



Expertise
and insight
for the future

Wael Awad

Game Testing Automation Guidance

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

22 Sep 2021

PREFACE

Thanks to all Kuuasema Oy (1) personnel for supportive inputs to surveys and interviews of data collection. Special thanks to Ville Jääskeläinen as my instructor for this thesis and special thanks to my family and specially my wife for creating the atmosphere that enabled me to do this study.

Special thanks to the interviewed Kuuasema personal, Panu Alku, Olli Pekkarainen, Johannes Heinonen, Tom-Johan Björklund, Pauli, Jani Hämäläinen, Niko Rintala, Oona Saloranta and from our customer side Alejandro Olmos Pardo and Dmytro Naida. Big thanks also to Tuomas Luohelainen for support with the game code and test code, as well Jimi Willing and Johannes Heinonen again for closer support with the implementation section.

Espoo, 22.09.2021

Wael Awad

Author	Wael Awad
Title	Game Testing Automation Guidance
Number of Pages	38 pages + 14 appendices
Degree	Master of Engineering
Degree Programme	Information Technology
Instructor(s)	Ville Jääskeläinen, Head of Information Technology Master's program
<p>This thesis is a typical industrial case study, and the Kuuasema environment was used to execute the idea behind the study. The target was to automate tests in gaming applications and get a clear guide on how the company could proceed in this field.</p> <p>The problem addressed in the study was that the local market is lacking a simple guidance to do test automation in the game industry. Thus, it was important to conduct this study and provide this type of guidance especially for newcomers in the game industry, saving a lot of time when setting up the work environment. The study also shows some real, simple code examples to the Quality Assurance (QA) engineers to proceed and see things in a simple way.</p> <p>The study was straightforward starting from definitions and the selection of the required SW until starting to use and combine everything with code samples to see actual results.</p> <p>The test application frameworks are something that have existed for long and many new SW applications are being created rapidly to support more and more application types. New frameworks, new execution environments, new programming languages, more user-friendly interfaces are coming to market all the time. To this point, it is hard to focus on all of them. This thesis selected the game industry for the scope and just focused on driving and guiding the user as to how to proceed in setting up an environment and getting ready to produce a simple game to be tested. Each company has its market needs to focus on, but the case company focuses on iOS and Android devices and taps.</p> <p>The thesis also discusses the best practices a QA person could benefit from.</p>	

Keywords	Test frameworks, Unity testing, testing guidance, Game testing, Game test automation

Contents

1	Introduction	1
1.1	State of Art	3
1.2	Problem Statement and Objective	4
1.3	Thesis Structure	4
2	Kuuasema Company State Analysis	6
2.1	Automation Relevancy Survey Results	6
2.1.1	Kuuasema Survey Chart Results Regarding Game TAF	6
2.1.2	Kuuasema Free Text-based Survey Results for Game TAF	11
2.2	Interviews Summary by Functions	14
3	Automated Testing	19
3.1	Why Automation	19
3.2	Types of Test Automation Frameworks	21
3.3	Test Automation Frameworks Selection	23
3.3.1	AltUnity Tools from Altom	23
3.3.2	AltRunner Tool from Altom	25
3.3.3	AltWalker Tool from Altom	25
3.3.4	AltTap Tool from Altom	26
3.3.5	TurnTable Tool from Altom	27
4	Implementation and Testing	29
4.1	Environment Setting/Integrating Framework	30
4.2	Diego Clash Game Code	33
4.3	TAF Code Sample or/and Chart Logic	33
4.4	GitHub Setup	34
4.5	Running Codes Samples Using Jenkins	34
4.6	TAF Report Generation	35
5	Results and Analysis	37
5.1	Analysing Code Reports	37
5.2	Gained Values Analysis	37
5.3	Future View Based on Analysis	39
6	Discussions and Conclusions	40

List of Abbreviations

AAT	Automated Acceptance Tests
Android	Google's open-source mobile operating system Term definitions:
BDDF	Behavior Driven Development Framework
CD	Continuous Delivery
CI	Continuous Integration
DDTF	Data Driven Testing Framework
F2P	Free 2 Play
IAP	the in-app purchase
IDE	Integrated Development Environment
iOS	iPhone Operating System (apple)
IoT	Internet Of Things
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
MT	Manual Testing
OS	Operating System
PC	Personal Computer
QA	Quality Assurance
SW	Software
TA	Test Automation
TAF	Test Automation Framework
UI	User Interface
UX	User Experience

Term Definitions:

Android	A modern smart mobile/tablets operating system developed by google. Simple free open-source product based on Linux operating system. This is a very common platform for games and widely used. Developing games is an Android huge application industry.
iOS	A modern smart mobile/pads operating system owned by Apple. Apple is a well know brand of smart mobiles and pads. As well they have monitors tv and yet more. iOS is an important platform for game industry with its various phone and pad types.
Game	A mobile game is an entertaining graphical UI that is usually used by players to interact with them either offline or online using an internet connection. Games can be either free to play or one-time payment or containing continuous purchases within the game itself. Games consist of many categories to mainly set out the player out of the real world and enjoy his mobile capabilities to run them and enjoy time.
Jenkins	It is an open-source cost free Continuous Integration (CI) tool for the purpose of building and testing of the produced Software (SW). For the moment, the Jenkin acts as a Change Delivery (CD) tool, this means that the whole SW is coupled for fast delivery purposes. Notice that Unity is deployed in Jenkins as a plug-in. Test automation framework integrates with Jenkins to be in access to the whole package while running the scripts.
GitHub	It is an open-source version control and mainly used to control the code from anywhere in the world within a particular project member. It integrates with Jenkins and the builds can be triggered to run after each code commit if needed.
Unity	Unity is mainly for games, and it has own Integrated Development Environment (IDE) and it is developed by a company called unity technologies. Unity supports a powerful Unity3D user friendly that supports people of different level of knowledge building games of different purposes to the players. Those games can be either smart mobile games of Android or iOS or

even web games or console type of games. At the same time there is a concept of Unity1D and Unity2D for flat type of games that does not require a powerful device to run. They also can run on powerful devices, and they have a nice market size. Unity supports the programming language C# - C sharp, as the Unity supports mainly those Object-Oriented scripted programming languages.

- Altom Is test automation framework offering a lot of tools to add values of testing games on Android or iOS devices and integrates with latest technologies in the market. They also offer integration with Unity and game object to make automating them test wise possible. Yet offers extra services with some physical tools like an Arm to simulate human movement or devices to simulate tapping of button in some complicated sequence operations. More at Altom.com (2)
- Customer In the gaming industry the gaming company can do a game and promote it for players and gain the benefits. Sometimes customers also inquire some of those gaming companies to build for them such games. Our games like Dirt Bike Unchained, Bike2 Unchained, in the market already was inquired by a customer. The games are made by using Unity.
- Test types
1. Functional testing for finding out the problems with the product functionality for a higher detection of faults, this grows with the game growing with more features all the time.
 2. Then the Smoke testing, regression testing, or Sanity checks are also important in the term of testing, and they are mainly used to be performed on the build before releasing it to the market by running a specific set of tests to make sure the build is not broken.
 3. Performance and Load testing of which is important when a lot of players are playing the game and interacting with its user interfaces.
 4. Usability testing of which to detect those hidden problematic faults during the player journey within the game.
 5. Accessibility testing to test the game compliance with those industry standards.
 6. Compatibility testing when running those games on different platforms and detecting what can break them.

7. Security testing and make sure that hackers can not break into the games to load free assets, so simply discover vulnerability.

DevOps DevOps stands for development and operations. It is an amalgamation of practices and paraphernalia intended in design to maximize and corporate ability of delivering a set of applications to serve the SW coding to faster expansion.

C# It is called C sharp. The C sharp is a Microsoft own programming language. And the C# is one of the most advanced programming languages in the market. It is highly used with professional SW developers. The language can be used for client side and for backend server side as well. It only depends on the supported imported libraries. It is a very popular computer programming language for the reason of an easy-to-read syntax and the variety of programming areas such as, web applications, mobile applications, developing games and providing a lot of business-related applications and yet much more. For an experienced programmer, learning C# can be very fast as it does not require more knowledge than the basics, but being fluent requires a while. C# is a great future programming language, and its existence is never disappearing according to the market future needs. NET is free those days to be used by all people globally, hence the C# is part of a thing called .NET, of which allows many programming languages with a collection of libraries, mobile related, games and Internet of Things and yet more. So simple the .NET is a framework that provides the optimal guiding on how to create a big variety of different application using many different programming languages such as C#. C# is a widely supported language with test automation frameworks.

Java/JDK Java is a programming language that can be used to create simple application to run on your machine or even being distributed around the network servers. Java is more of a backend development programming language, and it is of the best languages that handles the big data and android related development as well. It is great with mobile application, games, and other types of applications as well. Java can run on many OS platforms such as windows, UNIX machines, Linux, and Macintosh. Java syntax is simple to understand for a programmer and have some familiarity with object oriented. Java programming language is also supported widely by test application frameworks in the market.

Java Development Kit (JDK) is the java development environment. It is used to build the java applications and all other components by using Java as a programming language. JDK contains the Java Virtual Machine (JVM) mainly executing the java code and it also provides the environment required for that. To press on this a little, the JVM depends on the used platform. For example, windows JVM is different from the Mackintosh JVM etc. Though Java is an independent computer programming language from any platform. JDK also contains Java Runtime Environment (JRE), and it mainly combines Java code by using the JDK required libraries for running it on the JVM in which result the functional program done.

Python Is one of the greatest computer programming languages for its wide area usage in a very many fields such as programming of the web or mobile applications or gaming applications. Additionally, Python is used for Artificial Intelligence programming and machine learning. It can be yet further used to program operating systems. Python is not like C# or Java for an easy to learn programming language to learn. Python requires time to learn and master and this for sure easier for an experienced programmer, but not an easy task for a beginner and to the end all depends on what was planned to get out of it. At the same time, the syntax of python is clear and can be read and interpreted nicely. Python is widely supported by the test automation frameworks, and this makes its library one of the biggest libraries of the market bundle. Great language, but no simple.

1 Introduction

Software testing has existed for a long time, and its purpose was always to verify that the software is functional according to the customer quality perspectives. Usually, Quality Assurance (QA) designers and QA engineers do the testing decisions on how the testing could be performed. Testing can be either manual, meaning that no code or step scenarios are involved while testing, or automated by creating a set of set of test cases that can run automatically, then produce results and reports in a systematic way.

In theory, automated testing is based on a set of automated test cases that resides in a Test Automation Framework (TAF) to replace as much manual testing as possible. For example, the testing of the assets handling is important, the In-App Purchase (IAP) purchases, pressing of buttons, state transitions, etc. Those could be automated to some extent to reduce their massive testing repeatedly. It is more accurate to automate the calculated values or currencies using TA, because a human might need a calculator to do that.

The importance of this topic grows as the games within the company grow specially in size over the years. Maintenance work could also increase by time and catching possible bugs in advance is of the targets. Running a set of automated tests is very pleasant and provides nice reports of what has failed/passed. However, automated testing cannot be used in all game testing scenarios because games are visual, and they must be observed on how they look and react to the QA personnel eye.

The focus of the present study is more on mobile games of Android and iOS devices and only one game is used as a reference game, i.e. Diego Clash of which there is a game built by Unity for the scope of this thesis. Figure 1 illustrates the thesis flowchart.

