自動 GUI テストの使い方マニュアル

oniprog

最終更新 2013/07/13

□はじめに

自動 GUI テスト，とは，Graphical User Interface（GUI）を自動操作して，要所要所でスナップ画像を撮り，その画像を過去のものと比較することで，動作がおかしくなっていないかなどのチェックを行うテストのことです．

本プログラムは，Visual Studio2010 を使ってビルドしています．

自動 GUI テストには，外部操作型と内部操作型があります．外部操作型は，対象アプリに何も手を加えず，マウスメッセージやキーボードメッセージを模擬することで，対象アプリの動作を制御してテストを行う手法です．この手法では，動作の再現性が悪く，またテストの再利用もなかなか難しくなります．しかし，初期投資コストは少ないといえます．逆に，内部操作型では，対象アプリに直接操作を制御するためのプログラムを埋め込みます．そのため，動作の再現性がよく，テストの再利用性にも優れていますが，プログラムを埋め込むための作業などがあり，初期投資コストは大きくなります．

本プログラムは，どちらにでも対応可能ではありますが，メインターゲットは内部操作型にしています．

3 つのプログラムから構成されています．

1 つは，自動テストを駆動するプログラムで AutoGUITestCpp という名前です．このプログラムには，サーバ化してポートをたたかれるのを待ってから起動するのを無限に繰り返すモードと，すぐに起動して処理し終了する 2 つのモードがあります．これは設定ファイルでポート番号として 0 以外の整数を指定するとサーバーモードとなります．この自動テストを駆動するプログラムのサーバーモードは Jenkins から使われることを想定しています．wget -t 1 localhost:XXXX という形で，ポートをたたくことで自動 GUI テストを駆動させることができます．なぜ Jenkins 内から直接呼ばないのか，というと，Jenkins 内から呼ぶと，GUI の描画ができないため，自動 GUI テストができないから，です．そのため，間接的にポートを介して呼ぶ仕組みを作りました．

2 つめのプログラムは，テスト結果を目視して確認するためのプログラムで，ErrorImageViewer と言います．自動 GUI テストを作った当初は，HTML ファイルを生成してその結果を目視して確認していました．このとき，許容値よりも小さな違いしか無いファイルは表示しないようにしていました．しかし，OpenGL で描画した結果の出力は毎回微妙に異なるため，なかなか思うようにテストが収束せず，毎回大量の画像を目視する必要がありました．この問題を解決するために，疑似学習機能を搭載したこのプログラムを実装しました．疑似学習機能，というのは，正常画像，と判定された画像を，フォルダに蓄積していき，次回以降の判定に再利用する，というものです．これにより，だんだんと，誤判定が減っていき，目視の負担もなくなることが期待されます．

3 つめのプログラムは，アプリ埋め込み型の例，として作成したサンプルプログラムで，SampleApp と名付けました．このプログラムの描画は，GDI であり，OpenGL の描画のように，毎回描画が異なる，ということはほぼ起きません．添付されているテストスクリプトにより，画面を操作しつつ，スナップ画像を撮るサンプルテストを動作させることができるようになっています．

□ビルド作業

■必要なソフトのインストール作業

boost_1_53_0 と lpng162 のソースをダウンロードして，用意してある同名の空フォルダの中に配置してください．また，Python2.7 もセットアップをダウンロードしてインストールしておいてください．

boost_1_53_0 のビルドについて

以下のコマンド群を DOS 窓で実行します．なお，OS や VC や Python のバージョンによって微妙に修正が必要なのでご注意ください．

また，#が行頭にある行はコメントですので打ち込まないでください．

（DOS 窓を出すには，ディスクトップ上で右クリックして，新規作成→ショートカットの作成を実行し，cmd.exe と打ち込んでください）

```
# Visual Studio のビルド環境を整えます．(x86)がつくのは OS が 64 ビットのときです．
# 32 ビットのときはなくなります．10.0 の部分は，Visual Studio 2010 を使っているため
# です．
"c:¥Program Files (x86)¥Microsoft Visual Studio 10.0¥vc¥vcvarsall.bat"

# Boost のビルドのためのプログラムをビルドします．
bootstrap.bat

# project-config.jamを開いて下記を追加
using python : 2.7 : C:¥¥Python27¥¥python ;

# コマンドラインで以下を実行して Boost をビルドします．
b2 --build-type=complete toolset=msvc-10.0
```

