Assignment 3: Animation, and Automating Animation

This is the second of the two-part (Assignments 2 and 3) exploration of animation. Now the task is to automate the animation process procedurally, getting yourself into scripting in Python. We'll go from simple Python scripting in Maya's script editor to using Nimble to develop in Py-Charm code that is executed on its interpreter with remote invocation of maya commands (cmds) in Maya.

Readings: Same as the readings for Assignment 2, plus introductory tutorials for PyCharm if you are not familiar with this IDE.

Deadlines: Two submissions during this week. Note that EOD = 11:59 PM on the specified date

EOD February 4: Work-in-progress demo due midnight Wednesday (see below)
EOD February 9: Final version due midnight Monday (debrief in Tuesday's class)

Punctuality: ≥50% deducted if late (without prior arrangement and/or credible excuse).

1. This is to first get you familiar with execution from Maya's Python tab of the script editor. Open the script editor and enter the following script which simply displace by 10 units (cm, by default) in X whatever Maya object is currently selected. Then select some object in the scene, and push the green 'go' arrow in the script editor to run:
```
import maya.cmds as cmds
cmds.move(10.0, 0.0, 0.0, relative=True)
```
2. Now, with reference to the install notes, follow the steps to install Python, Qt, PySide, native Git app, Pycharm, Nimble, etc. This will allows remote script invocation between PyCharm and Maya. Carefully follow the step-by-step instructions that were emailed Sunday February 1 at 8:21 PM. (And since there were two earlier versions with minor errors, the corrected installation instructions are also included at the bottom of this assignment. Several students have already accomplished this install successfully, so the basic instructions have been verified.)
3. Next, write a more involved few-statement script in Pycharm to *create* a sphere and *translate* it to some location procedurally. Write this in PyCharm (but you'll not execute it there), then copy-and-paste from PyCharm into the Maya Python tab of the script editor. Select and execute and you should see the results successfully in the scene. Check the attributes of the sphere to see that it is indeed translated as intended.
4. Next, take the same little script from Step 3, and get ready to execute it in PyCharm's interpreter. First get Maya ready to receive communications via Nimble. You'll need to execute in Maya **import nimble** followed by **nimble.startServer(0)** (or if you have followed the Install instructions Step 8.6 to the extent of creating a tool for this operation in Maya's UI shelf, then simply click that tool). Now edit your PyCharm script, changing the import of cmds to be: **from nimble import cmds**. Then create a new scene in Maya, hold your breath (use safety glasses if available), run your script in PyCharm. You should see the same results in the Maya scene, but created by remote execution. Resume breathing, remove safety glasses if used.

5. Next, we start to use a modern and popular GUI builder, Qt. As you can see from the MayaPy application, one can use Qt to create a user interface to control the execution of scripts in PyCharm. Create a copy of the MayaPy project within PyCharm for your experiments and study the software architecture. This is in keeping with a 'best practice' for organizing the various components. Note that within the src/mayapy package is the MayaPyApplication. You'll be digging down to the views directory to customize the presentation of widgets that will have callbacks to your own methods. Now you can start to develop in PyCharm and perform remote execution in Maya.

   Scripting the bubbles! Note that some of the more elaborate bubble simulations one might have come up with involves a lot of setup (e.g., of deformers) that would not be readily programmed. In the following, aim for a reasonable script that creates a bubble and animates its growth and ascent within an (otherwise invisible) column of boiling water. Consider what you can program and what is perhaps best regarded an asset that is set up 'manually' and used by a program. This is very much a trade-off in modern graphics, the division of labor between asset creation and subsequent procedural use. So:

6. Now create a Python script (call it bubble.py) which will create and animate a bubble via Nimble. (Once executed, scrub along the time line to see it animate). Later you can render the animation, but that's not the goal now. Once debugged, provide this script as an attached .py file. One should then be able to copy-and-paste it into the script editor, run it, and have the fully keyframed bubble animation created, so that if the play button of the animation is hit, the bubble is created and rises in the scene.
7. Now, create a Python script (call it bubbles.py) that creates a container-full of bubbles, at random locations and start times and other initial conditions. This should be a standalone program that executes in Maya's script editor.
8. Integrate your single bubble and multi-bubble implementations (Steps 6 and 7) to be executed via a Qt-based interface. Adopt a copied of the MayaPy project as a new PyCharm project to control the UI.
9. Write a report on this assignment. Turn in a snapshot (debrief plus code attachments) that documents how far you have gotten by midnight Wednesday (February 4) so we can have some review in class on Thursday (February 5). Then prepare the finalized version with a revised and more extensive debrief/commentary with code attachments to email by the following midnight Monday (February 9).