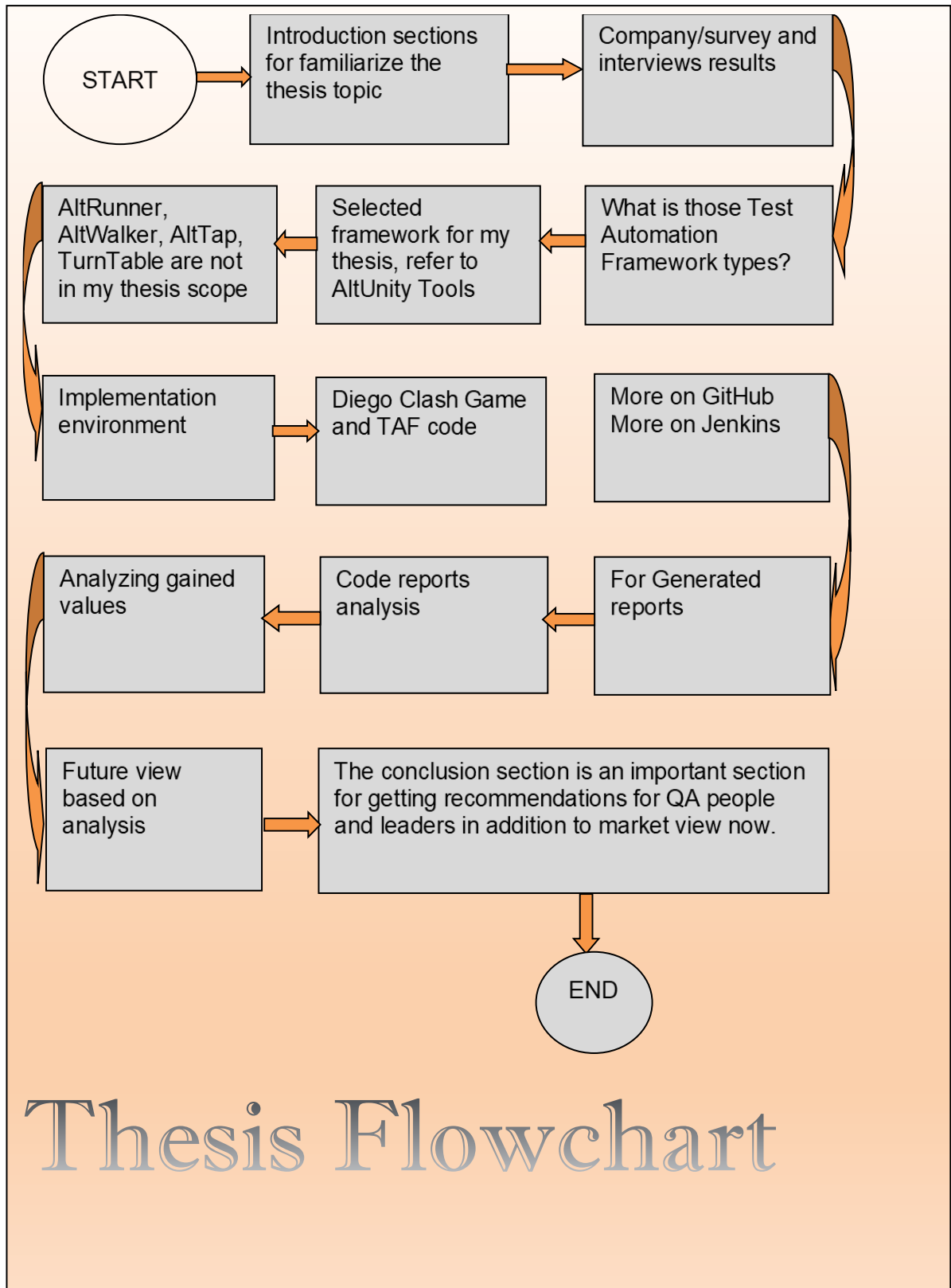


Figure 1 Thesis Flowchart

Gaming market is huge, and it contains games of strategy, sports, location based, arcade, card games, casino, lifestyle, puzzle, racing, RPG, shooter, and simulation. Puzzle and strategy games are the most famous these days, but each game type has its audience.

Games usually require a lot of visual testing, and this is a very hard part for a machine to perform. The human eye only can spot such errors. Machines could spot some errors, but this might be a very expensive operation and may not be worth the investment.

A summary based on the Newzoo Global Games Market Report 2021 (3) is given below:

Growth in game market is continuous and will be reaching 218.7 billion in 2024. The mobile device games are dominating, and it is reasonable since all have it in hand. With around 60% of the players are playing using mobile gaming. Console games and PC games players are kind of declining. It may worth mentioning the COVID-19 (corona virus disease 2019) may have affected a huge jump of gaming as people were forced to be more at home.

Sites such as Appannie (4) are talking a lot of the market of gaming and much of details on how it provides the analytics of the games. This helps mainly project managers and stake holders to study market and find the needs and fix the required needs to monetize in a much better way.

1.1 State of Art

Many gaming businesses lack a robust game automation solution. In the game sector it is quite impossible at the same time to get rid of manual testing because of the visual related features. Games must be observed by a human, and they must look pleasant to the eye and playing them requires quite a lot of a manual focus. At Kuuasema, many games are developed yearly and when it comes to the testing part, the manual testing is the only method used today. At the same time, there are a lot of areas that can be utilized and being automated to cover more testing boundaries and the reduction of manual testing.

Taking into consideration the existence of Test Automation Framework (TAF) will help solve testing of the game parts that can be scripted. TAF product usually offers some tools, various programming language support, configuration mechanisms to do integration, settings and running of the created tests end to end until the point of generating results and reports that are easy to interpret and read by different people at the company for further processing and concerns.

No direct guidance to combine those technologies together benefitting junior QA people were available. Such guidance could also benefit more experienced QA engineers as well.

1.2 Problem Statement and Objective

The aim of the study was to provide an effective way for automated testing guidance of mobile games. QA personnel require a way to start automating game testing with the co-existence of manual testing. The business requires time and cost savings in the form of good game automation testing.

The objective was to propose a way to guide QA personnel in the game industry business toward effective business cost and effort saving. The target was also to guide the QA users to execute and find a balance between manual and automated testing. Guidance is provided to do test automation in the game industry, helping in saving time, effort, maintenance, and money required by the testing to the company and game industry in general.

1.3 Thesis Structure

The thesis is divided into six chapters and the content of each chapter is as follows:

- The first Chapter contains the Introduction and introduces the topic lightly to the reader as to the meaning of the topic and the state of the art as a theory part, the problem statement and objective and the author contribution.
- The second chapter goes through the company results of the survey and the pre-prepared interview questions asked from several persons from the company and also the customer side. The results are listed and discussed and explained further.
- The third chapter starts to tell why TA/TAF is needed and what kind of benefits was gained out of it over the traditional manual testing used with games now. All the different types of automation common methods are discussed in addition to a study on which TAF was selected for the present study.
- The fourth chapter provides a solution mechanism to some current problems and how they have been automated (e.g., code pieces, either C#, Python or JavaScript). Additionally, Chapter 4 discusses the settings and configuration of the TAF within the company process, i.e., the integration of the TAF within the Jenkins CD tools

- The fifth chapter provides report examples as an outcome and discusses them further, this chapter also provides some calculations to the gained values of introducing TAF to the company based on several factors such as cost, efforts, maintenance, etc. Chapter 5 also discusses further theoretical considerations of the TAF usage at the company and what kind of benefit was obtained.
- The sixth chapter contains the discussion and conclusions based.

2 Kuuasema Company State Analysis

Kuuasema is a gaming company founded in 2004, providing enhanced game development with highly professional development teams. The company provides also support and maintenance for the Free to play games (F2P). Over the years, Kuuasema has made a great trust with F2P games with a proven record of advancing all time. By providing games for mobile, pc, console, Kuuasema has a great set of games made and brands that are well recognized.

Over the years, Kuuasema has provided over a hundred games with different platforms such as Android, iOS. Games such as Angry Birds, Donald Duck, NBA, Bike2 unchained and Dirt Bike Unchained games and way more. The company provides game design, client, and backend programming, UI/UX, Art, Live Ops capabilities to produce and deliver games to the market. With many trusted partners over the years and the co-operation has been always high and successful.

The QA at Kuuasema is performing well, but the testing part of game development is hundred percent manual. The need of a mixture of manual and automated testing would add a big value to the company way of doing the testing. This paper intends to improve on this part by introducing the automated testing and how it can be integrated to function along with manual testing.

2.1 Automation Relevancy Survey Results

Kuuasema personnel were asked to answer seven questions. Some of the questions were yes or no questions and some were checkboxes. The last one was a free text based question to give freedom to the people to tell what they felt about the topic. Altogether 26 people answered to the survey.

2.1.1 Kuuasema Survey Chart Results Regarding Game TAF

Eleven people seemed to just have heard of TAF, nine people seemed to know TAF, and six people seemed not to even have heard of it. Figure 2 illustrates the answers given to the first question.

Is the term Test Automation (TA) /Test Automation Framework (TAF) familiar to you ?

26 responses

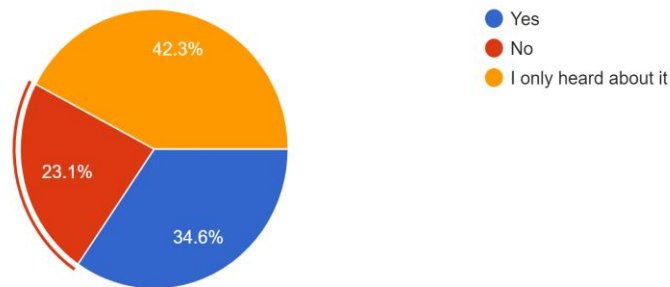


Figure 2 TAF familiarity at the company

The second question was whether Kuusasema developers thought the company could benefit from TAF. According to the answers given eight people said yes (blue), and ten people found it probable (orange), seven people were not able to answer (green) and one person did not see this possibility. See Figure 3 below.

Kuusasema developers are currently doing some TA code without using a particular TAF. Do you think Kuusasema can benefit out of TAF to maintain current testing code and yet more?

26 responses

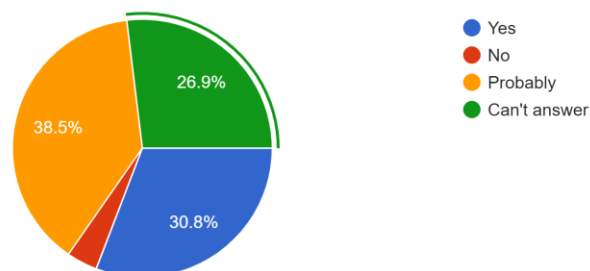


Figure 3 what could be utilized from current unit test automation at the company.

Next, the developers were asked as to what percentage they believed TAF usage could bring added value. The answers are shown in Figure 4 below.

TAF can be introduced to Kuuasema to improve test quality, speed and cost. To which percentage you believe TAF usage brings an added value?

26 responses

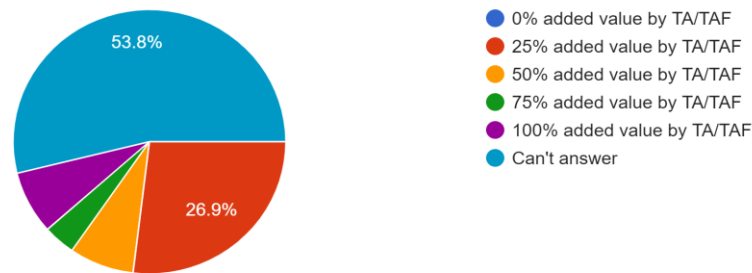


Figure 4 Company personal view on quality improvement and cost saving

Figure 4 is further analysed in Table 1 below.

Table 1 Result analysis of Added Value for TAF to the company

Added value	Results/persons	Analysis
0%	0	Natura, all people agree on the value
25%	7	-
50%	2	-
75%	1	-
100%	2	-
Cannot answer	14	Natural, because not all people are familiar of TAF, and this requires prove of this in the scope of this thesis

The following calculation formula shows the average added value result of TAF introduction to the company by those twelve people who stated values from 25%-100%.

Formula used: (people count * (1/total count who gave answers other than 0) * Add Value percentage).

All of those are calculated to each Added value percentage and summed to each other as the following.

$$\underline{(7 * (1/12) * 0.25) + (2 * (1/12) * 0.50) + (1 * (1/2) * 0.75) + (2 * (1/12) * 1.0) = 0.46}$$

The result of 46% is a very high for the added value by the people who have some idea of TAF. The final value can be calculated later by sending a final survey to the company personnel after some result sharing.

The fifth question was about what programming languages the developers were familiar with, see Figure 5.

