УДК 004.4′23
DOI: 10.14489/vkit.2020.12.pp.047-052

В. В. Кугуракова, канд. техн. наук, **О. А. Бедрин**
(Казанский федеральный университет, Казань, Республика Татарстан);
e-mail: vlada.kugurakova@gmail.com

СИСТЕМА АВТОМАТИЗАЦИИ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ ДЛЯ ПЛАТФОРМЫ UNITY

Предложен обход технического ограничения стандартного класса Input, *который позволяет значительно сократить время этапа тестирования приложений. Инструмент автоматического выполнения и проверки тестовых наборов действий не является нативным для приложений, что вызывает проблемы с распознаванием элементов взаимодействий из-за обилия* (*отсутствия*) *графического интерфейса пользователя. Описан процесс проектирования и разработки внутреннего инструмента, точно распознающего элементы взаимодействий путем декорирования механизмов считывания ввода игрового движка Unity. Ранее для Unity не было решений для автоматизации success-тестов и других подходов для функционального тестирования приложений, кроме как через механизмы стандартного пакета unit-тестирования. Однако для того чтобы покрыть все возможные сценарии действий внутри проекта, пришлось бы писать свой модуль тестов под каждый из аспектов либо на его основе реализовывать сложную универсальную систему. Кроме решения задач тестирования, такая форма записи действий пользователя в виртуальной среде может быть использована для генерации сценариев обучающих тренажеров, что поможет избежать рутинной работы по формированию треков обучения. Так, погружения в виртуальную биотехнологическую лабораторию эксперта, который может правильно пройти все фазы выбранного анализа, достаточно, чтобы записать контрольные точки такого сценария для обучения студентов.*

Ключевые слова: функциональное тестирование; тест на нормальность; дымовое тестирование; тест удобства использования; Unity; ориентированная на данные группа технологий (DOTS); сущностно-компонентная система (ECS).

V. V. Kugurakova, O. A. Bedrin (Kazan Federal University, Kazan, Tatarstan Republic)

FUNCTIONAL TEST AUTOMATION SYSTEM FOR UNITY APPLICATIONS

The tool for automatically executing and validating test sets of actions is not native to applications, which causes problems with recognizing interaction elements due to the abundance or, conversely, lack of GUI elements. This work describes the design and development of an internal tool that provides accurate recognition of interaction elements by decorating the input reading mechanisms of the Unity game engine. Previously, there were no solutions for Unity to automate success tests, testing the main flow of a use case, or other approaches for functional testing of applications, except through the mechanisms of the standard unit testing package. However, no matter how good the approach of using unit testing tools is, in order to cover all possible scenarios of actions within the project, you would have to either write your own test module for each of the aspects, or implement a complex universal system on its basis. The article's graceful workaround for the technical limitation of the standard Input class dramatically reduces the time required for the application testing phase. In addition to solving testing problems, such a form of recording user actions in a virtual environment can be used to generate training simulator scenarios, which will help to avoid the routine work of generating training tracks. So, immersion in a virtual biotechnology laboratory of an expert who can correctly pass all phases of the selected analysis (for example, ELISA) is enough to record the checkpoints of such a scenario for teaching students.

Keywords: Functional test; Sanity test; Smoke test; Usability test; Unity; DOTS; ECS.

Статья поступила в редакцию 13.08.2020 г.

Введение

Проведем расширение редактора платформы *Unity* для записи и проигрывания сценариев, представляющих собой входные данные со всех периферийных устройств, направленного на автоматизацию одного из видов тестирования «белого» ящика, а именно на функциональное тестирование приложений. Попутно решим ряд дополнительных задач:

– создание расширяемой базы кода, которая отвечала бы всем требованиям принципов SOLID [1] (пять базовых принципов объектно-

ориентированного программирования), *DRY* (*Don't Repeat Yourself* – «не повторяйся») [2], бритвы Оккама [3];

– компоновка решения для легкой переносимости;

– обеспечение интеграции с существующими продуктами.

В документации платформы (или игрового движка, как ее принято называть в сообществе) *Unity* используется ряд устойчивых терминов, не употребляемых в другой литературе по программированию, которые поясним по ходу изложения.

Класс совместимых приложений. Рассмотрим набор ассетов* ATF (*ATF – Automated Test Framework* – фреймворк автоматизированного тестирования), используемый для автоматизации действий в целях функционального тестирования. Его можно интегрировать с любым проектом, система ввода которого построена вокруг класса из стандартной библиотеки *Unity* по взаимодействию с вводом Input. Под это описание подходит бóльшая часть Unity-приложений, и в этом состоит универсальность предложенного решения.

