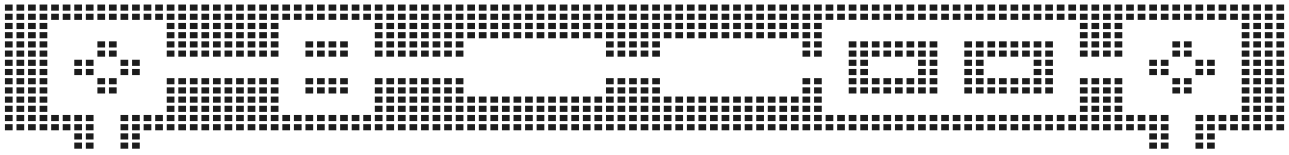

Técnicas BM 4.0

Por Marcos Navarro *Publicación I (17/12/2012)*

La presente publicación está dirigida a los programadores de calculadoras hp50g, principalmente a los que utilizan System RPL, para compartir con ellos experiencias e ideas que me han surgido durante el desarrollo de este proyecto que es BM 4.0.

CONTENIDO

---

1. Motores de Juego

El módulo principal de BME se llama M1, y es básicamente un bucle tipo BEGIN... UNTIL. Un aspecto importante de este módulo es que no se detiene a esperar que el jugador presione alguna tecla, sino que se mantiene ejecutándose continuamente. Luego, cuando el jugador presiona una tecla el bucle cambia su flujo de ejecución.

Este aspecto es fundamental para el motor de juego, ya que es lo que permite que sea dinámico, que el reloj se mantenga corriendo y que los otros personajes se muevan aun cuando el jugador está inmóvil.

La clave para lograrlo está en la instrucción GETTOUCH. Esta instrucción del lenguaje System RPL hace lo siguiente:

- Si el usuario presiona una tecla devuelve el número de dicha tecla y un TRUE.
- Si el usuario no presiona una tecla devuelve un FALSE.

Veamos algunos ejemplos:

1) El siguiente programa ejecuta PROG1 continuamente, pero cuando se presiona una tecla ejecuta PROG2 y sale del bucle:

```
::
        BEGIN
                GETTOUCH
                ITE     :: ID PROG1 FALSE ;
                        :: ID PROG2 TRUE ;
        UNTIL
;
@
```

2) El siguiente programa muestra un contador en la pantalla, que se va incrementando hasta que se presiona alguna tecla:

```
::
        #0                                      (Valor inicial del contador)

        BEGIN
                GETTOUCH
                #1+  DUP                        (Actualizando el contador)
                #>$  DISPROW1                   (Mostrando en la pantalla)
        UNTIL
;
@
```

3) El siguiente programa muestra un contador en la pantalla, que se va incrementando hasta que se presiona la tecla **B**:

```
::
        #0                                      (Valor inicial del contador)

        BEGIN
                GETTOUCH
                ITE     :: #2 #= ;
                        :: #1+  DUP #>$  DISPROW1 FALSE ;
        UNTIL
;
@
```

2. Cantidad Dinámica de Variables.

Para su funcionamiento, BM crea una serie de variables lambda, en las cuales se almacena todos los datos del juego: coordenadas, gráficos, listas, contadores, etc.

Las primeras 281 variables lambda de BM están presentes en todos sus juego, pero la cantidad adicional de variables va a depender del número de componentes de cada juego en particular. Digámoslo así: hay una cantidad rígida de 281 variables y una cantidad flexible que depende del número de componentes del juego que se va a cargar.

Es fácil para BM saber la cantidad total de variables que va a necesitar, ya que cada tipo de componente utiliza un número predeterminada de ellas:

- Los componentes MB requieren 16 variables.
- Los componentes PB requieren 16 variables.
- Los componentes OB requieren 16 variables.
- Los componentes AB requieres 5 variables.
- Los componentes CE requieren 32 variables.

Y existe 5 variables del espacio rígido donde se guarda las cantidades de componentes de un juego (ZxPack). Estas variables son:

- MBNum: contiene el número de MB de un juego.
- PBNum: contiene el número de PB de un juego.
- OBNum: contiene el número de OB de un juego.
- ABNum: contiene el número de AB de un juego.
- CENum: contiene el número de CE de un juego.

Luego para calcular el total de variables basta hacer el siguiente cálculo:

Nvars = 281+(MBnum * 16)+(PBNum * 16)+(OBNum * 16)+(ABNum * 5)+(CENum * 32)

O también:

Nvars = 281+[16 *(MBnum+PBNum+OBNum)]+(ABNum * 5)+(CENum * 32)

Finalmente para crear las variables:

#0 Nvars NDUPN NULLLAM SWAP NDUPN DOBIND

3. Aprovechando los Espacios Gráficos.

Como sabemos, la hp50g dispone de tres espacios gráficos:

El grob de texto (ABUFF)

El grob de gráficos (GBUFF)

El grob de menú (HARDBUFF2)

BME aprovecha los tres espacios:

En ABUFF coloca el mapa del juego.

En GBUFF mantiene el entorno gráficos donde se desarrolla el juego.

En HARDBUFF2 presenta información de la partida al jugador.

Lo fundamental aquí es que en BM cada píxel activado del ABUFF representa un componente del GBUFF. Para saber si existe algún componente en alguna coordenada basta utilizar la instrucción PIXON?

4. Retardo GC.

Para crear librarías más compactas, que ahorren memoria y se transfieran más rápido, se puede optar por comprimir parte de los archivos que la componen. El compresor BZ2 es en estos momentos la mejor opción para la hp50g. Ha de incluirse dentro de la librería el descompresor Ubz2 para poder descomprimir los archivos al momento de necesitarlos.

Pero en una ocasión se me presento un problema algo extraño al usar esta técnica. Sucede que mi programa descomprimía el archivo, que era una lista enorme, y guardaba sus componentes como variables locales. Luego ejecutaba un bucle que cada cierto tiempo realizaba una recogida de basura (Garbage Collection). El problema era que la recogida de basura tardaba muchísimo, y hacia el programa inservible.

He nombrado a este problema como “retardo GC”.

La **solución** es muy simple: “guardar el archivo como variable local después de descomprimirlo”. El módulo ZxLoader de BM hace esto para evitar el retardo GC.

Muchas gracias a Yann por su ayuda con este caso.