Computer Science 1 — CSci 1100
Lab 10 — Drawing, Classes, Testing
Fall Semester 2014

Lab Overview

So far, the outputs from our programs have been text files and, occasionally, images. In this lab, we will draw on a graphical display using a package called `Tk` (through the module `Tkinter`). This should give you an initial feel for graphical interface programming. Along the way we will practice a bit more with writing and testing classes. While reading Chapter 14 of the Python book before lab will help you understand what is happening, all the specific information necessary to complete the lab is provided here.

You will also notice that the objects we are working with in this lab are very similar to the bird and pig objects of HW 7. While you could certainly display the birds and pigs of HW 7 using the techniques from this lab, the work you are required to do for HW 7 and for Lab 10 are completely separate.

Checkpoint 0 – Get the ball moving...

Before we start with the real work of this lab, you need to first get familiar with the methods we are going to use. `Tkinter` allows you to create animations as well as create graphical user interfaces. We will only work with a single class from `Tkinter` called `Canvas` that allows you to draw objects in a window. Please download `lab10files.zip` and unzip it. Open `check0.py`, take a quick look, and then run it. You will see a single ball move across the screen until it reaches the right edge and then stop. The program ends when you close the window.

To begin to understand what is going on, take a look at `check0.py`. We start by initializing a canvas. A canvas is similar to an image, but the changes you make to the canvas are always visible (whereas in an image, you need to explicitly show the image). A canvas must be attached to an object of type `Tk`. In our code, this object is named `root`.

Read all the way through `check0.py`, using the comments to guide your understanding. Explore the code by making the following changes. For each change, rerun the code, and then restore the value you changed:

- Change the color to green.
- Change the starting location and the dx,dy values
- Change the wait time
- Comment out the code that deletes all previous drawings.

The last line of the program

```
root.mainloop()
```

means that the `root` window listens continuously to any commands (events) that are sent to it. We have added no listening in our code, so this just waits until the window is closed.

Once you have completed your experiments, please move onto Checkpoint 1, where the graded activity of the lab starts.

Checkpoint 1 — Ball class

Notice that the information about the ball is stored in several separate variables of the class `BallDraw`. This is a good indication that a class is needed to gather and encapsulate this information. Therefore, in Checkpoint 1 you will add a ball class that places all the functionality for the ball into a separate class, called `Ball`. The ball position, dx and dy values, radius, and color should be included as attributes. The `Ball` class must have the following methods:

- `__init__` to initialize a Ball object that takes as input x,y coordinates, dx,dy values, radius and the color,
- `position()` returns a 2-tuple containing the x and y positions of the center of the ball,
- `move()` changes the current location of the ball by adding the dx and dy offsets to the ball. It does not return anything,
- `bounding_box()` returns a box containing the ball in the form of a 4-tuple,
- `get_color()` returns the ball's color, and
- `some_inside(maxx,maxy)` that moves the `while` loop condition into the ball class.

Place the class in a separate module called `Ball.py`.

Test your Ball class by running `test_Ball.py`, a Python program that uses a module called Nose to test code. Notice that `test_Ball.py` imports `Ball`. The code in `test_Ball.py` has been written to run and check methods from the Ball class. Such automatic testing is a common feature of all significant programming projects.

Make some changes to `test_Ball.py` to see what happens when the tests and the output of code do not match. Finally, observe that the code in `test_Ball.py` will only run correctly if you have built the `Ball` class methods parameter lists and returns correctly.

As the last step in Checkpoint 1, copy `check0.py` to `check1.py` and modify it to make use of `Ball.py`. Make sure you import `Ball`!!

To complete Checkpoint 1: Show your class implementation, nose test execution results (including the introduction of some errors), and the working version of `check1.py` to a TA or mentor.

Checkpoint 2 — Bouncing off the Walls

Now things will get interesting. Instead of your ball disappearing from one end of the canvas, what if the ball just bounced when it hit the edge of the canvas and started moving in a different direction? In particular if it hits the left (`x==0`) or right (`x==maxx`) wall, it should negate its dx value, while if it hits the top (`y==0`) or bottom (`y==maxy`) wall, it should negate its dy value. This creates the effect of a bouncing ball with no friction and no spin. Accomplish this in a new method of the `Ball` class called `check_and_reverse` which should have two arguments: `maxx` and `maxy`. You will need to call this function from the main code each time you move the ball.

Add code in `test_Ball.py` to test `check_and_reverse`.

To complete Checkpoint 2: Show your class implementation, your new test cases and nose execution, and the working program to the TA or a mentor.

Checkpoint 3 — Multiple Balls

We finish the lab with a fun exercise. Using the `random` module create 10 balls with random initial locations, random dx and dy values, random radii, and random colors. To do so, start by including the `random` module in your code. Then, for random integers, use the function:

```
random.randint(min,max)
```

All initial positions should be within the canvas (10,390). Each dx and dy value should be in the range -8 to 8. Each radius should be between 5 and 10. The colors should come from

```
colorList = ["blue", "red", "green", "yellow", "magenta", "orange"]
```

and you can use the function

```
random.choice(colorList)
```

to pick a random color from this list.

Move each ball by its dx and dy values in each step. Instead of executing the function `chart_1.delete(tk.ALL)`, execute it every N-th step, varying N from 1 to 100 or 200. In other words, clear the screen after every N steps. Also, simply make your `while` loop be an infinite loop.

To complete Checkpoint 3: Show your working program to a TA or mentor.