GAutomator Unity自动化测试教程

GAutomator 通过Python实现Unity手游的UI自动化测试，强烈建议使用pycharm编辑python。可在bin目录下包含所有需要的组件。

注： GAutomator UE4版本，需要把config.py中的EngineType修改为Engine.UE4，默认为Unity

1 准备工作

1.1 介绍

通过Python实现Unreal 4手游的UI自动化测试。GAutomator测试运行在手机端，通过adb操控手机上的UE4手游，支持所有版本的Android手机。这个工具的主要功能包括：测试与Android手机之间的兼容性--测试手游在不同Android手机上的工作情况。功能性测试，PVP游戏可以自动化测试代替人力节省操作，PVE游戏可以自动大关完成冒烟测试。性能测试，云端测试能够手机CPU、内存、流量和FPS数据，能够标记不同的场景。

1.2 环境

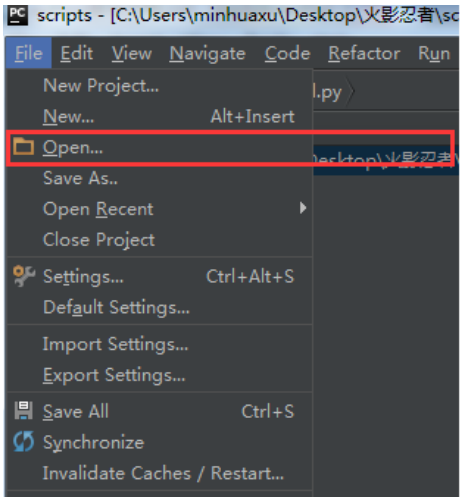
1 python: python 2.7

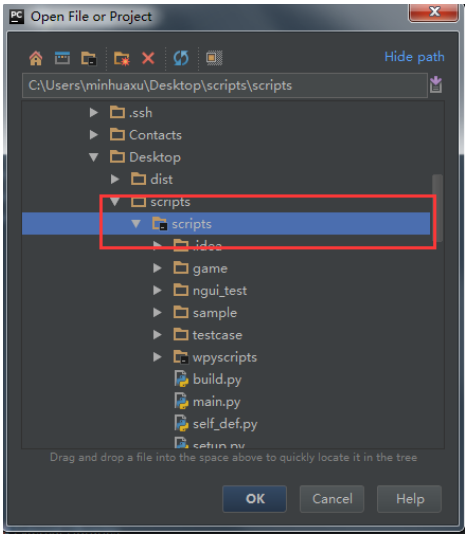
2 adb

请确保，你的path环境变量里面设置了adb
在cmd命令行里面输入adb devices，能够看到你的手机序列号

1.3 使用脚本

如果使用pycharm的话，直接打开scripts功能即可进行编辑使用

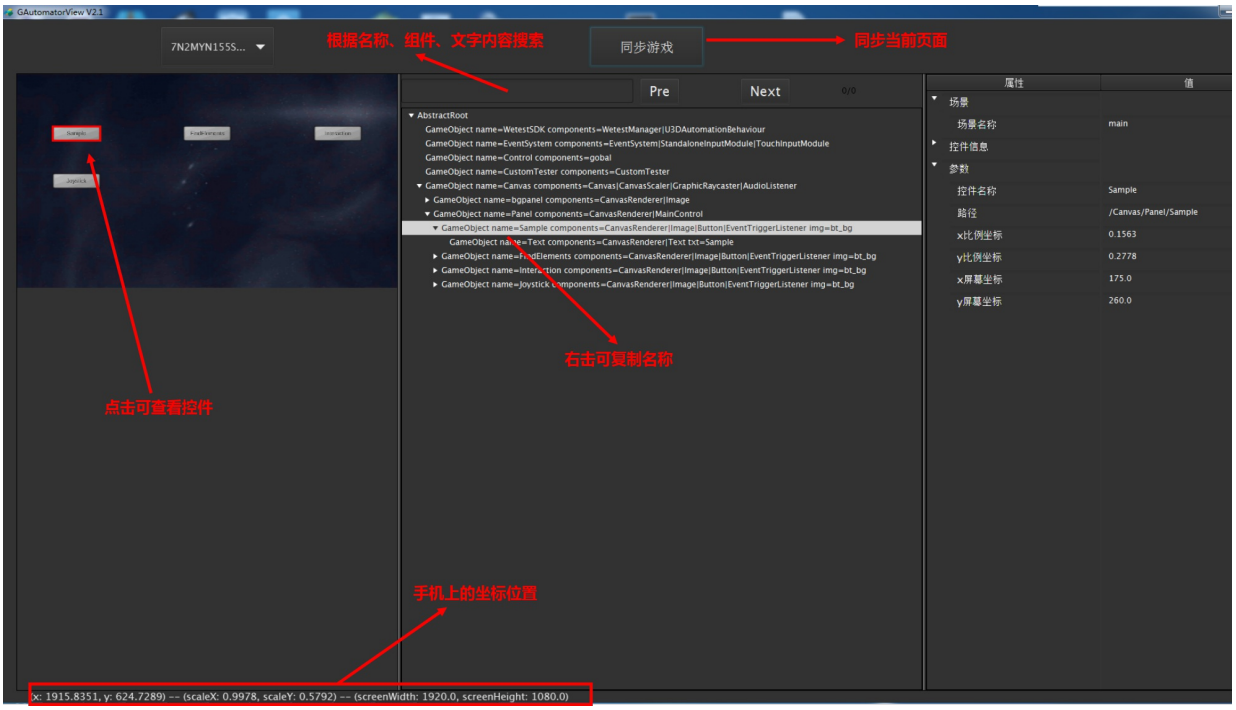




可以在testcase目录下面直接创建你需要的.py脚本，然后编写需要的逻辑

1.4 GAutomatorView

GAutomatorView工具可在http://wetest.qq.com/cloud/index.php/phone/blrooike下载 。GAutomator主要根据，Unity游戏中的GameObject的路径名称来编写逻辑。类似于UIAutomator需要有一个，控件查看器；GAutomator也提供了一款类似的，Unity游戏中控件查看器。
注：请勿将该软件放置在中文目录下



集成wetest sdk的游戏拉起后，点击同步按钮，就能获取到游戏界面和控件树

2 Getting Started

示例代码：sample/sample.py,示例apk游戏:sampel/wetest_demo_ue.apk

2.1 Simple Usage

已经安装好python及依赖库后，可以使用pycharm（请下社区版，社区版免费）直接打开工程，你可以下面的代码开始我们的测试

```
<a name="lib"></a>

#import lib path,only use in this demo
#import sys,os
#sys.path.append(os.path.abspath(os.path.join(os.getcwd(), "..\\")))

import wpyscripts.manager as manager

def test():
    version = engine.get_sdk_version()
    logger.debug("Version Information : {0}".format(version))

    scene = engine.get_scene()
    logger.debug("Scene :   {0}".format(scene))

    sample_button = engine.find_element("Sample")
    logger.debug("Button : {0}".format(sample_button))
