9.5 Events

An *event* is any user interaction with a program. This might involve pressing a key, clicking the mouse button, and moving or dragging the mouse. Every time the user initiates any of these actions the system generates an event signal. Some widgets are designed to react to these signals. A button, for example, waits for a mouse button to be clicked on it; when this happens the buttons callback function is called. Any widget can be programmed to listen for specific events and then call designated functions when these events occur. In this section we will see how this *event programming* works.

The first step in getting a widget to respond to event signals is to tell the widget to listen for signals. If w is a widget, then

```
w.focus_set( )
```

tells the widget to pay attention to signals.

The next step is to register a function as the callback for a specific event. Again, if w is the widget this is done with

```
Widget.bind(w, <event descriptor>, <function>)
```

Here Widget is the superclass for all widget classes. The <event descriptor> is a string, such as "Button−1", which refers to the left mouse button (the only mouse button on a Mac). Here is a table of event descriptors:

Descriptor	Event
$" < KeyPress - a > "$	The users presses the "a" key
$" < KeyPress - A > "$	The user presses the "A" key
	The other letters and digits are similar.
$" < KeyPress - space > "$	The user presses the space bar
$" < KeyPress - Return > "$	The user presses the Return (or Enter) key
$" < KeyPress - Up > "$	The user presses the up-arrow
$" < KeyPress - Down > "$	The user presses the down-arrow
$" < KeyPress - Left > "$	The user presses the left-arrow
$" < KeyPress - Right > "$	The user presses the right-arrow
$" < Any - KeyPress > "$	The user presses any key.
$" < Button - 1 > "$	The user clicks the left mouse button (or only mouse button)
$" < Button - 2 > "$	The user clicks the middle mouse button
$" < Button - 3 > "$	The user clicks the right mouse button
$" < B1 - Motion > "$	This represents a dragging event with the left mouse button

The callback function for any event binding should take one argument, which is an object representing the event. We usually call this argument "event". The specifics of the object will vary according to the type of event that has occurred. With mouse events event.x and event.y are the coordinates of the mouse at the time the event occurred. With keyboard events event.keysym is a string representing the key that was pressed. For letter keys this is the letter as a string: "A", or "a", "space", "Return" and so forth.

Here, for example, is the code needed to empower the "q" key to exit the program:

```
class GUI(Frame):
          ..
        global canvas
        canvas = Canvas(self, width=500, height=500, background="white
        canvas.grid(row=1, column=0)
        canvas.focus_set()
        Widget.bind(canvas, "<KeyPress-q>", self.quitter)

    def quitter(self, event):
        self.quit()
```

This binds the "<KeyPress−q>" description to a function that calls the quit() method of the Frame class. Note that the callback for the binding must take an event argument,so we cant directly connect to the self.quit() method.

If you want to interact with objects that have been drawn on the canvas, it is important to be able to determine which objects are under the cursor when the mouse button is clicked. While it would be possible to program this yourself, the tk system provides some hooks that make this easy. When shapes are created it is possible to give them "tags", which are strings. The canvas.getttags(**object**) method returns a list of all of the tags that have been given to a specific object. The canvas.find_withtag(tag) method returns a list of all of the objects that have the given tag. As the mouse moves around the canvas, the system gives a tag "current" to the topmost (most recently drawn) object that the mouse is currently over. So we can find the object the mouse is currently over by finding the shape in the first entry of canvas.find_withtag("current")

