# RESUMEN TUTORIAL “LA CHURRERA” DE LOS MOJON TWINS

Copiamos la carpeta principal de la churrera y le cambiamos el nombre a la carpeta y a */dev/churromain.c* por el de nuestro proyecto. Editamos el archivo */dev/make.bat* cambiando %1 por el nombre que le pusimos a *churromain.c .*

**Tipos de tiles:**
0) traspasable
1) traspasable y matante
2) traspasable y oculto
4) plataforma
8) obstáculo
10) interactuable
16) destructible

**Plantilla de photoshop:**
256x48 px.
Rejilla de 16x16 px. con 2 subdivisiones.

**Tileset de 16 tiles:**
El tileset se divide en la sección de mapa (16 primeros tiles) y la sección especial (6 tiles siguientes).
El tile 0 se suele dejar de color de fondo.
Los tiles 1 al 13 pueden ser de cualquier tipo.
Solo el tile 14 puede ser de tipo interactuable empujable.
Solo el tile 15 puede ser de tipo interactuable cerrojo.
El tile 16 es la recarga de vida.
El tile 17 es el objeto.
El tile 18 es la llave.
El tile 19 es el fondo alternativo.
El tile 20 es la recarga de munición.
El tile 21 es la recarga de tiempo.
Se puede activar el sombreado automático, para eso debemos añadir una tira de tiles abajo del tileset normal, con los tiles sombreados.

**Tileset de 48 tiles:**
No hay tiles de sombras. Del 0 al 47 podemos usarlos para lo que queramos (del 15 al 18 pueden seguir siendo especiales). El fondo alternativo (19) no se aplica.

*Al terminar el tileset (16 O 48) lo grabaremos como *work.png* en la carpeta */gfx*. Copiar los 16 primeros tiles de *work.png* y grabarlos como *mappy.bmp* en */gfx*. El primer tile siempre debe ser negro entero.

**El Charset:**
Lo siguiente es crear un charset con 64 caracteres del 32 al 95 ASCII. grabarlo como *fuente.png* en */gfx*.

**El Tileset final:**
Usamos la utilidad */utils/reordenator.exe work.png udgs.png* que cortará los tiles en trozos de 8x8 y los grabara en el nuevo archivo *udgs.png*.
Luego creamos un nuevo archivo en photoshop de 256x64 px. y pegamos abajo el *udgs.png* y arriba el *font.png* y lo grabamos como *tileset.png* en */gfx*.

**SevenUp:**
Abrir e importar el archivo *tileset.png*. Configurar el *byte sort priority* con el *Char line* arriba en la lista. Exportar y guardar en */dev/tileset.h*. Luego manualmente cambiar la línea 279, que contiene el primer tile, y sustituir los 0 por 7.

**Mappy:**
para crear un mapa nuevo poner el tamaño de los tiles. 16x16 y el ancho y alto en tiles, cada pantalla tiene 15 tiles de ancho por 10 de alto. Luego debemos

importar nuestro *mappy.bmp*. Es recomendable activar la rejilla (dividers) a 240x160.
Grabar de vez en cuando como *FMP*. Al finalizar, grabarlo como *FMP*, y como *MAP* en la carpeta */MAP*.

Por último, ejecutar el comando *mapcnv* que está en */UTILS* con los parámetros:
- archivo MAP.
- ancho del mapa en pantallas.
- alto del mapa en pantallas.
- ancho de la pantalla en tiles (15)
- alto de la pantalla (10)
- tile_cerrojo (15, o 99 si no se usa)
- *packed*, si el tileset es de 16 tiles, de 48 no se pone.

Quedaría algo así como *mapcnv mapa.map 8 3 15 10 15 packed*. Obtenemos un archivo *mapa.h* que movemos a */DEV*.

**Sprites:**
Imagen que contiene 16 sprites de 16x16 píxeles, y sus correspondientes máscaras. (256x32) Usar el blanco sobre negro para los sprites, y el negro sobre rojo para las máscaras.

Spriteset de vista lateral:
- principal, derecha andando, frame1
- principal, derecha andando o parado, frame2
- principal, derecha andando, frame3
- principal, derecha saltando
- principal, izquierda andando, frame1
- principal, izquierda andando o parado, frame2
- principal, izquierda andando, frame3
- principal, izquierda saltando
- enemigo tipo1, frame1
- enemigo tipo1, frame2
- enemigo tipo2, frame1
- enemigo tipo2, frame2
- enemigo tipo3, frame1
- enemigo tipo3, frame2
- plataforma movil, frame1
- plataforma movil, frame2

Spriteset de vista cenital:
- principal, derecha andando, frame1
- principal, derecha andando, frame2
- principal, izquierda andando, frame1
- principal, izquierda andando, frame2
- principal, arriba andando, frame1
- principal, arriba andando, frame2
- principal, abajo andando, frame1
- principal, abajo andando, frame2
- enemigo tipo1, frame1
- enemigo tipo1, frame2
- enemigo tipo2, frame1
- enemigo tipo2, frame2
- enemigo tipo3, frame1
- enemigo tipo3, frame2
- enemigo tipo4, frame1
- enemigo tipo4, frame2

Guardar el archivo como *sprites.png* en */gfx*. Luego desde */gfx* ejecutamos...
*..\utils\sprncv.exe sprites.png ..\dev\sprites.h*

**Sprites extra:**
Hay tres sprites más en */dev/extrasprites.h*.
17) Explosión.

18) Uso interno. No cambiar.
19) Bala o proyectil.
Para cambiar el sprite de la explosión, tenemos que...
- Crearnos un tileset vacío de 256x32.
- Dibujar en la primera casilla la nueva explosión(16x16) y su máscara.
- Guardarlo como *extra.png* en */gfx*.
- Usar la utilidad *sprncv* para obtener el *extra.h*.
- Cortar las secciones a,b,c del sprite 1 del archivo *extra.h* y pegarlas en *\dev\extrasprite.h* sustituyendo el sprite 17. (cambiar luego el nº1 por el 17)

Para cambiar el sprite de la bala, tenemos que modificar la sección 19_a de *dev\extrasprite.h*, usando binario en vez decimal.
por ejemplo, en vez de...

```
defb  0, 0
defb  0, 0
defb 24, 0
defb 60, 0
defb 60, 0
defb 24, 0
defb  0, 0
defb  0, 0
```

podemos poner esto para el mismo gráfico. En binario vemos a simple vista el diseño. No se usa máscara.

```
defb @00000000, 0
defb @00000000, 0
defb @00011000, 0
defb @00111100, 0
defb @00111100, 0
defb @00011000, 0
defb @00000000, 0
defb @00000000, 0
```

**Pantallas fijas:**
En la churrera tenemos tres tipos de pantalla...

Pantalla de título:
Debe incluir los tipos de controles (1-Teclado, 2-Kempston, 3-Sinclair).
Guardar en */gfx* como *title.png*.

Pantalla de marco:
Puede ir en la de título, ahorrando memoria. Está dividida en varias zonas:
-Zona de juego. (240x160)
-Marcadores de vida, llaves, objetos, enemigos mataos. Se colocan en alguna parte del marco. Dos caracteres para cada marcador. Colocar en posición alineada a caracter. 0 a 31 para el eje X y 0 a 23 para el eje Y.
Grabar como *marco.png* en */gfx*.

Pantalla final:
Sin restricciones, ocupa toda la pantalla, y también es opcional.
Grabar como *ending.png* en */gfx*.

Luego con SevenUp importaremos estos *PNG* y los convertiremos en *SCR*, dentro de */gfx*. Por último comprimimos los archivos *SCR*, y los pasamos a */dev*. Para ello usamos los comandos...

```
..\utils\apack.exe title.scr title.bin
..\utils\apack.exe marco.scr marco.bin
..\utils\apack.exe ending.scr ending.bin
```

**Colocando enemigos:**
Los enemigos de tipo 1, 2 o 3 (y el 4 en vista cenital) describen trayectorias lineales y se dibujan con el primer, segundo o tercer (o cuarto) sprite de enemigos del tileset. Se definen el punto de inicio y el punto de final describiendo una linea recta vertical u horizontal.
El enemigo sigue esa línea imaginaria yendo y viniendo ad infinitum. Si el punto

de inicio y el del final no están en la misma fila o columna, el sprite se mueve dentro del rectángulo que describen ambos puntos, rebotando en sus paredes y moviéndose en diagonal.

Los enemigos de tipo 4, cuando la vista es lateral, son plataformas móviles. Se comportan exactamente igual que los enemigos lineales.

Los enemigos de tipo 5 se pintan con el gráfico del tercer enemigo del tileset y tienen este comportamiento especial: se crean fuera de pantalla, y si el protagonista no está escondido, lo persiguen. Si el protagonista se esconde, se alejan hasta volver a salir de la pantalla. Por defecto, el código para mover y gestionar los enemigos de tipo 5 no se incluye en el motor, sino que ha de ser activado de forma explícita.

Los enemigos de tipo 7 aparecen en el punto donde los creas y se dedican a perseguir al jugador. No pueden traspasar el escenario. Si el jugador puede disparar y los mata, volverán a aparecer al ratito en el mismo sitio de donde salieron por primera vez. El gráfico con el que se dibujan se elige al azar entre todos los disponibles. Funcionan mejor en vista cenital. Al igual que los enemigos de tipo 5, el código para gestionarlos no se incluye en el motor por defecto y hay que hacerlo explícitamente.

**Hotspots:**
Los hotspots son una posición dentro de la pantalla donde puede haber un objeto, una llave, o una recarga. En cada pantalla se define un único hotspot. Cada hotspot tiene asociado un valor.
Si se le da un valor "0" este hotspot estará desactivado y nunca aparecerá nada.
Si se le da un valor "1", aparecerá un objeto.
Si se le da un valor "2", aparecerá una llave.
Los hotspots con valores "1" o "2" se consideran "activos".
Una vez que hayamos cogido el objeto o la llave de un hotspot activo, el motor puede que haga aparecer una recarga de vida en ese lugar la próxima vez que entremos en esa pantalla. Cuando hablamos de objetos nos referimos al item que se dibuja con el tile número 17 del tileset.

**Colocador:**
Lo primero que tendremos que hacer es copiar el archivo *.MAP* y *.BMP* al directorio donde está el Colocador, que es el directorio */enems*. Aparecerá la pantalla principal en la que, o bien cargaremos un proyecto existente, o configuraremos uno nuevo. En las dos primeras casillas habrá que introducir el nombre de nuestro mapa y de nuestro tileset. Posteriormente hay cuatro casillas que definen el tamaño del mapa y de las pantallas. *MAP_W* y *MAP_H* deberían contener el ancho y alto de tu mapa medido en pantallas. *SCR_W* y *SCR_H* son para especificar el ancho y alto de las pantallas en tiles. (Para la Churrera estos valores son 15 y 10.) La última casilla que hay que rellenar es el número de enemigos que aparecerá en cada pantalla. (3) Cuando esté todo relleno pulsamos en *NUEVO* y entonces podremos empezar a trabajar.