Ранее для Unity не было решений для автоматизации ни success-тестов, ни тестирования главного потока сценария использования, ни других подходов для функционального тестирования приложений, кроме как через механизмы стандартного пакета unit-тестирования. Однако как бы ни был хорош подход использования инструментария unit-тестирования, для того чтобы покрыть все возможные сценарии действий внутри проекта, пришлось бы либо писать свой модуль тестов под каждый из аспектов, либо на его основе реализовывать сложную универсальную систему. Актуальность задачи автоматического тестирования повышается с ростом стоимости разработки проекта.

Выбор класса Input в качестве основы для перехвата и записи связан с тем, что этот инструмент популярен и распространен среди других Unity-проектов. Авторы провели выборку из Unity-проектов, размещенных на GitHub**, с преобладающим количеством использования CSharp и с максимальным количеством форков – проектов, в основе которых лежит выбранный проект, – со стабильным обновлением и максимальной поддержкой своих сообществ. Размер такой выборки составил 3970 проектов из 99 509. Для наглядной оценки математического ожидания количества использований был выбран бутстраповский доверительный интервал с уровнем доверия 90 %, подтвердивший нулевую гипотезу о преобладающем использовании классов ввода в Unity-проектах.

Решаемые задачи. Для реализации проекта необходимо было определить архитектурные паттерны проектирования, концепцию системы как главной единицы управления бизнес-процессами, а также перечень конкретных необходимых систем, структурные паттерны проектирования для системы записи действий, поведенческие паттерны проектирования для системы проигрывания, структурные паттерны для системы хранилища записанных данных, для системы сохранения и загрузки хранилища и для системы инъекции зависимостей.

Модули проекта реализованы с использованием базового архитектурного паттерна «Композиция», который в скором времени устареет. Ему на смену придет *DOTS* (*Data-Oriented Technology Stack*) – множество технологий, направленных на создание кода, который опирается на данные с учетом оптимизации, т.е. стек, в котором процесс проектирования ведется от данных, а не от области применения [4]. Однако при таком стечении обстоятельств разработанная библиотека инстру-ментов не теряет технологических преимуществ, поскольку ее архитектура позволяет без лавинного эффекта расширять и дополнять основной функционал. Благодаря этому довольно легко обернуть любой компонент предлагаемого решения, используя паттерн *ECS* (*Entity Component System*), – структурный паттерн, позволяющий быстро читать, создавать и удалять большие массивы единиц данных из DOTS. При этом получаем гибридный комплекс данных и логики [5].

Дополнительное требование при реализации данного проекта – возможность сериализовы-

* Ассет (англ. asset) – группа простых и составных цифровых ресурсов, которая хранится в Unity-проекте.
** GitHub.com – крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

вать, экспортировать и импортировать записанные данные для последующего использования в построении сценария для тренажеров в виртуальной реальности [6]. Такая форма записи действий пользователя в виртуальной среде может быть использована для генерации in situ-сценария обучающего тренажера [7] при записи правильного прохождения всех последовательных действий выполняемого эксперимента или технологического процесса, что поможет избежать рутинной работы по формированию треков обучения [8].

Связанные работы. Задача автоматизации функционального тестирования сегодня считается тривиальной, так как во многих популярных платформах (Web, iOS, Android) она уже решена. Существуют инструменты для приложений с различными возможностями и характеристиками, например Selenium, Appium, Robotium, UI Automator. Формы решения задачи функционального тестирования существуют разные, например:

– для Web – WebDriver для записи и имитации ввода [9];

– для iOS и Android – система RPC-вызовов [10];

– компьютерное зрение для распознавания, как и с чем взаимодействует пользователь [11], если нет возможности вмешаться в поток вывода для облегчения записи.

Особняком стоит вопрос об автоматизации тестирования приложений на платформе Unity. Каждая упомянутая выше форма решения задачи в этом случае проигрывает в эффективности степенью удобства интеграции, так как зачастую приложения, разработанные на Unity, насыщены графикой и в общем случае, не все имеют графический интерфейс. Предлагается новая форма решения задачи, основанная на внутренней логике *Unity*, а именно на возможности декорирования методов взятия ввода, обусловленной внутренней архитектурой системы.

