

Southern university of science and technology

Middle term report

Optimization and application of monkey test

Supervisor

Zhang Yuqun

|  |  |
| --- | --- |
| *Team member:*  Wang Zelin  Guo Mengwei  Hu Sizhe | *Student Number:*  11612715  11610615  11610211 |

**Background**

As the fast pace of Android app development and evolution continues, effective quality assurance for industrial Android application becomes increasingly necessary and demanding. User Interface testing, aiming to uncover potential app defects by mimicking human interactions, is a popular approach to ensure the quality of mobile apps and has long been a important testing methods.

There are three most popular automated GUI testing forms in the world. Model-based techniques rely on abstract models of an AUT as the basis for test generation. Monkey is a popular random GUI testing tool that is part of the Android Software Development Framework (SDK). It generates GUI tests by interacting with random screen coordinates on Android devices. Dynamic event extraction techniques do not use a preexisting abstract model to generate test cases. Dynamic event extraction techniquesdo not generate infeasible test cases since the test cases are based on runtime interaction with the AUT.

This work adopts a dynamic event extraction approach. We try to use reinforcement learning techniques to optimize event selection in an attempt to improve code coverage relative to random test generation.

**Limitations of Monkey**

The randomness of monkey testing often makes the bugs found difficult or impossible to reproduce. Unexpected bugs found by monkey testing can also be challenging and time consuming to analyze. In some systems, monkey testing can go on for a long time before finding a bug. For smart monkeys, the ability highly depends on the state model provided, and developing a good state model can be expensive.

**widget obliviousness**

- Monkey will trigger events on a random coordinates of a screen while it can’t sense the location of widgets on a screen. Besides, it will generate many effect-free events which won’t increase the line coverage and activity coverage. At some cases it will keep staying in one page and do the meaningless click until it triggers the return button.

**state obliviousness**

- Monkey is oblivious to the GUI state before and after an event, which leads to its unawareness of state switching event. The state obliviousness will cause repeated actions, even infinite loop. It will result in redundant explorations and occupy much of the exploration time.

**Useless exploration**

During exploration, monkey explores newly generated states for a long time. However,these newly generated states do not significantly increase the line or activity coverage, since these states lead to many redundant exploration actions. For instance, in WeChat, monkey click one ‘call failed’ and add another ‘call failed’. However, this didn’t add any code coverage.

**Approach**

**1.Design flow**

The agent will be a procedure in PC. It use android app activity statement as input, output next operation like action type, click position and so on. We use ADB to connect phone with PC (agent). After each operation in phone, it transmits information to agent, the information includes the current activity statement, unit statement and so on. The agent uses this information to choose next operation and transmits this operation to the phone.

This flow will iterate until the time is over or get requested code coverage. Till now, we had built this organization. It can work as monkey, get information from the phone, do choice randomly and transmits this operation to the phone.

**2.Read current state**

We have done some researched about the general way of getting thestate of android application. Until now, we used the combination of ADB(android debug bridge) and UIAutomator to get the state of application. UIAutomator has offered a tool named UIAutomator viewer, which can produce a xml file contains all the details of current widgets. With the help of this tool, we produce the xml through ADB and pull it back to computer. In the computer side, we run a python script to analysis this script and store the information of current widgets. They are transformed to a type of class named widget.

**3. generate next event**

After we get the widgets’ list, the next step is to choose the next action. In addition to the basic structure of widgets, the xml file generated by UIAutomator also contains the functionality of widgets. For example, whether it can be clicked(clickable) or scrolled (scrollable). We will choose limited widgets from the whole list base on their functionality. After that, we will randomly pick one from the chosen widgets and click it. The Q-learning strategy of the choosing process will be put in consideration in our future work.

**4. operate the event**

When we get the chosen widget and its information. The next step is operating on it. We use another tool in android SDK to reach this goal. Monkey runner is designed to operation on android application by executing python script. That means we can directly add the code of operating after the code of generation, in the same python script. For now, we only implement the click action. When the operation is done, the program will wait for a period of time until we received the signal of operating next event.

**5.code coverage**

Code coverage is a measurement of how many lines/blocks/arcs of your code are executed while the automated tests are running. The program with high test code coverage can cover more source code while running, which suggests it has a lower chance of containing undetected software bugs compared to a program with low test coverage. In general, the code coverage can be classified as four types:

* **Instructions coverage**: The degree of assembly instructions that have been executed.
* **Lines coverage**: The degree of source code lines that have been executed.
* **Methods coverage**: The degree of self-defined methods have been called.
* **Classes coverage**: The degree of the classes that have been initialized.
* **Branch coverage**: The degree the branches in each control structure (such as if-else, switch-case) that has been visited.

We use code coverage to evaluate this work. The rule can be code, block, class and package coverage. To get this UI test coverage, we need to do instrumentation to our app. At first, we used Emma to do this. However, it need ant to generate app, this cause some bugs. Later we choose jacoco to instrument. It work excellent and can reach our request.

**Future work**

**Q-learning in test generation:**

图片包含 文字

描述已自动生成What we now suppose is that AUT represents the environment and testing tool is the agent, GUI state is the state, event is the action the agent needs to take. The goal of the agent is to learn a sequence of actions that maximize the reward.

Figure 1: The pseudo code of Q-learning algorithm

Lines 5-10 set the initial Q-value to the value Vinit for events that have never been executed. The getMaxValueEvent procedure on line 11 selects the event that has the maximum Qvalue from the events in the current GUI state and line 12 executes the selected event. The call to getAvailableEvents on line 20 gets the available events in the GUI state resulting from executing a selected event in the previous state. Lines 21-22 calculate the reward and discount factor for the executed event as defined in equations 1 and 3 respectively. The getMaxValue procedure on line 23 returns the maximum Q-value in the resulting state. This value represents an estimate of future rewards that may be accrued by executing the selected event. Each time an event is executed, it is added to the event sequence for the current test case and the Q-value estimate is updated on line 25. And test generation can stop until the code coverage result converges.

**Optimize tools.**

We use uiautomator to read current statement and use monkeyrunner to send operator. However, there are still some blemish in the work. The uiautomator takes some second to generate xml file. This time is to long for a UI test tool. And the monkeyrunner can not run python as windows environment, processes in running will influence each other and cause wrong result. Besides these, the monkey runner can not send system operation to the phone. We will try new ways to solve these problem in future work.

# Reference

[1] David Adamo, Md Khorrom Khan, Sreedevi Koppula, and Renée Bryce. 2018. Reinforcement learning for Android GUI testing. In Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation (A-TEST 2018). ACM, New York, NY, USA, 2-8. DOI: https://doi.org/10.1145/3278186.3278187

[2] Aravind Machiry, Rohan Tahiliani, and Mayur Naik. 2013. Dynodroid: an input generation system for Android apps. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013). ACM, New York, NY, USA, 224-234. DOI: https://doi.org/10.1145/2491411.2491450

[3] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon. 2012. Using GUI ripping for automated testing of Android applications. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012). ACM, New York, NY, USA, 258-261. DOI=http://dx.doi.org/10.1145/2351676.2351717

[4] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: multi-objective automated testing for Android applications. In Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016). ACM, New York, NY, USA, 94-105. DOI: https://doi.org/10.1145/2931037.2931054