Porting a Python Program to Android

Tierra Montgomery
Fayetteville State University
Faculty Mentor: Albert Chan
Fayetteville State University

Abstract
In this article, we present the results of porting a desktop Python program (that is, a classroom example for entry level programming course) to the Android mobile platform by using Scripting Layer for Android (SL4A) and Python for Android (P4A). We show that the mobile platform can be used successfully to study interests in entry level programming courses while maintaining the choice of programming languages for simplicity. We also enhanced our ported program with Android specific features to further arouse students' interest.

1. Introduction

There are two goals in the algorithm design process: to create a correctly behaving algorithm and to make it as effective as possible. With today's abundant computing powers, the latter goal is usually not much of a concern; except in some very difficult problems, see [3][4] for example. The first goal, on the other hand, remains a concern. Programmers need to spend a large amount of time in coding and/or testing to create a correctly behaving algorithm. A good foundation in programming courses may provide good start on understanding algorithm design and it would be convenient for students to be able to practice wherever they are. In this study, we look at the possibility of using mobile devices as a platform to teach programming.

2. Background Information

2.1. Python

Programming languages have a large impact on software development. In theory, once an algorithm has been developed and verified to be correct, it should be straightforward to translate it into software using a language of the programmer's choice. However, the choice of programming languages may be influenced by many factors: the client, the employer, the programmer, or even the targeted software platform. The client may dictate that the software is to be developed in a particular language: for example, the United States Department of Defense once required all software developed for them to use the Ada programming language [11]. The programmer may be more familiar with one language than another: for example, a large percentage of students at Fayetteville State University choose to use Python to code their senior projects because that is the language with which they are most familiar. And finally, not all languages are available for all platforms. This is usually the case for smartphone platforms that only support limited choices of programming languages due to hardware limitations.

Python is a language designed to be easy to learn and easy to use [10]. It is very suitable for use in teaching introductory programming courses because some of its syntax enforces good programming styles (e.g., the use of indentation has a meaning in Python programs compared to most other languages in which indentation is used merely for to enhance readability). Also, the non-compiling (that is, Python is mostly an interpreted language) approach makes it easy for students to see immediate results of their code, hence allowing positive feedback to the students' learning experience. In fact, many institutions have converted their entry-level programming courses to use Python. Some institutions (such as Fayetteville State University) have been using Python for the entire entry level programming courses up to CS2 (Data Structures and Algorithms) [1]. There are also numerous textbooks designed for introducing Python as a first language in programming for these entry-level courses (for example, see [7] and [12]).

2.2. *Android*

Google, along with other handset manufacturers, formed the Open Handset Alliance (OHA) in 2007 [5]. The main goal of the alliance was to develop the next generation wireless platform. Unlike other platforms available at the time, the goal of OHA was the creation of an open platform that was freely available. In the following year, OHA introduced the Android platform and launched a beta program for developers. The first Android handset began shipping in late 2008. Today, the latest version of Android is version 4 [8].

Before Android, almost all mobile platforms available were proprietary and controlled by a single company (for example, iOS by Apple, Blackberry OS by Research-In-Motion, and Windows CE by Microsoft). Probably the only exception was the Java 2 Mobile Edition (J2ME), which was designed to target the mobile platform. Although many proprietary mobile platforms are actually J2ME-based (one example is the Blackberry OS), it has not achieved the popularity as planned. Developers for mobile devices were still required to work with proprietary systems. This made it very difficult to develop software to target multiple hardware platforms. The availability of Android platform broke this monopoly. Not only do phone manufacturers now have an additional choice in phone OS, this new choice is also open source – meaning that anyone interested can look into the design to see how it works and maybe even contribute to the development of the OS.

Android is based on a specialized Java virtual machine (the Dalvik Virtual Machine, or DVM), which is implemented on top of a Linux platform. Almost all Android applications are written in Java. However, the main difference between Android Java applications and desktop Java applications (and mobile applications for other wireless platforms) is that Android also utilizes XML to describe the layouts of the applications, therefore decoupling the user interfaces from the applications' functionalities. One obvious advantage is that it is now easier to modify the layouts of the applications without affecting the code that controls the functionalities of the software.

2.3. *Scripting Layer for Android*

The Scripting Layer for Android (SL4A) was an experimentation environment developed by Google. The purpose of SL4A is to allow programmers to write programs using their scripting languages of choice. Release 3 of SL4A required the written programs (the scripts) to run within the environment.

