UDC 004.42,94

K. Hazin, A. Volokyta

VIDEO GAME TEST AUTOMATION APPROACH

Abstract: The article is devoted to the research of video game test automation and means to achieve it. Test automation helps speed up the development process and create video games more efficiently. But rates of test automation in the industry are low. Classic means of test automation do not easily apply to the game development field, plus they don’t cover non-functional layers of development. It is caused by the nature of the field with rapid changes in design and requirements and its multidisciplinarity, where correctness is not enough to assess the quality of software. Considerable attention is paid to examine different levels of test granularity and their effect on testing: assertions, unit tests, integration tests, End-to-End tests and non-functional tests. The article overviews different levels of testing and examines a case study for each level, describing how much value it brings to the team, which adopted it. The solution provides a test approach that is easy to maintain, and enables testing of different non-functional qualitative criteria.

Keywords: test automation, pyramid of testing, assert, non-functional testing.

Introduction

Among other software development fields, video games are one of the least tested of them, and one of the most complex. Nowadays it takes years to patch a modern game for it to be arguably bug-free, and the development takes a lot of time and resources. The job market is full of QA specialists, but the number of bugs in released game titles does not seem to be receding. On the contrary, more modern, bigger games release more buggy, even if they have more than enough QA, and it takes them a lot longer to release and a lot more money to make. One of the reasons for this lies in the lack of test automation during game development. In the figure 1 you can see the comparison of two development timelines. The yellow line shows the development with the test automation. And we can see that the bug count is a lot more manageable.

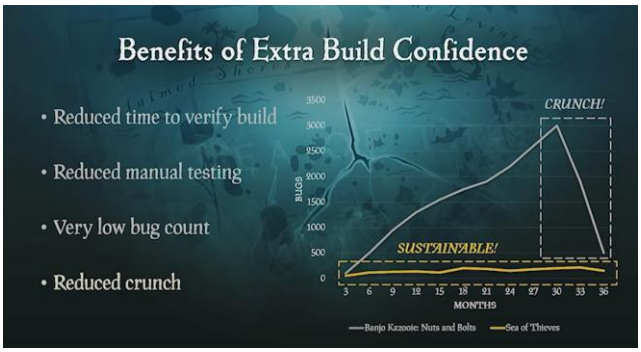

Figure 1. Automated vs manual testing results [1]

 ISSN 1560-8956

Problem tasks

Nowadays, when games only grow in sizes, the development takes longer and longer, and it takes much more people than earlier. One of the ways to reduce people and time cost is to automate processes. One of the least automated game development fields is testing. Test automation can help greatly reduce time and number of people involved to deliver a game. We can see comparisons for automated and manual testing processed in figures 2 and 3, where the same development team at Rare Ltd. compares their projects. In figure 2 we can see how test automation lets them reduce time to verify build from 10 days to 1.5. And in figure 3 we can see how test automation helps them reduce the required testers number from 50 to 17 people.

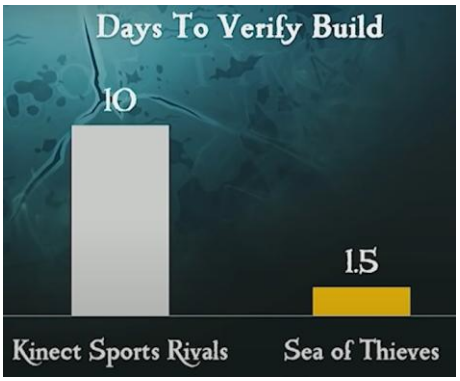

Figure 2. Time to verify build comparison [1]

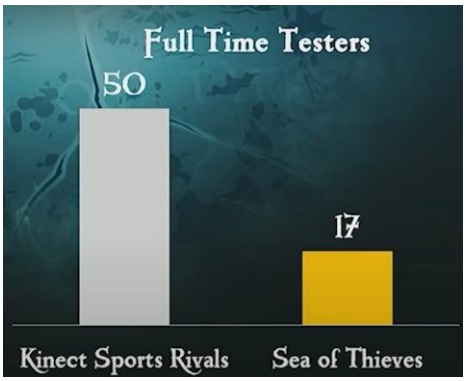

Figure 3. Test team size comparison [1]

