Game Utils for Amstrad PCW

Version 1.0

19 December 2014

This library consist of four classes oriented to the programming of games.

Class GameSystem

Contains diverse methods for manage the system. The methods are static and have to be called by means of the class, for example *GameUtils.SetStack(#8000)*.

Methods

static void SetStack(int address)

Establishes the stack pointer to the value *address* and stores the current value of the stack pointer. When the methods declare a big quantity of local variables no static suits to establish the stack pointer in a free zone of memory wider that the one who uses by defect the CPC, for example #A6FC, if the program is in a low direction of memory.

static void RestoreStack()

Restores the stack pointer to the value stored in the last use of *SetStack*. The use of *RestoreStack* has to be in the same block of code that the *SetStack* previous that has to restore.

static void DisableInterruptions()

Switch off the Z80 mode 1 interruptions and stores the current direction of the interruptions.

static void EnableInterruptions()

Switch on the Z80 mode 1 interruptions restoring the direction stored in the last use of *DisableInterruptions*. The use of *EnableInterruptions* has to be in the same block of code that the *DisableInterruptions* previous that has to restore.

static void SetInterruptions(int address)

Establishes as routine of Z80 mode 1 interruptions the address given like parameter, storing previously the current value of the direction of interruptions. Can indicate like direction the one of

a function of the program with the operator & preceding to the name of the function, with the parenthesis after him.

static void RestoreInterruptions()

Restores the direction of Z80 mode 1 interruptions stored in the last use of *SetInterruptions*. The use of *RestoreInterruptions* has to be in the same block of code that the *SetInterruptions* previous that has to restore.

Class GameKeyboard

Contains constant and methods for the reading of the keyboard that improve the speed with regard to the existent in the BIOS. The constants are public and can be used preceded of the name of the class, for example *GameKeyboard.keyA*. The methods are static and have to be called by means of the class, for example *GameKeyboard.ReadKeys()*.

Constant

The constants represent the keys of the Amstrad CPC. Consult the file source of the class *GameKeyboard.ccz80++* to know all the constants.

Methods

static void ReadKeys()

Realises a reading of the keyboard and stores the state of all the keys.

static int GetKeys()

Generates a list of the keys pressed detected in the last use of *ReadKeys*. It returns the address of start of the list generated. The list is formed by values of size byte and after the last element is the value -1. If it has not found any key pressed the value -1 is at the beginning of the list.

static short TestKey(short number)

Checks if the key indicated as parameter was pressed in the last use of *ReadKeys*. It returns 0 if it was not pressed or 1 if it was it.

Class GameScreen

Contains methods for the management of the screen, partly improving the routines of the BIOS.

The coordinates of screen have as origin the left upper corner and the following values: horizontal from 0 to 719, vertical from 0 to 255.

The images used can generate with the program *SpriteEditorPCW* and specify in the program in a one-dimensional table of type short, passing his direction to the methods that require an image by means of the operator & preceding to the name of the table.

For avoid problem should not be used memory from address #8000 to up, and set the stack address in #8000 or less with the method *GameSystem.SetStack*.

Methods

static void ActivateMapping()

Enables the screen memory, banks 1 and 2, in the upper memory area. Thus the access to screen memory for reading and writing can be done from the address #9930 to #C32F, the roller RAM from #C600 to #C7FF, and the character set from #C800 to #FFFF.

static void DeactivateMapping()

Disables the screen memory from upper memory area restoring the banks 6 and 7 in this area.

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

Draws in the coordinates *x*, *y* the image whose address give like *image*.

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

Check that in the coordinates *x*, *y* finds drawn the image whose direction give like *image* and does not superimpose with any another element of the screen.

static void Clear()

Erases the screen with ink 0.

static void WaitFlyback()

Expects until the following flyback of screen to do a pause and improve the movement of the sprites.

Class GameSprite

Represents a sprite in the screen. It has public members that define the position, image, parameters and behaviour of the sprite. The class also provides constant for manage the public members mentioned and the value of return of the methods.

With regard to the coordinates of screen, the images and the use of memory it is necessary to take into account the indicated for the class *GameScreen*.

Constant

For the members actionLimitUp, actionLimitDown, actionLimitLeft and actionLimitRight:

actionNone: indicate that the sprite will not do at all when arriving to the limit.

actionStop: indicate that sprite will stop when arriving to the limit.

actionRebound: indicate that the sprite will rebound when arriving to the limit.

actionMove: indicate that the sprite appears by the contrary side when arriving to the limit.

actionRemove: indicate that the sprite disappears and will b e disable when arriving to the limit.

