.

As you can see, it's a bit tedious. I recommend that you build mini-projects in 48K as you make the levels, and finally you build a 128K version with all.

In addition, you can use extra space to push more compressed screens, or even code to use passwords to jump directly to levels. You can see examples of all this in Goku Mal 128.

### Type 3 Hotspots

We have made this modification, proposed in the forum, fixed to blow of directive. If you define

```
#define USE_HOTSPOTS_TYPE_3 // Alternate logic for recharges.
```

The recharges will appear only and exclusively where you place them, using the type 3 hotspot.

### Pause / Abort

If it is defined

```
#define PAUSE_ABORT // Add h = PAUSE, y = ABORT
```

Code is added to enable the "h" key to pause the game and the key "And" to interrupt the game. If you want to change the assignment you will have to touch the code in mainloop.h

## Message catching objects

If it is defined

```
#define GET_X_MORE // Shows "get X more"
```

A message will appear with the items you have left each time you take one.

### 3.99.3b

Minimal revision. It is arranged to be able to have 128K games with Only 1 level (ie use MODE\_128K without COMPRESSED\_LEVELS).

Right now there are two examples that can help you if you want to make a 128K game:

- Goku Mal: 128K with compressed levels. See this doc and the sources of the game.
- The new adventures of Dogmole Tuppowisky: 128K with only one level, more info in The forum of mojonía.

Also, in spare I added the file extern-texts.h whose contents you can use In extern.h if you want an easy way to display text on the screen using the EXTERN command n of the script.

### 3.99.3c

```
#define PLAYER_CAN_FIRE_FLAG 1
```

If set, the indicated flag controls whether the player can (1) or not (0) fire.

## Items Engine

We want to make sure there is a small inventory on screen and can Select an object from it, and we also want the objects that make up The inventory are not fixed and we can know, from the script, which object Is selected.

- In an initial section of the script, we will define "the itemset" (we must Put names to things, even if they are names as dull as this): how many Spaces have, where they are placed, and how objects are distributed. Something like that:

```
Code:
ITEMSET
# Number of holes:
SIZE 3

# Position x, y
LOCATION 2, 21

# Horizontal / vertical, spaced
DISPOSITION HORZ, 3

# Color and characters to paint the selector
SELECTOR 66, 82, 83

# (If defined) which tile represents the empty tile
EMPTY 31

# Flag containing which gap is selected
SLOT_FLAG 14

# Flag containing what object is in the selected hole
ITEM_FLAG 15
END
```

- An object is represented by its tile. If we have a crown on tile 10, The crown object will be 10. If in an inventory hole is 10, It means that in that hollow is the crown. A value of 0 will always represent An empty space. This simplifies the code an awful lot.

- There will be changes in the script. ITEM n = t means that in slot "n" is The object represented by tile t. We therefore define the following terms:

```
Code:
IF ITEM n = t
IF ITEM n <> t
```

They verify that in the space "n" is or not the object of tile "t".

```
IF SEL_ITEM = t
```

Check that in the space selected by the selector is the object of Tile "t"

And the following commands:

```
SET ITEM n = t
```

Sets tile n in tile t. Obviously, to remove an object from the gap N, we will set a 0.

There is a limitation, therefore, in the number of objects that the Character at a time. With a little head, as I said, it can be managed This very well, and with a minimum of code added to the engine we have a Quite powerful tool. All this has to be combined with the flags for Have full functionality. With ITEM's, we can only know if we have Not an ITEM in the inventory, but not if it has already been used. For that we need The flags.

How is this used? Let us give an example.

Imagine that on screen 6 we have a "crown" object, represented by the Tile 33, and we have it in (7, 7). In addition, the flag indicating its status is 3, Which will be worth 0 when we have not yet caught it or anything, to paint it on the screen.

```
Code:
ENTERING SCREEN 6
IF FLAG 3 = 0
THEN
SET TILE (7, 7) = 33
END
END
```

We will manage to take it. We can do it in basic mode or mode Virguero. Let's look at the basic mode first. In the basic mode we assign "by hand" A fixed gap for each item. The crown will be placed in the hole 2:

```
PRESS_FIRE AT SCREEN 6
IF PLAYER_TOUCHES (7, 7)
IF FLAG 3 = 0
THEN
SET FLAG 3 = 1
SET TILE (7, 7) = 0
SET ITEM 2 = 33
END
END
```

The game with flag 3 is simply so it does not re-draw. When the Flag 3 valga 1 will not re-paint the object when re-entering the Screen, or try to pick it up again. Otherwise, what is done is Make the object 33 in the gap 2.

