9.4 Animation

It is tempting to try to animate objects by giving each object a loop that describes its motion, as in

```
def randomCircle(self):
        c = Circle(250, 50, 30, "red")
        while True:
            c.Moveby(0, -0.1)
            (x, y)= c.pos()
            if y < 0:
                break
```

This suffers from two problems. First, it isnt interruptable; if we try to exit from the program, perhaps via a Quit button, the program will wait until this animation is finished before exiting. Even worse, if we have several objects with code like this only one will move at a time, continuing its motion until it is finished before the next object starts to move.

To make our animations more effective, we need three simple components:

1. We will give every object a NextStep( ) method that performs one small step of the objects entire motion.

2. We will store all of the moving objects in an AnimationList. One step of the global animation will consist of running through the AnimationList and telling each object in it to take its NextStep( ).

3. Instead of looping through this process, which would not be interruptable, we will make use of a special method of the Canvas class:

   ```
   canvas.after(<time to pause>, <function>)
   ```

   This method pauses for a small amount of time (measured in milliseconds), then calls the given function.

This does everything we want. If our individual NextStep() methods arent too complex and if we dont have too many objects in the animation list, this can make the motions look continuous and smooth. By adjusting the first argument of the canvas. after( ) call we can fine-tune the speed of the animation. All of our widgets that need user control will still be functional; the system will respond to them after each pass through the AnimationList.

For a first example, we have a program with a drawing menu with two options: *Square* and *Circle*. When the user selects one of these options a random shape of that type is drawn and added to the AnimationList. The NextStep function is particularly simple: it moves the shape downward a small amount. When the object gets to the bottom of the canvas it is deleted. Here is the NextStep( ) method for each object:

```
def NextStep(self):
        self.Moveby(0, 2)
```

and here is the Animate( ) function:

```
def Animate():
    for object in AnimationList:
        object.NextStep()
        (x, y) = object.pos()
        if y > 500:
            i = AnimationList.index(object)
            del AnimationList[i]
            object.delete()
    canvas.after( 50, Animate )
```

What follows is the complete program for this, omitting the definition of the drawing classes that we saw in the last section:

```
from tkinter import *
from time 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)

        self.MakeDrawingMenu(MenuBar)

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

        global canvas
        canvas=Canvas(self, width=500, height=500, \
          background="white")
        canvas.grid(row=1, column=0)

        global CircleList
        CircleList = []
```

Program 9.4.1: Squares and Circles

```
    def MakeDrawingMenu(self, MB):
        Draw_button = Menubutton(MB, text='Draw')
        Draw_button.menu = Menu(Draw_button)
        Draw_button['menu'] = Draw_button.menu
        Draw_button.grid(row = 0, column = 1)
        Draw_button.menu.add_command(label='Square', \
          command=self.randomSquare)
        Draw_button.menu.add_command(label='Circle', \
          command=self.randomCircle)

    def randomSquare(self):
        colors=["red","green","blue","yellow","orange","brown"]
        x = randint(1, 500)
        y = randint(1, 500)
        size = randint(5, 50)
        color = colors[ randint(0, len(colors)-1) ]
        s = Square( x, y, size, color)
        AnimationList.append(s)

    def randomCircle(self):
        colors=["red","green","blue","yellow","orange","brown"]
        x = randint(1, 500)
        y = randint(1, 500)
        size = randint(3, 25)
        color = colors[ randint(0, len(colors)-1) ]
        c = Circle( x, y, size, color)
        AnimationList.append(c)
```

Program 9.4.1: Squares and Circles, continued

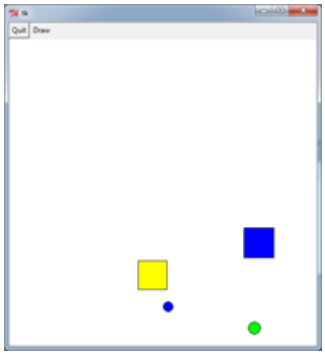
```
def Idle():
    for object in AnimationList:
        object.NextStep()
        (x, y) = object.pos()
        if y > 500:
            i = AnimationList.index(object)
            del AnimationList[i]
            object.delete()
    canvas.after( 50, Idle )

def main():
    global AnimationList
    AnimationList = []
    window = GUI()
    Idle()
    window.mainloop()

main()
```