Модифицированный адаптер

Общее техническое решение можно представить в виде диаграммы платформы, представленной на рисунке. Каждый ее блок – это изолированная группа методов *API* (*Application Programming Interface*), решающая свои задачи:

– непосредственно сам игровой движок *Unity*;

– абстракция, объединяющая управление вводом;

– собственно API, который вызывает нативные методы по запросу ввода;

– сущность, которая является реализацией объекта обработки потока данных через мост, объединяя статичные методы по перехвату/симуляции ввода и обернутые события;

Диаграмма платформы решения

– абстракция хранения и манипуляции записанных действий;

– абстракция записи действий;

– абстракция модуля по сохранению/загрузке записанных действий на базе стандартного класса PlayerPrefs;

– абстракция хранилища записанных действий.

Для выполнения поставленных задач был разработан модифицированный адаптер, который перехватывает и симулирует ввод. Для его эффективной реализации органично использовать несколько архитектурных паттернов*:

– Interceptor – для перехвата и подмены входных данных с периферийных устройств [12];

– Broker – для интеграции и взаимодействия с встроенной системой управления входными данными *Unity* [13];

– *PAC* (*Presentation–Abstraction–Control*) – для организации взаимодействия зависимых систем [14].

Для подмены стандартного класса Input был создан перехватчик ATFInput.

Так как класс Input содержит только статичные методы, декорация их для перехвата внутри ATFInput позволила совместить классический перехват ввода и облегчить в дальнейшем перехват событий ввода.

DI-контейнер для объектов сцены. В качестве основного функционала контейнера представлен метод, который осуществляет проверку на участие в инъекции зависимостей, поиск подходящей зависимости и непосредственно инъекцию. Все эти этапы работают по известному механизму, описанному в стандарте JSR-299 [15].

Адаптер для класса Input. Бизнес-процесс класса адаптера, в котором осуществляются перехват и вычисление относительно текущего состояния систем управления записью и хранилища, описывается тремя ключевыми методами, которые вызываются стеком. По перехвату входных данных вычисляется результат из хранилища независимо от того, записывается ли процесс или проигрывается.

Реализация системы основана на главном свойстве структуры данных «Очередь» – когда происходит запись, то пара FakeInput (тип перехватываемого ввода) и объект значения записывается в очередь, которая располагается внутри системы хранилища. При проигрывании выполняется выгрузка данной очереди в отдельную переменную и при каждом вызове перехватываемого метода происходит выдача того, что было первым в очереди и далее, пока она не опустошится.

Система сохранения и загрузки является PAC-компонентом, она используется напрямую только хранилищем записанных действий. За ее реализацию отвечает интерфейс, основанный на работе с встроенным классом PlayerPrefs. Для сохранения сложных объектов эта реализация использует сериализацию в JSON – объект всего хранилища либо только одной записи в зависимости от метода, который вызывается.

Интеграция с Unity-проектом

Для внедрения в любой Unity-проект возможностей автоматизации функционального тестирования ATF используется система интеграции, которая может функционировать в двух режимах:

– автоматическом. В этом случае автоматически сканируется вся кодовая база еще не интегрированного и поэтому стороннего проекта, находятся файлы исходного кода с расширением ".cs", сигнализирующим о том, что именно они и являются скриптами проекта (а не прочими ресурсами), и по очереди с помощью жадного перебора выполняется замена имени класса Input на ATFInput.Instance, при этом не создавая новых файлов с интегрированным перехватчиком;

– полуавтоматическом. При этом система наполняется путями к файлам исходного кода вручную с помощью вспомогательного окна, а затем задействует подстановку, как и в случае с автоматическим режимом: создать дополнительный файл с суффиксом ATF, где будет

* Паттерн проектирования – часто встречающееся решение определенной проблемы при проектировании архитектуры программ.

внедрен перехватчик, или же провести замену без создания дополнительного файла. Все дальнейшее управление системой интеграции происходит через графической пользовательский интерфейс.

Прагматичность использованного подхода хорошо иллюстрирует видео [16] с кубом, который крутится в ответ на действия пользователя, получаемые с ATFInput, с отражением всех промежуточных состояний.

Причем при создании окон управления не были использованы ни «singletone» заготовки, ни написанный DI-контейнер ввиду особенностей распределения времени выполнения у *Unity* – пространство имен, взаимодействующее с классами, которые регламентируют внешний вид и поведение пользовательских окон изолированно от других пространств, что делает невозможным использование DI и других вспомогательных структур.

Заключение

Реализованный комплекс ассетов позволяет автоматизировать процесс тестирования приложений *Unity*, используя его собственные внутренние механизмы и инструменты. После простой интеграции любой Unity-проект, использующий для ввода класс Input (таких приложений подавляющее большинство), получает в дополнительном интерфейсе отражение всей гаммы промежуточных состояний.

