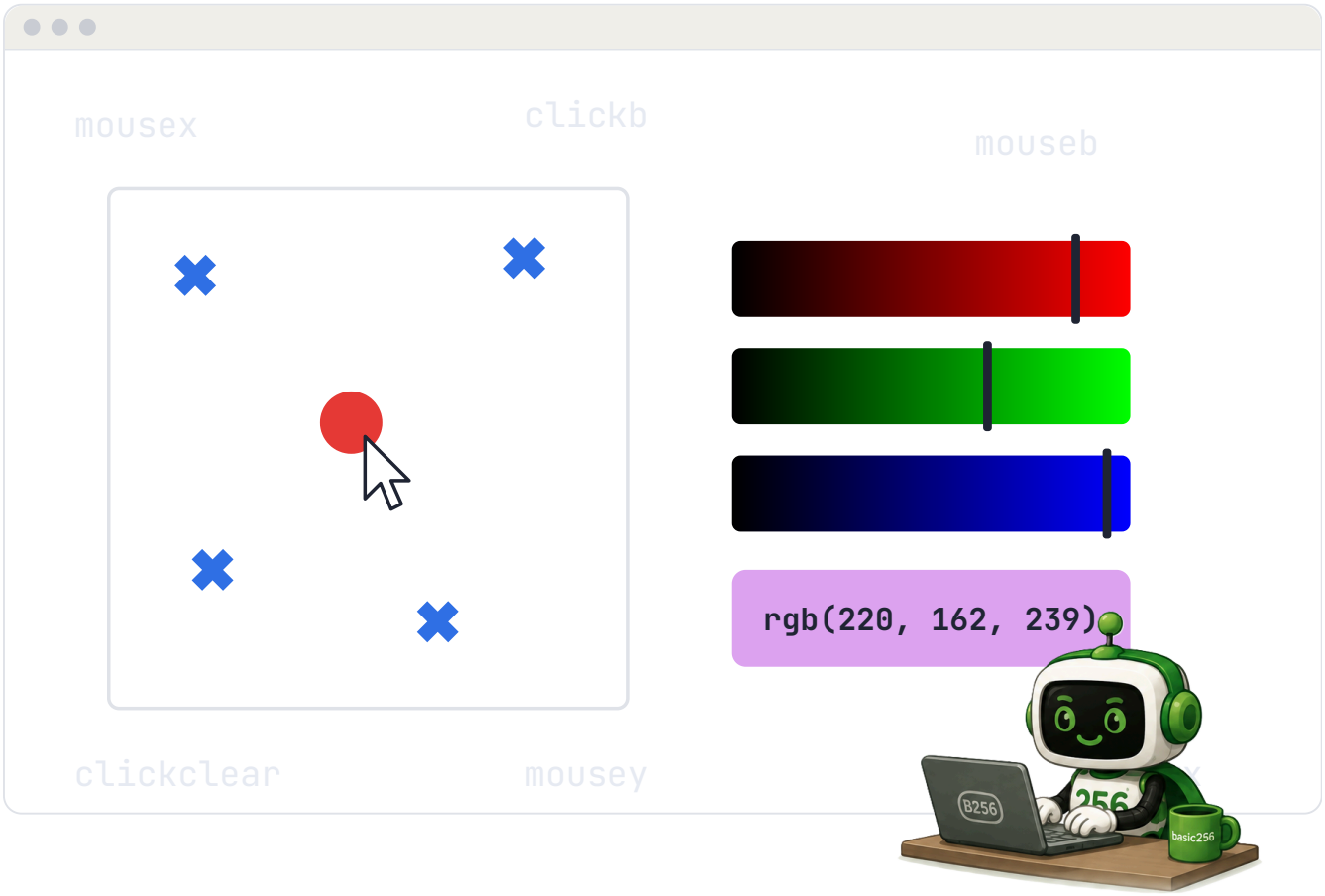
LEARN TO CODE · THE MOUSE

Chapter 11

# Mouse Control — Moving Things Around

So far your programs got their input from the keyboard. In this chapter they listen to the **mouse**. There are two ways to do that: **tracking mode**, where the program asks where the mouse is right now and which buttons are held down, and **clicking mode**, where BASIC-256 remembers the last click until your program is ready for it. The Big Program uses the mouse to build a colour chooser with sliders.




© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 1

# Tracking mode

Where is the mouse right now, and which buttons are held down?

In tracking mode your program uses three functions: **mousex**, **mousey** and **mouseb**. They tell where the mouse pointer is over the graphics output and which buttons are pressed — at the very moment the program asks. If the mouse pointer is not over the graphics output, **mousex** and **mousey** both return −1.

Program 76 draws a red ball under the mouse pointer, over and over, until you click the left button.

mousetrack.kbs — BASIC-256

```
# mousetrack.kbs - track the mouse with a circle

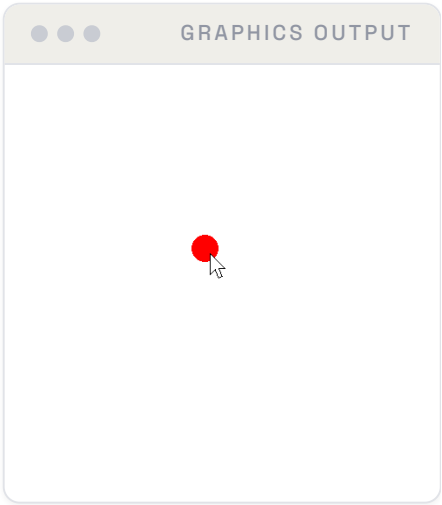
print "Move the mouse around the graphics output."
print "Click the left mouse button to quit."

fastgraphics

# do it over and over until the user clicks left
while mouseb <> MOUSEBUTTON_LEFT
   # erase the screen
   clg
   # mousex is -1 when the mouse is off the output
   if mousex > -1 then
      # draw the ball where the mouse is
      color red
      circle mousex, mousey, 15
   end if
   refresh
end while

print "All done."
end
```

GRAPHICS OUTPUT



The red ball follows the mouse pointer.

**Program 76** · Mouse Tracking

TEXT OUTPUT

```
Move the mouse around the graphics output.
Click the left mouse button to quit.
All done.
```

Every time round the loop the program clears the screen, draws the ball where the mouse is now, and shows the new picture with **refresh**. Because of **fastgraphics** (Chapter 8) you never see the screen half drawn. When the

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

pointer leaves the graphics output, **mousex** becomes −1 and the **if** skips the drawing, so the output is just cleared. Without the **if**, a quarter of the ball would stay stuck in the top left corner, around the point (−1, −1).

### NEW CONCEPT

```
mousex  or  mousex()
mousey  or  mousey()
mouseb  or  mouseb()
```

These three functions tell where the mouse is *now*. Mouse movements outside the graphics output are not recorded. The brackets are optional.




**mousex** The *x* coordinate of the mouse pointer, from 0 to **graphwidth** − 1. Returns −1 when the pointer is not over the graphics output.

**mousey** The *y* coordinate of the mouse pointer, from 0 to **graphheight** − 1. Returns −1 when the pointer is not over the graphics output.

**mouseb** The mouse buttons that are held down right now, as a number. Use the constants in the table below. If several buttons are held down at the same time, **mouseb** returns their values added together: left and right together give 3.

| VALUE | CONSTANT | MEANING |
|---|---|---|
| 0 | MOUSEBUTTON_NONE | no button is pressed |
| 1 | MOUSEBUTTON_LEFT | the left button is pressed |
| 2 | MOUSEBUTTON_RIGHT | the right button is pressed |
| 4 | MOUSEBUTTON_CENTER | the middle button (or the scroll wheel) is pressed |

At the start of every run, before the mouse has moved over the graphics output, all three return 0.




### EXPLORE

If you set up your own coordinates with **window** (Chapter 2), **mousex** and **mousey** give the position in *your* coordinates, so they can have decimals. Only −1 is passed on unchanged, to say that the mouse is not on the graphics output.