Imagine that in screen 12 we have to use it in coordinate 5, 8. Well You have to check that the selected item is 33:

```
PRESS_FIRE AT SCREEN 12
IF SEL_ITEM = 33
THEN
SET ITEM 2 = 0
# more things
END
END
```

If the selected object is 33 (which can only occur if We put it in the slot 2), we remove it from the inventory (putting a 0 in the slot 2) and then we do more things.

The virgin mode is for the object to go to the selected hole. For that we use The indirection allowed by the scripting engine with the # operator. Remember That we are using the flag 10 to represent the selected hole. Let's play with that. Also, check that the gap is free!

```
PRESS_FIRE AT SCREEN 6
IF PLAYER_TOUCHES (7, 7)
IF FLAG 3 = 0
IF FLAG 10 <> 0
THEN
# Evil! The gap is not free!
SOUND 2
END

IF PLAYER_TOUCHES (7, 7)
IF FLAG 3 = 0
IF FLAG 10 = 0
THEN
SET FLAG 3 = 1
SET TILE (7, 7) = 0
SET ITEM # 10 = 33
END
END
```

What do we do? Then place the object of tile 33 (our crown) in space Selected, which is nothing more than the one stored in flag 10 (remember Which # 10 means "the value of flag 10").

To prove that we have it, then the same thing.

What do you think? Doubts? Something to comment? If it does, I'll do exactly as I have described.

## **MT Engine MK2 v 0.8**

We rewrote the almost whole engine. The enemy module is still missing We want to reorganize. That's why we're not in 1.0 yet.

The new engine works almost the same as the Churrera 3.X, but it works Faster and takes up less memory. There are also a lot of new things, like The "hitters" of Ninjajar (for now to give hosts, but soon to do Swords), better multi-phase support in scripting, able to jump from one phase To another, improved item engine...

## **Floating Objects**

```
#define ENABLE_FLOATING_OBJECTS
```

They are interactive tiles that are not part of the map. They are placed from the Script. For now the engine can handle two types.

The floating objects are placed on each screen from the script:

```
ADD_FLOATING_OBJECT t, x, y
```

## Carriable boxes

```
#define ENABLE_FO_CARRIABLE_BOXES  
#define CARRIABLE_BOXES_ALTER_JUMP 180  
#define FT_CARRIABLE_BOXES 16
```

They are boxes that can be transported. You need us to reserve 10 blocks of More sprites (see main.c, at first) for an extra sprite. The boxes are taken And deposit by pressing DOWN. Boxes are affected by gravity and are Stacking each other.

You can define the tile they use with the FT\_CARRIABLE\_BOXES directive. The Objects use the behavior defined for this tile. It will be the value That you have to give "t" at the time of placing them from the script, for example To place one of these boxes in position 7, 8 having defined that its Tile is the 16 we do:

```
ADD_FLOATING_OBJECT 16, 7, 8
```

If you define CARRIABLE\_BOXES\_ALTER\_JUMP, the maximum value of the speed of Jump will change to the specified value when we carry a box.

## Item containers

```
#define ENABLE_FO_OBJECT_CONTAINERS  
#define FT_FLAG_SLOT 30
```

These are intended to be used with scripting inventory engine. Each container actually represents a flag of the scripting system. To the Paint the screen will paint the tile whose number is stored in the flag corresponding.

To create them from the script:

```
ADD_FLOATING_OBJECT 128 + f, x, y
```

Where f is the flag we want to represent. For example, if we are going to use the Flag 10 to represent a container in position 4, 4 of the screen, We should create it like this:

```
ADD_FLOATING_OBJECT 138, 4, 4
```

As this is a bit confusing, we have added an alias. The same can be done calling to:

```
ADD_CONTAINER f, x, y
```



The previous example would be:

```
ADD_CONTAINER 10, 4, 4
```

The engine reacts to these blocks by exchanging the selected object of the Inventory with the one in the container.

