GAMUT: GAMe-based learning approach for teaching Unit Testing

Renata Faria Gomes
Universidade Federal do Ceará
Fortaleza, Brazil
renatafaria@alu.ufc.br

Valéria Lelli
Universidade Federal do Ceará
Fortaleza, Brazil
valerialelli@ufc.br

ABSTRACT

Software testing is essential to ensure the quality of a system. One of the first levels of testing is the Unit Testing, which aims to test the smallest part of the software, such as objects, methods or classes or modules. Motivated by the relevance of unit tests in the software development process and the lack of undergraduate courses where students can relate the theoretical concepts of tests to practical classes, we propose a game-based learning approach, called GAMUT, linked by a narrative for teaching unit tests. The approach consists of three steps: a game to introduce the concepts of unit testing such as testing doubles and the given-when-then structure; a video lesson that uses similar code of the game to explain and exemplify the previous concepts; and an activity with a challenge, in which the students can practice what they learned for example the writing unit tests. The approach was applied to an undergraduate class of a Verification and Validation course at a university. The results showed that the approach helped to engage the students in the learning process of unit testing since most of them were able to successfully complete the proposed activity. Also, the students enjoyed the game, the narrative and the lucidity of the proposed activity.

KEYWORDS

Unit Testing; Learning; Game based.

ACM Reference Format:
Renata Faria Gomes and Valéria Lelli. 2021. GAMUT: GAMe-based learning approach for teaching Unit Testing. In *20th Brazilian Symposium on Software Quality (SBQS'21), November 8–11, 2021, Vitória, Brazil.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3493244.3493263

1 INTRODUCTION

Testing a software consists of executing a system by simulating input behaviors and analyzing whether the outputs are as expected [34]. To ensure the quality of a system, several types of testing can be used in the software development process. One of the first levels of testing is the unit testing, which focuses on testing the internal logic of the smallest part of a program, called a unit, such as methods and classes in object-oriented programming [15].

In this context, unit tests play a vital role in the software quality since they can be applied in early stages of coding allowing the developers quickly validate their code and thus avoiding that errors being propagated in later stages [4]. However, there is a lack of trained professionals with this knowledge in the industry. According to Valle [34], this lack is closely linked to the traditional method of teaching, whose the main methodology is based on theoretical classes and thus fails to provide practical environments that motivate students in the learning process.

Current solutions have focused on active methodologies in the teaching process to engage the students such as approaches that use problem solving, projects, programming, hybrid teaching and games [21]. Several approaches leverage the gamification elements (*e.g.,* games) to facilitate the teaching of software testing [7][10][11][20]. For example, Lelli et al. [20] have proposed a gamified methodology for teaching Software Engineering (SE) courses by using narratives through an online platform designed to gamification (*ClassCraft*), however, the topics related to software testing are more introductory and thus do not focus on unit tests. Elbaum et al.[10] proposed a game, in which, one of levels is related to white box testing. However, the game missed a key gaming element: a fantasy narrative [23]. Indeed, the narrative of a gamification is a very important element for education since it exemplifies real-world content through metaphors that help students in their understanding [23].

Therefore, this work proposes a game-based learning approach, called GAMUT, for teaching unit testing. Our approach also uses elements of gamification, specially a narrative, to make the teaching of unit tests more practical and accessible for undergraduate students. The approach consisted of three stages: (i) a web game based on a trial-and-error [33] to introduce the concepts of unit testing such as testing doubles, and *given-when-then* structure in a way to engage the students; (ii) a video lesson that uses similar code of the game to explain the theoretical-practical concepts of unit tests; and (iii) an activity with a challenge to allow the students to practice the writing unit tests.

In this study, we have investigated the following research questions:

RQ1. Does our game-based learning approach help the students to learn beyond theory?

RQ2. Does our game-based learning approach help the students to explore the theoretic concepts in the proposed practical activities?

RQ3. What are the benefits that students experience by using our approach to learn?

Due to the COVID-19 pandemic, the approach was applied remotely in a software verification and validation (V&V) course at the Federal University of Ceará (UFC). The approach was evaluated individually by the students, they were invited to answer two questionnaires and a forum with subjective questions. Also, the students were organized in 9 teams to solve the proposed activity. The results showed that the students enjoyed the game and the playful narrative provided through the approach. The students rated the approach between 8-10 on scale of 1 to 10. The results of the activity grading showed that most of teams scored high. Indeed, 4 out of 9 teams completed the exercise successfully. However, only 2 teams were able to correctly solve the challenge and 2 students mentioned that the activity was difficult to solve.

The remainder of this paper is organized as follows. In Section 2, the concepts of gamification and game based and unit testing are presented. In Section 3, the studies related to games that rely on testing teaching and the gamification are discussed. The methodology, the conception of the game, the production of the video lesson and the practical activity are presented in sections 4, 5, 6 and 7, respectively. The results obtained with the application of the approach are presented in Section 8 and a discussion is presented in Section 9. The main threats related to this work is presented in Section 10 and, finally, in Section 11, the final considerations and future directions for this work are presented.

2 BACKGROUND

In this section we present the two topics related to our work: the use of gamification and game-based in education and unit testing.

2.1 Gamification and Game-based learning

Gamification is when game components, for example, objectives, rules, challenge and interactivity, are incorporated to non-game activities. [17][9]. This technique is widely used as a potential solution to make education more entertaining and engaged [11].

Game-based learning is defined when the educational content is incorporated into a game to engage students [6]. Game-based learning differs from gamification in education since the last one uses elements from the games (*e.g.,* avatars, scores) whereas the a game-based learning approach uses properly a game [1] to enhance the learning experience. In this work, we have used a game not only to engage students but also to provide them examples and metaphors.

Al-Azawi et al. [1] state that the Game-based learning uses techniques such as motivation, relevant practice, story, emotional elements, goals and challenges. Some elements from the gamification is also used, for example, the competition in an activity that will be discussed in Section 7.

In this context, there are some key elements related to the engagement: *emotional elements*; a *design* that creates a convincing atmosphere; and a great *narrative* [16] [11].

The *emotional elements* give some experience to the player, for example, victory, defeat and responsibility. In this way, the player engaged to conquer the objective [16]. Furthermore, the aesthetics (*e.g.,* colours, drawings, feedback) complements the emotional elements, resulting to the player to be more immersed in the game. Adding a challenging element, tension peaks that requires attention, also may contribute to the player's immersion in the game [3].

