Efficient Acceptance Testing Framework for Interactive Computer Game Applications

Wook Hyun Kim  
Department of Computer Engineering, Yeungnam University, Gyeongsan, Republic of Korea.

Young-Ho Sohn  
Department of Computer Engineering, Yeungnam University, Gyeongsan, Republic of Korea.

Dong Won Park  
Department of Game Engineering, Pai Chai University, Daejeon, Republic of Korea.

Abstract
In modern computer games market, highly automated testing becomes one of key factors of success. At current stage, industry of computer games is not mature enough for automated testing. The paper describes method of bringing automated quality assurance techniques from more established computer engineering fields to computer game development. Automated acceptance testing framework, based on Python, is created to help implementing agile test-driven development into creation of interactive computer game applications.

Keywords: Automated acceptance testing, Test-driven development, Refactoring, Python, Co-routines.

Introduction

A. The Need of Automated Testing
There is no unified approach for automated testing. Some experienced companies try to develop their own in-house verification solutions that do not share any common base. While still most of the companies relies on hard work of quality assurance department, that do all the testing manually. These results in three major problems: human resource problem, human error problem, time problem. The below text will provide overview of each problem.

Human resource problem – nowadays games has a huge set of features that need to be tested. These features can include core game mechanics, online server integration, database server integration, various input device support, high-end graphics card support, low-end graphics card support, etc. Game play time for modern computer games cause additional problem during testing. As an example, approximate time to finish Max Payne 3 is 10 hours, and Crysis 2 is 12 hours. Although a QA team does not have to play a game from start to end every testing session time, it still requires much time to complete important test cases.

Human error problem – As human being we are not guaranteed to perform correctly all the time. Highly stressed environment, lack of motivation, and constant deadlines can lead people to make more mistakes.

Time problem – Although we can have big enough QA team, there is still a critical path for testing. The path cannot be broken down on parallel items, and should be performed completely by a single person.

B. Test Driven Development
Using automated testing gives benefits not only for code that is already written. A well established and popular technique, called test driver development, or TDD, suggests writing tests before the actual functionality is implemented. In context of test-driven development a test is not used solely as verification mechanism, but also as guidance for both design and implementation. By having test in place, we define scope of future functionality. Consequently, making the test pass shows that feature is completed and covers the scope.

A developer who uses Test-Driver Development is working within next loop:
  i. Write failing test case.
  ii. Get it to compile.
  iii. Make it pass by writing just enough functionality to make the test working.
  iv. Refactor code.
  v. Repeat.

The figure below shows relationship between items in loop items.

C. Automated Testing and Refactoring
Refactoring is a technique of changing the internal implementation without changing the external behavior. Changing implementation without adding any useful feature can be seen as unacceptable waste of resources, especially by non-technical people. However, it is proven that without
It is possible to translate this analogy to interactive gaming applications. In context of games, we have predefined set of input (input received from user controllers: keyboard, mouse, joysticks, etc.), and we have predefined set of outputs (an image on monitor, sound effects, vibration feedbacks, etc.). This is shown on Fig. 3.

![Figure 2: GCC compiler generates output.obj and output.log from input.c](image)

Fig. 2 shows how GCC Compiler transforms input files into object file (output.obj) and log file (output.log). In case of failure in processing input.c the output object file is not generated. However, the output.log is generated always, and it is from output.log we can get the actual reason of the compilation error.

It is possible to translate this analogy to interactive gaming applications. In context of games, we have predefined set of input (input received from user controllers: keyboard, mouse, joysticks, etc.), and we have predefined set of outputs (an image on monitor, sound effects, vibration feedbacks, etc.). This is shown on Fig. 3.

![Figure 3: Game Application Process User Input and Generates Output](image)

The original motivation for the solution was the way programming language compilers are developed and tested. Programming language compilers have limited set of inputs and outputs corresponding to the input. The actual input is defined by language specification, and thus well predictable, even if the underlying implementation does not support it yet.

**Solution Description**

**A. Motivation**
The original motivation for the solution was the way programming language compilers are developed and tested. Programming language compilers have limited set of inputs and outputs corresponding to the input. The actual input is defined by language specification, and thus well predictable, even if the underlying implementation does not support it yet.