### Alias in script

Because doing Ninjajar we wanted to go crazy with so much flag, we have Added aliases. We define a block at the beginning of the script like this:

```
DEFALIAS
    $ ALIAS N
    ...
END
```

From then on, we can substitute "N" for "\$ ALIAS". For example if We use flag 2 to open a green door and flag 3 to see if we have Spoken with the ogre we do:

```
DEFALIAS
    $ PUERTA_VERDE 2
    $ HABLA_OGRO 3
END
```

In the script we can use the alias instead of the numeric:

```
...
IF FLAG $ HABLA_OGRO = 0
THEN
EXTERN 10
SET FLAG $ HABLA_OGRO = 1
END
...
```

### MT Engine MK2 v 0.85

Changes and additions for the second charge of Leovigildo. They are a lot to see If I remember:

#### Throwing Carriable Boxes

You can launch the CARRIABLE\_BOXES boxes by pressing FIRE. Boxes kill The bugs and count them in a flag:

```
#define CARRIABLE_BOXES_THROWABLE    // If defined, carriable boxes are throwable!
#define CARRIABLE_BOXES_COUNT_KILLS 2 // If defined, count # of kills and store in flag 2.
```

## Single-display mode.

You can only change the screen by without scripting. Detects the change of screens of a lifetime when the child sticks to the edge. This is for making screen games on screen.

```
#define PLAYER_CANNOT_FLICK_SCREEN // If defined, automatic screen flicking is disabled.
```

## Counting Enemies

Count how many enemies are on the screen and put them in a flag. To do Games of "kill them all to pass" or things like that, of the stick "si You kill all the bugs something happens ".

```
#define COUNT_SCR_ENEMS_ON_FLAG 1 // If defined, count # of enemys on screen and store in flag #
```

## Show level

Each time you change the screen, display the screen number +1. In plan number of level. For screen-to-screen games.

```
#define SHOW_LEVEL_ON_SCREEN // If defined, show level # whenever we enter a new screen
```

## Enemies Module

Let's change the enemy module, so let's get ready.

## Old Style Enemies

As the old module I will not erase it, you can continue using it if you specify

```
#define USE_OLD_ENEMS // If defined, use old enems (like in Churrera)
```

## Disable Platform

Disable mobile platforms in platform games. Now you can have Four different enemies.

```
#define DISABLE_PLATFORMS // If defined, type 4 are enemies in side-view mode
```

## Resurrecting Enemies

Enemies resurrect as they enter the screen. Okay, this was it, but Now has some changes:

```
#define RESPAWN_ON_ENTER // Enemies respawn when entering screen  
#define RESPAWN_ON_REENTER // Respawn even on a REENTER in the script  
// (by default REENTER does not respawn enemies!)
```

If you activate the first one, when entering with the doll in a screen the enemies, they will come back to life.

If you activate, in addition, the second, the enemies will come back to life also after the REENTER command in the script.

We have improved a lot of things, including the timer, which was something broken (there will still be many things of the Churrera that are broken, let's go little To little).

## MT Engine MK2 v 0.86

Phantomas Engine Edition. Now you can make Phantomas games – this Opens the possibility of adding more easily movement engines Purely linear (without inertia).

```
// Phantomas Engine
//
// Comment everything here for normal engine
#define PHANTOMAS_ENGINE 1 // Which phantomas engine:
                          // 1 = Phantomas 1
                          // 2 = Phantomas 2
                          // 3 = LOKOsoft Phantomas
                          // 4 = Abu Simbel Profanation

#define PHANTOMAS_FALLING4 // Falling speed (pixels / frame)
#define PHANTOMAS_WALK 2 // Walking speed

#define PHANTOMAS_INCR_1 2 // Used for jumping
#define PHANTOMAS_INCR_2 4
#define PHANTOMAS_JUMP_CTR 16 // Total jumping steps up & down

// Most things from now on will not apply if PHANTOMAS_ENGINE is on ...
// Try ... And if you need something, just ask us ... Maybe it's possible to add.

// For example, BOUNDING_BOX_8_BOTTOM works for PHANTOMAS / PROFANANTION engines.
```

It's simple (or not). There are four engine types, as seen in the code. Then there are configuration parameters.

As it is, by selecting engine 1 the movement will be as in Phantomas 1, engine 2 will do as Phantomas 2, and engine 4 as Abu Simbel Profanation... But playing with values you will get others things.

Engines 1, 2 and 4 are based on two types of jumps. At engine 1 we have high jump (2 high tiles, 1 wide) and long jump (1 tile high, 4 wide). In engine 2 we have a long jump (slightly more than 2 tiles of high, 2 tiles of width) and short (1 tile x 1 tile), and in addition We can change the direction of the jump in the middle of the air. In the engine 4 we have jumps just like in engine 2, but we can not change the Direction and also if we press only jump the doll will jump to arrive (It is necessary to press jump + left or right to jump sideways).

