Game Utils para Amstrad PCW

Versión 1.0

19 diciembre 2014

Esta librería consta de cuatro clases orientadas a la programación de juegos.

Clase GameSystem

Contiene métodos diversos para gestionar el sistema. Los métodos son estáticos y deben ser llamados mediante la clase, por ejemplo *GameUtils.SetStack(#8000)*.

Métodos

static void SetStack(int address)

Establece el puntero de la pila al valor *address* y almacena el valor actual del puntero de pila. Cuando los métodos declaran una gran cantidad de variables locales no estáticas conviene establecer el puntero de pila en una zona de memoria libre más amplia que la que usa por defecto el CPC, por ejemplo #A6FC, si el programa se encuentra en una dirección baja de memoria.

static void RestoreStack()

Restaura el puntero de pila al valor almacenado en el último uso de *SetStack*. El uso de *RestoreStack* debe estar en el mismo bloque de código que el *SetStack* previo que debe restaurar.

static void DisableInterruptions()

Desactiva las interrupciones generadas en el modo 1 del Z80 y almacena la dirección actual de las interrupciones.

static void EnableInterruptions()

Activa las interrupciones generadas en el modo 1 del Z80 restaurando la dirección almacenada en el último uso de *DisableInterruptions*. El uso de *EnableInterruptions* debe estar en el mismo bloque de código que el *DisableInterruptions* previo que debe restaurar.

static void SetInterruptions(int address)

Establece como rutina de interrupciones del modo 1 del Z80 la dirección dada como parámetro, almacenando previamente el valor actual de la dirección de interrupciones. Se puede indicar como

dirección la de una función del programa mediante el operador & precediendo al nombre de la función, con los paréntesis tras él.

static void RestoreInterruptions()

Restaura la dirección de interrupciones del modo 1 del Z80 almacenada en el último uso de *SetInterruptions*. El uso de *RestoreInterruptions* debe estar en el mismo bloque de código que el *SetInterruptions* previo que debe restaurar.

Clase GameKeyboard

Contiene constantes y métodos para la lectura del teclado que mejoran la velocidad respecto a las existentes en el BIOS. Las constantes son públicas y pueden ser usadas precedidas del nombre de la clase, por ejemplo *GameKeyboard.keyA*. Los métodos son estáticos y deben ser llamados mediante la clase, por ejemplo *GameKeyboard.ReadKeys()*.

Constantes

Las constantes representan las teclas del Amstrad CPC. Consultar el fichero fuente de la clase *GameKeyboard.ccz80++* para conocer todas las constantes.

Métodos

static void ReadKeys()

Realiza una lectura del teclado y almacena el estado de todas las teclas.

static int GetKeys()

Genera una lista de las teclas pulsadas detectadas en el último uso de *ReadKeys*. Devuelve la dirección de inicio de la lista generada. La lista está formada por valores de tamaño byte y tras el último elemento se encuentra el valor -1. Si no se ha encontrado ninguna tecla pulsada el valor -1 se encuentra al principio de la lista.

static short TestKey(short number)

Comprueba si la tecla indicada como parámetro se encontraba pulsada en el último uso de *ReadKeys*. Devuelve 0 si no estaba pulsada o 1 si lo estaba.

Clase GameScreen

Contiene métodos para la gestión de la pantalla, en parte mejorando las rutinas del BIOS.

Las coordenadas de pantalla tienen como origen la esquina superior izquierda y los valores siguientes: horizontal de 0 a 719, vertical de 0 a 255.

Las imágenes utilizadas pueden generarse con el programa *SpriteEditorPCW* y especificarse en el programa en una tabla unidimensional de tipo short, pasando su dirección a los métodos que requieran una imagen mediante el operador & precediendo al nombre de la tabla.

Para que los métodos de esta clase trabajen sin problema no se deben utilizar la memoria desde la dirección #8000 en adelante, y establecer la pila en la dirección #8000 o inferior con el método *GameSystem.SetStack*.