Release 4 and later allows the developers to package the scripts in a way that looks like other native Android applications [5].

SL4A is a generic scripting platform for Android [6]. It supports several scripting languages, including BeanShell (an interpreted version of the Java language), JRuby, Linux shell, Lua, Perl, PHP, Python, and Rhino. Like the Android platform, SL4A, along with the interpreters it supports, is available as a free download from Google [8].

3. Methodology

In this article, we describe our project on porting and enhancing existing Python programs originally developed for the desktop environment to the Android platform using SL4A. One of the purposes of this project was to study the feasibility of building a set of curriculum material to introduce mobile programming using Python in entry level programming courses.

Each student in the class was assigned to work on one program; in this report we describe the findings from the project to move desktop Python programs to Android. Other students in the class did different projects; for example, other students investigated porting a desktop Blackjack game using SL4A.

We began with a desktop Python program – a text-based Mastermind game – from a textbook used in a software development course [7]. We first moved the program to SL4A and verified that it worked as described in the textbook. This is not a difficult step as the Python version in SL4A (Python for Android, or P4A) contains a virtually complete subset of the normal Python distribution [1]. We note that the program we were porting is a text-based program as the graphics library commonly used in desktop Python (e.g., Tkinter) is not available in P4A due to hardware differences.

Next we enhanced the program with two features that are only available on the mobile platforms. The first was to text-to-speech (TTS) technology to make the mobile devices “say” the text displayed on the screen; the other was adding dialog boxes.

3.1. Text-To-Speech (TTS)

TTS is a feature of the Android platform. This feature allows the application to “speak” the content of choice through automated human voice. This is basically a safety feature. One example of how this feature is utilized is to let the Android smart phone “speak”t the content of incoming text messages. This frees the user from looking at the phone screen in order to know what the text messages are about.

SL4A exposes the TTS feature to scripts (of all languages supported) through a special Android object. In Python for Android (P4A), this object is available in the android package. Once created, we can use the “ttsSpeak” method of this Android object to say any contents of our choice [1]. The process is simple, we searched though our code to find all instance of the Python “print” statements, and added a corresponding “ttsSpeak” method call after every found “print” statement.

There was one difficulty in this procedure. Originally, our program printed out a list of color patterns using color initials (for example “RGBYW” to represent a color pattern of red-green-blue-yellow-white). The use of initials in text output is acceptable in the text-based output (and may be essential as it makes the output easier to comprehend), but becomes a problem as the TTS engine treats the string as a word. Because most color initials are consonants and cannot form a word, the TTS produces a very strange sound and usually cannot be understood. Even in the rare case that the initials actually form a word (e.g. “PIG” – stands for “purple-ivory-green”), TTS will not produce the right sound. In this example, it will say “pig” (as a word). What we want is to have TTS to spell out the initials as letters, so it should actually say “P-I-G” instead of “pig”.

To solve this problem, we put a space character between each letter of the pattern. In this way, TTS will treat each letter as a single word, and by default, it will spell out and “say” only that letter.

Using TTS in the scripts is a useful feature as some of the users may be driving a car when they use the program. Of course, the users should not be playing games while driving, but the same technology can also be applied to other software developed under SL4A.

3.2. Dialog Boxes

While TTS is used to handle the output in the mobile environment, dialog boxes can be used to handle both the input and visual output. Even in the desktop environment, dialog boxes are often used to give user important information that the user needs to respond to before allowed to move forward, or to collect input from the user.

The advantage of using dialog boxes is twofold. First, it can reduce or eliminate keyboard input by providing buttons and/or combo boxes; these widgets will also make the program more user-friendly. Second, by reducing or completely eliminating keyboard input, it can reduce or eliminate user errors.

The advantage of using dialog boxes in mobile platforms is more obvious. Most phones have a small screen size (4- to 5-inch for phones, 6- to 10-inch for tablets) compared to the desktop environment (12- to 21-inch or larger). If we take into consideration that many phones do not come with a built-in keyboard and have to depend on a virtual keyboard, it is clear that the virtual keyboard, though it usually takes up half of the screen, will be too small for the fingers of many users. Thus, it would be inconvenient to have the users type input responses to the application through keyboard. Using dialog boxes can partially solve this problem, although it still cannot solve all the problems of requiring user input in multi-tasking situations.

There are several ways to use dialog boxes in SL4A, all through the same Android object described in the last subsection. In fact, the Android object is a gigantic object that exposes almost all Android features (called Android Facades [6]) to the supported scripting languages in SL4A.