Here is how we might use all of this to create a simple circle-drawing program. We set up the interface to have a Draw button for circles and a set of radio buttons to select the current drawing color. To allow for dragging circles after they have been drawn we bind the left mouse button to a function that selects the circle into which we have clicked, and we bind the dragging operation with the left mouse button to a function that moves the current circle. We also bind the "d" key on the keyboard to a function that deletes the circle the mouse is currently over. This much of our interface definition is as follows:

```
class GUI(Frame):
        .......
        global ColorChoice
        ColorChoice = StringVar()
        ColorChoice.set("red")
        r1=Radiobutton(MenuBar, value = "red", text="red",\
            variable=ColorChoice)
        r1.select()
        r1.grid(row=0, column = 1)
        r2=Radiobutton(MenuBar,value="green",text="green",\
           variable = ColorChoice)
        r2.grid(row=0, column = 2)
        r3=Radiobutton(MenuBar,value="blue",text = "blue",\
           variable = ColorChoice)
        r3.grid(row=0, column = 3)
        r4=Radiobutton(MenuBar,value="cyan",text="cyan",\
           variable = ColorChoice)
        r4.grid(row=1, column = 1)
        r5=Radiobutton(MenuBar,value="magenta",text="magenta",\
           variable = ColorChoice)
        r5.grid(row=1, column = 2)
        r6=Radiobutton(MenuBar,value="yellow",text="yellow",\
           variable = ColorChoice)
        r6.grid(row=1, column = 3)

        CircleButton = Button(MenuBar, text="Circle", \
          command = self.DrawCircle)
        CircleButton.grid(row = 0, column = 4)

        global canvas
        canvas = Canvas(self, width=500, height=500, \
          background="white")
        canvas.grid(row=1, column=0)
        canvas.focus_set()
        Widget.bind(canvas, "<Button-1>", self.SelectCircle)
        Widget.bind(canvas, "<B1-Motion>", self.Drag)
        Widget.bind(canvas, "<KeyPress-d>", self.DeleteCircle)
```

We draw circles (after the Draw button is clicked) at a random position using the ColorChoice selected with the radio buttons:

```
    def DrawCircle(self):
        x = randint(0, 500)
        y = randint(0, 500)
        color = ColorChoice.get()
        c = Circle(x, y, 20, color)
        CircleList.append(c)
```

We move a circle by clicking inside it, then dragging the mouse with the button depressed. The initial click calls the function

```
def SelectCircle(self, event):
        global current
        L = canvas.find_withtag("current")
        for c in CircleList:
            if c.my_shape in L:
                current = c
                break
        global last_pos
        last_pos = (event.x, event.y)
```

This sets a global variable current, which is the object in our Circle class that contains the "current" tag. It also sets the initial value of a global variable last_pos, that we use in the dragging operation.

The mouse-drag callback looks at the current mouse position and the current circle (set in the SelectCircle ( ) function) by the difference between the current position and the last position. It would be slightly simpler to move the current circle to the current mouse position, but that has the unfortunate effect of making the circle jump so that its center is at the mouse position when you start dragging. This unexplained jump is awkward for users, so instead we keep track of the mouses relative motion and use that as the basis for the circle movement. Of course, each time the Drag( ) function is called we need to reset the value of our global variable last_pos.

```
    def Drag(self, event):
        global last_pos
        current.Moveby(event.x-last_pos[0], \
          event.y-last_pos[1])
        last_pos = (event.x, event.y)
```

Finally, we delete a circle in two steps. We need to both erase it from the screen and remove it from our CircleList. Erasing it means finding its shape and calling the canvas.delete( ) method on this shape. To remove it from the CircleList, we must first find the Circle object that contains this shape, then the index of this object in our CircleList, and finally delete this element from the list. Altogether we get the following code:

```
    def DeleteCircle(self, event):
        print "deleting"
        shape = canvas.find_withtag("current")
        if shape == ():
            return
        canvas.delete(shape[0])
        for c in CircleList:
            if c.my_shape == shape[0]:
                i = CircleList.index(c)
                del CircleList[i]
```

Putting all of these together, here is the full program:

```
from tkinter import *
from random import *

class GUI(Frame):
    def __init__(self):
        Frame.__init__(self, None)
        self.grid()

        MenuBar = Frame(self)
        MenuBar.grid(row = 0, column = 0, sticky=W)

        QuitButton=Button(MenuBar, text="Quit", \
          command = self.quit)
        QuitButton.grid(row = 0, column = 0)

        global ColorChoice
        ColorChoice = StringVar()
        ColorChoice.set("red")
        r1=Radiobutton(MenuBar,value = "red",text="red",\
           variable=ColorChoice)
        r1.select()
        r1.grid(row=0, column = 1)

        r2=Radiobutton(MenuBar,value="green",text="green",\
           variable = ColorChoice)
        r2.grid(row=0, column = 2)
        r3=Radiobutton(MenuBar,value="blue",text = "blue",\
           variable = ColorChoice)
        r3.grid(row=0, column = 3)
        r4=Radiobutton(MenuBar,value="cyan",text="cyan",\
           variable = ColorChoice)
        r4.grid(row=1, column = 1)
        r5=Radiobutton(MenuBar,value="magenta",text="magenta",\
           variable = ColorChoice)
        r5.grid(row=1, column = 2)
        r6=Radiobutton(MenuBar,value="yellow",text="yellow",\
           variable = ColorChoice)
        r6.grid(row=1, column = 3)
```