Métodos

static void ActivateMapping()

Activa la memoria de pantalla, bancos 1 y 2, en la zona alta de memoria. Así el acceso a la memoria de pantalla para lectura y escritura se puede hacer desde la dirección #9930 a #C32F, a la roller RAM desde #C600 a #C7FF, y al juego de caracteres desde #C800 hasta #FFFF.

static void DeactivateMapping()

Desactiva la memoria de pantalla de la zona alta de memoria restaurando los bancos 6 y 7 en esa zona.

static void PutImage(int x, short y, int image)

Dibuja en las coordenadas *x*, *y* la imagen cuya dirección se da como *image*.

static short TestImage(int x, short y, int image)

Comprueba que en las coordenadas *x*, *y* se encuentra dibujada la imagen cuya dirección se da como *image* y no se superpone con ningún otro elemento de la pantalla.

static void Clear()

Borra la pantalla con tinta 0.

static void WaitFlyback()

Espera hasta el siguiente flyback de pantalla para hacer una pausa y mejorar el movimiento de los sprites.

Clase GameSprite

Representa un sprite en la pantalla. Dispone de miembros públicos que definen la posición, imagen, parámetros y comportamiento del sprite. La clase también proporciona constantes para gestionar los miembros públicos mencionados y el valor de retorno de los métodos.

Respecto a las coordenadas de pantalla, las imágenes y el uso de la memoria hay que tener en cuenta lo indicado para la clase *GameScreen*.

Constantes

Para los miembros actionLimitUp, actionLimitDown, actionLimitLeft y actionLimitRight:

actionNone: indica que el sprite no hará nada al llegar al límite.

actionStop: indica que sprite parará al llegar al límite.

actionRebound: indica que el sprite rebotará al llegar al límite.

actionMove: indica que el sprite aparecerá por el lado contrario al llegar al límite.

actionRemove: indica que el sprite desaparecerá y se desactivará al llegar al límite.

Para el valor de los miembros *keyUp*, *keyDown*, *keyLeft* y *keyRight*:

keyNothing: representa que la tecla asignada con este valor no es utilizable por el jugador.

Para el miembro *motionType*:

continuousNo: indica movimiento no continuo, es decir, que el sprite sólo se mueve al detectar pulsación de una tecla de movimiento.

continuousYes: indica movimiento continuo, es decir, que el sprite se mueve continuamente en dirección según la última pulsación de una tecla de movimiento hasta que se pulse otra tecla de movimiento diferente.

Para el valor de retorno del método *IsActive*:

activeNo: sprite no activo.

activeYes: sprite activo.

Para el valor de retorno de los métodos *UpdateAuto* y *UpdateByKeys*:

outUp: indica que el sprite ha llegado al límite superior.

outDown: indica que el sprite ha llegado al límite inferior.

outLeft: indica que el sprite ha llegado al límite izquierdo.

outRight: indica que el sprite ha llegado al límite derecho.

Para el valor de retorno del método *IsCollision* sin argumentos:

collisionNo: indica que el sprite no colisiona.

collisionYes: indica que el sprite colisiona.

Miembros

x: posición horizontal del sprite en pantalla.

y: posición vertical del sprite en pantalla.

image: dirección de la tabla unidimensional de tipo short que contiene los valores para la imagen del sprite.

xIncrement: incremento horizontal positivo o negativo para el movimiento del sprite, inicialmente con valor 0.

yIncrement: incremento vertical positivo o negativo para el movimiento del sprite, inicialmente con valor 0.

xSpeedKeys: incremento para el movimiento del sprite cuando se pulsa la tecla de movimiento izquierda o derecha, inicialmente con valor 0.

ySpeedKeys: incremento para el movimiento del sprite cuando se pulsa la tecla de movimiento arriba o abajo, inicialmente con valor 0.