以上で完了です．

lpng162 のビルドについて

すでにビルド済みのライブラリファイルを Git のプロジェクトに含めてあるので，ビルドする必要はありません．ただし，ヘッダーファイルを使っているので，ソースを配置することは必要です．

□自動テストを駆動するプログラム AutoGUITestCpp の使用方法

■Config ファイルについて

Debug フォルダにサンプルの config.txt ファイルが置いてあります．このファイルを編集することで，動作を変えることができます．

初期状態では，下記のようになっています．

```
{
	"ApplicationPath" : "..¥¥..¥¥..¥¥Debug¥¥SampleApp.exe",
	"TestFolder" : "..¥¥TestFolder",
	"OutputHTML" : "result.html",
	"ErrorValue" : 5,
	"KillTime" : 30000,
	"ServerPort" : 0,
	"RunScriptName" : "test_script.py",
	"RunArgument" : "-test"
}
```

ApplicationPath

これは，テスト対象のアプリのパスを記述します．もし外部操作型の自動 GUI テストをするならば，外部操作を行うプログラムを指定することになるでしょう．

テストスクリプトがあるフォルダの下の Result フォルダからの相対パスか，絶対パスで指定します．

TestFolder

テストスクリプトやテスト結果などが格納されるフォルダです．AutoGUITestCPP.exe があるフォルダからの相対パスで指定します．このフォルダには，テスト名別のフォルダがあり，その各フォルダの下にテストスクリプトが置いてあるという構造になります．

OutputHTML

すでに使われなくなった機能ではありますが，テスト結果を見るときに使う HTML ファイルの名前を指定します．

ErrorValue

この値も使われなくなっています．前回と今回の画像の差の値がこの値よりも大きいときは異常があったと見なし，HTML にその画像などを付け加えます．

KillTimer

ms 単位で，アプリが動作してよい時間を指定します．この時間を超えるとアプリは強制終了させられます．

ServerPort

このソフトをサーバー化するときは，ポート番号を指定してください．そのポートで待ち受け，接続がなされたら，一回テストを実行し，また待ち状態に入る，ということを繰り返すようになります．0 の値が指定されたら，すぐにテストを実行し，テストの実行が終わったら終了する，という動作になります．

RunScriptName

実行するテストスクリプトの名前を書きます．なお，このスクリプトはテスト対象アプリが実行される直前に Result フォルダにコピーされます．そして Result フォルダをカレントフォルダとしてテストが実行されます．

RunArgument

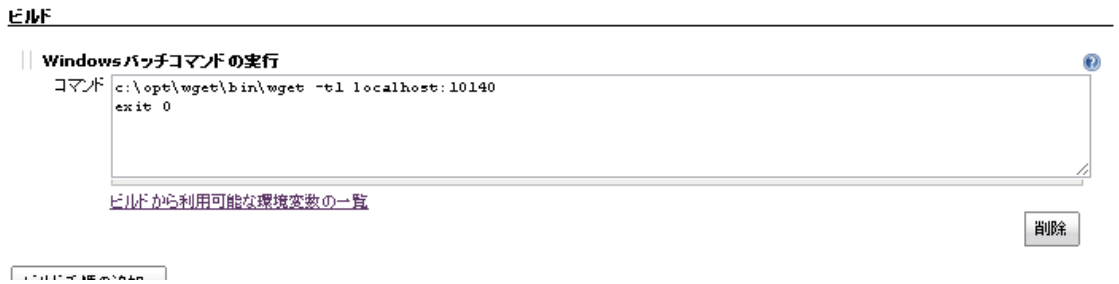
テスト対象のアプリに，テストのときに渡す引数を指定します．この引数で，対象アプリにより，テストモードで起動することなどを指示することができます．

