PixelMath 2015 Release Notes

(Notice added in Summer, 2021) We have found that PixelMath2015 should be used with Java versions 11 or lower. For example, do not use it with Java 16; the Calculator "Compute" button won't work, and the Python "Execute Buffer" won't work. The Microsoft builds of Java 11 for Window, Mac, and Linux should work fine with PixelMath2015, and they are available at

https://docs.microsoft.com/en-us/java/openjdk/download

January 5, 2015:

PixelMath 2015 contains a number of improvements over PixelMath 2012. Some are bug fixes and stability enhancements. Others are new features to improve convenience and flexibility.

Transparency.

While PixelMath has always had as a design goal to make image pixel information easy to see, other aspects of images such as how much memory they consume has been difficult to see. Feedback messages from PixelMath used to be shown mostly in the Java Console. Now there is a PixelMath console accessible from the launchpad. The two most recent lines of messages are also shown directly on the launchpad.

Image memory usage information is now more accessible. An estimate of an image's memory usage can be found via each Image Frame's View menu: View -> Size Info. The size required by an image depends on whether Undo is allowed. By default, Undo is allowed, unless the image is of type complex color. But undoability can be turned off on the Representation menu. A rule of thumb for estimating memory usage is 10 bytes per pixel for the image in an Image Frame, plus 2 bytes per window pixel for displaying it on the screen. This is because PixelMath typically has up to 5 different representations of each image in memory at a time: the Java Image object which may or may not be compressed, an integer array for accessing RGB values, another integer array for the undo image, an offscreen image representing what's to be shown (which could actually be much larger than the image, if the window is large but the image is small), and a Swing-internal representation of the display of the image.

More Powerful Python Editing Features

The new features include code completion, undo/redo, find/replace, and popup menus for copy/paste. The syntax highlighting for Python is now more complete. The code completion feature offers the user a way to access the calling signatures for the most important PMPixelMath functions in the Applications Program Interface, such as pmNewComputedImage, pmOpenImage, etc.

Memory Management Features

Three memory-management facilities are now built into PixelMath. One is simply a means of querying the Java Virtual Machine's "Runtime" facility to find out how much total memory and free memory it has. Another feature is an automatic check, each time a new image is loaded from a file, of whether its use of memory would account for more than 30% of free memory. If so, the system pops up a dialog box suggesting to the user that an automatic reduction of the size of the image be performed. The system normally proposes reduction by a factor of 2 in each direction, meaning that the memory required by the image is reduced by 75%. A third feature is an automatic system for performing or suggesting that Undo be turned off, particularly for the largest kinds of images: complex color images. In addition to these features, the software now makes it easier for the Java garbage collector to reclaim the memory used by image frames that have been deleted by the user.

Support for Making Presentations

A new display mode is available for Image Frames: Full-screen mode. In this mode, an image is shown without menus, status line, window title bar, or toolbars. However, it can still be zoomed in or out, and the options to show pixels and show numbers are still available. This means that a PixelMath Python program can be easily written that creates a visually cleaner kind of presentation than could be easily presented before. Here is an example of using Python to show a small image in full-screen mode for 3 seconds:

```
def testFullScreen():
  wn = pmNewComputedImage("Purple grad.", 32, 256, "RGB(x * 8, 0, y)")
  pmFullScreen(wn, True)
  pmSleep(3000)
  pmFullScreen(wn, False)
```

Also to simplify making presentations, four new API functions are provided to support drawing of text on images. They are the following.

```
g=pmGetGraphics(wn) #returns a Graphics2D object g. Here wn is an int window number.
f=pmMakeFont(family, style, size) # family is a string such as "Helvetica"; style and
      size are ints. style=1 for normal.
pmDrawString(g, theString, x, y, f) # g is a Graphics2D object. theString is the text
      to be drawn. x and y tell where to position the upper-level corner of the
      text.  f is a font made with pmMakeFont.
pmUpdateImageNumbers(wn) # needed to get the image displayed properly after the
      graphics calls. wn is the int window number.
```

Another feature related to giving presentations with PixelMath is the ability to configure runnable demonstrations that can be double-clicked in the operating system or launched from the OS command line. In Windows, for example, you can create one or more folders of demos. Such a folder would contain a copy of PixelMath2015.jar, all the images and Python scripts needed for the demos, and one

Windows .bat file for each demo. Here is an example of such a .bat file. (Let's call this one `demoMona.bat`.)

```
ECHO Starting PixelMath Demonstration with the Mona Lisa Image ...
java -jar PixelMath2015.jar --init=showMona.py
pause
```

The files needed in the folder for this demo would be: PixelMath2015.jar, demoMona.bat (containing the above; only the line starting with Java is really necessary), showMona.py (the lines of one such possible file are shown below as an example), and the image file Mona256.jpg needed in this example.

```
# showMona.py :
print "Let's welcome Mona Lisa!"
wn = pmOpenImage(0, "Mona256.jpg")
for i in range(2): pmZoom(wn, 128, 128)
pmFullScreen(wn, True)
pmSleep(5000) # wait 5 sec.
pmFullScreen(wn, False)
for i in range(2): pmUnzoom(wn, 128, 128)
```

Other Enhancements and Fixes

The PixelMath Calculator now has an additional help-menu option: bringing up an electronic copy of the Quick Reference PixelMath Calculator Formulas sheet. Examples from this sheet can easily be selected, copied, and pasted into the calculator.

PixelMath2012 contained a bug limiting the ability to display small images properly. Now the user can view and zoom down on an image as small as a single pixel. The range of allowable zoom factors has been increased both at the low end and the high end.

S. Tanimoto, University of Washington, Dept. of Computer Science and Engineering