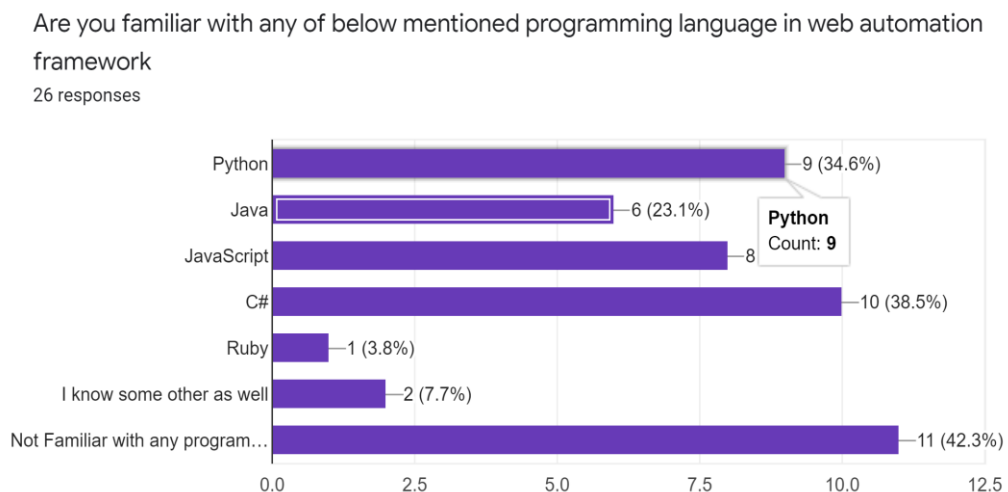


Figure 5 Computer programming languages familiarity at company

Eleven people were not familiar with programming, and this could be since many people's job at the company does not involve programming. This leaves 15 people who are familiar with one or several programming languages. As Figure 5 shows, 10 people are familiar with C#, 9 people with Python and 8 people with JavaScript, those three languages being the best known. In the TAF world, those are the exact three ideal languages widely used.

Question number five addressed the features desired from automated game testing. See Figure 6 below:

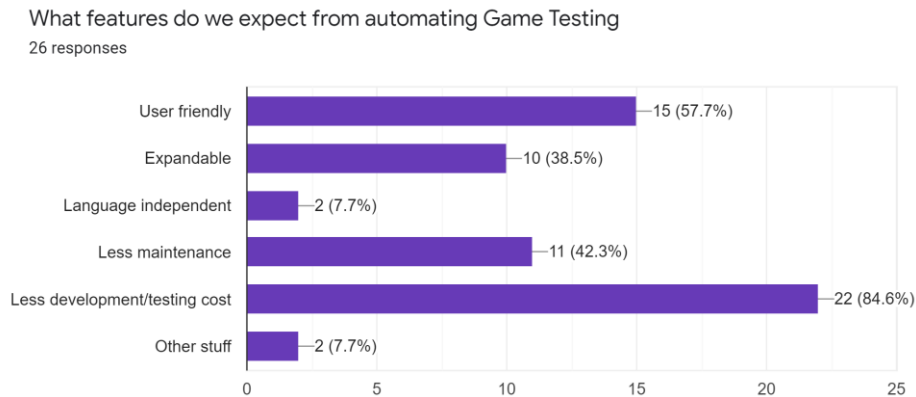


Figure 6 Features company personal believe the TAF would bring us.

The Figure states that 22 people out of 26 believed automated testing would generate less development and testing costs. This is a clear indicator that the company people are willing to see this in action. Fifteen people believed automated game testing would prove beneficial to user friendliness. Less maintenance and expandability are around the third place which indicates people believe in cost savings and reducing the trouble that comes out of the long process of maintenance. Language independency is correct, because not every language is fit to be used by TAF, mainly C#, Python and JavaScript languages are the most common ones.

Question six inquired the personnel's willingness to give TAF a chance if it were almost cost free. See Figure 7.

If I would tell you that the TAF usage by Kuuasema is closer to a cost of free. Shall we go for it?
26 responses

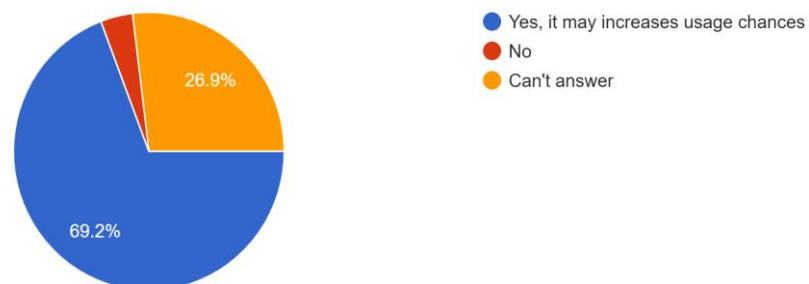


Figure 7 Could the cost be a factor to introduce the TAF to the company.

It is very clear from the above results that the saving of the cost is an additional factor toward adapting the TAF to the company. Some people of not much of knowledge stated that they cannot answer. One person is of the opinion no. Based on the answers given, the following interpretations as to the main factor to adapt such a framework could be phrased:

1. Saving of company QA and developer efforts/time spent on doing the manual tests.
2. Saving of company and customer effort on the long maintenance process.
3. Early discovery of failed tests that have passed earlier with nice reproducing steps toward developers.
4. Automating functional tests and smoke tests/sanity checks is of a great value over time with bigger projects.
5. Now come the cost factor. If the formula proved to be efficient from step 1 to step 4, then the cost saving is going to be successful.

The company could consider TAF adaptation into its process more closely based on the findings of the present study.

2.1.2 Kuuasema Free Text-based Survey Results for Game TAF

Here are the results of the 26 people having answered to the following question: Kindly give me your feedback on the games TA/TAF topic and what kind of benefits/problems this may bring us to your opinion?

- Good for online functionality and some specific cases. Could be difficult to implement.
- Benefits are obvious. Kuuasema test team is relatively small, and this can help them tremendously. Also, it would reduce somewhat both external testing and manual testing. Anything that helps making testing easier and with less manual labor is worth to try.
- Testing out new things is useful and a good practice. Make sure to have an idea what the value is for us, test things as rapidly as possible and then measure the value and decide if it should be implemented in our practices.
- Test automation can help the technical development and possibly find technical problems in the UI implementations if more elaborate systems are in use. But

many problems are out of the test automations reach like problems with UI and game design. But that of course is same with any systems developed for human users.

- Benefits on using TA/TAF, as it might help to improve the games by testing in a more dynamic way (onboardings, churn points, etc.).
- Could use some briefing on how this works, what it requires from me (artist) and how to use this system to get maximum efficiency.
- It might make things faster, but still maybe not reliable, lacking the human touch.
- Pros: API design is more robust, backend-client interaction can be implemented in sandbox before integrating it to the client. Cons: It will create some more overhead. Over reliance might introduce UX issues
- In my experience, test driven development (or behavior driven development) normalizes development speed. In test driven development, complex features require less work – since they are being developed from the user's perspective – while simpler features require more work, as a trivial implementation still needs high quality tests to be designed, written, and maintained. The overall result is a more predictable development velocity, and of course higher software quality which has many benefits which I'm sure people are familiar with. A general mentality among game programmers seems to be that programmers simply want to see the thing work. Testing fits very poorly into this process because its benefits are not immediately obvious when compared to simply writing the feature and then exploratory testing the feature. This makes it hard to employ these techniques in practice. But as our teams develop their process, and projects grow more complex for various reasons, and the focus shifts from the artisan "game making" in pre-production stages to data-driven live ops, they are slowly getting familiar with the concept of software quality. As expected, being in favor of any new technique that improves software quality. However now, feeling that the impact cannot be easily analyzed because not all developers are familiar with any kind of automated testing, and the inspiration for such a practice should rise from within. With that in mind, some teams would be ready for it, and others... well, the time will come.
- Believed it will have a great added value to our testing mechanisms.
- Optimally the testing toolset would enable QA testers to setup play mode testing themselves - some current tools look like they may push more workload onto programmers.

- Pros: reduced time of maintenance and pre-submission testing /QA (just adding new auto-tests, instead of testing older functionality manually), easier hand-over in case of the QA engineer change on the team, reduced number of human mistakes, improved documentation on each test assertion. Cons: increased time to familiarize the automation QA/development on the team with the new piece of technology (test framework, etc.), additional time is needed to write the automated tests.
- Sounds like a good option to investigate more.
- Creating and designing test cases and maintaining them takes time and effort. Tests written usually pass but the game still has a lot of bugs. Maybe fewer than without.
- Automating some UI flow testing could be beneficial.
- For the start the development speed would go down, but eventually it will speed up. In my humble opinion, minor productions do not benefit anything from TA/TAF, in other words, this is meant for the bigger games/projects.
- TAF would need to support Unity games.
- It should give us savings that improve the economy.
- Hopefully, automated testing will reduce bugs and regressions at release.
- Interesting topic and would like to know more out of it.

The answers are very positive from the company people. People would like to see TAF taken into action, even if they are not familiar with it. They believe and trust it to bring more added value to the company such as effort saving, maintenance cost saving, faster service providing, and cost reductions for long run big game projects. Also, the thoughts regarding this being started in a slower motion are valid because the start requires the personnel to familiarize themselves with the tool and how it integrates to the system. Figure 8 shows the courses offered by BBST.

The learning curve might be a big measure here to consider, because it is a new tool to be deployed and to be used. A set of well selected courses from the site (5) is well supported by the selected TAF by this thesis. For example, **Error! Reference source not found.** below lists a few important courses offered by BBTS.



Figure 8 Courses offered by the bbst, courses with certification and hands on

Familiarity with programming languages such as C#, Python and Java is also important, learning those could be a challenge as well to those who do not know them.

TAF is not just about learning the framework, it is also about a way more to learn around it. Once TAF and testing confidence, computer languages knowledge, Unity, Jenkins, GitHub are familiar, the work starts to fly faster.

Finally, to say that people are on the positive side, and they also spoke of their fear of cons. The cons as stated above all were about the start of TAF introduction to the company. It is natural for any new product at any company to fall into learning and harder start. But once learned, this is the moment benefit are started to be gained out of it.

2.2 Interviews Summary by Functions

The interviews were done to the following functions at Kuuasema, the following list includes a light description of the various functions:

- Game design (designs the features offered to the player based on customer requests)
- Game UI/UX design (defines the game layout and how it looks like to attract the player of the game)
- Game Client Code Engineer (game frontend programmer to apply the design and UI/UX related stuff to create the game by doing a code)
- Game Backend Engineer (backend that hides the request logic by the client side for the player, such as storage and resources and requests handling, server type of technology)
- Artist (the one person who does the graphics of the game)
- QA (the person who does the testing of the game)
- Management (any person from top level who deals with customers and see the game as a player would do, but from Kuuasema perspective)

- Customer (the people who order the game from Kuuasema and make sure that it monetizes with our assistance to gain either money or reputation in the market)
- Producer (in game industry a producer is the project manager of the game, making sure that the feature is implemented and communicates with customers, also make sure the correct practices are used and ensure agile and team communications are in place)

Ten questions were created for the sake of interviewing Kuuasema personnel and the customers requesting the game. The interview was based on the game Dirt Bike Unchained which this thesis uses a reference for the survey. But for the proof of concept the game Diego Clash was used.

Table 2 combines all the functions answers. If the answer is related to a particular area, it will be highlighted accordingly.