    logger.debug("Button Bound: {0}".format(engine.get_element_bound(sample_button)))
    screen_shot_click(sample_button)

test()
```

上面的代码可以保存为sample.py,然后运行

```
python samle.py
```

请确保，wetestdemo游戏已经拉起，GAutomator库能够查找到

2.2 实例详解

wpyscripts.manager模块提供了自动化测试所需的所有功能，提供与引擎、手机、报告相关的内容，也提供了日志实现

```
import wpyscripts.manager as manager
```

下一步，创建Engine和日志实例

```
engine=manager.get_engine()
logger=manager.get_logger()
```

*`engine.get_sdk_version()`*能够获取Unity版本信息、Wetest sdk版本信息，能够获取该信息时，证明脚本已经成功连上游戏。如果获取失败，则会抛出`WeTestNativeEngineDllError`异常,抛出该异常可能是手机USB线没有连好或者手机开发者选项未打开。
*logger.debug("")*输出对应日志，请使用manager.get_logger()获取的实例，避免脚本在云端wetest.qq.com使用时出错。

```
version=engine.get_sdk_version()
logger.debug("Version Information : {0}".format(version))
```

*`engine.get_scene()`*获取当前游戏界面对应地图(Level关卡)名称

*`engine.find_element("Sample")`*查找当前界面中路径为Sample的节点，如果存在则返回Element，不存在则返回None。当前节点返回的仅为UMG的节点信息，UMG的节点名称唯一，因此find_element仅传节点名称即可。
查找到的节点samle_button（*Element*），有两个属性object_name,instance。object_name代表的是节点的名称，在UMG中该节点名称一定唯一。
*`engine.click(sample_button)`*尝试点击samle_button这个UObject的中心点。

```
sample_button=engine.find_element("Sample")
logger.debug("Button : {0}".format(sample_button))
engine.click(sample_button)
```

GAutomator包含3大接口

```
engine=manager.get_engine()
reporter=manager.get_reporter()
device=manager.get_devcie()
```

- engine:Unity相关内容，主要包括控件获取，游戏操作。**Unity与UE4 engine的接口不一致，具体详见文档**。GAutomator根据配置，选择对应的实现，Unity引擎实现为UnityEngine类，**UE4引擎为UnRealEngine类**
- reporter:云端报告相关，截图、标记操作过程、性能数据打标签
- device:手机设备相关，如屏幕长宽高、转向，也包括QQ登录等。

2.3 wetest云端兼容测试

GAutomator编写好的测试脚本，只需要非常简单的修改，就能wetest云端上做兼容测试。云端几千台手机，按照脚本执行游戏。wetest能够发现兼容问题，同时高度还原执行现场，包括手机日志、崩溃信息、截图、执行过程等。

