

# A gamified web based system for computer programming learning

Giuseppina Polito, Marco Temperini<sup>\*</sup>

*Sapienza University of Rome: Università degli Studi di Roma La Sapienza, Italy*

## ARTICLE INFO

### Keywords:

Gamification  
Automated assessment  
Computer programming learning  
Testing

## ABSTRACT

The availability of Automated Assessment tools for computer programming tasks can be a significant asset in Computer Science education. Systems providing such kind of service are built around an interface, allowing to administer the tasks (exercises to train programming skills), and show the results, accompanied by meaningful feedback. To produce such results, they apply techniques ranging from static analysis of program correctness, to testing-based evaluation. These systems can also support Competitive Programming, which is known to have educational meaning too. We developed the 2TSW system, supporting the automated correction of computer programming tasks, in a gamified web-based environment. The system let the student access a list of assignments (programming tasks), submit solutions to them, and have such solutions tested and graded. Accomplished tasks let the student gain experience points, represented also by medals, recognition of mastery on a topic, and improvements on a personal characterization of the student's status. The personal profile allows the student to monitor her/his proceedings and achievements. The gamified structure of the system, together with the availability of real-time automated assessment, offers the opportunity for an increasing level of students' personal engagement and motivation. Here we describe the system, and report on an experimentation, where students of a Bachelor Programme in Computer Engineering, first year, used 2TSW. In particular, we 1) present findings about the students' feedback, coming from a questionnaire administered after the experience, and 2) provide the reader with an analysis of the participation data, based on simple statistic tests. The students' feedback let us conclude that they appreciated the 2TSW gamified experience, perceived the system as useful, and maintained a high level of engagement. The data analysis allowed for less decisive conclusions, although it showed proof of the effectiveness of the system as a learning aid.

## 1. Introduction

Learning Computer Programming can be a tough matter. It is well known, for instance, that Higher Education students, even in Study Courses where programming is an essential skill (such as Computer Science and Engineering) have a hard time learning and training the principles, as well as being able to put them skillfully at work (Du et al., 2016). The reason might be in an impatient attitude of young generations, expecting quick results out of quick interaction. We are not arguing on this aspect; rather we notice that a great deal of responsibility is in the application of traditional teaching methodologies, insufficient to deal with the problem (Agapito & Rodrigo, 2017; Venter, 2020).

Technology Enhanced Learning (TEL) is the field where new technologies are considered, and new methodologies are studied for their application, with the aim to overcome the above mentioned limitations.

In this paper we deal with two important aspects of research in TEL, namely the use of Gamification, to improve the learning experience in

Programming courses, and the application of Automated Assessment techniques, to provide timely feedback for the learners' solutions (programs) to programming problems.

The use of Automated Assessment of programs is important in Computer Science education, as it multiplies the opportunities for the students to perform programming training, and to increase their programming skills (Ala-Mutka, 2005), while reducing the teacher's consequent grading burden. Embedded in a learning system, it can 1) allow for quick evaluation of programming proficiency, 2) make the evaluation formative, by means of meaningful feedback, and 3) enhance problem solving skills (Enstrom et al., 2011). Its usefulness is witnessed by several studies (Brusilovsky & Sosnovsky, 2005; de Souza et al., 2011; Edwards & Perez-Quinones, 2008; Joy et al., 2005). In addition, the rise of the Massive Open On-line Courses (MOOCs) generated further interest, for the automated assessment technology, its applicability, and scalability (Pieterse, 2013).

One interesting use of Automated Assessment is also in the field of

<sup>\*</sup> Corresponding author.

E-mail address: [marte@diag.uniroma1.it](mailto:marte@diag.uniroma1.it) (M. Temperini).

Programming Contest Systems, that support Competitive Programming (Comb  fis & Wautelet, 2014; Dagien  , 2010). These systems make the participants be *contestants*, and compete on the solution of a set of proposed programming tasks. This is not the main aim of a learning system; however, 1) it makes possible to increase motivation and engagement, on programming activities, at least for those (possibly many) students having a predisposition to competition, and 2) above all, it has shown to be effective on learning (Audrito et al., 2012; Blumenstein et al., 2008; Wang et al., 2011).

On the other hand, to master programming the learner must write and test programs, solving several programming problems, and maybe solving a same problem in different ways. This makes engagement a crucial issue. Engagement can be reduced, if the learner is bored, or if (s) he lacks motivation, or if (s)he lacks patience, and cannot be distracted by the wish of getting results in the quickest way.

Gamification has proven helpful for students, to increase motivation, and to allow gaining and maintaining engagement in training activities (Deterding et al., 2011; Kapp, 2012; Maia & Graeml, 2015). Basically, a learning environment is “gamified” through the integration in it of features that are originated in game environments. Among such elements are the use of experience points, connected to successful learning activities; the provision for fast feedback; the use of achievements, as well as levels of experience and badges, to allow the learner to trace her/his progression; the use of leaderboard, also to provide some level of competition; and the use of quests, to foster a longer term engagement.

This article wishes to contribute to the research area in Computer Programming Education, in a twofold manner: 1) we will present a web-based system, called 2TSW, where the use of automated assessment of programs is provided in a gamified learning environment; and 2) we will illustrate the results and limits of an experiment we conducted, using 2TSW in a higher Education context (undergraduate course on Computer Programming for novices).

In 2TSW, a “Course on Computer Programming” is basically a set of programming problems, administered by a teacher, and joined by students. So, in principle, several courses can be active at the same time in the system, for different classes. Currently the system supports the use of C programming language only. After login, the learner can access a set of programming problems, select one of them, and submit a program as solution of the problem. For each submission the system executes the program on several test cases (specified by the teacher while defining the problem), and returns an assessment page, where the program performance is summarized, feedback is given by describing the tests outcomes, and Experience Points (EP) are awarded, depending on the level of success of the proposed solution. A badge-representation of the outcome is also awarded (a medal or the “wooden spoon”). A set of problems ranging over the same topic defines a *category*, and a *category badge* represents the student’s proficiency on the problems met in that category. The Status of the student is one of the seven profile-badges that picture the overall performance of the learner in the course. The student’s profile page shows the personal achievements (EP, badges, status), and data about the submitted solutions), plus a visual estimate of the learner’s position in the class (with respect to EP and performance of the problems solutions).

In the following we will present the 2TSW system, and the experiment we performed. In particular, we will present in full the experimental data analysis, by which we tried to measure what effectiveness 2TSW has, or is promising to have.

After the experiment we collected participants’ feedback. The description of the feedback is presented in a conference paper (Polito et al., 2019), and a complete report would violate the editor rules, so we will only report the information we gathered from the students’ answers, without presenting the full analysis.

After this introduction, Sec. 2 offers an overview of related work available in literature. Then, 2TSW is presented in Sec. 3, while Sec. 4 gives an account of the elements of Gamification that are implemented in 2TSW, basing on the well-known categorization in (Deterding, 2013).

Before experimenting the system, we wanted to ensure that its behavior would be reliable and stable. So we devised a simulated experimentation, with artificial students, whose interactions with the systems were executed, by us, based on their individual models. The design and implementation of such simulation is described in another conference paper (Polito & Temperini, 2018): to avoid violating the editor rules, Sec. 5 reports in short on this simulated trial. Sections 6, through 8 describe the experiment we performed. In particular, Sec. 6 presents the data gathering, while Sec. 7 reports on the findings obtained through the questionnaire administered to the participant students, and Sec. 8 describes the simple statistical analysis we performed. In Sec. 9 we discuss some limitations this research can be blamed with, related to the sample’s quality, and the low amount of available data. We draw some final remarks in Sec. 10.

## 2. Related work

This section provides a description of the research work currently available about the educational use of automated assessment to support learning activities in programming, and on Gamification applied in educational contexts, in general. We also report on the uses of Gamification devised to support computer programming education. To summarize we could propose the following observations.

Gamification in education is studied since two decades, basically starting almost at the same time of the raising of gamification itself. In fact, it is still intensely studied, and the studies don’t seem to reach univocal results, about the boundaries, limits, and advantages of it. It has to be stressed that the majority of the results are positive, however the topic seems definitely worth of continuing experimental and design efforts.

In addition, the research activity on the use of advanced technology to support automated assessment of students’ programs, dates back even longer, and seems to be expanding. In particular, this research topic seems to have expanded in the last decade, supported by the developing web technology. On the other hand, it also seems on the verge of a further and extensive expansion, happening when the fast progress of Web Technology, and relatively new tools coming from Artificial Intelligence, will be ripe to easily support new advancements. These considerations seem to suggest that, for this topic, further research is expected by the scientific community.

Finally, the use of automated assessment in a gamification context is more seldom studied, and this might support the usefulness of the work presented in this paper.

### 2.1. Automated assessment of computer programs in education

Various kinds of automated support to programming education are met in research, since decades. Work on such topic is, for instance, described in (Hollingsworth, 1960). The widest area of investigation seems to be related to introductory programming courses, where students learn to write programs, according to a programming language syntax and semantics, and to solve problems (Gupta & Dubey, 2012; Hristova et al., 2003; Ala-Mutka, 2005; Pieterse, 2013).