Table 2 Survey interview questions to Kuuasema and customer

Question	Personal Interview answer combination
1. What would you like to say about game automated testing? What do you understand of game automated testing?	Tests are to be seen runnable on devices (android and iOS). The tests could save game design early decisions to increase customer engagement and provide more reliable game. Usage of game functions (libraries could be integrated). Automation is of common understanding by all functions of the company and customer to automate and save some resources and possible costs.
2. Does Kuuasema see a place for Game Test Automation framework within its current system setup?	All personal were in favor of game test automation to be introduced to Kuuasema. If it could be fit or not was an open issue to be defined by this Thesis scope.
3. Do you believe we should automate tests to a specific	Each of the interviewed people were interested about automating as much

<p>area of the game? And why?</p>	<p>as possible out of the game in question. But specific areas where more about buttons tapping and core game play.</p>
<p>4. What areas to your opinion requires to be test automated by TAF?</p>	<p>Each function area mainly cares of seeing their stuff working. Hence Client coders care the game functions as it should form the vision of the player. Backend cares that the returned data is correctly validated. Producer and designer and management cares more about saving design mistakes and customer satisfaction of monetizing the game and players happiness about it. UI and Art is the hardest to be automated because it requires a human interaction to see and report of Art and UI quality.</p>
<p>5. What do you think Kuuasema could benefit the most out of test automation?</p>	<p>Most of the people tend to say, we could</p>
<p>6. Do you want to see test result reports? If yes, what area reports interest you the most? If no, why?</p>	<p>The answer yes was by all the interviewed people. Each interviewed function stated an area which mainly related to his function. Art and UI/UX people cared of the menu states and button sequences. Client care that game functions as design stated. Backend of the care they returned values based on request are correctly used and handled by client. Design save time with customer before a new proposed value to monetize the game are tested (feeding those values to an</p>

	<p>automated set of tests could save a lot of trouble later). A producer care about all what was listed above as he/she see the overall picture of the project.</p>
<p>7. When analyzing those test automation results (success/fail), in which form you would like to see those results? Any preferences of presenting of the result format?</p>	<p>All people agree of seeing the results in an easy way to read the reports. Most people like to see Charts and graphs. Many also like to see a combination of graphs and text explanation. Developers care about seeing the actual fault reported by some of the market bug reporting tools. But still a quick list of failed tests is nice to see, especially those cases that have succeeded in an earlier build of the game.</p>
<p>8. Do you believe we could save money/effort(time)/maintenance because of TAF introduction?</p>	<p>Everyone answered yes. But sometime this yes was in a concern. The reason is do we really know that the TAF tool cost compared to the used resources from us worth it or not.</p>
<p>9. If the TAF studied in this scope to be introduced is of low cost. Do you recommend its usage at Kuuasema? Why?</p>	<p>This remains to be seen once this thesis do some calculation in chapter Results and Analysis and Discussions and Conclusions. Any save of cost is good for the company. Everyone recommends TAF usage. Some at the level of being a bit careful. Customer stated that if this proves time and cost save, they can be willing to pay for the cost of TAF based on contract negotiations.</p>
<p>10. Who must be responsible for the framework test code/configuration? And why?</p>	<p>Clearly everyone agrees and told that QA is to be responsible. Additionally, developers are supporting QA with</p>

	functions or libraries that could be utilized by QA from the Unit tests to TAF. Designers might have a say as helpers as well.
--	--

A set of an extremely professional people were interviewed from the company level as stated in the start of this section.

Everyone seems to see a chance and a great possibility that this is going to work but requires a lot of effort to be used. As QA in the company will be clearly the responsible for TAF and its integration to the process, the most effort will be consumed from them. Developers, designers, and others will support by providing guidelines and advice related to coding or execution chart definition. The company was being very helpful and offered the possibility the reason being it saves QA's and others' work time and load as well.

3 Automated Testing

Manual Testing (MT) is mainly testing of the games manually step by step without any automated steps. There are no scripting or tools used when executing manually. Manual testing is slow and consumes time and once something fails, it might require a lot of effort to find where it has happened.

This is reflected to the fact of manually trying to execute the suspected scenarios. In the game industry there is no escape from the manual testing when it comes to the observation part. The reason is that the games are observed on how they look from the art point of view and how the User Interface (UI) is seen by the players. Manual testing is great when it comes to observation, and it could be more relaxing to the QA people not to think of creating automated testing and being familiar with its tools, deployment or integration to the system.

A mixture of MT and Test Automation TA is the best when it comes to the gaming industry. The major reason is that not every possible testing scenario is achievable by either of the cases.

3.1 Why Automation

TA is the way of practice to run a set of tests automatically to save time and effort and produce a convenient set of results. This is mainly for improving the quality of the Software (SW). QA is the major sector using this mechanics to detect faults fast enough and report them adequately. By saving time, effort, and money, which is the main purpose of the TA, it means the testing criteria must be well defined to be automated.

On the other hand, the design of the TA set must be able to run repeatedly. There must be a clear design to not fail this part. TA must consider a lot of issues such as setting up of an environment, then execution of the set of test cases in addition to clearing up the data and resetting of the environment used.

There are many types of tests that can be automated. It is very important to know that what can be automated must be clear enough. Automating feelings or given feedback, etc., is not possible. This leads to the question of what can be automated.

When defining the tests to be performed, it must be clear what is to be automated. For example, the unit testing can be automated, because they focus on one functionality of the system to be tested. Unit Testing focus on testing of the code without much of external dependencies. But when talking about the Integration Tests, dependencies must be considered, and this is more complicated. The dependencies on the network availability of some service might simply break the tests. Meanwhile Automated Acceptance Tests (AAT) is also important when it comes to acceptance of a not yet released feature. AAT require co-operation from QA and developers, and this is to make future regression testing wiser and does the required. After that the regression testing can be planned as it verifies that new code developed does not break the already existing functionality. Many other types of testing can be automated such as performance testing in a way of loading the system and putting pressure on it. At the end Smoke Testing is one efficient set of test cases to verify that product is good to go, this is by ensuring service is up and running with all the dependencies required.

When the game to be tested, the environment is set ready in place (this may require special kind of settings, depending on the application). After that, the test suit defined will be executed and triggered. After that, the test results and reports are generated and ready to be evaluated by the QA, developers or who is interested. Chapter 4 ‘

Implementation and Testing' tell more of this in more depth

The conclusion is that the TA is great and can be utilized for different sets of testing types, but this does not mean manual testing is obsolete. Everything is not possible to automate, especially in game sector, hence the manual testing plays a big part of it.

Automation frameworks existed to host code that will integrate with some service to produce test results and reports. This will also mean that it may offer the possibility to design and create those set of tests. At some point it will be realized that the framework used is a set of practices and tools together to help testing to proceed. Quality Assurance (QA) Engineers are the most beneficial part of TAF.

The speed of the game loading and debugging is important. As far as going further this will matter a lot to spot faults easier. Many computer-based languages can be used to create and run automated tests. For example, C# or Java or Python and many more. It depends at the end of the selected test framework and what kind of computer languages it supports for scripting.

It would be great if TAF could integrate with some other frameworks in the market. Multi-communication between different TAF is a great add and could be many reasons such as some tests require another scripting language capabilities to unionize some tests that cannot be easily achieved with the current one in hand. This is only mentioned when talking about the value of this feature only.

One of the most important aspects when choosing a TAF is to consider that the reports are generated in a very nice reading format. It is very important for others to understand the reports without struggling. Reports that contain charts, figures, tables, screenshots are of more value and could utilize more game user acquisition for the game.

3.2 Types of Test Automation Frameworks

TAF provides all the required benefits to the user when executing a set of tests efficiently and reporting the results. Before selecting the correct framework, the different types are briefly listed and explained.

1. Module Based Testing Framework (MBTF)

A module-based testing approach, which means that the modules can be of any 1 – N numbers and a script will be representing them. The good thing of this is easy to maintain and highly scalable. This would be a good type when it comes to maintenance part as well. On the other hand, this type might be data specific and a change in data might require dramatic changes.

2. Library Architecture Testing Framework

It could be said that the library architecture testing framework is based on the MBTF framework, but this make sure common things are set together like a library. So common steps are bundled together and whenever required it is called and used. This is cost-efficient framework type, and the re-usability is a major benefit. This is also hard to maintain, and the libraries may make things more complicated.

3. Data Driven Testing Framework (DDTF)

The main idea of data driven testing framework is to separate the logic of the test suit scripts created from the data to be used as an input. External data base is used, and files can be of any format that could be read by the scripts. A particular mechanism can be used to fetch the data and make a use of it. This type reduces the number of scripts used and most important that data change will not affect the scripts. By this a more flexible and maintainable script will be available. As a side effect, the data reading mechanics and programming skills are highly required.

4. Keyword Driven Testing Framework

Keyword driven testing framework is driven from the DDTF type. This means that the storage of the data to be tested and the keywords are stored in the same data base. Based on the nature of the keyword the code will be executed. This type provides similar cons as the DDTF, and additionally the scripting familiarity is not required, and this can be used across a lot of test scripts.

5. Hybrid Testing Framework

Hybrid testing framework means that many of different types of frameworks can be used and utilized together for bringing the most out benefit. This module will bring all the cons and pros for all types. This requires a professional QA personal do it. It requires knowledge of many deferent types and how to make the best out of them to reduce money and increase efficiency and stability.

6. Behavior Driven Development Framework (BDDF)

Behavior driven development framework is more for members of QA personnel that does not need to be a super programmer. This brings value to all parties from developers, QA, Business personal, Management, etc. This depends on the used framework that will offer different tools like cucumber, JBehave and so on.

DDTF would seem the best fit for the TA framework selection.

DDTF was selected in this study. The reason is that most of the games these days utilize data that comes from the backend (called server sometimes). The backend serves the client and provides it with all details it needs based on request. Backend also provides some details as a result feed to the client, so the client acts based on the received data.

1. Backend is not critical for performance, because it is basically called when client requires some data.
2. Backend hiding of data toward the client and communicates toward cloud to store this data is a very nice thing.
3. Resources are handled by backend and given to Client based on request.

DDTF utilizes libraries (calls to those libraries will do the communication to the client) provided by the client coders as they have coded the game. Those will be used and utilized by QA personnel. QA creation of tests will require to deal with the backend as well to gain some data that can be passed to the used libraries to obtain results to confirm a pass/fail test case mechanism.

3.3 Test Automation Frameworks Selection

There are quite many available TAFs in the market. Many are either open-source (free of charge) or costly ones. TAF can be used at the personnel level or at the company's level.

The TAF selected here was Altom.com, it supports functional testing, performance and load testing, usability testing, compatibility testing, security testing, regression testing and yet has a great number of tools supported to serve game development. They are listed in the following sections in relevancy order. Altom is a DDTF framework.

AltUnity Tools are the most interesting tools for the scope of this thesis as they cover the mobile game test automation. AltWalker, AltRunner and AltTap and TurnTable tools are worth listing as well to show the variety of test scenario possibilities.

3.3.1 AltUnity Tools from Altom

Most important is the package that integrates to the Unity. They are called AltUnity Tools (6).

There are 3 offerings of AltUnity tools:

1. AltUnity Tester package. This is most important for interacting with the unity objects and do the writing of the code of TAF. It is worth listing that the access to Unity is required. The AltUnity Tester package support is free on discord channels, refer to Altom pages for adding yourself to the channel.
2. AltUnity Inspector package. This package gives some independence of the Unity with the access to the game as a black box. But this require a line of code to be added to build mechanism so it can be seen when testing it. Additionally, this also offers interactions with the game during the testing by pressing tabs or doing a move by a mouse, etc.
3. AltUnity Pro will be offering the AltUnity Inspector package in addition to more features. There will be a rich reporting tools by Altom.com and not only those that can be generated by Python or whatever. Greater support to the most popular Cloud service providers with an extended support to the customers.

AltUnity Inspector and AltUnity Pro cost monthly or yearly fees. Those fees are per license. The additional AltUnity Tools that are free will be an add on the Unity Personal that will be used for the purpose of the thesis prove of concept. Figure 9 below lists the AltUnity packages.