### TRY IT

Make the ball blue while you hold the right button down: before drawing, test whether `mouseb = MOUSEBUTTON_RIGHT`. Then draw a cross-hair instead of a ball: a line across the whole output through **mousey**, and one from top to bottom through **mousex**.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 2

# Clicking mode

BASIC-256 remembers the last click, so your program can deal with it when it is ready.

The second way to use the mouse is called clicking mode. When a mouse button is clicked on the graphics output, BASIC-256 stores where the pointer was and which button was clicked. Your program reads them with **clickx**, **clicky** and **clickb**, whenever it is ready. Once it has dealt with a click, a **clickclear** statement resets them, so that the next click can be recognised.

Program 77 marks every spot you click with the left button with a blue X. A click with the right button stops the program.

mouseclick.kbs — BASIC-256

```
# mouseclick.kbs - X marks the spot where you click

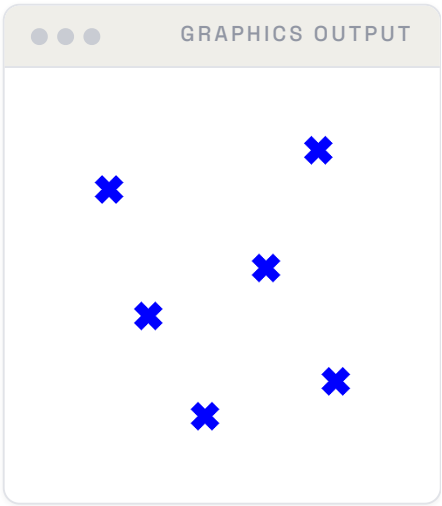
print "Click the left mouse button to mark your spot."
print "Click the right mouse button to stop."

clg
color blue

do
   # forget the last click and wait for a new one
   clickclear
   while clickb = MOUSEBUTTON_NONE
      pause .01
   end while
   # mark a left click with an X
   if clickb = MOUSEBUTTON_LEFT then
      stamp clickx, clicky, 8, {-1, -2, 0, -1, 1, -2,
         2, -1, 1, 0, 2, 1, 1, 2, 0, 1,
         -1, 2, -2, 1, -1, 0, -2, -1}
   end if
until clickb = MOUSEBUTTON_RIGHT

print "All done."
end
```

GRAPHICS OUTPUT



Six clicks with the left mouse button.

**Program 77** · Mouse Clicking

TEXT OUTPUT

```
Click the left mouse button to mark your spot.
Click the right mouse button to stop.
All done.
```

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

The program repeats a **do** ... **until** loop from Chapter 8. Each time it first clears the last click, then waits in a small **while** loop until a button is clicked. The **pause .01** lets the computer rest while it waits. A left click is marked with an X, drawn with **stamp** from Chapter 9 at 8 times its size. A right click is not marked: it ends the loop. Clicking the middle button does nothing at all.




NEW CONCEPT

```
clickx  or  clickx()
clicky  or  clicky()
clickb  or  clickb()
```

These three functions return where the mouse pointer was and which button was clicked, at the moment of the last click on the graphics output. They keep these values until the next click or a **clickclear**. Clicks outside the graphics output are ignored.

**clickb** gives the *one* button that was clicked, with the same values as **mouseb**. The second click of a double click also adds **MOUSEBUTTON_DOUBLECLICK** (32): a double click with the left button gives 1 + 32 = 33. That is why Program 77 doesn't mark the second click of a double click — the first click has already put an X there.




NEW CONCEPT

```
clickclear
```

**clickclear** sets **clickx**, **clicky** and **clickb** back to 0, so **clickb** is **MOUSEBUTTON_NONE** until the next click. Every run also starts with them at 0.




TRACKING OR CLICKING?

Use **tracking mode** when something should follow the mouse, or happen for as long as a button is held down, as in drawing or dragging. Use **clicking mode** when you only care about single clicks. **mouseb** only knows what is pressed at the moment you ask, so a quick click while your program is busy drawing can slip by unnoticed. **clickb** remembers it until you clear it.




