ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 2, pp: (303-310), Month: April-June 2014, Available at: www.researchpublish.com

Future Practicability of Android UI Development with Monkey and Monkey Runner Tools

Shruti Mukherjee

Computer Science and Engineering
Narula Institute of Technology, Agarpara, kolkata,West bengal, India

***Abstract*: Today we are in the major technological verge and in this situation we have to choose a side from PC devices and Mobile devices. As known from the market reviews users of different ends mostly prefer Mobile devices for their daily usage and because of that mobile technology is growing at a vast range and very rapidly. And the most triggered software of this technology we all know is Android .The possibility or future scopes for Android are beyond imagination. The large and speedy growth of Android always makes Developers work in different aspects and explore more and more about this technology. In This paper I propose the future of Android Development on the prospect of MONKEY and MONKEYRUNNER. As known by many developers Java is the main coding language for Android application development, but there are many more unknown things about Monkey and Monkey Runner which could give Android UI Development a new path from the developer to the end-users. They can be those components which can make Android grow faster and give more directions.**

***Keywords*: Android application development, Monkey Tool, Android Monkey runner, Future development with monkey runner.**

I. INTRODUCTION

Android has some built in UI testing tools. These tools can be used for automated UI testing. However the tools are not so simple to use. Monkey and MonkeyRunner are just 2 examples of them which are tends to be very similar by name and by looks but actually they are two different things, much apart from each other.

The Monkey is a program that runs on your emulator or device and generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events. You can use the Monkey to stress-test applications that you are developing, in a random yet repeatable manner. The Monkey is a command-line tool that that you can run on any emulator instance or on a device. It sends a pseudo-random stream of user events into the system, which acts as a stress test on the application software you are developing.

On the other hand The MonkeyRunner tool provides an API for writing programs that control an Android device or emulator from outside of Android code. With MonkeyRunner, you can write a Python program that installs an Android application or test package, runs it, sends keystrokes to it, takes screenshots of its user interface, and stores screenshots on the workstation. The MonkeyRunner tool is primarily designed to test applications and devices at the functional/framework level and for running unit test suites, but you are free to use it for other purposes. The MonkeyRunner tool is not related to the UI/Application Exerciser Monkey, also known as the monkey tool. The monkey tool runs in an adb shell directly on the device or emulator and generates pseudo-random streams of user and system events. In comparison, the MonkeyRunner tool controls devices and emulators from a workstation by sending specific commands and events from an API. These are the main works of them currently in progress.

II. QUALITY EVOLUTION

- ***Monkey (aka. UI/Application Exerciser Monkey)***

This tool is a command line based tool that can be primarily used to stress test your application UI. It is the simplest tool to use. The Monkey includes a number of options, but they break down into four primary categories:

- Basic configuration options, such as setting the number of events to attempt.
- Operational constraints, such as restricting the test to a single package.
- Event types and frequencies.
- Debugging options.

When the Monkey runs, it generates events and sends them to the system. It also *watches* the system under test and looks for three conditions, which it treats specially:

- If you have constrained the Monkey to run in one or more specific packages, it watches for attempts to navigate to any other packages, and blocks them.
- If your application crashes or receives any sort of unhandled exception, the Monkey will stop and report the error.
- If your application generates an *application not responding* error, the Monkey will stop and report the error.

Depending on the verbosity level you have selected, you will also see reports on the progress of the Monkey and the events being generated.

You can launch the Monkey using a command line on your development machine or from a script. Because the Monkey runs in the emulator/device environment, you must launch it from a shell in that environment. You can do this by prefacing a db shell to each command, or by entering the shell and entering Monkey commands directly.

The basic syntax is:

```
$adb shell monkey [options] <event-count>
```

With no options specified, the Monkey will launch in a quiet (non-verbose) mode, and will send events to any (and all) packages installed on your target. Here is a more typical command line, which will launch your application and send 500 pseudo-random events to it:

```
$adb shell monkey –p your package.name –v 500
```

The monkey tool can also be used to run a specific set of commands on the application. However it is easier to use the MonkeyRunner for this purpose.

On the connected device (to run on emulator simply replace -d with -e)

```
./adb -d shell monkey -p package_name --port 1080 &
./adb -d forward tcp:1080 tcp:1080
telnet localhost 1080
```

Now the following will be printed on cmd line..

```
Trying ::1...
Trying 127.0.0.1...
Connected  to  localhost.
Escape character is '^]'.
```

Now you can type in your instructions

```
>tap 150 200
```

you can write all instruction into a script (script.txt) as below..

```
# monkey
tap 100 180
type 123
tap 100 280
press DEL
press DEL
press DEL
press DEL
press DEL
press DEL
press DEL
press DEL
type -460.3
```

now run..

```
./adb -d shell monkey -p package_name --port 1080 &
./adb -d forward tcp:1080 tcp:1080
nc localhost 1080 < script.txt
```

The table below lists all options you can include on the Monkey command line.