云端执行脚本时，会执行testcase.runner下的run函数，只需要把自己的业务逻辑加入到这个函数中即可

```
import traceback

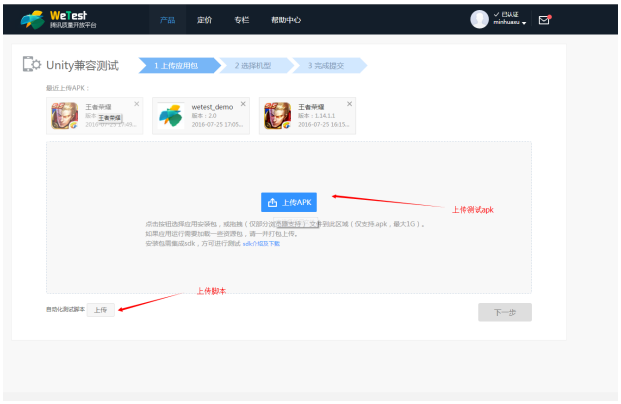
try:
    from sample.sample import *
except Exception,e:
    traceback.print_exc()

def run():
    """
        业务逻辑的起点
    """
    try:
        test()
    except Exception,e:
        traceback.print_exc()
```

然后，运行scripts目录下的，build.py

```
python build.py
```

会在scripts目录下产生一个,wpyscripts_upload.zip。只有企业用户才可以使用云端测试，请登录wetest.qq.com，联系工作人员了解详情。



2.4 本地运行

注：调试时手动启动游戏，运行到指定界面，运行对应的脚本即可，如调试大厅界面的代码，游戏跑到大厅界面，再运行自动化测试逻辑。不需要从main.py启动

GAutomator支持一台PC在多台android手机上同时测试。在config.py文件中，可以配置，完成单台手机测试的情况。

测试的游戏包名

```
class TestInfo(object):
    PACKAGE="com.tencent.wetest.demo" # 测试包名
```

测试账号

```
class Account(object):
    QQNAME="" #QQ账号
    QQPWD="" #QQ密码
    WECHATNAME="" #微信账号
    WECHATPWD="" #微信密码
```

引擎选择（默认为Unity)
UE4用户一定要配置为UE4

```
<a name="Engine"></a>

### Engine Type
Unity="unity"
UE4="ue4"

class Engine(object):
    Type=UE4 #Type="unity" # unity or ue4
```

一般一个工程通过main方式启动，只能测试一个游戏，所以直接在main.py里面写死，也避免参数传入的麻烦。
1、测试一台手机，如果PC上USB只连接一台手机，直接启动main.py即可(前提配置好测试包名)

```
python main.py
```

2、测试多台手机，如果PC上USB连接超过一台手机，需要通过命令行的方式启动

```
adb devices #查看当前手机序列号

saaaweadf        device
asdfadfadf   device
```

获取到当前PC连接的手机序列号之后，通过命令行的方式控制脚本在指定的手机上进行测试。

```
python main.py --qqname=2952020110 --qqpwd=wetestpwd --engineport=50031 --uiport=19000 --serial=saaaweadf
python main.py --qqname=2952020111 --qqpwd=wetestpwd --engineport=50032 --uiport=19001 --serial=asdfadfadf
```

上面的命令分别代表，在序列号"saaaweadf"手机上测试，测试时使用的QQ号为2952020110,密码为wetestpwd，与引擎建立映射的网络端口号为50031,与UIAutomator服务建立映射的网络端口为19000。第二条命令类似。

命令行参数含义如下：

```
--qqname:qq账号，每部手机应该都不一样
--qqpwd:qq密码
--wechataccount:微信账号
--wechatpwd:微信密码
--othername:其他任何账号
--otherpwd:其他任何账号的密码
--engineport:与手机端的sdk服务建立网络映射，填入的为本地的网络端口号（如,50031），不同手机之间要确保不同
--uiport:与手机端的UIAutomator服务建立网络映射，填入的为本地的网络端口号（如,19008），不同手机之间要确保不同
--serial:adb devcies能够查看手机的序列号，不同的序列号代表不同的手机
```

3 Locating Elements

UE4目前版本仅支持UMG控件的查找与操作
engine模块提供了一种UI控件的查找方式。示例：sample/UE4/find_elements.py

- *find_element*

3.1 find_element

*find_element*通过Unity的GameObject.Find()方法查找游戏中的的gameobject。*find_element*通过GameObject的名称查找对象，名字中可以包含'/'代表GameObject树中的一层。这方法只返回当前激活(active)的gameobject。
当界面上有两个一模一样路径的gameobject时，只返回其中的一个。代码示例：

```
#import sys,os,time
#sys.path.append(os.path.abspath(os.path.join(os.getcwd(), "..","..")))

def test_find_element():
    """
```

```
        使用engine.find_element查找游戏中的节点
    :return:
    """
    button = engine.find_element("Button_0")
    bound = engine.get_element_bound(button)
    logger.debug("Button : {0},Bound : {1}".format(button, bound))
    engine.click(button)

    button = engine.find_element("Button_1")
    bound = engine.get_element_bound(button)
    logger.debug("Button : {0},Bound : {1}".format(button, bound))
    engine.click(button)

    unexited_gameobj = engine.find_element("Test")
    if unexited_gameobj is None:
        logger.debug("Test GameObject not find")

test_find_element()
```

