# Survival of the Tested: Gamified Unit Testing Inspired by Battle Royale

Antonio Materazzo
*Politecnico di Torino*
Turin, Italy
s289015@studenti.polito.it

Tommaso Fulcini
*Politecnico di Torino*
Turin, Italy
tommaso.fulcini@polito.it

Riccardo Coppola
*Politecnico di Torino*
Turin, Italy
riccardo.coppola@polito.it

Marco Torchiano
*Politecnico di Torino*
Turin, Italy
marco.torchiano@polito.it

*Abstract*—While testing takes on a fundamental role to verify software quality and correctness, it often results to be overlooked in the educational field and students often approach it unwillingly, due to its repetitiveness.

Our aim is to exploit gamification to engage students by providing them with dynamics like competition, self-expression, and personal improvement.

We designed and developed Unit Brawl, a gamified application meant to manage multiple rounds, each one consisting of students developing Java programs and unit tests to be executed on each other. The players collect points by writing correct code that does not make the other players' test cases fail, or by writing test cases capable of detecting defects in the other players' code.

The results of a preliminary evaluation to assess the functionality and performance of Unit Brawllook promising. They make us confident about its stability, so we plan an evaluation with students in order to verify the effectiveness of the applied game elements in enhancing the students' interest towards testing topics and their learning.

*Index Terms*—Software Engineering Education, Gamification, Software Testing, Unit Testing, Java

## I. INTRODUCTION

Nowadays, more and more areas of our society are supported by the direct or indirect use of some type of software. Therefore, it can be deduced that software quality and correctness ensure the ordinary course of our everyday life runs on a regular basis. These two software aspects can be examined by means of testing. that is verifying that the behaviour of a system or its components matches their requirements.

Despite these premises, the activity of testing is often overlooked, postponed, or performed poorly [1], leading to user dissatisfaction, interruption of services, economic losses, and, in the worst-case scenarios, risks to public safety.

In the educational field, the teaching of test-related concepts is often absent or overshadowed; moreover, students themselves feel unmotivated to design and carry out tests, due to the nature of the task, which is often considered repetitive and inherently uncreative [2] [3]. The production of poorly-written code, on the other hand, can lead to problems that might remain hidden for a long time, and it may require difficult and time-consuming activities to detect and correct the source of the defects. Hence, it becomes necessary for academic courses to introduce a structured and, yet, enjoyable way of teaching software testing [4].

Gamification is defined as *"the use of game design elements in non-game contexts"* [5]. The main perceived advantage of gamification is to enhance the motivation and engagement of users of a particular task that they would not perform willingly otherwise, ultimately improving their user experience. Gamification elements have been applied in latest years in many areas of society, from business to medical, from services to marketing, leading to benefits when the mechanics are applied and integrated with each other in the right way. In academic literature, it is possible to find many tools leveraging gamification for teaching purposes, concerning information technology in general, but also software testing. Many of them report positive results following the carried-out experiments [6], although there are some studies [7] which report negative effects, too.

With this work, we aim at designing and developing Unit Brawl, a gamified platform to support students with the development and testing processes in Java language, in order to discuss its effects on students' performances and involvement. The final and desirable aim of this work is to improve the quality of students' learning about code development and testing. The tool we propose is a multiplayer gamified environment, to perform unit-level white and black box testing of Java applications. To the best of our knowledge, the approach we propose is the first attempt at introducing a significant multiplayer gamified component to such testing activity.

Unit Brawl is intended to support practice classes in the scope of the Object-Oriented Programming (OOP) course held at the Bachelor Computer Engineering course at Politecnico di Torino. The educational context where the application is going to be exploited plays a fundamental role in the decision of which gamification mechanics to implement and how to integrate them. Considering its current version, Unit Brawl could also be integrated into other Software Engineering teaching realities other than Politecnico di Torino's OOP course, following the constraints and rules explained in the following sections.