Game testing in most companies consists of manual testing of the functional part of the program. Automated testing is only a small part for them. This is due to the fact that the requirements for this software change faster than in other areas of software development. That’s because this kind of software interacts very closely with the user and its success directly depends on how interested the player is in it. Because of this, regular playtests are conducted to assess how much the game meets expectations and that it evokes exactly the emotions in the player that are intended, and depending on this, the requirements for the software change. There is also a very high need for optimization in terms of speed and memory usage. Because of this, the code changes very often and when covering the code with autotests, a lot of effort and time is spent on maintaining the relevance of these autotests, because the code that is tested often changes due to frequent changes in requirements.

Classic test automation approaches use the test pyramid as the foundation of their test structure (Fig.4). But it is disconnected from the state of the industry, which causes the low rates of test automation in game development. This article will address all levels of test automation and how it fits in the industry.

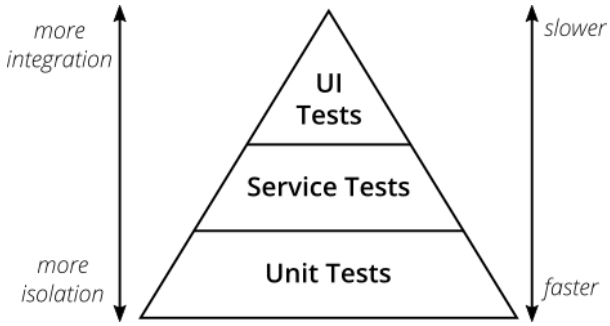

Figure 4. Classic test pyramid [2]

Unit tests

Unit tests are one of the most overlooked test categories in game development, because code base changes very fast and classic unit tests become obsolete quickly, and constantly need updating. Their support cost rises with each code change.

Kevin Dill, in his talk at the Game Developers Conference [3], discussed the specifics of video game code coverage with automated tests. He introduces an alternative definition of unit tests to one presented in the book «Art of unit testing» [4].

The definition presented in the book is: “A unit test is an automated piece of code that invokes a unit of work in the system and then tests a single assumption about the behavior of that unit of work.

A good unit test must:

- Be fully automated;
- Be able to be run in any order if it is part of a test suite;
- Consistently return the same result;
- Be readable;
- Be easy to maintain;
- Reliable;
- Run quickly;
- Run in RAM (for example, without accessing files or a database);
- Have full control over all parts that are running (use system swapping to achieve isolation as needed);
- Test a single logical concept in the system at a time.”

Kevin Dill in his report points out the need to increase the level of granularity of tests. And if we return to the points mentioned above, he corrects them:

- run fast. Up to a certain limit. Not a critical attribute. Since tests can be divided into performance levels. Fast ones are run locally, and slower ones are run during the CI/CD process;
- run in RAM. Not necessarily. The main argument for this point was to ensure the speed of passing tests. So if you need to load files, or a database, or a network, then this can be neglected;

 ISSN 1560-8956

- test a single logical concept in the system at a time and have full control over all parts that are running. Kevin Dill calls these points clearly harmful in the context of video game testing automation. Since requirements change very often, implementations change because of this. And each change in implementation requires changing the code that tests this implementation and the replaced parts of this system that are used for other systems. This is what caused a lot of time spent on maintaining unit tests and the difficulty of implementing them into the system before.

Kevin Dill's solution is to increase the granularity of unit tests. In the general sense, such tests would be called integration or service tests. At the same time, in order to maintain the robustness of the system at a low level of granularity, he suggests implementing tests in the code itself that is to be tested. That is, to make assertions of the state of the system, the state of input and output parameters at a low level in the program code itself. This approach let Kevin Dill adopt unit testing for his team, after ten years of unsuccessful attempts.

Unit tests allowed his team to find old bugs, which were unnoticed by QA for years. The most significant change was the developer velocity. This allowed developers to make more rapid changes, because when they change something they don’t need to check in with other developers to know if the changes are safe. It became a lot easier to make changes to the old code, which hadn't been touched in a long time. And at the same time, it didn’t take a lot of maintenance, because they were high level enough to be resilient to rapid code changes in the details of implementation.