Category	Option	Description
General	--help	Prints a simple usage guide.
	-v	Each -v on the command line will increment the verbosity level. Level 0 (the default) provides little information beyond start-up notification, test completion, and final results. Level 1 provides more details about the test as it runs, such as individual events being sent to your activities. Level 2 provides more detailed setup information such as activities selected or not selected for testing.
Events	-s <seed>	Seed value for pseudo-random number generator. If you re-run the Monkey with the same seed value, it will generate the same sequence of events.
	--throttle <milliseconds>	Inserts a fixed delay between events. You can use this option to slow down the Monkey. If not specified, there is no delay and the events are generated as rapidly as possible.
	--pct-touch <percent>	Adjust percentage of touch events. (Touch events are a down-up event in a single place on the screen.)
	--pct-motion <percent>	Adjust percentage of motion events. (Motion events consist of a down event somewhere on the screen, a series of pseudo-random movements, and an up event.)
	--pct-trackball <percent>	Adjust percentage of trackball events. (Trackball events consist of one or more random movements, sometimes followed by a click.)
	--pct-nav <percent>	Adjust percentage of "basic" navigation events. (Navigation events consist of up/down/left/right, as input from a directional input device.)
	--pct-majornav <percent>	Adjust percentage of "major" navigation events. (These are navigation events that will typically cause actions within your UI, such as the center button in a 5-way pad, the back key, or the menu key.)
	--pct-syskeys <percent>	Adjust percentage of "system" key events. (These are keys that are generally reserved for use by the system, such as Home, Back, Start Call, End Call, or Volume controls.)
	--pct-appswitch <percent>	Adjust percentage of activity launches. At random intervals, the Monkey will issue a startActivity() call, as a way of maximizing coverage of all activities within your package.
	--pct-anyevent <percent>	Adjust percentage of other types of events. This is a catch-all for all other types of events such as keypresses, other less-used buttons on the device, and so forth.
Constraints	-p <allowed-package-name>	If you specify one or more packages this way, the Monkey will *only* allow the system to visit activities within those packages. If your application requires access to activities in other packages (e.g. to select a contact) you'll need to specify those packages as well. If you don't specify any packages, the Monkey will allow the system to launch activities in all packages. To specify multiple packages, use the -p option multiple times — one -p option per package.
	-c <main-category>	If you specify one or more categories this way, the Monkey will *only* allow the system to visit activities that are listed with one of the specified categories. If you

		don't specify any categories, the Monkey will select activities listed with the category Intent.CATEGORY_LAUNCHER or Intent.CATEGORY_MONKEY. To specify multiple categories, use the -c option multiple times — one -c option per category.
Debuggin g	--dbg-no-events	When specified, the Monkey will perform the initial launch into a test activity, but will not generate any further events. For best results, combine with -v, one or more package constraints, and a non-zero throttle to keep the Monkey running for 30 seconds or more. This provides an environment in which you can monitor package transitions invoked by your application.
	--hprof	If set, this option will generate profiling reports immediately before and after the Monkey event sequence. This will generate large (~5Mb) files in data/misc, so use with care. See Traceview for more information on trace files.
	--ignore-crashes	Normally, the Monkey will stop when the application crashes or experiences any type of unhandled exception. If you specify this option, the Monkey will continue to send events to the system, until the count is completed.
	--ignore-timeouts	Normally, the Monkey will stop when the application experiences any type of timeout error such as a "Application Not Responding" dialog. If you specify this option, the Monkey will continue to send events to the system, until the count is completed.
	--ignore-security-exceptions	Normally, the Monkey will stop when the application experiences any type of permissions error, for example if it attempts to launch an activity that requires certain permissions. If you specify this option, the Monkey will continue to send events to the system, until the count is completed.
	--kill-process-after-error	Normally, when the Monkey stops due to an error, the application that failed will be left running. When this option is set, it will signal the system to stop the process in which the error occurred. Note, under a normal (successful) completion, the launched process(es) are not stopped, and the device is simply left in the last state after the final event.
	--monitor-native-crashes	Watches for and reports crashes occurring in the Android system native code. If --kill-process-after-error is set, the system will stop.
	--wait-dbg	Stops the Monkey from executing until a debugger is attached to it.

These are the contents of a monkey tool.

- ***Monkey Runner***

It is a tool which provides an API for writing programs that control an android device from outside of android code. You can write python programs to test the applications on one or more devices and/or emulators. You can do following things and more with Monkey Runner.

- Installs an application or test package
- Runs an application
- Send keystrokes or touch events to it
- Take screen shots of the user interface

- Store screen shots on your workstation

You can do all those things from your PC or laptop remotely.
This is primarily designed to test applications and devices at the functional/framework level and for running unit/functional test suites.

Unique features of MonkeyRunner includes

- Multiple device control
- Functional testing with screen capture
- Regression testing – run an application against a particular result
- Extensible automation

It uses Jython, an implementation of python that uses the Java programming language.