In program assessment, errors may be uncovered basically by means of two types of program analysis. The first type is the *Static Analysis*, that produces its feedback without executing the program. Here the analysis is based on an examination of the program’s syntax and static semantics. Approaches of this type can be based on compiler error detection, error explanation (Hristova et al., 2003; Watson et al., 2012), structured similarity between marked and unmarked programs (Naud   et al., 2010), and also nonstructural analysis, with plagiarism detection and keyword search (Khirlunizam & Md, 2007). The second type of analysis works on the dynamic semantics, and possibly on the logic, of the program. It points out errors by means of testing. This means that the program is not examined *looking at the code*, but rather *running* it over particular sets of input data, specially devised to unveil problems; then

the output, obtained through the execution, is compared with the one that would be expected from a correct program. Competitive learning tools, used to manage Programming Contests (Leal & Silva, 2003), are based on this kind of analysis. Notably, Wang (2011) combines the two approaches: before performing dynamic testing, the program undergoes static analysis, where compilation errors, and similarity with “model programs”, are checked.

Contest Management Systems are dedicated to competitive programming, rather than focusing on the formative aspects. However, these systems can be used to support learning, introducing competition in the learning process. Indeed, contests are an active field of research also for the good return on competence that a competitive learning activity can foster for the students (Comb  fis & Wautelet, 2014; Dagien  , 2010; Garcia-Mateos & Fernandez-Aleman, 2009). On the other hand, competition can in turn be a negative spur for students, conducive to diminished engagement, especially when the system is used on a voluntary self-assessment basis by the students, who might dislike the idea of a continuous comparison with their peers.

In (Brusilovsky & Sosnovsky, 2005), an approach to the evaluation of student’s programming knowledge is based on a different perspective: the student’s proficiency is measured as the ability to answer “code-execution questions”. The student is exposed to questions; each question is related to a code fragment, and to an expression to be evaluated. Then, the answer regards the value of the expression after the execution of the code. Joy (2005) describes the BOSS online submission and assessment system. BOSS is a composite system, able to receive programming solutions by students and to apply them a collection of predefined tests. BOSS has also subsystems dealing with privacy, safe execution, and plagiarism detection for the solutions. In (Enstroem et al., 2011) the system Kattis is described. It is deemed to propose students with programming exercises, and grade the solutions. The exercises represent a means for both practicing programming, and reasoning about theoretical aspects. Kattis is also used to support programming competitions (such as the ACM-ICPC finals). From an architectural point of view, Kattis is a client-server application, accessible via web. Once a student has submitted a solution to a programming problem, the “judge sub-system” elaborate a “judgement”, by applying the code on secret tests. The “verdict” is then released to the student.

The research in (Gupta and Dubey, 2012), adds to the field of assessment through static-analysis, presenting an enrichment of the program testing technique, based on *program verification*.

A differently constructive approach to learning is the one based on Test Driven Development (TDD). In this case the development of a suite of test for a student’s submission is asked to the student itself (and then those test can be integrated with teacher’s additional tests). This practice can be beneficial for students in a programming course, as shown in (Edwards 2003; 2008). The use of tests provided by students is also in (de Souza et al., 2011), where the assessment is corroborated by teacher-provided reference tests and solutions.

An innovative method to produce automated assessment of a student’s program is presented in (Conejo et al., 2018), where the program undergoes a series of pre-defined tests, and a list of responses is produced. The responses are booleans or more articulated grade values. Then, the string of responses is used by the scoring process, based on the application of two assessment theories (the Classical Test Theory, and the Item Response Theory). The responses are basically representing items of interest of the program to be evaluated. The scoring results are shown (to students and teachers) through a set of interfaces that function as a kind-of dashboard. The items are defined by the teacher, while authoring the “exercise”. Such items can be related to various aspects of the program (static-syntactic features, dynamic-run-time performance, and also the complexity of algorithm coded by the program). So the items can be quite fine-grained, providing rich feedback that can be complex to read for the student, while as well effective for formative assessment.

A practical approach to computer programming education, is sought

in (Queir  s, 2019), with the idea that solving programming problems, and having the proposed solution assessed by means of tests, is the key to effective training on programming. In a landscape of automated assessment systems, and problems repositories, that are mostly stand-alone and poorly interoperable, this work proposes an architectural framework where services can be orchestrated, to provide all the necessary functionalities to support a course, such as the problems retrieval, the individual modeling, the adaptation of the offered learning activities, and the automated assessment of programs. The paper also foresees the use of such an architecture for an actual course: this application would consist of the use of the architectural features described in the paper, and in their integration with the interfaces and gamification features needed to complete the support to the course. The idea is that each course could quasi-easily implement its specific gamified approach. Due to the mainly architectural interest, the paper does not provide an actual application of gamification, which is foreseen as future work.

## 2.2. Gamification for education

Gamification is recognized as a very significant methodology, suitable to foster motivation and engagement in students. Its results are widely appreciated (Sailer et al., 2017; van Roy & Zaman, 2018), and sometimes challenged (Dom  nguez et al., 2013), which makes it a live and progressing research field. The literature review in (Koivisto & Hamari, 2019) examined a wealth of articles, dedicated to various fields. It provided a thorough set of research directions. Among the notable conclusions, the review pointed out that most of the controlled experimental works (of which the majority was in Education) gave rather *mixed* than thoroughly *positive* results. Such results were in terms of factors such as motivation, engagement, enjoyability, perceived effectiveness, participation, and number of tasks accomplished (NB not necessarily programming tasks). Quoting from the conclusions, *gamification is not a silver-bullet type of solution for achieving positive results and success, in either the research sphere, or in practice* (Koivisto & Hamari, 2019). Recently (Gomez, 2020), presented an interesting description of processes of gamification of the class context, which may make it simpler for teachers to gamify their courses. The conclusions of the article were in that efforts should be made to further work on the application of gamification in education, and on the methodologies to apply it to courses.

## 2.3. Gamification for programming education

Regarding the application of Gamification to programming education, there are several initiatives reported in literature. The works in (Elbaum et al., 2007) and (Fraser, 2017) show examples of Gamification applied in the educational area of Software Testing (Fotaris et al., 2015). presents a study where a gamified quiz system was offered for use by students of a university course in basic programming. The treated students appeared to have increased their willingness to attend the course, and the consequent grading results. No automated assessment of programming submissions was included in the system (not necessary for the quiz-format of the students’ tasks).

(Piteira et al., 2018) elaborates on the need to keep working on the application of gamification in computer programming education. The paper proposes a selection of gamification principles and features, and describes their application to an on-line course. It presents mainly methodological or architectural contributions, regarding the students’ interest and approval of the gamification features. So, the methods to assess learning are not of specific interest there, and are not described in deep. Similarly, the use of an automated assessment mechanism for the programs is not implied.

In (Kasahara et al., 2019) the authors present an analysis of how the Cyclomatic Complexity of the students’ code resulted improved, in a Gamified environment. In this paper the Cyclomatic Complexity was

taken as a measure of goodness for the code, and used to infer a better engagement: as long as these assumptions are correct, the results of the experience are fairly good.

An application of principles of gamification, such as medals (badges) and leaderboard, in a system proposing programming challenges to teams of students, is described in (Rojas-López et al., 2019). The challenges are presented through narration, to foster engagement. In this system the analysis of the students' submissions is done by the teachers, based on a checklist; so no automated tool provides feedback/grading of learner's programming artifacts.

(Layth Khaleel et al., 2019) experimented the use of gamification features for a web-based system supporting computer programming learning at a novice level. The system administers tests (quizzes) and evaluate the learner's comprehension of basic concepts. The quizzes may contain fragments of programs, but the system does not support student's submission of programs and the related automated feedback/grading.

An experimental application of elements of gamification (such as leaderboard, challenges, tournament) in a programming class, is presented in (Figueiredo & García-Peñalvo, 2020), where good results are reported on attendance to classes, participation, and reduction of failure. The students showed good appreciation of the opportunity, and also proposed some warnings. Evaluations and awarding of points were managed by the teachers, and no automated grading system was applied to computer programs.

The recent review of literature in (Venter, 2020) offers an analysis of research initiatives specifically focusing on the application of gamification to programming courses, in Higher Education, published in the period 2014–2019. The review found 21 papers to examine, of which 17 were applying actual implementations of Gamification. The majority of such studies used new implementations of gamified systems, rather than adopting existing gamified platforms (such as [www.codingame.com](http://www.codingame.com), [KhanAcademy.org](http://KhanAcademy.org), or Kahoot!). Among the findings, we might underline two main aspects, related to 1) the gamification elements used in the applications, and 2) the effects on learning. The most used gamification elements were, by far, Leader Boards, Badges, and Points, as opposed to scarcely used Avatars and Progress Bars. The effects of gamification appeared to be usually *positive* for engagement, but quite *mixed* for “Programming Knowledge”. We assume that “Programming Knowledge” is the category where programming skills fall, so these final results seem not very positive.