Assertions

Tigerbeetle inc. also advocates assertions. In 3 years, they have developed a state-of-the-art database management system that processes financial transactions faster and more reliably than their counterparts [5]. Their code writing style combined with a testing system allowed them to achieve this goal [6].

One of the most important points from their code writing style is about assertions. They assert each function at the input and output for data correctness, and all checks are performed on the shiping version too. That is, if there is any error in the code, the program crashes. Since the correctness of calculations is critical for a financial database. And if correctness has not been achieved, then this is a critical error. It also allows you to outline the positive and negative space of the program's operation, which allows you to write more reliable and stable code in a shorter period of time.

Together with simulation testing, this helps them keep both reliability and quality at a very high level. Their system is fully deterministic, which allows them to fully simulate the entire distributed database system. Simulation testing can speed up the time many times and help catch very complex and hidden errors. These methods allowed them to achieve success in such a short period of time. A demonstration of simulation testing can be seen in the fig. 5.

Figure 5. Tigerbeetle’s simulation testing [5]

As an example, Jamie Brandon, developer in Tigerbeetle in his blogpost describes how he debugged and fixed a very complex low-level caching problem in just 8-10 hours using assertions, instead of months worth of debugging trying to fix it [7].

Integration tests

At the Game Developers Conference 2019, Robert Masella presented test automation using in-engine scripts [1]. Often, game engines have built-in tools for programming scripts visually, or using simplified interpreted languages that do not require compiling the project, and are executed in an already compiled project. They are often used by developers to write specific passage scenarios, the so-called scripts. Scripts can also be used to automate testing. When a script specifies behavior, and if it matches the expected behavior as a result, then the test is considered successful.

Used as integration tests, since only individual mechanics are tested, most often one by one test scenarios, and take an average amount of time. These scripts are often built into CI/CD processes, where they regularly validate the written code or content. An example of such a script can be seen in figure 6.

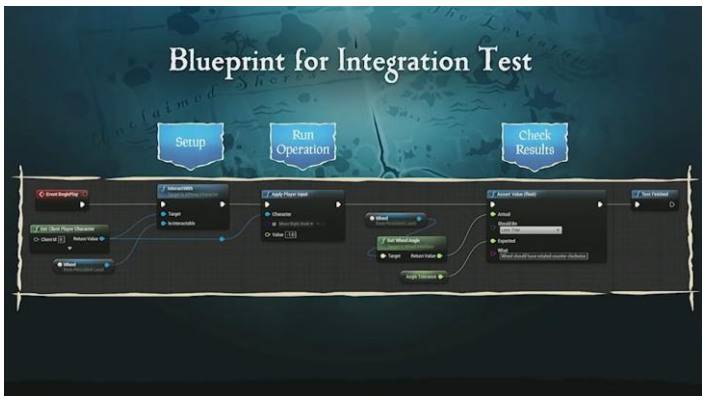

Figure 6. Sea of thieves integration testing [1]

 ISSN 1560-8956

End-to-End tests

To automate End-to-End (E2E) testing, i.e. tests at the top of the testing pyramid, there is a need to increase the granularity of testing compared to integration testing. That is, there is a need to test complete parts of the game, without human participation. This pairs really well with assertions, which in combination can give high system resilience and bug discoverability, mimicking Tigerbeetle’s simulation testing.

Methods of End-to-End testing widely vary from game to game, because the end product can be vastly different and different levels of complexity require different approaches to E2E testing. Some examples of prominent implementations of E2E testing methods can be found in The Division 2 [8] and Retro City Rampage [9]. In Retro City Rampage this approach helped to pass 9 simultaneous console certifications with just only one QA covering all regression testing cases, which were not automated.

These methods allowed developers to test more than just game functioning: performance, level design problems, player heat maps, greatly reduced costs of smoke and regression testing.

Non-functional testing

In his talk at Games Gathering 2021[10], Serhiy Protsenko describes the use of Reinforcement Learning algorithms to automate testing of casual and hyper-casual video games. In addition to classic applications for functional testing, Serhiy also highlighted the capabilities of Reinforcement Learning agents for testing other qualitative criteria by which game quality can be assessed, such as balance and accessibility.