■テスト対象アプリの実行について

テスト対象アプリは，カレントフォルダに，テスト結果の画像を出力するようにします．また，テストスクリプトは，テスト対象アプリの実行の直前に，カレントフォルダにコピーされています．

■Jenkins への登録について

config.txt の ServerPort に指定したポートを Jenkins から叩くようにします．wget を使うと確実です．　ビルドの中の Windows バッチコマンドの実行を使います．



なぜ Jenkins から AutoGUITestCpp を直接呼ばないのか，というと，直接呼ぶと，GUI 画面が表示されず，まともにテストが行えないから，です.

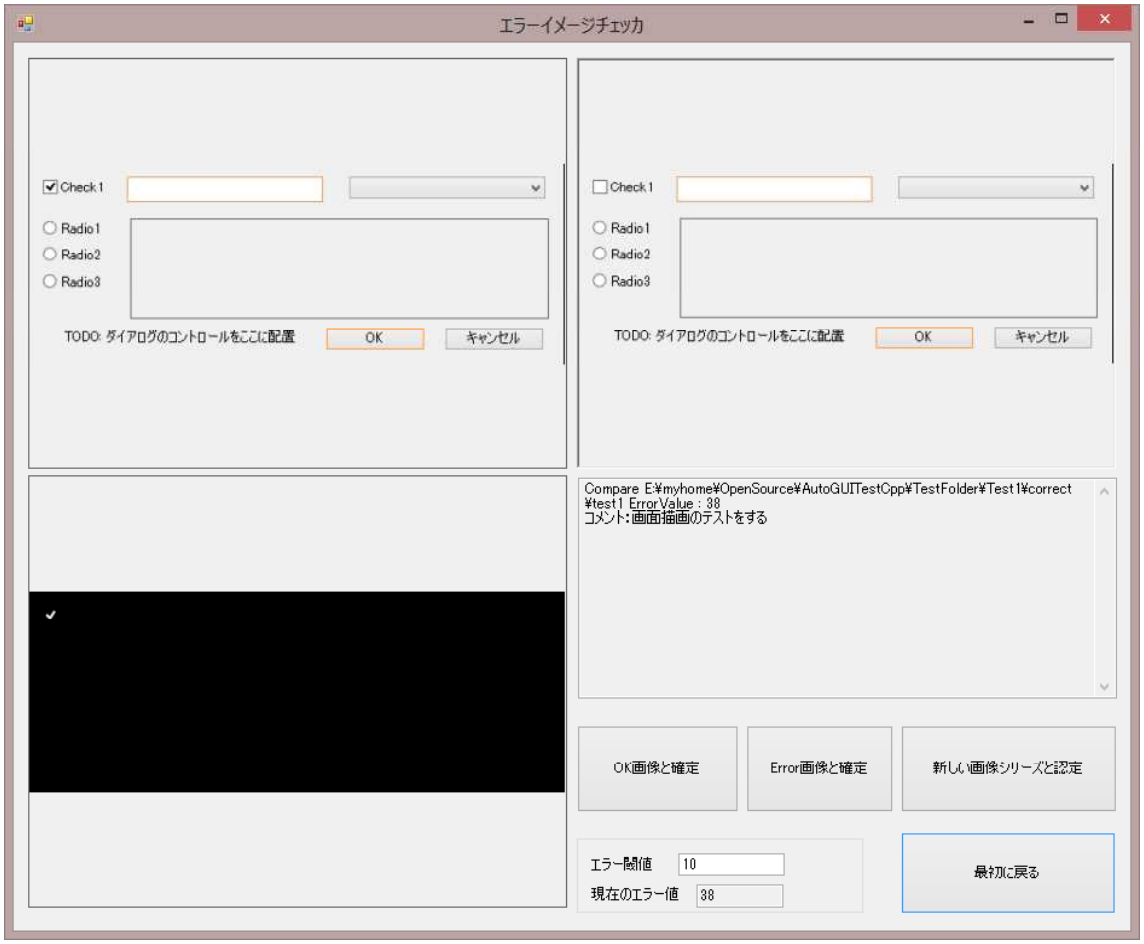
□テスト結果の目視プログラム ErrorImageViewer

■使い方

まず ErrorImageViewer.exe をテストフォルダに配置します．具体的には，Config.txt の TestFolder に指定したフォルダに置きます．

自動 GUI テストを実行後，初めて ErrorImageViewer.exe を実行すると，最初のテスト結果が自動的に正答として扱われ，”もうデータがありません”と表示されます．

2 回目以降，自動 GUI テスト結果に差異が表れたときには，下記画面のようになります．



左上が今回のテスト結果画面で，右上が前回のテスト結果画面（正解とおもわれるもの）で，左下が差分画像となります．

エラー閾値よりも大きなエラーを持つ画像が見つかると，そこで検索がストップして上画面のような待ち状態になります．

なお，エラーのカウント方法については，1 つのピクセルの輝度値が異なるとき，それをエラー 1 つと数えます．

OK 画像と確定ボタン，を押すと，今回のテスト結果画像が，正答結果として考えられ

るフォルダに移動されます．以後，エラー値の計算は，正答結果として考えられる画像すべてに対して行われ，その中での最小値を正式なエラー値とします．これにより，疑似学習を行い，誤検出を防ぎます．

異常画像と確定ボタン，を押すと，何も起きません．次の画像に移ります．再度，最初から見たときに，また，同じ画像で止まりますので，エラー画像を見逃すことはありません．