#### 2.4. Can automated assessment and gamification collaborate for programming education?

From the previous subsection we can conclude that systems, where a gamified environment for programming training is enriched with automated assessment technology, are not frequently studied. From studies of the past we might also take the warning about even the very use of assessment, automated or not, in a gamified educational system: sometimes it is associated to other potentially stress-inducing features (such as the Peer-Versus-Peer engagement, or the organization of Tournaments among the students), as capable of harming the student's engagement, when not accompanied by a timely and enlightening feedback (Fotaris et al., 2015; Lee et al., 2013). So, considering the scarcity of application, and the need to study effects and potential problems, it seems reasonable to work on this kind of systems.

During the development of our 2TSW initiative, we experienced difficulties to approach the task by just adopting already existing systems. The reasons for that are twofold: On the one hand, using a system that was developed to cover the needs of a specific higher education course/university, can be difficult: we found that existing experimental systems may too narrowly frame our activities; there might even be hardware/software problems, when the local technological, logistic, and personnel framework could not easily be adapted to the needs of the system. Moreover, we also had to deal with possible language problems

of the students (all the available systems are of course developed in English).

Eventually, we developed the idea of 2TSW as a tool to be used in the specific environment of our Faculty's classes, having also in mind the need to offer a solution in the local language. Moreover, we wished to offer the students a system where, at least for the overall “status” of the user proficiency (a part of the student profile), a more compelling characterization was used, than the traditional “medal” approach. Indeed, we used medals for the single exercise report, and quite traditional badges for the “categories” of similar exercises; but we also intended to use a more personal label for the individual general proficiency, based on the consideration (derived from experience) that a “fun aspect” can be appreciated by the students, tickle their curiosity, and eventually keep them at doing exercises.

### 3. The 2TSW system

As mentioned earlier, in 2TSW the teacher can be administrator of a “course”, which is basically a set of programming problems available to the enrolled students. Fig. 1 shows the list of programming problems (also called Programming Tasks in 2TSW) available to a student (*stud1*) in the “C.8” course. Several courses for programming training can be managed in 2TSW: in Fig. 1 the course's identifier and name are in a kind-of box, which is an active element of the page (a *select input element*). Such select element gives access to the other courses which the student is enrolled in.

When a student submits a program (to solve a problem), the solution is tested, by using unit testing, and the feedback is shown to the student (Fig. 2).

A programming task is related to a specific topic of interest. The tasks related to a same topic are grouped under the topic's *category*. For instance, in course C.8 there is a category *Usage of Reference Pointers*, collecting all the exercises available to train on the use of pointers in C programming. For each task, the solution can grant the submitting student with an amount of Experience Points, depending on the difficulty and the level of completeness of the solution. Such amounts are stated by the teacher, while defining the programming task in the system.

A task-related medal is awarded to the student, if the solution is successful. Gold/Silver/Bronze medals are awarded, depending on the quality of the solution. Unsuccessful solutions are awarded with the Wooden Spoon. Fig. 2 shows a bronze medal, associated to the comments given to the student after testing her/his solution.






In relation to the evaluation of a submitted task, Fig. 3 shows the feedback received after a successful evaluation. In particular, a detail is shown about the level bars that allow the student to measure the distance of her/his solution from the best solution, and to see the level of performance in this task with respect to the other learners (anonymously).

The *panoply* is the representation of the student's profile. Fig. 4 shows a student's panoply: the student's Status (see below) is visible, together with the results obtained through her/his solutions. Category Badges show the student's accomplishments in the various categories, rendered by labels (Amateur, Beginner, Expert, Champion, or Legend). A category badge is awarded depending on the ratio between the Experience Points won by solving tasks in that category, and the maximum available points for those tasks. A student is allowed to submit further solutions for the same task. Only the last submission is used for the panoply updates.

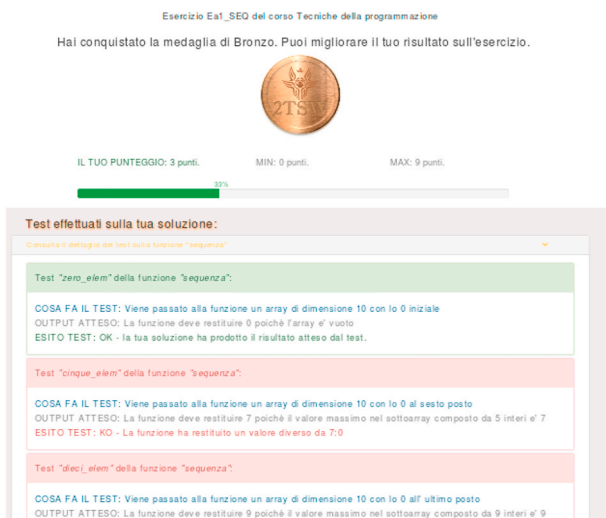
The Status of a student is awarded according to the Experience Points (s)he gained. The statuses badges are shown (except for one) in Fig. 5, which is a rendering of the Leader Board. The Leader Board allows 1) the teacher to see through the students' performances, and 2) the students to monitor their performances and make comparisons with their peers.

Through the panoply, depending on the privileges, the student and the teacher can access various levels of details about students. While a student can access only basic information about another student, (s)he



Home				C.8 - Tecniche della Programmazione	COS'È 2TSW	BADGES	DESK(STUD1)	LOGOUT
Esercizi presenti nel corso C.8 - Tecniche della Programmazione								
Esercizio	Categoria	Traccia	Partecipati					
 Ea1_SEQ	Puntatori	L'esercizio richiede la scrittura della seguente funzione: - int sequenza(int *seq) che prende in ingresso un array di interi e ne stampa il massimo. La sequenza dei numeri da controllare e' terminata da uno 0, mentre la dimensione dell' array e' N uguale a 10.	<a href="#">Inserisci una soluzione</a>					
 Ea1_SMI	Puntatori	L'esercizio richiede la scrittura di una funzione che abbia il seguente prototipo: - int sommamda(int *arr, int quanti, int daDove) in cui 'arr' e' un array di interi di lunghezza N, 'quanti' indica quanti numeri vanno sommati, 'daDove' indica l'indice di inizio della somma. La funzione deve restituire un intero che rappresenta la somma degli elementi richiesti. Utilizzare la seguente definizione: #define N 10. NB: La funzione deve controllare se il valore 'quanti' e' maggiore dei numeri rimanenti da sommare e se l'indice 'daDove' e' maggiore della dimensione dell'array.	<a href="#">Inserisci una soluzione</a>					
 Voli_2	Tabelle Statiche	L'esercizio richiede l'implementazione di tre funzioni: - int aggiungiVolo(TipoTabella t, char cod[], char dest[], int o, int m, int p) che prende in ingresso un puntatore alla tabella e i dati di un volo e li inserisce nella tabella, restituisce 1 in caso di successo, 0 in caso contrario. Le strutture da utilizzare sono le seguenti: - #define MAXVOLI 10 - struct ora { int ore, minuti; }; - struct volo { char codice[6]; char * destinazione; struct ora oraPartenza; int postiLiberi; }; - typedef struct volo TipoVolo; - typedef struct { TipoVolo arrayVoli[MAXVOLI]; int quantiVoli; } TipoTabella;	<a href="#">Inserisci una soluzione</a>					
 Voli_3	Tabelle Statiche	L'esercizio richiede l'implementazione di tre funzioni: - int indiceVolo(TipoTabella t, char cod[]) che prende in ingresso una tabella e un codice e restituisce l'indice dove e' memorizzato il volo; - int aggiungiVolo(TipoTabella t, char cod[], char dest[], int o, int m, int p) che prende in ingresso un puntatore alla tabella e i dati di un volo e li inserisce nella tabella, restituisce 1 in caso di successo, 0 in caso contrario; - int eliminaVolo(TipoTabella t, char cod[]) che prende in ingresso un puntatore alla tabella ed il codice del volo da eliminare, restituisce 1 in caso di successo, 0 nel caso di un codice falso. Le strutture da utilizzare sono le seguenti: - struct ora { int ore, minuti; }; - struct volo { char codice[6]; char * destinazione; struct ora oraPartenza; int postiLiberi; }; - typedef struct volo TipoVolo; - typedef struct { TipoVolo arrayVoli[MAXVOLI]; int quantiVoli; } TipoTabella;	<a href="#">Inserisci una soluzione</a>					
 Liste_1	Liste	L'esercizio consiste nell'implementare due funzioni: - void insTestaLista(TipoLista * plis, TipoElemLista elem) che prende in ingresso un puntatore ad una lista e un elemento da inserire e lo inserisce in testa; - TipoLista	<a href="#">Inserisci una soluzione</a>					

**Fig. 1.** List of the available programming tasks (the textual descriptions are irrelevant, here). Student stud1 is enrolled in course C.8. The course identifier C.8 is visible in figure, on top of the web page. Also the name of the course is visible: “Tecniche della Programmazione”, which can be translated as Foundations of Computer Programming. The icons associated to the tasks say whether the problem was solved and how by stud1. In this case three tasks were solved (Gold, Silver, Bronze medals). One task was not successful (Wooden Spoon). Another task, among those visible in figure, is yet to do.