The remainder of the paper is structured as follows: firstly, Section II explores other works related to gamification and software testing topics; next, Section III illustrates the problem and analyzes the proposed solution in terms of the adopted gamification mechanics, and in terms of the design choices for implementation; in Section IV we discuss the weaknesses

1

of the approach; lastly, in Section V we provide a final resume, along with improvements and new directions that we aim to follow as our future work.

## II. Background and Related Work

### A. Software testing

Referring to the definition reported by the "IEEE Standard Glossary of Software Engineering Terminology" [8], software testing is "*the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component*". Therefore, it can be deduced the importance of this activity, which guarantees to track the presence of defects in order to solve them before the product release.

There are various possibilities to categorize tests, based on their goal or on their sphere of action. A first, useful categorization concerns the level of specificity:

- **Unit testing** verifies the behaviour of single, isolated software units;
- **Integration testing** combines the components and evaluates the interactions between them;
- **User Interface testing** is focused on the user interface and on the possible interactions between the user and the system;
- **System testing** is executed on the complete system;
- **Acceptance testing** is related to the relationship with the stakeholders, allowing to prove the meeting of predetermined requirements.

In his book "Succeeding With Agile" [9], Mike Cohn defines the "test pyramid", an execution strategy of tests belonging to these categories. The pyramid is grounded on the unit testing level, which needs a large number of tests: at least one for each unit. Unit level is the starting point of the testing process in order to intercept the first bugs in the modules. Next, the above level is integration testing, whose goals are – among others – to streamline the responsibilities of the underneath level, and to verify that the different units correctly embed. At the top of the pyramid is placed UI testing, which can focus on verifying that graphical components are correctly linked to their respective function.

It is worth mentioning an additional (by no means exhaustive) classification, based on the employed techniques:

- **Black-box testing**, to analyze output data as a function of input ones, ignoring interior features of the component or system;
- **White-box testing**, to examine the application or component's behaviour after executing statements, paths or branches;
- **Mutation testing**, whose goal is to find bugs in the software by introducing changes and testing all these versions;
- **Exploratory testing**, carried on by users who can freely explore the system, increasing their level of knowledge during the process.

The decision of which tests to implement in the development phases depends on the goals to be achieved or on the aspects to be analyzed. Black-box testing is meant to verify the functional correctness of the System Under Test (SUT), providing inputs to its modules or interfaces and verifying if the outputs match the expected result, with no visibility of the code. Whilst, white-boxing is meant to verify the correctness of the SUT by having access to the underlying code and analyzing the coverage and the paths followed on the source code. Mutation testing consists in introducing small, intentional changes to the source code of a program, and then running the original test suite to see if any of the tests detect the change: if the test suite is able to detect the changes, it is providing good coverage of the program's behavior, otherwise, it may be missing some important cases and could be improved. Finally, exploratory testing is a technique allowing testers to conduct the activity in an informal and unstructured way. This technique is often done in an incremental and iterative manner, with the tester adjusting the approach based on what learned about the application.

The testing techniques involved in Unit Brawl are black-box and white-box testing, in order for the students to focus on the coverage of the SUT requirements and code quality.

### B. Gamification

In order to frame the gamification topic, Robson et al., with MDE framework [10], provided a rigorous characterization of what falls under the generic name of game element, distinguishing between mechanics, dynamics, and emotions.

*Mechanics* are elements predetermined by the designers of the experience and do not vary from player to player; they concern actions players can take and the whole context, with its rules and limits. *Dynamics* are behaviors and ways to interpret and exploit mechanics which can be partially predicted during the design phase, but which actually emerge only after players join the experience. Lastly, *emotions* are internal states in which players find themselves during the various phases of the gamified experience; from the point of view of the designer, they are the goal to be reached with the choice of which mechanics to implement. All of these elements influence each other and the success of the gamified application will highly depend on how well they are integrated.

Another characterization concerning the existing game mechanics, the way they can be implemented, their effects, and their relations is provided by the framework developed by Yu-kai Chou in the so-called *Octalysis* [11]. In this framework, he distinguishes eight different groups of gamification mechanics, called *Core Drives*, according to the aspects of players' engagement they concern (e.g. narration, sociality, loss, etc.).