Para navegar por el mapa (para que salga otra pantalla en la rejilla) usaremos las teclas de los cursores. Si pulsamos *ESC* se cerrará el programa. Pero cuidado: se sale sin más. Cuidado, cuidado, cuidado. Para grabar tenemos la tecla *S*. Por convención usamos *.ene* como extensión para los archivos del Colocador. Este archivo *.ene* no nos sirve para usarlo directamente. Para poder tener nuestros enemigos en el juego necesitaremos que el colocador exporte código C. Esto es hace pulsando la tecla *E*. Cuando lo hagamos, aparecerá un cuadro de diálogo parecido al del guardar. Ahí deberemos escribir *enems.h*, pulsar *OK*, y copiar este archivo a */dev*.

**Poniendo enemigos y plataformas:**
Lo más sencillo es colocar enemigos lineales (horizontales, verticales, o diagonales). Lo que se hace es definir una trayectoria, un tipo, y una velocidad. Lo primero que tenemos que hacer es colocar el ratón sobre la casilla de inicio de la trayectoria y hacer click. Esta posición será la inicial, donde aparecerá el enemigo, y además servirá como uno de los límites de su

trayectoria. Cuando hagamos esto, aparecerá un cuadro de diálogo donde deberemos introducir el tipo del enemigo. Recuerda que en el caso de enemigos lineales será un valor de 1 a 4, 4 para las plataformas en vista lateral. Ahora lo que el programa espera es que le digamos donde acabaría la trayectoria. Nos vamos a la casilla donde debe acabar la trayectoria y hacemos click de nuevo. Veremos como se nos muestra gráficamente la trayectoria y aparece un nuevo cuadro de diálogo en el que nos preguntan por la velocidad. El valor introducido será el número de píxels que avanzará el enemigo o plataforma por cada cuadro de juego. Estos valores son 1, 2 o 4. Los enemigos tipo 5 da igual donde los pongas: siempre aparecen desde fuera de la pantalla. La velocidad que pongas también es lo de menos. Con los enemigos de tipo 7 sólo se tendrá en cuenta la posición de inicio. La velocidad y la posición del final serán ignoradas igualmente. Para eliminar o editar los valores de un enemigo que ya hayamos colocado, basta con hacer click sobre la casilla de inicio de la trayectoria (donde aparece el numerito que indica el tipo).

**Colocando hotspots:**
Recordemos que cada pantalla admite un único hotspot. Eso significa que el número total de llaves y objetos necesarios para terminar el juego no puede exceder el número de pantallas. Para colocar el hotspot de la pantalla actual, simplemente hacemos click con el botón derecho en la casilla donde queremos que aparezca la llave o el objeto, con lo que haremos aparecer un pequeño cuadro de diálogo donde deberemos introducir el tipo.

**El archivo "config.h":**
Tamaño del mapa
```
#define MAP_W 8 //
#define MAP_H 3 // Map dimmensions in screens
```

Posición de inicio
```
#define SCR_INICIO 16 // Initial screen
#define PLAYER_INI_X 1 //
#define PLAYER_INI_Y 7 // Initial tile coordinates
```

Posición de fin
```
#define SCR_FIN 99 // Last screen. 99 = deactivated.
#define PLAYER_FIN_X 99 //
#define PLAYER_FIN_Y 99 // Player tile coordinates to finish game
```

Número de objetos
```
#define PLAYER_NUM_OBJETOS 99 // Objects to get to finish game
```

Este parámetro define el número de objetos que tenemos que reunir para terminar el juego. El conteo de objetos y la comprobación de que lo tenemos todos es automática y emplea este valor: en cuanto el jugador tenga ese número de objetos se mostrará la pantalla del final. Pondremos un valor fuera de rango, nuestro 99 querido, para que el motor ignore el conteo de objetos automático.

Vida inicial y valor de recarga
```
#define PLAYER_LIFE 15 // Max and starting life gauge.
#define PLAYER_REFILL 1 // Life recharge
```

Aquí definimos cuál será el valor inicial de vidas del personaje y cuánto se incrementará cuando cojamos una vida extra. En juegos en los que las colisiones producen rebotes y el enemigo nos puede golpear muchas veces, las "vidas" se consideran "energía" y se usan valores más altos, como 99, y recargas más generosas, de 10 o 25.

Juegos multi nivel
```
//#define COMPRESSED_LEVELS // use levels.h instead of mapa.h and enems.h
//#define MAX_LEVELS 2 // # of compressed levels
//#define REFILL_ME // If defined, refill player on each level
```

Tamaño de la caja de colisión

```
#define BOUNDING_BOX_8_BOTTOM // 8x8 aligned to bottom center in 16x16
//#define BOUNDING_BOX_8_CENTERED // 8x8 aligned to center in 16x16
//#define SMALL_COLLISION // 8x8 centered collision instead of 12x12
```

La segunda opción (recuadro centrado) está pensada para vista cenital. La primera funciona bien con vista lateral o vista cenital "con un poco de perspectiva". Si queremos colisión de 16x16 desactivamos ambas. La tercera directiva se refiere a las colisiones contra los enemigos. Si activamos SMALL_COLLISION, los sprites tendrán que tocarnos mucho más para darnos. Con SMALL_COLLISION los enemigos son más fáciles de esquivar. Funciona bien en juegos con movimientos rápidos.

Directivas generales

```
#define PLAYER_AUTO_CHANGE_SCREEN // Player changes screen automaticly
```

Si definimos esto, el jugador cambiará de pantalla cuando salga por el borde. Normalmente se deja activada.

```
//#define PLAYER_PUSH_BOXES // If defined, tile #14 is pushable
//#define FIRE_TO_PUSH
```

Estas dos directivas activan y configuran los bloques empujables. Activar la primera (PLAYER_PUSH_BOXES) activa los bloques, de forma que los tiles #14 (con comportamiento tipo 10, recuerda) podrán ser empujados. La segunda directiva, FIRE_TO_PUSH, es para definir si el jugador debe pulsar fire además de la dirección en la que empuja o no.

```
#define DIRECT_TO_PLAY // If defined, title screen = game frame.
```

Esta directiva, se activa para que el título del juego sirva también como marco. Si tienes un title.bin y un marco.bin separados, deberás desactivarla.

```
//#define DEACTIVATE_KEYS // If defined, keys are not present.
//#define DEACTIVATE_OBJECTS // If defined, objects are not present.
```

Estas dos directivas sirven para desactivar llaves u objetos. Si tu juego no usa llaves y cerrojos, deberás activar DEACTIVATE_KEYS. Si no vas a usar objetos, activamos DEACTIVATE_OBJECTS.

```
#define ONLY_ONE_OBJECT // If defined, only one object can be carried
#define OBJECT_COUNT 1 // Defines FLAG to be used to store object #
```

Si activamos la primera, ONLY_ONE_OBJECT, sólo podremos llevar un objeto. Una vez que cojas un objeto, la recogida de objetos se bloquea y no puedes coger más. Para volver a activar la recogida de objetos tendremos que usar scripting. La segunda directiva, OBJECT_COUNT, sirve para que en el marcador de objetos, en lugar de la cuenta interna de objetos recogidos, se muestre el valor de uno de los flags del sistema de scripting. Los scripts tienen hasta 32 variables o "flags" que podemos usar para almacenar valores y realizar comprobaciones. Cada variable tiene un número. Si definimos esta directiva, el motor mostrará el valor del flag indicado en el contador de objetos del marcador. Desde el script iremos incrementando dicho valor. Sólo necesitaremos definir OBJECT_COUNT si somos nosotros los que vamos a llevar la cuenta. Si no vamos a usar scripting, o no vamos a necesitar controlar a mano número de objetos recogidos, tendremos que comentar esta directiva para que no sea tomada en cuenta.

```
//#define DEACTIVATE_EVIL_TILE // If defined, no killing tiles are detected.
```

Descomenta esta directiva si quieres desactivar los tiles que te matan (tipo 1). Si no usas tiles de tipo 1 en tu juego, descomenta esta linea y ahorrarás espacio, ya que así la detección de tiles matantes no se incluirá en el código.

```
//#define PLAYER_BOUNCES // If defined, collisions make player bounce
//#define FULL_BOUNCE
//#define SLOW_DRAIN // Works with bounces. Drain is 4 times slower
#define PLAYER_FLICKERS // If defined, collisions make player flicker
```

Estas dos directivas controlan los rebotes. Si activas PLAYER_BOUNCES, el jugador rebotará contra los enemigos al tocarlos. La fuerza de dicho rebote se controla con FULL_BOUNCE. Si se activa, los rebotes serán mucho más bestias porque se empleará la velocidad con la que el jugador avanzaba originalmente, pero en sentido contrario. Desactivando FULL_BOUNCE el rebote es a la mitad de la velocidad. Si definimos, además, SLOW_DRAIN, la velocidad a la que perderemos energía si nos quedamos atrapados en la trayectoria del enemigo será cuatro veces menor. Esto hace el juego más asequible. FULL_BOUNCE y SLOW_DRAIN dependen de PLAYER_BOUNCES. Si PLAYER_BOUNCES está desactivada, las otras dos directivas son ignoradas. Activando PLAYER_FLICKERS logramos que el personaje parpadee si colisiona con un enemigo (siendo invulnerable durante un corto período de tiempo). Normalmente elegiremos entre PLAYER_BOUNCES y PLAYER_FLICKERS, pero pueden funcionar juntas.

```
//#define MAP_BOTTOM_KILLS // If defined, exiting the map bottom kills.
```

Desactiva esta directiva si quieres que, en el caso de que el personaje vaya a salirse del mapa por abajo, el motor le haga rebotar y le reste vida. Si tu mapa está cerrado por abajo, desactívala para ganar memoria.

```
//#define WALLS_STOP_ENEMIES // If defined... erm...
```

Esta directiva está relacionada con los tiles empujables. Si defines bloques empujables, puede interesarte que los enemigos reaccionen ante ellos y alteren sus trayectorias. Si no usas tiles empujables o te da igual que los enemigos los traspasen, desactívala para ganar un montón de espacio.

```
//#define ENABLE_PURSUERS // If defined, type 7 enemies are active
//#define DEATH_COUNT_EXPRESSION 8+(rand()&15)
```

Activando ENABLE_PURSUERS activamos los enemigos de tipo 7. Si tenemos activado el motor de disparos, cuando matemos a un enemigo de tipo 7 tardará un tiempo en volver a salir. Dicho tiempo, expresado en número de cuadros (frames), se calcula usando la expresión definida en DEATH_COUNT_EXPRESSION. La que se ve en el ejemplo es 8 más un número al azar entre 0 y 15 (o sea, entre 8 y 23 frames).

```
//#define ENABLE_RANDOM_RESPAWN // If defined, flying enemies
//#define FANTY_MAX_V 256 // Flying enemies max speed.
//#define FANTY_A 12 // Flying enemies acceleration.
//#define FANTIES_LIFE_GAUGE 10 // Amount of shots needed to kill.
```

Activando ENABLE_RANDOM_RESPAWN activamos los enemigos de tipo 5. Las siguientes directivas sirven para configurar su comportamiento. FANTY_MAX_V define la velocidad máxima. Para hacerte una idea, divide el valor entre 64 y el resultado es el número de píxels que avanzará por cada cuadro (frame) de juego como máximo. Si definimos 256, el enemigo volador podrá acelerar hasta los 4 píxels por frame. FANTY_A es el valor de aceleración. Cada cuadro de juego, la velocidad se incrementará en el valor indicado en dirección hacia el jugador, si no está escondido. Cuanto menor sea este valor, más tardará el enemigo en reaccionar a un cambio de dirección del jugador. FANTIES_LIFE_GAUGE define cuántos tiros deberá recibir el bicharraco para morirse, si es que hemos activado los tiros.