Program 9.5.1: Drawing With Circles

```
        CircleButton = Button(MenuBar, text="Circle", \
          command = self.DrawCircle)
        CircleButton.grid(row = 0, column = 4)

        global canvas
        canvas = Canvas(self, width=500, height=500, \
         background="white")
        canvas.grid(row=1, column=0)
        canvas.focus_set()
        Widget.bind(canvas, "<Button-1>", self.SelectCircle)
        Widget.bind(canvas, "<B1-Motion>", self.Drag)
        Widget.bind(canvas, "<KeyPress-d>", self.DeleteCircle)

        global CircleList
        CircleList = []

    def DrawCircle(self):
        x = randint(0, 500)
        y = randint(0, 500)
        color = ColorChoice.get()
        c = Circle(x, y, 20, color)
        CircleList.append(c)

    def SelectCircle(self, event):
        global current
        L = canvas.find_withtag("current")
        for c in CircleList:
            if c.my_shape in L:
                current = c
                break
        global last_pos
        last_pos = (event.x, event.y)

    def Drag(self, event):
        global last_pos
        current.Moveby(event.x-last_pos[0], event.y-last_pos[1])
        last_pos = (event.x, event.y)
```

Program 9.5.1: Drawing With Circles, continued

```
    def DeleteCircle(self, event):
        print "deleting"
        shape = canvas.find_withtag("current")
        if shape == ():
            return
        canvas.delete(shape[0])
        for c in CircleList:
            if c.my_shape == shape[0]:
                i = CircleList.index(c)
                del CircleList[i]

class Shape:
    def __init__(self, vertices, color):
        self.color = color
        self.vertices = vertices
        self.my_shape = None

    def pos(self):
        return (self.vertices[0][0], self.vertices[0][1])

    def Moveto(self, a, b):
        # This moves the shape to point (a,b)
        (x, y) = self.pos()
        d0 = a-x
        d1 = b-y
        self.Moveby(d0, d1)

    def Moveby(self, a, b):
        # This moves the shape a units horizontally
        # and b units vertically
        canvas.move(self.my_shape, a, b)
        canvas.update()
        for v in self.vertices:
            v[0] = v[0] + a
            v[1] = v[1] + b

    def ChangeColor(self, color):
        # This changes the shape's color.
Possible colors include
        # "white", "black", "red", "green", etc.
        canvas.itemconfigure(self.my_shape, fill = color )
        canvas.update()
        self.color = color
```

Program 9.5.1: Drawing With Circles, continued

```
class Oval(Shape):
    def __init__(self, x, y, hrad, vrad, color):
        # This creates an oval centered at (x, y) with
        # horizontal radius hrad and vertical radius vrad
        Shape.__init__(self, [[x-hrad, y-vrad], \
          [x+hrad, y+vrad]], color)
        v0 = self.vertices[0]
        v1 = self.vertices[1]
        self.my_shape = canvas.create_oval(v0[0], v0[1], \
          v1[0], v1[1], fill = color)

    def pos(self):
        v0 = self.vertices[0]
        v1 = self.vertices[1]
        return ( (v0[0]+v1[0])/2, (v0[1]+v1[1])/2 )

class Circle(Oval):
    # Creates a circle centered at (x, y) with the given radius
    def __init__(self, x, y, radius, color):
        Oval.__init__(self, x, y, radius, radius, color)

def main():
    window = GUI()
    window.mainloop()

main()
```

Program 9.5.1: Drawing With Circles, conclusion