In addition we have used to add support for enemies 100% custom To the enemy module, and we have included a pair as "addons": drops and Arrows, which use their own sprites. In addition, there is a new utility To convert sprites for these goings-on.

```
#define ENABLE_DROPS // Enemy type 9 = drops
#define ENABLE_ARROWS // Enemy type 10 = arrows
```

These enemies are placed with the setter. Look at the files

```
\ Dev \ addons \ drops \ move.h
\ Dev \ addons \ arros \ move.h
```

To see how your values are specified. Or the question in the forum.

The best thing is that if you want to use this we put a message or something asking The example micro-game where everything is seen in action. I do not think so Let's get it out of the way, I do not have the time or the energy.

## MT Engine MK2 v 0.87

Edition Leovigildo III. It has few purely new things, but it brings a bunch of internal improvements, bug fixes, optimizations...

### The new module of enemies.

To use it, make sure to comment #define USE\_OLD\_ENEMS.

Now enemies are much more flexible. Each defines several things:

- What sprite he uses, from 0 to 3.
- What kind of movement does it take: linear, flying..
- Whether he fires or not.

It is also prepared to make it very easy to get more behaviors.

Everything is specified in the enemy type, which is divided into several fields at bit level.

7	6	5	4	3	2	1	0
X	B	B	B	B	F	S	S

Where:

- X is reserved to mark if an enemy is crushed.
- BBBB is the type of movement. For now there are implemented these:

0001 (1) - round-trip linear, as always.

0010 (2) - flying. Like the fantys type 6.

0011 (3) - tracker. The bum type 7 always.

1000 (8) - mobile platform. As "1" but mobile platform.

If you use type 2, you have to enable `ENABLE_FLYING_ENEMIES`. Yes. Use type 3, enable `ENABLE_PURSUE_ENEMIES`.

- F is whether it triggers (1) or not (0). If you activate it for some enemy, remember that you have to enable `ENABLE_SHOOTERS` and set `MAX_COCOS` and other things.

- SS is the sprite number, according to spriteset, from 0 to 3 (00, 01, 10, 11).

The "type" of enemy is calculated, therefore, using this formula calculation:

$S + 4 * F + 8 * B$ , where S is the sprite, F is fired, and B is the behavior

To place them you can calculate the value of the type of enemy using The above formula (is binary) or use the MK2 Putter there In / enems, where you can set the values separately.

### **Modifications to floating objects**

- You can now better control your behavior with respect to the Gravity - if we want to use them in games of genital view, more than nothing:

```
#define FO_GRAVITY  
#define FO_SOLID_FLOOR
```

Activating the first, the FO will fall if there is no floor underneath. With the They will stop when they reach the bottom sign of the screen instead To disappear.

### **Floating object that hurt**

FOs "kill" while you carry them on their backs. This is for the nonsense Of this game, I do not know if it will serve for anything else... Anyway I do not It cost to put it in the config instead of a custom paranoia...

```
#define CARRIABLE_BOXES_DRAIN 7
```

### **Floating objects with Springs**

Couplings. FOs can be springs. If you fall on them, you will rebound. You can set the maximum bouncing speed.

```
#define CARRIABLE_BOXES_CORCHONETA  
#define CARRIABLE_BOXES_MAX_C_VY 1024
```

For this to work you have to give it the "bouncing" behavior To the tile that represents the corkboard. This means that in a Future version we can use bouncing tiles that are not FO, only Defining the behavior... I believe.

```
// 0 = Walkable (no action)
// 1 = Walkable and kills.
// 2 = Walkable and hides.
// 4 = Platform (only stops player if falling on it)
// 8 = Full obstacle (blocks player from all directions)
// 10 = special obstacle (pushing blocks OR locks!)
// 16 = Breakable (#ifdef BREAKABLE_WALLS)
// 32 = Conveyor
// 64 = CUSTOM FO -> SHEET!
```

## Scripting FO

This can be useful for many things but you have to use it carefully. HE Used in Leovigildo III to detect that we threw a The head to the tamer.