Installing Nimble and Associated Building Blocks

Here are the steps to follow to get our environment going, incorporating Python, PyCharm, Git, QtCreator, and Maya, and building on Qt and PySide. It's somewhat complicated, but once set up, should serve us well. Please adhere carefully to the following order:

Step 1: Python
1.1) Download **Python 2.7.8** from https://www.python.org/download/releases/2.7.8/
Use the default installation location. For example, in OS X, Python should be in:
/Library/Frameworks/Python.framework/Versions/2.7/bin/python

Step 2: Qt 4.8.6 (*GUI builder, which, using PySide (installed below), connects widgets (spin boxes, combo boxes, push buttons) to callbacks in Python. QtCreator (installed below) will be used to lay out the widgets within the GUI.)*
2.1) Download the free community version of **Qt 4.8.6** from
http://download.qt.io/archive/qt/4.8/4.8.6/.
Please use this version 4.8.6 specifically, not the latest version of Qt.

Step 3: PySide 1.2.2 *(PySide is a set of Python bindings for Qt)*
3.1) Only after you have installed Qt 4.8.6, follow by downloading and installing the binaries of **PySide 1.2.2** for your platform from https://pypi.python.org/pypi/PySide/1.2.2.
3.2) If your installation of PySide fails to install, go back to Step 2 and reinstall Qt 4.8.6, then try to re-install PySide (the correct version of Qt must be present when installing PySide).

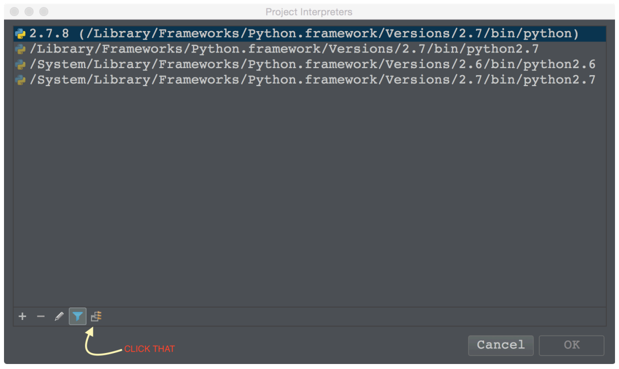
Step 4: Git and Github *(version control system and web-based project repository)*
4.1 Download and install the **Git** native app from https://help.github.com/articles/set-up-git
4.2 Set up your own Git account. Follow the instructions on their webpage.
4.3 If not already familiar with Git, read and understand the Getting-Started-Git-Basics:
http://git-scm.com/book/en/Getting-Started-Git-Basics
4.4 Follow **kentrosaurus** and this class's **MayaPy** project on GitHub.
4.5 Create your own path to a local Git repository on your computer, such as:
Users/yourHomeFolder/Documents/Maya/code
4.6 Download from GitHub the following repositories:

Nimble:	https://github.com/sernst/Nimble
PyAid:	https://github.com/sernst/PyAid
PyGlass:	https://github.com/sernst/PyGlass

and place them in your local Maya/code folder as three subfolders, one for each repository (Nimble, PyAid, and PyGlass), i.e.:
Users/yourHomeFolder/Documents/Maya/code/Nimble
Users/yourHomeFolder/Documents/Maya/code/PyAid
Users/yourHomeFolder/Documents/Maya/code/PyGlass

Step 5: PyCharm 4 *(integrated development environment, or IDE)*

5.1) Download and install the free community version of PyCharm 4 from:
https://www.jetbrains.com/pycharm/download/
5.2) Launch PyCharm and in **preferences>Project Interpreter,** at the top of the popup window, select as the Python interpreter your Python version 2.7.8 (but don't hit OK just yet).
5.3) While still in the preferences popup window for selecting the Project interpreter, add the following site packages using the + symbol at the bottom of this window: **numpy**, **pip**, **Markdown**, **MarkupSafe**, **SQLAlchemy**, **alembic**, **appdirs** (not needed if you are using Linux), and **distribute**, **requests**, **setuptools**, and finally **wsgiref**. Now, finally, hit OK. Note that if you have trouble with with pip on mac or windows, you can get some suggestions from:
http://stackoverflow.com/questions/17271319/installing-pip-on-mac-os-x
http://stackoverflow.com/questions/4750806/how-to-install-pip-on-windows
5.4) Reopen the project interpreter settings in preferences and confirm that you have successfully added these site-package paths to numpy, pip, Markdown, etc.
5.5) Again, open **Preferences>Project Interpreter** and find the Project Interpreter combo box near the top of that window. Be sure you have selected the 2.7.8 interpreter (in case you have others also loaded). Next, look for the downward arrow on the right side of that combo box and choose **Show All**. That will pop up another window that has the selected interpreter at the top and at the bottom on the left are five buttons (+, -, and three icons). If you mouse over the *right-most* icon, the tool tip says "Show paths for the selected interpreter". Click that icon (in the screen shot to the right, I have pointed to that icon). When you click that icon you'll see another popup window listing all Interpreter Paths. You need to add paths to PyGlass, Nimble, and PyAid (see those in **bold** below):



```
2.7.8 (/Library/Frameworks/Python.framework/Versions/2.7/bin/python)
/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7
/System/Library/Frameworks/Python.framework/Versions/2.6/bin/python2.6
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7
```