It is usual case when some new the features of programming language are not supported by compiler. A programming language compiler developer can write new code directly against the input file. If the output still contains errors, the new language feature is not working correctly. When the feature is completed, the compiler generated certain output, a binary file created from output, and no errors in log.

The input file is then added to compiler test suite, which is regularly run to ensure that there are no regressions in the functionality of compiler. In this context, tests perform play several roles: ensuring that compiler is working correctly, defines the scope of implemented features, defines the scope of not supported features, ensures that there is no regressions, after a new feature is added to the compiler.

**B. Sensing**
In computer program every action has some sort of feedback or response. The analysis of the feedback can tell whether the action lead to successful results or failed. The feedback can have various forms: showing a message box with details of results, updating database, and moving robotic arm in certain direction, playing sound or displaying certain character animation in computer games.

Some feedbacks are easy to analyze, while some requires not trivial efforts. For example, moving robotic arm requires having robot connected to test machine during the verification session. Another example is testing real world banking operations, which requires creation and manipulation of real money. More specific to computer games, are getting feedback of graphics or sound systems: rendering objects with selected shader, or playing sounds on object collisions. While those types of feedbacks are easy to analyze by having human as testers, those are hard to automate or sense.

In the paper, *sensing* is defined as ability to automatically recognize feedback of a system. Sensing assumes removing human from equation of testing, which leads to more effective and valid testing.

Most commonly, a computer game is graphical application with high number of varying components. In contrary to established user interface (UI) Desktop application frameworks, a game does not have a common UI framework. Therefore, the absence results in inability of creating a testing framework that can cover every computer game. While for desktop application we have number of solutions.

The proposed solution for sensing is using the most common way of outputting information from the game application, log file. An acceptance testing framework starts an application in parallel, and monitors changes made to log file with certain interval. Every single line is processed by framework and transformed to events. Later, the events are analyzed with user provided event processing plugins. The main responsibility of the plugins is transforming raw events to high level testing events.

**C. Controlling**
The section describes a way the testing framework is controlling the execution of game. Certain interface between a test and a game application must be created in order to execute tests. The interface is based on TCP communication: Game acts as server, while acceptance test framework as client. The communication is abstracted using small module that is designed for easy integration with gaming application (Fig. 4). In production the interface can be disabled, or used for remote debugging in some complex cases. This is especially useful in situation when a problem cannot be reproduced in local developer environment.

![Figure 4: TCP Protocol based Communication between Game Application and Testing Framework](image)

### D. Faking

In some cases, it is impossible to always use real components during automated testing. For example, a system integrated with third-party payment system cannot be used during automated testing sessions, because the testing requires handling real money transactions. The better approach would be using a test system that replicates behavior of the real payment system. There is no out of the box solution included into the testing framework. There is no way to fake it from outside of the application. Thus, proper care should be taken by the application designers. A module that communicates with the payment system must be well encapsulated and interchangeable by fake payment module.

### Implementation Description

#### A. Testing Framework Introduction

In result of research, many generic cases were found, that are used repeatedly during acceptance test case creation for different projects. The boilerplate repeated code was encapsulated in a testing framework with working name Sentinel. The section provides information about next:

- Detail on design and implementation of Sentinel testing framework.
- Integration of testing code with existing game code.

Python programming language was chosen for both the framework implementation and test cases definition, because it is high-level powerful language and relatively easy to study. Python abstracts us from low-level programming problems, such as memory management, pointer usage, platform specific code, etc. The disadvantage is the performance of Python, as it is interpreted language that runs on a virtual machine. As testing framework is designed to do a performance critical computation, the disadvantage is diminished.

### B. gevent Library

The framework is running game application is separate process, which means that existing unit-testing frameworks (such as unit test module from standard Python library) cannot be used as foundation for the acceptance testing framework. Sentinel framework is launching game application, sending messages to it, and waits for the game to produce output in log file. An obvious solution would be using threads. However, Python provides high-level entities for asynchronous communication, called greenlets. More specifically the framework is using gevent - a coroutine-based networking library that uses greenlets to provide high-level asynchronous API on top on libevent event loop.