**Fig. 3.** Again on the submission feedback. The silver medal represents a good performance; the textual comment above the medal provides also an encouragement to try to make it better. The first bar (green in the picture) shows the level of the solution with respect to the best possible solution (in this case 13 points were granted, out of 19 maximum). The second (blue) bar provides a comparison with the peers; in this case the performance is described as “better than the 20 % of the other learners”.

**Fig. 2.** A solution has been submitted, and assessed. Here the feedback is shown. A medal represents the evaluation (Bronze, Silver, Gold are assigned depending on the amount of gained Experience Points. The Wooden Spoon means that the solution was insufficient. For each test performed on the code, then, the result is shown. The test rationale is described (the actual text is irrelevant, here), so to provide the student with some matter for reflection, and self-correction.

can access a detailed presentation of each one of her/his own solutions (Fig. 6).

A student can also access her/his “trend”, which is a representation of the curve of her/his performance in time: the trend can be shown in relation to different aspects:

- about a single task: showing the improvements obtained while developing several different solutions for that same task (if available);
- about a category: showing the curve of evolution of the category badges. This allows to associate the student's work on tasks of that category, to the progression of the student's proficiency on the topic, toward the current estimated value;

- about the overall status of the student: depicting the evolution of the student's status. (This third type of performance curve is the one actually shown in Fig. 7.

#### 4. Gamification aspects of the 2TSW system

We designed 2TSW also based on categorizations of the Gamification feature, such as those in (Dicheva et al., 2014; Sillaots, 2014) and especially (Deterring, 2013). We can summarize as follows, the typical game elements that are to be considered when gamifying a learning activity. Please notice that in each point/element we also add considerations, about how 2TSW does implement the related concept.

- 1) The concept of growing grade. Here we use the word “grade” to mean the representation of proficiency that the students can see in her/his profile. Well, it is natural to expect that certain actions in the system might let such grade grow, as well as certain others might make it decrease. Decreasing would actually inspire a twofold negative attitude, in the student: on the one hand the dejection due to seeing a



**Fig. 4.** A student's panoply (condensed, to fit into this page): the status (*Big Chieftain*) is shown, together with information about the performance in the categories of tasks. For each category the level (badge), and the medals are shown. A medal, or spoon, is granted for each undertaken task. The longer bar points out the level of experience points gathered by the students. The shorter bar says how the student is positioned in the class (in this case the students has more points than the 56 % of her/his peers).



**Fig. 5.** Fragment of the Leader Board. The Status badges are, in order of programming prowess, Zombie, Common Earthling, Vita da Mediano (which is a citation from a song, non-translatable; it just means the dull work of a Middlefield player in football), Big Chieftain, Genius, Supernatural, Deity (no Deity in this instance of the Leader Board).



**Fig. 6.** Detailed, visual/textual, presentation of the performance of a student in a single task (as seen by the student): quality of the solution is shown by the medal, and by the gained Exp. Points. Beside showing the task's category, the output also present the ratio of points gained Vs. maximum available points for the task (68 % in this case).

loss, and on the other hand the avoidance of further attempts to propose solutions, fearing to increase such loss.

In 2TSW the student tries a new solution only with the intent of getting better results. If the new solution is worse than the previous, one can fall back to the better performance. In this way the Experience Points can only grow, which has a positive effect of encouragement.

2) Provision of extensive feedback. Student's actions (such as submitting a solution in 2TSW) are answered in a predictable and justifiable



**Fig. 7.** Trend of the Status evolution for a student (Stud10). While using the system, the student progressed from Zombie to Big Chieftain, quite quickly. Then paused for some time, and eventually raised to Genius.

way, giving feedback, comments, and possibly encouragement in response. All the consequences are shown in various ways (board, badges, graphics) that allow the student to monitor her/his state, also in comparison with others. This can inspire a positive sense of competence.

In 2TSW the use of growing experience points, the representation of the status, the categories and tasks badges, the graphics showing the progresses, and of course the feedback consequent to the test done on a submitted solution, support the provision of extensive feedback.

3) Nested and progressive challenges. Here the concept is in that the “challenges” a learner has to face can be met progressively, in a twofold sense: 1) the major challenges have to be structured in a sequence, and 2) the individual challenge can also be decomposed in more short term tasks, so to allow the learner build the solution to a challenge by solving smaller problems. In this way, at any moment the learner can see that the challenges can be met in an orderly manner, and that the solution to a challenge can be produced progressively, by sub-challenges of smaller import. This is expected to inspire, in the learner, the idea that the current task is doable by her/his current capabilities, and the next one in sight will be doable as well, as (s)he, by then, will have developed further capabilities.

In 2TSW the tasks are organized in categories. A category corresponds to a (more or less broad) topic, which is met in a series of tasks associated to that category. This organization brings into 2TSW the logic of nested and decomposed challenges, where a category is supposed to be the big challenge, and the tasks play the role of shorter term targets. In turn the categories could be considered as medium term objectives, whereas reaching a given, higher, status is interpreted as the long term challenge.

4) Autonomous Choice. This factor of Gamification means that the student should have the possibility to select the tasks to meet, rather than being forced along a fixed path. The possibility to select a task independently strengthens the sense of self-determination, and allows also for self-assessment: one might be pleased to see that the chosen task was actually solvable, with the current abilities, or (s)he could be made aware of the fact that the current abilities are not yet sufficient, and should be sharpened, before trying that task again.

In 2TSW the learner is confronted by a series of categories, of growing complexity, and (s)he is allowed to start solving task from a higher category than the first. The categories apply a multiplicative weight to their tasks’ experience points, according to the difficulty of the concepts involved in the tasks. In his way, simpler categories add less points than higher categories. On the other hand, a learner that feels the need for more repeated training on simpler categories, can work there longer, in order to cultivate her/his abilities, and become able to meet the further challenges.

5) Freedom to Fail. This is an element of great success in games. It is granted when the learner is allowed to fail a task without hard consequences, or no consequence at all. Failing may lead to further attempts on the same task, or it may let the learner devise a different game path toward progresses in the game.

In 2TSW, basically, this aspect is supported by the fact that the learner is not going to be penalized by submitting wrong solutions. In the current state of the system, though, there are limitations depending on the relatively small number of tasks actually available. If the learner has few tasks to choose from, there aren’t ways to circumvent a task and get the same experience points elsewhere. We think that this limitation would be easily overcome, when the system will be populated with hundreds of tasks, rather than tens.

6) Competition and Collaboration. These are possibly contradictory concepts, as they are enunciated in (Deterding, 2013). The possibility to compare one’s own performance with the others’ is allowed quite extensively in games. In a learning setting, however, it can elicit the fear of being judged by peers, and so it can have negative effects. Collaboration can actually co-exist with competition, when there is the possibility to gain experience, and experience points by teaming up.

In 2TSW we did not implement aspects of collaboration, and allowed

competition only in that it is supported by the visibility of the Leader Board. Unfortunately, mechanisms to soothe the fear of comparison with peers, and of judgement, are not yet implemented in 2TSW.

In the previous points we have recalled a classification of the main aspects, or elements, that should be considered in a Gamification project for learning. We think that 2TSW is able to put in practice many of such elements, and that an experimentation would be worthwhile.

In order to prepare an experimentation, we made an analysis of the behavior of the system during its use. We thought that a verification, by means of a simulated experiment, would help see whether the system is able to accompany the progresses of several students, and correctly report on them. In the following section we discuss such a simulated experiment, concluding that the system was ready for use with real students.

## 5. A preliminary functional test of 2TSW

After having designed and implemented 2TSW, we wished to test whether collection and management of data were correct. We decided to pursue a simulated experiment, trying to feed the system with as realistic data as possible. To this aim we defined simulated students (sim-students), and executed, for each one of them, several interactions with the system (mainly submission of programs). To give structure to the experiment, and manage to see how the system correctly computed different profiles for different students, we defined a quite informal sim-student-model, called *typology*. A sim-student *typology* is a couple  $\langle \text{attitude}, \text{competence} \rangle$ , where *attitude* is a label (with value “Challenger” or “Minimalist”), and *competence* is another label, with possible values as “Low”, “Average”, “Good”, or “High”. A Challenger is a student who tends to retry problems, to solve them better. Challengers are supposed to aim at the best possible Status, and collect the most valuable medals. On the contrary, a Minimalist is supposed to do just what is needed to reach a reasonably good advancement (let’s assume that anything much higher than Zombie would be enough). Moreover, a Minimalist is not going to retry a solved problem, if just it was solved sufficiently.

All the details of this experiment are reported in the conference paper (Polito & Temperini, 2018). Here we provide a summary.