Basically, if activated, when a FO "falls", its type and position is stored In three flags (configurable) and PRESS\_FIRE is called in the script of that screen.

```
#define ENABLE_FO_SCRIPTING
#define FO_X_FLAG 1
#define FO_Y_FLAG 2
#define FO_T_FLAG 3
```

This is starting to be very scary. I was already scared in Ninजार. Now it's scary.

## MT Engine MK2 v 0.88

Edited by Leovigildo F. WTF? He's an EASTER EGG, but I've expanded The engine in various directions. Some may be useful to others Games, and others do not. Let's see if I remember everything:

### Scripting stuff without scripting

They are things that can be done with scripting, but being simple and You can store the pots in an array, if you were just going to use scripting For this, you save it and you can fit more. Namely:

```
- engine / levelnames.h
```

Allows you to name each screen. Names must have A fixed length and defined all in the same chain, all followed And in order. You can see it in levelnames.h itself, where you can also Set up location and color and such. To activate:

```
#define ENABLE_LEVEL_NAMES
```

```
- engine / extraprints.h
```

## Additional Tiles Impressions

Allows you to define extra tile impressions for each screen. There is a lot of people who have used scripting only for this (which seems to me a Shame, if you ask me). You do not need to activate the scripting, the Code takes up very little and each print only 2 bytes. To activate:

```
#define ENABLE_EXTRA_PRINTS
```

To set what will be printed, edit extraprints.h. There Defines an array for each screen with extra prints. Each extra print is It consists of 2 bytes: "xy" and "tile". "Xy" uses 4 bits for X and 4 for Y. It is Very easy to manage in hexadecimal, "x" goes from 0 to F and "y" from 0 to 9. The Byte "tile" is simply the tile number. The list ends with a 0xff (value 255).

For example, to print a tile 17 at the position X = 10, Y = 2, the two Bytes would be 0xA2, 17. To print a tile 33 at the position X = 5, Y = 7 The two bytes would be 0x57, 33.

## Simple Item Manager (SIM)

Finally, there is another array \* prints with an entry for each screen of the Map. If there are no extra prints on a screen, a 0 is set. If yes There are, the array of prints of the corresponding screen is set.

```
- engine / sim.h
```

SIM stands for "Simple Item Manager", and is used to manage items and Without requiring scripting for simple games in which there is X Objects in Y containers across the map, and the game is terminated when The X objects have been placed in other places. No more.

```
#define ENABLE_SIM
```

Activates the SIM. This will put redundant code with the system Of scripting, so both systems are NOT COMPATIBLE. If you use Scripting, manage your objects by hand.

The SIM has a few directives to configure it:

```
// General
#define SIM_MAXCONTAINERS 6
#define SIM_DOWN
// # define SIM_KEY_M
// # define SIM_KEY_FIRE
```

The first, SIM\_MAXCONTAINERS, defines the maximum number of containers (Other than objects) that will be in the game. A container may be Empty or contain an object. Games in which you have to put three Objects in three different sites, for example, will need six Containers: the 3 that will contain the objects, and 3 empty ones with the "final destination".

The next three, define which key is used to interact (pick / Leave object). Respectively, below, M or FIRE. Define only one, like With scripting.

A component of the SIM is inventory. The inventory is exactly the Same that you get when you use scripting and you define it in your script. HE Configured with the following directives:

```
// Display:
#define SIM_DISPLAY_HORIZONTAL
#define SIM_DISPLAY_MAXITEMS      2
#define SIM_DISPLAY_X              24
#define SIM_DISPLAY_Y              21
#define SIM_DISPLAY_ITEM_EMPTY    31
#define SIM_DISPLAY_ITEM_STEP     3
#define SIM_DISPLAY_SEL_C         66
#define SIM_DISPLAY_SEL_CHAR1     62
#define SIM_DISPLAY_SEL_CHAR2     63
```

If `SIM_DISPLAY_HORIZONTAL` is set, the inventory will be displayed in a horizontal line. If not defined, it will be displayed in a vertical line.

`SIM_DISPLAY_MAXITEMS` defines the number of slots in the inventory.

`SIM_DISPLAY_X` and `SIM_DISPLAY_Y` indicate the coordinate of the screen where The inventory is displayed. `SIM_DISPLAY_ITEM_STEP` defines how many Character cells will draw a new slot from the coordinates initials.

`SIM_DISPLAY_ITEM_EMPTY` specifies which tile represents the empty slot.

`SIM_DISPLAY_SEL_C` specifies the color of the selector, and `SIM_DISPLAY_SEL_CHAR1` And `SIM_DISPLAY_SEL_CHAR2` which two characters of your charset to use for To draw it.