file:///Library/Python/2.7/site-packages/pip-1.4.1-py2.7.egg
file:///Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7
file:///Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/plat-darwin
file:///Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/plat-mac
file:///Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/plat-mac/lib-scriptpackages
file:///Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/lib-tk
file:///Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/lib-dynload
file:///Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages
file:///Library/Python/2.7/site-packages
file:///Users/kent/Documents/Maya/code/PyGlass/src
file:///Users/kent/Documents/Maya/code/Elixir/src
file:///Users/kent/Documents/Maya/code/Nimble/src
file:///Users/kent/Documents/Maya/code/PyAid/src

Then click OK. Now PyCharm will have access to those site-packages.

Step 6: Nimble *(remote communication bridge to access Maya's Python interpreter)*

Nimble was written by Scott Ernst and available for this course. The GitHub address for Nimble is: https://github.com/sernst/Nimble. From the description:

> *"Nimble enables transparent execution of Maya commands, both built-in and custom-made, from an external Python interpreter no matter the version or platform differences between this external interpreter and Maya's built-in interpreter. At the same time Nimble can be run within the Maya Python interpreter directly, so code developed with Nimble works in both external and Maya Python interpreters in a completely transparent fashion. …"*

6.1) Find your **Maya.env** file. For example, on Mac OS this file is likely at:

```
Users/yourHomeFolder/Library/Preferences/Autodesk/maya/2014-x64/Maya.env
```

6.2) Edit your Maya.env file to add a **PYTHONPATH** pointing to **Nimble/src** and to **PyAid/src**. For Windows, this is:

```
PYTHONPATH=c:\\path\to\Nimble\src;c:\\path\to\PyAid\src
```

and on OS X and Linux:

```
PYTHONPATH=/path/to/Nimble/src:/path/to/PyAid/src
```

For example, my Maya.env file on OS X has:

```
PYTHONPATH=/Users/kent/Documents/Maya/code/Nimble/src:/Users/kent/Documents/Maya/code/PyAid/src
```

Step 7: QT Creator 2.8.1 *(GUI layout tool)*

1) Download and install Qt Creator 2.8.1 from:

http://download.qt.io/official_releases/qtcreator/2.8/2.8.1/

Step 8: Integration and startup

8.1) Create a folder to contain your PyCharm projects, e.g., on OS X I have:

```
/Users/kent/PycharmProjects
```

8.2) Log in to github, and download (as a zip or clone in desktop) the project MayaPy:

https://github.com/Kentrosaurus/MayaPy

8.3) Move that local repository to your PycharmProjects and rename its folder MayaPy.

8.4) Launch PyCharm and create a project pointing to PycharmProjects/MayaPy.

8.5) In PyCharm, examine the project and make sure you have the MayaPy/src directory marked as *source root*.

8.6) Start Maya, open script editor, be sure to select the Python tab, and enter and run the following:

```
import nimble
nimble.startServer(0)
```

Since you will need to do that operation repeatedly, select those two commands in the Python script editor, and middle mouse drag them to your shelf and hold the mouse there for a moment to make a clickable shelf entry. You may wish to then open the shelf editor to give it a short name (e.g., "NimS"). An argument of 2 instead of 0 would create verbose mode responses within the script editor, which will be good for debugging later. So you might want to put that on the shelf as well.

8.7) Once the Nimble server is running on Maya's interpreter, launch PyCharm and build MayaPy and run the MayaPy application. When all compiles you should be able to launch the main window which displays three buttons (for assignments 1 and 2 and to start Nimble).

8.8) Given that Maya is already waiting Nimble communications, click the button in the MayaPy application to start the Nimble server; which should turn green.
8.9) In the MayaPy application, click the *assignment 1* button. When that window pops up, you will see a button to create a cylinder in the scene. Click that button. If you get a cylinder to appear in Maya, then you have a complete path from PyCharm's Python interpreter launching a GUI which has a button that calls back to your Python app, which then runs a function that tells Maya to create a cylinder. Hopefully, happiness will ensue.