Motor de disparos

```
//#define PLAYER_CAN_FIRE // If defined, shooting engine is enabled.
```

Si activamos esta directiva, estamos incluyendo el motor de disparos. Las siguientes directivas sirven para configurar su comportamiento.

```
//#define PLAYER_BULLET_SPEED 8 // Pixels/frame.
//#define MAX_BULLETS 3 // Max number of bullets on screen.
```

La directiva PLAYER_BULLET_SPEED controla la velocidad de las balas. 8 píxels por cuadro es un buen valor. Un valor mayor puede hacer que se pierdan colisiones, ya que todo lo que ocurre en pantalla es discreto (no continuo) y si un enemigo se mueve rápidamente en dirección contraria a una bala que se mueve demasiado rápido, es posible que de frame a frame se crucen sin colisionar.
El valor MAX_BULLETS controla el número máximo de balas que podrá haber en pantalla. (3)

```
//#define PLAYER_BULLET_Y_OFFSET 6 // vertical offset from the player's top.
//#define PLAYER_BULLET_X_OFFSET 0 // horz offset from the player's left/right.
```

Estas dos directivas definen dónde aparecen las balas cuando se disparan. El comportamiento de estos valores cambia según la vista:
Si es en vista lateral, sólo podemos disparar a la izquierda o a la derecha. En ese caso, PLAYER_BULLET_Y_OFFSET define la altura, en píxels, a la que aparecerá la bala contando desde la parte superior del sprite del personaje. Esto sirve para ajustar de forma que salgan de la pistola o de donde queramos. PLAYER_BULLET_X_OFFSET se ignora completamente. Si es en vista cenital, el comportamiento es igual que el descrito si miramos para la izquierda o para la derecha, pero si miramos para arriba o para abajo la bala aparecerá desplazada lateralmente PLAYER_BULLET_X_OFFSET píxels desde la izquierda si miramos hacia abajo o desde la derecha si miramos hacia arriba.

```
//#define ENEMIES_LIFE_GAUGE 4 // Amount of shots needed to kill enemies.
//#define RESPAWN_ON_ENTER // Enemies respawn when entering screen
```

Estas de aquí sirven para controlar qué ocurre cuando las balas golpean a los enemigos normales (de tipo 1, 2 o 3). En primer lugar, ENEMIES_LIFE_GAUGE define el número de tiros que deberán llevarse para morirse. Si activamos RESPAWN_ON_ENTER, los enemigos habrán resucitado si salimos de la pantalla y volvemos a entrar. Los enemigos muertos se van contando y dicho valor puede controlarse desde el script.

Scripting

```
//#define ACTIVATE_SCRIPTING // Activates msc scripting and flags.
#define SCRIPTING_DOWN // Use DOWN as the action key.
//#define SCRIPTING_KEY_M // Use M as the action key instead.
```

Si activamos ACTIVATE_SCRIPTING, todo el sistema de scripting será incluido en el motor. Las otras dos definen cuál será la tecla de acción: abajo o fire. Sólo una de las dos podrá estar activada.

Directivas relacionadas con la vista cenital:

```
//#define PLAYER_MOGGY_STYLE // Enable top view.
//#define TOP_OVER_SIDE // UP/DOWN has priority over LEFT/RIGHT
```

Si activamos PLAYER_MOGGY_STYLE, el juego será de vista cenital. Si no está activada, será de vista lateral. La siguiente, TOP_OVER_SIDE, define el comportamiento de las diagonales. Esto tiene utilidad sobre todo si tu juego tiene, además, disparos. Si se define TOP_OVER_SIDE, al desplazarse en diagonal el sprite mirará hacia arriba o hacia abajo y, por tanto, disparará en dicha dirección. Si no se define, el sprite mirará y disparará hacia la izquierda o hacia la derecha. Depende del tipo de juego o de la configuración del mapa te interesará más una u otra opción. No se puede disparar en diagonal.

Directivas relacionadas con la vista lateral:

```
#define PLAYER_HAS_JUMP // If defined, player is able to jump.
```

Si se define esta, el jugador puede saltar. Si no se han activado los disparos, fire hará que el jugador salte. Si están activados, lo hará la tecla "arriba".

```
//#define PLAYER_HAS_JETPAC // If defined, player can thrust a jetpac
```

Si definimos PLAYER_HAS_JETPAC, la tecla "arriba" hará que activemos un jetpac. Es compatible con poder saltar. Sin embargo, si activas a la vez el salto y el jetpac no podrás usar los disparos.

```
#define PLAYER_KILLS_ENEMIES // If defined, stepping on enemies kills them
#define PLAYER_MIN_KILLABLE 3 // Only kill id >= PLAYER_MIN_KILLABLE
```

Estas activan el motor de pisoteo. Con PLAYER_KILLS_ENEMIES activado, el jugador podrá saltar sobre los enemigos para matarlos. PLAYER_MIN_KILLABLE nos sirve para hacer que no todos los enemigos se puedan matar. Ojo con esto: si ponemos un 1 podremos matar a todos, si ponemos un 2, a los enemigos tipo 2 y 3, y si ponemos un 3 sólo a los de tipo 3. O sea, se podrá matar a los enemigos cuyo tipo sea mayor o igual al valor que se configure.

```
//#define PLAYER_BOOTEE // Always jumping engine.
```

Esta directiva activa el salto continuo. No es compatible con PLAYER_HAS_JUMP o con PLAYER_HAS_JETPAC. Si activas PLAYER_BOOTEE, tienes que desactivar los otros dos.

```
//#define PLAYER_BOUNCE_WITH_WALLS // Bounce when hitting a wall.
```

El jugador rebota contra las paredes. Esto también funciona en vista cenital.

```
//#define PLAYER_CUMULATIVE_JUMP // Keep pressing JUMP to JUMP higher
```

Esta funciona en conjunción con PLAYER_HAS_JUMP. Si se define, al pulsar la tecla de salto empezaremos saltando poquito y cada vez que vayamos rebotando ganaremos más y más altura.

Configuración de la pantalla

```
#define VIEWPORT_X 1 //
#define VIEWPORT_Y 0 // Viewport character coordinates
```

Definen la posición (siempre en coordenadas de carácter) del área de juego. Nuestro área de juego empezará en (0, 1), y esos son los valores que le damos a VIEWPORT_X y VIEWPORT_Y.

```
#define LIFE_X 22 //
#define LIFE_Y 21 // Life gauge counter character coordinates
```

Definen la posición del marcador de las vidas.

```
#define OBJECTS_X 17 //
#define OBJECTS_Y 21 // Objects counter character coordinates
```

Definen la posición del contador de objetos, si usamos objetos.

```
#define OBJECTS_ICON_X 15 // Objects icon character coordinates
#define OBJECTS_ICON_Y 21 // (use with ONLY_ONE_OBJECT)
```

Estas dos se utilizan con ONLY_ONE_OBJECT: Cuando "llevemos" el objeto encima, el motor nos lo indicará haciendo parpadear el icono del objeto en el marcador. En OBJECTS_ICON_X e Y indicamos dónde aparece dicho icono (el tile con el dibujito de la caja). Como verás, esto obliga a que estemos usando el icono en el marcador, y no un texto u otra cosa.

Efectos gráficos

```
//#define USE_AUTO_SHADOWS // Automatic shadows made of darker attributes
//#define USE_AUTO_TILE_SHADOWS // Automatic shadows using tiles 32-47.
```

Estas dos se encargan de las sombras de los tiles, y podemos activar sólo una de ellas, o ninguna. Si activamos USE_AUTO_SHADOWS, los tiles obstáculo dibujan sombras sobre los tiles traspasables usando sólo atributos (a veces queda resultón, pero no siempre). USE_AUTO_TILE_SHADOWS emplea versiones sombreadas de los tiles de fondo para hacer las sombras, tal y como explicamos. Desactivando ambas no se dibujarán sombras.

```
//#define UNPACKED_MAP // Full, uncompressed maps.
```

Si definimos UNPACKED_MAP estaremos diciéndole al motor que nuestro mapa es de 48 tiles.

```
//#define NO_MASKS // Sprites are rendered using OR
//#define PLAYER_ALTERNATE_ANIMATION // If defined, animation is 1,2,3,1,2,3...
```

La primera (NO_MASKS) hace que los sprites no estén enmascarados, lo cual ahorra memoria. Sin embargo, el conversor de sprites que actualmente está en la churrera no es capaz de exportar spritesets sin máscaras, por lo que por ahora esta directiva no te servirá de mucho. PLAYER_ALTERNATE_ANIMATION cambia el orden en el que se realiza la animación del personaje cuando anda. Normalmente, la animación es 2, 1, 2, 3, 2, 1, 2, 3... Pero si activas esta directiva será 1, 2, 3, 1, 2, 3,...

Configuración del movimiento del personaje principal

En esta sección configuraremos como se moverá nuestro jugador. Es muy probable que los valores que pongas aquí tengas que irlos ajustando por medio del método de prueba y error. Tendremos lo que se conoce como Movimiento Rectilíneo Uniformemente Acelerado (MRUA) en cada eje: el horizontal y el vertical. Básicamente tendremos una posición (digamos X) que se verá afectada por una velocidad (digamos VX), que a su vez se verá afectada por una aceleración (o sea, AX). Los parámetros siguientes sirven para especificar diversos valores relacionados con el movimiento en cada eje en vista lateral. En los de vista cenital, se tomarán los valores del eje horizontal también para el vertical.

Eje vertical en vista lateral:

Al movimiento vertical le afecta la gravedad. La velocidad vertical será incrementada por la gravedad hasta que el personaje aterrice sobre una plataforma u obstáculo. Además, a la hora de saltar, habrá un impulso inicial y una aceleración del salto, que también definiremos aquí.

```
#define PLAYER_MAX_VY_CAYENDO 512 // Max falling speed
#define PLAYER_G 48 // Gravity acceleration

#define PLAYER_VY_INICIAL_SALTO 96 // Initial junp velocity
#define PLAYER_MAX_VY_SALTANDO 312 // Max jump velocity
#define PLAYER_INCR_SALTO 48 // acceleration while JUMP is pressed

#define PLAYER_INCR_JETPAC 32 // Vertical jetpac gauge
#define PLAYER_MAX_VY_JETPAC 256 // Max vertical jetpac speed
```

Caída libre: La velocidad del jugador, que medimos en pixels/frame será incrementada en PLAYER_G/64 pixels/frame^2 hasta que llegue al máximo especificado por PLAYER_MAX_CAYENDO/64. Estos valores afectan al salto; A mayor gravedad, menos saltaremos y menos nos durará el impulso inicial. Modificando PLAYER_MAX_CAYENDO podremos conseguir que la velocidad máxima, que se alcanzará antes o después dependiendo de PLAYER_G, sea mayor o menor. Usando valores pequeños podemos simular entornos de poca gravedad como el espacio exterior, la superficie de la luna, o el fondo del mar. El valor de 512 (equivalente a 8

píxels por frame) podemos considerarlo el máximo.