In order to emphasize the importance of integration and balancing between these categories, he subdivided them into two different macro groups. One of them ("Right Brain"/"Left Brain") differentiates mechanics based on logical thinking from the ones based on emotional aspects: the first ones result to be used more widely due to their ease of implementation and quickness in providing feedback, but their abuse would
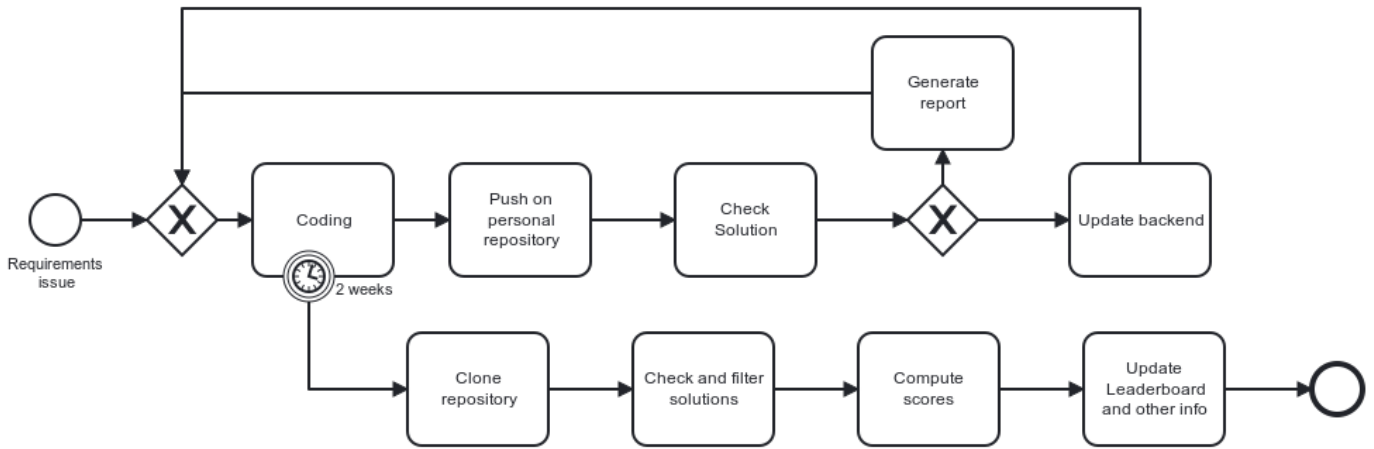
Fig. 1. BPMN representing the workflow of to-be practice classes.

bring to stagnation, driving away potential users. The second subdivision ("Black Hat" and "White Hat") deals with the feelings the mechanics arouse in the user: some of them are quicker in the results, but negative in the long run; others can be used to balance with more long-term positive sensations.

Looking at the application of gamification in the software testing field, various tools have been developed and in most cases, they have led to positive results.

One of the most successful is Code Defenders [12]. It is a turn-based gamified application with two roles facing each other on a Java class: one is the attacker, and the other one plays the role of the defender. The attacker has to develop code mutants (variants of the class with the same functionality), while the defender's goal is to expand with new test cases an existing test suite to find them out. The game intelligently integrates mechanics belonging to narration, sociality, creativity, and satisfaction deriving from progress.

In the field of Exploratory GUI testing, Fulcini et al. defined a gamified framework [13] to make it more appealing and effective, experiencing positive results. They implemented it as a plugin for Scout [14], a tool that enriches the system under test with information to support the tester. After each session, a final score is assigned to the tester (taking into account her interactions, time spent, and issues signaled), based on which a leaderboard is generated. These two elements are introduced as means to motivate the user, but they are supported by others like a progress bar based on the number of interactions, avatars, easter egg injection to insert a random factor, and exploration highlights to give the testers feedback when they discover new pages.

