# A Jumping Sprite

## Table of Contents

These two demos show one way to code a jumping sprite. Essentially, both demos are the same, the only difference being the first uses [BranchLabelEvents] and the second uses called SubEventHandlers. There are certainly other ways to achieve jumping. The two demos here are intended for the novice game coder.

### The Sprite Basics

The first step is to understand how to use sprites. If you are just learning how to code sprites, I highly recommend The Sprite Byte Tutorials by - Alyce .

You can also find these installments of the Sprite Byte Series by - Alyce in the Liberty BASIC Newsletters

Sprite Byte: The Absolute Minimum Liberty BASIC Newsletter #132

Sprite Byte: All About Masks Liberty BASIC Newsletter #143

Sprite Byte: User-Controlled Sprite Liberty BASIC Newsletter #119

Sprite Byte: Scrolling Background Liberty BASIC Newsletter #120

Sprite Byte: Scaling Liberty BASIC Newsletter #121

Sprite Byte: Shooting Liberty BASIC Newsletter #122

Sprite Byte: Shooting Multiple Missiles Liberty BASIC Newsletter #124

Sprite Byte: Computer-Controlled Sprite Liberty BASIC Newsletter #125

Sprite Byte: Collision Detection Liberty BASIC Newsletter #126

Sprite Byte: Changing the Sprite Image Liberty BASIC Newsletter #128

Sprite Byte: Manual Cycling and Sound Liberty BASIC Newsletter #129

Sprite Byte: Adding a Scoreboard or Status Panel Liberty BASIC Newsletter #131

Sprite Byte: Cycling Animation and the Timer Liberty BASIC Newsletter #133

Sprite Byte: More Cycling Liberty BASIC Newsletter #134

Sprite Byte: Making Sprite Graphics Persist Liberty BASIC Newsletter #137

## Sprite Jumping

There are two ways to cause movement of sprite. One is to issue a **SPRITEMOVEXY** command. The other is to find the current `x`, `y` coordinates of the sprite with the **SPRITEXY?** command, increment the `x`, `y` or both, and then issue a **SPRITEXY** command. The technique used in these two demos uses the second method. **SPRITEXY?** and **SPRITEXY** are **two different commands**.

There are two parts to a jump, the *ascending* motion and the *descending* motion. It is the `y` value that determines height. Because `y` begins at the upper border of the graphics window (or graphicbox) and increments down, movement toward a lesser `y` results in an upward movement and movement toward a greater `y` results in a downward movement. Coding a jump must move upward to a minimum `y` and then back to the baseline `y`. Ideally, if the sprite is already in a *left* or *right* movement, the *jump* movement should jump in that direction. If sprite is in a *standstill* state, the *jump* movement should be vertical. It is up to the coder to keep track of which of the three movements states (*left*, *right*, and *jump*) are active.

Active states are kept in variable flags. This demo uses the flags `xDir` and `yDir` to keep track of active direction states.

- xDir = 0 ' No horizontal movement
- xDir = 1 ' Movement to Left
- xDir = 2 ' Movement to Right
- yDir = 0 ' No vertical movement
- yDir = 1 ' Movement Up
- yDir = 2 ' Movement Down

Coding should allow both `xDir` and `yDir` to be in active movement states simultaneously.

## The Demos

The first demo uses [BranchEventLabels]. Because all code resides within the main program, all variables are recognized throughout. The `Timer` is fired every 50 milliseconds. Each time the `Timer` is fired, a **DRAWSPRITES** command is issued, *whether or not any change in the `x` or `y` variables has been made*.

`OldKey` holds the ASCII value of the last key pressed. `NewKey` holds the ASCII value of the current key pressed. If horizontal movement is active and
OldKey = NewKey

then horizontal movement stops. Horizontal cannot be stopped until the jump is completed and the sprite returns to baseline `y`. In this way, the user can both initiate and cease horizontal movement.

The second demo uses all SubEventHandlers. Because variables assigned in the main program are local to the main program, the three flag variables, `OldKey`, `xDir` and `yDir` must be declared as **Global** variables. Currently, there is a bug when using a timer to fire a sub event handler, causing the program to hang. Including a **WAIT** statement before **END SUB** compensates for that bug.