Salto: Los saltos se controlan usando tres parámetros. El primero, PLAYER_VY_INICIAL_SALTO, será el valor del impulso inicial; Cuando el jugador pulse la tecla de salto, la velocidad vertical tomará automáticamente el valor especificado en dirección hacia arriba. Mientras se esté pulsando salto, y durante unos ocho frames, la velocidad se seguirá incrementando en el valor especificado por PLAYER_INCR_SALTO hasta que lleguemos al valor PLAYER_MAX_VY_SALTANDO. Esto se hace para que podamos controlar la fuerza del salto pulsando la tecla de salto más o menos tiempo. El período de aceleración, que dura 8 frames, es fijo y no se puede cambiar, pero podemos conseguir saltos más bruscos subiendo el valor de PLAYER_INCR_SALTO y PLAYER_MAX_VY_SALTANDO. Normalmente encontrar los valores ideales exige un poco de prueba y error. Hay que tener en cuenta que al saltar horizontalmente de una plataforma a otra también entran en juego los valores del movimiento horizontal, por lo que si has decidido que en tu juego el prota debería poder sortear distancias de X tiles tendrás que encontrar la combinación óptima jugando con todos los parámetros.

Los dos valores que quedan tienen que ver con el jetpac. El primero es la aceleración que se produce mientras pulsamos la tecla arriba y el segundo el valor máximo que puede alcanzarse. Si tu juego no usa jetpac estos valores no se utilizan para nada.

Eje horizontal en vista lateral / comportamiento general en vista cenital:

El siguiente set de parámetros describen el comportamiento del movimiento en el eje horizontal si tu juego es en vista lateral, o los de ambos ejes si es en vista cenital. Estos parámetros son mucho más sencillos:

```
#define PLAYER_MAX_VX 256 // Max velocity
#define PLAYER_AX 48 // Acceleration
#define PLAYER_RX 64 // Friction
```

La primera, PLAYER_MAX_VX, indica la velocidad máxima a la que el personaje se desplazará horizontalmente (o en cualquier dirección si es en vista cenital). Cuanto mayor sea este número, más correrá, y más lejos llegará cuando salte (horizontalmente). Un valor de 256 significa que el personaje correrá a un máximo de 4 píxels por frame. PLAYER_AX controla la aceleración que sufre el personaje mientras el jugador pulsa una tecla de dirección. Cuando mayor sea el valor, antes se alcanzará la velocidad máxima. Valores pequeños hacen que "cueste arrancar". Un valor de 48 significa que se tardará aproximadamente 6 frames (256/48) en alcanzar la velocidad máxima. PLAYER_RX es el valor de fricción o rozamiento. Cuando el jugador deja de pulsar la tecla de movimiento, se aplica esta aceleración en dirección contraria al movimiento. Cuanto mayor sea el valor, antes se detendrá el personaje. Un valor de 64 significa que tardará 4 frames en detenerse si iba al máximo de velocidad. Usar valores pequeños de PLAYER_AX y PLAYER_RX harán que el personaje parezca resbalar. Casi siempre "se juega mejor" si el valor de PLAYER_AX es mayor que el de PLAYER_RX.

Comportamiento de los tiles

Tiles que matan, que son obstáculos, plataformas, o interactuables. En esta sección de *config.h* definimos el comportamiento de cada uno de los 48 tiles del tileset completo. Tendremos que definir los comportamientos de los 48 tiles sin importar que nuestro juego utilice un tileset de 16. Por lo general, en estos casos, todos los tiles del 16 al 47 tendrán tipo "0", pero es posible que necesitemos otros valores si empleamos los tiles extra para pintar gráficos y adornos desde el script. Para poner los valores, simplemente abrimos el tileset y nos fijamos, están en el mismo orden en el que aparecen los tiles en el tileset:

```
unsigned char comportamiento_tiles [] = {
0, 8, 8, 8, 8, 8, 0, 8, 4, 0, 8, 1, 0, 0, 0,10,
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
};
```

Como vemos, tenemos el primer tile vacío (tipo 0), luego tenemos cinco tiles que son obstáculos (tipo 8), otros de tipo 4 (plataforma), tipo 1 (unos pinchos que matan), y un tile cerrojo (tipo 10, interactuable).

Características nuevas:

```
#define BREAKABLE_WALLS // Breakable walls
#define BREAKABLE_WALLS_LIFE 1 // Amount of hits to break wall
```

La primera directiva, BREAKABLE_WALLS, activa esta característica e incluye todo el código necesario para que haya tiles destructibles (si no la activas, por mucho que tengas tiles de tipo 16 no pasará nada). La segunda, BREAKABLE_WALLS_LIFE, define el número de disparos que deben recibir los tiles destructibles para romperse.

Para combinar tipos de tiles, sólo hay que sumar los tipos que queramos combinar. De ahí que haya "agujeros" en la numeración. Por ejemplo, para hacer un obstáculo que se pueda romper, tendríamos que hacer que ese tile tuviese tipo obstáculo (8) y tipo destructible (16), o sea, que fuese de tipo 8 + 16 = 24.

```
#define MAX_AMMO 99 // If defined, ammo is not infinite!
#define AMMO_REFILL 50 // type 3 hotspots refill amo, using tile 20
//#define INITIAL_AMMO 0 // If defined, ammo = X when entering game.
```

La primera directiva, MAX_AMMO, si está definida, hace que las balas se acaben y que su valor máximo sea el que se especifica. Si quieres balas infinitas, simplemente comenta esta definición. La siguiente, AMMO_REFILL, indica cuántas balas pillamos cuando cogemos una recarga de munición. La tercera, INITIAL_AMMO, si está definida, hace que al principio del juego tengamos el número de balas especificado. Si no se define, el número de balas al empezar será el maximo, o sea, el que dice en MAX_AMMO. Para colocar las recargas lo haremos con el colocador, y usando los hotspots. Hasta ahora habíamos usado hotspots de tipo 1 para los objetos y de tipo 2 para las llaves. Para colocar recargas, necesitaremos hotspots de tipo 4. Además, en nuestro tileset, el tile número 20 debe ser el correspondiente a la recarga.

```
#define TYPE_7_FIXED_SPRITE 4 // If defined, type 7 enemies are always #
#define EVERYTHING_IS_A_WALL // If defined, any tile <> type 0 is a wall.
```

Hasta ahora, los enemigos de tipo 7 salían del sitio que tú definías en el colocador y se liaban a perseguirte. El gráfico asociado era elegido al azar entre los cuatro enemigos del spriteset. Ahora permitimos que seas tú el que decida qué gráfico lleva, y que sea este siempre. Además, estos enemigos avanzaban pero se veían detenidos por los tiles de tipo obstáculo (tipo 8). Ahora podemos configurar que cualquier tipo de tile que no sea traspasable (tipo 0) los detenga. La primera directiva es para fijar un número de gráfico de enemigo para los enemigos de tipo 7. La segunda es para que los enemigos te persigan solo por los tiles de tipo 0. Si comentas esta última, el comportamiento será el mismo que hasta ahora: te perseguirán por cualquier tile y se detendrán solo con los de tipo 8.

```
#define TIMER_ENABLE
```

Se añade a la churrera un temporizador que podemos usar de forma automática o desde el script. El temporizador toma un valor inicial, va contando hacia abajo, puede recargarse, se puede configurar cada cuántos frames se decrementa o decidir qué hacer cuando se agota.Con TIMER_ENABLE se incluye el código necesario para manejar el temporizador. Este código necesitará algunas otras directivas que especifican la forma de funcionar:

```
#define TIMER_INITIAL 99
```

```
#define TIMER_REFILL 25
#define TIMER_LAPSE 32
```

TIMER_INITIAL especifica el valor inicial del temporizador. Las recargas de tiempo, que se ponen con el colocador como hotspots de tipo 5, recargarán el valor especificado en TIMER_REFILL. El valor máximo del timer, tanto para el inicial como al recargar, es de 99. Para controlar el intervalo de tiempo que transcurre entre cada decremento del temporizador, especificamos en TIMER_LAPSE el número de frames que debe transcurrir.

```
#define TIMER_START
```

Si se define TIMER_START, el temporizador estará activo desde el principio. Tenemos, además, algunas directivas que definen qué pasará cuando el temporizador llegue a cero. Hay que descomentar las que apliquen:

```
#define TIMER_SCRIPT_0
```

Definiendo esta, cuando llegue a cero el temporizador se ejecutará una sección especial del script, ON_TIMER_OFF. Es ideal para llevar todo el control del temporizador por scripting.

```
//#define TIMER_GAMEOVER_0
```

Definiendo esta, el juego terminará ("GAME OVER") cuando el temporizador llegue a cero.

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

Si se define TIMER_KILL_0, se restará una vida cuando el temporizador llegue a cero. Si, además, se define TIMER_WARP_TO, además se cambiará a la pantalla espeficada, apareciendo el jugador en las coordenadas TIMER_WARP_TO_X y TIMER_WARP_TO_Y. Usa esto si estos datos van a ser fijos durante el juego.

```
//#define TIMER_AUTO_RESET
```

Si se define esta opción, el temporizador volverá al máximo tras llegar a cero de forma automática. Si vas a realizar el control por scripting, mejor deja ésta comentada.

```
#define SHOW_TIMER_OVER
```

Si se define ésta, en el caso de que hayamos definido o bien TIMER_SCRIPT_0 o bien TIMER_KILL_0, se mostrará un cartel de "TIME'S UP!" cuando el temporizador llegue a cero.

Desde el script se definen estas comprobaciones y comandos:

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

Que se cumplirán si el valor del temporizador es mayor o igual o menor o igual que el valor especificado, respectivamente.

```
SET_TIMER a, b
```

Sirve para establecer los valores TIMER_INITIAL (a) y TIMER_LAPSE (b) desde el script.

```
TIMER_START
```

Sirve para iniciar el temporizador.

```
TIMER_STOP
```

Sirve para parar el temporizador.

Acerca de los tiles empujables:

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

Activando ENABLE_PUSHED_SCRIPTING, el tile que se pisa y sus coordenadas se almacenarán en los flags especificados por las directivas MOVED_TILE_FLAG, MOVED_X_FLAG y MOVED_Y_FLAG, respectivamente. En el código que se muestra, el tile pisado se almacenará en el flag 1, y sus coordenadas en los flags 2 y 3.

```
#define PUSHING_ACTION
```

Si definimos ésta, además, cuando empujemos un bloque se ejecutarán las secciones PRESS_FIRE AT ANY y PRESS_FIRE de la pantalla actual del script. En este caso, se cumplirá la condición "JUST_PUSHED" y desde el scripting podremos controlar qué pasa cuando empujamos un bloque.