Once all this is defined, we will have to open engine / sim.h to finish To configure our game.

In sim.h, two arrays are defined: `sim_initial` and `sim_final`. The first defines The location of the containers on the map and its initial contents; he Second defines a final state that will win the game if it is reached.

`Sim_initial` is an array of structures. An entry is defined for each Container of the game (in total, `SIM_MAXCONTAINERS` entries). Each entry Has a format `{n_pant, XY, tile}`, where `n_pant` is the screen where Find, `XY` are the coordinates (4 bits X, 4 bits Y, as in extraprints) And `tile` is the tile that represents the object contained in the container at Principle of the game.

For example `{10, 0x54, 32}` will cause the display 10 to have a container At the position `X = 5, Y = 4`, initially having the object 32.

`{3, 0xB3, 0}` will cause the screen 3, at the position `X = 11, Y = 3`, to have an empty container.

`Sim_final` is an array of numbers. It simply specifies which item should Have in each container to finish the game.

For example, imagine a silly game where there is an object in the screen 0 And another on screen 1, and you have to swap them to win. The objects They will both appear in `X = 7, Y = 4`, and will be represented with tiles 20 and 21.



In that case, SIM\_MAXCONTAINERS would be worth 2 and our arrays would be:

```
SIM_CONTAINER sim_initial [SIM_MAXCONTAINERS] = {  
    {0, 0x74, 20},  
    {1, 0x74, 21}  
};
```

```
Unsigned char sim_final [SIM_MAXCONTAINERS] = {21, 20};
```

As simple as this. At first, the containers contain the objects 20 and 21, and at the end must contain the objects 21 and 20.

**IMPORTANT NOTE: SIM needs to activate Floating Objects of type Container in config.h**

```
#define ENABLE_FO_OBJECT_CONTAINERS
```

In addition: if you have not tangled should be the same, but make sure everyone That the value of FT\_FLAG\_SLOT in config.hy of FLAG\_SLOT\_SELECTED in Yes they correspond. It looks like a sandbox, but it's like this for possible future Expansions.

## Improvements in JETPAC

The Jetpac occurred to us in the Churrera 1.0 and we program it, but without try. We did not use it until the Churrera 3.1, when we did Cheril the Goddess, and it was very rawro. Then we use it on Jet Paco.

From the beginning we had thought of giving chicha with refills, fuel That is over, and things like that, but even NOW has not been done.

```
#define PLAYER_HAS_JETPAC  
#define JETPAC_DEPLETES 4  
#define JETPAC_FUEL_INITIAL 25  
#define JETPAC_FUEL_MAX 25  
#define JETPAC_AUTO_REFILLS 2  
// # define JETPAC_REFILLS  
// # define JETPAC_FUEL_REFILL 25
```

PLAYER\_HAS\_JETPAC activates this system for a lifetime. If you only define This, you will have a jetpac as in jetpaco. Cool.

If you activate JETPAC\_DEPLETES with value "X", the jetpac will have fuel that is Will exhaust each X frames, with X a power of 2 (2, 4, 8, 16 ...). Having this active, we can define more behaviors.

JETPAC\_FUEL\_INITIAL and JETPAC\_FUEL\_MAX are required in any case if Active

JETPAC\_DEPLETES. Specify the fuel value at the beginning of the game And the maximum value that can be reached.

With this we have made the fuel to be spent. Now we must decide how Recover it

If you activate JETPAC\_AUTO\_REFILLS with value "Y", the jetpac will recharge only When not being used, each Y frames, with Y a power of 2.

If, instead, you activate JETPAC\_REFILLS, you will see refills placed Such as type 6 hotspots in the colander. Each recharge will recharge the number of Units specified in JETPAC\_FUEL\_REFILL.

## Whoa

Yes, that's what I say. For an easter egg. I have also fixed a few bugs That I've seen out there and cleaned up some things.

## MT Engine MK2 v 0.88c

**Warning:** As of this version the classic module of handling of enemies (#define USE\_OLD\_ENEMS) will no longer be supported. It will still be there, but it is likely that there will be many things that are obsolete. It should not be used.

**TODO:** Write an application that changes a format.ene file Old to the new format, in case you want to do some other rehash of An old game of the Churrera.

This release integrates some improvements in scripting, such as strings Of decorations (see below) or the sword of Sir Ababol as type of HITTER.