Also, Serhiy Protsenko in his talk highlights the possibility of using Reinforcement Learning to synthesize new game levels. For this, Reinforcement Learning agents are trained to different levels of player skills. Using hyperparameters to construct levels, it is possible to generate levels of different difficulty using this method. An example of using hyperparameters to generate levels can be seen in the fig. 7.

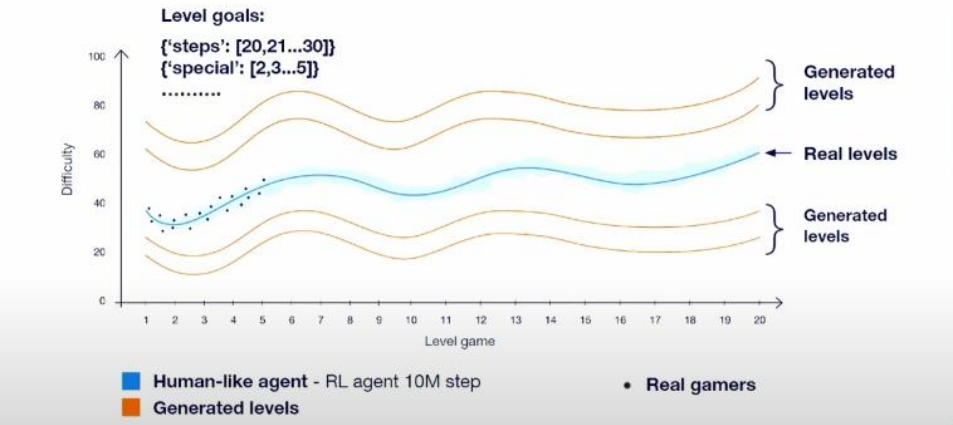

Figure 7. Synthesis of new levels using RL agents [10]

Good balance results in games being more fun and exciting for the players [11]. This approach to non-functional testing allows game designers to reduce iteration time and balance games with more precision, enhancing quality of the product.

Modified test pyramid

From the sections considered, we can conclude that the classic testing pyramid does not meet the needs and standards of the industry. Low levels of the pyramid don’t adapt to change quickly enough, and high levels don’t cover the multidisciplinary nature of the field. And to fix this, a new scheme was drawn up to systematize the testing approach. It consists of five layers, unlike the usual three-layer pyramid. It has an additional layer on top and bottom. The lower layer is responsible for assertions in the code at the smallest level of granularity to keep quality top-notch on the lowest level possible, but retaining flexibility. The highest layer is responsible for automating non-functional quality criteria, which can help with non-technical, multidisciplinary aspects of the industry. The updated pyramid itself can be seen in the fig. 8.

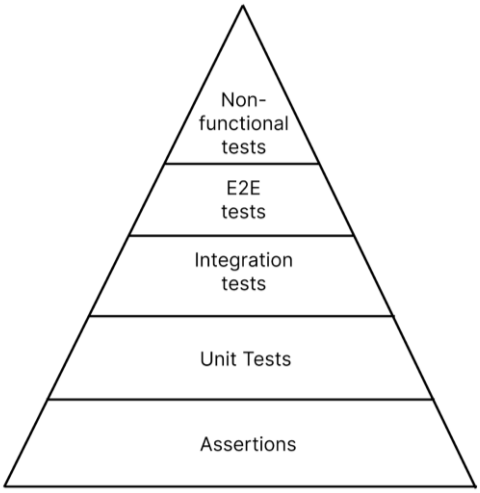

Figure 8. Modified test pyramid

Use case

As an example of usage of this technology we can consider test automation of a multiplayer shooter game.

Assertion level is covered in the code itself. Shooter games heavily rely on correct shooting physics, so asserting in-depth physical calculations should give us a correct result. And assertions help pinpoint the exact cause of the error.

Unit tests level covers distinct feature calls, getting a bit higher level of granularity, than regular unit tests. So instead of setting up unit tests for each physics call, we should cover more complex and complete scenarios, for example a test case shooting a bullet, from start to finish, its physics calculations, collisions, etc. On one hand it shows us a realistic core game

 ISSN 1560-8956

scenario, so it generally should never change, and costs of supporting it should be minimal. On the other hand it’s granular enough to find the reason for the problem, if this test fails.