Shifting the attention to test data generation, Amiri-Chimeh et al. developed Rings [15], a single-player mobile game with a purpose, whose aim is to make users (with no programming skill required) generate test data by solving puzzles. Puzzles are strictly linked to constraints that identify some paths of the program: solving a puzzle reflects finding input data that executes the related code path. They consist of a pipe network (with an entrance, an exit, branches, and filters) and rings

of different diameters, colors, and shapes within them. The player's goal is to configure the rings' properties in order to pass all the filters in at least one path from the entrance to the exit. The game succeeds in making human-based test data generation more fun, abstracting it from technical concepts by integrating elements like points, rankings, and levels to give players a way to appreciate their results.

## III. PROPOSED APPROACH

### A. Conceptualization

As stated above, the application context of Unit Brawl will be the "Object-Oriented Programming" course in Politecnico di Torino, through which students learn concepts related to object-oriented programming by means of the Java language.

Currently, students can attend weekly practice classes during the course. They consist of exercises about writing one or more Java classes with some methods to match predefined criteria. A repository is assigned to each student, so they can upload their solution by the deadline. After the deadline, a set of unit tests is executed on each submission to evaluate it, and a test report is sent by email to the respective author.

The scope of these practice classes excludes test development: as a result, many students pay less attention to each aspect of the requirements, and the submitted solutions present failures. In the end, various bugs emerge after the execution of the final tests. Moreover, the engagement of the students typically declines in the long run: a descending trend is verified in their attendance after a few weeks due to the non-mandatory nature of the laboratories.

Our goal is to overcome these weaknesses by introducing Unit Brawl , a gamified platform to enhance engagement and user experience promoting participation in practice class activities.

The main difference with the as-is reality is that each exercise proposed during practice classes will not only involve writing Java code, but also the related unit tests. When the deadline is reached, each test suite will be executed against every solution submitted by the other players in a battle-royale
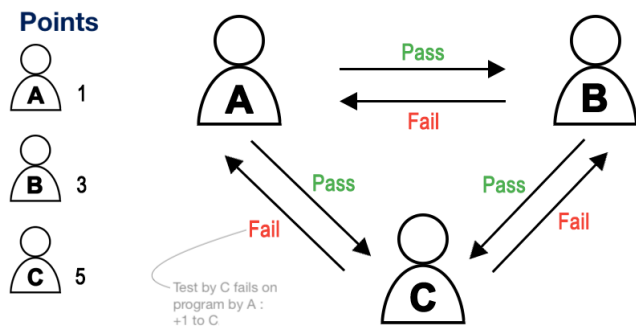
Fig. 2. An example of a hypothetical session involving three players.

fashion. Students can also use a web application to manage some aspects of the process.

### B. Game rules

The workflow of to-be practice classes (shown in Figure 1) is made of two main phases: coding and evaluation.

*Coding* phase begins with the presentation of the exercise, which contains the description of classes to implement and general requirements (such as names and methods' signatures). Students can fork an existing GitLab project (with a pre-defined structure) and have two weeks to submit a solution – i.e. commit and push to the repository – along with some unit tests, whose maximum number varies according to the complexity of the assignment. In this phase, students can push their solutions on their repository as many times as they want: each push will trigger a pipeline (implemented by means of GitLab's Continuous Integration) intended to verify that the submitted tests satisfy the following conditions:

1) the test suite compiles;
2) the test suite does not exceed the maximum size (in terms of number of test cases) allowed for that specific practice class;
3) tests do not fail on a pre-defined (and undisclosed to the students) *ideal* solution.

If any of the above conditions is not satisfied, students can see a report through GitLab's interface. Otherwise, their tests are executed on their own classes in order to obtain the coverage percentage and update a specific section of the web application related to achievements.