Es aconsejable poner límites en tu mapa para que el jugador no pueda escaparse del mapa y que el motor haga cosas raras, pero si tu mapa es estrecho puede que quieras aprovechar toda la pantalla. En ese caso, puedes activar:

```
#define PLAYER_CHECK_MAP_BOUNDARIES
```

Que añadirá comprobaciones y no dejará que el jugador se salga del mapa. ¡Ojo! Si puedes evitar usarlo, mejor. Ahorrarás espacio.

Hasta ahora habíamos dejado sin código los enemigos de tipo 6. Para usarlos, ponlos en el colocador de enemigos como tipo 6 y usa estas directivas...

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

La primera los activa, la segunda define qué sprite va a usar (menos 1, si quieres el sprite del enemigo 3, pon un 2. Sorry por la guarrada, pero ahorro bytes). La tercera dice cuántos píxels ve de lejos el bicho. Si te ve, te persigue. Si no, vuelve a su sitio (donde lo hayas puesto con el colocador).

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

Define ahí la aceleración y la velocidad máxima de tus tipo 6. Si vas a usar también tipo 5 y quieres otros valores, sé un hombre y modifica el motor.

Configuración de teclado / joystick para dos botones:

Hay Juegos de vista lateral que se juegan mejor con dos botones, uno de salto y otro de disparo. Si activas esta directiva:

```
#define USE_TWO_BUTTONS
#define FANTY_A 12
```

El teclado será el siguiente, en vez del habitual:
A izquierda
D derecha
W arriba

S abajo
N salto
M disparo

Si se elige algún joystick, FIRE y M disparan, y N salta.

Ahora podrás permitir que el jugador dispare hacia arriba o en diagonal. Para ello define esto:

```
#define CAN_FIRE_UP
```

Esta configuración funciona mejor con USE_TWO_BUTTONS, ya que así separamos "arriba" del botón de salto. Si no pulsas "arriba", el personaje disparará hacia donde esté mirando. Si pulsas "arriba" mientras disparas, el personaje disparará hacia arriba. Si, además, estás pulsando una dirección, el personaje disparará en la diagonal indicada.
Por velocidad, las balas no llevan máscaras. Esto funciona bien si el fondo sobre el que se mueven es oscuro (pocos pixels INK activos). Sin embargo, hay situaciones en las que esto no ocurre y se ve mal. En ese caso, podemos activar máscaras para las balas:

```
#define MASKED_BULLETS
```

**SCRIPTING**

Secciones:

*ENTERING SCREEN n:* con n siendo un número de pantalla, se ejecutan justo al entrar en una nueva pantalla, una vez dibujado el escenario, inicializados los enemigos, y colocados los hotspots. Las podemos utilizar para modificar el escenario o inicializar variables. Por ejemplo, asociada a la pantalla 3, podemos colocar un script que compruebe si hemos matado a todos los enemigos y, si no, que pinte un obstáculo para que no podamos pasar.

*ENTERING ANY:* se ejecutan para TODAS las pantallas, justo antes de ENTERING SCREEN n. O sea, cuando tú entras en la pantalla 3, se ejecutará primero la sección ENTERING ANY (si existe en el script), y justo después se ejecutará la sección ENTERING SCREEN 3 (si existe en el script).

*ENTERING GAME:* se ejecuta una sola vez al empezar el juego. Es lo primero que se ejecuta. Lo puedes usar para inicializar el valor de variables, por ejemplo.

*PRESS_FIRE AT SCREEN n:* con n siendo un número de pantalla, se ejecuta en varios supuestos estando en la pantalla n: si el jugador pulsa el botón de acción, al empujar un bloque si hemos activado la directiva PUSHING_ACTION, o al entrar en una zona especial definida desde scripting llamada "fire zone" si hemos activado la directiva ENABLE_FIRE_ZONE. Normalmente usaremos estas secciones para reaccionar a las acciones del jugador.

*PRESS_FIRE AT ANY:* se ejecuta en todos los supuestos anteriores, para cualquier pantalla, justo antes de PRESS_FIRE AT SCREEN n. O sea, si pulsamos acción en la pantalla 7, se ejecutarán las cláusulas de PRESS_FIRE AT ANY y luego las de PRESS_FIRE AT SCREEN 7.

*ON_TIMER_OFF:* se ejecuta cuando el temporizador llegue a cero, si tenemos activado el temporizador y hemos configurado que ocurra esto con la directiva TIMER_SCRIPT_0.

Guardando valores: Los flags

Los flags, son variables donde podemos almacenar valores que posteriormente podremos consultar o modificar desde el script. Muchas veces necesitaremos recordar que hemos hecho algo, o contabilizar cosas. Para ello tendremos que

almacenar valores, y para ello tenemos las flags. En principio, tenemos 16 flags, numeradas del 0 a al 15, aunque este número puede modificarse fácilmente cambiando una definición de *definitions.h* (#define MAX_FLAGS). Cada flag puede almacenar un valor de 0 a 255. La mayoría del tiempo sólo estaremos almacenando un valor booleano (0 o 1). En el script, la mayoría de las comprobaciones y comandos toman valores numéricos. Por ejemplo, *IF PLAYER_TOUCHES 4,5* evaluará a "cierto" si el jugador está tocando la casilla de coordenadas (4, 5). Si anteponemos un # al número, estaremos referenciando el valor del flag correspondiente, de forma que *IF PLAYER_TOUCHES #4, #5* evaluará a "cierto" si el jugador está tocando la casilla de coordenadas almacenadas en los flags 4 y 5, sea cual sea este valor.

*¿Cómo activar el scripting?*

tendremos que hacer dos cosas: primero activarlo y configurarlo en *config.h*, y luego modificar nuestro *make.bat* para incluirlo en el proyecto. Empecemos por *config.h*. Las directivas relacionadas con la activación y configuración del scripting son estas:

```
#define ACTIVATE_SCRIPTING // Activates msc scripting and flag related stuff.
#define SCRIPTING_DOWN // Use DOWN as the action key.
//#define SCRIPTING_KEY_M // Use M as the action key instead.
//#define SCRIPTING_KEY_FIRE // User FIRE as the action key instead.
//#define ENABLE_EXTERN_CODE // Enables custom code to be run using EXTERN n
//#define ENABLE_FIRE_ZONE // Allows to define a zone which auto-triggers "FIRE"
```

ACTIVATE_SCRIPTING, es la que activará el motor de scripting, y añadirá el código necesario para que se ejecute la sección correcta del script en el momento preciso. Es la que tenemos que activar sí o sí. De las tres siguientes, tendremos que activar solo una, y sirven para configurar qué tecla será la tecla de acción, la que lance los scripts PRESS_FIRE AT ANY y PRESS_FIRE AT SCREEN n. La primera, SCRIPTING_DOWN, configura la tecla "abajo". Esta es perfecta para perspectiva lateral, ya que esta tecla no se usa para nada más. La segunda, SCRIPTING_KEY_M habilita la tecla "M" para lanzar el script. La tercera, SCRIPTING_KEY_FIRE, configura la tecla de disparo (o el botón del joystick) para tal menester. Obviamente, si tu juego incluye disparos, no puedes usar esta configuración. La siguiente directiva, ENABLE_EXTERN_CODE, la dejarás normalmente desactivada a menos que seas un maestro churrero. Hay un comando de script especial, EXTERN n, donde n es un número, que lo que hace es llamar a una función de C situada en el archivo *extern.h* pasándole ese número. En esta función puedes añadir el código C que te de la gana. Por último, ENABLE_FIRE_ZONE sirve para que podamos definir un rectángulo especial dentro del area de juego de la pantalla en curso. Normalmente, usaremos en ENTERING SCREEN n para definir el rectángulo usando el comando SET_FIRE_ZONE x1, y1, x2, y2. Cuando el jugador esté dentro de este rectángulo especial, se ejecutarán los scripts PRESS_FIRE AT ANY y PRESS_FIRE AT SCREEN n de la pantalla actual. Esto viene realmente bien para poder ejecutar acciones sin que el jugador tenga que pulsar la tecla de acción.

Lo siguiente es configurar bien *make.bat*, activando la compilación e inclusión del script. Si abres *make.bat* y te fijas, verás que al principio hay una llamada a *msc*. Este es el compilador de scripts, que recibe el nombre del archivo de script, el nombre de salida (que será *msc.h*) y el número de pantallas total de tu juego:

```
echo ### COMPILANDO SCRIPT ###
cd ..script
msc cadaver.spt msc.h 20
copy *.h ..dev
```

El nombre de tu script será el nombre de tu juego con un *.spt* como extensión. El archivo se ubica en */script*. Si entras en */script* verás un *churromain.spt*. Cámbiale el nombre para que coincida con el nombre de tu juego (el mismo de tu archivo .c de dev). No te olvides del número de pantallas, que es muy

importante. Si no lo pones bien el código del intérprete de scripts se generará mal. Si ahora vas a */script* y abres el archivo con el script (que antes se llamaba *churromain.spt* y has renombrado con el nombre de tu juego) verás que está vacío, o casi. Sólo trae un esqueleto de una sección.

```
# Título tonto
# Copyleft 201X tu grupo roneón soft.
# Churrera 3.1
# flags:
# 1 -
ENTERING GAME
IF TRUE
THEN
SET FLAG 1 = 0
END
END
```

Como por ahora las variables (flags) se identifican con un número es buena idea hacerse una lista de qué hace cada una para luego no liarnos.

```
# Cadàveriön
# Copyleft 2013 Mojon Twins
# Churrera 3.99.2
# flags:
# 1 - Tile pisado por el bloque que se empuja
# 2, 3 - coordenadas X e Y
# 4, 5 - coordenadas X e Y del tile "retry"
# 6, 7 - coordenadas X e Y del tile puerta
# 8 - número de pantallas finalizadas
# 9 - número de estatuas que hay que colocar
# 10 - número de estatuas colocadas
# 11 - Ya hemos quitado la cancela
# 12 - Pantalla a la que volvemos al agotarse el tiempo
# 13, 14 - Coordenadas a las que volvemos... bla
# 15 - Piso
# 0 - valor de 8 almacenado
# 16 - Vendo moto seminueva.
```

vamos a crear un script sencillo que introduzca adornos en algunas pantallas. Vamos a extender el tileset de Dogmole, incluyendo nuevos tiles que no colocaremos desde el mapa (porque ya hemos usado los 16 que tenemos como máximo), sino que colocaremos desde el script. Ahí hay un montón de cosas que vamos a colocar desde el scripting. El sitio ideal para hacerlo son las secciones ENTERING SCREEN n de las pantallas que queramos adornar, ya que se ejecutan cuando todo lo demás está en su sitio (*scriptdogmole.spt*).
El pintado de los tiles extra se hace desde la lista de comandos de una cláusula. Como queremos que la cláusula se ejecute siempre, emplearemos la condición más sencilla que existe: la que siempre evalúa a cierto:

```
# Vestíbulo de la universidad
ENTERING SCREEN 0
# Decoración y pedestal
IF TRUE
THEN
END
END
```

Esto significa que siempre que entremos en la pantalla 0, se ejecutarán los comandos de la lista de comandos de esa cláusula, ya que su única condición SIEMPRE evalúa a cierto. El comando para pintar un tile en la pantalla tiene esta forma:

```
SET TILE (x, y) = t
```