上面的代码可以保存为find_elments.py,从wetest_demo_ue点击FindElements，然后运行

```
python find_elments.py
```

UE4的UMG中UI控件的名称一定是唯一的，可以作为标识存在。如果查找的名称存在则返回Element，如果不存在则返回None

3.2 节点位置查找

3.2.1 节点在屏幕上的位置

*engine.get_element_bound(element)*能够获取节点在屏幕中的位置。GAutomator所有的操作都是通过触屏进行的，因此获取节点在屏幕上的位置是进行交互操作的基石。

```
def test_click():
    # 点击节点
    element = engine.find_element("ClickBtn")
    bound = engine.get_element_bound(element)
    logger.debug("Button : {0},Bound : {1}".format(element, bound))

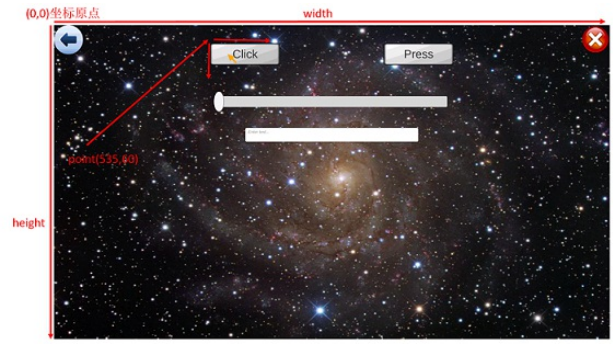
    engine.click(bound)
    time.sleep(1)
    engine.click(element)

    # 点击坐标
    time.sleep(2)
    engine.click_position(600.0, 100.0)

test_click()
```

```
[{u'existed': True, u'width': 200.688, u'visible': True, u'height': 99.9, u'instance': 0, u'path': u'ClickBtn', u'y': 140.4, u'x':
```

`engine.get_element_bound(Element)`获取的是ElementBound，获取Element的左上角在屏幕上的坐标，和Element的长宽高。遵循的是手机的坐标系，以左上角为坐标原点，上下边框为width，左右为height。



4 交互

找到节点后的第一件后，就需要对寻找到的节点进行操作。示例：sample/interaction.py

```
engine.click(button)
```

Engine执行操作后，会等到操作执行完成后才会返回。engine.click(Element)返回为True的话，只保证执行了button中心点的点击事件，不能确保button对应的事件被有效执行（有弹出框，遮住的情况就可能使点击无效）。

4.1 点击操作

*engine.click()*允许传入Element和ElementBound。如果传入的是Element，会先去查找ElementBound,然后再计算出节点的中心位置进行点击。所以，在有ElementBound的情况下，应该首先传入ElementBound。

```
def test_click():
    #点击节点
    element = engine.find_element("ClickBtn")
```

```
    bound=engine.get_element_bound(element)
    logger.debug("Button : {0},Bound : {1}".format(element,bound))

    engine.click(bound)
    time.sleep(1)
    engine.click(element)

    time.sleep(2)
    engine.click_position(600.0,100.0)

test_click()
```

上面的代码可以保存为interaction.py,从wetestdemo点击Interaction，然后运行

```
python interaction.py
```

程序会连续点击3下，Click按钮。

- `engine.click(bound)`会点击，Click的中心节点(point.x+withd/2,point.y+height/2)
- `engine.click(element)`首先回去查找element节点的ElementBound，然后计算出中心点，在进行点击
- `engine.click_position(600.0,100.0)`直接点击屏幕坐标为(600.0,100.0)的坐标。**手机屏幕尺寸发生变化，点击将无效，不能点击到期望位置**

4.2 long press长按

*engine.press()和engine.press_position*与click相似，多一个时间参数，表示长按的时间（单位ms,毫秒）

```
def test_press():
    element=engine.find_element("PressBtn")
    engine.press(element,5000)
    time.sleep(2)
    engine.press_position(1200,100,3000)

test_press()
```

上面的代码可以保存为interaction.py,从wetestdemo点击Interaction，然后运行

```
python interaction.py
```

- `engine.press(element,5000)`,`PressBtn`节点连续按住5s
- `ngine.press_position(1200,100,3000)`,(1200,100)点，连续按住3s

4.3 swipe滑动

engine.swipe(start_element, end_element, steps, duration=1000)和engine.swipe_position(start_x,start_y,end_x,end_y, duration=1000)，可以从一个节点滑动到另外一个节点。duration以毫秒为单位，为滑动的时长。滑动时长不能不能精确控制，只是尽可能接近。。*swipe与swipe_position*动作执行完之后返回，由SDK负责执行动作，不能并行的执行动作。如下面的示例中，第一个动作执行完后，才会执行第二个动作。

```
def test_swipe():
    start_e = engine.find_element("ClickBtn")
    end_e = engine.find_element("PressBtn")
    engine.swipe(start_e, end_e,2000)

    time.sleep(5)

    silder = engine.find_element("Slider_0")
    if silder:
        bound = engine.get_element_bound(silder)
        engine.swipe_position(bound.x+3, bound.y + bound.height / 2.0, bound.x + bound.width, bound.y + bound.height / 2,3000)
test_swipe()
```

