The Beauty and Joy of Computing

Lab Exercise 8: SnapPy Exploration and Testing

Objectives

This lab is different from previous labs. You will gain more experience and familiarity with Python in this lab, but unlike previous weeks there aren't specific concepts or learning objectives that are the focus of the lab. You will experiment with an animation library in Python, create a simple animation of your own, and then write up a brief report on your experiences.

There is no quiz for this week's lab (the lab itself counts for the full 100 points). Enjoy Fall Break!

Activities (In-Lab Work)

In this lab you will be experimenting with a Python module that I am working on that adds animation capabilities to Python with operations that are patterned after BYOB. Since the newer version of BYOB is called "Snap!" and this is in Python, I am calling this library SnapPy. You will re-do some of the earlier animation BYOB exercises in SnapPy, and then explore some on your own to make your own animation.

Activity 1: In this activity you will set up the computer with the software needed for the remainder of the lab, and pull up and experiment with a sample animation program. SnapPy is built on top of a free Python game library called Pygame. The standard Pygame install procedure on Windows requires administrator privileges, so we cannot do this on school-managed systems. While there are work-arounds to get the library to install, we are going to do something a little different, which you can do on your own system as well if you want to experiment more with Python.

We will use a special distribution of Python called "Portable Python," which is designed so that you can have a full Python working environment stored on a USB thumb drive that you can carry around with you and use on different systems. We will install it on the hard drive of the lab computer rather than a thumb drive, because that is faster for our purposes. First you need to download the correct version of Portable Python: Go to http://portablepython.com, click "Download" at the top, and then click "Older versions". The version you want to download is version 2.7.3.2 (this is the first version that included PyGame, and it is less than half the size of the latest version in the 2.7 series). Click on this to download the file, and the go into the downloads folder and run this .exe file to start the install procedure. When it prompts you for a location, go to the beginning of the folder name and put in "C:\" to indicate the local harddrive. Finally, follow the rest of the installation prompts. It will take a while to load everything, so be patient!

To check whether your Portable Python installation will run, click on "Computer" in the Start Menu, browse into the C:\ drive, and then into the Portable Python folder. Inside, you'll find the IDLE program, so you should double-click this to run it. If an IDLE window opens, similar to what

you saw in the previous two labs, then everything is working fine! Leave the IDLE program up and running while you do the following steps.

Next, you will need to copy a set of files into a folder to use for this lab. The files are all in a “Zip file” on the class web site, just under this Lab 8 write-up. If you haven’t used Zip file before, it is just a single file that acts as a container for compressed versions of other files. You can transfer thousands of files with a single file transfer and then “unzip” or “extract” all the individual files after the file transfer is complete. In this particular case, Lab8.zip contains 17 individual files that you will need for this Lab. Depending on which browser you are using this may look slightly different, but in Firefox you would click on the Lab8.zip link and select the “Open with Windows Explorer” option to see what the zip file contains. This will open a regular file browser window, and you can copy the Lab8 folder over to a convenient location, such as your UNCG S: drive. For the rest of this writeup, let’s assume that you have stored this folder and all its contents at S:\CSC100\Lab8

You should have IDLE, the Python development environment that is part of “Portable Python,” running from earlier, so select “File/Open” in the IDLE Shell window, and browse to S:\CSC100\Lab8 in order to open “Lab8-Activity1.py”.

Open the Lab1 write-up from the class web page and review the BYOB code for the conversation between Alonzo and the hovering dragon (Activity 3). That’s exactly what the code in “Lab8-Activity1.py” does, with one change: I don’t have the green flag start function finished in SnapPy, so everything is started by broadcasting a signal named ‘Go’. There’s also one extra event handler routine that will move Alonzo up when the “w” key is pressed. Read through and study the Python code and see how the BYOB scripts are translated into Python code.

To start the Python code, press “F5” in the Lab8-Activity1.py program window to load the code, and you should see the “RESTART” message in the shell window. You should also see a window open up with some familiar looking characters in it! Remember the event handler for the “w” key? Try pressing “w” a couple of times and see what happens.

You can also interact with these characters using BYOB-style Python functions. For example, try typing “`alonzo.changeYBy(40)`” in the Python shell. Did you see Alonzo move up? Try repeating, but with -40 as the argument to move Alonzo back down. Try some of the other functions from the list of SnapPy functions at the end of this write-up. They work the same as similarly-named BYOB functions, but be careful with locations: These are (x,y) coordinate pairs, where those parentheses are part of the location notation. In other words, if you want to make Alonzo move to location (-20,-50) you would say alonzo.goTo((-20,-50)) - the two sets of parentheses are there because the outer ones are from the function notation (parentheses around arguments) and the inner ones are part of the location coordinate notation. That’s hard to explain, but it makes perfect sense if you think about for a bit.