<p>AltUnity Tester free</p> <p>Open source tool on Unity Asset Store</p> <p>Download</p> <p>Enable end-to-end test automation *</p> <ul style="list-style-type: none"> ✓ Interact with Unity objects ✓ Simulate device inputs ✓ Modify methods and properties ✓ Run tests on PC, Android or iOS ✓ Support for tests written in C#, Python, Java ✓ Integrate with BitBar and AWS ✓ CI ready <p>Community support through Discord and Google Group</p> <p><small>* Access to the source code and the Unity Editor is needed</small></p>	<p>AltUnity Inspector</p> <p>€18 <small>(+VAT) per seat per month</small></p> <p>Desktop application for Mac OS and Windows</p> <p>Start free trial <small>no credit card required</small></p> <p>Inspect games instrumented with AltUnity Tester</p> <ul style="list-style-type: none"> ✓ Fully functional without the Unity Editor ✓ Access the game objects and their properties without access to the source code ✓ Display the scenes of the game live ✓ Use mouse, keyboard, touchscreen and joystick actions to interact with the game from Inspector ✓ Load different scenes ✓ Control the speed of the game <p>Priority support, response within 2 working days</p>	<p>AltUnity Pro</p> <p>COMING SOON</p> <p><small>We are adding new features to our existing solution</small></p> <p>Get notified</p> <p>AltUnity Tester and Inspector included, plus:</p> <ul style="list-style-type: none"> ✓ Run tests on Console and WebGL games ✓ Generate test reports ✓ Integrate with any popular cloud service ✓ Generate tests automatically based on recorded actions ✓ Extended support ✓ Consultancy
--	--	--

Figure 9 AltUnity Package offering contents.

The AltUnity Tester package is independent of the Inspector and Pro package. So, it can be downloaded for free and integrated with company Unity, by the Unity Personal pack. Once those are set correctly, the packages can then be built with the AltUnity Tester and the TAF scripts can start to run.

3.3.2 AltRunner Tool from Altom

AltRunner is one of Altom tools that uses Appium Test Framework, the following statement explains it in more depth:

“If you do automated tests through Appium on mobile devices, you know how annoying it is to do the setup every time, for every device. AltRunner does this setup automatically whenever you plug in a device. You can also run tests in parallel on multiple devices. In its enhanced reporting module, you can investigate failed tests.” (7).

3.3.3 AltWalker Tool from Altom

AltWalker is a very useful tool from Altom to define the flow of the tests:

“An open-source model-based testing framework for automating test execution. You design your tests as a directed graph and AltWalker creates the test flow and executes them.”, (8). Figure 10 shows one AltWalker chart scenario example.

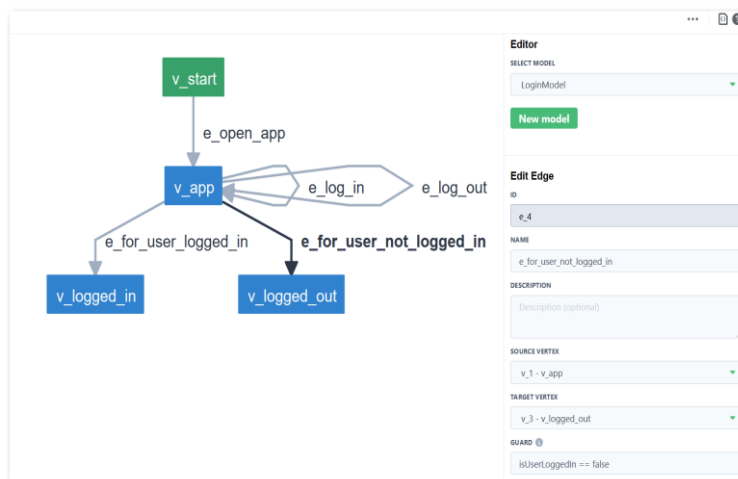


Figure 10 AltWalker chart scenario example

These days, most of the games use Unity engines that are free of charge at the personal level to provide possibilities for game creation. Unity engines provide two kinds of possible offerings:

1. 2D with lots of tutorials and assets to make doing of 2D games fun.
2. 3D with yet a lot of tutorials and assets and making great 3D games as the major intention of the Unity is the 3D games.

Unity provides C# as its incipient scripting computer language, yet Unity offers a lot of plugins to form an editor. This means some functionalities are possible to be drop-pable.

As a result, there are no limitation as to how much testing could use out of Unity. The main reason is that Unity offers many components/objects and those can interact with each other by actions defined by the coder.

One benefit has been the possibility of a set of supporting libraries from the game developers. So, once those components/methods/properties are being available, QA

could proceed to test creation. The test creation requires an interaction with the loaded game on the device, so that the input of touches or Keypress can be detected.

Unity has the Pro and Plus variation, but Altom.com does not require them for the usage of their tools. If they are already licensed, it is not a problem, but not a requirement.

3.3.4 AltTap Tool from Altom

AltTap is a mechanical tool for automating touchscreen related game stuff:

“AltTap’s special power is that of performing automated tests on any type of touchscreen device, even where it’s not possible to interact programmatically with the application. (9)

AltTap simulates a human tester by using a more complex stylus to perform click or swipe actions on a touchscreen and push actions on physical buttons. The process also uses an image-recognition algorithm for detecting elements or performing asserts.”

It is a power supplied device that capture images and record them with sounds and store them for reports later. Altom offers this service to the companies requiring tapping tests. This is not in the scope of this proof of concept but listing it here to show that such physical device exists and Kuuasema can utilize it for some contracted test requests. Figure 11 shows an AltTap robot.

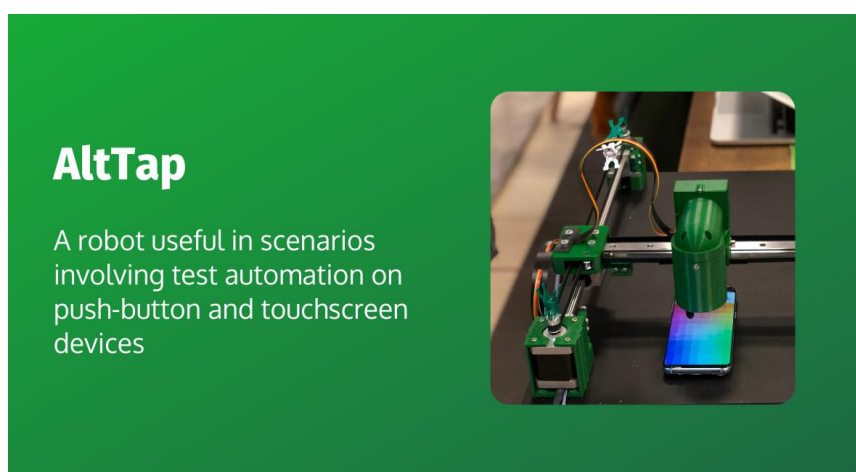


Figure 11 AltTap that perform touch mechanism when testing game UIs

3.3.5 TurnTable Tool from Altom

TurnTable is a mechanical tool to simulate hand move:

“A robotic arm that allows us to simulate a user’s hand and head movements in order to automatically perform advanced and reliable tests on apps that require various movements, such as games, VR apps or photo apps.”, (10)

The robotic arm and how it works is very interesting on utilizing the movements a player of the game can do. Once the tests start, the sensors on the devices used will start to run as a human would have been doing. Figure 12 shows a physical device meant to simulate human arm for game testing.

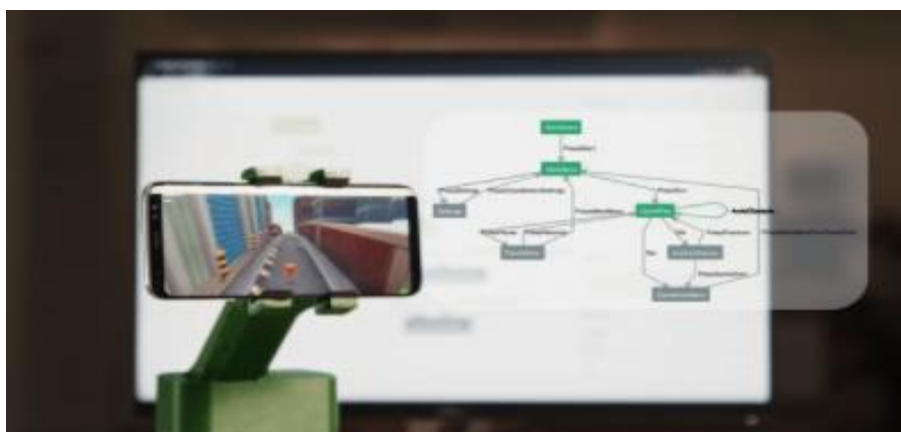


Figure 12 Physical device to simulate human arm for testing games.

Figure 12 shows a service provided by Altom.com to automate test scenarios based on customer contracted request. This is listed in this document as it could be used by the company, by not within the scope of the thesis.

4 Implementation and Testing

This chapter is about the process of downloading TAF and integrating it with the company setup, what it requires from the company and how much of effort it can be and how complicated it is. Next, the chapter contains the game code to be tested and some explanation around how it works. The created code of Diego Clash is only a simple guiding game code example. Furthermore, the chapter contains three test code samples. The samples run to FAIL one test case and PASS two test cases. This could be done with three languages, C#, Python and Java, but C# was selected here.

This chapter also focuses on the storing of the generated code. Also, how it can utilize the GitHub free source version control system. In this example, it is simple to handle, but the bigger and more complicated the system is, the more this tool usage makes sense. Next, the chapter focuses briefly on Jenkins and the required plugins to activate the CI/CD of the Diego Clash game. The focus is also limited as that can be learned from the net. Finally, the chapter focuses on the generation of the test result reports. The reports are not Altom.com specific format because of Altom “AltUnity Pro” package is not yet ready to offer this feature. The report is specific to the Programming language package offerings. This is clarified in Section 4.6 whether all languages can offer an easy-to-read test result report.

This section utilizes the TAF code using three different languages: C#, Python and Java.

The three languages are supported by the TAF Altom.com. The code written was interacting with the Unity function calls. Those functions are built mainly with C# as unity is supporting this language.

The three languages interact with C# as Table 3 shows.

Table 3. Language mapping ...

Table 3 Languages that can be used to do TAF code

TAF code language	Unity code language	Result
3 C# test code examples, one to fail and 2 to succeed (selected for this thesis)	C#	FAILED, PASSED, PASSED
3 Python code examples, one to fail and 2 to succeed	C#	FAILED, PASSED, PASSED
3 Java code examples, one to fail and 2 to succeed	C#	FAILED, PASSED, PASSED

Table 3 is a simple table to tell that the coding of TAF can be done with multiple languages. For this thesis scope the two passed and one failed cases are made using C# code.

4.1 Environment Setting/Integrating Framework

The examples and environment setting are specific to Windows in this scope. For another Operating System such as Apple Macintosh or other, then it is just about different downloads and could be found and used in a similar fashion to the ones used in this thesis scope. Otherwise, the steps are quite a lot the same.

This section explains the steps to be taken to utilize the TAF with the Kuusasema environment. Based on the thesis writer experience in the gaming industry this is very close to the case for all companies, because the companies try to save cost and utilize solutions that work well as open-source products.

From the moment the SW is to be created with all the actions around that until it reaches the market in a nicely automated fashion, it is all about Continuous integration/Continuous Delivery process:

- CI is more like automating changes that is made to the code for one single kind of SW project, and the GitHub version control can be also supplemented with other tools.

- CD is a wider concept for developers. It allows to automate the testing in a wider concept than just unit testing. This means that the deliverables can be verified with more than one dimension before they get deployed to the customers. The test types meant here could be testing reliability, security, performance, loading times, regression, etc. CD offers those automated channels to reduce manual errors or even terminate them yet provide a kind of standard feedback either positive or negative to the coders, also to give the possibility for faster products to the market.

The market is full of products that support CI/CD. After full research “Jenkins” was selected for several reasons of which some are listed below:

- Free open-source product. But it is important to know the hosting server for Jenkins is not free. It is also worth mentioning that it is my responsibility to maintain and update it.
- It has over 1000 plugins available and supported. But not all the are essential. The aim selected defines what plugins are required. Those plugins can be found from manage Jenkins and Plugins.
- For example, GitHub is a plugin in Jenkins, Unity3D is a plugin in Jenkins, there are C#, Python, Java plugins, Cloud, etc. The ones listed are those required by this thesis domain. But Jenkins cover all what world may need of plugins.
- Jenkins is integrating very nicely with almost the whole DevOps tools.

After a little explanation of CI/CD and Jenkins selection, the following can be stated: CD packages what CI has built and tested. CD offers the build and configuring it, packaging its SW, and doing the deployment regulations. The deployment part will be the final step towards the releasing of the product to the market.