上面的代码可以保存为interaction.py,从wetest_demo_ue点击Interaction，然后运行

```
python interaction.py
```

从Press的中心点按钮按下，一直Move到Click的中心点，中间经过50步，最后执行Up动作，持续时长大致为2秒

```
start_e = engine.find_element("ClickBtn")
end_e = engine.find_element("PressBtn")
engine.swipe(start_e, end_e,2000)
```

无论swipe的步长设置为多少，都会立刻返回。立刻执行swipe_position函数，swipe_position也需要动作执行完之后才会返回，但是游戏中还不会马上执行这个动作。需要swipe执行完成后，才会执行swipe_position的动作。

```
 silder = engine.find_element("Slider_0")
    if silder:
        bound = engine.get_element_bound(silder)
        engine.swipe_position(bound.x+3, bound.y + bound.height / 2.0, bound.x + bound.width, bound.y + bound.height / 2,3000)
```

4.4 获取文字内容

可以获取到游戏中的文字内容。NGUI能够获取到UILable、UIInput、GUIText组件上的文字内容，如果GameObject上不包含以上组件，将抛出异常。UGUI能够获取Text、GUIText组件上的文字信息。示例在interaction.py中，wetest_demo.apk需要在interaction界面。

```
def test_get_element_txt():
    e=engine.find_element("Click/Text")
    text=engine.get_element_text(e)
    logger.debug("Text = {0}".format(text))
```

上面的代码在sample/interaction.py中，运行该函数可以获取文字内容"Click"

5 Mobile设备

*engine.get_device()*类device提供与手机相关信息的API，也提供简单的操作。示例：sample/devices_tester.py

5.1 屏幕尺寸与转向

```
def test_get_display_size():
    display_size=device.get_display_size()
    logger.debug(display_size)

    rotation=device.get_rotation()
    logger.debug("Rotation : {0}".format(rotation))

test_get_display_size()
```

获取屏幕尺寸，DisplaySize类包括width、height单位为px。

5.2 顶层Package与Activity

```
def test_get_top_package_activity():
    top_activity=device.get_top_package_activity()
    logger.debug(top_activity)

test_get_top_package_activity()
```

上面的代码可以保存为devices_tester.py,再任意界面启动

```
python devices_tester.py
```

*device.get_top_package_activity()*获取手机当前界面的TopActivity对象，包含顶层app的包名和Activity名称。

```
package name = com.tencent.wetest.demo,activity = com.unity3d.player.UnityPlayerActivity
```

5.3 回退键

GAutomator本身不提供对标准Android控件的支持，所以当界面上出现标准控件时将无法进行操作。因此，提供了回退（Back）操作，返回到游戏Activity。

```
def test_back():
    device.back()
test_back()
```

上面的代码可以保存为devices_tester.py,再任意界面启动

```
python devices_tester.py
```

*device.back()*与按Android的回退键效果一致。

6 云端报告

*engine.get_reporter()*获取的Reporter类封装了与云端报告相关的内容，本地实现为空，只有在云端运行的时候才会有效果。游戏自动化测试过程中需要保持测试现场，所以在云端运行过程中需要标记测试过程和截图。Reporter主要负责与功能

```
import sys, os, time

#sys.path.append(os.path.abspath(os.path.join(os.getcwd(), "..\\")))

import wpyscripts.manager as manager

engine = manager.get_engine()
logger = manager.get_logger()
reporter = manager.get_reporter()


def screen_shot_click(element):
    logger.debug("screen_shot_click")
    if element is None:
        return
    bound = engine.get_element_bound(element)
    logger.debug(bound)
    pos_x = bound.x + bound.width / 2
    pos_y = bound.y + bound.height / 2
    reporter.capture_and_mark(pos_x, pos_y, locator_name=element.object_name)
    engine.click_position(pos_x, pos_y)


def enter_find_elmeents():
```

```
    find_elements_button = engine.find_element("FindElements")
    logger.debug(find_elements_button)
    screen_shot_click(find_elements_button)
    time.sleep(1)


def back_main():
    find_elements_button = engine.find_element("Back")
    logger.debug(find_elements_button)
    screen_shot_click(find_elements_button)
    time.sleep(1)


def test_capture_and_mark():
    for index in range(1,6):
        name="level{0}Btn".format(index)
        e=engine.find_element(name)
        screen_shot_click(e)

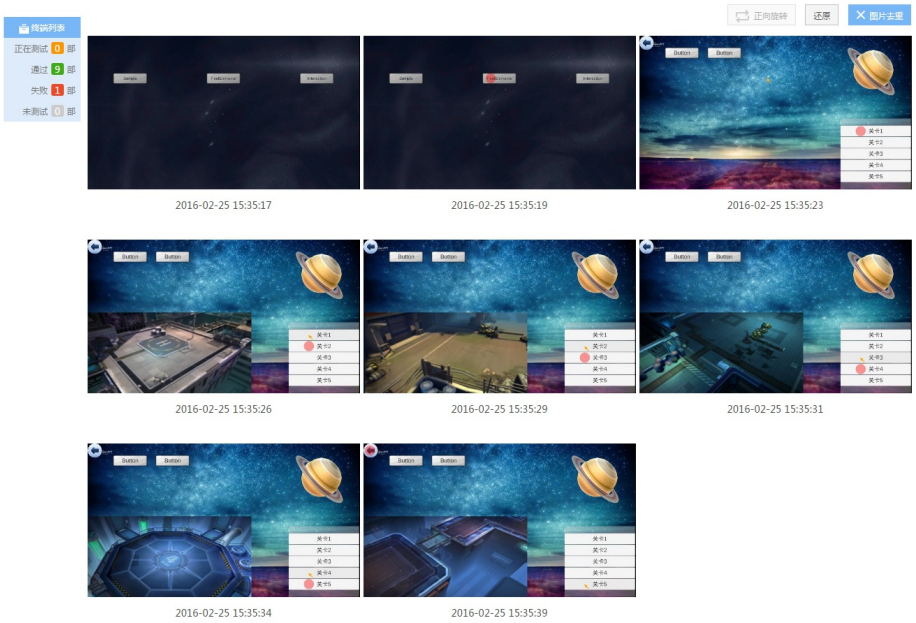

def test_reporter():
    enter_find_elmeents()
    time.sleep(2)
    reporter.add_start_scene_tag("Find_Scene")
    test_capture_and_mark()
    reporter.add_end_scene_tag("Find_Scene")
    time.sleep(2)
    back_main()
    reporter.screenshot()
```