Lastly, the *narrative* is defined as a group of rules that are experienced during the game, in a way that the player interacts with fictitious universe [31]. The narrative is a passive component which facilitates users to understand the objectives and take decisions that will impact on their results, positive or negative [28]. In education, the narrative is a key element since it exemplifies the real world elements using metaphors that assist players understand the abstract subjects more efficiently[23].

2.2 Unit Testing

Software testing is a fundamental part of Software Engineering because it detects if a program works as it should against the requisites [24]. In this context, a relevant test level is the unit testing which is defined as the process of verifying the logic of the smallest part of a software [15].

The first concept, related to unit testing, used in this work is contextualized in [18], which the white box testing writing could be facilitate by using a structure called *given-when-then.* This method divides the test in three sections:

(1) *Given* specifies the existing preconditions before this test, for example, if a previous configuration of an object is required.
(2) *When* is related to the behaviour that will change the previous conditions to the one that will be verified by the unit testing.
(3) *Then* describes the outcome of the behaviour, ensuring that tests have the expected results.

The give-then-structure can be exemplified by a test of the average calculation: **given** an *array* with the values between 6 and 8, **when** the method *calculateAvarage* is called passing as parameter this *array*, **then** the expected return must be 7.

Our teaching approach is also based on simulated objects, called testing doubles. According to Mackinnon et al. [22], testing doubles is an easier way to develop and to maintain the testing of complex scenarios. Briefly, the testing doubles are used to predict unknown domains, simplifying the testing structure [22]. For example, a function to get data from a remote database will be tested using a double that returns a static, independent and known value.

Some academic studies have been used the term *mock* instead of *doubles*, However, this term is adopted to summarize all different types of doubles, as presented in [2] [5] [12]. In this paper, we presented such terms individually as follows:

- **Dummy** is an object without implementation, it is used only to fill methods that require arguments [25].
- **Spy** [19] states an object is used instead of the dependencies to spy the behaviour and the calculated values. Therefore is possible to compare the expected value to the one whose was computed. This double can be also used to measure how many times the function was called.
- **Stub** [5] defines the *stubs* as being fake code implementations that always return the expected results during unit tests.

Moreover, the SUT (System Under Test) expression is used in this paper following the definition presented in [25]: *in the context of unit testing for object-oriented systems, the SUT is any class that is being tested.*

3 RELATED WORK

Elbaum et al.[10] have proposed a web game, called Bug Hunt, where students can apply testing techniques to solve problems.

The game is structured into four missions, one of them is related to white-box testing. In this mission, the source code is available and exists a place to create test cases. The game score increases as code coverage increases. The game has a small context in every mission but, in general, there is almost any narrative then, it is more similar to gamification than an game-based approach. In Bug Hunt, students have the opportunity to practice but has no explanation of how to do it or show best practices for finishing a challenge.

Beppe et al. [4] have proposed a card game to work the concepts of Software Testing. The authors focus on the types of testing such as functional, performance, acceptance and load testing. The gameplay is based on several attacks that the students will use to activate their powers, for example, the tester card. The authors use an educational game to help students to connect the types of testing to their purpose. However, the card game focuses on the theoretical concepts of types of software testing and does not work unit tests.

The study of Valle et al. [14] present a game platform where students are characters who jumps and defeat enemies by using testing challenges. The game brings several elements of an educational game such as narrative, characters and enjoyment. The game focuses on functional, structural and error-based testing techniques. Although the game works with structural testing, the focus is limited to the theoretical concepts of structural testing criteria.

Lelli et al. [20] have proposed a gamification methodology for the teaching of SE courses. The authors use a digital platform, *Classcraft*, to apply the methodology. In the context of the narrative, the students play as heroes to complete the missions. Each mission works a module and each module is based on the SE course's contents. The missions could be asynchronous lessons, activities or educational games. In the authors' methodology, we identified gamification and game-based techniques. The methodology also works both theoretical and practical activities in the proposed missions. However, these missions are more related to SE in general than software testing and thus they do not work the writing of unit tests.

Rojas and Fraser [30] have proposed the Code Defenders game. The idea is one player, called defender, creates a unit test for a class and the other player, called attacker, writes a mutant of that class by introducing some errors to see whether the defender detects the error. The authors explore the writing of unit tests but they do not explain how to write them for instance by showing some techniques as we propose in our approach.

Table 1 summarizes the the aforementioned related studies. Our work proposes an approach that uses a game as motivation and key elements of gamification in a unit testing context, by teaching the theory and using practical exercises and well-known practices for writing unit testing. Only the studies of [10] and [30] focus on the practice of unit tests but none of them address theoretical content for helping students to learn the best practices of such tests. Lelli et al. [20] incorporate both gamification and game-based approach into their methodology, besides that mix theoretical recorded lessons with practical activities. However, the content of unit tests is not worked. [4] and [14] present interesting testing games with enjoyment and pleasure, narrative and challenge, which are elements related to our work. However, the main difference is that our work is more focused on unit testing and how to write it. In addition, we also incorporates part of gamification such as competition between students and activities with scores [1].

Table 1: Comparison of related work

Work	**Content type**		**Game-based**	**Unit Testing**
	Practical	Theoretical		
[10]	x			x
[4]		x	x	
[14]	x	x	x	
[20]	x	x		
[30]	x		x	x
Our work	x	x	x	x

4 METHODOLOGY

Our methodology is divided into three main phases: (i) a game; (ii) a video lesson; and (iii) a practical activity. There is a fourth omnipresent element called narrative which works as a connector for the aforementioned phases.

To apply our approach the steps are as follows: the students first played the game. Then they watched the video lesson, which is recorded to avoid Internet connection problems. Lastly, they could practice the content with a practical activity with a challenge.

The Figure1 shows all the phases and their sub-phases to develop our approach.

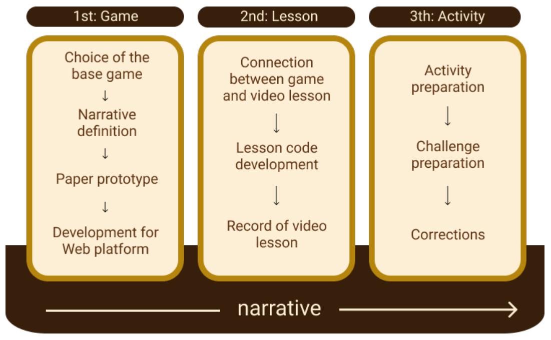


Figure 1: Phases and sub-phases of the work

The first phase is the conception of the game. The goal is to introduce the narrative and the unit test subject in a playful, fun and challenging way. This conception was conducted in four steps. The first one was the selection of a game to support the gameplay and based on the game the narrative could be defined. Then, the gameplay was added by using low-fidelity prototypes (paper prototype) and, finally, after a refinement, the GAMUT was developed to a Web platform.