The CD tool usually considers the cost and powerful automation of the product released. And by this the old traditional ways of SW management do not make sense as this Jenkins CI/CD tool is about improving the speed of SW delivery to the market and saving time to corporates and providing high rapidity.

The following products are required to be installed for the scope of the study (the required plugins installation go through with the orchestration of Diego Clash game delivery

1. Unity Personal installation is to be done and performed. It is also recommended to use a tutorial for learning the basics of Unity. Unity is the best way to create games as it offers Unity with multiple variations as stated in section remember that Unity Personal is free to use, but for bigger corporations pricing of the Unity will apply according to their terms) – Unity Personal installation website (11). Windows installation is in the scope of the study.

In this document domain, Unity is not fully explained, because it is a long topic of its own. Doing a study by using a tutorial go through and select some game examples to take the programmer with a complete tour on how to build and execute the game. To learn Unity a reader must follow Unity learning guide website (12). This is a great tour for starting the use of Unity. Also note that the link of learning the Unity also leads doing the installation as one of the learning steps. The installation can also be directly as listed above.

2. GitHub Desktop is a good tool for replicating code (a versioning system tool, explained in Terms section at the start of this Thesis) and the browsing of the code in an easy way. For GitHub desktop installation, please follow-on guide of installation website behind (13). Windows installation is in the scope of this thesis.
3. Once the game “Diego Clash” code is written (the code is in an appendix and is referred to in Section 4.2). This game is then utilized to be tested by the TAF code (More of this in Section 4.3). The code can be on a local machine or shared to someone for replication as step two above stated. Diego Clash can be added to GitHub (More of GitHub in Section 4.4). If a particular Unity version to run the game is required a notification will pop-up guiding of what to be done (in case a particular version is required to be downloaded to support running of own game). Unity Hub can have very many different installs of unity versions and they can be added/removed based on your needs. Please refer to Appendix 1 for the image reference.
4. The image shown in Appendix 1 shows the game in question “Diego Clash”. The required Unity version for that was not available after the installation of Unity Personal. That has required another Unity version in which the tool instructed downloading it and it was indicated by warning triangle that was on top of “TAF-Diego-Clash” project. Clicking on that, lead to install the required version “2019.4.28f1”. This may take a long time as unity is a huge package. After that the installs can be observed as in Appendix 2.
5. Install of the Altom.com TAF by following the website behind (14). After opening this link there is a button “Open in Unity”, just click that and the rest is done automatically

(the installation of the package will integrate with Unity). Other packages are not explained in this thesis domain. Appendix 3 shows how does this view looks like.

4.2 Diego Clash Game Code

For this thesis purpose a new game called “Diego Clash” was created. The purpose of the game was to experiment and act as prove of concept using the TAF code on it to fail and pass some of the test cases. Diego Clash and its code belongs to the company in question and was done for the reason of simplicity for the reader. Diego Clash code is listed below with a light explanation only as the purpose and scope of this thesis is not to teach code writing.

1. C# code for the Game Data. In the Diego Clash game that data handled is gold. Please refer to Appendix 4 for the code sample of game data.
2. C# code for the Game Play Manager. This is the major file of the whole game of which the buttons and the top bar manager and prize claiming are all handled. Those files are coming next in this thesis context. A reader must be a little familiar with C# code to read and interpret this code. Please refer to Appendix 5 for Game Play Manager code (part one and part two).
3. C# code for the Main Menu Manager for starting of the game and quitting of the game, Appendix 6 contains the code for starting and quitting of the game.
4. C# code for handling the buttons numbering and their names, images and what is to be activated once clicked. Please refer to Appendix 7 for buttons handling code.
5. C# code for the Top Bar Manager, here the gold amount text and the gold amount updates are happening. This code also has the mechanism to put the player back to main menu based on clicking. Please refer to Appendix 8 for Top Bar Manager code.

4.3 TAF Code Sample or/and Chart Logic

This section shows to the reader the TAF code used to fail/pass some of the Diego Clash game uses C# programming language. All TAF code samples will initiate the socket connection opening and closing after the test is being executed.

1. C# code for failure in Claiming of some prizes. TAF code file is: testFail.cs will load level 1 scene and then fails to claim the prize. Please refer to Appendix 9 for the test fail code sample.
2. C# code to test Diego Clash game starts and can move from main menu level. TAF code file is TestMainMenuToLevel1.cs will load the Main Menu sense and performs tapping of which starts the game successfully to pass the test. Please refer to Appendix 10 for the code related to this pass test.
3. C# code to test playing of Diego Clash game at level 2. Test code file named: Test-PlayLevel2.cs: Code will load sense 2, then tap all the buttons, then claim the prizes and verify a successful tapping and prize collection. Please refer to Appendix 11 for the code related to this passed test.

4.4 GitHub Setup

Within this project it was decided to use GitHub as a code versioning system. The following figures show “Diego Clash” code game file structure and TAF code that is testing it. The meta file is created automatically by Jenkin CI/CD tool. This is required by admins to make sure all projects contain this certain metadata.

1. Diego Clash GitHub structure as the following
Please refer to Appendix 12 for the code structure of Diego Clash code.
2. TAF code for doing some testing to Diego Clash code
Please refer to Appendix 13 for the code structure of Diego Clash TAF code.

Running Codes Samples Using Jenkins This simple code does not require mainly Jenkins as it was possible to run and execute earlier using Unity. But trying to list a few of the steps that could be of good help as this is not in the main scope of this thesis.

This following are what to be considered:

1. The installation of Jenkins described lightly

The installation of Jenkins for windows will happen by following the website (15), so follow the instructions when installing the Jenkins carefully, also make sure before that to have the recommended JDK and JRE installed. After the installation is being completed, Jenkins will run locally on own machine <http://localhost:8080/>. Please refer to the Appendix 14 on how Jenkins looks like after the installation. Exploring more those different Jenkins UI elements and learn more by following the website (16).

2. Writing the required Windows batch file (.bat) that runs the tests (a bash script can be used as well) and Running AltUnity from command line is also an important step
Example of the file:

Batch file example:

```
<UnityPath>/Unity -projectPath $PROJECT_DIR -executeMethod
AltUnityTestRunner.RunTestFromCommandLine -tests
MyFirstTest.TestStartGame -logFile logFile.log -batchmode -quit
```

Note the variables are to be defined or otherwise a usage of hard coded paths is a must (not recommended).

3. Setting up Jenkin to run the batch file
Please follow building of Jenkins job instructions website (17) for more details on how to build the Jenkins job.

4.5 TAF Report Generation

From unity a simple report is received as Figure 13 illustrates:

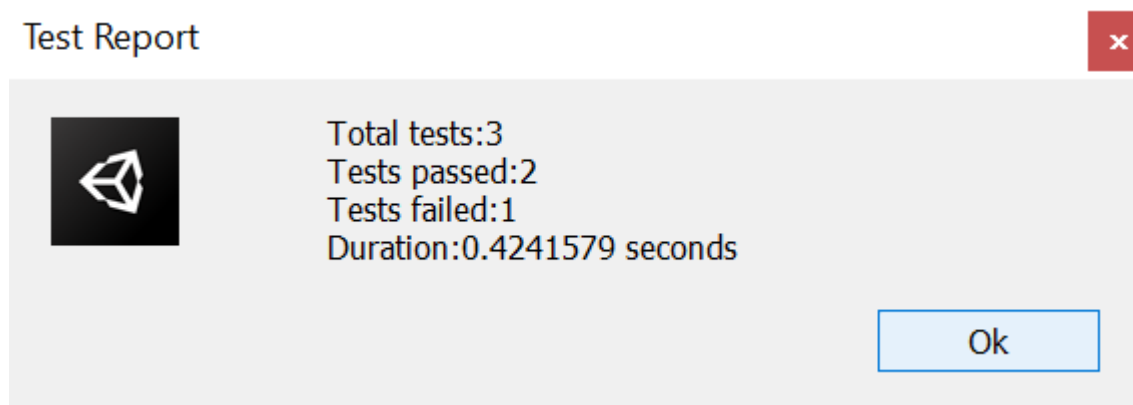


Figure 13 TAF report

This indicates that the tests performed were all successful. The three tests were designed by two of them passing and one failing.

It is possible to use Allure Framework for more accurate reports if required. Please follow the documentation of Allure Framework (18) on usage. A lot of different languages are allowed to be used with this framework and it gives a nice reporting mechanism to the QA people and management as well. Failed test cases and succeeded ones can be obtained and being shown in many ways of wish. Please follow the Allure Framework for further details on how to use.

5 Results and Analysis

This section discusses the code reports, the gained values by this guidance and future considerations as well.

5.1 Analysing Code Reports

The code reports were not made very accurate as this thesis has used the reports from Unity. Those reports are kind of simple and tell what was happening when running the tests. As in Section 4.3, the three code samples were run to prove that TAF code is functional and possible to run and give results. Refer to Section 4.3 for more about the code samples used.

As further mentioned in Section 4.5, the Allure Framework could currently be used to provide exact reports. Those could contain more details for QA or Project managers/Stake holders for further study the failures. Many different types of failures can be obtained and not all concern the interest of all. It is recommended that a QA person with the Project Manager would agree to whom they can be sent. For example, functional faults are for engineers/coders, analytics related to stakes holders, faults caused by incorrect design to designers of the game, etc.

The Diego Clash game was tested by Altom TAF code, and the reports are clearly successful and proof that the game works as expected.

5.2 Gained Values Analysis

This thesis did not directly use the code of any of Kuuasema games. The idea was to create a new game called "Diego Clash". The testing code generated depends on how to use the decided TAF.

It is recommended to always look at each of the frameworks selected to create the TAF code and the provided documentation on how to use it. In this thesis scope Altom selection required looking at their documentation and learning how to start. Bit by bit, the

knowledge grows and more complicated TAF scripts can be written to perform the required tests.

Altom AltUnity Tester does not require a very highly paid Unity package and it works well with the free version of it. The problem is if the company to apply it uses a professional licence to the product, then the one using the Altom solution will also require a key to use one of the licenced Unity products and this is too expensive. As an example, one unity pro license cost \$1800 as an annual fee and if two QA personal would need this, then two seats would cost \$3600 annually.

The efficiency of using the TAF code is making sense for long lasting projects of many years. The reason is that the creation of the TAF code will be efficient to run over the years for those parts that for example does not change that much in the core engine of the game or similar. New code will always require new TAF tests.

Notice that the automated Smoke tests run many times monthly and could be the same tests running on multiple platforms. Time effort wise this may save up to 5 hours/month of time for one project that requires to run the tests on two different platforms twice a month, not that Kuuasema one time smoke test run average has been around 75 minutes/ platform. Table 4 shows the calculations related to time saved.

Table 4 Calculation table of time saving/project smoke test run

Smoke test for 4 weeks of time	Platform	Time used in hours
2 Android runs	Android – device	75 min x 2 = 2.5 hours
2 iOS runs	iOS – device	75 min x 2 = 2.5 hours
		Total = 5 hours saving time

For more devices more time is saved as Table 4 would indicate.

Functional tests for a particular feature are good to run once the future gets new content. A feature can be one of the game functionalities. For example, adding more buttons to the game and making sure that the functionality did not break and then the TAF code could be run again and if it passes then all good. But this usage is less active than the

smoke tests scenarios because changes to a particular feature do not necessarily happen that easily. It is also hard to know how much time or resources this could in effect save.

Running the whole TAF code at once after introducing a new major feature could be a very good idea. This could be because many features interacting with each other is a major matter, especially in the game field of industry. Time saving cannot be measured for this, but from the quality aspects, this is a great step.

Time is required from QA personnel to create the TAF code. This may require some support from coders and designers assisting code and scenarios creations. Depending on the test case it could take from 15 minutes up to an hour of time. This also depends on the simplicity of the TAF selected and how well it is documented to be used.

5.3 Future View Based on Analysis

The usage of TAF consume a lot of time and require some education, for example programming related and learning of TAF. Results bring more to the quality side but does not seem to save the time resources that much.

There are always other choices to plan the tests within Kuuasema by QA personnel and request another professional company to perform their automation.