*Evaluation* phase takes place when the deadline is reached – two weeks –. To begin with, all the submitted solutions undergo the same testing process described before, obtaining 1 point if they pass this step or getting filtered if they don't. At this point, the remaining solutions are tested by using the test suites developed by all the other players: students get 1 point for each failure found by their tests on other players' solutions and 1 point per each adversary test that does not find any bug in her code. Suppose that players A, B, and C each develop a single test case to be executed on the others' code. The full tests outcome is represented in Figure 2; at the end of the session, player C will score 5 points (1 point for passing validation tests, 2 test cases detecting failures on

enemy code, 2 enemy test cases passing on their code); player B will score 3 points (1 point for passing validation tests, one test case detecting failures on enemy code, one enemy test case passing on their code); player A will score 1 point for passing validation tests.

### C. Applied Game Elements

The selection of the gamified mechanics to implement was made not just by taking into account the concept of gamification itself, but above all focusing on the educational context into which the platform will be integrated. Learning is a highly subjective experience, so we concentrated on gamification core drives that could be as generalizable as possible: the main drive is the competition one, but it is supported by elements concerning sociality, self-improvement, and self-expression.

*Points* are the foundation of the majority of the following mechanics. Calculated as explained above, they play a dual role: they represent the status – allowing the user to appreciate its progress compared to herself or other players– and they are exchangeable points, through which users can access aesthetic customization.

*Leaderboards* are the main consequence of points' integration. Their main goal is to show users' progress in relation to other players' results. However, two potential issues were found about their implementation:

- a full and completely visible leaderboard could demotivate users with a lower ranked;
- an unbalanced leaderboard could lead users with few points to give up practice classes.

To overcome these problems, we designed two different types of leaderboards: one depending on the result of the single practice class and a global one (Figure 3). These are generated by points calculated as explained in the previous subsection: the first 10 positions are shown with information about their results and those of the two players ranked right above and below her. By doing so, we reward players performing the best, equally giving the other users clues about the achieved results. Points related to the second type are accumulated lab by lab and they are assigned to players according to the position they reach in the specific practice class. In this way it is possible to shorten distances between global positions, making them insensitive to single points' instabilities of single practice classes.

*Avatars* [1] are one of the most powerful elements to balance the competition drive. Their contribution is manifold:

- they stimulate creativity through customization;
- they exploit the desire to possess rare and expensive items;
- they act as means of self-expression.

Users can buy different avatars through coins, which can be obtained by converting points (one-to-one ratio) related to practice classes.

*Achievements* have been introduced to further emphasize the importance of reaching personal goals. Their objectives are not

---

[1] Avatars' images are generated through getavataaars.com

| # | Player | | Points |
|---|---|---|---|
| 1 | | Mattia | 14000 |
| 2 | | Clodia | 13965 |
| 3 | | ant | 13930 |
| 4 | | Neil | 7400 |
| 5 | | jack_99 | 2314 |
| 6 | | Wander | 1985 |

Fig. 3. An example of the global leaderboard with players' positions, avatars, nicknames, and points.

| Game Element | Purpose | Octalysis Core Drive | White / Black Hat |
|---|---|---|---|
| Points | Quantify user's progress and buy avatars | Accomplishment, Ownership | White |
| Leaderboard | Allow the user to appreciate her progress compared to other players' | Accomplishment, Ownership | White |
| Avatar | Balance the competition drive | Ownership, Empowerment | White |
| Achievement | Emphasize the importance of reaching personal goals | Accomplishment | White |
| Live Feedback | Give players real-time feedback about their progress | Accomplishment | White |
| Scarcity of time | Maximize students' focus and productivity | Scarcity | Black |

strictly related to competition, but they aim at giving players goals concerning other aspects of testing (for example, related to tests' coverage percentage on their solution) or platform exploration. Each achievement is combined with a *progress bar* in order to give players real-time feedback about their progress. When the achievement is completed, the progress bar disappears.

Finally, the *scarcity of time* is a concept intrinsic to practice classes' periodic nature. It gives the player the right state of tension to feel the possibility to obtain benefits acting in the short term, maximizing students' focus and productivity. Visually it is depicted through the expiration date of the practice class, which turns into a progress bar as soon as the deadline approaches in order to empower the sense of urgency for the last short period of time.