## Demo 1: Using Branch Event Labels

```
    Nomainwin
' OldKey holds last pressed key
    OldKey = 0

' xDir and yDir hold moving directions
    xDir = 0: yDir = 0

    WindowWidth = 757
    WindowHeight = 595

    UpperLeftX = Int((DisplayWidth - WindowWidth) /2)
    UpperLeftY = Int((DisplayHeight - WindowHeight) /2)

    Menu #demo, "&Options", "E&xit", [Quit]
    Graphicbox #demo.gb1, 0, 0, 750, 550

    Open "Controlling Sprites" for Window as #demo
    #demo, "Trapclose [Quit]"

' Load the background bmp
```

```
    Loadbmp "bg", "SPRITES\BG1.bmp"
    #demo.gb1, "Down; Background bg; Drawsprites"

' Load the sprites
    Loadbmp "cm1", "SPRITES\cave1.bmp"
    Loadbmp "cm2", "SPRITES\cave2.bmp"
    #demo.gb1, "Addsprite cm cm1 cm1 cm2 cm2"

' Set the initial cyclesprite command to 0
    #demo.gb1, "Cyclesprite cm 0"

' Set inital x, y variables (cm facing right to start)
    #demo.gb1, "Spritexy cm 350 450"

' Trap keypresses
    #demo.gb1, "When characterInput [KeyPress]"
    #demo.gb1, "Setfocus"

' Set the timer
    Timer 50, [SeeSprites]
    Wait

[Quit]
    Timer 0
    Unloadbmp "bg"
    Close #demo
    End

[KeyPress]
    NewKey = Asc(Right$(Inkey$, 1))
    #demo.gb1, "Spritexy? cm x y"
    Select Case NewKey
        Case 37
            #demo.gb1, "Spriteorient cm mirror"
            If OldKey = NewKey Then
                xDir = 0
                NewKey = 0
            Else
                xDir = 1
            End If
        Case 38
            yDir = 1
        Case 39
            #demo.gb1, "Spriteorient cm normal"
            If OldKey = NewKey Then
                xDir = 0
                NewKey = 0
            Else
                xDir = 2
            End If
    End Select
    OldKey = NewKey
    Wait

[SeeSprites]
    #demo.gb1, "Spritexy? cm x y"
    Select Case yDir
        Case 1 ' Up
            y = y - 10
            If y < 350 Then
                yDir = 2
                y = 350
            End If
        Case 2 ' Down
            y = y + 10
            If y > 450 Then
                yDir = 0
                y = 450
            End If
    End Select
    Select Case xDir
        Case 1 ' Left
            x = x - 7
            If x < 5 Then
                xDir = 0
                x = 10
            End If
        Case 2 ' Right
            x = x + 7
            If x > 710 Then
                xDir = 0
                x = 700
            End If
    End Select
    If xDir + yDir > 0 Then
        #demo.gb1, "Cyclesprite cm 1"
    Else
        #demo.gb1, "Cyclesprite cm 0"
    End If
    #demo.gb1, "Spritexy cm ";x;" ";y
    #demo.gb1, "Setfocus; Drawsprites"
    Wait
```

## Demo 2 Using Sub Event Handlers

```
    Global OldKey, xDir, yDir
' OldKey holds last pressed key
    OldKey = 0

' xDir and yDir hold moving directions
    xDir = 0: yDir = 0

    WindowWidth = 757
    WindowHeight = 595

    UpperLeftX = Int((DisplayWidth - WindowWidth) /2)
    UpperLeftY = Int((DisplayHeight - WindowHeight) /2)

    Menu #demo, "&Options", "E&xit", QuitByMenu
    Graphicbox #demo.gb1, 0, 0, 750, 550

    Open "Controlling Sprites" for Window as #demo
    #demo, "Trapclose QuitByTrap"

' Load the background bmp
    Loadbmp "bg", "SPRITES\BG1.bmp"
    #demo.gb1, "Down; Background bg; Drawsprites"

' Load the sprites
    Loadbmp "cm1", "SPRITES\cave1.bmp"
    Loadbmp "cm2", "SPRITES\cave2.bmp"
    #demo.gb1, "Addsprite cm cm1 cm1 cm2 cm2"

' Set the initial cyclesprite command to 0
    #demo.gb1, "Cyclesprite cm 0"

' Set inital x, y variables (cm facing right to start)
    #demo.gb1, "Spritexy cm 350 450"

' Trap keypresses
    #demo.gb1, "When characterInput KeyPress"
    #demo.gb1, "Setfocus"

' Set the timer
    Timer 50, SeeSprites
    Wait
```

```
Sub QuitByTrap handle$
    Timer 0
    Unloadbmp "bg"
    Close #handle$
    End
End Sub

Sub QuitByMenu
    Call QuitByTrap "#demo"
End Sub

Sub KeyPress handle$, key$
    NewKey = Asc(Right$(key$, 1))
    #demo.gb1, "Spritexy? cm x y"
    Select Case NewKey
        Case 37
            #demo.gb1, "Spriteorient cm mirror"
            If OldKey = NewKey Then
                xDir = 0
                NewKey = 0
            Else
                xDir = 1
            End If
        Case 38
            yDir = 1
        Case 39
            #demo.gb1, "Spriteorient cm normal"
            If OldKey = NewKey Then
                xDir = 0
                NewKey = 0
            Else
                xDir = 2
            End If
    End Select
    OldKey = NewKey
End Sub

Sub SeeSprites
    #demo.gb1, "Spritexy? cm x y"
    Select Case yDir
        Case 1 ' Up
            y = y - 10
            If y < 350 Then
                yDir = 2
                y = 350
            End If
        Case 2 ' Down
            y = y + 10
            If y > 450 Then
                yDir = 0
                y = 450
            End If
    End Select
    Select Case xDir
        Case 1 ' Left
            x = x - 7
            If x < 5 Then
                xDir = 0
                x = 10
            End If
        Case 2 ' Right
            x = x + 7
            If x > 710 Then
                xDir = 0
                x = 700
            End If
    End Select
    If xDir + yDir > 0 Then
        #demo.gb1, "Cyclesprite cm 1"
    Else
        #demo.gb1, "Cyclesprite cm 0"
    End If
    #demo.gb1, "Spritexy cm ";x;" ";y
    #demo.gb1, "Setfocus; Drawsprites"
' Currently there is a bug in the sub timer
' requiring a WAIT statement here
    Wait
End Sub
```

## Table of Contents

## Changing the Height, Arc and Speed

The minimum `y` value in these two demos is `350`. Once the sprite ascends to a height of `350`, the sprite begins its descent. *Decreasing* that value will cause the sprite to jump higher. When `xDir` is active (> 0), each **DRAWSPRITES** command will move the sprite horizontally by `7` pixels (*left* or *right*). When `yDir` is active (> 0), each **DRAWSPRITES** command will move the sprite vertically by `5` pixels (*up* or *down*). Adjusting the `xDir` to a *lesser* number will result in a *steeper* jump. Adjusting `xDir` to a greater number will result in a *longer* jump. The `Timer` is set to `50` milliseconds. A *lesser* number will *increase* the animation speed. A *greater* number will *decrease* the animation speed. - JanetTerra