In the experiment we had 10 sim-students, with varied typologies, enrolled in a course (“C8”), comprised of 11 programming tasks, over 4 categories (*reference pointers*, *array/struct*, and data structures *table* and *linked list*). All the sim-students started from Zombie Status. The experiment was divided in two phases.

During Phase1 each sim-student behaved according to its initial typology. The quality of the submitted solutions was in agreement with the competence. For a Challenger the number of submitted solutions would be 5 or 6, while the number of retried solutions would be 2 or 3. For a Minimalist the number of submitted solutions would be 3 or 4 (no resubmissions). At the end of Phase1 the sim-student might likely have changed Status (depending on the Experience Points awarded to the solutions). Also, at the end of Phase1 a positive effect of the activity was assumed on sim-students’ proficiency. So we updated the competence of the sim-student, making it, in general, slightly better. Notice that we did not change the attitude component of the typology, as we postulated that it would not change on a short term. During the second phase, the same steps of Phase1 were executed, based on the updated typologies.

So, for instance, the following sim-students participated in the experiment:

- Stud3, with initial typology  $\langle \text{Challenger}, \text{Average} \rangle$ ;
- Stud5, with initial typology  $\langle \text{Challenger}, \text{Good} \rangle$ ;
- Stud8, with initial typology  $\langle \text{Minimalist}, \text{Average} \rangle$ ;
- Stud9, with initial typology  $\langle \text{Minimalist}, \text{Good} \rangle$ .

We recall here the list of Status labels: *Zombie*, *Common\_Earthling*, *Vita\_da\_mediano*, *Big\_Chieftain*, *Genius*, *Supernatural*, and *Deity*. The label *Vita\_da\_mediano* was inspired by the title of a popular song. It is not

directly translatable; it just represents the dull work of a Middlefield player in football).

During Phase1, Stud3 and Stud5 submitted 5 solutions, repeating two of them (or retrying twice a problem), while Stud8 and Stud9 submitted 3 solutions. At the end of the phase.

- Stud3 raised its Status to Big\_Chieftain, and was granted typology < Challenger, High>;
- Stud5 raised its Status to Vita\_da\_mediano, and had < Challenger, Average > typology (actually we managed two sub-levels for “Average” – see later – and Stud5 raised to the upper sub-level);
- Stud8 raised its Status to (only) Common\_Earthling, and its typology did not change;
- Stud9 raised to Vita\_da\_mediano, and the typology became < Minimalist, High>.

During Phase2, with respect to solutions submitted/repeated, the sim-students maintained the behavior displayed in Phase1. At the end of the phase, Stud3, Stud5, and Stud9 raised their Status to Genius, while Stud8 remained a Common\_Earthling.

To match solutions and competence for the sim-students’ submissions, we adopted the following informal method: when competence was High, the submission’s Experience Points would be between 80 % and 100 % of the maximum points defined for the task. If competence was Good, the points would be between 60 % and 80 %. If competence was Low, the points would be between 0 % and 20 %. About Average competence, we defined two sub-levels, granting points either between 20 % and 40 %, or between 40 % and 60 %. During Phase1, Average competence would grant points only in the first sub-level. Before starting the Phase 2, however, each sim-student with Average competence was assigned randomly one of the sub-levels, to simulate the different developments that different students can have in reality.

In conclusion of this section some words should be spent about the limits of the presented approach to 2TSW testing.

On the one hand, our typologies of sim-students are not coming from a deep psychological-educational investigation, nor they are supported by previous experiments managing large amounts of data. In fact, they were stated based on teaching experience, collected informally, from laboratory learning activities conducted in previous years. In such activities the students were requested to answer questions and produce small pieces of code, and the evaluation was not automated, implying direct interaction with the teacher. In some of such activities, interactions among students were also allowed. Out of these experiences, we appreciated an array of behaviors, especially with regard to students that wished to update their answers (to increase their quality), possibly by several sequential attempts.

On the other hand, we did not wish to perform a formal software testing process. The aim of the experiment was to feed the system with quasi-realistic data, and verify how it is able to trace the behavior of the students, and how the flow of change in the status is in agreement with the typologies we assigned to the sim-students, and their evolution.

With these limits, we concluded that the outcome of the simulated experiment showed that the system would be fit to support a truer experimentation, with real students. Such an experimentation is the topic of the next section, and of the rest of the paper.

## 6. Field evaluation: data collection

In this experiment the (real) students were enrolled in a course on *Foundations of Computer Programming*, held for the *Bachelor in Computer Engineering* at *Sapienza University*. The 2TSW corresponding course was comprised of 11 programming tasks, over 4 categories (*reference pointers*, *array/struct*, and *data structures table* and *linked list*). It was basically a copy of the C.8 course used in Sec. 5. Also due to organizational and technical problems, the experiment started only 2 weeks before of the end of the semester (mid-May), and was then prolonged till

Fall (mid-September). Participation was spread along the mentioned period, with students enrolling mostly at the beginning. Basically each individual student used the system for a limited time, weeks at most, due to the fact that the beginning of the experiment was almost at the end of the semester, and the whole period of time was not the best one for participation.

In sum we had 12 students registered, of which only 10 did actually solve at least some problems. The students were all male, ranging between 19 and 21 years of age.

The participation of the students was totally on a voluntary basis; no direct gain, on the final exam results, was offered. The only “baits” were in a set of gadgets (mouse, pen drive, keychain, nice pens) that were promised to be assigned based on the Leader Board.

A significant information is that, to date, only 9 of the participants have actually taken the final exam, and the related final grade.

The students, about halfway in the course, were shortly examined in a colloquium (called *Intermediate Exam*). So this colloquium was about the basic notions of programming, met so far: 1) basics of architecture and number systems; 2) programming with the if and while constructs; 3) defining functions; 4) use of basic types for variables; and 5) use of arrays. The rest of the course would deal with more advanced programming, over dynamic data structures.

In Table 1 the participants’ data are shown. In the table, IE is the mark received after the Intermediate Exam, FG (Final Grade) is the grade obtained by the student after the final exam, and Exp.P. is the number of Experience Points gained during the use of 2TSW.

As a matter of fact, the low number of participants in the experiment denotes a limit of the research work we are presenting. However, we considered that the overall results of this experience can be suggestive, if not strongly confirmative, of the system’s effectiveness, and deserving to be communicated. The reasons for that opinion are in the following two points:

- the participants’ feedback (as shown in the next section) was quite satisfactory; the overall 2TSW experience received very high marks, on all the aspects we investigated;
- the data analysis (reported in Sec.8) shows that 1) considering IE representative of the population (the students in the class), there is no significant difference between the sample and the population; and 2) considering FG as variable of interest, all the comparisons between sample and class show a positive effect for the sample’s members.

We propose some further observations, about the limits of this research, in Sec. 9.

**Table 1**

Students participating in the experiment. IE is the grade in the Intermediate Exam. FG is the grade in the final exam. The last column reports the Experience Points accumulated during the use of the system. The empty cells in the FG column corresponds to students that have not yet taken the final exam at the time of writing this paper. Descriptive data for the sample’s FG are as follows: mean 28.089, st.dev. 3.063.

Learner’s id.	IE	FG	Exp.P.
a1	30	31.1	301.4
a2	29	29	297.8
a3	28	28	282.8
a4	30	30	267.5
a5	31	32	142.4
a6	30		69.5
a7	25	27.2	117
a8	30	26	81
a9	24	21	19.8
a10	24		0
a11	30	28.5	155.7
a12	26		0



## 7. Field evaluation: feedback from participants

After the experimentation, we asked the participants to fill in a questionnaire (reported in Table 2). The method of analysis and the findings have been presented in a conference paper (Polito et al., 2019), so in this section we are summarizing them. We intended to collect feedback on the following 5 respects.

- 1) “2TSW Experience”: how the experience offered by the experiment was appreciated by the participants;
- 2) “Usefulness”: how the participants perceived 2TSW as useful to improve their programming skills;
- 3) “Engagement”: what level of engagement the participants experienced (as witnessed, for instance, by the availability to repeat the solution of a problem until perfection, or maximum recognition of experience points);
- 4) “Gamification Experience”: how the students appreciated the “gamified dimension” of the 2TSW learning experience; and

**Table 2**

The 29 main questions. In general, the responder was requested to label each statement by her/his agreement (1 = strongly disagree, 2 = disagree, 3 = neutral, 4 = agree, 5 = strongly agree); q26, q28, and q29 are the only exceptions (and their answer choices are listed in this table).