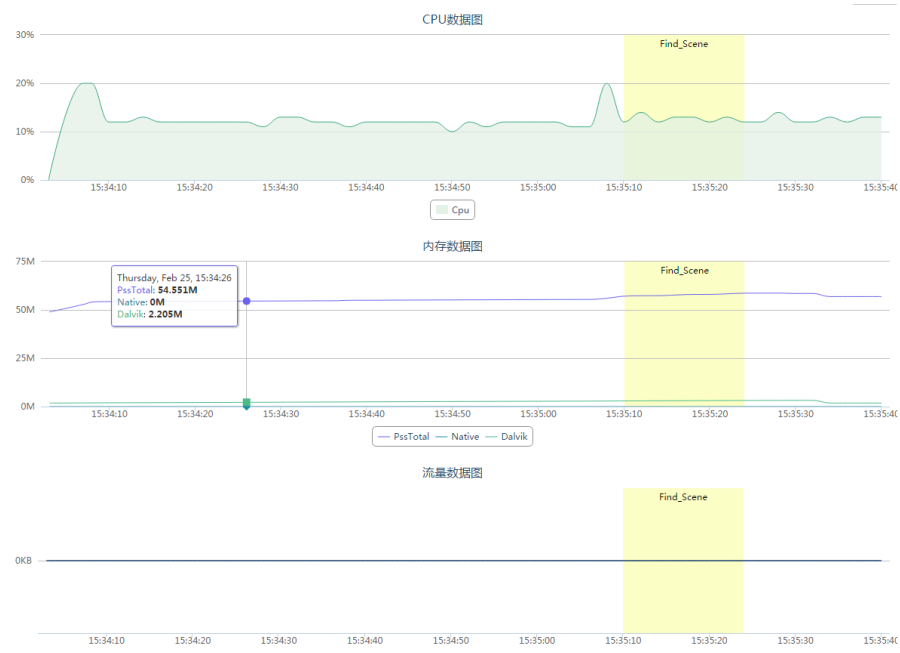
runner.py里面，调用test_reporter()。上传到平台后的结果的运行结果（同事在几百台手机上运行）

```
import traceback

try:
    from sample.reporter_tester import *
except Exception,e:
    traceback.print_exc()

def run():
    """
        业务逻辑的起点
    """
    try:
        test_reporter()
    except Exception,e:
        traceback.print_exc()
        stack=traceback.format_exc()
        logger.debug(stack)
```

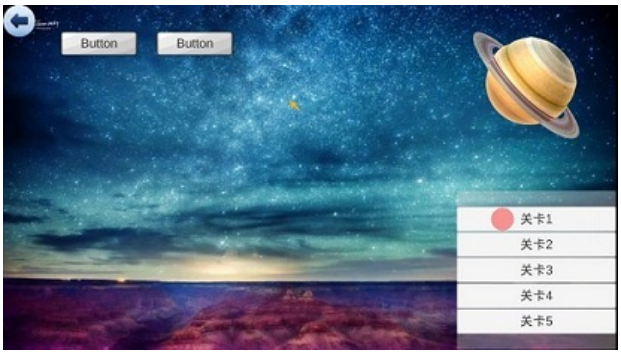




6.1 截图与操作过程标记

```
def screen_shot_click(element):
    logger.debug("screen_shot_click")
    if element is None:
        return
    bound = engine.get_element_bound(element)
    logger.debug(bound)
    pos_x = bound.x + bound.width / 2
    pos_y = bound.y + bound.height / 2
    reporter.capture_and_mark(pos_x, pos_y, locator_name = element.object_name)
    engine.click_position(pos_x, pos_y)
```