The second phase is to produce a video lesson that works the theoretical-practical content using the elements of GAMUT to exemplify the writing of unit tests. For example, the theoretical part presents the battle system of the game and the some possible tests such as the attack function. The practical part is developed by using a code similar to the game, the goal is to show to the students how to write unit tests for the functions that they played previously in the game.

To produce the video lesson, we should first think a way to connect it to the web game. Thus, we decided to develop a code based on the game to be used as an example. Once the code was ready, the video lesson was recorded and available on a digital platform.

The last phase is the practical activity with a challenge. The students were invited to practice what they've learnt on the online lesson by doing a homework activity. This activity was created to be done in teams and the time period of one week.

The students received a link to download the code[1] used in the video lesson. The code has some unit tests created as an example. The goal of the activity is to create more tests to non-tested functions in that code. We also proposed a challenge to the students find out problems with the logic of that code by using unit tests. To encourage the competitiveness, teams who successfully completed the challenge earned extra points.

5 GAME CONCEPTION

The game conception was performed in four steps:

(1) the selection of a game to be adapted in the context of our proposal;
(2) the narrative definition;
(3) the paper prototype; and
(4) the game development for the Web platform.

These steps are presented in details in the next subsections.

5.1 Game adaptation

Our proposed game was adapted from the conceptual idea of a boardgame called *Mastermind* [33]. The gameplay of *Mastermind* is based on a trial and error method: the player's goal is to discover a randomly generated password composed of four different colours.

The trial and error were chosen because tests usually are based on inputs and outputs: given an input the result is checked. For example, the given-when-then structure has inputs in the given-phase and outputs in then-phase. Mapping the trial-and-error concept for testing, we can see the trial as an input and the error as a kind of output. We used this concept in the game - the colors combination (one trial) being the given-phase and then the guessing (error or victory) being the then-phase. We believe using the game at the beginning had a positive impact on our approach since students first exercised the conceptual idea of unit testing without writing the test itself.

However, the *Mastermind* game does not have a narrative and other fantasy elements, which are key elements of games since they act intrinsically and provide metaphors and analogies that will be used to link the narrative to the lesson [23]. Additionally, the narrative is more than a thinking guide element, it is a way to mentor the students actions [28]. Motivated by this reason, we decided to include a narrative to connect our approach: the game, the video lesson and the activity.

5.2 Narrative definition

According to [9], the elements of a narrative for gamified educational projects are: an actor, choice elements, an interactivity, an event sequence, a space and time to interact.

We initially defined the space and time in which the game was located. The selected theme for the game story is in the Middle Earth, an universe based on the Nordic mythology inside the temporal window of Middle-age [26].

Next, we selected the character of the student who will play – the actor – who in the game would be a training soldier. The goal of the training is to defeat a monster and to achieve this purpose the player should choose which weapons are the correct ones for the attack, this last part is the choice element of the narrative, providing also the interactivity of the game.

Every attack is an event sequence, each time the player is closer to the victory or the defeat. In addition, the player can also test an attack since she is still learning how to attack the monster. For this purpose we created a training dummy to target the test attacks. Also, the dummy gives feedback about the attack allowing that the warrior fixes what is wrong and try again.

Lastly, the interaction time was limited by the number of attacks and tests attempts which will be explained in the Section 5.3.

Once the narrative was defined, we started the development of the gameplay. We initially designed a paper prototype because the lower cost and higher adaptability [13]. It is noteworthy that the game must include concepts of testing, in such a case is inevitable testing several gameplays followed by multiple changes.

5.3 Paper prototype

In this subphase, the gameplay was tested and adapted by using a low definition prototype made with paper. Beyond the mechanics of the original game (*Mastermind*), it was required to consider the narrative, the unit testing concepts and do how many changes it should be necessary to the gameplay fits in properly, what was achievable using this kind of prototype.

The mechanic to be adapted is a sequence composed of four (out of six) different colors, the first player choose a correct sequence and hide it, then the second player should guess that sequence in order to win, by trial and error. The rounds is repeated until the second player guess the correct sequence or the number of attempts is over (*i.e.* 12) .

Considering the original gameplay, the first change we made was replacing the colors to weapons (*e.g.,* knife, bow and arrow). There is two reasons for this change, the first one is because weapons are more related to the narrative and the second reason is because the gameplay only based on colors can be confusing for colorblind students[29]. The weapons are sequenced on a board and used to attack and defeat the monster. Figure 2 shows a picture of the paper prototype during the tests.

Regarding the number of players, we verified the possibility that extends the game for more than two players. We conducted two tests: one with five players and other in pairs. In both cases, one player was responsible for choosing the correct attack and the others try to guess it. However, we observed that the game with so many players was tiring and less competitive than playing in pairs. Also, playing the game in pairs was less engaged to the player that chose the correct attack. Based on these results, we defined that the final game should be played individually.

We also designed how a player could achieve the game's objective (*e.g.,* win the monster by choosing a right attack). In the game's prototype, the players could test their plays before an attack. To meet this requirement, we introduced the idea of life points, for

[1]https://doi.org/10.5281/zenodo.5504644

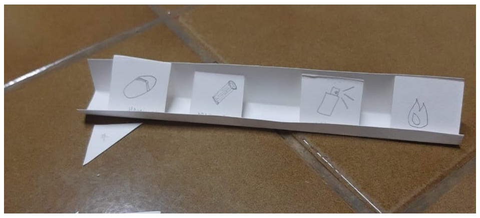


Figure 2: Paper prototype

example, every mistake that a player makes in an attack, she looses a life point. Initially, the number of testing attempts were not limited, but the maximum of attack was three. However, we observed that the players could test several times their attack before a real one since they would not lose points. This unlimited testing attempts made the game so easier, thus, we set a maximum number of testing plays. After a few moves, we balanced the life points for three and the testing points for five, and the player could recover the testing points if she looses a life point during an attack.

During the game's prototype testing, we also observed that players had difficulty remembering the latest tests and what they should change to make the right attack. So, we included this functionality and observed that the players felt more confident to modify and test their guesses, avoiding attack repetition.

5.4 Web Game Development

Once the narrative and the gameplay were defined, we started the development of the game. Given the pandemic of COVID-19, we chose to develop a digital game to be available on an online platform. The game narrative is based on the Middle-Age and thus the design of the game elements such as weapons and the characters (dummy and monster) were designed to meet this medieval period. Also, the game's interface colors are brown and beige to remember the medieval shield. Figure 3 illustrates the design of the game.