Now try running the animation: Type “`snappy.broadcast('Go')`” in the Python shell. You should see the whole animation play out.

Next, try making some new event handlers. Copy the event handler for the “w” key and paste it three times, changing the keys to “a”, “s”, and “d” keys to make Alonzo left, down, and right. Reload by pressing “F5” and make sure Alonzo moves around.

Activity 2: You’ll make your own animation in this activity. Start by selecting “File / New File” in the Python shell to create a new, blank program. Do an immediate “Save As” to save your new (blank) program as “Lab8-Activity2.py” – note that Lab8-Activity1 will stay open so that you can use it as an example to follow, and you can cut and paste from one window to the other if you’d like to.

For your animation, some BYOB costumes were loaded when you extracted files from the Lab8.zip file. The following files are available, which correspond to the costumes with the same names in BYOB: duck1.png, horse1-b.png, ghost1.png, ghoul1-a.png, troll.png, girl1-standing.png, and girl1-walking.png. You also can make available any free costumes you can find online simply by saving them in the Lab8 folder. Costumes can either be loaded when you create a sprite with `snappy.Sprite()` or you can add them as additional costumes to an existing sprite using `loadCostume()`. Use the code in Lab8-Activity1 as an example to follow.

Refer to the end of this handout for a quick SnapPy guide and function listing. Use this list for some ideas on what you can do in your animation! Remember that names in Python are always case sensitive, so get your capitalization right, and there is almost no error-checking or robustness protections in SnapPy so if you mess up you may have to close out IDLE and re-start everything. Save often! Spend some time and make a decent animation, but stop at least 30 minutes before the end of the lab period so you can complete Activity 3.

Activity 3: Finally, use Word to write a one-page report on your experience with SnapPy. Think about how your experience with Python and SnapPy in this lab differs from what you did in Lab 1 with BYOB. Save this as Lab8.doc (or Lab8.docx).

Write at least one paragraph comparing the systems. Is the 2-window IDLE interface confusing? Do you ever get the program and shell windows confused? Did you have difficulties with aspects of the BYOB interface?

Write at least one paragraph on what aspects were easy or difficulty for you in SnapPy, as compared to BYOB. Did you have a hard time getting function names correct? Was the overall structure of the SnapPy program easy to follow? Did you encounter error messages that you couldn’t understand?

Finally, write a paragraph thinking about this: Do you think you could have written the Python animation in the first lab of the semester, like we did in BYOB? What would be the most difficult parts of trying to do this in Lab 1? What aspects would have been better in Python than BYOB?

Submission

You should submit your Python programs (Lab8-Activity1.py and Lab8-Activity2.py) as well as your experience report (Lab8.doc).

SnapPy Reference

SnapPy includes almost all of the basic movement functions of BYOB, but the “Pen” functions are not implemented yet, as are some of the action features like detecting collisions. In addition, costumes must be referred to by costume number (starting at costume number 0) rather than name. Remember that locations are given in (x,y) notation where the parentheses are necessary. In the following list of functions, “`sprite.`” at the beginning stands in for any particular sprite. For example, if alonzo is supposed to move 30 steps, you would say “`alonzo.move(30)`”. The functions are organized below by what category you would find them under in BYOB.

Functions from the BYOB “Motion” category

```
sprite.move(steps)
sprite.turnCW(degrees)
sprite.turnCCW(degrees)
sprite.pointInDirection(dir)
sprite.setRotStyle(style)
sprite.goTo(pos)
sprite.glideForTo(duration, to)
sprite.changeXBy(deltaX)
sprite.setXTo(newX)
sprite.changeYBy(deltaY)
sprite.setYTo(newY)
```

Functions from the BYOB “Looks” category

```
sprite.loadCostume(fileName)
sprite.switchToCostume(number)
sprite.sayFor(text, duration)
sprite.say(text)
sprite.setSize(size)
```

Functions from the BYOB “Control” category

```
sprite.wait(duration)
```

Hat blocks: The equivalent of a hat block is including one of the following lines immediately before a function definition, which controls what event makes the following definition run. Functions like this that are written as event handlers should not have any parameters.

```
@snappy.startOnReceiving(signalName)
@snappy.startOnKey(key)
```

Non sprite-specific functions: Some functions operate on a larger scale, independent of a particular sprite. For example, sending a broadcast message is not done any differently

regardless of what sprite sends it. For such a function, you always put "`snappy.`" at the beginning of the function name:

```
snappy.broadcast(signalName)
```

Starting and stopping: You must call `snappy.start()` before any other SnapPy functions are called. You can use `snappy.stop()` to halt any ongoing scripts.