6 Discussions and Conclusions

The aim of the study was to take the reader through by introducing the test automation in principle, as well as to give an idea of what the possible frameworks are and what was selected and why. Going through the selected TAF as well as building a small game and testing it was all done for the convenience of the reader.

Calculations were made to prove that the TA is very good to the games business but requires a lot of effort from the QA personnel to learn and have some tools to comply with the SW used by the company.

The main test parts of games are visual and require very observable QA people as to the look and feel parts and as to how the game looks like when playing it. Assets and calculation and doing some requests are the parts that can be tested by automation. It is worth knowing that Unit testing is made by developers of the game SW within Unity.

All in all, if the company wishes to utilize a growing functional testing suit to be executed every now and then or each time a new feature arrives. It is also very handy with the smoke tests that are executed periodically before releasing the game SW.

Whether the automation can save time or not depends on the company support and if they wish to invest in purchasing licenses for example to Unity, Altom TF or any other. TAFs also may ask for support, and this costs according to each framework selected. After the cost, it is important to calculate the need for either courses required for the QA personnel and if the time to create the TAF suits is feasible or not. As listed in this document, it is being concluded that the TAF is best working with bigger games/systems that grow with time because the test suits make sense to execute repeatedly once a new major functional feature hit on.

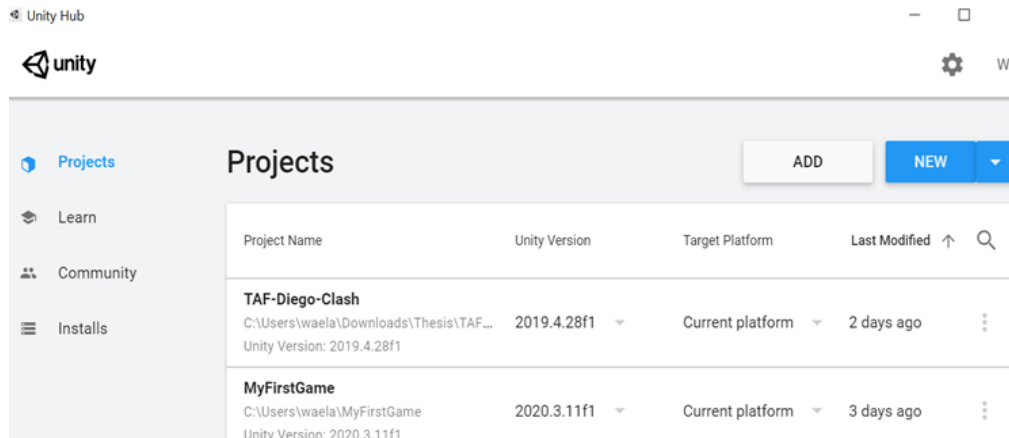
References

- 1 Kuuasema | We create the best games for your audience [Internet]. Kuuasema. 2021 [cited 6 September 2021]. Available from: <https://www.kuuasema.com/>
- 2 Altom. 2021. Altom || Software Testing Services, Tools and Courses. [online] Available at: <https://altom.com/>
- 3 Newzoo Global Games Market Report 2021 | Free Version | Newzoo [Internet]. Newzoo. 2021 [cited 15 September 2021]. Available from: <https://newzoo.com/in-sights/trend-reports/newzoo-global-games-market-report-2021-free-version/>
- 4 App Annie | The App Analytics and App Data Industry Standard [Internet]. App Annie. 2021 [cited 15 September 2021]. Available from: <https://www.appannie.com/en/>
- 5 BBST® By Cem Kaner - Black Box Software Testing Courses Online [Internet]. BBST®. 2021 [cited 10 September 2021]. Available from: <https://bbst.courses/>
- 6 AltUnity Tools - Unity Test Automation - Altom [Internet]. Altom. 2021 [cited 10 September 2021]. Available from: <https://altom.com/testing-tools/altunitytester/>
- 7 AltRunner - Altom [Internet]. Altom. 2021 [cited 10 September 2021]. Available from: <https://altom.com/testing-tools/alrunner/>
- 8 AltWalker - Model Based Testing Tool - Altom [Internet]. Altom. 2021 [cited 10 September 2021]. Available from: <https://altom.com/testing-tools/altwalker/>
- 9 AltTap - Altom [Internet]. Altom. 2021 [cited 10 September 2021]. Available from: <https://altom.com/testing-tools/alttap/>
- 10 TurnTable - Altom [Internet]. Altom. 2021 [cited 10 September 2021]. Available from: <https://altom.com/testing-tools/turntable/>
- 11 Technologies U. Download Unity [Internet]. Unity Store. 2021 [cited 10 September 2021]. Available from: https://store.unity.com/download?_ga=2.177678629.1846901966.1623995212-984039917.1623482624&_gl=1%2A237n9w%2A_ga%2AOTq0MDM5OTE3LjE2MjM0ODI2MjQ.%2A_ga_1S78EFL1W5%2AMTYyNDxMTQ0My40LjEuMTYyN-DxMTUxOS41MQ..&ref=personal
- 12 Start creating - Unity Learn [Internet]. Unity Learn. 2021 [cited 10 September 2021]. Available from: <https://learn.unity.com/project/start-creating>
- 13 11. GitHub Desktop [Internet]. GitHub Desktop. 2021 [cited 10 September 2021]. Available from: <https://desktop.github.com/>
- 14 12. [Internet]. 2021 [cited 10 September 2021]. Available from: <https://assetstore.unity.com/packages/tools/utilities/altunity-tester-ui-test-automation-112101>
- 15 Windows [Internet]. Windows. 2021 [cited 10 September 2021]. Available from: <https://www.jenkins.io/doc/book/installing/windows/>

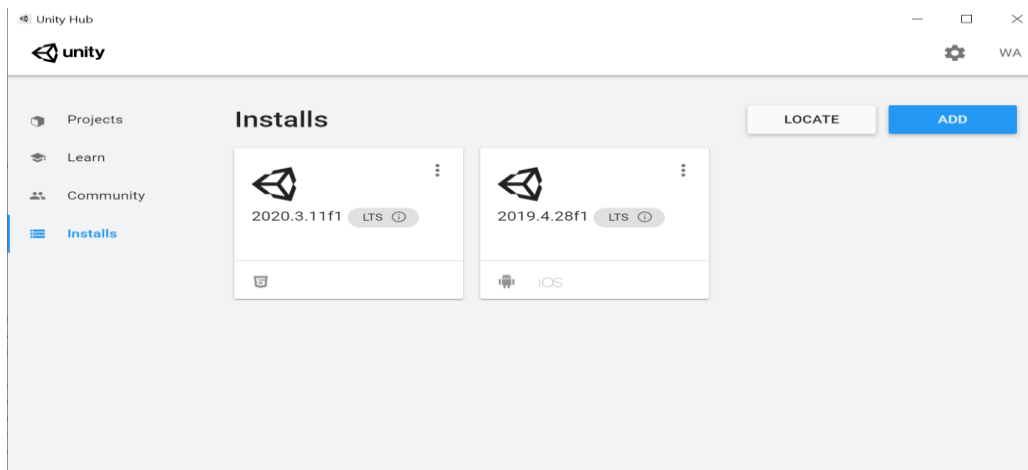
- 16 Tutorials overview [Internet]. Tutorials overview. 2021 [cited 10 September 2021]. Available from: <https://www.jenkins.io/doc/tutorials/>
- 17 Jenkins - Setup Build Jobs [Internet]. Tutorialspoint.com. 2021 [cited 22 September 2021]. Available from: https://www.tutorialspoint.com/jenkins/jenkins_setup_build_jobs.htm
- 18 Allure Framework [Internet]. Docs.qameta.io. 2021 [cited 16 September 2021]. Available from: <https://docs.qameta.io/allure/>

Appendices

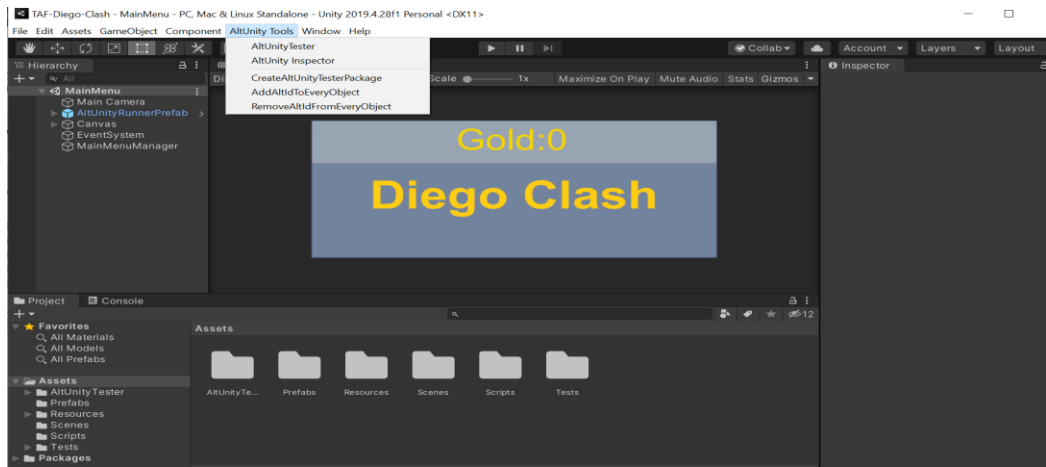
Appendix 1. Unity project main page



Appendix 2. The version suggested to be used by Unity



Appendix 3. The view when AltUnity Tester and AltUnity Inspector are installed



Appendix 4. Game data C# code

```
1 public static class GameData {  
2     public static int GoldAmount;  
3     public static bool HasBug = false;  
4 }
```

Appendix 5: Game play manager part 1 and part 2

Part 1:

```
1 using System.Collections.Generic;
2 using UnityEngine;
3 using UnityEngine.SceneManagement;
4
5 public class GameplayManager : MonoBehaviour {
6     public int GoldFromLevel => buttonAmount;
7     private readonly Dictionary<int, NumberButton> buttons = new Dictionary<int, NumberButton>();
8
9     [SerializeField]
10    private int buttonAmount;
11
12    [Header("UI References")]
13    [SerializeField]
14    private Transform buttonContainer;
15    [SerializeField]
16    private GameObject claimPrizeButton;
17    [SerializeField]
18    private GameObject nextLevelButton;
19    [SerializeField]
20    private NumberButton numberButtonPrefab;
21    [SerializeField]
22    private TopBarManager topBarManager;
23    private static GameplayManager instance;
24    private int nextButtonNumber;
25
26    private void Awake() {
27        if (instance == null) instance = this;
28    }
29
30    public void Start() {
31        GenerateLevel();
32    }
33
34    private void OnDestroy() {
35        if (instance == this) instance = null;
36    }
37
38    private void GenerateLevel() {
39        for (var i = 1; i <= buttonAmount; i++) CreateButton(i);
40
41        RandomizeButtons();
42        nextButtonNumber = 1;
43    }
44
45    private void RandomizeButtons() {
46        foreach (var buttonEntry in buttons) {
47            var siblingIndex = Random.Range(0, buttonAmount);
48            buttonEntry.Value.transform.SetSiblingIndex(siblingIndex);
```