The game[2] was developed in *Javascript* language with the framework *framework Jquery*[3].

The home screen of the game has a tutorial that a player can read and close it by clicking on "I'm ready" button. Then, the player can start the game.

The game area is divided into four areas. The top left area contains the board with the weapons and on the top right we can see the panel that shows the attack and life points. In this panel, the player can see how many life points and tests are still available to be used. When a player misses an attack or a test or the number of tests reaches zero, the background of the image changes to get the player's attention, for example, if the points decrease the image of the number of points flashes or if the number of tests reaches zero, the background is completely darkened.

On the middle area, there are the actions buttons such as *attack*, *test* and the *history*, which shows the latest tests made by the player. Such buttons also change when a player interacts with them. For example, when the *test button* is clicked by a player, it darkens and the following text becomes visible to the player: "Change the weapons to continue". The goal is to avoid that a player makes same attacks and looses points unnecessarily.

On the bottom area is where the characters of the game can be visualized, the dummy on the left side and the monster on the right side, in the middle there is a speech balloon where the tips will appear. This balloon changes to signal the responses of each character. For example, when the *Test* button is clicked, the balloon points to the dummy and indicates the result of the player's attack. Or if the *Attack* button is clicked, the balloon points to the monster and informs whether the attack is correct or not, if it is incorrect the number of life points is visualized or if is correct a *pop-up* appears indicating that the player won the game.

We implemented an almost hidden option which is a *developer mode*, its position is like the idea of an *Easter egg*[4]. When this mode is activated some interface elements are modified to be closer to development environment: the monster and the dummy start to be the dependency and the testing double, respectively. These changes can be seen in Figure 3.

The concept of the testing doubles works as a metaphor because the student can write a test using a real dependency or using a testing double, similarly she can use an attack to a real monster or against a training dummy.

6 VIDEO LESSON

The video lesson[5] was elaborated to introduce the concepts related to the writing of unit tests. This phase was divided into three: the mapping between the GAMUT game and the lesson; the example of the code development; and the recording of the video lesson.

To create a connection between the game and the lesson, we first planned how the concepts of unit tests would be instructed and which game elements would be brought to the lesson. In this context, the lesson was divided into two parts: theoretical and practical.

In theoretical part, the unit testing concept was introduced. We have used the example of the game in the lesson, which could be divided into functionalities and every functionality has it's own functions, for example, the battle system has the attack function.

The practical part aimed to show to students how to write unit tests. To facilitate the understanding of the concepts, we used a simple code created in Python, the logic is almost the same to the code of GAMUT game but without graphic interface. The goal is to create connections between the game and the lesson, helping the students to understand better the logic of the tested system.

Initially, we explained the code, then a simple test was written, showing how the library *unittest*[6] works, for example, testing an initial configuration of a class. Then, we presented how to organize the tests using the functions *setup* and *tear down* and the structure *given-when-then* [18], such as "*Given* two correct weapons, *when* the test function is calling, *then* the player should not win the game".

Other content presented in the lesson was how to work with code dependencies. For example, to test the attack function we need to compare the monster sequence to a user sequence and check whether the number of weapons is the expected one. However,

[2]The online game is available on: https://gamut.games
[3]https://jquery.com

[4][32] *Easter egg* states is a hidden function of a game that generates curiosity and make the players want to discover "what is going to happen if I click on this button?".
[5]The slides of the lesson are available on: https://doi.org/10.5281/zenodo.5504633
[6]https://docs.python.org/pt-br/3/library/unittest.html

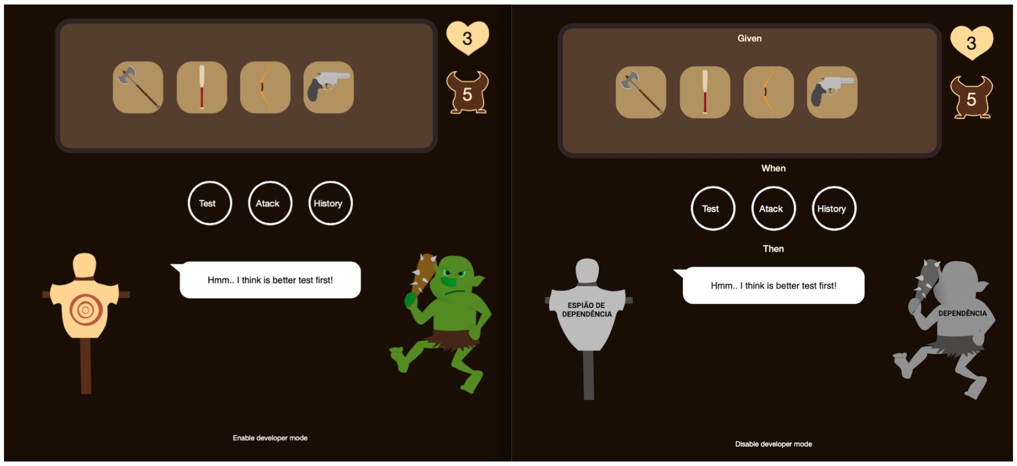


Figure 3: On left the normal game and on right the developer mode.

how it is possible to give a "three correct weapons" if the monster sequence is random? In such a case, the testing double is important to replace the return of the monster sequence to a testing double's known sequence.

The video lesson was previously recorded to avoid problems with Internet connection. The duration of the video lesson is 44 minutes. Also, we uploaded the video in an online video sharing and social media platform[7]. This video was presented in a synchronous online lesson which was conducted remotely through a virtual learning environment of UFC called Solar[8], for Computer Science students of the V&V course at UFC. In this synchronous online lesson, the students could watch the recorded video lesson and ask questions and / or share comments about the topic.

7 PRACTICAL ACTIVITY

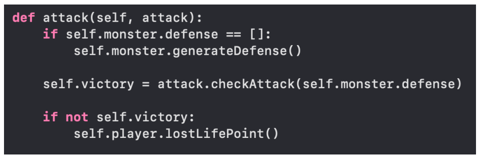
To practice the concepts of unit testing introduced in the lesson, we proposed to the students an activity based on the code presented previously in the video lesson.

The activity[9] had two mandatory exercises and one optional challenge. It should be done in teams between 1 to 4 students to encourage teamwork and knowledge share, with a total of nine teams being organized. The activity should be delivered in one week and, during this time, the students had free access to the professor and the teaching assistants to clarify their doubts.