新しい画像シリーズと認定ボタン，を押すと，古い正答画像一式がすべて削除され，新たに今回のテスト結果画像が正答画像として記録されます．これは，開発により GUI をいじったときなど，明らかにバグによる差異ではないとき，新たに記録を開始するためのボタンです．

最初に戻るボタンを押すと，最初から同じ動作を繰り返しますが，異常画像，以外を選択した画像は，当然スキップされます．

□サンプルのテスト対象アプリ

■テストスクリプトの内容

テストスクリプトを見ると，このテスト対象アプリに対して，どんな操作ができるのかがわかります.

TestFolder¥Test1¥test_script.py を下記に示します.

```
#coding：UTF-8

UT_SetChekcBox(0)
UT_Sleep(1)
UT_CaptureScreen(u"test1.png")

UT_SetRadio(1)
UT_Sleep(1)
UT_CaptureScreen(u"test2.png")

UT_SetEditBox(u"test 2")
UT_Sleep(1)
UT_CaptureScreen(u"test3.png")

UT_SetComboBox(1)
UT_Sleep(1)
UT_CaptureScreen(u"test4.png")

UT_SetPicture(1)
UT_Sleep(1)
UT_CaptureScreen(u"test5.png")

UT_Sleep(5)
UT_Terminate()
```

UT_SetCheckBox，UT_SetRadio，UT_SetEditBox，UT_SetComboBox，UT_SetPicture などが GUI を内部操作するための関数です．数字を受け取る関数では，0 と 1 が指定可能です．文字列を受け取る関数に対しては u を付けて Unicode 文字にすることを忘れないでください.

また，このファイル自体の文字コードも UTF-8 にしておくことを忘れないでください.

UT_CaptureScreen で，画面のキャプチャを取り，それをカレントフォルダに指定されたファイル名で保存しています.

UT_Sleep ではメッセージループを回しつつスリープしています．メッセージループを回さないと，単にハングアップした感じになり，期待したとおりには動作しません.

各々の関数をどうやって実装しているか，などは，ソースを見てみてください.

以上