### D. Implementation

The system is made of four main components (Figure 4):
- A **backend server**, the center of the communications between components; it contains business logic and manages data access.
- **GitLab's Continuous Integration mechanisms**, with the aim of running tests on the submitted solution to give players fast feedback without overloading the server.
- **Two web frontends** to give users (both students and administrators) access points to interact with the system.

User experience has been designed starting from the study of users' possible interactions with the platform, leading to the creation of prototypes, including low-fidelity (paper prototypes) and high-fidelity (digital mockup) before the actual implementation of the various parts [16]. Each design decision was made without losing sight of the available computational power.

The *backend server* has been developed by means of the Express framework. The project is organized following a stack-like structure, from routes and controllers to intercept the request to modules to manage database access. Between these levels are placed the services, which contain the business logic to manage requests and carry on game processes.

*GitLab Continuous Integration* (CI) pipeline is defined in a specific file contained in the root repository and it's executed by a remote environment offered by GitLab itself. The pipeline's purpose is to validate the player's solution after each push on the repository, to get test coverage after executing them on her own solution, and to send the data to the server to update the achievements section. The user's GitLab nickname is attached to the request in order to allow the server to associate requests coming from GitLab's environment with its identity. Furthermore, when a player joins a practice class, it's up to them to specify their repository's URL so the server can clone it before the validation phase starts.

Finally, the two *web frontends* have been developed leveraging the React library. React Bootstrap framework has been used extensively to fasten the development and to give continuity to the user interface.

Players' frontend contains sections to which is possible to navigate, containing:

- *Leaderboard*, which allows users to consult the global leaderboard;
- *Labs* (shown in Figure 5), to recover information about expired or ongoing practice classes and to possibly join the latest;
- *Shop*, to check the avatars full list and to buy them;
- *About*, which explains Unit Brawl's rules;
- *Profile*, where players can find information about personal data, past results, purchased avatars, and achievements' completion percentage (Figure 6).

The administrator's frontend gives the possibility to consult the global and –– here — full leaderboard, past practice classes' results, to manage ongoing practice classes or to start new ones, and to download reports concerning the state of the system (players, practice classes, results, etc.).
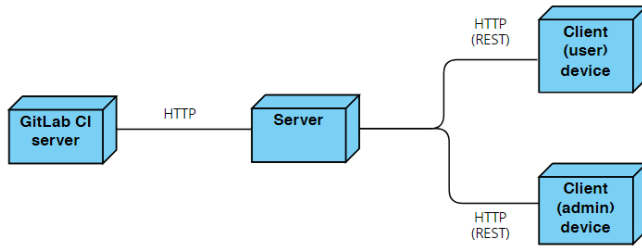
Fig. 4. A Unified Modeling Language (UML) deployment diagram showing the system components and the way they communicate.



Fig. 6. An example of the Profile section, showing player's achievements information.



Fig. 5. An example of the Lab section, showing Clodia's position and the ones right above and below her.

The current state of the implementation of the platform is publicly available at this[2] repository.

*E. Preliminary Evaluation*

A preliminary evaluation was carried on in order to verify the application's behavior and to collect performance metrics.

We started with a solution for a practice class held in the academic year of 2021/2022 and we produced some variations of it, in order to simulate solutions proposed by students. To do so, we injected zero or more bugs in each variation and we wrote tests in order to catch zero or more bugs in other solutions.

Two sessions were carried on. Each one began with the administrator logging into the system and starting a new practice class, inserting – among others – *ideal* solution's URL and the maximum number of submittable tests (7) per each solution. Students registered and logged into the system and joined that practice class, specifying their solution's repository URL. Then, we forced the expiration of the practice class in order to start the evaluation process.