The first exercise is to create testing doubles and the second is to test the functions *rightAttack*, *rightTool* and *checkIfWin* of the code. An example of the activity is the testing of the function *checkIfWin*, the students have to first create a *stub* of the monster to return a known sequence and then send a correct sequence to this function, if it returns true then we detect the function is working correctly.

The proposed challenge was to identify problems with the code by using unit testing to find the problems. An example of a possible answer for the challenge is: *given* a warrior without life points, *when* she attacks, *then* she should not lose more life points (because the player already lost the game). Figure 4 shows the code of the attack function, there is no code structure to verify if the user can attack or if there are remaining life points to continue playing.

The teams who solved the challenge received the total score. However, the other teams who tried the challenge but failed to complete it also received some scores, our intention was to reward some teams who worked hard to solve the challenge.


```
def attack(self, attack):
    if self.monster.defense == []:
        self.monster.generateDefense()

    self.victory = attack.checkAttack(self.monster.defense)

    if not self.victory:
        self.player.lostLifePoint()
```

Figure 4: Code of the attack function

8 RESULTS AND ANALYSIS

We have applied our approach to introduce unit tests for Computer Science undergraduate students in a V&V course of a university. The V&V course is an elective course, the prerequisite is that students have completed the mandatory Software Engineering course, which is offered at 5th semester. Thus, the students who attend the V&V course are usually in the last semesters of their degree. We have applied our approach in the 2020.2 semester which was conducted remotely due to the of COVID-19 pandemic, in this semester 27 students were enrolled.

We have worked the unit testing topic after the functional tests lessons, and thus the students have already knowledge about functional tests. For example, we have introduced the functional testing concepts and black box techniques such as equivalence partitioning, boundary value analysis and error guessing. The students have practiced by specifying functional test cases using those techniques for real-world systems.

We introduced the unit testing topic using our proposed approach. The unit testing lesson was synchronous (*i.e.* web conference) and conducted into a virtual learning environment (VLE). This lesson was organized in three steps: the GAMUT game; the recorded video lesson and the homework activity. The timeline of

[7]The lesson is available on: https://www.youtube.com/watch?v=R6aWbck5KRg
[8]http://solar.virtual.ufc.br
[9]The full description is available on: https://doi.org/10.5281/zenodo.5504647

this lesson is illustrated in Figure 5. First, the students spent 15 minutes to play the Gamut game and they were invited to send a picture of a successful victory to the course synchronous chat - 17 out of 20 students who attended the web conference completed the challenge. The goal was to engage them in the game activity. Then, the students watched the recorded video lesson. If the students have questions or doubts during this lesson, they are free to ask by microphone or chat. Last, we presented the activity and the students had one week to conclude it.

The activity was elaborated to be done in teams and so the students were organized into nine teams. We also created a forum to collect their feedback and clarify the doubts during the activity.

We shared with the students two questionnaires[10], the first one is to collect their feedback about the game and the video lesson, this questionnaire was based on MEEGA+[27] using 5-points Likert scale [8]. The second questionnaire[11] is to to evaluate the whole approach, including the activity, so the goal was to the identify the difficulties they faced during the resolutions of the exercises (write unit tests based on the code game) and the challenge. We also could evaluate our approach looking at the exercises resolution and reading the forum posts.

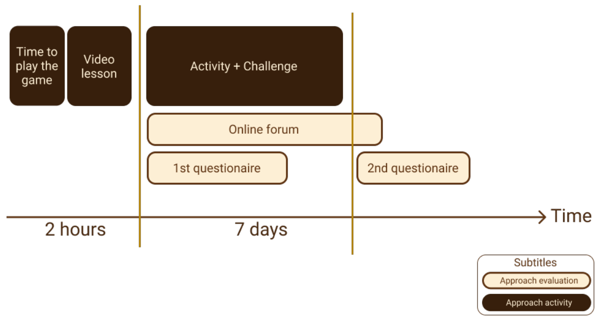


Figure 5: Timeline of the approach activities

In the next subsections, we present the results of each evaluation.

8.1 Game and video lesson evaluation

The first questionnaire was applied to evaluate the game and the video lesson, and was answered by 15 students. All of them are at least in the 7th semester and most of them are between 22 and 25 years old. Only one student was not used to play digital games.

Regarding the students' knowledge about unit testing, one student (6.7%) had no experience, 11 (73.3%) had some experience, and three students (20%) were used to unit testing. Also, none of the students had previously played educational software testing games.

As aforementioned, we have used the Model for the Evaluation of Educational Games (MEEG+A[27]) to evaluate the game. Thus, the questions of the questionnaire about the game aimed at evaluating the Usability and the Experience topics. The first topic – *Usability* – is divided into aesthetics and learning subtopics; and the second topic - *Experience* - is divided into challenge, satisfaction, enjoyment, focus attention, relevance and learning perception subtopics.

[10]https://forms.gle/Tpb5wzR99hkdMhrH6
[11]https://forms.gle/Fbz161bn1gyTDFGm9

Every question was answered using 5-points Likert scale: 1 totally disagreed and 5 totally agree. Figure 6 summarizes the students' answers about the *Usability* and *Experience* aspects of the game and the video lesson.

Regarding *Usability*, its subtopic *aesthetics* has two questions. The first one asked if the game was attractive and intuitive (Question 1), 4 (26.7%) students totally agreed and 11 (73.3%) agreed. The second asked if the text, fonts and colours of the game match and are consistent (Question 2), 6 (40%) students totally agreed and 9 (60%) agreed. When we asked if the game was easy to learn and play (Question 3), 7 (46.7%) students totally agreed, 7 (46.7%) agreed and 1 (6.6%) totally disagreed the game was easy to play.

In the *Experience* topic, the *challenge* subtopic had one question to check if the game was properly challenging (Question 5), 6 (40%) students totally agreed, 6 (40%) agreed and 1 (6.7%) totally disagreed. Regarding the *satisfaction* subtopic (*e.g.,* if the students would recommend the approach to their friends - Question 7), 8 (53.3%) students agreed, 6 (40%) agreed and 1 (6.7%) disagreed. Regarding the *Fun* subtopic, we asked if the students had fun playing the game (Question 6), 4 (26.7%) students totally agreed, 9 (60%) agreed and 2 (13.3%) were neutral. In the *focus attention* subtopic, 5 (33.3%) students totally agreed and 10 (66.7%) agreed that the game kept their attention (Question 4).