The easiest way to use a dialog-like service is to use the "makeToast" API call. This will show an alert box briefly on screen, and the alert box will be dismissed after some pre-determined delay. However, we did not use the "makeToast" API call in our program. We used three dialog-related API calls in our program. The "getInput" API call will create an input dialog to allow the user to enter his/her name. This will require the user to use the keyboard to type a response as there is no other way to obtain this information easily. The same method call, when used in conjunction with the "dialogSetItems", can also be used to create a dialog box with choices.

The "dialogCreateAlert" API call creates an alert box to display information to the user. Unlike the "makeToast" method, the alert box will not be dismissed until the user clicks a button.

4. Testing

Testing was mainly accomplished using the Android simulator under the Windows environment. The Android simulator that comes with the Android Development Kit (ADK) is a fairly complete simulation of an Android phone and/or tablet. Because we do not use special features (such as GPS, networking, accelerometer, camera, or the phone functionality), we were able to test the software completely in the simulator until it was stable enough to be tested on real device (it is well known that, unlike a desktop or laptop computer, it is very difficult to restore a smart phone to the factory state once it has been corrupted).

Once the software was running consistently, we downloaded the software to a real Android phone (HTC rhyme, running Android 2.3 Gingerbread) and made sure it performed as expected on the real phone. We also made sure it did not interfere with other software and/or functionalities on the phone. In order for the game to be considered stable and performing as expected, the game had to be able to be played several times on the phone without the game or phone crashing. Figure 1 shows our testing procedure in the Android emulator.

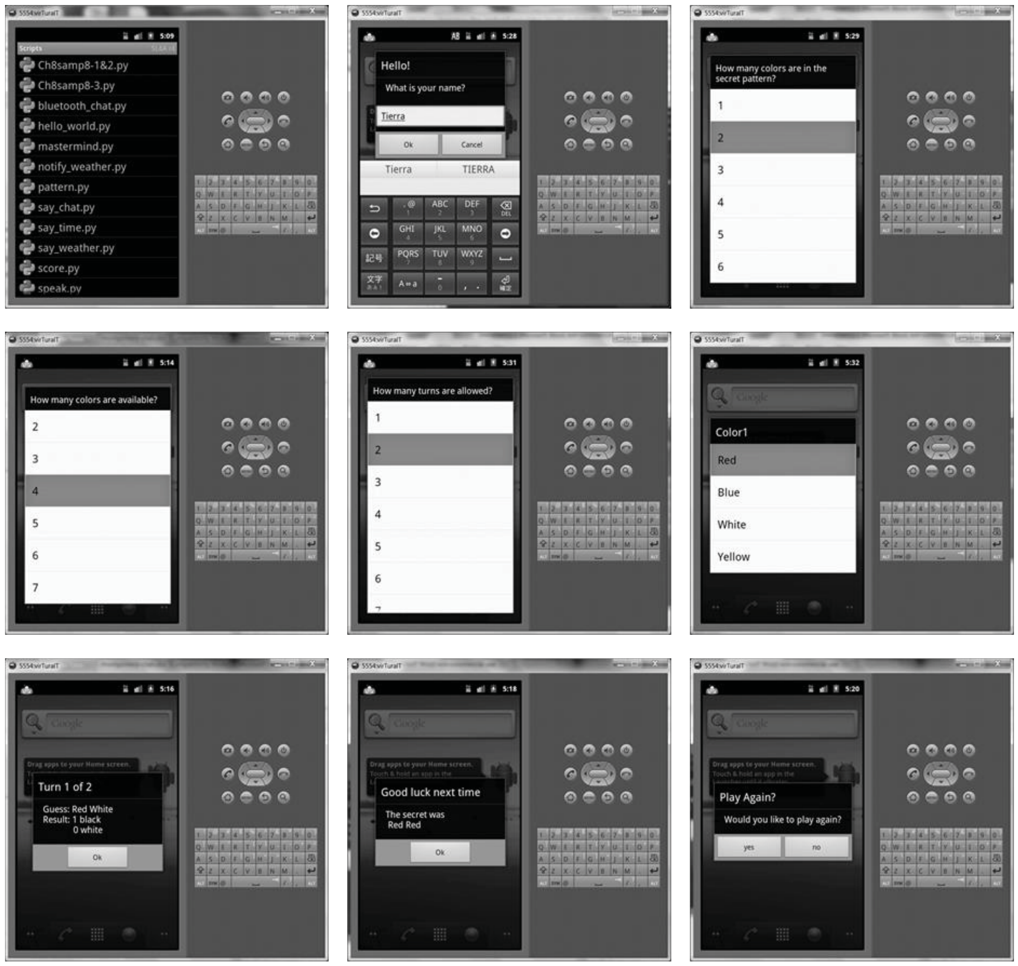
5. Response to the Project and Future Work