The first simulation involved 3 solutions, each one passing tests about compilation, the ones on the *ideal* solution, and about the maximum number of tests allowed. The second simulation involved 8 solutions, characterized as follows:

- 1 solution had a syntax error;
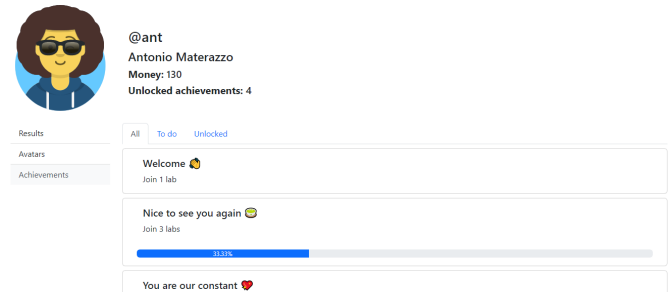- 1 solution had an error in its tests, failing against the *ideal* solution;

[2]https://github.com/antmat99/unit-brawl

- 1 solution contained 8 tests, exceeding the maximum number allowed;
- 4 solutions passed the tests mentioned previously.

The system behaved consistently with the expectations. Regarding the first simulation, each solution underwent the final process, bringing to their score calculation. After that, achievement completion percentages have been updated and points related to the global leaderboard have been assigned. The second simulation's first three solutions got correctly filtered, while the last four took part in the next steps.

With each student logging in, the web application showed information in line with these outcomes.

## IV. LIMITATIONS

The current implementation of Unit Brawl presents some limitations related implementation aspects, discussed in the following.

The first identified limitation concerns the GitLab's Continuous Integration file, which is an asset prone to tampering, being visible to and editable by the student who owns the project. Possible mitigations are to treat as unreliable data coming from this GitLab's Continuous Integration (which is already implemented) and to introduce a unified login mechanism to associate GitLab's usernames with the ones stored by the server.

Additionally, analyzing some hypothetical game situations we identified a possible way of exploiting the game mechanics worthy of further investigation. It concerns the possibility for a player to develop clone tests, i.e. writing multiple different tests which find the exact same bug on an enemy solution. On one hand, if clone tests actually find a bug, they will ensure the player as many points as their number, on the other hand, if they do not succeed in finding any bug, they will turn out to be a waste of time and resources that the user could have employed to cover more cases.

Regarding the generalizability of the platform, we consider the battle-royale approach as applicable to any kind of verification and validation activity, therefore the game mechanics described are not limited to white and black-box unit testing of Java code. The generalizability of the approach is strongly dependent on the tooling that is available for each possible target language and considered testing phase.

The selection of the gamified mechanics to implement has been performed based on state-of-the-art literature, by implementing the most widely adopted mechanics. As many related works highlight, it is not guaranteed that gamified mechanics tested on a specific activity will guarantee the same outcomes when applied in different contexts. There is therefore a possibility that some gamified aspects implemented may turn out detrimental to the student's engagement and learn in the practice classes.

Lastly, despite being subjected to the preliminary evaluation described previously, Unit Brawl is yet to be tested on the field. A thorough evaluation with users is the next scheduled activity, along with the analysis of its result to assess the effectiveness of chosen game mechanics implementation.

## V. Conclusion and Future Work

In this paper, we introduced Unit Brawl, a gamified application to foster unit testing activity in a Java course. It consists of multiple rounds through which players develop Java programs and unit tests to be executed on opposing solutions. The scores computed in each round are used to build a general leaderboard and can be converted to virtual goods that can be used to buy avatars.

With Unit Brawl we aim to bring students closer to the concept of testing, exploiting gamification to make this activity more exciting and interesting to perform by integrating aspects that go outside testing itself. Our final goal is to increase students' focus on the topic and, consequently, their quality of learning about it.

Competition has been chosen as the main drive around which to design the experience. However, elements like avatars and achievements were integrated to provide cross-cutting objectives, complementary to the competition one. By doing so, we avoided generalizing the concept of learning to get closer to as many ways of learning as possible.

After a preliminary evaluation, the system behaved consistently, making us confident about the current implementation's stability.