### Experimentation and Results

#### A. Selecting Existing Interactive Computer Game Application

Since programming language for the framework is Python, it was logical to select Python-based game application for the purpose of testing. The most widely framework for making games using Python is PyGame. The official website of the PyGame provides many games based on PyGame framework. For the purpose of testing, the rpgworld game was selected ([http://code.google.com/p/rpg-world/](http://code.google.com/p/rpg-world/)). The genre of the game is top-down side-scrolling arcade game, similar to classic Legends of Zelda game. This provides good foundation for our test, while does not limit in future feature extension. Figure 5 shows the screenshot of rpgworld game that was selected for experiment.

#### B. Integrating Testing Framework Functionality with the Game

Integration of the testing framework requires making certain changes to the game's source code. There are two major changes that should be done: log format changes and test server integration. Log messages must be changed in order to support proper sensing functionality. All the messages written to log file must be changed to next format `{<LogCategory> } <LogMessage>`. Following log format allows Sentinel testing framework to automatically parse log messages. Later, the log message is processed by user defined code inside of test case. Test Server functionality integration consists of adding simple TCP-based server. The role of server is to accept messages from test case or test console. For Python programming language based games, the test server functionality is provided by acceptance framework itself. Figure 6 shows the integration of test server into rpgworld game.
C. Experimental Results

The testing time was greatly decreased. The coin achievement automated test takes just a one 2 seconds to be tested, while manual test would take around 12 seconds and requires additional typing from tester’s side. The table 1 provides details about performed testing. The win in testing time is apparent.

<table>
<thead>
<tr>
<th>Test name</th>
<th>Manual</th>
<th>Automated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Application</td>
<td>2</td>
<td>0.3</td>
</tr>
<tr>
<td>Test Level Loading</td>
<td>4</td>
<td>0.5</td>
</tr>
<tr>
<td>Coin Achievement</td>
<td>15</td>
<td>2</td>
</tr>
</tbody>
</table>

In order to start testing only a single command must be executed from console. Then failure during the test execution is automatically detected, and, possibly, could be sent to developers by email. Therefore, it is possible to execute the tests at night time. Later, in the morning the programming team can get comprehensive report about failures. The number of lines to integrate testing functionality is considered to be extremely small comparing to original source code lines. The original number of code lines is 3664, while the added number of code lines is 187. Therefore, with adding 5% of code we have integrated automated testing to already existing game application.

D. Future Work

At the moment, framework does not actively provide utilities for Faking part. Since, Faking is an important concept that supports automatic testing; the area should be further explored. There are many libraries that provide mock functionality for different languages, but there is no solution that can be easily integrated into any game application. Nowadays, middleware used for making computer games is widely varying. Some provide completely toolset to produce games; others are used as library that covers one specific function of a game, such as rendering, physics or sound. One of the future works, that immediately gives benefit to a framework user is automated report about test code coverage. The code coverage provides information about which part of underlying code was actually called. This information helps both to identify unused pieces of code ("dead" code), and optimize test cases. Test case optimization means having fewer test cases, while covering more code.

Conclusion

In modern computer games market, highly automated testing becomes one of key factors of success. The problem becomes more severe in online gaming market, where the frequency of game releases is high. In spite of it, majority of game companies does not apply automated testing during development. Several team tried to introduce automated testing, with different degree of success and scope of application. Based on analysis of the attempts and advances in computer engineering related to quality assurance, the paper outlined core ideas for successful testing automation. The core ideas are: sensing - ability to get feedback from application in automated way; controlling - ability to control application from test; faking - ability to test application in isolation by replacing expensive dependencies with stub modules. As result of the paper, automated testing framework, Sentinel, was created. The testing performed with the framework and existing game showed great results, both in integration cost and testing automation results. The research results can already be applied to real game development, and still contains many opportunities for improvements, which will be handling within future research.

Acknowledgement

This work was supported by the 2015 Yeungnam University Grant. The corresponding author is Dong-Won Park.

References

[3] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-
Oriented Software. Addison-Wesley Professional (1994)