This project and the resulting software have been presented and demonstrated in several intra-college symposia [9]. The presentations were well-received and generated discussion and suggestions. The initial project was finished within five weeks. With follow-up improvements, we were able to finish in one year. This shows that the procedure of porting Python game to the Android platform can be efficient and effective. . Further, we did not see any suspected interference on the phone.

We believe most classroom examples can be similarly ported to the SL4A platform without many problems so that they can be used as successful teaching tools. Probably the only exceptions are those with graphics. SL4A does not support graphics at this time, and we do not know if Google has any plans to include graphical support in SL4A.

One way to enhance the project would be to develop a graphical API for P4A, and provide a bridge interface binding to commonly used classroom graphical packages to allow use the graphical API to use this graphics. One of the graphical packages used in an entry level classroom is Calico. Within Calico, many packages can be used to create various objects. For instance, during an entry level class, a project consisted of creating a class, using Python, that allowed one to input bank number and information would show in a box. But utilizing that graphical package provides more "creativity" with a project.

In this article, we have described our project of porting a desktop Python program (that is a classroom example for entry level programming course) to the Android mobile platform by using Scripting Layer for Android (SL4A) and Python for Android (P4A). The experience we obtained was positive and exciting. However, while most of the classroom examples (those that involve only text input and output) can easily be ported to the Android platform, others (especially those involve using graphics) still need work to make the porting experience seamless.



Hello!
What is your name?
Tierra

How many colors are in the secret pattern?

How many colors are available?

How many turns are allowed?

Drag apps to your Home screen.

Turn 1 of 2
Guess: Red White
Result: 1 black
0 white

Good luck next time
The secret was
Red Red

Play Again?
Would you like to play again?

Figure 1: Steps involved in the testing process. Upper panel from left to right: SL4A showing a list of available scripts; MasterMind game begins with request for user to enter name and number of pegs in pattern. Middle panel from left to right: Program requests colors, number of rounds and set of color choices. Bottom panel from left to right: Result displayed after colors are entered, process repeats across rounds, and at end of game, player has choice to play another set or cancel the game.

References

1. M. Almeida et. al, *Computer Science Course Syllabi*, Fayetteville State University, 2013, available at URL: http://www.uncfsu.edu/macsc/sy labi/syllabi-computer-science/ as of August 2013.

2. Anonymous (posted with alias as *numbercruncher*), *A Very Silly Introduction to Android SL4A's Text-To-Speech Functionality*, 2012, available at URL: http://python for analysts.pythonblogs.com/211_p thon for analysts/archive/1257_a_very_silly_introduction_to_android sl4as_text to speech_functionality.html as of August 2013.

3. A. Chan, *Solving Large Graph Problems*, Ph.D. Dissertation, 2003.

4. A. Chan and F. Dehne, *A coarse grained parallel algorithm for max mum weight matching in trees*, in Proceedings of 12th IASTED International Conference Parallel and Distributed Computing and Systems (PDCS'2000), 134-138, 2000.

5. S. Conder and L Darcey, *Android Wireless Application Development*, 2nd edition, Addison-Wesley Professional, 2011.

6. P. Ferrill, *Pro Android Python with SL4A*, Apress, 2011.

7. M. Goldwasser and D. Letscher, *Object-Oriented Programming in Python*, Prentice Hall, 2008.

8. Google, *Official SL4A Website*, available at http://code.google.com/p/ android-scripting as of August 2013.

9. T. Montgomery, *Android Mobile Computing*, presented in the 2012 State of North Carolina Undergraduate Research and Creativity Symposium (SNCURCS 2012), available at http://prezi.com yigql99hlbcx/?utm_campaign=share&utmmedium=copy&rc=ex0share as of August 2013.

10. Python Foundation, *Official Python Website*, available at http://www python.org as of August 2013.

11. J. Rosen, *The Ada Paradox(es)*, Ada Letters, ACM SIGAda, Vol. 24, No. 2, pp. 28-35, August 2009.

12. J. Zelle, *Python Programming: An Introduction to Computer Science*, 2nd edition, Frankin, Beedle & Association, 2008.