For the value of the members *keyUp*, *keyDown*, *keyLeft* and *keyRight*:

keyNothing: represent that the key assigned with this value is not usable by the player.

For the member *motionType*:

continuousNo: it indicates movement no continuous, is to say, that the sprite only move when detecting press of a key of movement.

continuousYes: it indicates continuous movement, is to say, that the sprite move continuously in direction as the last press of a key of movement until it press another key of different movement.

For the value of return of the method *IsActive*:

activeNo: sprite no active.

activeYes: sprite active.

For the value of return of the methods *UpdateAuto* and *UpdateByKeys*:

outUp: indicate that the sprite has arrived to the upper limit.

outDown: indicate that the sprite has arrived to the lower limit.

outLeft: indicate that the sprite has arrived to the left limit.

outRight: indicate that the sprite has arrived to the right limit.

For the value of return of the method *IsCollision* without arguments:

collisionNo: indicate that the sprite no collide.

collisionYes: indicate that the sprite collide.

Members

x: horizontal position of the sprite in screen.

y: vertical position of the sprite in screen.

image: addres of the one-dimensional table of type short that contains the values for the image of the sprite.

xIncrement: horizontal increase positive or negative for the movement of the sprite, initially with value 0.

yIncrement: vertical increase positive or negative for the movement of the sprite, initially with value 0.

xSpeedKeys: increase for the movement of the sprite when is presses the key of left or right movement, initially with value 0.

ySpeedKeys: increase for the movement of the sprite when is presses the key of movement up or down, initially with value 0.

limitUp: vertical coordinate of the upper limit for the sprite, initially 0.

limitDown: Vertical coordinate of the lower limit for the sprite, initially equal to 200 minus the number of vertical points of the image of the sprite.

limitLeft: horizontal coordinate of the left limit for the sprite, initially 0.

limitRight: horizontal coordinate of the right limit for the sprite, initially equal to the width of the screen as the mode selected minus the number of horizontal points of the image of the sprite.

actionLimitUp: type of action when the sprite reaches the upper limit, initially equal to actionNone.

actionLimitDown: type of action when the sprite reaches the lower limit, initially equal to actionNone.

actionLimitLeft: type of action when the sprite reaches the left limit, initially equal to actionNone.

actionLimitRight: type of action when the sprite reaches the right limit, initially equal to actionNone.

keyUp: number of the key that detect to move the sprite upwards, initially equal to keyNothing.

keyDown: number of the key that detect to move the sprite downwards, initially equal to keyNothing.

keyLeft: number of the key that detect to move the sprite to the left, initially equal to keyNothing.

keyRight: number of the key that detect to move the sprite to the right, initially equal to keyNothing.

motionType: type of movement for the control of the sprite via keyboard.

Methods

short IsActive()

Returns if the sprite is active or not.

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

Activates the sprite and draws it in screen in the coordinates *xInitial*, *yInitial* with the image *imageInitial*.

void Remove()

Deactivates the sprite and erases it of screen.

void Update()

Moves the sprite in screen to the position and with the image of the current values of the members *x*, *y* and *image*. These values have to have been modified by the program.

short UpdateAuto()

Modifies the members x and *y* by the members *xIncrement* and *yIncrement* and moves the sprite in screen to the position and with the image of the new values *x*, *y* and *image*.

Returns a value that indicates if the sprite has gone out of his limits. If this has happened will have realised the action established and the sprite in any case will draw out of the limits. If it has produced the exit in more than a limit the value returned will be the sum of the indicators of exit of screen of all limits. For example if it has arrived to the upper and left limit, the value given back will be *outUp* + *outLeft*.

short UpdateByKeys()

Reads the keyboard and checks the keys established for the members *keyUp*, *keyDown*, *keyLeft*, *keyRight* and as those that are pressed moves the sprite in screen. If the type of movement specified in the member *motionType* is continuous the sprite move in screen although it do not detect any press of the keys.

It returns a value that indicates if the sprite has gone out of his limits. If this has happened will have realised the action established and the sprite in any case will draw out of the limits.

void Back()

Moves the sprite to the previous position after the use of *UpdateAuto* or *UpdateByKeys*. It can be useful use this method in case to detect a collision.

short IsCollision()

Returns if the sprite has collided as his current situation in screen. The collision detects checking that there is not any object in screen superimposed with the sprite.

short IsCollision(GameSprite sprite)

Returns if the sprite has collided with the sprite received as parameter as their current situations in screen. This test does considering the sprites like rectangles, by what can indicate collision although only superimpose blank areas of the sprite.