TRY IT

Change Program 77 so that a click with the middle button clears the screen. Then give every X a random colour.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout



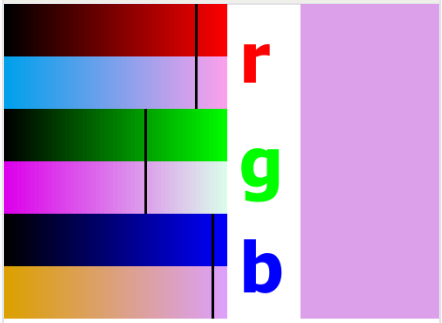

BIG PROGRAM

# Color Chooser

This program uses the mouse to move three colour sliders, so that you can see all 16,777,216 (256 × 256 × 256) colours that **rgb** can mix. Click a slider, or hold the button down and drag.

Each slider has two strips. The top strip shows only its own colour, from 0 on the left to 255 on the right. The bottom strip shows the whole colour you would get if you moved this slider there and left the other two alone. The black bar marks the value chosen now. The swatch on the right shows the colour, and the text below it gives the **rgb** you can type into your own programs.

GRAPHICS OUTPUT




**rgb(220, 162, 239)**

TEXT OUTPUT

```
Color chooser - find a color
Click and drag the red, green and blue sliders.
```

In Program 78, the main loop waits until a button is held down, then reads **mousex** and **mousey** once, into *x* and *y*, so that the rest of the loop works with one position even if the mouse moves meanwhile. A click on the swatch (*x* of 330 or more) is ignored, and so is a −1. The value of *y* tells which slider was clicked, and *x* becomes its new value. A click just past the end of a slider, on a letter, gives 255. As long as you hold the button down, the loop keeps going round, so the slider follows the mouse.

**display** clears the screen and draws everything again, ending with a single **refresh**, so the sliders, the swatch and the numbers always change together.

colorchooser.kbs — BASIC-256

```
# colorchooser.kbs - find a color
# click and drag the red, green and blue sliders

fastgraphics

print "Color chooser - find a color"
print "Click and drag the red, green and blue sliders."

# variables to store the color parts
r = 128
g = 128
b = 128

call display(r, g, b)
```

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

```
while true
   # wait for a mouse button
   while mouseb = MOUSEBUTTON_NONE
      pause .01
   end while
   x = mousex
   y = mousey
   # only use clicks on the sliders, not on the swatch
   # (x is -1 if the mouse has just left the graphics output)
   if x > -1 and x < 330 then
      if x > 255 then x = 255
      # the red slider: y from 0 to 119
      if y < 120 then r = x
      # the green slider: y from 120 to 239
      if y >= 120 and y < 240 then g = x
      # the blue slider: y from 240 to 359
      if y >= 240 and y < 360 then b = x
      call display(r, g, b)
   end if
end while
end

subroutine colorline(r, g, b, x, y)
   # draw part of a slider: a line 60 pixels high
   # from x,y down, in the color r, g, b
   color rgb(r, g, b)
   line x, y, x, y + 59
end subroutine

subroutine redsliderbar(r, g, b)
   # draw the red slider from 0,0 to 255,119
   font "Tahoma", 60, 700
   color rgb(255, 0, 0)
   text 270, 10, "r"
   for t = 0 to 255
      # red, and red mixed with the green and blue
      call colorline(t, 0, 0, t, 0)
      call colorline(t, g, b, t, 60)
   next t
   # slider
   color black
   rect r - 1, 0, 3, 120
end subroutine

subroutine greensliderbar(r, g, b)
   # draw the green slider from 0,120 to 255,239
   font "Tahoma", 60, 700
   color rgb(0, 255, 0)
   text 270, 130, "g"
   for t = 0 to 255
      # green, and green mixed with the red and blue
      call colorline(0, t, 0, t, 120)
      call colorline(r, t, b, t, 180)
   next t
   # slider
   color black
   rect g - 1, 120, 3, 120
```