Integration tests cover complete simulated in-game scenarios. For example running, jumping, shooting lots of bullets in different conditions. It again steps up granularity and highlights problems happening during systems interactions.

For E2E algorithmic testing bots are the best fit for the job. Full server with bots starts and they play a full match. This helps with performance testing, load testing and heat maps. Also covers the network part, where all bots can be connected with various levels of network stability.

For non-functional testing the best fit are RL-agents, which will find various exploits and disbalances. They could cover the balance edge cases of maps, weapons, strategies.

Why not combine non-functional and E2E testing then? Because RL agents are non-deterministic, they need to be regularly retrained, and their learned strategy can skip over some functional problems. Algorithmic bots show more deterministic results and you can have a guarantee that a specific case is covered. For example there is a big event on map, where all human players are most likely to participate. Algorithmic bots can be guaranteed to participate, but if this event is poorly balanced and reward isn’t sufficient enough, RL bots will never participate in it as intended. It can be viewed as algorithmic bots are the equivalent of smoke checks, while RL-based non-functional testing covers edge cases and helps find disbalances.

Conclusion

The proposed approach for game automation addresses the disconnection between the classic test automation approach and the real state of the video game industry.

Different levels of test granularity were researched. The resulting model consists of 5 test layers: assertions, unit tests, integration tests, E2E tests and non-functional tests. Lower levels help with creating a robust technical foundation, while still being flexible and making maintenance easier. Unit tests step up in granularity, making them easier to maintain, while assertions cover the low level of implementation detail. The higher levels allow us to cover the multidisciplinary part of the development and assess more complex non-functional quality criterias such as performance, balance, and accessibility, improving overall non-technical quality of software as well as technical.

With the help of the new test automation approach, we can speed up the development and increase its efficiency. The test automation process can reach new qualitative levels, while being easier to maintain, than classic software development test automation methods. The new approach takes into account features of the game development process and addresses the need for rapid change, while keeping the functional and non-functional quality high.

REFERENCES

1. Automated Testing of Gameplay Features in 'Sea of Thieves' https://youtu.be/X673tOi8pU8?si=0yRdgDt5MPcK9muB (last accessed: 17.03.2025)

2. *Martin Fowler*. The Practical Test Pyramid. https://martinfowler.com/ articles/practical-test-pyramid.html (last accessed 17.03.2025)

3. *Kevin Dill*. Where The $@*&% Are Your Tests?! https://youtu.be/IW5i9DjKT3U?si=bOfmtYQDHLWu-t7t (last accessed 17.03.2025)

4. *Roy Osherove*. The art of unit testing. https://www.artofunittesting.com/definition-of-a-unit-test (last accessed 17.03.2025)

5. Tigerbeetle DBMS presentation https://youtu.be/sC1B3d9C_sI?si=qWGkzBURrRR8NmzR (last accessed 17.03.2025)

6. Tigerbeetle code style https://github.com/tigerbeetle/tigerbeetle/blob/main/docs/TIGER_STYLE.md (last accessed 17.03.2025)

7. *Jamie Brandon*. 0031: 2022, systems distributed, random ids, deleting tombstones, disorderly compaction, juggling blocks, code review woes, holiday shutdown, searching for implementors, everything is copy, sharing the page cache after fysncgate, 9/10 climbers, rise and fall of peer review, real-world concurrency https://www.scattered-thoughts.net/log/0031 (last accessed 22.03.2025)

8. Automated Testing: Using AI Controlled Players to Test 'The Division' https://www.gdcvault.com/play/1026382/Automated-Testing-Using-AI-Controlled (last accessed 17.03.2025)

9. Automated Testing and Instant Replays in Retro City Rampage https://youtu.be/W20t1zCZv8M?si=rzEvScrCFw-6Cfsw (last accessed 17.03.2025)

10. Serhiy Protsenko - How human-like AI bots can take gaming to a new level https://youtu.be/R-h93kDUNQk?si=6vir7svCYtwMYYzk (last accessed 17.03.2025)

11. *A. Becker and D. Görlich*, “What is Game Balancing? - An Examination of Concepts”, paradigmplus, vol. 1, no. 1, pp. 22-41, Apr. 2020.