The *relevance* subtopic had four questions, the first one is about the connection between the unit testing concepts covered in the game and the video lesson (Question 8), 5 (33.3%) students totally agreed, 9 (60%) agreed and 1 (6.7%) disagreed that the connection is clear. The second question is about if the game facilitated the introduction of concepts that were explained later in the video lesson (Question 9), 9 (60%) students totally agreed and 6 (40%) agreed. The third question is related to the relevance for learning (Question 10), 5 (33.3%) students totally agreed and 10 (66.7%) agreed. The last question is about the students' motivation to improve their knowledge of unit tests after the lesson (Question 11), 4 (26.7%) students totally agreed, 8 (53.3%) students agreed, 2 (13.3%) were neutral and 1 (6.7%) totally disagreed.

To evaluate the *learning perception* we elaborated five questions, the first one asked if the students felt able to write unit tests after the game and the video lesson (Question 12), 4 (26.7%) students totally agreed, 5 (33.3%) agreed, 5 (33.3%) were neutral and 1 (6.7%) disagreed. The second is about if the students prefer to learn unit tests through our approach (Question 13), 7 students totally agreed, 6 agreed and 2 were neutral. In the third question, we asked to the students if the approach was suitable for an introductory unit testing course (Question 14), 10 (66.7%) students totally agreed and 5 (33.3%) agreed. The fourth question is about if the students think that the game contributed for their learning about unit test (Question 15), 2 (13.3%) students totally agreed, 10 (66.7%) agreed and 3 (20%) were neutral. In the last question, we asked how the approach was efficient for the learning comparing with other activities that we worked in the course (*e.g.,* forum, elaboration of V&V artifacts), 8 (53.3%) totally agreed, 6 (40%) agreed and 1 (6.7%) was neutral.

Regarding the subjective questions, we observed that the students liked the joyful dynamic promoted by our approach, they mentioned that it helped them to keep concentrated during the lesson. They also said the game helped them to understand the examples presented in the video lesson. However, some students

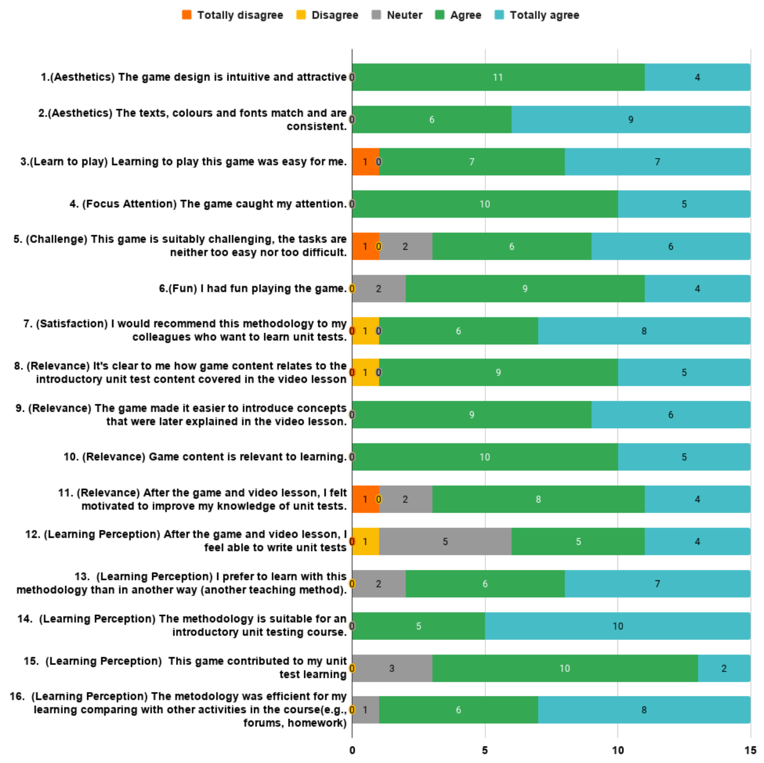


Figure 6: The summary of the results of the game and video lesson evaluation

complained about the recorded lesson, they said that the video was too long (44 min) and they had difficulties to keep full attention. They also said that the metaphors of the game could be more explicit.

8.2 Evaluation of the activity

The second questionnaire was answered for 8 students. The goal is to evaluate the whole approach: game, video lesson and the homework activity. Regarding their previous knowledge, in a scale from 0 to 5, the students answered: 1 (12.5%) student marked as zero (no knowledge), 3 (37.5%) marked one, 3 (37.5%) marked two, 1 (12.5%) marked three and none student marked the maximum (five). Besides that, 5 (62.5%) students said they could not finish the activity before attending the lesson, against 2 (25%) students who said they could do it, one (12.5%) student was neutral.

In this context, 7 (87.5%) students agreed that the objective of the approach and the activity were clear, only one (12.5%) was neutral. Besides that, all students agreed the evaluation criteria were clear. All students agreed they used the knowledge acquired in the game and video lesson to do the activity. Indeed, 6 (75%) students said the approach helped them to solve the exercises of the activity. Also, all students agreed the approach was consistent with the content. However, 2 (25%) students disagreed that the challenge has intermediate level, 5 (62.5%) agreed and 1 (12.5%) was neutral.

The last question, we asked how much the students were satisfied with the approach in a scale from 1 to 10, the students rated the approach between 8-10 as follows: 4 (50%) students rated 8, 2 (25%) rated 9 and 2 (25%) rated 10.

8.3 Forum evaluation

As part of the approach, the students were invited to participate of an online forum available on a university VLE . To guide the discussion, we asked two initial questions:

- Do you like the game? It helped you to understand the unit testing?
- Talk about the difficulty that you had to write the unit tests.

Regarding the first question, the students said that they liked the game, that helped to understand the concepts, but it just occurred when the developer mode was activated since the concepts with this mode were more explicit. Also, they said that the game helped to solve the exercises. Some examples of the answers: *"I liked the Game. The developer mode really helped me to understand the unit testing concepts."*; and *"I think it was really interesting! It helped a lot to reason about the activity and elaborate the tests cases."*.

In the second question, the students said they had difficulties with the chosen language (*Python*), one student said *"The concept was clear, my difficulty was more with the Python language than the testing structure itself."*. Other students had difficulties with the difference between each testing double, for example, one student said *"I had difficulty to understand the function and characteristics of the testing double."*. They also mentioned difficulties in checking that a unit test works correctly, for example, one student said *"Depending on the case, I felt difficulty to understand what should be tested, specifically the attack tests."*.