Program 9.4.1: Squares and Circles, concluded

Here is a picture of this programs window as it runs:



The next example has particles bouncing off walls indicated by the edges of the canvas. This time we make a class Particle to represent the moving bodies. Particle is just a subclass of Circle, with a few added properties. To make the particle move we maintain a vector <x_dir, y_dir> that holds a change amount for the x- and y-directions. At each step of the animation we add x_dir to the particles x-coordinate and y_dir to its y-coordinate. This moves the particle in a straight, not necessarily horizontal or vertical, line. To bounce off the vertical walls, on which the x-coordinate is a constant, we maintain the same value of y_dir and change x_dir to x_dir . Similarly, to bounce off the horizontal walls, where the y-coordinate is constant, we change y_dir to y_dir . Here is the resulting Particle class:

```
class Particle(Circle):
    def __init__(self):
        x = randint(0, 500)
        y = randint(0, 500)
        self.radius = 10
        colors = ["red","green","blue","yellow",\
          "purple","orange","magenta"]
        color = colors[ randint(0, len(colors)-1) ]
        Circle.__init__(self, x, y, self.radius, color)
        self.x_dir = randint(-3, 3)
        self.y_dir = randint(-3, 3)

    def NextStep(self):
        self.Moveby(self.x_dir, self.y_dir)
        (x, y) = self.pos()
        if x < 5 or x > 495:
            self.x_dir = -self.x_dir
        if y < 5 or y > 495:
            self.y_dir = -self.y_dir
```

Here is the complete 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)

        MoreButton = Button(MenuBar, text="More", \
          command = self.MoreParticles)
        MoreButton.grid(row = 0, column = 1)

        PauseButton = Button(MenuBar, text="Pause", \
          command = Pause)
        PauseButton.grid(row = 0, column = 2)

        global canvas
        canvas = Canvas(self, width=500, height=500, \
          background="white")
        canvas.grid(row=1, column=0)

    def MoreParticles(self):
        p = Particle()
        AnimationList.append(p)
```

Program 9.4.2: Bouncing Off the Walls

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

    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", etc.
        canvas.itemconfigure(self.my_shape, fill = color )
        canvas.update()
        self.color = color

    def delete(self):
        canvas.delete(self.my_shape)

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

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)
```

Program 9.4.2: Bouncing Off the Walls, continued

```
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)

class Particle(Circle):
    def __init__(self):
        x = randint(0, 500)
        y = randint(0, 500)
        self.radius = 10
        colors = [ "red", "green", "blue", "yellow", \
          "purple", "orange", "magenta"]
        color = colors[ randint(0, len(colors)-1) ]
        Circle.__init__(self, x, y, self.radius, color)
        self.x_dir = randint(-3, 3)+ 0.5
        self.y_dir = randint(-3, 3)+ 0.5

    def NextStep(self):
        self.Moveby(self.x_dir, self.y_dir)
        (x, y) = self.pos()
        if x < 5 or x > 495:
            self.x_dir = -self.x_dir
        if y < 5 or y > 495:
            self.y_dir = -self.y_dir

def Animate():
    if pause:
        return
    for x in AnimationList:
        x.NextStep()
    canvas.after( 1, Animate )

def Pause():
    global pause
    if pause:
        pause = False
        Animate()
    else:
        pause = True

def main():
    global AnimationList
    AnimationList = []
    global pause
    pause = False

    window = GUI()
    Animate()
    window.mainloop()

main()
```

Program 9.4.2: Bouncing Off the Walls, concluded