Предложенное адаптируемое решение, на данный момент единственное, успешно прошло модерацию и было опубликовано в официальном магазине ассетов Unity Asset Store. Благодаря обходу технического ограничения стандартного класса Input можно сократить время этапа тестирования и параллельно решать важные задачи по уменьшению рутинной работы в проектировании сценариев для виртуальных тренажеров.

Библиографический список

1. **Мартин Р. С., Ньюкирк Д. В., Косс Р. С.** Быстрая (гибкая) разработка программ на Java и C++: принципы, шаблоны, практика. М.: Вильямс, 2001. 752 с.

2. **Haoyu W., Haili Z.** Basic Design Principles in Software Engineering // Fourth International Conference on Computational and Information Sciences. IEEE Computer Society. 17 – 19 August 2012. Chongqing, China. 2012. P. 1251 – 1254.

3. **Occam's** Razor / A. Blumer, A. Ehrenfeucht, D. Hasler, M. K. Warmuth // Information Processing Letters 24. 1987. No. 6. P. 377 – 380.

4. **High-Performance** Multithreaded Stack of Unity Information-Oriented Technologies [Электронный ресурс]. URL: https://unity.com/ru/dots (дата обращения: 13.10.2020).

5. **Unity** Foundation – Introduction to ECS [Электронный ресурс]. URL: https://unity3d.com/ru/learn/tutorials /topics/scripting/introduction-ecs (дата обращения: 13.10.2020).

6. **Кугуракова В. В.** Математическое и программное обеспечение многопользовательских тренажеров с погружением в иммерсивные виртуальные среды: дис. ... канд. техн. наук: 05.13.11; защищена 18.04.19 / Кугуракова Влада Владимировна. Казань, 2019. 187 с.

7. **Kugurakova V. V.** Automated Approach to Creating Multi-user Simulators in Virtual Reality // CEUR Workshop Proceedings. 2018. No. 2260. P. 313 – 320.

8. **Visual** Editor of Scenarios for Virtual Laboratories / V. V. Kugurakova et al. // Developments in the Design of Electronic Systems. Conference Publication Services (DeSE 2017). 14 – 16 June 2017. Paris, France, 2017. P. 242 – 247.

9. **Vila E., Novakova G., Todorova D.** Automation Testing Framework for Web Applications with Selenium WebDriver: Opportunities and Threats // Proceedings of the International Conference on Advances in Image Processing (ICAIP 2017). 25 – 27 August 2017. Bangkok, Thailand, 2017. P. 144 – 150.

10. **Performance** of Automation Testing Tools for Android Applications / A. M. Sinaga, P. Adi Wibowo, A. Silalahi, N. Yolanda // 10th International Conference on Information Technology and Electrical Engineering (ICITEE). 24 – 26 July 2018. Bali, Indonesia, 2018. P. 534 – 539.

11. **Mozgovoy M., Pyshkin E.** Unity Application Testing Automation with Appium and Image Recognition // In: Tools and Methods of Program Analysis. TMPA 2017. Communications in Computer and Information Science. 2017. No. 779. P. 139 – 150.

12. **Pattern-Oriented** Software Architecture / D. Schmidt, M. Stal, H. Rohnert, F. Buschmann // Patterns for Concurrent and Networked Objects. 2001. No. 2. P. 109 – 140.

13. **Architectural** Pattern "Broker" [Электронный ресурс]. URL: https://cs.uno.edu/~jaime/Courses/4350/ broker.ppt (дата обращения: 13.10.2020).

14. **Coutaz J.** PAC: An Object-oriented Model for Implementing User Interfaces // ACM SIGCHI Bulletin. 1987. No. 19. P. 37 – 41.

15. **JSR 299**: Contexts and Dependency Injection for the JavaTM EE platform [Электронный ресурс]. URL: https://jcp.org/en/jsr/detail?id=299 (дата обращения: 13.10.2020).

16. **Automated** Test Framework DevScene [Электронный ресурс]. URL: https://youtu.be/yQSvvuGT9MI (дата обращения: 13.10.2020).