Part 2:

```

48         buttonEntry.Value.transform.SetSiblingIndex(siblingIndex);
49     }
50 }
51
52 private void CreateButton(int i) {
53     var button = Instantiate(numberButtonPrefab, buttonContainer);
54     button.Initialize(i);
55     buttons.Add(i, button);
56 }
57
58 public static void NumberButtonClicked(int orderNumber) {
59     instance.NumberButtonClicked_Impl(orderNumber);
60 }
61
62 private void NumberButtonClicked_Impl(int orderNumber) {
63     if (nextButtonNumber > buttonAmount) return;
64     if (orderNumber != nextButtonNumber) {
65         ResetLevelProgress();
66         return;
67     }
68
69     buttons[orderNumber].ToggleActive(true);
70
71     nextButtonNumber++;
72     if (orderNumber == buttonAmount) ShowClaimPrizeButton();
73 }
74
75 private void ShowClaimPrizeButton() {
76     claimPrizeButton.gameObject.SetActive(true);
77 }
78
79 public void OnClick_ClaimPrizeButton() {
80     GameData.GoldAmount += GameData.HasBug ? -1 : buttonAmount;
81     claimPrizeButton.SetActive(false);
82     nextLevelButton.SetActive(true);
83     topBarManager.UpdateValues();
84 }
85
86 private void ResetLevelProgress() {
87     foreach (var buttonEntry in buttons) buttonEntry.Value.ToggleActive(false);
88     nextButtonNumber = 1;
89 }
90
91 public void OnClick_LoadNextLevel() {
92     var currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
93     var nextSceneIndex = currentSceneIndex + 1 < SceneManager.sceneCountInBuildSettings ? currentSceneIndex + 1 : 0;
94     SceneManager.LoadScene(nextSceneIndex);
95 }
96 }

```

Appendix 6: Code for starting and quitting of the game

```

1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3  using UnityEngine.UI;
4
5  public class MainMenuManager : MonoBehaviour {
6      [SerializeField]
7      private Toggle bug;
8
9      private void Start() {
10         bug.isOn = GameData.HasBug;
11     }
12
13     public void OnClick_StartGame() {
14         SceneManager.LoadScene("Level1");
15     }
16
17     public void OnClick_QuitGame() {
18         Application.Quit();
19     }
20
21     public void OnToggleScoreBug(bool bug) {
22         GameData.HasBug = bug;
23     }
24 }

```

Appendix 7: Code for handling of buttons numbering and their names

```

1
2  using UnityEngine;
3  using UnityEngine.UI;
4
5  public class NumberButton : MonoBehaviour {
6
7      [SerializeField]
8      private Text buttonText;
9      [SerializeField]
10     private Image activeImage;
11
12     private int orderNumber;
13
14     public void Initialize(int i) {
15         ToggleActive(false);
16         orderNumber = i;
17         buttonText.text = $"{orderNumber}";
18         buttonText.name = $"ButtonText_{orderNumber}";
19         activeImage.name = $"ButtonImage_{orderNumber}";
20         gameObject.name = $"Button_{orderNumber}";
21     }
22
23     public void OnClick() {
24         GameManager.NumberButtonClicked(orderNumber);
25     }
26
27     public void ToggleActive(bool b) {
28         activeImage.gameObject.SetActive(b);
29     }

```

Appendix 8: Code for handling of Top Bar Management

```
1 using System;
2 using UnityEngine;
3 using UnityEngine.SceneManagement;
4 using UnityEngine.UI;
5
6 public class TopBarManager : MonoBehaviour {
7
8     [SerializeField]
9     private Text goldAmountText;
10
11     private void Start() {
12         UpdateValues();
13     }
14
15     public void OnClick_BackToMenu() {
16         SceneManager.LoadScene("MainMenu");
17     }
18
19     public void UpdateValues() {
20         goldAmountText.text = GameData.GoldAmount.ToString();
21     }
22 }
```

Appendix 9: Test fail code sample

```
1 using NUnit.Framework;
2 using Alton.AltUnityDriver;
3
4 public class TestFail
5 {
6     public AltUnityDriver AltUnityDriver;
7     //Before any test it connects with the socket
8     [OneTimeSetUp]
9     public void Setup()
10    {
11        AltUnityDriver = new AltUnityDriver();
12    }
13
14    //At the end of the test closes the connection with the socket
15    [OneTimeTearDown]
16    public void TearDown()
17    {
18        AltUnityDriver.Stop();
19    }
20
21    [Test]
22    public void Test()
23    {
24        AltUnityDriver.LoadScene("Level1");
25        var claimPrizesButton = AltUnityDriver.FindObject(By.NAME, "Button_ClaimPrize", By.NAME, "", false);
26        Assert.IsTrue(claimPrizesButton.enabled);
27    }
28 }
```

Appendix 10: Test pass of Diego Clash game starts and can move from main menu level

```

1  using Alton.AltUnityDriver;
2  using NUnit.Framework;
3
4  public class TestMainMenuToLevel1 {
5      private AltUnityDriver altUnityDriver;
6
7      //Before any test it connects with the socket
8      [OneTimeSetUp]
9      public void Setup() {
10         altUnityDriver = new AltUnityDriver();
11     }
12
13     //At the end of the test closes the connection with the socket
14     [OneTimeTearDown]
15     public void TearDown() {
16         altUnityDriver.Stop();
17     }
18
19     [Test]
20     public void Test() {
21         altUnityDriver.LoadScene("MainMenu");
22         altUnityDriver.FindObject(By.NAME, "Button_StartGame").Tap();
23         Assert.AreEqual("Level1", altUnityDriver.GetCurrentScene());
24     }
25 }

```

Appendix 11: Test pass of Diego Clash game playing of Diego Clash game at level 2

```

1  using Alton.AltUnityDriver;
2  using NUnit.Framework;
3
4  public class TestPlayLevel2 {
5      private AltUnityDriver altUnityDriver;
6
7      //Before any test it connects with the socket
8      [OneTimeSetUp]
9      public void Setup() {
10         altUnityDriver = new AltUnityDriver();
11     }
12
13     //At the end of the test closes the connection with the socket
14     [OneTimeTearDown]
15     public void TearDown() {
16         altUnityDriver.Stop();
17     }
18
19     [Test]
20     public void Test() {
21         altUnityDriver.LoadScene("Level2");
22         //Assert that ButtonText_1 text content is "1"
23         Assert.AreEqual("1", altUnityDriver.FindObject(By.NAME, "ButtonText_1").GetText());
24
25         //Assert that Button 1 highlight image is not enabled
26         Assert.IsFalse(altUnityDriver.FindObject(By.NAME, "ButtonImage_1", By.NAME, "", false).enabled);
27         altUnityDriver.FindObject(By.NAME, "Button_1").Tap();
28         //Assert that highlight image is enabled
29         Assert.IsTrue(altUnityDriver.FindObject(By.NAME, "ButtonImage_1").enabled);
30         altUnityDriver.FindObject(By.NAME, "Button_2").Tap();
31         altUnityDriver.FindObject(By.NAME, "Button_3").Tap();
32         altUnityDriver.FindObject(By.NAME, "Button_4").Tap();
33         altUnityDriver.FindObject(By.NAME, "Button_5").Tap();
34         altUnityDriver.FindObject(By.NAME, "Button_6").Tap();
35
36         var claimPrizesButton = altUnityDriver.FindObject(By.NAME, "Button_ClaimPrize");
37         Assert.IsTrue(claimPrizesButton.enabled);
38
39         int goldFromLevel = int.Parse(altUnityDriver.FindObject(By.NAME, "GameManager").GetComponentProperty("GameManager", "GoldFromLevel"));
40         var goldAmountText = altUnityDriver.FindObject(By.NAME, "GoldAmount");
41
42         int goldAmountBefore = int.Parse(goldAmountText.GetText());
43         claimPrizesButton.Tap();
44
45         int goldAfter = int.Parse(goldAmountText.GetText());
46
47         Assert.AreEqual(goldAmountBefore + goldFromLevel, goldAfter);
48     }
49 }

```

Appendix 12: Test pass of Diego Clash game playing of Diego Clash game at level 2

Kuuasema / TAF-Diego-Clash Private

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main TAF-Diego-Clash / Assets / Scripts /

Add AltUnityTester, add couple tests + one failing

..

GameData.cs	Basic unity test project
GameData.cs.meta	Basic unity test project
GameplayManager.cs	Add AltUnityTester, add couple tests + one failing
GameplayManager.cs.meta	Basic unity test project
MainMenuManager.cs	Basic unity test project
MainMenuManager.cs.meta	Basic unity test project
NumberButton.cs	Add AltUnityTester, add couple tests + one failing
NumberButton.cs.meta	Cleaning and renaming
TopBarManager.cs	Basic unity test project
TopBarManager.cs.meta	Basic unity test project

Appendix 13: Test pass of Diego Clash game playing of Diego Clash game at level 2

Kuuasema / TAF-Diego-Clash Private

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

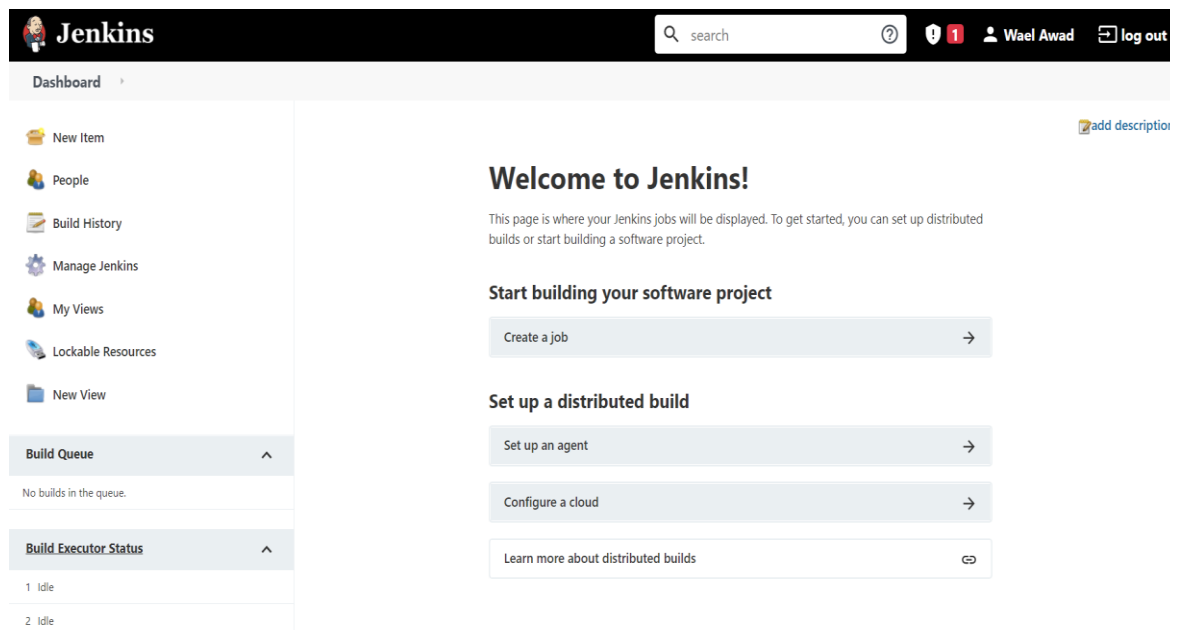
main TAF-Diego-Clash / Assets / Tests / Editor /

Add AltUnityTester, add couple tests + one failing

..

TestFail.cs	Add AltUnityTester, add couple tests + one failing
TestFail.cs.meta	Add AltUnityTester, add couple tests + one failing
TestMainMenuToLevel1.cs	Add AltUnityTester, add couple tests + one failing
TestMainMenuToLevel1.cs.meta	Add AltUnityTester, add couple tests + one failing
TestPlayLevel2.cs	Add AltUnityTester, add couple tests + one failing
TestPlayLevel2.cs.meta	Add AltUnityTester, add couple tests + one failing

Appendix 14: Jenkins view after installation



The screenshot shows the Jenkins dashboard interface. At the top, there is a navigation bar with the Jenkins logo, a search bar, a help icon, a notification icon with a red '1', the user name 'Wael Awad', and a 'log out' button. Below the navigation bar is a 'Dashboard' section with a left sidebar containing menu items: 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', 'Lockable Resources', and 'New View'. The main content area features a 'Welcome to Jenkins!' message, a sub-header 'Start building your software project', and a 'Create a job' button. Below this is another sub-header 'Set up a distributed build' with three buttons: 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'. On the left, there are two expandable sections: 'Build Queue' (showing 'No builds in the queue.') and 'Build Executor Status' (showing two 'Idle' executors).

Jenkins search ? ! 1 Wael Awad log out

Dashboard

[add description](#)

Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job →

Set up a distributed build

Set up an agent →

Configure a cloud →

Learn more about distributed builds ↻

Build Queue ^

No builds in the queue.

Build Executor Status ^

1 Idle

2 Idle