As the immediate follow-up to this work, we plan to perform a thorough evaluation of the gamified system, to assess the benefits deriving from the gamification aspect in the educational context. We plan to support all the practice classes of the Object-Oriented Programming course (thereby during an entire semester) with the platform, in order to evaluate the learning benefits obtained by the students.

In parallel, we plan to implement and evaluate additional gamification mechanics, by following well-established trends in the literature related to the implementation of a narrative for gamified tasks, and the utilization of graphical metaphors to better resemble an actual game, and to abstract the activities performed by the students. The importance of having metaphors to visualize metrics, or a problem to solve is underlined by Balogh et al. [17], where authors use elements of a virtual city (e.g. building, gardens, ...) to represent various code metrics.

## References

[1] M. Beller, G. Gousios, A. Panichella, S. Proksch, S. Amann, and A. Zaidman, "Developer testing in the ide: Patterns, beliefs, and behavior," *IEEE Transactions on Software Engineering*, vol. 45, no. 3, pp. 261–284, 2019.

[2] G. Fraser, A. Gambi, and J. M. Rojas, "Teaching software testing with the code defenders testing game: Experiences and improvements," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2020, pp. 461–464.

[3] D. Towey and T. Y. Chen, "Teaching software testing skills: Metamorphic testing as vehicle for creativity and effectiveness in software testing," in *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, 2015, pp. 161–162.

[4] M. Marabesi and I. Frango Silveira, "Towards a gamified tool to improve unit test teaching," in *2019 XIV Latin American Conference on Learning Technologies (LACLO)*, 2019, pp. 12–19.

[5] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "c," in *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, ser. MindTrek '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 9–15. [Online]. Available: https://doi.org/10.1145/2181037.2181040

[6] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work? – a literature review of empirical studies on gamification," in *2014 47th Hawaii International Conference on System Sciences*, 2014, pp. 3025–3034.

[7] C. Almeida, M. Kalinowski, A. Uchôa, and B. Feijó, "Negative effects of gamification in education software: Systematic mapping and practitioner perceptions," *Information and Software Technology*, vol. 156, p. 107142, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584922002518

[8] "Ieee standard glossary of software engineering terminology," *IEEE Std 610.12-1990*, pp. 1–84, 1990.

[9] M. Cohn, *Succeeding with Agile: Software Development Using Scrum*, 1st ed. Addison-Wesley Professional, 2009.

[10] K. Robson, K. Plangger, J. H. Kietzmann, I. McCarthy, and L. Pitt, "Is it all a game? understanding the principles of gamification," *Business Horizons*, vol. 58, no. 4, pp. 411–420, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000768131500035X

[11] Y. Chou, *Actionable Gamification: Beyond Points, Badges, and Leaderboards*. Createspace Independent Publishing Platform, 2015. [Online]. Available: https://books.google.it/books?id=jFWQrgEACAAJ

[12] J. M. Rojas and G. Fraser, "Code defenders: A mutation testing game," in *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2016, pp. 162–167.

[13] T. Fulcini and L. Ardito, "Gamified exploratory gui testing of web applications: a preliminary evaluation," in *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2022, pp. 215–222.

[14] M. Nass, E. Alégroth, and R. Feldt, "On the industrial applicability of augmented testing: An empirical study," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2020, pp. 364–371.

[15] S. Amiri-Chimeh, H. Haghighi, M. Vahidi-Asl, K. Setayesh-Ghajar, and F. Gholami-Ghavamabad, "Rings: A Game with a Purpose for Test Data Generation," *Interacting with Computers*, vol. 30, no. 1, pp. 1–30, 02 2017. [Online]. Available: https://doi.org/10.1093/iwc/iww043

[16] M. M. I. of Technology, "Prototyping," 2018, accessed: 2022-11-15.

[17] G. Balogh, T. Gergely, Beszédes, and T. Gyimóthy, "Using the city metaphor for visualizing test-related metrics," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 2, 2016, pp. 17–20.