8.4 Activity grading

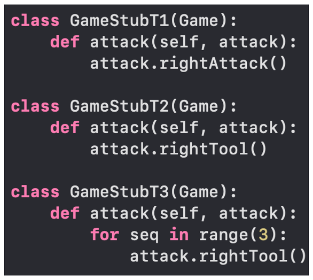
With the grading of the activity, we could understand better which subjects the students understood more and in which they had more difficulties. The two exercises were mandatory and composed part of the grade of the V&V course. The challenge was optional but they could reveal how engaged the students were.

The grade of the exercises could be from 0 to 1. The challenge could be only correct or not. If a team complete the challenge successfully, they win 0.3 extra points. However, if a team at least tried it, they win 0.1 extra points, as we said previously some teams worked hard and to motivate them, we gave scores.

All teams did the activity, 4 out of 9 got the maximum score, 3 got from 80% to 90% success rate and only 2 got from 60% e 70% success rate. Regarding the challenge, 7 out of 9 teams tried, but only 2 of them answered correctly.

Based on the grading of the exercise, we noticed the students understood the use of the *given-when-then* structure, since they tried to use it in their resolutions. All the teams also wrote correctly the implementation of the *spy* and *dummy* testing doubles.

However, we noticed the students had difficulties to understand the *stub* double, the most usually error was: the students created one *stub* for every test case, as showed in Figure 7. One way to do it is create a configurable stub and change their properties for each test case. The second more usually mistake happened when they tried to check if the method of some functions were working correctly, including calling the required methods. For example, in the Figure 7, the test was supposed to check if given a sequence with at least one correct tool the function would call the method *rightTool*. However, instead of calling the attack method itself and checking if the variable *rightToolCount* was incremented, they call the function *rightTool* directly, then the test was always passing.



```
class GameStubT1(Game):
    def attack(self, attack):
        attack.rightAttack()

class GameStubT2(Game):
    def attack(self, attack):
        attack.rightTool()

class GameStubT3(Game):
    def attack(self, attack):
        for seq in range(3):
            attack.rightTool()
```

Figure 7: Code with incorrect resolution

Another point that we noticed is that the most teams preferred the *stubs*, but the teams who used the *spy* double performed all tests correctly. We also observed that three teams had not implemented the *spy* double and one had not implemented the *dummy*, this result indicated that the activity's objective could be more clear about which kinds of testing doubles the students should implement.

9 DISCUSSION

In this section we discussed the following research questions (RQs) based on the results of the application of our approach:

RQ1. Does our game-based learning approach help the students to learn beyond theory?

One of the goals of our approach is to explore practical activities to make students more engaged. In this way, we proposed two practical activities – the game and the exercise; and one theoretical – the recorded lesson.

We noticed the students were engaged to play the game, however, the game acted more like an theoretical activity rather than being a practical one. Also, we have used code examples in this lesson, writing them while we explained the subject. We believe these examples helped to promote a more hands-on experience. However, the exercises proposed in our approach were the activity that most contributed to practical learning. We noticed that the students tried to apply the theoretic learning in their resolutions. For example, they tried to write the unit tests following the techniques we presented in the video lesson and most of them (7 out of 9) were engaged to solve the challenge.

Considering the results of the three different activities, mixing theory with practice, we believe that our approach provides experiences beyond theory.

RQ2. Does our game-based learning approach help the students to explore the theoretic concepts in the proposed practical activities?

In the V&V course we have worked previously with functional testing. So, all students had already experience with this type of testing. In our proposed approach we focus specifically on teaching how to write unit tests. The theoretical concept was presented in the recorded video lesson and applied in the exercises. For example, the *given-when-then* was firstly presented in the video lesson, the students learned why to use it, how it works and how to write it. Then, that lesson showed a practical example using the Python code. Next, the student had to implement the unit tests using the *given-when-then* structure. Lastly, the students were challenged to find a problem in our code - we noticed the students still were using this structure, an indication they absorbed this theoretical content.

However, we observed that the same did not occur in the testing dummies. The students learned the main concept, but they had difficulties to understand each dummy kind separated. In this case, the practical activity helped us to understand that this concept was not clearly explained in the video lesson.

RQ3. What are the benefits that students experience by using our approach to learn?

We noticed the students were engaged with the game and the playful examples. Being engaged was one of the benefits of our approach, sine as students were interacting and curious about the subject and even more engaged during the exercise. Moreover, the unit testing lesson of this course used to be more theoretical, focusing on basic unit testing content. Using our approach, we provide theory, practice, and teach students more advanced techniques such as the *given-when-then* structure and the testing dummies.

10 THREATS TO VALIDITY

In our study, we identified the following main threats to validity: the small number of students; the programming language to write the tests; and the limitation on the recorded video. In our study, we had a limited number of students to apply our approach. To minimize this threat, we have applied the approach on a course that has students with different profiles (*e.g.,* from different undergraduate courses). Also, the course is offered as optional at last semesters and its prerequisite is the SE course and thus the students have a basic theoretical knowledge about Software Testing. Regarding the programming language that we used to teach unit tests, we chose a programming language common for the students - Python. Moreover, we have used Python in the introductory programming courses at our university, and this language is widely used by industry. Other threat we identified is the limitation of the recorded video since we recorded the video beforehand the lesson to avoid Internet issues. To minimize this threat, we designed the lesson to be synchronous (game, video and explanation of the exercises). Also, we reserved a moment for questions after the video and we provided several communication channels after the lesson such as an online forum and Telegram to follow up the students during the exercises resolution.

11 CONCLUSION AND FUTURE WORKS

This paper presented the game-based learning approach, called GAMUT, for teaching unit testing. The proposed approach consisted of three steps: a game, a recorded video lesson and a practical activity. To engage the students in a playful way, we have used the Middle Age narrative to connect these steps. The approach starts with the game which works as trial and error method to exemplify unit tests. Then, the video lesson explains the theoretical and practical concepts related to the writing of unit tests by using the example of game code. Last, the students could practice the writing of unit tests doing a homework activity. We have applied the approach for the students of Computer Science in the context of a V&V course of a university. The results showed that the students were engaged in this approach activities since most of them played the game, attended the lesson and tried to solve the challenge. Indeed, the results of the activity correction showed that most of teams scored high. Moreover, the students were satisfied with the approach - they rated it between 8-10 on a scale of 1 to 10.

As future work, we will improve the game by making developer mode the default mode. We believe this improvement could facilitate the understanding since the concepts will be more explicit. We also intend to adapt our approach to the face to face classes focusing on the gameplay to encourage the teamwork between the students.

REFERENCES

[1] Rula Al-Azawi, Fatma Al-Faliti, and Mazin Al-Blushi. 2016. Educational gamification vs. game based learning: Comparative study. *International Journal of Innovation, Management and Technology* 7, 4 (2016), 132–136.

[2] A. Arcuri, G. Fraser, and R. Just. 2017. Private API Access and Functional Mocking in Automated Unit Test Generation. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 126–137.

[3] Arthur Silva Bastos, Renata Faria Gomes, Clemilson Costa dos Santos, and José Gilvan Rodrigues Maia. 2018. Synesthesia: A study on immersive features of electronic games. *SBC Journal on Interactive Systems* 9, 2 (2018), 38–51.

[4] Thiago A Beppe, Ítalo Linhares de Araújo, Bruno Sabóia Aragão, Ismayle de Sousa Santos, Davi Ximenes, and Rossana M Castro Andrade. 2018. GreaTest: a card game to motivate the software testing learning. In *Proceedings of the XXXII Brazilian Symposium on Software Engineering*. 298–307.

[5] Roberta Coelho, Uirá Kulesza, Arndt von Staa, and Carlos Lucena. 2006. Unit testing in multi-agent systems using mock agents and aspects. In *Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems*. 83–90.

[6] Heather Coffey. 2009. Digital game-based learning. *Learn NC* (2009).

[7] Gabriela Martins de Jesus, Fabiano Cutigi Ferrari, Daniel de Paula Porto, and Sandra Camargo Pinto Ferraz Fabbri. 2018. Gamification in Software Testing: A Characterization Study *(SAST '18)*. Association for Computing Machinery, New York, NY, USA, 39–48.

[8] Robert F DeVellis. 2016. *Scale development: Theory and applications*. Vol. 26. Sage publications.

[9] Michele D Dickey. 2011. Murder on Grimm Isle: The impact of game narrative design in an educational game-based learning environment. *British journal of educational technology* 42, 3 (2011), 456–469.

[10] Sebastian Elbaum, Suzette Person, Jon Dokulil, and Matt Jorde. 2007. Bug hunt: Making early software testing lessons engaging and affordable. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 688–697.

[11] Gordon Fraser. 2017. Gamification of software testing. In *2017 IEEE/ACM 12th International Workshop on Automation of Software Testing (AST)*. IEEE, 2–7.

[12] Steve Freeman, Tim Mackinnon, Nat Pryce, and Joe Walnes. 2004. Mock roles, not objects. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. 236–246.

[13] Tracy Fullerton, Chris Swain, and Steven Hoffman. 2004. *Game design workshop: Designing, prototyping, & playtesting games*. CRC Press. 180 pages.

[14] P. Henrique Dias Valle, A. M. Toda, E. F. Barbosa, and J. C. Maldonado. 2017. Educational games: A contribution to software testing education. In *2017 IEEE Frontiers in Education Conference (FIE)*. 1–8.

[15] IEEE. 1990. IEEE Standard Glossary of Software Engineering Terminology (Std. 610.12-1990).

[16] Tan Chin Ike, Tan Wee Hoe, Julian Lee Eng Kim, and Ng Yiing Y'ng. 2021. Exploring User Experience from an Emotional Context When Designing Immersive Games for Education. *Journal of ICT in Education* 8, 1 (2021), 10–25.

[17] Gabriela Kiryakova, Nadezhda Angelova, and Lina Yordanova. 2014. Gamification in education. Proceedings of 9th International Balkan Education and Science Conference.

[18] Joakim Korhonen. 2020. Automated Model Generation using Graphwalker Based On Given-When-Then Specifications.

[19] Erik Krogen. 2016. *Bond: A Spy-based Testing and Mocking Library*. Technical Report. Technical Report. Electrical Engineering and Computer Sciences, University of California at Berkeley.

[20] Valéria Lelli, Rossana MC Andrade, Lavinia M Freitas, Rubens AS Silva, Francisco Gutenberg S Filho, Renata Faria Gomes, and Jan Sousa de Oliveira Severo. 2020. Gamification in Remote Teaching of SE Courses: Experience Report. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*. 844–853.

[21] Ana Paula Lopes, Mariano Soler, Rocío Caña-Palma, Luis Cortés, M. Bentabol, Amparo Bentabol, Maria Munoz, Ana Esteban, and M Luna. 2019. Gamification in Education and Active Methodologies at Higher Education. https://doi.org/10.21125/edulearn.2019.0480

[22] Tim Mackinnon, Steve Freeman, and Philip Craig. 2000. Endo-testing: unit testing with mock objects. *Extreme programming examined* (2000), 287–301.

[23] Thomas W Malone. 1981. Toward a theory of intrinsically motivating instruction. *Cognitive science* 5, 4 (1981), 333–369.

[24] Arthur Marques, Franklin Ramalho, and Wilkerson L Andrade. 2014. Comparing model-based testing with traditional testing strategies: An empirical study. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 264–273.

[25] Gerard Meszaros. 2007. *xUnit test patterns: Refactoring test code*. Pearson Education.

[26] Rafael da Silva Nunes. 2017. Leituras do Poder na Terceira Era da Terra-Média: um ensaio baseado em O Senhor dos Anéis. *Espaço e Cultura* 41 (2017), 148–166.

[27] Giani Petri, C Gresse von Wangenheim, and Adriano Ferretti Borgatto. 2016. MEEGA+: an evolution of a model for the evaluation of educational games. *INCoD/GQS* 3 (2016), 1–40.

[28] Donald E Polkinghorne. 1988. *Narrative knowing and the human sciences*. Suny Press.

[29] Loana Russo Barbosa Ramos. 2019. Proposta de modo de acessibilidade para o jogo Nihilumbra: tornando o jogo mais acessível para jogadores daltônicos. TCC (Graduação em Sistemas e Mídias Digitais)-Universidade Federal do Ceará, Fortaleza.

[30] José Miguel Rojas and Gordon Fraser. 2016. Code defenders: a mutation testing game. In *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 162–167.

[31] Katie Salen, Katie Salen Tekinbaş, and Eric Zimmerman. 2004. *Rules of play: Game design fundamentals*. MIT press.

[32] Kevin A Stein and Matthew H Barton. 2019. The "Easter egg" syllabus: Using hidden content to engage online and blended classroom learners. *Communication*

Teacher 33, 4 (2019), 249–255.

[33] Jeff Stuckman and Guo-Qiang Zhang. 2005. Mastermind is NP-complete. *arXiv preprint cs/0512049* (2005).

[34] Pedro Valle, Ellen Francine Barbosa, and José Maldonado. 2015. Um mapeamento sistemático sobre ensino de teste de software. In *In Proceedings of Brazilian Symposium on Computers in Education (SBIE)*, Vol. 26. 71.