limitUp: coordenada vertical del límite superior para el sprite, inicialmente 0.

limitDown: coordenada vertical del límite inferior para el sprite, inicialmente igual a 200 menos el número de puntos verticales de la imagen del sprite.

limitLeft: coordenada horizontal del límite izquierdo para el sprite, inicialmente 0.

limitRight: coordenada horizontal del límite derecho para el sprite, inicialmente igual a la anchura de la pantalla según el modo seleccionado menos el número de puntos horizontales de la imagen del sprite.

actionLimitUp: tipo de acción cuando el sprite alcanza el límite superior, inicialmente igual a actionNone.

actionLimitDown: tipo de acción cuando el sprite alcanza el límite inferior, inicialmente igual a actionNone.

actionLimitLeft: tipo de acción cuando el sprite alcanza el límite izquierdo, inicialmente igual a actionNone.

actionLimitRight: tipo de acción cuando el sprite alcanza el límite derecho, inicialmente igual a actionNone.

keyUp: número de la tecla que se detectará para mover el sprite hacia arriba, inicialmente igual a keyNothing.

keyDown: número de la tecla que se detectará para mover el sprite hacia abajo, inicialmente igual a keyNothing.

keyLeft: número de la tecla que se detectará para mover el sprite hacia la izquierda, inicialmente igual a keyNothing.

keyRight: número de la tecla que se detectará para mover el sprite hacia la derecha, inicialmente igual a keyNothing.

motionType: tipo de movimiento para el control del sprite mediante el teclado.

Métodos

short IsActive()

Devuelve si el sprite está activo o no.

void Create(int xInitial, short yInitial, int imageInitial)

Activa el sprite y lo dibuja en pantalla en las coordenadas *xInitial*, y*Initial* con la imagen *imageInitial*.

void Remove()

Desactiva el sprite y lo borra de pantalla.

void Update()

Mueve el sprite en pantalla a la posición y con la imagen de los valores actuales de los miembros *x*, *y* e *image*. Estos valores deben haber sido modificados por el programa.

short UpdateAuto()

Modifica los miembros x e y según los miembros *xIncrement* e *yIncrement* y mueve el sprite en pantalla a la posición y con la imagen de los nuevos valores *x*, *y* e *image*.

Devuelve un valor que indica si el sprite se ha salido de sus límites. Si esto ha sucedido se habrá realizado la acción establecida y el sprite en ningún caso se dibujará fuera de los límites. Si se ha producido la salida en más de un límite el valor devuelto será la suma de los indicadores de salida de pantalla de todos límites. Por ejemplo si se ha llegado al límite superior e izquierdo, el valor devuelto será *outUp* + *outLeft*.

short UpdateByKeys()

Lee el teclado y comprueba las teclas establecidas para los miembros *keyUp*, *keyDown*, *keyLeft*, *keyRight* y según las que estén pulsadas mueve el sprite en pantalla. Si el modo de movimiento especificado en el miembro *motionType* es continuo el sprite se mueve en pantalla aunque no se detecte ninguna pulsación de las teclas.

Devuelve un valor que indica si el sprite se ha salido de sus límites. Si esto ha sucedido se habrá realizado la acción establecida y el sprite en ningún caso se dibujará fuera de los límites.

void Back()

Mueve el sprite a la posición previa tras el uso de *UpdateAuto* o *UpdateByKeys*. Puede ser útil utilizar este método en caso de detectar una colisión.

short IsCollision()

Devuelve si el sprite ha colisionado según su situación actual en pantalla. La colisión se detecta comprobando que no hay ningún objeto en pantalla superpuesto con el sprite.

short IsCollision(GameSprite sprite)

Devuelve si el sprite ha colisionado con el sprite recibido como parámetro según sus situaciones actuales en pantalla. Esta comprobación se hace considerando los sprites como rectángulos, por lo que puede indicarse colisión aunque sólo se superpongan zonas de los sprite que estén en blanco.