Q	
q1	In my opinion it is important that 2TSW provided me with immediate feedback about my code
q2	I feel a special satisfaction when my code passes the 2TSW tests
q3	In my opinion 2TSW is reasonably easy to use
q4	When my code passed some of the tests I keep working at it, in order to enhance it and let it pass all the tests
q5	If my task code is not accepted by 2TSW I feel motivated to look for errors and fix them
q6	In my opinion 2TSW is helpful improving my programming capabilities
q7	In my opinion 2TSW is helpful forcing me to write better (more correct) code
q8	In my opinion 2TSW is helpful measuring one's programming capabilities
q9	Using 2TSW was in general a good experience, and constructive
q10	I found in general innovative the mechanics implemented in 2TSW
q11	Before I submit a task solution, I check the code thoroughly for errors
q12	I'd like to participate, in future, in a programming contest, even if tasks might be tougher
q13	When I work on a programming task I get easily tired
q14	When I work on a task, my aim is to pass the task with as little effort/work as possible, even if it is not with the best grade
q15	Before sending a solution, I'd like to have that the algorithm was the best I can come up with
q16	The tasks definitions in 2TSW are adequate
q17	I would like to use systems like 2TSW (where automated assessment of complex tasks is performed) in other courses
q18	In my opinion, using 2TSW on a regular basis along the course would increase considerably my workload
q19	In my opinion, using 2TSW on a regular basis along the course would have given me the opportunity and motivation to do more training on programming
q20	In my opinion, using 2TSW on a regular basis along the course would have been beneficial in increasing my programming skills
q21	In my opinion the comments and suggestions coming from 2TSW when a solution is tested, were useful to improve my solutions
q22	While using 2TSW I felt interested and motivated to catch medals and category labels and to improve my status
q23	While using 2TSW I've have felt in part like participating in a game
q24	I liked, in 2TSW, aspects like the score, the experience points, the personalized leaderboard, the badges for my accomplishments, and the status
q24	While using 2TSW I've have felt in part like participating in a game
q25	While using 2TSW I felt bored
q26	I've felt threatened by the possibility to compare my results publicly (by username) through a leaderboard (Yes/No)
q27	I think that the possibility to look at my results, through a leaderboard, is beneficial for my motivation and engagement, in using the system and making my programming capabilities better
q28	Do you perceive that the use of the debugger is now (after the 2TSW experience) more important than earlier? (Yes/No)
q29	Would you have liked to have more problems available in 2TSW? (“Yes, many more”, “More or less it's ok like it is”, “No, they are too many already”)

- 5) “Openness”: how open the student would be to the perspective of using other web based systems featuring automated assessment for different subject matters than programming.

The questionnaire was comprised of 36 questions, and it was available in local language. 29 questions were addressing the above mentioned aims, while the rest concerned age, sex, name (not mandatory), whether topics in computer programming were met earlier in high school (two questions), number of problems solved, and free text observations/hints. We discuss the findings coming from (some of) these last questions in the last section of this paper. Almost all the questions had answers in a 5 values Likert scale (expressing a level of agreement with the sentence reported in the question, from least agreement (1) to fullest (5). To such a format we had three exceptions. Not all the enrolled students answered the questions, and eventually, we had 7 responders, among the active participants.

The overview of the answers tells us that the results were very good. In particular, if we compute the average of the answers' average marks, we get 4.37, which denotes an overall quite positive appreciation by the responders.

To analyse the questionnaire also with focus on the 5 aspects mentioned at the beginning of this section, we grouped the questions in 6 categories (two for *Gamification Experience*), according to the relevance of the questions for the categories.

In the following we describe the categories and the related findings, also explaining what questions are part of what categories.

### 7.1. 2TSW experience

Here we collected the questions 1, 3, 9, 10, 16, 18, as they help evaluate how the learning experience was appreciated by the participants, and how a student would like the idea of extending the usage of 2TSW to the whole course time.

We considered the average marks for these questions, and computed the average of these values. The result is 4.19, which says that the 2TSW-based learning experience was well endured and liked by the responders. It appeared, though, that the students are somewhat preoccupied, by the possibility that an extensive use of 2TSW might be conducive to a significant increase of workload. However, they show, not only in this category, a firm availability to use 2TSW all over the course.

Overall, the experience of use of the system, even discounting the short period of experimentation, and the preoccupations about possible increase of workload, was more than positive.

### 7.2. Usefulness

Here we collected the questions 1, 6, 7, 8, 18, 19, 20, 21, 28 as they help evaluate how the participants considered 2TSW able to help improve one's programming skills.

The average of the average marks is 4.21, which tells us that the responders considered the system quite useful for the development of their programming skills, in spite of the mild preoccupation for a possible increasing workload.

### 7.3. Engagement

Here we collected the questions 2, 4, 5, 11, 13, 14, 15, 22, 25, 29, as they help evaluate how the participants were willing to keep trying to solve the same problem, if the solution produced so far could be improved, and how available they were to undertake further problems.

The average of the average marks is 4.36, representative of quite a good result in terms of engagement. In particular, we noticed that 1) the participants were not feeling tired, during the work on solutions; 2) the programming work was not usually interrupted after reaching the minimum result; and 3) boredom, in the interaction with the system, is

not at all an issue.

#### 7.4. Gamification Experience

Here we collected the questions that appeared useful to see how the participants appreciated the “gamified learning experience”. Two aspects, in this category appeared orthogonal and worth of separate consideration, so we defined two sub-categories:

- The category *Gamification Experience 1* collected the questions 22, 23, 24, 25, with the intent to measure how the student was perceiving her/his learning activity in a gamified dimension.
- The category *Gamification Experience 2* concentrated on one of the several factors of a gamified approach, the Leader Board. So this category collects the questions 26, 27.

For the first sub-category the answers show an appreciative feedback and state (question 25) that the use of the system was not at all boring: the average value of the average marks was 4.57. In *Gamification Experience 2* the average was 4.72, indicative of the fact that using the Leader Board was taken very positively, which is in contrast with results found in many research works - cfr. for instance, (Venter, 2020).

#### 7.5. Openness

This category relates to the learners’ willingness to use a web application like 2TSW, where automated assessment of learning tasks was provided. Questions 12, 17, and 18 were included.

The average of the answers’ average marks is 4.24, corresponding to a good availability of the learners. The answers to question 12 (average mark 4.57) suggest that the students were enticed by the idea of participating in a programming contest. The key question (17) had an average mark equal to 4.86, witnessing a great interest of the learners for the use of systems providing automated assessments, in general. The question q18 was one of the few needing a normalization, to be used for the computation of average marks. In fact, for it a lower mark would mean a higher satisfaction for the responder. This question had average mark equal to 2.71, normalized to 3.29. It was added in the current category as it is related to the expectations of “additional heavy work”, induced by the possible use of 2TSW all along the course. From the answers to this question we see that the students are somewhat preoccupied of the possible work overload, should a system (or more systems in different courses) like 2TSW be made mandatory. The value for this question is, however, still above neutrality, so these preoccupations are not overwhelming.

### 8. Analysis of the experimental data

Table 1 showed us that the mean Final Grade of the students in the sample is 28.089, with standard deviation 3.063. The related data for the whole class of course students is as follows: mean FG 26.02, and standard deviation 2.70. So, there is a difference between the class’ FGs and the sample’s ones. However, this does not necessarily imply an effect of the system on the sample’s members.

To organize an analysis of the available data, and to discern the effect of the system’s use, we considered a categorization of the students, based on the IE mark. IE (cfr. Sec. 6) comes from the Intermediate Exam, hence it gives some measure of the proficiency of the students in initial topics of the course. In the Intermediate Exam, the students were graded based on marks going from A+ (top) to D. The whole evaluation grid was: A+, A, A-, A/B, B+, B, B-, B/C, C+, C, C-, C/D, D+, D, corresponding to grades between 31 and 18 (as usual in our country the university grades have such numeric span, with 31 representing the top mark: *30 cum laude*). The same marks span is used for FG.

We have to point out that there is a component of “encouragement” in the Intermediate Exam, so the grades were 46 % of the “A” type (A+,

A-, A/B), 36 % of “B” type (B+, B, B-, B/C), and 18 % of the remaining types (only one D).

As mentioned earlier, the experiment involved a sample of 12 students, on a population of 64 students. At the time of this writing only 9 students in the sample, and 39 among the other students, had taken a final grade. Hence 9 is the dimension of the sample, and 39 the dimension of the population, excluding the sample. To have a preliminary descriptive analysis of the experimental data, the following Pearson correlations were computed:

- IE-FG-ALL (between IE and FG, for all the students who made the final exam);
- IE-FG-A (between IE and FG, limited to the students having an A-type IE mark);
- IE-FG-B (between IE and FG, limited to the students having a B-type IE mark);
- IE-FG-A-B (between IE and FG, limited to the students having an A or a B-type IE mark);
- IE-FG-C-D (between IE and FG, limited to the students having a C or D-type IE mark);
- IE-FG-SMP (between IE and FG, for the students in the sample)

We also computed two additional correlations related to the sample:

- IE-XP, comparing the proficiency in the intermediate exam and the performance in the experiment (represented by the experience points accumulated while using 2TSW);
- XP-FG, comparing the performance in the experiment and the final exam grade.