Donde (x, y) es la coordenada (recuerda, tenemos 15x10 tiles en la pantalla, por lo que x podrá ir de 0 a 14 y de 0 a 9) y t es el número del tile que queremos pintar. Con el mapa abierto en mappy delante para contar casillas y ver donde tenemos que pintar las cosas, vamos colocando primero el pedestal y luego todas las decoraciones:

```
# Vestíbulo de la universidad
ENTERING SCREEN 0
# Decoración y pedestal
IF TRUE
THEN
# Pedestal
SET TILE (3, 7) = 22
SET TILE (4, 7) = 23
# Decoración
SET TILE (1, 5) = 29
SET TILE (1, 6) = 20
SET TILE (1, 7) = 21
SET TILE (6, 6) = 20
SET TILE (6, 7) = 21
SET TILE (7, 7) = 28
SET TILE (1, 2) = 27
SET TILE (1, 3) = 28
SET TILE (2, 2) = 29
SET TILE (2, 3) = 27
SET TILE (3, 2) = 32
SET TILE (3, 3) = 33
SET TILE (9, 1) = 30
SET TILE (9, 2) = 30
SET TILE (9, 3) = 31
END
END

ENTERING GAME
IF TRUE
THEN
WARP_TO 0, 12, 2
END
END
```

¿Qué hace esto? Pues hará que, al empezar el juego, se ejecute esa lista de cláusulas formada por una única cláusula, que siempre se ejecutará (porque tiene IF TRUE) y que lo que hace es trasladarnos a la coordenada (12, 2) de la pantalla 0, porque eso es lo que hace el comando WARP:

```
WARP_TO n, x, y
```

Nos traslada a la pantalla x, y nos hace aparecer en las coordenadas (x, y).

Un ejemplo; El script de Dogmole:

El script de este juego va a ser muy sencillo. Lo primero que tenemos que mirar es qué vamos a necesitar almacenar para destinar algunas flags para ello. En nuestro caso, como el motor ya se encarga de contar los monjes que hemos matado, sólo necesitaremos ir contando las cajas que vamos depositando y además necesitaremos recordar si hemos quitado la piedra o no. Vamos a usar dos flags: la 1 y la 3.

```
# flags:
# 1 - cuenta general de objetos.
# 3 - 1 = puerta de la universidad abierta.
```

Si el flag 3 vale 1, significará que hemos matado a todos los monjes, y en ese

caso habría que modificar esa pantalla para borrar la piedra de la posición que tenemos anotada.

```
# Entrada de la universidad
ENTERING SCREEN 2
      # Control de la puerta de la universidad.
      IF FLAG 3 = 1
      THEN
            SET TILE (12, 7) = 0
      END
END
```

Un buen sitio para comprobar que hemos matado a todos los monjes es al entrar en cualquier pantalla, o sea, en nuestra sección ENTERING ANY.

```
# Abrir la universidad
ENTERING ANY
      IF ENEMIES_KILLED_EQUALS 20
      IF FLAG 3 = 0
      THEN
            SET FLAG 3 = 1
            SOUND 7
            SOUND 8
            SOUND 9
            SOUND 7
            SOUND 8
            SOUND 9
      END
END
```

El pedestal de objetos está en la pantalla 0. Ocupa las posiciones (3, 7) y (4, 7). Vamos a escribir ahora un trozo de script que, si pulsamos la tecla de acción en la pantalla 0, comprueba que estamos tocando el pedestal y que llevamos un objeto, para eliminar ese objeto e incrementar en uno la cuenta. Lo primero que tenemos que resolver es la detección de que estamos tocando el pedestal. Si el pedestal ocupase un sólo tile en (x, y), sería muy sencillo:

```
IF PLAYER_TOUCHES x, y
```

En vez de esto, usaremos...

```
IF PLAYER_IN_X x1, x2
IF PLAYER_IN_Y y1, y2
```

tx1 valdra 3, ty1 valdrá 7, tx2 valdrá 4 y ty2 valdrá también 7. De ese modo, siguiendo las fórmulas:

```
x1 = 3 * 16 - 15 = 33
x2 = 4 * 16 + 15 = 79

y1 = 7 * 16 - 15 = 97
y2 = 7 * 16 + 15 = 127
```

O sea, que para tocar el pedestal, el sprite debe tener el pixel superior izquierdo entre 33 y 79 en la coordenada X y entre 97 y 127 en la coordenada Y. Además, tendremos que comprobar que llevemos una caja en el inventario. Sería algo así:

```
PRESS_FIRE AT SCREEN  0
      # Detectar pedestal.
      # Lo detectamos definiendo un rectángulo de píxels.
      # Luego comprobamos si el jugador ha cogido un objeto.
      # Si todo se cumple, decrementamos el número de objetos e incrementamos
FLAG 1
```

```
      IF PLAYER_IN_X 33, 79
      IF PLAYER_IN_Y 97, 127
      IF PLAYER_HAS_OBJECTS
      THEN
            INC FLAG 1, 1
            DEC OBJECTS 1
            SOUND 7
      END
END
      # Fin del juego
      # Si llevamos 10 cajas, ¡hemos ganado!
      IF FLAG 1 = 10
      THEN
            WIN GAME
      END IF
END
```

"la zona de fuego", o fire zone. Esta zona de fuego no es más que un rectángulo en pantalla, especificado en píxels. Si el jugador entra en el rectángulo, el motor se comporta como si hubiese pulsado acción. La zona de fuego se desactiva automáticamente al cambiar de pantalla, por lo que si la definimos en un ENTERING SCREEN n, sólo estará activo mientras estemos en esa pantalla. Si al entrar en la pantalla 0 definimos una zona de fuego alrededor del mostrador, en cuanto el jugador lo toque se ejecutará la lógica que hemos programado en el script para dejar el objeto que lleve e incrementar el contador. La zona de fuego se define con el comando SET_FIRE_ZONE, que recibe las coordenadas x1, y1, x2, e y2 del rectángulo que queramos usar como zona de fuego. Si queremos hacer coincidir la zona de fuego con un rectángulo formado por tiles, como es nuestro caso, se aplican las mismas fórmulas que explicamos antes. O sea, que vamos a usar exactamente los mismos valores.
Tenemos que activar la directiva correspondiente en nuestro *config.h*:

```
#define ENABLE_FIRE_ZONE // Allows to define a zone which auto-triggers "FIRE"
```

Hecho esto, sólo tendremos que modificar la sección ENTERING SCREEN 0 añadiendo al final el comando...

```
# Fire zone (x1, y1, x2, y2):
SET_FIRE_ZONE 33, 97, 79, 127
```

<u>Más ejemplos; Sgt. Helmet Training Day</u>

En este juego la misión es recoger las cinco bombas, llevarlas a la pantalla del ordenador (pantalla 0) para depositarlas, y luego volver al principio (pantalla 24). Hay muchas formas de hacer esto. La que usamos nosotros para montarlos es bastante sencilla:
Podemos contar el número de objetos que llevamos desde el script, por lo que las bombas serán objetos normales y corrientes del motor. Las colocamos con el colocador como hotspot de tipo 1. Cuando lleguemos a la pantalla del ordenador, haremos una animación chula colocando las bombas alrededor. Usamos el colocador porque mola para saber las coordenadas de cada casilla (si pones el ratón sobre una casilla salen las coordenadas arriba del todo) y apuntamos en un papel donde las vamos a pintar. Usaremos el flag 1 para comprobar que hemos colocado las bombas. Al principio del juego valdrá 0, y lo pondremos a 1 cuando coloquemos las bombas. Cuando entremos en la pantalla 24, que es la pantalla principal, comprobaremos el valor del flag 1, y si vale 1, terminará el juego. Además, iremos imprimiendo textos en la pantalla con lo que vamos haciendo.

Recordemos que en *config.h* había tres directivas que mencionamos por encima hace algunos capítulos:

```
#define LINE_OF_TEXT          0     // If defined, scripts can show text @ Y = #
#define LINE_OF_TEXT_X        1     // X coordinate.
#define LINE_OF_TEXT_ATTR     71    // Attribute
```

Sirven para configurar donde sale una linea de texto que podremos escribir desde el script con el comando TEXT. Para ello dejamos sitio libre en el marco.
Lo primero que hará nuestro script, por tanto, será definir un par de mensajes que aparecerán por defecto al entrar en cada pantalla, dependiendo de valor del flag 1. Esto lo hacemos en la sección ENTERING ANY. Esta sección, recordemos, se ejecuta al entrar en cada pantalla, justo antes de la sección ENTERING SCREEN n correspondiente. Atención a esto: nos permitirá definir un texto general que podamos sobrescribir fácilmente si hace falta para alguna pantalla en concreto, ya que si ponemos texto en ENTERING SCREEN n sobrescribirá el que pusimos en ENTERING ANY al ejecutarse después. Para imprimir texto en la linea de texto definida, usamos el comando TEXT. El texto que le sigue va sin comillas. Usaremos el carácter de subrayado _ para representar los espacios. Es conveniente, además, rellenar con espacios para que, si hay que sobrescribir un texto largo con uno corto, se borre entero. La longitud máxima de los textos dependerá de tu marco de juego y de cómo hayas definido su posición. En nuestro caso la hemos colocado en (x, y) = (1, 0) porque tenemos borde a la izquierda y a la derecha, con lo que la longitud máxima será de 30 caracteres. Escribamos nuestra sección ENTERING ANY, pues. Hemos dicho que imprimiremos un texto u otro dependiendo del valor del flag 1:

```
ENTERING ANY
    IF FLAG 1 = 0
    THEN
        TEXT BUSCA_5_BOMBAS_Y_EL_ORDENADOR!
    END
    IF FLAG 1 = 1
    THEN
        TEXT MISION_CUMPLIDA!_VUELVE_A_BASE
    END
END
```

Vamos ahora con la chicha. Lo primero que haremos será escribir las condiciones para la pantalla del ordenador, que es la pantalla 0. En esta pantalla tenemos que hacer varias cosas. Tengámoslas claras antes de empezar:
Siempre que entremos tendremos que pintar el ordenador, que está compuesto por los tiles 32 a 38 del tileset. Lo haremos como hemos visto, con SET TILE.
Además, tendremos que definir un area de fuego alrededor del ordenador para que el juego detecte automáticamente cuando nos acercamos a él. Si volvemos a entrar en la pantalla después de haber colocado las bombas (puede pasar), tendremos que coscarnos de ello y pintar también las bombas. Si entramos por primera vez (no hemos puesto las bombas) escribiremos un mensajito de ayuda que diga "PON LAS CINCO BOMBAS Y CORRE". Si nos acercamos al ordenador, habrá que hacer la animación chula de poner las bombas, y además colocar el flag 1 a 1. Las cuatro primeras cosas se hacen al entrar en la pantalla, y la última al pulsar acción (o entrar en la zona de fuego).

```
ENTERING SCREEN 0
    # Siempre: pintar el ordenador.
    IF TRUE
    THEN
        SET TILE (6, 3) = 32
        SET TILE (7, 3) = 33
        SET TILE (8, 3) = 34
        SET TILE (6, 4) = 36
        SET TILE (7, 4) = 37
        SET TILE (8, 4) = 38
        SET_FIRE_ZONE 80, 32, 159, 95
    END
```