References

1. Martin R. S., N'yukirk D. V., Koss R. S. (2001). *Fast (flexible) development of programs in Java and C ++: principles, patterns, practice.* Moscow: Vil'yams. [in Russian language]
2. Haoyu W., Haili Z. (2012). *Basic Design Principles in Software Engineering.* Fourth International Conference on Computational and Information Sciences, pp. 1251 – 1254. IEEE Computer Society. Chongqing.
3. Blumer A., Ehrenfeucht A., Hasler D., Warmuth M. K. (1987). Occam's Razor. *Information Processing Letters 24,* (6), pp. 377 – 380.
4. *High-Performance Multithreaded Stack of Unity Information-Oriented Technologies.* Available at: https://unity.com/ru/dots (Accessed: 13.10.2020).
5. *Unity Foundation – Introduction to ECS.* Available at: https://unity3d.com/ru/learn/tutorials/topics/scripting/introduction-ecs (Accessed: 13.10.2020).
6. Kugurakova V. V. (2019). *Math and software for multi-user simulators with immersion in immersive virtual environments.* Kazan'. [in Russian language]
7. Kugurakova V. V. (2018). Automated Approach to Creating Multi-user Simulators in Virtual Reality. *CEUR Workshop Proceedings, 2260,* pp. 313 – 320.
8. Kugurakova V. V. et al. (2017). *Visual Editor of Scenarios for Virtual Laboratories. Developments in the Design of Electronic Systems.* Conference Publication Services (DeSE 2017), pp. 242 – 247. Paris.
9. Vila E., Novakova G., Todorova D. (2017). *Automation Testing Framework for Web Applications with Selenium WebDriver: Opportunities and Threats.* Proceedings of the International Conference on Advances in Image Processing (ICAIP 2017), pp. 144 – 150. Bangkok.
10. Sinaga A. M., Adi Wibowo P., Silalahi A., Yolanda N. (2018). *Performance of Automation Testing Tools for Android Applications.* 10th International Conference on Information Technology and Electrical Engineering (ICITEE), pp. 534 – 539. Bali
11. Mozgovoy M., Pyshkin E. (2017). Unity Application Testing Automation with Appium and Image Recognition. In: Tools and Methods of Program Analysis. TMPA 2017. *Communications in Computer and Information Science, 779,* pp. 139 – 150.
12. Schmidt D., Stal M., Rohnert H., Buschmann F. (2001). *Pattern-Oriented Software Architecture.* Patterns for Concurrent and Networked Objects, (2), pp. 109 – 140.
13. *Architectural Pattern "Broker".* Available at: https://cs.uno.edu/~jaime/Courses/4350/broker.ppt (Accessed: 13.10.2020).
14. Coutaz J. (1987). *PAC: An Object-oriented Model for Implementing User Interfaces.* ACM SIGCHI Bulletin, 19, pp. 37 – 41.
15. *JSR 299: Contexts and Dependency Injection for the JavaTM EE platform.* Available at: https://jcp.org/en/jsr/detail?id=299 (Accessed: 13.10.2020).
16. *Automated Test Framework DevScene.* Available at: https://youtu.be/yQSvvuGT9MI (Accessed: 13.10.2020).

При цитировании использовать:

Кугуракова В. В., Бедрин О. А. Система автоматизации функционального тестирования для платформы Unity // Вестник компьютерных и информационных технологий. 2020. Т. 17, № 12. С. 47 – 52. DOI: 10.14489/vkit. 2020.11.pp.047-052

Kugurakova V. V., Bedrin O. A. (2020). A functional test automation system for the Unity platform. *Vestnik komp'yuternyh i informatsionnyh tekhnologiy, Vol. 17,* (12), pp. 47 – 52. [in Russian language] DOI: 10.14489/vkit. 2020.12.pp.047-052

ООО «Издательский дом «Спектр», 119048, Москва, ул. Усачева, д. 35, стр. 1.
Http://www.idspektr.ru. E-mail: info@idspektr.ru
Учредитель – ООО «Издательский дом «Спектр».
Редакция журнала: тел. **(495) 589 56 41, (495) 514 76 50; http://www.vkit.ru; e-mail: vkit@idspektr.ru**
Корректор *Евсейчев А. И.* Инженеры по компьютерному макетированию: *Евсейчев А. И.*, *Смольянина Н. И.*
Сдано в набор 11.10.20 г. Подписано в печать 30.11.20 г. Формат 60×88 1/8. Бумага офсетная. Печать офсетная.
Усл. печ. л. 6,37. Уч.-изд. л. 6,1. Свободная цена.
Оригинал-макет и электронная версия подготовлены в ООО «Издательский дом «Спектр».
Отпечатано в типографии ООО «Белый Ветер»
115054, Москва, ул. Щипок, 28. E-mail: wwprint@mail.ru. Http://www.wwprint.ru