The correlations are as follows:

- IE-FG-ALL 0.535
- IE-FG-A 0.619
- IE-FG-A-B 0.441
- IE-FG-B 0.193
- IE-FG-C-D 0.818
- IE-FG-EXP 0.786
- IE-XP 0.566
- XP-FG 0.687

There appears to be a correlation, in general, between the performance in the intermediate exam and that at the end of the course (IE-FG-ALL). “B” students appear to be the most unpredictable (IE-FG-B is very low), while “C” and “D” students have a high correlation between intermediate and final grade (IE-FG-C-D). As a matter of fact, very few “C” or “D” students made actually the exam.

The fact that IE-FG-ALL is not very high can be easily explained, by considering that the most complex topics in the syllabus are inevitably met in the second part of the course: the student’s performance is not entirely determined by the situation at the intermediate exam (fortunately).

The participants in the experiment were mainly “A” students (2/3); if we consider those that, at the time of writing this paper, had not yet passed the final exam, A students rise to 78 % (no C or D were found in the sample). This allows to draw some preliminary conclusions about the usefulness of 2TSW:

XP-FG is not very high, but it is sufficiently high to say that there is correspondence between the proficiency in the final exam and the behavior during the experiment. This is a good result, in our opinion: in particular, it suggests the possibility to unveil difficulties for students, so to help the teacher administering remedial activities.

In relation to the effectiveness of 2TSW, as a learning tool, the above correlations can’t be considered decisive. On the one hand, IE-XP denotes a correlation between behaviors in the intermediate exam and in 2TSW, which is in line with IE-FG-ALL. At the same time, the behaviors

in 2TSW and in the final exam have a slightly better correlation (XP-FG) than IE-FG-A: considering the prevalence of "A" students in the sample, this might suggest that the students exposed to the system actually gained something, in terms of learning.

There is a clear difference between IE-FG-SMP and IE-FG-ALL, as well as between the former and IE-FG-A-B: this, however, could not allow to conclude about the effectiveness of 2TSW, as the sample is small, and moreover the participation in the system was on a voluntary basis, so the sample could be biased (for instance, we might just have a prevalence of students who wanted to take any chance to make programming exercises, and would have had the same final grades in any case).

So, we thought that some more light could be shed on the effectiveness of the system, by the following simple statistic comparisons, implemented as single sample Z-tests.

### 8.1. Statistical analysis of the Experiment's data

Here we propose some analyses, with the intent to compare the sample with progressively more homogeneous (to the sample) parts of the class.

Each analysis is performed on the final grade FG. In order to obtain the above mentioned homogeneity, we use the classification based on the IE mark: while we might consider the sample not representative enough, of the whole class, we might think that it would represent better the part of the class that had similar IE grades (i.e. the A and B-type students). In addition, we might reduce (painfully) the sample, to only those students that got an A-type IE, and compare them with the part of the class that had such an IE grade too. In this case we have "presumably good" students compared, and we might accept that the sample's students had an effect after all, with respect to equally good (or even better) students who did not use 2TSW.

With the above explained line of thought, we first performed a test, having IE as variable of interest, to see whether there are significant differences between the sample and the population. In this test we limited the consideration to the following subsets of sample and population:

- the sample was reduced from 12 to 9, retaining only the students that had participated actively in the experiment and had taken the final exam;
- the population was reduced, by retaining only the students that had got an IE of type A or B, and had taken the final exam. (Remember that the students in the sample have only type A or B IE).

The test was two-tailed, with  $\alpha = 0.05$ . The null Hypothesis was that there is no significant difference between sample and population. The population mean, computed over the 34 students with a FG, was 27.63, with sigma 2.10. The sample average, was 28.556, with sigma 2.315.

We had  $p = 0.186$ , with z-score 1.322 (smaller than the z-value 1.960), so it was not possible to reject the null hypothesis. Hence, from the point of view of IE, it cannot be stated that there is significant difference between the sample and the population (i.e. the sample could be considered acceptably representative).

We then performed the following tests:

- 1) comparison of sample's data with class data;
- 2) comparison of sample's data with class data, where the class data was limited to those students that got IE of A or B type;
- 3) comparison of sample's data with the class data, where both the sample and the class data were limited to those students that got IE of A type;

For all tests we assumed FG as the variable of interest, for which the mean and standard deviations were computed. For the computations we

used 8 fractional digits, which in the following we will round to 3. The tests were one-tailed, with  $\alpha = 0.05$ , and considered the following Hypotheses:

H0 the experiment had no significant effect:  $\mu \geq \mu_{\text{sample}}$

H1 the experiment had significant effect:  $\mu < \mu_{\text{sample}}$  where  $\mu$  is the population mean of the FG values, and  $\mu_{\text{sample}}$  is the related sample's mean.

### 8.2. Comparison of Sample's and class data

The population mean, where 39 students had a FG, was 26.023, with standard deviation (sigma) 2.698. The sample average over the 9 participants with FG, was 28.089, with sigma 3.063.

We had  $p = 0.0109$ , with z-score 2.297 (greater than the z-value 1.6449), so the null hypothesis was rejected.

### 8.3. Comparison of Sample's and class data: only students with A or B-type IE mark

In this case we cut the population data, excluding the students that had C or D IE marks. This did not dramatically reduce the number of elements, as very many of such students had not yet passed the final exam. The sample remained as in the previous test (the sample's students have A or B-type IE marks).

The population mean, where 34 students had a FG, was 26.309, with sigma 2.431. The sample average over the 9 participants with FG, was 28.089, with sigma 3.063.

We had  $p = 0.014$ , with z-score 2.197 (greater than the z-value 1.6449), so the null hypothesis was rejected.

### 8.4. Comparison of Sample's and class data: only students with A-type IE mark

In this case we cut the population data, excluding the students that had B, C or D-type IE marks. The population mean, where 22 students had a FG, was 26.614, with sigma 2.737. The sample average (limited to the 7 participants with FG, and with IE of type A) was 29.229, with sigma 1.863.

We had  $p = 0.006$ , with z-score 2.527 (greater than the z-value 1.6449), so the null hypothesis was rejected. In this case we performed also a z-test with  $\alpha = 0.01$ , obtaining that the z-score was greater than the z-value 2.3263, allowing to reject the null hypothesis as well.

In conclusion of this section we can say that some effects of using 2TSW might have been revealed, with the limitations that we anticipated, and that we discuss in the next section.

## 9. Limitations of the presented study

We think that there are two important limitations in the work we have presented here. One is related to the fact that the participation in the experiment was totally on voluntary basis. The other limitation is in the low number of students who participated in the experiment (i.e. the limited dimension of the sample). In this way, the sample might be not representative of the population of the students in the course.

To deal with these limitations, we tried to make use of the partial characterization of the students, provided by the Intermediate Exam. We designed a test route, based on single sample z-tests, in order to show that 1) the sample did not have significant differences with the population (considering only the part of students that had a grade of type A or B in the Intermediate Exam), and 2) that three comparisons between sample's and population's Final Grade revealed significant differences.

We will have to repeat the experiment, with a new version of 2TSW, in more suitable and successful conditions, in order to confirm the results of effectiveness that this analysis is suggesting.

## 10. Conclusions

2TSW is a web based system, presenting the student with a gamified learning environment where an automated assessment sub-system supports training on computer programming. The Gamification features available in the system are based on concepts like Badges, Leader Board, Peers' Profile Comparison (possibly anonymized).

We reported on the field experimentation that we conducted. We administered a post-experience questionnaire to the participating students, on which we conducted a qualitative analysis. From the analysis we saw that the participants appreciated quite warmly the opportunity to use a gamified system featuring automated assessment of programs, and perceived the system as useful to improve their programming skills. Moreover, the participants showed to be fairly engaged in the solution of the programming problems, and attracted by the gamified aspects of the system. Finally, the questionnaire answers revealed a good availability of the students to the use of systems offering automated assessment of complex learning activities, as well on programming as on other disciplines.

We proposed a set of simple statistical analyses of the few data we had. Such analyses allow us to say that the sample, though limited, was not significantly different from the population of the class; hence it could be considered representative of the whole population. About effects on learning, the presence of some effects of the use of 2TSW was suggested by the analyses conducted on the Final Grades of the students.

We have also discussed the limits of the present research, which suggest further experimentation to confirm the results.

In Sec.5 we postponed the discussion about some side questions, proposed in the questionnaire, that were not directly related to the analysis conducted in that section. They were 6 questions, and we think that a discussion about two of them could be useful here. The first question asked whether the student had studied some of the topics met during the experiment earlier, in the High School. About this, we saw that students have had some prior experience, at an elementary level; one student declared to have had an experience at a level similar to the one met in the course, on some of the course's topics. We also had one student (with 11 problems solved, we say proudly) who had had no experience at all. The second question was just asking for an open answer, with notes, hints, or observations inspired by the experience. Some of the participants provided answers to this question, and we considered them helpful to define directions for future work. In particular, we received one comment complaining about *the solution not being accepted for unknown reasons*, one comment advising that *the solution might be accepted also if the program was not actually a general solution, just fitting in the specification of a given test*. Our intention to work on two items of future work was strengthened by those answers.

The first item regards the improvement of the system, by the creation of a sub-system supporting an easy definition of tests. Currently the definition of a test is quite a laborious task for the teacher. Simplifying that task would allow to increase more easily the amount of tests available for a programming problem, and to manage a random selection of a subset of such tests at assessment time.

The second item of future work is planning and implementing a new experimental activity, allowing to have a more numerous set of students, working with 2TSW along a longer period of time.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

- Agapito, J., & Rodrigo, M. M. (2017). Designing an intervention for novice programmers based on meaningful gamification: An Expert evaluation. In *Proc. 25th int. Conf. On computers in education* (pp. 736–745), 2017.
- Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2), 83–102.
- Audrito, G., Demo, G. B., & Giovannetti, E. (2012). The role of contests in changing informatics education: A local view. *Olympiads in Informatics*, 6.
- Blumenstein, M., Green, S., Fogelman, S., Nguyen, A., & Muthukumarasamy, V. (2008). Performance analysis of game: A generic automated marking environment. *Computers & Education*, 50(4), 1203–1216.
- Brusilovsky, P., & Sosnovsky, S. (2005). Individualized exercises for self-assessment of programming knowledge: An evaluation of QuizPACK. In *J. Ed. Resources in computing* (Vol. 5). ACM, 3.
- Comb  is, S., & Wautelet, J. (2014). Programming trainings and informatics teaching through online contests. *Olympiads in Informatics*, 8.
- Conejo, R., Barros, B., & Bertoa, M. F. (2018). Automated assessment of complex programming tasks using SIETTE. *IEEE Transactions on Learning Technologies*, 12(4), 470–484.
- Dagien  , V. (2010). Sustaining informatics education by contests. In *Teaching fundamentals concepts of informatics* (pp. 1–12). Springer.
- Deterding, S. (2013). *Gameful design for learning*. TD Magazine. . (Accessed July 2013).
- Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011). From game design elements to gamefulness: Defining “gamification”. In *Proc. MindTrek'11* (ACM).
- Dicheva, D., Irwin, K., Dichev, C., Talasila, S., & Salem, W. (2014). A course gamification platform supporting student motivation and engagement. In *Proc. IEEE int. Conf. On web and open access to learning*.
- Dom  nguez, A., Saenz-de-Navarrete, J., de-Marcos, L., Fern  ndez-Sanz, L., Pag  s, C., & Mart  nez-Herr  iz, J. J. (2013). Gamifying learning experiences: Practical implications and outcomes. *Computers & Education*, 63, 380–392.
- Du, J., Wimmer, H., & Rada, R. (2016). “Hour of Code”: Can it change students' attitudes toward programming? *Journal of Information Technology Education: Innovations in Practice*, 15, 52–73.
- Edwards, S. H. (2003). Improving student performance by evaluating how well students test their own programs. *J. Educational Resources in Computing*, 3(3).
- Edwards, S. H., & Perez-Quinones, M. A. (2008). Web-CAT: Automatically grading programming assignments. In *Proc. ITICSE*. ACM, 328–328.
- Elbaum, S., Person, S., Dokulil, J., & Jorde, M. (2007). Bug hunt: Making early software testing lessons engaging and affordable. In *Proc. ACM/IEEE int. Conf* (pp. 688–697). on Software Engineering (ICSE).
- Enstrom, E., Kreitz, G., Niemela, F., Soderman, P., & Kann, V. (2011). Five years with Kattis – using an automated assessment system in teaching. In *Proc. FIE conference*. IEEE.
- Figueiredo, J., & Garc  a-Pe  n  lva, F. J. (2020). Increasing student motivation in computer programming with gamification. In *Proc. IEEE global engineering education conference*.
- Fotaris, P., Mastoras, T., Leinfellner, R., & Rosunally, Y. (2015). Who wants to Be a pythonista? Using gamification to teach computer programming. In *Proc. 7th int. Conference on education and new learning technologies EDULEARN*.
- Fraser, G. (2017). Gamification of software testing. In *Proc. IEEE/ACM 12th int. Workshop on automation of software testing*.
- Garc  a-Mateos, G., & Fernandez-Aleman, J. L. (2009). Make learning fun with programming contests. In *Transactions on edutainment II* (pp. 246–257). Springer.
- Gomez, S. A. (2020). Games and gamification in the classroom. In D. Burgos (Ed.), *Radical solutions and eLearning: Practical innovations and online educational technology. Lecture notes in educational technology series* (pp. 101–115). Springer.
- Gupta, S., Dubey, S. K., & SIP. (2012). Automatic assessment of programming assignment. In I. T. C. S. Proc (Ed.), *JSE-2012*, 315–323. CS & IT-CSCP.
- Hollingsworth, J. (1960). Automatic graders for programming classes. *Communications of the ACM*, 3(10), 528–529.
- Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). *Identifying and correcting java programming errors for introductory computer science students*. 3. SIGCSE.
- Joy, M., Griffiths, N., & Boyatt, R. (2005). The BOSS online submission and assessment system. In *J. Educational resources in computing* (Vol. 5). ACM, 3.
- Kapp, K. M. (2012). *The gamification of learning and instruction: Game-based methods and strategies for training and education*. John Wiley & Sons.
- Kasahara, R., Sakamoto, K., Washizaki, H., & Fukazawa, Y. (2019). Applying gamification to motivate students to write high quality code in programming assignments. In *Proc. ACM conf. On innovation and technology in computer science education* (pp. 92–98).
- Khirulnizam, A., & Md, J. (2007). A review on the static analysis approach in the automated programming assessment systems. In *Proc. Nat. Conf. On software engineering and computer systems, pahang, Malaysia*.
- Koivisto, J., & Hamari, J. (2019). The rise of motivational information systems: A review of gamification research. *International Journal of Information Management*, 45, 191–210.
- Layth Khaleel, F., Sahari Ashaari, N., & Tengku Wook, T. S. M. (2019). An empirical study on gamification for learning programming language website. *Jurnal Teknologi*, 81(2).
- Leal, J., & Silva, F. (2003). Mooshak. A web-based multi-site programming contest system. *Software: Practice and Experience*, 33, 567–581.
- Lee, M. J., Ko, A. J., & Kwan, I. (2013). In-game assessments increase novice programmers' engagement and level completion speed. In *Proc. 9th ACM conference on computing education research* (pp. 153–160).



- Maia, R. F., & Graeml, F. R. (2015). Playing and learning with gamification: An in-class concurrent and distributed programming activity. In *Proc. IEEE frontiers in education conference*.
- Naudé, K., Greyling, J., & Vogts, D. (2010). Marking student programs using graph similarity. *Computers & Education*, 54, 545–561.
- Pieterse, V. (2013). Automated assessment of programming assignments. In , Vol. 13. *Proc. CSERC* (pp. 45–56). ACM.
- Piteira, M., Costa, C. J., & Aparicio, M. (2018). Computer programming learning: How to apply gamification on online courses? *Journal of Information Systems Engineering and Management*, 3(2), 11.
- Polito, G., & Temperini, M. (2018). A Gamified Approach to Automated Assessment of Programming Assignments. In M. Chang, et al. (Eds.), *Challenges and Solutions in Smart Learning. Lecture Notes in Educational Technology* (pp. 3–12). Springer.
- Polito, G., Temperini, M., & Sterbini, A. (2019). Automated assessment of computer programming assignments, in a gamified web based system. In *Proc. 18th Int. Conference on Information Technology Based Higher Education and Training* (pp. 1–9). ITHET.
- Queirós, R. (2019). PROud-A gamification framework based on programming exercises usage data. *OR Informatie*, 10/2, 1–14.
- Rojas-López, A., Rincón-Flores, E. G., Mena, J., Garcia-Penalvo, F. J., & Ramirez-Montoya, M. S. (2019). *Engagement in the course of programming in higher education through the use of gamification*. 18 pp. 583–597. Univ Access Inf Soc.
- van Roy, R., & Zaman, B. (2018). Need-supporting gamification in education: An assessment of motivational effects over time. *Computers & Education*, 127, 283–297.
- Sailer, M., Hense, J. U., Mayr, S. K., & Mandl, H. (2017). How gamification motivates: An experimental study of the effects of specific game design elements on psychological need satisfaction. *Computers in Human Behavior*, 69, 371–380.
- de Souza, D. M., Maldonado, J. C., & Barbosa, E. F. (2011). ProgTest: An environment for the submission and evaluation of programming assignments. In *Proc. SEET* (IEEE).
- Venter, M. (2020). Gamification in STEM programming courses: State of the art. In *Proc. IEEE global engineering education conference* (pp. 859–866).
- Wang, T., Su, X., Ma, P., Wang, Y., & Wang, K. (2011). Ability-training-oriented automated assessment in introductory programming course. *Computers & Education*, 56(1), 220–226.
- Watson, C., Li, F., & Godwin, J. (2012). Bluefix: Using crowd-sourced feedback to support programming students in error diagnosis and repair. In , Vol. 755. *Proc. ICWL2012* (pp. 228–239). LNCS. Springer.