*reporter.capture_and_mark(pos_x, pos_y, locator_name = element.object_name)*将会截取当前手机屏幕，并在pos_x,pos_y位置标记一个红点。



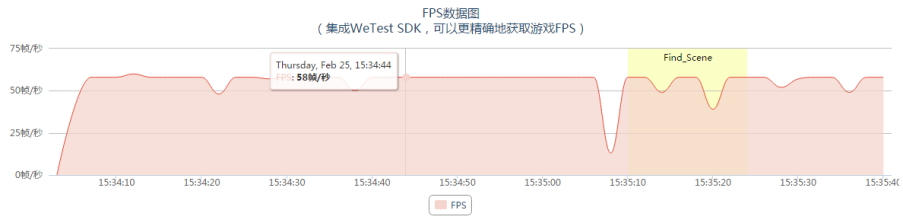
6.2 截图

*reporter.screenshot()*在云端会截图在报告里面体现，在本地运行时会截图并放在运行目录下的screenshot目录下面。

6.3 打标签

```
reporter.add_start_scene_tag("Find_Scene")
reporter.add_end_scene_tag("Find_Scene")
```

*reporter.add_start_scene_tag("")和reporter.add_end_scene_tag("")*一定是成对出现的，先start然后end，里面的标签内容需要一样。



注：配合engine.get_scene()效果更佳

6.4 报告错误

GAutomator并不是使用常见的，unittest作为测试的底层框架，因此并无断言，无法做功能测试。report_error接口，可用于错误报告，并且在运行目录下生成一份_wetest_testcase_result.txt用户记录报告的内容。该文件的报告格式与unittest的测试报告格式一致，因此在云端测试时可现实具体的信息。

```
report.report_error("testcase","content")
report.report_error(u"report_test",u"Report test error 中文")
```

*reporter.report_error(name,message)*接口调用的过程中，会在日志中输出。脚本运行结束时，runner.run中，会调用_report_total(),将所有的判断结果输出到_wetest_testcase_result.txt中。除了输出message和test_case_name之外GAutomator还会加上调用堆栈。name的名称尽可能不重复。name与message传入的编码方式需要一致，如果存在中文的情况下必须要使用UTF-8编码格式。

7 实战用例

举例最常见的，较难处理的引用场景scripts/testcase/tools.py封装了，场景的使用场景

7.1 记录操作流程

自动化测试记录操作流程，有利于出现bug时定位和复现。所以原则上，应该记录每一步操作。tools.py里面封装了一个接口，能够在截图上标记点击的位置，然后执行点击操作，点击完成等待相应的时间。

- screen_shot_click(element,sleeptime)接口,传入需要点击的节点和点击后等待时间。

```
def screen_shot_click(element, sleeptime=2):
```

```
    """
        点击，并标记红点。
    :param element: 需要点击的element
    :param sleeptime:
    :return:
    """
    if element is None:
        return
    try:
        bound = engine.get_element_bound(element)
    except WeTestRuntimeError, e:
        bound = None
    if not bound:
        return
    logger.debug(bound)
    pos_x = bound.x + bound.width / 2
    pos_y = bound.y + bound.height / 2
    logger.debug("screen_shot_click_pos x = {0},y = {1},name = {2}".format(pos_x, pos_y, element.object_name))
    report.capture_and_mark(pos_x, pos_y, locator_name=element.object_name)
    engine.click_position(pos_x, pos_y)

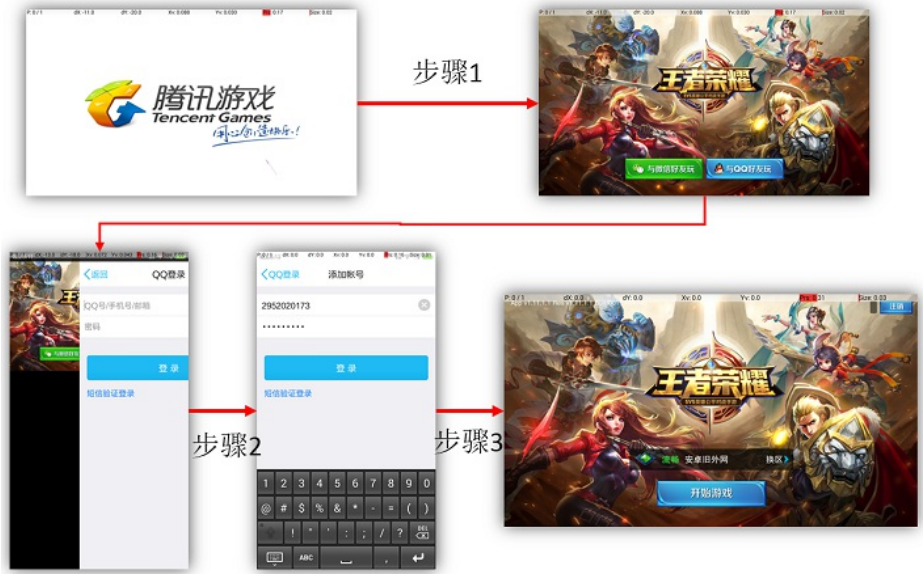
    time.sleep(sleeptime)
```