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

```
end subroutine

subroutine bluesliderbar(r, g, b)
   # draw the blue slider from 0,240 to 255,359
   font "Tahoma", 60, 700
   color rgb(0, 0, 255)
   text 270, 250, "b"
   for t = 0 to 255
      # blue, and blue mixed with the red and green
      call colorline(0, 0, t, t, 240)
      call colorline(r, g, t, t, 300)
   next t
   # slider
   color black
   rect b - 1, 240, 3, 120
end subroutine

subroutine display(r, g, b)
   clg
   call redsliderbar(r, g, b)
   call greensliderbar(r, g, b)
   call bluesliderbar(r, g, b)
   # draw the swatch
   color rgb(r, g, b)
   rect 340, 0, 160, 360
   # write the color the way you would type it
   color black
   font "Tahoma", 30, 700
   text 10, 390, "rgb(" + r + ", " + g + ", " + b + ")"
   refresh
end subroutine
```

**Program 78** · Big Program — Color Chooser




TRY IT

Make the program print the **rgb** text in the text output when you click the right mouse button, so that you can copy it into another program. Remember that **mouseb** is 2 while the right button is held down.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout




PRACTICE

# Exercises

## WORD SEARCH

a t b y l c w t u f t s i l
i j r a e l c k c i l c w e
f o r i g h t x e s u o m n
e c h y d d t n k w f s n b
d c h y w o y h v j j i a p
c l n n y u g b u t t o n o
y i g b k b y a j u h l n i
f c y c c l e b k c i l c n
j k n e i e s t o m f w s t
c x o n l c u a y o h p l e
u i n t c l o h e u y e i r
i u e e l i m a p s h u d f
l v s r v c s n s e c c e d
g t f e l k a h g b y m r c

**Find:** button, center, clickb, clickclear, clickx, clicky, doubleclick, left, mouseb, mousex, mousey, none, pointer, right, slider

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout



## Problems

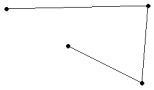
Try these — they only use what this chapter and the earlier ones covered.

**1** Write a program that draws a series of connected lines and shows the points as the lines are drawn. Each time the user clicks the left mouse button, draw a small circle at that point, print its coordinates, draw a line to the point before it (if it is not the first point), and remember the point, so that it can be the start of the next line. Repeat this until the user clicks the right mouse button.

TEXT OUTPUT

```
80,100
310,95
300,220
180,160
```

GRAPHICS OUTPUT

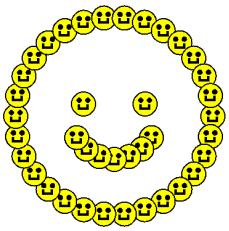


**2** Write a program that lets the user paint with the mouse like a paintbrush. While the user holds the left button down, draw a dot where the mouse is. To make the line wider, draw a circle with a radius of 2 or 3 instead of a single point.

For extra skill: when the user presses the right button, change the paint to a random colour.

**3** Use the smiling face subroutine from Problem 1 of Chapter 10 to make a drawing program with smiles. When the user clicks a point on the graphics output, draw a face there. Then try **mouseb** instead of **clickb**, so that holding the button down and moving the mouse draws a whole trail of faces.

GRAPHICS OUTPUT



Chapter 11 · Mouse Control — Moving Things Around. Modernised layout of the BASIC-256 beginner's guide, updated for BASIC-256 2.3. Programs are numbered 76–78, continuing from Chapter 10. Corrected for 2.3: when the mouse pointer is not over the graphics output, `mousex` and `mousey` return −1 (not the last position), so the mouse tracking program only draws the ball when `mousex` is above −1, and the colour chooser ignores a −1; all three return 0 before the mouse first moves; `clickb` holds the one button that was clicked, not a combination; a double click adds `MOUSEBUTTON_DOUBLECLICK` (32). Fixed in the colour chooser: the red slider now shows its black bar like the green and blue ones; the RGB text is drawn before the `refresh` (it used to show the previous colour); the comments give the right ranges; clicks on the swatch no longer move a slider. The mouse clicking program no longer puts an X where you right-click to stop, and Problem 1 stops with a right click. The programs are redrawn for the 500 × 500 graphics output and use a bold font (weight 700). New: the Explore note on `window`, the button table and the "tracking or clicking?" box. Every program was run in BASIC-256 2.3. Original text © 2019 James M. Reneau, licensed under CC BY-NC-SA 3.0 US. This adaptation is shared under the same licence. Mascot: BitBot.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout