Sprite Byte Tutorials Lesson Five: User-Controlled Sprite

by Alyce Watson http://alycesrestaurant.com/
-
Alyce

Table of Contents

In Sprite Byte Tutorials Lesson Four: Moving A Sprite we discussed several ways we can cause a sprite to be moved using code. If we want to create interactive games, we must allow a game's user to control a sprite, too.

Two Ways for Users to Control a Sprite

There are several ways in which a user can interact with a computer game. We'll discuss the two easiest ways. He can press a key on the keyboard, or move the mouse. Both of these events can be trapped in windows of type "graphics" or in a graphicbox. They are trapped with the **when** statement, which looks like this:

```
print #handle, "when event eventHandler"
or
#handle "when event eventHandler"
```

We must make note of two important points. Before events can be trapped, the graphicbox or graphics window must receive a **Setfocus** command, like this:

```
#handle "setfocus"
```

If the **Setfocus** command has not been issued, mouse and keyboard events are not trapped with **When**.
It's also important to note that in versions of Liberty BASIC including the current version 4.03, the name of the event is case sensitive. "MOUSEMOVE" is not the same as "mouseMove" for instance. Always refer to the helpfile to learn the proper case for events trapped with **When**.
The **eventHandler** can be any valid sub or branch label.
After the **When** statement is issued along with a **Setfocus** statement, the graphicbox or graphics window causes the specified event handler to be invoked when the user causes the event. For particulars, see below.

Keyboard Control

It's very easy to trap key presses in a graphicbox or graphics window. These two lines are all that is needed:

```
 #w.g "setfocus" 'MUST setfocus to graphicbox
 #w.g "when characterInput [update]"
```

We can even give both commands in a single statement separated by semi-colons, like this:

```
 #w.g "setfocus; when characterInput [update]"
```

The "event" to trap is **characterInput** and it is case sensitive. Any time a user presses a key, the program goes to the specified event handler. We've used a branch label called "[update]".

Inkey$

When a keyboard event is triggered, the value of the key pressed is placed in the special Liberty BASIC variable called **Inkey$**. Remember, variable names are case sensitive. If the user presses the "g" key, Inkey$ will contain "g".
We'll want to use the arrow keys to allow the user to move a sprite in a game. The arrow keys do not have a character representation. Liberty BASIC places two values in Inkey$ when an arrow key is pressed.
Here's a brief explanation from the helpfile topic, "Using virtual key constants with Inkey$"
Special keys like Alt, Ctrl, Shift, the Arrow keys, etc. are not coded like the letters, numbers and other symbols. They have special values, and are preceded by a value of 32 (or less) when they are trapped by Inkey$.

We check the rightmost character contained in Inkey$ to see if an arrow key is pressed. We check the ASCII value of that character with the **ASC()** function:

```
asc(right$(Inkey$,1))
```

The ASCII values of the arrow keys are as follows:

- 37 = left arrow
- 38 = up arrow
- 39 = right arrow
- 40 = down arrow

Here's how we'll check to see if an arrow key was pressed and determine which one it was:

```
 if asc(right$(Inkey$,1))=39 then print "Right Arrow"
 if asc(right$(Inkey$,1))=37 then print "Left Arrow"
 if asc(right$(Inkey$,1))=38 then print "Up Arrow"
 if asc(right$(Inkey$,1))=40 then print "Down Arrow"
```

In a real program, we'd increment (increase the value of) the sprite's X location if the right arrow were pressed. We'd decrement (decrease the value of) the sprite's X location if the left arrow were pressed. We'd increment (increase the value of) the sprite's Y location if the down arrow were pressed. We'd decrement (decrease the value of) the sprite's Y location if the up arrow were pressed.
Here's some example code:

```
 'right arrow = X gets bigger:
 if asc(right$(Inkey$,1))=39 then spriteX = spriteX + 3
 'left arrow = X gets smaller:
 if asc(right$(Inkey$,1))=37 then spriteX = spriteX - 3
 'up arrow = Y gets smaller:
 if asc(right$(Inkey$,1))=38 then spriteY = spriteY - 3
 'down arrow = Y gets bigger:
 if asc(right$(Inkey$,1))=40 then spriteY = spriteY + 3
 'locate sprite at new position and update display
 #w.g "spritexy guy ";spriteX;" ";spriteY
 #w.g "drawsprites"
```

Notice that we **must** issue a "spritexy" command to change the location of the sprite and we **must** issue a "drawsprites" command to update the display.
See the first demo program at the end of this lesson.

User Events and Other Events

It is sometimes necessary to allow other events to happen during a game that allows the user to move a sprite. For instance, a display of the time elapsed might need to be updated, or a computer controlled sprite might need to move, regardless of whether the user is moving his sprite. The second demo program at the end of this lesson includes a statictext that shows the number of seconds the game has been running and it also allows the user to move a sprite. A timer is used to update the display at regular intervals.

```
 timer 300, [updateDisplay]
```

When the timer fires, this code is run:

```
 'display time elapsed since start of program
 #w.s "Elapsed time: ";time$("seconds") - startTime
```

Notice that only the last portion of the code below is activated when the timer fires([updateDisplay]), but the entire routine is activated ([updatePosition]) when the user presses a key. There are many ways to handle program flow, depending upon the needs of a particular game. This is just one way. Can you think of others?

```
[updatePosition]
 'right arrow = X gets bigger:
 if asc(right$(Inkey$,1))=39 then spriteX = spriteX + 3
 'left arrow = X gets smaller:
 if asc(right$(Inkey$,1))=37 then spriteX = spriteX - 3
 'up arrow = Y gets smaller:
 if asc(right$(Inkey$,1))=38 then spriteY = spriteY - 3
 'down arrow = Y gets bigger:
 if asc(right$(Inkey$,1))=40 then spriteY = spriteY + 3
[updateDisplay]
 'locate sprite at new position and update display
 #w.g "spritexy guy ";spriteX;" ";spriteY
 #w.g "drawsprites"
 'display time elapsed since start of program
 #w.s "Elapsed time: ";time$("seconds") - startTime
 wait
```

Mouse Control

We can trap several mouse events in a graphicbox or graphics window that has received a "Setfocus" command. We'll trap the "mouseMove" event for our demos. Note that "mouseMove" is case sensitive, so type it exactly as written. We'll cause the sprite to appear wherever the user moves the mouse. We trap the mouse movement like this:

```
 #w.g "setfocus" 'MUST setfocus to graphicbox
 #w.g "when mouseMove [update]"
```

We can do it in a single statement, separated by semi-colons.

```
#w.g "setfocus; when mouseMove [update]"
```

After a mouse event has been triggered, Liberty BASIC places the mouse coordinates relative to the graphicbox or graphic area of a graphics window into the special variables **MouseX** and **MouseY**. Remember that variable names in Liberty BASIC are case sensitive, so type them as written.
Since we know where the mouse is located, we can locate the sprite in the same spot and update the display with "Drawsprites".

```
spriteX = MouseX : spriteY = MouseY
#w.g "spritexy guy ";spriteX;" ";spriteY
#w.g "drawsprites"
```

See the third demo at the end of this lesson for a program that moves the sprite when the user moves the mouse.

Centersprite

If you try the third demo program, you'll see that the cursor appears at the upper left corner of the sprite. If instead you want the cursor to be centered over the sprite, you can issue a "Centersprite" command, like this:

```
'This causes a sprite's x, y location to
'refer to the center of the sprite,
'rather than the default upper left corner.
#w.g "centersprite guy"
```

The fourth demo is identical to the third demo in allowing the user to move the sprite by moving the mouse. It is different because the sprite is centered under the mouse cursor.

Challenge

Take any of the four programs that follow and add code to insure that the user-controlled sprite never leaves the visible portion of the window!