The Monkey Runner API is contained in three modules in the package com. android. Monkey runner:

- MonkeyRunner: A class of utility methods for monkeyrunner programs. This class provides a method for connecting monkeyrunner to a device or emulator. It also provides methods for creating UIs for a monkeyrunner program and for displaying the built-in help.
- MonkeyDevice: Represents a device or emulator. This class provides methods for installing and uninstalling packages, starting an Activity, and sending keyboard or touch events to an application. You also use this class to run test packages.
- MonkeyImage: Represents a screen capture image. This class provides methods for capturing screens, converting bitmap images to various formats, comparing two MonkeyImage objects, and writing an image to a file.

MonkeyDevice

A monkeyrunner class that represents a device or emulator accessible by the workstation running monkeyrunner. This class is used to control an Android device or emulator. The methods send UI events, retrieve information, install and remove applications, and run applications.

You normally do not have to create an instance of MonkeyDevice. Instead, you use Monkey Runner. Wait For Connection () to create a new object from a connection to a device or emulator. For example, instead of using:

newdevice=MonkeyDevice()

you would use:

newdevice=MonkeyRunner.waitForConnection()

MonkeyImage

A monkeyrunner class to hold an image of the device or emulator's screen. The image is copied from the screen buffer during a screenshot. This object's methods allow you to convert the image into various storage formats, write the image to a file, copy parts of the image, and compare this object to other MonkeyImage objects.You do not need to create new instances of MonkeyImage. Instead, use MonkeyDevice.takeSnapshot() to create a new instance from a screenshot. For example, use:

newimage=MonkeyDevice.takeSnapshot()

You can extend the monkey runner API with classes you write in the Java programming language and build into one or more .jar files. You can use this feature to extend the monkey runner API with your own classes or to extend the existing classes. You can also use this feature to initialize the monkey runner environment.To provide a plugin to monkey runner, invoke the monkey runner command with the -plugin <plugin_jar> argument .In your plugin code, you can import and extend the the main monkey runner classes Monkey Device, Monkey Image, and Monkey Runner in com. android. Monkey runner . Note that plugins do not give you access to the Android SDK. You can't import packages such as com. android. app. This is because monkeyrunner interacts with the device or emulator below the level of the framework APIs.

The plugin startup class

The .jar file for a plugin can specify a class that is instantiated before script processing starts. To specify this class, add the key MonkeyRunner Startup Runner to the .jar file's manifest. The value should be the name of the class to run at startup. The following snippet shows how you would do this within an ant build script:

```
<jar jarfile="myplugin" basedir="${build.dir}">
<manifest>
<attribute name="MonkeyRunnerStartupRunner" value="com.myapp.myplugin"/>
</manifest>
</jar>
```

To get access to monkeyrunner's runtime environment, the startup class can implementcom.google.common.base.Predicate<PythonInterpreter>. For example, this class sets up some variables in the default namespace:

```
package com.android.example;

import com.google.common.base.Predicate;
import org.python.util.PythonInterpreter;

public class Main implements Predicate<PythonInterpreter> {
   @Override
   public boolean apply(PythonInterpreter anInterpreter) {

       /*
       * Examples of creating and initializing variables in the monkeyrunner environment's
       * namespace. During execution, the monkeyrunner program can refer to the variables "newtest"
       * and "use_emulator"
       *
       */
       anInterpreter.set("newtest", "enabled");
       anInterpreter.set("use_emulator", 1);

       return true;
   }
}
```

These are the main working format of MonkeyRunner tools with which work has been going to give the user a better experience of the UI.

III. CONCLUSION

From my experience, monkey testing is really good for detecting flaws of the application in terms of:

1. Memory leaks: sometimes it is impossible to track scenarios generating excessive memory usage (say basic fast rotation, subsequent button clicks etc.).
2. Monkey also helps identifying test cases; unintended, strange uses of the applications that lead eventually to crashes.
3. Using monkey tests you can also somehow measure performance of the application, when used by "heavy" users.

I would say, that monkey testing does not stand in opposition to unit/instrumentation testing, but it is yet another way to test, that your application is working as intended.

Of course it also depends on the software is about to be tested, but in my opinion it is not always that easy to determine what happens if your button is clicked, then 9px above the button is touched and eventually a phone activity is run. And also I like to use MonkeyRunner because its really portable (Linux, Mac and Windows), easy to setup and can work easily across many different devices and emulators. Also, sometimes with instrumentation you get crashes that are unrelated to the app, but are rather because of the instrumentation implementation. With MonkeyRunner you will know what caused the crash. Future scopes on the field of Android UI development for these two are very bright as the platform it gives the users and so as the developers and huge and fast growing.

REFERENCES

[1]. Android developers site, tools monkeyrunner and monkey, monkeyimage ,monkeydevice.

[2]. Harini’s webspace, “Android Application UI Testing (with monkey and monkeyrunner)”

[3].Android as I learn, **“**Android testing with monkeyrunner – a monkeyrunner tutorial”.

[4]. Developer zone, “Automated Android* Application Testing” by Roman Khatko(INTEL).