El área equivale al rectángulo formado desde el tile (x, y) = (5, 2) hasta el (9, 5). O sea, un reborde de un tile alrededor de los seis tiles que ocupa el ordenador.

```
    # Si ya hemos puesto las bombas: pintarlas.
```

```
    IF FLAG 1 = 1
    THEN
        SET TILE (4, 4) = 17
        SET TILE (4, 2) = 17
        SET TILE (7, 1) = 17
        SET TILE (10, 2) = 17
        SET TILE (10, 4) = 17
    END
```

Ahora solo queda poner un texto de ayuda si no hemos colocado aún las bombas. Como esta sección se ejecuta después de ENTERING ANY, el texto que imprimamos aquí sobrescribirá el que ya hubiese. Es por eso, además, que usamos espacios en blanco alrededor: centrarán el texto y eliminarán los caracteres del texto anterior, que es más largo:

```
    # Si no, mensajito.
    IF FLAG 1 = 0
    THEN
        TEXT _PON_LAS_CINCO_BOMBAS_Y_CORRE_
    END
END
```

Ahora sólo queda reaccionar a la zona de fuego, en la sección PRESS_FIRE AT SCREEN 0. Haremos algunas comprobaciones y luego haremos la animación:

```
PRESS_FIRE AT SCREEN 0
    IF PLAYER_IN_X 80, 159
    IF PLAYER_IN_Y 32, 95
    IF OBJECT_COUNT = 5
    IF FLAG 1 = 0
    THEN
        SET FLAG 1 = 1
        SET TILE (4, 4) = 17
        SHOW
        SOUND 0
        SET TILE (4, 2) = 17
        SHOW
        SOUND 0
        SET TILE (7, 1) = 17
        SHOW
        SOUND 0
        SET TILE (10, 2) = 17
        SHOW
        SOUND 0
        SET TILE (10, 4) = 17
        SHOW
        SOUND 0
        TEXT ____AHORA_VUELVE_A_LA_BASE____
    END
END
```

Veámoslo poco a poco. Lo primero es comprobar que estamos donde tenemos que estar (el jugador siempre puede pulsar la tecla acción en vez de entrar en la zona de fuego, y no mola ejecutarlo si el jugador está en cualquier sitio). Eso lo hacemos como ya hemos visto: con PLAYER_IN_X y PLAYER_IN_Y y las mismas coordenadas de la zona de fuego. Lo siguiente es comprobar que tenemos las cinco bombas, o lo que es lo mismo, que tenemos cinco objetos. Esto se hace con OBJECT_COUNT, que representa el número de objetos que el jugador lleva recogidos. Por último, muy importante, hay que comprobar que aún no hemos dejado las bombas, o cosas divertidas podrían pasar. Si se cumplen todas estas condiciones, pondremos el flag 1 a 1 (ya hemos puesto las bombas) y hacemos la animación, que consiste en ir pintando una a una las bombas y tocando un sonido. Ves ahí el comando SHOW, necesario porque los cambios que hagamos en la pantalla no serán visibles hasta que se actualice, cosa que pasa normalmente al volver al

bucle principal, pero no en medio de la ejecución de una cláusula. Como queremos que se vea cada bomba justo después de pintarla, llamamos a SHOW. Cada sonido, además, parará la ejecución durante unos instantes (estamos en modo 48K), lo que nos viene genial. Por último, imprimiremos un texto de ayuda, de nuevo con espacios a los lados para completar los 30 caracteres máximos y borrar lo que hubiese del texto anterior.

Si seguimos con nuestro guión, lo próximo que había que hacer era volver a la pantalla inicial, que es la 24. Lo que queda por hacer es bastante sencillo: consiste en comprobar, al entrar en la pantalla 24, que el flag 1 vale 1. Esto sólo pasará si anteriormente hemos colocado las bombas, por lo que no necesitamos más. Simplemente comprobamos eso y, si se cumple, terminamos el juego con éxito. Nada más sencillo que hacer esto:

```
ENTERING SCREEN 23
    IF FLAG 1 = 1
    THEN
        WIN
    END
END
```

**Preparando nuestro script de compilación; "make.bat"**

```
@echo off
cd ..\map
..\utils\mapcnv mapa.map 8 3 15 10 15 packed
copy mapa.h ..\dev
cd ..\dev
zcc +zx -vn dogmole.c -o dogmole.bin -lndos -lsplib2 -zorg=25000
..\utils\bas2tap -a10 -sLOADER loader.bas loader.tap
..\utils\bin2tap -o screen.tap -a 16384 loading.bin
..\utils\bin2tap -o main.tap -a 25000 dogmole.bin
copy /b loader.tap + screen.tap + main.tap dogmole.tap
del loader.tap
del screen.tap
del main.tap
del dogmole.bin
echo DONE
```

El script cambia al directorio del mapa para volver a generar *mapa.h* y copiarlo a */dev*, por si acaso hemos hecho algún cambio para que quede reflejado automáticamente. Esta es una de las líneas que tendrás que cambiar, indicando los parámetros correctos de *mapcnv* (en concreto, el tamaño en pantallas, o si no usas cerrojos para poner 99 en lugar de 15, o si usas mapas de 48 tiles para quitar packed):

```
cd ..\map
..\utils\mapcnv mapa.map 8 3 15 10 15 packed
copy mapa.h ..\dev
cd ..\dev
```

La siguiente linea compila el juego:

```
zcc +zx -vn dogmole.c -o dogmole.bin -lndos -lsplib2 -zorg=25000
```

Esto no es más que una ejecución del compilador z88dk que toma como fuente *dogmole.c* (y todos los archivos .h que incluye), y saca un binario en código máquina *dogmole.bin*. La dirección de compilación es 25000.

Acto seguido, el script crea tres cintas en formato *.tap*. La primera contiene un cargador en BASIC (que puedes modificar si te pones muy friki, el fuente está en *loader.bas*). La siguiente contiene la pantalla de carga. La última contiene el *dogmole.bin* que acabamos de compilar:

```
..\utils\bas2tap -a10 -sLOADER loader.bas loader.tap
..\utils\bin2tap -o screen.tap -a 16384 loading.bin
..\utils\bin2tap -o main.tap -a 25000 dogmole.bin
```

Fíjate en el -sLOADER de la primera linea, la que genera el cargador BASIC. Ese es el nombre que aparece tras el Program: al cargar la cinta en el Spectrum, así que, si quieres, puedes cambiarlo por otra cosa. Recuerda que los nombres de archivo de cinta de Spectrum pueden tener un máximo de 10 caracteres. Vamos a cambiarlo para que salga DOGMOLE:

```
..\utils\bas2tap -a10 -sDOGMOLE loader.bas loader.tap
```

Cuando ya tenemos las tres cintas, cada una con un bloque (cargador, pantalla, y juego) lo único que queda por hacer es unirlas:

```
copy /b loader.tap + screen.tap + main.tap dogmole.tap
```

Y, para terminar, hacemos un poco de limpieza, eliminando los archivos intermedios que ya no nos sirven para nada:

```
del loader.tap
del screen.tap
del main.tap
del dogmole.bin
```

**Compilando**

Lo primero que tenemos que hacer es abrir una ventana de linea de comandos e irnos a la carpeta */dev*. Una vez hecho esto, vamos a ejecutar un script de z88dk que establece algunas variables de entorno. Recordad que lo instalamos en C:\z88dk10. Ejecutamos esto:

```
C:\z88dk10\setenv.bat
```

Una vez que hayamos hecho esto, podemos ejecutar nuestro script make.bat. Salga bien o salga mal, no cierres la ventana de linea de comandos, seguramente tendrás que volver a compilar mil veces más (sobre todo si estás ajustando los valores del movimiento o modificando el mapa para arreglar cosas).

TABLA DE SONIDOS

| n | Sonido | |
|---|---|---|
| 0 | Enemy destroyed | |
| 1 | Enemy hit | |
| 2 | Something | |
| 3 | Jump | |
| 4 | Player hit | |
| 5 | Enemy destroyed 2 | |
| 6 | Shot | |
| 7 | Item #1 | (item) |
| 8 | Item #2 | (key) |
| 9 | Item #3 | (life) |

**ANEXO I - Brainstorming**

Tengo que almacenar cláusulas y acciones. Lo primero que tengo que hacer, para intentar diseñar un formato sencillo, común y compacto, es ver cuántos tipos de cláusulas y cuántos tipos de acciones necesito, y qué tipos de cláusulas y acciones voy a tener. Esto es lo primero. Vayamos por partes...

El tema de las cláusulas es que cada script se compondrá por un montón de estas

comprobaciones, que podrán ser lanzadas de dos formas: al pulsar SPACE (acción) o al entrar en la pantalla.

Una cláusula, en pseudolenguaje, puede ser tal que así. Por ejemplo, en la pantalla 8 pulsamos FIRE al lado de un ordenador (en el tile 7, 4) para meter un diskette (ITEM 3). Esto deberá abrir una puerta en la pantalla 15. Para esto, usamos el FLAG 1

```
PRESS_FIRE AT SCREEN 8          ; Pulsamos ACCIÓN en la pantalla 8
    IF PLAYER_HAS_ITEM 3        ; Si tenemos el ITEM 3
    IF PLAYER_IN_X 6, 8         ; Posición del jugador:
    IF PLAYER_IN_Y 3, 5         ; Define un rectángulo centrado en el tile 7, 4
    THEN                        ; Si se cumple todo lo de arriba...
        SET ITEM 3, 0           ; Perdemos el ITEM 3
        SET FLAG 1, 1           ; Ponemos FLAG 1 a 1
    END                         ; Fin de las acciones
END                             ; Fin de las comprobaciones
```

En la pantalla 15, deberemos comprobar el estado del flag 1 para eliminar la puerta, que ocupa los tiles (8, 4) y (9, 4):

```
ENTERING SCREEN 15              ; Entramos en la pantalla 15
    IF FLAG 1 = 1               ; Si FLAG 1 vale 1
    THEN                        ; Si se cumple todo lo de arriba...
        SET TILE (8, 4) = 0     ;
        SET TILE (9, 4) = 0     ; Ponemos el tile 0 en lugar de la puerta
    END
END
```

Otra cosa importante: el compilador además generará el código del intérprete de forma que sólo genere las comprobaciones de las cláusulas y acciones utilizadas en el script, para ahorrar memoria.