Four Demos

The following demonstration programs require bitmaps in your Liberty BASIC sprites folder and they must be run from the Liberty BASIC root directory.

```
'add a sprite and locate it at x=10, y=30
'allow user to press arrow buttons to move sprite
nomainwin
loadbmp "smiley", "sprites\smiley.bmp"
loadbmp "landscape", "sprites\bg1.bmp"
WindowHeight = 300 : WindowWidth = 400
graphicbox #w.g, 0, 0, 400, 300
open "User-Controlled Sprite 1" for window_nf as #w
 #w "trapclose [quit]"
 #w.g "down"
 #w.g "background landscape"
 #w.g "addsprite guy smiley"
 spriteX = 10 : spriteY = 30
 #w.g "spritexy guy ";spriteX;" ";spriteY
 'set up event trapping for key presses
 #w.g "setfocus" 'MUST setfocus to graphicbox
 #w.g "when characterInput [update]"
 #w.g "drawsprites" 'update screen

wait
[update]
 '37 = left arrow
 '38 = up arrow
 '39 = right arrow
 '40 = down arrow
 'Arrow keys make Inkey$ two characters long,
 'so we check the rightmost character with
 'the right$() function.
 'right arrow = X gets bigger:
 if asc(right$(Inkey$,1))=39 then spriteX = spriteX + 3
 'left arrow = X gets smaller:
 if asc(right$(Inkey$,1))=37 then spriteX = spriteX - 3
 'up arrow = Y gets smaller:
 if asc(right$(Inkey$,1))=38 then spriteY = spriteY - 3
 'down arrow = Y gets bigger:
 if asc(right$(Inkey$,1))=40 then spriteY = spriteY + 3
 'locate sprite at new position and update display
 #w.g "spritexy guy ";spriteX;" ";spriteY
 #w.g "drawsprites"
 wait
[quit]
 unloadbmp "landscape"
 unloadbmp "smiley"
 close #w : end
```

```
'Add a sprite and locate it at x=10, y=30.
'Allow user to press arrow buttons to move sprite.
'Allow another event to take place, independently
'from user sprite movement.
nomainwin
loadbmp "smiley", "sprites\smiley.bmp"
loadbmp "landscape", "sprites\bg1.bmp"
WindowHeight = 350 : WindowWidth = 400
graphicbox #w.g, 0, 0, 400, 300
statictext #w.s, "",40,300,100,40
open "User-Controlled Sprite 2" for window_nf as #w
 #w "trapclose [quit]"
 #w.g "down"
 #w.g "background landscape"
 #w.g "addsprite guy smiley"
 spriteX = 10 : spriteY = 30
 #w.g "spritexy guy ";spriteX;" ";spriteY
 'set up event trapping for key presses
 #w.g "setfocus" 'MUST setfocus to graphicbox
 #w.g "when characterInput [updatePosition]"
 #w.g "drawsprites" 'update screen
 'save time program started in variable
 startTime=time$("seconds")
 timer 300, [updateDisplay]
wait
[updatePosition]
 '37 = left arrow
 '38 = up arrow
 '39 = right arrow
 '40 = down arrow
 'Arrow keys make Inkey$ two characters long,
 'so we check the rightmost character with
 'the right$() function.
 'right arrow = X gets bigger:
 if asc(right$(Inkey$,1))=39 then spriteX = spriteX + 3
 'left arrow = X gets smaller:
 if asc(right$(Inkey$,1))=37 then spriteX = spriteX - 3
 'up arrow = Y gets smaller:
 if asc(right$(Inkey$,1))=38 then spriteY = spriteY - 3
 'down arrow = Y gets bigger:
 if asc(right$(Inkey$,1))=40 then spriteY = spriteY + 3
[updateDisplay]
 'locate sprite at new position and update display
 #w.g "spritexy guy ";spriteX;" ";spriteY
 #w.g "drawsprites"
 'display time elapsed since start of program
 #w.s "Elapsed time: ";time$("seconds") - startTime
 wait
[quit]
 timer 0
 unloadbmp "landscape"
 unloadbmp "smiley"
 close #w : end

'add a sprite and locate it at x=10, y=30
'allow user to move sprite with mouse
nomainwin
loadbmp "smiley", "sprites\smiley.bmp"
loadbmp "landscape", "sprites\bg1.bmp"
WindowHeight = 300 : WindowWidth = 400
graphicbox #w.g, 0, 0, 400, 300
open "User-Controlled Sprite 3" for window_nf as #w
 #w "trapclose [quit]"
 #w.g "down"
 #w.g "background landscape"
 #w.g "addsprite guy smiley"
 spriteX = 10 : spriteY = 30
 #w.g "spritexy guy ";spriteX;" ";spriteY
 'set up event trapping for key presses
 #w.g "setfocus" 'MUST setfocus to graphicbox
 #w.g "when mouseMove [update]"
 #w.g "drawsprites" 'update screen

wait
[update]
 'mouse coords are in MouseX and MouseY
 spriteX = MouseX : spriteY = MouseY
 #w.g "spritexy guy ";spriteX;" ";spriteY
 #w.g "drawsprites"
 wait
[quit]
 unloadbmp "landscape"
 unloadbmp "smiley"
 close #w : end



'add a sprite and locate it at x=10, y=30
'allow user to move sprite with mouse
'center sprite under cursor
```

```
nomainwin
loadbmp "smiley", "sprites\smiley.bmp"
loadbmp "landscape", "sprites\bg1.bmp"
WindowHeight = 300 : WindowWidth = 400
graphicbox #w.g, 0, 0, 400, 300
open "User-Controlled Sprite 4" for window_nf as #w
 #w "trapclose [quit]"
 #w.g "down"
 #w.g "background landscape"
 #w.g "addsprite guy smiley"
 spriteX = 10 : spriteY = 30
 #w.g "spritexy guy ";spriteX;" ";spriteY
 'This causes a sprite's x, y location to
 'refer to the center of the sprite,
 'rather than the default upper left corner.
 #w.g "centersprite guy"
 'set up event trapping for key presses
 #w.g "setfocus" 'MUST setfocus to graphicbox
 #w.g "when mouseMove [update]"
 #w.g "drawsprites" 'update screen

wait
[update]
 'mouse coords are in MouseX and MouseY
 spriteX = MouseX : spriteY = MouseY
 #w.g "spritexy guy ";spriteX;" ";spriteY
 #w.g "drawsprites"
 wait
[quit]
 unloadbmp "landscape"
 unloadbmp "smiley"
 close #w : end
```