There is no game with this, but the Espadewr demo.

## Decorations

Basically it is to do the engine / extraprints.h but from the script. Until now we decorated the screens with extra tiles putting strings Of SET TILE (x, y) = t. That's a pain in writing and maintaining and Also occupied 4 bytes per tile.

Now we can define tiles of tiles in which the whole set of tiles Tiles will occupy  $2 * n + 2$ , where n is the number of tiles. A good improvement Compared to the original, which occupied  $4 * n$  bytes.

The theme is like this: simply enter this in the ENTERING\_SCREEN (or in Any site that supports commands: this serves to change good Bits of the screen from the script and can come great to implement Puzzles that modify the stage:

```
IF TRUE
THEN
    DECORATIONS
        X, y, t
        X, y, t
        ...
    END
END
```

Each line x, y, t defines the position and the number of an extra tile of decor. You can put all you want.

This serves to save a lot of memory and have well-decorated screens. Generally you do NOT need the 48 tiles of a tileset extended to the time. There is always a group of tiles that repeats more. The idea is Set that tileset as the main tileset (0-15) and place the tileset Others as decorations.

In addition, the new map2bin is responsible for detecting them automatically and Generate the necessary script lines.

## **REENTER, REDRAW, REHASH**

REDRAW has been redesigned to work better and faster. Redraw Again the screen from the buffer, and this includes everything we have Placed before with SET TILE (x, y) = t.

REHASH re-enters the screen to, among other things, initialize the enemies. It is necessary if an enemy is changed from linear type to type Flying, since you have to initialize some variables.

REENTER, as always: it's REDRAW + REHASH.

## **Change enemies and backup enemies**

In addition to turning them off and on (Ninजार!), We can now change the type Of the enemies. The type contains whether or not it fires, what pattern of movement Follow, and what sprite he has.

```
ENEMY n TYPE t
```

Sets type "t" for enemy "n".

However, it must be borne in mind that this is destructive: if we change the Type of an enemy, the original type will be lost forever.

In multi-level games this is no problem because with decompressing New level let's get ready.

For single-level games, we have introduced an "enemy backup" that We can activate from config.h using:

```
#define ENEMY_BACKUP
```

The backup takes up 3 bytes per screen and saves the original type of all Enemies.

From scripting, we can do:

**ENEMIES RESTORE**

It restores to its original values the enemies of the current screen.

**ENEMIES RESTORE ALL**

Restore the type of ALL level enemies.

If you only need to restore them at the start of each game, you can go from using ENEMIES RESTORE ALL in the script and activate the RESTORE\_ON\_INIT directive in Config.h

## Sword

We have added the sword of Sir Ababol 2 as a new type of hitter (interna-mind). For the moment, it is not possible to have a fist and a sword in it Game, although we will do something to make it possible in the future.

```
#define PLAYER_HAZ_SWORD
```

The screen graphic is defined where all others: in extrasprites.h

## More things

We have changed everything but now my memory fails... To see, things misce-Laneas that I remember:

```
#define PLAYER_WRAP_AROUND
```

Use it only if #define PLAYER\_CANNOT\_FLICK\_SCREEN is set. With Both active, in addition to not being able to leave the screen, if we approach a Lateral end we will leave on the contrary.

## MT Engine MK2 v 0.89 (Nicanor Edition)

This version corresponds to the game "Nicanor the Profaner". Includes one New version of msc3, 3.92

## Easier Scripting with Alias

As of version 3.92 of msc3, you can skip the word FLAG if you use alias. That is, the compiler will accept

```
SET $ KEY = 1
```

and also

```
IF $ KEY = 1
```

## Safe Spot

If you define #define DIE\_AND\_RESPAWN in config.h, the player, upon dying, goes to Reappear in the "last safe point". If DIE\_AND\_RESPAWN is active, the Engine saves the position (screen, x, y) each time we Tile nontransferable (other than a floating object).

We can control the definition of the "safe spot" from our Script. If we decide to do this (for example, to define a "checkpoint" Manually), it is convenient to deactivate that the engine stores the safe Spot automatically with:

```
#define DISABLE_AUTO_SAFE_SPOT
```

Whether or not we do this, we can define the safe spot from the script with these two commands:

SET SAFE HERE	Sets the "safe spot" to the current position of the player.
SET SAFE n, x, y	Sets the "safe spot" to screen n in the Coordinates (of tile) (x, y).