Especiales:

| Sección | Se codifica como |
|---|---|
| ENTERING ANY | ENTERING #MAX_PANTS |
| ENTERING GAME | ENTERING #(MAX_PANTS+1) |
| ON_TIMER_OFF | ENTERING #(MAX_PANTS+2) |
| PRESS_FIRE AT ANY | FIRE #MAX_PANTS |
| PLAYER_GETS_COIN | FIRE #(MAX_PANTS+1) |
| PLAYER_KILLS_ENEMY | FIRE #(MAX_PANTS+2) |

Variables (FLAGS) La mayoría de los parámetros numéricos pueden ser sustituidos por el valor de un flag con #n significando "el valor del flag n"

**I.1. Tipos de cláusulas (IFs)**

* IF PLAYER_HAS_ITEM x
  Descripción: Evaluará a CIERTO si el jugador tiene el item X en su inventario. Opcode: 01 x
* IF PLAYER_HASN'T_ITEM x
  Descripción: Evaluará a CIERTO si el jugador NO tiene el item X en su inventario. Opcode: 02 x
* IF FLAG x = n
  Descripción: Evaluará a CIERTO si el flag "x" vale "n".
  Opcode: 10 x n
* IF FLAG x < n
  Descripción: Evaluará a CIERTO si el flag "x" < n.
  Opcode: 11 x n
* IF FLAG x > n
  Descripción: Evaluará a CIERTO si el flag "x" > n.
  Opcode: 12 x n

* IF FLAG x <> n
      Descripción: Evaluará a CIERTO si el flag "x" <> n.
      Opcode: 13 x n
* IF FLAG x = FLAG y
      Descripción: Evaluará a CIERTO si el flag "x" = flag "y".
      Opcode: 14 x y
* IF FLAG x < FLAG y
      Descripción: Evaluará a CIERTO si el flag "x" < flag "y".
      Opcode: 15 x y
* IF FLAG x > FLAG y
      Descripción: Evaluará a CIERTO si el flag "x" > flag "y".
      Opcode: 16 x y
* IF FLAG x <> FLAG y
      Descripción: Evaluará a CIERTO si el flag "x" <> flag "y".
      Opcode: 17 x y
* IF PLAYER_TOUCHES x, y
      Descripción: Evaluará a CIERTO si el jugador está tocando el tile (x, y).
      Opcode: 20 x y
* IF PLAYER_IN_X x1, x2
    Descripción: CIERTO si el jugador está horizontal entre los tiles x1, x2.
    Opcode: 21 x1 x2
* IF PLAYER_IN_Y y1, y2
    Descripción: CIERTO si el jugador está vertical entre los tiles y1, y2.
    Opcode: 22 y1 y2
* IF ALL_ENEMIES_DEAD
      Descripción: Evaluará a CIERTO si todos los enemigos están muertos.
      Opcode: 30
* IF ENEMIES_KILLED_EQUALS n
      Descripción: Evaluará a CIERTO si el número de enemigos eliminados es n.
      Opcode: 31 n
* IF PLAYER_HAS_OBJECTS
      Descripción: Evaluará a CIERTO si el jugador tiene objetos.
      Opcode: 40
* IF OBJECT_COUNT = n
      Descripción: Evaluará a CIERTO si el jugador tiene N objetos.
      Opcode: 41 n
* IF NPANT n
      Descripcion: Evaluará a CIERTO si el jugador está en la pantalla n.
      Opcode: 50 n
* IF NPANT_NOT n
      Descripcion: Evaluará a CIERTO si el jugador NO está en la pantalla n.
      Opcode: 51 n
* IF JUST_PUSHED
      Descripción: viene de acabar de empujar (!)
      Opcode: 60
* IF TIMER >= x
      Descripción: el timer >= x.
      Opcode: 70 x
* IF TIMER <= x
      Opcode: 71 x
* IF TRUE
      Descripción: Siempre CIERTO.
      Opcode: F0
* THEN
      Descripción: Termina las evaluaciones y comienza a ejecutar:
      Opcode: FF

**I.2. Tipos de acciones**

* SET ITEM x = n
      Da al estado del item X el valor N. 0 = no lo tienes, 1 = lo tienes.
      Opcode: 00 x n
* SET FLAG x = n
      Da el valor N al flag X.

Opcode: 01 x n
* INC FLAG x, n
Incrementa el valor del flag X en N.
Opcode: 10 x n
* DEC FLAG x, n
Decrementa el valor del flag X en N.
Opcode: 11 x n
* ADD FLAGS x, y
x = x + y.
Opcode: 12 x y
* SUB FLAGS x, y
x = x – y.
Opcode: 13 x y
* SET TILE (x, y) = n
Establece que el tile (x, y) de la pantalla sea el n.
Opcode: 20 x y n
* INC LIFE n
Incrementa el valor de la vida en n.
Opcode: 30 n
* DEC LIFE n
Decrementa el valor de la vida en n.
Opcode: 31 n
* FLICKER
El jugador pasa a estado flicker.
Opcode: 32
* INC OBJECTS n
Añade n objetos más.
Opcode: 40 n
* DEC OBJECTS n
Resta n objetos (si objects >= n; si no objects = 0)
Opcode: 41 n
* SWAP x, y
Intercambia el valor de los flags x e y
Opcode: 14 x y
* PRINT_TILE_AT (x, y) = n
Opcode: 50 x y n
* SET_FIRE_ZONE x1, y1, x2, y2
Opcode: 51 x1y1x2y2
* SHOW_COINS
Opcode: 60
* HIDE_COINS
Opcode: 61
* ENABLE_KILL_SLOWLY
Opcode: 62
* DISABLE_KILL_SLOWLY
Opcode: 63
* ENABLE_TYPE_6
Opcode: 64
* DISABLE_TYPE_6
Opcode: 65
* ENABLE_MAKE_TYPE_6
Opcode: 66
* DISABLE_MAKE_TYPE_6
Opcode: 67
* REENTER
Re-entra en la pantalla (con todo lo que ello significa).
Opcode: 6F
* REDRAW
Sólo redibuja la pantalla.
Opcode: 6E
* WARP_TO n, x, y
Salta a la pantalla n en x, y.
Opcode: 6D n x y
* SET_TIMER a, b

```
      Timer = a, rate = b.
      Opcode: 70 a b
* TIMER_START
      Enciende el temporizador.
      Opcode: 71
* TIMER_STOP
      Apaga el temporizador.
      Opcode: 72
* SOUND n
      Peta_el_beeper (n);
      Opcode: E0 n
* SHOW
      Actualiza la pantalla.
      Opcode: E1
* RECHARGE
      Recarga toda la vida.
      Opcode: E2
* TEXT
      Imprime un texto.
      Opcode:     E3 chars255
* EXTERN n
      Ejecuta una función "extern" pasándole como parámetro N.
      Opcode: E4 n
* GAME OVER
      Pierde el juego.
      Opcode: F0
* WIN GAME
      Gana el juego.
      Opcode: F1
* END
      Termina el bloque.
      Opcode: FF
```

**ANEXO II. IDEAS COMPILACIÓN**

El código compilado:

```
[code]defb 18               ; Este bloque ocupa 18 bytes
defb 0x01                   ; Es del tipo "Press fire"
defb 0x01, 0x03             ; IF PLAYER_HAS_ITEM 3
defb 0x21, 0x06, 0x08       ; IF PLAYER_IN_X 6, 8
defb 0x22, 0x03, 0x05       ; IF PLAYER_IN_Y 3, 5
defb 0xFF                   ; THEN
defb 0x00, 0x03, 0x00       ; SET ITEM 3, 0
defb 0x01, 0x01, 0x01       ; SET FLAG 1, 1
defb 0xFF                   ; END[/code]
```

El intérprete generado:

```
[code]
// Comprobaciones

continue = 0;
terminado = 0;
while (!terminado) {
    c = read_byte ();
    switch (c) {
        case 0x01:
            // IF PLAYER_HAS_ITEM x
            // Opcode: 01 x
            x = read_byte ();
            if (items [x].status == 0)
                terminado = 1;
```

```
            break;
        case 0x02:
            // IF PLAYER_HASN'T_ITEM x
            // Opcode: 02 x
            x = read_byte ();
            if (items [x].status == 1)
                terminado = 1;
            break;
        case 0x10:
            // IF FLAG x = n
            // Opcode: 10 x n
            x = read_byte ();
            n = read_byte ();
            if (flags [x] != n)
                terminado = 1;
            break;
        // etcétera
        case 0xFF:
            terminado = 1;
            continue = 1;
    }

if (continue) {
    // Acciones
    terminado = 0;
    while (!terminado) {
        c = read_byte ();
        switch (c) {
            case 0x00:
                // SET ITEM x n
                // Opcode: 00 x n
                x = read_byte ();
                n = read_byte ();
                items [x] = n;
                break;
            case 0x10:
                // INC FLAG x, n
                // Opcode: 10 x n
                x = read_byte ();
                n = read_byte ();
                flags [x] += n;
                break;
            // etcétera
            case 0xFF:
                terminado = 1;
        }
    }
}[/code]
```

**CARACTERÍSTICAS DISPONIBLES (MK2 0.89)**

- Vista lateral o a vista de pájaro.
- Pantalla de título/marco de juego, y pantalla final.
- Cambio de sprites de bala y explosión.
- Hasta 3 enemigos en pantalla de un total de 4.
- Enemigo tipo 1; Lineales. Pueden disparar.
- Enemigo tipo 2; Voladores.
- Enemigo tipo 3; Perseguidores.
- Enemigo tipo 8; Plataformas móviles.
- Enemigo tipo 9; Gotas verticales.
- Enemigo tipo 10; Flechas horizontales.
- Los enemigos pueden activarse/desactivarse y cambiar su tipo.
- 1 Hotspot por pantalla.
- Hotspot 1; Objeto. Un solo objeto a la vez, o recogida de múltiples objetos.
- Hotspot 2; Llave. Requiere otro tile de tipo cerrojo. (10)
- Hotspot 4; Recarga de munición.
- Hotspot 5; Recarga de temporizadores.
- Al retirar un Hotspot puede aparecer una recarga de vida.
- Fin de juego al conseguir un determinado número de objetos.
- Mensaje automático de “GET X MORE” al coger un objeto.
- Colisiones de 8x8 centrada, o 16x16.
- Tiles empujables. Pueden detener a los enemigos (tile 14 de tipo 10)
- Tiles de tipo 1 matan. (lava, pinchos, etc.)
- El player puede rebotar al tocar enemigos.
- El player puede parpadear temporalmente al tocar enemigos. (invulnerable)
- El player puede disparar. Lateral, arriba, diagonal.
- Los enemigos pueden resucitar al entrar el player de nuevo en pantalla.
- El player puede saltar en vista lateral. La potencia de salto es acumulativa.
- El player puede usar un JetPac.
- El player puede saltar sobre el enemigo y matarlo.
- El player puede rebotar contra los obstáculos.
- Tiles destructibles (tipo 16)
- Tiles animados. (2 tiles que se intercambian)
- Temporizadores. (99 a 0)
- Dos combinaciones de teclas: QAOP-ESPACIO ; WASD-NM. H = pausa. Y = Abortar.
- Posibilidad de mostrar textos largos en la zona de juego, y mensajes fuera.
- Inventario de objetos.
- Objetos arrojables contra los enemigos.
- Se pueden poner nombre a las pantallas y mostrarlos.
- Se pueden hacer juegos de pantalla única, y solo pasar en determinados casos.
- En juegos de pantalla única, se puede salir por un extremo y aparecer en el otro.
- HITTERS! El player puede golpear con puño, espada y látigo.
- Checkpoints para recomenzar desde estos puntos al morir.
- Melodía inicial y sonidos eventuales preseleccionados.