传入的element为空或者element的位置找不到，则自动跳过。

```
qq_button = engine.find_element("/BootObj/CUIManager/Form_Login/LoginContainer/pnlMobileLogin/btnGroup/btnQQ")
screen_shot_click(qq_button, 6)
```

7.2 QQ或微信登录

QQ或者微信登录，设计到Activity的切换和Android标准控件的操作,操作过程复杂，但是相对较为固定。在云端运行时，每次拉起游戏之前，都会清理数据，所以每次都需要重新登录。每次登录的过程如下所示：



对应的处理代码如下所示： ```python def login(): # 步骤1，等待到达登录界面 wait_for_scene("SceneName")

```
# 选择QQ登陆
qq_button = find_elment_wait("/BootObj/Panel/btnQQ")
screen_shot_click(qq_button, 6)

#步骤2 ，等待进入QQ登录界面，packagename为com.tencent.mobileqq，如果是微信登录界面package为com.tencent.mm
wait_for_package("com.tencent.mobileqq")
device.login_qq_wechat_wait(120)
time.sleep(10)
```

```
#步骤3，等待QQ登录界面退出，切换到游戏界面
select_btn = find_elment_wait("/BootObj/Panle/selectBtn")
```

```
1. 步骤1：等待进入到登录选择scene，如何获取scene名称，请看[1.4 GAutomatorView](#1.4)。wait_for_scene("SceneName")，会一直查询，直到进入名称为
2. 步骤2：从游戏的Activity切换到QQ或者微信的登录界面需要一定的时间。`wait_for_package("com.tencent.mobileqq")`检查顶层包名，直到QQ的顶层包名(微
**注：账号由云端自动分配。本地调试时请修改wpyscripts/wetest/device.py下面`native_deivce.__init__(self)`中的账号密码**
3. 步骤3：等待进入游戏界面，直到出现某个element为止。


<a name="7.3"></a>

## 7.3 异常处理
对于GAutomator的异常处理是一件非常头痛的事情，在设计框架的过程中也是左右为难，本质原因在于手机的端的不稳定性。不稳定主要包括以下几方面：
1. adb不稳定:windows的adb及其不稳定长期连接过程中不可避免的会出现断开连接的情况。出现断开的情况在腾讯可能有应用宝tadb.exe端口抢占、IOA、QQ浏览器及其
1. 游戏不稳定:SDK部分与UI相关的内容运行在UI主线程，当游戏暂停时可能会出现timeout的情况。如，QQ登录按钮跳转到登录界面，分享按钮，游戏会退出前台主线程等
1. UIAutomator不稳定：UIAutomator并不是一个非常稳定的服务，可能会出现操作无效的情况。


框架本身，只要是框架处理不了的异常，都会抛给调用者。

1. 测试编写的过程中，如果出现操作的内容，可能会让游戏退出前台engine相关接口尽量catch。
1. 对于操作可有可无的，也尽量catch。如点击操作不影响测试流程，如攻击按钮，可以选择catch

<a name="8"></a>

# 8 实际使用接口
GAutomator主要的大三类接口engine,reporter，device属于颗粒度非常细的接口，尽可能的原子化，但是直接使用这部分内容进行开发的话，并不是一件容易的事情。
<a name="8.1"></a>

## 8.1 screen_shot_click 点击控件截图并记录轨迹
在使用过程中该接口基本，可以替代GameEngine.click。操作流程为截图->点击的位置标记红点->点击->sleep指定的时间，这个操作过程是比较理想的，也是一个最基本
wetest平台截图的速度非常快，对性能影响也极低，可以对每一个操作步骤均进行截图。
*`screen_shot_click(element, sleeptime=2, exception=False)`*
element:可以为Element实例，也可以为需要点击的name
sleeptime:点击完成后sleep的时间
exception:异常发生时，如果exception为True则抛出异常，如果exception为False则不抛出异常返回False

example:
```python
from testcase.tools import *

button=engine.find_element("LoginQQ")
screen_shot_click(button,sleeptime=5,exception=True)

screen_shot_click("Attack",sleeptime=0)
```

截图并标记轨迹如下所示，该部分功能仅限wetest平台测试有效：

8.2 screen_shot_click_pos 点击位置截图并记录轨迹

screen shot click pos与screen shot click的区别是，一个点击的是UI控件，一个纯粹是位置信息。操作流程两个是一致的，操作流程为截图->点击
```

的位置标记红点->点击->sleep指定的时间。

`screen_shot_click_pos(pos_x,pos_y, sleeptime=2, exception=False)`

pos_x:x坐标位置

pos_y:y坐标位置

sleeptime:点击完成后sleep的时间

exception:异常发生时，如果exception为True则抛出异常，如果exception为False则不抛出异常返回False